WebSphere Message Broker Version 7.0.0.8

Contents

Chapter 14. Reference 3579

Chapter 1. WebSphere Message Broker

This information center provides information for IBM® WebSphere® Message Broker Version 7.0.

The information contained here supports Version 7.0.0.8. This information center is integrated with the WebSphere Message Broker Toolkit and with the WebSphere Message Broker Explorer. It is also available online. Always refer to the WebSphere Message Broker online information center to access the most current information.

Topics that have been added since the previous release of this information, and updated sections within topics, are highlighted with a start of change icon preceding the updated information and an end of change icon following the updated information, as shown in this paragraph.

If you are migrating components and resources from previous versions, see Chapter 3, "Migrating and upgrading," on page 137.

New users: how to find the information you want

The Contents pane on the left lists the categories of task information that are available; for example, **Migrating** and **Developing applications**. Each task category also contains explanations of the concepts related to those tasks, such as "Coexistence", "Message flows", and "Publish/subscribe".

The Start here category in the Contents pane contains topics that provide an introduction for new users to getting started, developing applications, broker administration, and troubleshooting tasks in WebSphere Message Broker. Each topic in the **Start here** category provides overview information and links to relevant topics in the information center. Read the Start here topics and follow the links to help you to navigate through the product documentation and find the information that you want. Alternatively, you can use the following instructions to help you find the information that you want.

Click the top-level topic that you are interested in. If it describes and lists information that covers your area of interest, expand the section to read the associated concepts that give an overview of what you need to understand to perform tasks in this area. Before you start, make sure that you are familiar with the concepts and terminology that are associated with the tasks.

The navigation tree in the Contents pane provides a framework in which you can start to understand the tasks and the context in which you will complete them. Follow the related links at the end of each task to find information about other related tasks. You can move backward and forward to revisit topics, and you can use the **Refresh/Show Current Topic** button if you have moved to a different area of the table of contents, and you want to check where you are.

Expand the **Reference** section to find reference information that is relevant to the task area that you are interested in. You can see that the sections within the Reference section largely correspond to the task categories. You will probably have to refer to one or more of these reference topics to complete any particular task.

For more information, see "Task, concept, and reference topics." When you are more familiar with what information is available, and where to find it, try using the index and the search facility, which are described later in this topic.

Experienced users: how to find the information you want

If you are familiar with what information is available, and where to find it in this information center, you probably have a specific piece of information in mind. You can navigate through to the information in the Content pane, or you can take a more direct route to find specific information:

Search

Enter your search string in the Search box, then click **Go**. If your string includes blanks, and you want the whole string treated as one string, enclose it in quotation marks; for example "search string". The search results are presented in the Search Results pane on the left, and you can switch between the Search Results pane and the Content pane, which contains the navigation tree, by clicking the tabs at the bottom of the pane.

You can limit your search to a particular area of the navigation tree, which can make a search much quicker and produce fewer results. To limit the search, click **Search scope**, then click the **New** button. Select the area or areas you want to include in your new scope and enter a name for the scope so that you can reuse it another time.

Index Find the term that you want in the Index. If more than one entry exists for the subject that you looked up, click the one that is most appropriate for your current task.

Use the letter links at the top of the index to link to the sections that start with that letter; use the letter at the start of a section of index entries to link back to the top of the index.

If you are new to this version of WebSphere Message Broker, you might find some of the information in the Start here category in the Contents pane useful. The Start here topics include links to information about navigating the Information Center, using the WebSphere Message Broker Toolkit, migration and coexistence, and troubleshooting WebSphere Message Broker.

Task, concept, and reference topics

The topics in this information center are divided into task, concept, and reference topics.

Tasks and concepts

Task topics provide steps or actions that tell you how to complete the tasks to establish and maintain your broker environment.

Concept topics provide definitions and background information that help you to understand the product and the ways in which you can use it to solve your business problems. Check the Glossary for a definition of many of the terms introduced in these topics.

• Chapter 2, "WebSphere Message Broker overview," on page 5

These topics provide a good introduction to the product and its facilities. Expand **Product Overview** to read about the product, and what is new in this release. You can also read about, and invoke, the Quick Tour, an online guided introduction to the product components and options. Access the samples that are supplied too, to understand more about how your applications can use the broker function.

- Chapter 6, "Configuring brokers for development environments," on page 563
- Chapter 7, "Configuring brokers for test and production environments," on page 579
- Chapter 8, "Administering brokers and broker resources," on page 899
- Chapter 9, "Developing message flow applications," on page 1019
- Chapter 10, "Testing and debugging message flow applications," on page 3143
- Chapter 11, "Packaging and deploying," on page 3209
- Chapter 12, "Performance and monitoring," on page 3251
- Chapter 13, "Troubleshooting and support," on page 3345

Reference information

Reference topics provide supporting information that help you to complete the tasks. For example, they provide lists of commands, and tables of options and parameters.

- Security
- "Configuration and administration" on page 3657
- "z/OS configuration and administration specific information" on page 3979
- "Message flow development" on page 4015
- "Troubleshooting" on page 6864

Printing topics

You can print a single topic or a section of topics from this information center. Select a topic in the table of contents, then click the printer icon  . You can then choose to print just the selected topic, or the selected topic and all its subtopics.

Feedback

Your feedback on this information center is welcome. For example, you can report errors, identify missing information, or suggest improvement. A feedback link is included at the end of every topic; click this link to display a feedback form for the topic that you are viewing. Your feedback is logged in a database and is forwarded to the author of the topic.

For general feedback on the information center, see Feedback. This topic also contains alternative contact details (postal address, fax number, and email address) that you can use to include attachments with your feedback.

Links to more information

For more information about how to use this information center, and where to find product information about the Web, see the information center home page. The information center home page includes links to the following information:

- Migrating from previous versions
- WebSphere MQ family readme files
- IBM Redbooks®

- Articles on developerWorks®
- SupportPac offerings

Chapter 2. WebSphere Message Broker overview

This section provides introductory information to help you get started with IBM WebSphere Message Broker:

- "WebSphere Message Broker introduction"
- "What's new in Version 7.0?" on page 7
- Technical overview
  - Quick Tour
- Scenarios
- Start here
- "Samples" on page 98
- Legal information
- "Glossary of terms and abbreviations" on page 113
- "Accessibility features for WebSphere Message Broker" on page 135

WebSphere Message Broker introduction

You can use IBM WebSphere Message Broker to connect applications together, regardless of the message formats or protocols that they support.

This connectivity means that your diverse applications can interact and exchange data with other applications in a flexible, dynamic, and extensible infrastructure. WebSphere Message Broker routes, transforms, and enriches messages from one location to any other location:

- The product supports a wide range of protocols: WebSphere MQ, JMS 1.1, HTTP and HTTPS, Web Services (SOAP and REST), File, Enterprise Information Systems (including SAP and Siebel), and TCP/IP.
- It supports a broad range of data formats: binary formats (C and COBOL), XML, and industry standards (including SWIFT, EDI, and HIPAA). You can also define your own data formats.
- It supports many operations, including routing, transforming, filtering, enriching, monitoring, distribution, collection, correlation, and detection.

Your interactions with WebSphere Message Broker can be considered in two broad categories:

- Application development, test, and deployment. You can use one or more of the supplied options to program your applications:
  - Patterns provide reusable solutions that encapsulates a tested approach to solving a common architecture, design, or deployment task in a particular context. You can use them unchanged or modify them to suit your own requirements.
  - Message flows describe your application connectivity logic, which defines the exact path that your data takes in the broker, and therefore the processing that is applied to it by the message nodes in that flow.
  - Message nodes encapsulate required integration logic, which operates on your data when it is processed through your broker.
  - Message trees describe data in an efficient, format independent way. You can examine and modify the contents of message trees in many of the nodes that are provided, and you can supply additional nodes to your own design.

  - You can implement transformations by using graphical mapping, Java™, PHP, ESQL, and XSL, and can make your choice based on the skills of your workforce without having to provide retraining.
- Operational management and performance. WebSphere Message Broker includes the following features and functionality, which support the operation and performance of your deployment:
  - An extensive range of administration and systems management options for developed solutions.
  - Support for a wide range of operating system and hardware platforms.
  - A scalable, highly performing architecture, based on requirements from traditional transaction processing environments.
  - Tight integration with software products, from IBM and other vendors, that provide related management and connectivity services.

  WebSphere Message Broker is available in several modes, so that you can purchase a solution that meets your requirements. For more information, see "Operation modes" on page 48.

Application development

Your message processing applications, which you can run on more than 30 industry platforms, can connect to the broker by using one of the supported protocols already listed. Platforms from IBM, Microsoft, Oracle, and others are supported.

Diverse applications can exchange information in widely differing formats, with brokers handling the processing required for the information to arrive in the right place in the correct format, according to the rules that you have defined. The applications need only to understand their own formats and protocols, and not standards used by the applications to which they are connected.

Applications also have much greater flexibility in selecting which messages they want to receive, because you can apply filters to control the messages that are made available to them.

WebSphere Message Broker provides a framework that contains a wide variety of supplied, basic, functions along with user-defined enhancements, to enable rapid construction and modification of message processing rules.

Your applications can be integrated by providing message and data transformations in a single place, the broker. This integration helps to reduce the cost of application upgrades and modifications. You can extend your systems to reach your suppliers and customers, by meeting their interface requirements within your brokers. This ability can help you to improve the quality of your interactions, and allow you to respond more quickly to changing or additional requirements.

Messages are manipulated according to the rules that you define by using the WebSphere Message Broker Toolkit.

Operational management

WebSphere Message Broker supports a choice of interfaces for operation and administration of your brokers:

- The WebSphere Message Broker Toolkit

- The WebSphere Message Broker Explorer is a graphical user interface, based on the WebSphere MQ Explorer, for administering your brokers
- Applications that use the Administration API for WebSphere Message Broker (also known as the CMP API)
- A comprehensive set of commands, that you can run interactively or by using scripts

WebSphere Message Broker builds on the WebSphere MQ product, which provides assured, once-only delivery of messages between the applications. WebSphere MQ is included when you purchase WebSphere Message Broker.

WebSphere Message Broker is complemented by a wide variety of other IBM products such as Tivoli® Composite Application Manager for SOA, WebSphere Service Registry and Repository (WSRR), WebSphere Process Server, and WebSphere Transformation Extender (WTX).

Related concepts:

"WebSphere Message Broker technical overview" on page 27
WebSphere Message Broker enables information packaged as messages to flow between different business applications, ranging from large traditional systems through to unmanned devices such as sensors on pipelines.

"What's new in Version 7.0?"
Learn about the main new functions in IBM WebSphere Message Broker Version 7.0.

Related information:

- IBM Integration community
- WebSphere MQ Library web page
- WebSphere Transformation Extender Library web page
- WebSphere Service Registry and Repository Library web page
- WebSphere Business Process Management Information Center online
- Tivoli Composite Application Manager for SOA Information Center online

What's new in Version 7.0?

Learn about the main new functions in IBM WebSphere Message Broker Version 7.0.

Simplicity and productivity

In Version 7.0, both the product and its architecture have been simplified, and the product has fewer prerequisite products.

Streamlined components and prerequisite product requirements

Version 7.0 consists of a single runtime component, the broker. All commands and other programs now connect directly to a broker. The broker security model is now implemented by using WebSphere MQ queues, and therefore handles both brokers and queue managers.

Brokers maintain configuration data in the local file system

Brokers create and manage configuration data in an internal repository in the local file system, and have no requirement for a database. You can back up and restore the broker component and its internal repository by using the commands **mqsibackupbroker** and **mqsirestorebroker**.

Database support for message flows and user data is unchanged; however, supported versions of Relational Database Management Systems (RDBMS) (supplied by IBM and other vendors) have been updated on some platforms.

For more information about database support, see "Supported databases" on page 3591.

Optimized deployment and interactions with the broker

Applications that manage brokers and their resources connect directly to the broker. These applications include:

- The WebSphere Message Broker Explorer, an administration toolkit delivered with Version 7.0, which is described later in this section.
- The WebSphere Message Broker Toolkit.
- Commands; for example, **mqsideploy**, **mqsilist**, and **mqsistartmsgflow**.
- All applications that are written to the Configuration Manager Proxy (CMP) API, including the CMP Exerciser.

  For more details about the CMP API for Version 7.0, see "The Administration API for WebSphere Message Broker" on page 54.

You can control which users and applications are able to run commands against particular brokers or execution groups by using WebSphere MQ security, as described in the following section.

Administration security

Set up broker administration security to control the authority that is required by users to complete specific administrative tasks. You can enable security when you create a broker, or change it later on an existing broker. This option, which uses WebSphere MQ facilities, replaces Access Control Lists (ACLs) that were managed by the Configuration Manager in previous versions.

For further details, see "Broker administration security overview" on page 362.

For details of new and updated commands, see "Commands" on page 3672.

WebSphere Message Broker Explorer

The WebSphere Message Broker Explorer is an administration interface that is integrated as a plug-in into WebSphere MQ Explorer, so that you can administer both brokers and WebSphere MQ queue managers on local and remote computers.

Important status information is always on view, and you can access details about what each broker is doing, and has recently done. Configuration and other changes are monitored, and the user responsible for these changes is recorded.

The WebSphere Message Broker Explorer supports all the function that is provided by the Broker Administration perspective in the WebSphere Message Broker Toolkit in previous versions, and offers additional, more advanced features. Limited administration functionality is still available in the new Brokers view in the WebSphere Message Broker Toolkit to run a subset of operations.

The WebSphere Message Broker Explorer includes the following capabilities:

- You can create, delete, start, and stop local brokers without using the command line.
- You can see the relationships between your brokers and your queue managers.
- You can deploy a broker archive file to multiple execution groups in one step.
- You can see visualizations of your accounting and statistics data.
- You can see visualizations of your resource statistics data.
- You can configure broker properties, including creating and modifying configurable services.
- You can view the broker administration queue, and remove pending tasks that have been submitted to the broker.
- You can connect and configure settings for a DataPower® device.

For further details about the WebSphere Message Broker Explorer, see "WebSphere Message Broker Explorer" on page 57.

For further details about the Brokers view in the WebSphere Message Broker Toolkit, see "Brokers view" on page 6796.

Publish/subscribe support

All topic-based publish/subscribe operations are handled by WebSphere MQ. You can use WebSphere Message Broker facilities to extend publish/subscribe options to include content-based publishers and subscribers.

All applications use a single topic space that is managed by WebSphere MQ, and access control is handled by the queue manager. The concept of broker domains, which is valid in previous versions of WebSphere Message Broker, no longer exists; equivalent function for broker domains, broker topologies, and broker collectives is provided by WebSphere MQ clusters.

The Publication node uses WebSphere MQ publish/subscribe facilities. You can use the *NoMatch* terminal, new in Version 7.0, to identify scenarios in which no subscribers are registered to receive particular topics.

With these changes, your publish/subscribe network requires you to configure a broker only where you have content-based subscribers, not throughout the network. The content-based filters that you can specify can now include full ESQL expressions, including namespace support.

You can migrate JMS applications, and applications that use the MQRFH2 header, directly to WebSphere MQ.

WebSphere MQ Real-time Transport and WebSphere MQ Telemetry Transport nodes are no longer supported. Therefore, the following nodes have been removed:

- Real-timeInput

- Real-timeOptimizedFlow
- SCADAInput
- SCADAOutput

If you use the WebSphere MQ Real-time Transport, your applications can use equivalent qualities of service provided by WebSphere MQ. Migrate JMS real-time publishers and subscribers to the "read-ahead get" and "asynchronous-put" facilities of WebSphere MQ.

Message flows containing these nodes that you have migrated to WebSphere Message Broker Version 7.0 will not start until these nodes have been removed from the flow, and the flow has been redeployed.

Contact your account representative for more information about support for WebSphere MQ Real-time Transport and WebSphere MQ Telemetry Transport.

For more information about the changes to publish/subscribe see "Routing using publish/subscribe applications" on page 2215.

WebSphere Message Broker Toolkit

You can administer local and remote brokers in the Brokers view, which is new in Version 7.0. This view is integrated into the Broker Application Development perspective, so that you can access basic administration tasks while you are developing, deploying, debugging, and testing your applications.

The WebSphere Message Broker Toolkit includes an impact analysis tool that you can use to discover interdependencies between resources, and assess the effects of planned changes to those resources. See "Impact analysis: analyzing the effects of planned changes to your applications" on page 1150.

Samples are now accessible through the Samples and Tutorials page, and from the information center in the WebSphere Message Broker Toolkit. For a full list of the samples, see "Samples" on page 98.

Patterns

A pattern is a reusable solution that encapsulates a top-down tested approach to solving a common architecture, design, or deployment task in a particular context. This approach complements bottom-up development of creating message flows and nodes.

A number of patterns are supplied in the WebSphere Message Broker Toolkit, and you can use the Patterns Explorer, which includes comprehensive help, to simplify creation of common scenarios.

You can configure these patterns with values for use in your own environment to solve specific business problems. The supplied patterns use preferred techniques in message flow design, to produce efficient and reliable flows.

For more information, see "Patterns" on page 1310.

Universal connectivity for SOA

Additional nodes and configurable services expand the interaction of the broker with other products.

SCA nodes for WebSphere Process Server

Five new built-in message flow nodes are provided to improve the interaction between WebSphere Message Broker and WebSphere Process Server Version 6.2 by using Web Services (SOAP over HTTP) or WebSphere MQ bindings.

The nodes are the SCAInput, SCAReply, SCARequest, SCAAsyncRequest, and SCAAsyncResponse nodes.

For more information, see "Service Component Architecture (SCA) overview" on page 2096.

Enhanced support for the PHPCompute node

Support for the PHP scripting language is available on all operating systems on which WebSphere Message Broker is supported, except Solaris on x86-64. The PHPCompute node supports general-purpose transformation logic in the PHP language, and complements the Compute, JavaCompute, XSLTransform, and Mapping nodes. In addition, the set of supported PHP extensions has been increased. For more information, see "Using PHP" on page 2670 and "PHP extensions" on page 5324.

SAP, Siebel, and Peoplesoft enhancements

Version 7.0 improves connectivity with Enterprise Information Systems.

New WebSphere Adapter nodes

Use the SAPReply node to send a reply to an SAP synchronous callout. Use this node with an SAPInput node to implement a message flow application that acts as a remote function call (RFC) destination.

For more information, see "SAPReply node" on page 4682 and "BAPI inbound scenarios" on page 1943.

Generic IDoc routing

By using the SAPInput node in passthrough mode, WebSphere Message Broker can receive any IDoc, and route it according to IDoc type. By using this method, you can also use a single RFC program ID to receive all IDoc types, while still allowing individual IDoc processing.

For more information, see "Generic IDoc routing" on page 1976.

SAP high availability

You can deploy an SAP adapter and a message flow that contains an SAPInput node to two brokers on your network; these brokers can accept IDocs concurrently from the same SAP system so that you can build a highly available environment.

On distributed systems, two brokers share state by using queues on a third queue manager, which is running in multi-instance mode. Each broker has client connections to that queue manager.

On z/OS®, the shared state is stored on a shared queue. Each broker connects to the queue sharing group.

For more information, see "SAP high availability" on page 1947.

Iterative discovery

You can take an adapter component that was created by using the Adapter Connection wizard in WebSphere Message Broker Version

7.0, and update it with newly discovered objects from the Enterprise Information System (EIS) by running the Adapter Connection - Iterative Discovery wizard. This facility is known as *iterative discovery*. You can either add the new objects without modifying existing objects, or replace existing objects.

For more information, see "Enhancing existing adapters with newly discovered objects" on page 2063.

Iterative deployment

If your message flow acts as a gateway to an EIS, you can use it to call new services that did not exist when you developed the flow. You can also create an event handler to an EIS to handle new event types that did not exist when you first developed your message flow. In both cases, if a new service or event is provided by the EIS, you do not have to modify and retest the message flow. This facility is called *iterative deployment*.

For more information, see "WebSphere Adapters deployment" on page 3219.

Sequence and Resequence nodes

WebSphere Message Broker provides support for adding sequence numbers to messages, and for reordering messages in the message flow based on their sequence number. You can use the new Sequence node to add sequence numbers to the messages, and the new Resequence node to reorder the messages into their original sequential order.

For more information, see "Sequence node" on page 4736 and "Resequence node" on page 4651.

New configurable services for EDA nodes

You can use the following configurable services to define the WebSphere MQ queues on which EDA nodes store event state:

- Aggregation
- Collector
- Resequence
- Timer

You can also use these configurable services to specify timeouts for the nodes. For more information, see "Configurable services properties" on page 3766.

Dynamic operation management

Additional features provide better information and control of operations.

Multi-instance brokers

WebSphere Message Broker builds on the multi-instance queue manager support introduced in WebSphere MQ Version 7.0.1 to provide a highly available configuration with active and passive brokers.

Multi-instance brokers and queue managers store their configurations on shared network storage so that if a failure occurs in an active component, the passive component assumes the configuration and operation of the

active component. The use of queue managers in this way avoids the requirement for a high availability solution, such as HACMP™, supplied by a vendor software company.

For further information, see "Configuring for high availability" on page 826.

Audit and monitoring

You can now generate comprehensive audit and monitoring events from message flows, either at design time or operationally, for new and existing message flows. These events can be consumed by a diverse range of applications and systems, including WebSphere Business Monitor, WebSphere MQ and JMS applications, and vendor applications. In addition to business monitoring, you can use these events for business intelligence, and audit scenarios.

See "Business-level monitoring" on page 3319 for an overview of monitoring.

The following improvements to the monitoring of message flows are introduced:

- A filter can be applied to every event source, to control whether the event is emitted. See "Configuring monitoring event sources using monitoring properties" on page 3327 and "Monitoring profile" on page 6768.
- You can export the monitoring information about a message flow from the WebSphere Message Broker Toolkit, and import it into WebSphere Business Monitor Version 7.0 to generate a monitor model for your message flow. See "Creating a monitor model for WebSphere Business Monitor V7" on page 3341.
- You can choose whether the emission of monitoring events by a message flow is coordinated with the message flow transaction, is in an independent unit of work, or is not in a unit of work, which improves overall monitoring performance; see "Monitoring basics" on page 3320.
- Monitoring events now contain an integer counter, as well as the creation time of the events, for use in sequencing events. The Sequence tab has been removed from the WebSphere Message Broker Toolkit.

Resource statistics

You can collect statistics for some of the resources that are used by execution groups in the broker to help with problem diagnosis and broker optimization. Supported resources are the Java Virtual Machine (JVM), and the outbound sockets. For example, you can monitor the sockets that are used by SOAP nodes in your message flows.

You can start and stop statistics collection at broker or execution group level by using the WebSphere Message Broker Explorer, the CMP API, or the **mqsichangeresourcestats** command.

The resource statistics framework is based on the existing accounting and statistics for message flows, and generates periodic messages as publications that your programs can subscribe to. You can also view these statistics in the WebSphere Message Broker Explorer, which provides both numeric and graphical representations.

For further details, see "Monitoring resource performance" on page 3305.

Service Federation Management enablement

You can enable brokers for Service Federation Management. For further information, see “Working with Service Federation Management (SFM)” on page 911.

Platforms and environments

The Version 7.0 broker operates in 64-bit mode on z/OS and all distributed platforms, except Linux on x86 and Windows systems.

Platforms

- The following platforms now support only 64-bit operation, and have a smaller installation footprint:
  - AIX®
  - Linux on x86-64
  - Solaris on SPARC

  See “Migrating a Version 6.1 broker to Version 7.0 on distributed operating systems” on page 173 or “Migrating a Version 6.0 broker to Version 7.0 on distributed operating systems” on page 194 for configuration details.
- WebSphere Message Broker Version 7.0 is not supported on HP-UX on PA-RISC. If you use this platform, retain Version 6.1, and speak to your IBM representative about your requirements.

Java

Java 1.6 (Version 6) is supported in all environments. On IBM platforms, the IBM J9 engine is supplied, which benefits from reduced startup time and memory footprint.

Migration

You can migrate to WebSphere Message Broker Version 7.0 from WebSphere Message Broker Version 6.1, WebSphere Message Broker Version 6.0, and WebSphere Event Broker Version 6.0. You can also install WebSphere Message Broker Version 7.0 to coexist with previous versions on the same computer.

For details about how to migrate your components and data, and how components from different versions can interact, see Chapter 3, “Migrating and upgrading,” on page 137.

Related concepts:

“New function added in Version 7.0 fix packs” on page 15
Some fix packs and other maintenance packs deliver new functions.

Related tasks:

Chapter 3, “Migrating and upgrading,” on page 137
To migrate a broker domain to WebSphere Message Broker Version 7.0, plan your migration strategy, perform pre-migration tasks, migrate your domain components, and then complete post-migration tasks.

Related reference:

“Reviewing technical changes in Version 7.0” on page 205
Some minor changes in behavior are present in WebSphere Message Broker Version 7.0; for example, those changes caused by defects that have been fixed between versions.

New function added in Version 7.0 fix packs

Some fix packs and other maintenance packs deliver new functions.

The "What's new in Version 7.0?" on page 7 topic introduces you to the main new function in WebSphere Message Broker Version 7.0. This topic introduces you to the additional function that has been added in fix packs.

For detailed information about the contents of fix packs and other maintenance packs, see the WebSphere Message Broker support web page. Click **Download**, then **Recommended Fixes**, and select your product to view available fixes. The description of each fix pack includes links to **Release notes** (details of its content) and **Problems fixed** (a list of PMRs, APARs, and defects that are included).

To enable function that becomes available in WebSphere Message Broker fix packs, use the **-f** parameter on the **mqsichangebroker** command, as shown in the following example:

```
mqsichangebroker MB7BROKER -f 7.0.0.7
```

For more information about this command, see "**mqsichangebroker** command" on page 3723.

Fix pack V7.0.0.7 enhancements

Setting up the JNDI interface for the proxy servlet

The proxy servlet initialization parameters must be configured for the broker environment that the proxy servlet is connecting to each time the proxy servlet is deployed to the servlet container. It is now possible to configure the `web.xml` parameters only once through the JNDI in WebSphere Application Server, regardless of how many future deployments there might be of the proxy servlet. Because the JNDI configuration parameters take precedence over the initialization parameters in the `web.xml` file, using this method means that you need to set up at the application server side only once for any future deployments of the proxy servlet.

These setup tasks must all be completed in the WebSphere Application Server administrative console. For a full list of the steps, see "Setting up the JNDI interface for the proxy servlet" on page 886.

Configuring the XPath cache

An Execution Group keeps a cache of compiled XPath expressions to help reduce the processor usage of parsing and re-creating XPath expressions that are used repeatedly during Message Flow execution. This cache is shared by all Message Flows within an Execution Group. However, this default size might become a performance bottleneck for customers who use many XPath expressions with a single flow invocation completely invalidating the cache. Altering the size of the XPath cache might improve message flow performance. For examples and information about altering the default cache size, see "Configuring the XPath cache" on page 765.

Fix pack V7.0.0.6 enhancements

Setting FTP and SFTP servers dynamically

You can override the `Remote server and port` property on the FileOutput node by setting a value in the local environment.

For more information, see “Local environment overrides for the FileOutput node” on page 4443.

Verify the ODBC environment on Linux and UNIX systems

The **mqsicvp** command is run automatically when you start a broker by using the **mqsistart** command. The command checks that the broker environment is set up correctly. This checking has been enhanced to verify that the ODBC environment is configured correctly on Linux and UNIX systems. When you run this command from the command line on Linux and UNIX systems, it also validates the connection to all data sources that are listed in the `odbc.ini` file that have been associated with the broker by using the **mqsisetdbparms** command.

For more information, see “**mqsicvp** command” on page 3857.

Display the full content of BIP messages

You can view the full content of a runtime BIP message, including the user response and explanation sections, by using the **mqsiexplain** command.

For more information, see “**mqsiexplain** command” on page 3879.

Start an execution group with a different user ID to a broker on z/OS systems

On z/OS systems, you can change the user ID under which an execution group runs so that it can access resources according to the permissions assigned to it, rather than the permissions assigned to the main broker user ID.

For more information, see “Execution group user IDs on z/OS” on page 560.

New `ConnectionIdleTimeoutSec` property on the IMSConnect configurable service

You can use the **ConnectionIdleTimeoutSec** property on the IMSConnect configurable service to specify the idle connection timeout for cached IMS Connections. If a cached IMS connection is inactive for more than the specified **ConnectionIdleTimeoutSec**, then this connection becomes ineligible for reuse, and is removed from the cache, and closed cleanly.

For more information, see “Configurable services properties” on page 3766.

Configure the properties of an execution group while it is in offline mode

You can use the **-f** parameter on the **mqsireportproperties** and **mqsichangeproperties** commands to view and change properties on an execution group while it is in offline mode.

For more information, see “**mqsichangeproperties** command” on page 3756.

Additional problem collector for IBM Support Assistant Data Collector

Use the broker problem collector, which is installed with IBM Support Assistant Data Collector, to gather more extensive broker diagnostic documents.

For more information, see “Selecting a problem collector for IBM Support Assistant Data Collector” on page 3568.

New `timeoutThreads` property

An optional property that assigns additional processing threads to enable processing of timed out aggregation messages in the AggregateReply node. For more information, see "Processing timed out aggregation messages" on page 2739.

Fix pack V7.0.0.5 enhancements

Updates to the FileOutput node Basic properties

Support has been added for writing directly to an output file; see "FileOutput node" on page 4430 for more information.

Java shared classloader

A new shared classloading option has been introduced:

- **Execution group classloading** – allows only a single defined execution group to access and load any JAR files that are placed in the execution group `shared-classes` directory.

A new classloading precedence order is also defined.

For more information, see "Java shared classloader" on page 2637.

Support for `mqsimode` command on z/OS

The **`mqsimode`** command can be run on z/OS by customizing and submitting BIPMODE; see "**`mqsimode`** command" on page 3899 for more information.

New `jdbcProviderXASupport` property

An optional property that controls whether the broker connects to a database server using XA Protocol. For more information, see "Setting up a JDBC provider for type 4 connections" on page 684.

IBM Support Assistant Data Collector

Using IBM Support Assistant Data Collector, which is installed with WebSphere Message Broker, you can collect diagnostic documents and submit a problem report to IBM.

For more information, see "IBM Support Assistant Data Collector" on page 3565.

WebSphere MQ Version 7.1 and Version 7.5

This fix pack introduces support for WebSphere MQ Version 7.1 and Version 7.5 within WebSphere Message Broker, with the following restrictions.

- WebSphere MQ Version 7.1 and Version 7.5 must be configured as the primary installation.
- WebSphere MQ Version 7.1 and Version 7.5 must be installed in the default install location on AIX, HP-UX, Linux, and Solaris.

For the latest details of all supported levels of hardware and software, visit the WebSphere Message Broker Requirements website.

Fix pack V7.0.0.4 enhancements

New nodes for IBM Sterling Connect:Direct®

IBM Sterling Connect:Direct is a managed file transfer product that transfers files between, and within, enterprises. The following new nodes have been added to implement the additional features:

- "CDInput node" on page 4305
- "CDOutput node" on page 4312

WebSphere Message Broker nodes work as clients, connecting to an external Connect:Direct server through the IBM Sterling Connect:Direct Java Application Interface. For an overview of IBM Sterling Connect:Direct, see "IBM Sterling Connect:Direct overview and concepts" on page 1810.

Using LocalEnvironment variables with JMSOutput and JMSReply nodes
Use the DestinationData element with DestinationName and DestinationType child elements to specify the name and type of the JMS Destination. For more information, see "Using LocalEnvironment variables with JMSOutput and JMSReply nodes" on page 4242.

Creating a security profile for LDAP
Use the rejectBlankpassword property when configuring a security profile for Lightweight Directory Access Protocol (LDAP), to specify whether the security manager rejects a user name that has an empty password token without passing it to LDAP. For more information, see "Creating a security profile for LDAP" on page 435, "Configurable services properties" on page 3766, and "SecurityProfiles configurable service properties" on page 3801.

New HTTPConnector property autoRespondHTTPHEADRequests
Use the autoRespondHTTPHEADRequests property to specify if the connector handles HEAD requests for HTTP traffic in the same way that it does for SOAP traffic. For more information, see "Execution group HTTP listener parameters (SOAP and HTTP nodes)" on page 3805.

Millisecond timeouts on HTTPRequest and SOAPRequest nodes
Use the TimeoutMillis local environment setting to define a timeout in milliseconds and override the Request timeout (sec) property on the node. For more information, see "HTTPRequest node" on page 4488 and "Local environment overrides for the SOAPRequest node" on page 4850.

Proxy servlet configuration for SSL connections
Use the SSL connection options when deploying the proxy servlet `web.xml` to the servlet container. For more information, see "Proxy servlet configuration parameters" on page 878.

Support for additional data types on DB2®
GRAPHIC, VARGRAPHIC, LONGVARGRAPHIC, and DBCLOB data type support on DB2 is now provided for the broker. For more information, see "Support for Unicode and DBCS data in databases" on page 3668.

Fix pack V7.0.0.3 enhancements

HTTP Transport property hostnameChecking
Use the HTTP Transport property hostnameChecking to specify whether the host name of the server that is receiving the request must match the host name in the SSL certificate. For more information, see "HTTPRequest node" on page 4488, "SOAPRequest node" on page 4828, and "SOAPAsyncRequest node" on page 4750.

TDS Mnemonics
Use the TDS mnemonic string <X12_ERS> as an element repetition separator for X12. For more information, see "TDS Mnemonics" on page 5391.

Policy Sets and Policy Set Bindings editor
Use the **mustUnderstand** attribute in the Policy Sets and Policy Set Bindings editor to configure the security header of the consumer message. For more information, see "Policy Sets and Policy Set Bindings editor: Advanced panel" on page 6863.

New configurable service properties

The following configurable service properties are new for 7.0.0.3:

- Use the JDEdwardsConnection configurable service property assuredOnceDelivery to specify whether to provide assured once-only delivery for inbound events. For more information, see "Configurable services properties" on page 3766.
- Use the MonitoringProfiles configurable service property useParserNameInMonitoringPayload to determine when the payload is included in a monitoring message. For more information, see "Configurable services properties" on page 3766 and "Configuring monitoring event sources using a monitoring profile" on page 762.
- Use the IMSConnect configurable service property CodedCharSetID to change your IMS system or IMSConnect CCSID from the default value. For more information, see "Changing connection information for the IMSRequest node" on page 732 and "Configurable services properties" on page 3766.
- Use the FtpServer configurable service property preserveRemoteFileDate to specify whether files that are retrieved from a remote server by the FileInput node retain the last modified date on the server. For more information, see "FtpServer configurable service properties" on page 3794 and "Configurable services properties" on page 3766.

Fix pack V7.0.0.2 enhancements

Simplicity and productivity

To simplify the product and increase productivity, WebSphere MQ Telemetry Transport is now supported by WebSphere MQ, and you can do more with patterns.

WebSphere MQ Telemetry Transport

WebSphere MQ Telemetry Transport is supported from WebSphere MQ. For more information, see "Changes to nodes in WebSphere Message Broker Version 7.0" on page 2217.

Modifying pattern instances by using Java or PHP

Use Java or PHP code to modify pattern instances when the pattern user generates an instance of a user-defined pattern. For example, to modify the structure of a message flow that is based on the values of pattern parameters.

For more information, see "Modifying pattern instances by using Java or PHP" on page 1364.

Share your user-defined patterns with other users

Package your user-defined pattern into a pattern archive so that the user-defined pattern can be distributed to pattern users by adding the pattern archive to a pattern community site.

For more information, see "Packaging and distributing pattern plug-ins" on page 1397.

Universal connectivity for SOA

Additional nodes and configurable services expand the interaction of the broker with other products.

CICSRequest node enhancements

- You can specify a mirror transaction name on the CICSRequest node, which you can use to run CICS® Transaction Server for z/OS tasks and programs. This grouping greatly assists in collecting statistics, accounting, and aids decision making about task priority. For more information about mirror transactions, see "CICS Transaction Server for z/OS mirror transactions" on page 2189.
- The CICSRequest node support in WebSphere Message Broker provides direct communication with CICS (two-tier connection) by sending Distributed Program Link (DPL) requests over TCP/IP-based IP InterCommunications (IPIC) protocol, or communication with CICS through CICS Transaction Gateway for Multiplatforms (three-tier connection). For more information about the two-tier and three-tier connection models, see "CICS Transaction Server for z/OS overview" on page 2173 for a high-level overview, or "CICS Transaction Server for z/OS two-tier connectivity" on page 2177 and "CICS Transaction Server for z/OS three-tier connectivity" on page 2181 for detailed conceptual information.
- You can specify either a COMMAREA data structure or a channel data structure on the CICSRequest node to use as input for linking to CICS programs. The data structure that is specified as input returns the same data structure as output. Channels are an alternative for COMMAREAs, providing relief from the COMMAREA maximum size of 32766 bytes, and allowing greater flexibility in input and output data structures. For more information about using a COMMAREA or channel data structure, see "COMMAREA or channel data structures" on page 2183.
- The CICS Transaction Server for z/OS Channel Connectivity sample demonstrates how to call a channel-based CICS program. A CICS channel structure can be represented in WebSphere Message Broker by a message collection. This sample demonstrates how to create and populate a message collection for the CICSRequest node and how to process the collection after the call.

New EmailInput node

Use the EmailInput node to retrieve an email, with or without attachments, from an email server that supports Post Office Protocol 3 (POP3) or Internet Message Access Protocol (IMAP).

For more information, see "EmailInput node" on page 4394.

A sample that demonstrates how you can use the EmailInput node is also provided. For more information, see Email.

New JDEdwardsInput and JDEdwardsRequest nodes

Use the JDEdwardsInput and JDEdwardsRequest nodes to interact with a JD Edwards EnterpriseOne server. For example, you can use the JDEdwardsRequest node to discover JD Edwards EnterpriseOne business functions, XML lists, and real-time events.

For more information, see "JDEdwardsInput node" on page 4519 and "JDEdwardsRequest node" on page 4524.

A sample that demonstrates how you can use the "JDEdwardsRequest node" on page 4524 node is also provided. For more information, see JD Edwards Connectivity.

New FileRead node

Use the FileRead node to read one record, or the entire contents of a file, from within a message flow.

For more information, see "FileRead node" on page 4444.

The Message Routing sample has been enhanced, and now demonstrates how to use the FileRead node. For more information about how to process messages that are based on the contents of an XML or CSV file, see Message Routing.

FileInput node

Skip the first record in a file. The FileInput node reads the first record in the file but does not propagate the record to the Out terminal. Records are propagated as normal, from the second record onwards. Use this option when the first record is a header that does not need to be processed. It is not valid to use this option when using the whole file.

For more information, see "FileInput node" on page 4415.

SSL for TCP/IP nodes

You can configure the broker to use SSL for TCP/IP connections; see "Configuring TCP/IP client nodes to use SSL" on page 551.

HTTP compression for HTTP and SOAP nodes

You can configure HTTP nodes to use HTTP compression and decompression when sending and receiving messages. Similar HTTP compression functionality was introduced for the SOAP nodes in fix pack 7.0.0.1. For more information, see "Using compression with HTTP and SOAP nodes" on page 1597.

JMS transport for SOAP nodes

The SOAPAsyncRequest and SOAPAsyncResponse nodes support JMS as well as HTTP transport. You can import WSDL with bindings for both JMS and HTTP transport, and switch transports for the SOAPAsyncRequest node during a message flow. WS-Security and WS-Addressing are supported for SOAP/JMS, as well as transactionality. WebSphere Message Broker supports both W3C (standard) and IBM (proprietary) WSDL formats for SOAP/JMS. For more information, see "WSDL URI formats for JMS" on page 1668 and "SOAP over JMS" on page 6698.

Web services gateway

The SOAP nodes support a web services gateway mode, which does not require a WSDL to configure the SOAP nodes, and allows WebSphere Message Broker to handle generic SOAP request/response and one-way messages when used as a web services provider or consumer. WebSphere Message Broker can also act as a façade between multiple web service clients and multiple back-end web service providers.

For more information, see "Gateway operation mode for SOAP nodes" on page 1645.

A sample that demonstrates how you can use a web services gateway is also provided. For more information, see Web Services Gateway.

DatabaseInput node

The DatabaseInput node can now generate the code for a simple database query. After code generation, you can add custom code; see “Configuring a DatabaseInput node” on page 2120. By using an MQInput node with the built-in DatabaseInput node the WBI JDBC Adapter Migration sample re-creates a scenario of migrating a JDBC adapter to invoke a message flow.

A sample that demonstrates how you can use the DatabaseInput node is also provided. For more information, see DatabaseInput Node.

JSON domain

Fix pack V7.0.0.2 provides support for a JSON domain. Messages in the JSON domain are processed by the JSON parser and serializer. The JSON parser interprets a bit stream by using the JSON grammar, and generates a corresponding JSON domain logical message tree in the broker.

For more information, see “JSON parser and domain” on page 1128.

RESTful Web Service Using JSON sample

This sample shows how you can use WebSphere Message Broker to front an existing service as a RESTful web service, providing a JSON message format interface. The sample also shows how to consume the RESTful Web Service from a message flow.

For more information, see RESTful Web Service Using JSON.

Web Service Aggregation sample

The sample demonstrates how you can invoke a number of web services and amalgamate the results by using WebSphere Message Broker aggregation nodes. The sample illustrates how you can use aggregation for transports other than WebSphere MQ, and highlights any issues of which to be aware. For more information, see Web Service Aggregation.

Improved *order by* **support for the MQInput node**

You can now sort by any element in the message. For each value of that element, the messages are processed in arrival order. See “Optimizing message flow throughput” on page 587 and “MQInput node” on page 4594.

Dynamic operation management

Additional features provide better information and control of operations.

Resource statistics for parsers

View the statistics for a parser to view the number of input and output messages that are processed by a message flow, and determine if message flow parsers are using large amounts of memory. For more information, see “Resource statistics” on page 3306.

Platforms and environments

You can interact with more platforms and environments.

Configuring JMS nodes with Oracle AQ

You can configure the JMS nodes to communicate with Oracle AQ (Oracle 11g and above).

For more information, see “Configuring JMS nodes to communicate with Oracle AQ” on page 1712.

Integrating with the Rational® Team Concert client
You can integrate the Rational Team Concert client with the WebSphere Message Broker Toolkit.

For more information, see "Integrating the Rational Team Concert client with the WebSphere Message Broker Toolkit" on page 576

Execution group profiles
You can create and use profiles that apply to a specific execution group.

For more information, see "Execution group-specific command environment: Windows systems" on page 309 and "Execution group-specific command environment: Linux and UNIX systems" on page 312.

Fix pack V7.0.0.1 enhancements

- "Simplicity and productivity"
- "Universal connectivity for SOA" on page 24
- "Platforms and environments" on page 26

Simplicity and productivity

Additional patterns are provided to help solve specific business problems.

User-defined patterns
User-defined patterns extend the function of WebSphere Message Broker so that you can create patterns that you can reuse within your organization.

For more information, see "User-defined patterns" on page 1334.

Solar Pattern Authoring sample
This sample shows how you can build a user-defined pattern. The sample provides an example message flow project that calculates the sunrise and sunset times in a PHPCompute node. The sample also provides a pattern authoring project that configures a pattern.

For more information, see "Samples" on page 98.

Service Access from WebSphere MQ: one-way pattern
Use this pattern to process WebSphere MQ XML messages by using the data that the pattern contains to call a web service. Use this pattern to bridge the reliable WebSphere MQ messaging protocols of a client application with the synchronous requests to services to handle updates with an assurance that service failures, including timeouts, are reliably reported.

This pattern provides loose coupling between client applications and service providers in timing, protocols, and transport. It is appropriate for service interfaces to existing systems.

For more information, see "Built-in patterns" on page 1332.

Using a subflow as a user-defined node
Develop a user-defined node that packages a subflow, either in the same way that you create any other user-defined node that has its implementation based on Java, or by basing it on an existing subflow.

For more information, see "Using a subflow as a user-defined node" on page 3008.

Universal connectivity for SOA

Additional nodes and configurable services expand the interaction of the broker with other products.

New CORBARequest node
Use the CORBARequest node to call an external CORBA application over Internet Inter-Orb Protocol (IIOP). You can use this node to create a new external interface for existing CORBA applications.

For more information, see "CORBARequest node" on page 4349.

A sample that demonstrates how to use the CORBARequest node is also provided. For more information, see "Samples" on page 98.

New CICSRequest node
Use the CICSRequest node to call an external CICS Transaction Server for z/OS application over TCP/IP-based IP InterCommunications (IPIC) protocol. By using the CICS support that is provided in WebSphere Message Broker you can deploy CICS applications into a service-oriented architecture (SOA).

For more information, see "CICSRequest node" on page 4321.

A sample that demonstrates how to use the CICSRequest node is also provided. For more information, see "Samples" on page 98.

New WebSphere MQ File Transfer Edition nodes
The FTEOutput and FTEInput nodes transfer files across an existing WebSphere MQ File Transfer Edition network in a timely and reliable manner.

For more information, see "Managed file transfers using WebSphere MQ File Transfer Edition" on page 1869.

A sample that demonstrates how to use the FTEOutput and FTEInput nodes is also provided. For more information, see "Samples" on page 98.

New DatabaseInput node
The DatabaseInput node enables a message flow to respond to changes to data in a database.

For more information, see "Event-based database integration" on page 2118.

Samples that demonstrate how to use the DatabaseInput node are also provided. For more information, see "Samples" on page 98.

You can now create a message definition from a database definition (`.dbm` file); see "Creating a message definition from a database definition" on page 2941.

The New Message Set wizard now includes an option to specify database records; see "Creating a message set" on page 2842.

New SecurityPEP node
The SecurityPEP node enables you to invoke the message flow security manager at any point in the message flow between an input node and an output or request node.

For more information, see "SecurityPEP node" on page 4729.

A sample that demonstrates how to use the SecurityPEP node is also provided. For more information, see "Samples" on page 98.

JMS transport for SOAP nodes

The SOAPInput, SOAPReply, and SOAPRequest nodes support JMS as well as HTTP transport. You can import WSDL with bindings for both JMS and HTTP transport, and switch transports during a message flow. WS-Security and WS-Addressing are supported for SOAP/JMS. WebSphere Message Broker supports both W3C (standard) and IBM (proprietary) WSDL formats for SOAP/JMS. For more information, see "WSDL URI formats for JMS" on page 1668.

For more information, see "SOAP over JMS" on page 6698.

Querying WSDL with ?wsdl

A web service client can send an HTTP GET request with a `?wsdl` query string to message flows implementing web services, and receive a representation of the WSDL that was used to configure the input node that provides the endpoint for the service.

For more information, see "Using WSDL to configure message flows" on page 1664.

HTTP nodes can use the embedded listener in an execution group

You can configure your broker and execution groups so that the HTTP nodes use the listener within the execution group to communicate with clients and servers, in preference to the broker-wide listener. The latter configuration remains the default option, but the execution group listener might enhance message flow throughput.

For more information, see "HTTP listeners" on page 1589.

New HTTP Timeout terminal on HTTPInput and SOAPInput nodes

You can configure your HTTPInput and SOAPInput nodes to connect timeout processing nodes to the HTTP Timeout terminal.

On SOAPInput nodes, messages are propagated through this terminal only when you are using an HTTP binding. On HTTPInput nodes, messages are propagated through this terminal only when you have configured your broker and execution groups such that the HTTPInput node is using the embedded execution group listener.

For more information, see "Using timeouts with HTTP and SOAP nodes" on page 1595, "SOAPInput node" on page 4795, and "HTTPInput node" on page 4474.

Securing the connection to IMS by using SSL

You can use the IMSConnect configurable service to configure the IMSRequest node to use Secure Sockets Layer (SSL) protocol. For more information, see "Securing the connection to IMS by using SSL" on page 549.

Propagating security credentials to IMS

The IMSRequest node can use an identity that is present on an input message, and propagate it to IMS, by using the `Propagate` property on the security profile that is defined for the node. For more information, see "Propagating security credentials to IMS" on page 2144.

Closing unused connections to Enterprise Information Systems

To effectively maintain the pool of connections to SAP, Siebel, or PeopleSoft, you can set a connection timeout value on a configurable

service. The value determines how long a connection can be idle before it is closed. For more information, see “Configuring EIS connections to expire after a specified time” on page 726.

Propagating security credentials to SAP

The SAPRequest node can use an identity that is present on an input message, and propagate it to SAP, by using the `Propagate` property on the security profile that is defined for the node. For more information, see “Propagating security credentials to an SAP request” on page 2065.

New WS-Trust V1.3 compliant security token server (STS) support for message flow security

You can use a WS-Trust V1.3 compliant STS, such as Tivoli Federated Identity Manager (TFIM) V6.2, for message flow security to provide authentication, mapping, and authorization of the following additional security tokens:

- SAML assertions
- Kerberos tickets
- LTPA tokens
- RACF® PassTickets
- Universal WSSE tokens

This support is in addition to the existing support for Username, Username and password, and X.509 certificates. For more information, see “Identity” on page 390.

The support for this new security provider is in addition to the existing support for Lightweight Directory Access Protocol (LDAP) and TFIM V6.1. For more information, see “Message flow security” on page 382.

New WS-Security support for SOAP nodes

You can use the SOAP node support for Kerberos and SAML pass-through to provide SOAP message security. For more information, see “WS-Security” on page 765.

Platforms and environments

Support for the Windows platform is enhanced, to include Windows 7 and Windows Server 2008 R2. You can also now create 64-bit brokers and execution groups with the new Windows on x86-64 version of the product.

WebSphere Message Broker ODBC Database Extender package

This package is required when you are using WebSphere Message Broker to interface with an ODBC data source that is not supported through the DataDirect ODBC drivers.

For further information, see “Installing the WebSphere Message Broker ODBC Database Extender (IE02)” on page 273 and “Connecting to a database from Linux and UNIX systems using the WebSphere Message Broker ODBC Database Extender (IE02)” on page 682

Related concepts:

“What's new in Version 7.0?” on page 7
Learn about the main new functions in IBM WebSphere Message Broker Version 7.0.

Related tasks:

"Controlling the functional level of WebSphere Message Broker" on page 51
You can control the functional level of your broker to enable new functionality added in WebSphere Message Broker fix packs.

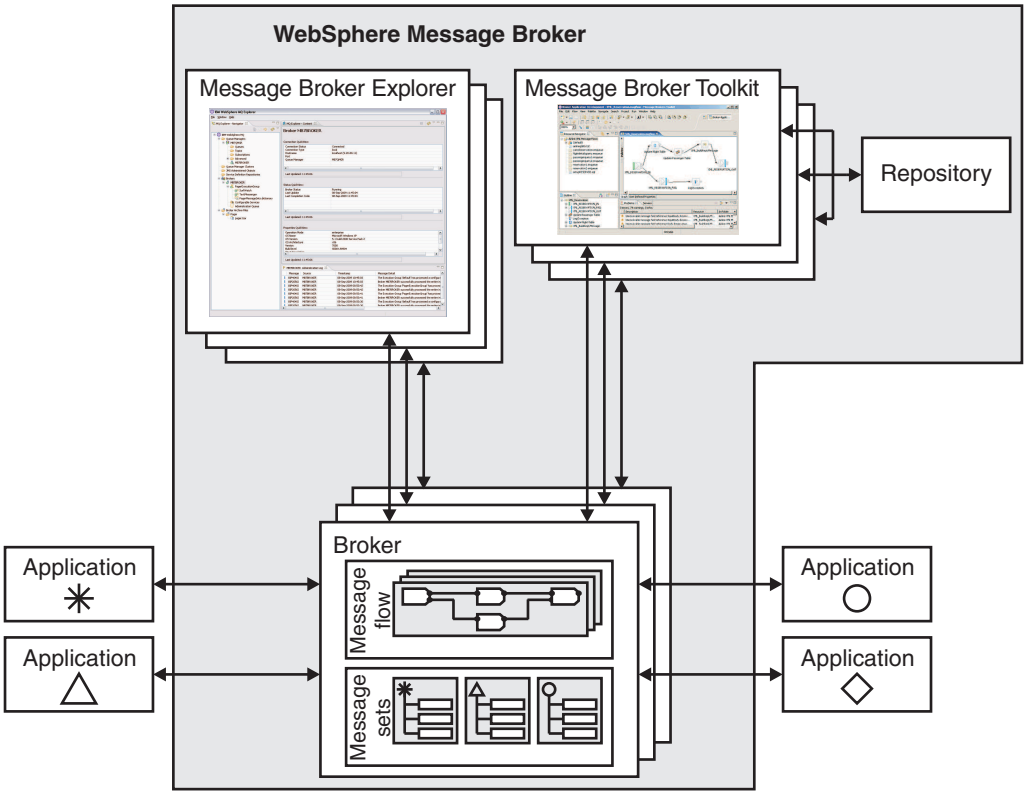
Chapter 3, "Migrating and upgrading," on page 137
To migrate a broker domain to WebSphere Message Broker Version 7.0, plan your migration strategy, perform pre-migration tasks, migrate your domain components, and then complete post-migration tasks.

Related reference:

"Reviewing technical changes in Version 7.0" on page 205
Some minor changes in behavior are present in WebSphere Message Broker Version 7.0; for example, those changes caused by defects that have been fixed between versions.

WebSphere Message Broker technical overview

WebSphere Message Broker enables information packaged as messages to flow between different business applications, ranging from large traditional systems through to unmanned devices such as sensors on pipelines.



For an animated introduction to the basic concepts and features of WebSphere Message Broker shown in the diagram, run the Quick Tour.

WebSphere Message Broker processes messages in two ways: message routing and message transformation.

Message routing

Messages can be routed from sender to recipient based on the content of the message.

The message flows that you design control message routing. A message flow describes the operations to be performed on the incoming message, and the sequence in which they are carried out.

Each message flow consists of the following parts:

- A series of steps used to process a message; see "Message flow nodes" on page 1024.
- Connections between the nodes, defining routes through the processing; see "Message flow connections" on page 1032.

IBM supplies built-in nodes and samples for many common functions. If you require additional functions, you can write your own user-defined nodes; see "User-defined extensions overview" on page 2971.

You create message flows in the WebSphere Message Broker Toolkit which is an integrated development and administration console.

Message transformation

Messages can be transformed before being delivered:

- They can be transformed from one format to another, perhaps to accommodate the different requirements of the sender and the recipient.
- They can be transformed by modifying, combining, adding, or removing data fields, perhaps involving the use of information stored in a database. Information can be mapped between messages and databases. More complex manipulation of message data can be achieved by writing code, for example in Extended SQL (ESQL) or Java, within configurable nodes.

Transformations can be made by various nodes in a message flow. Before a message flow node can operate on an incoming message, it must understand the structure of that message.

- Some messages contain a definition of their own structure and format. These messages are known as self-defining messages, which you can handle without the need for additional information about structure and format; see "Self-defining elements and messages" on page 1198.
- Other messages do not contain information about their structure and format. To process them, you must create a model of their structure; see "Message definition files" on page 1171.

The message definitions that you design are created within a message set which contains one or more message definitions. Message sets also categorize message definitions. The category facility, which you can extend using XSLT scripts, is used for generating Web Services Description Language (WSDL) and documentation; see "Message categories" on page 1200.

Like message flows, you create message models in the WebSphere Message Broker Toolkit. They can contain two types of information:

- The logical structure: the abstract arrangement and characteristics of the data, represented as a tree structure; see "The message model" on page 1160.

- One or more physical formats: the way the data is represented and delimited in the physical bit stream; see “Physical formats in the MRM domain” on page 1211.

Create the broker environment

The work of routing and transforming messages takes place in a broker. Within the broker, you can define one or more execution groups, which are processes in which message flows run.

The mode in which your broker is working can affect the number of execution groups and message flows that you can deploy, and the types of node that you can use. See “Restrictions that apply in each operation mode” on page 3657.

You can install and create one or more brokers on one or more computers that are running a supported operating system. If you create multiple brokers, you can configure your environment to provide protection against failure, and you can separate work across different divisions in a business.

You administer the broker by using product commands, the WebSphere Message Broker Explorer, the Brokers view within the WebSphere Message Broker Toolkit, or the Administration API for WebSphere Message Broker (also known as the CMP API) in your own applications.

Develop applications

After your system administrator has created your brokers, your application developers can create and modify message flows and message definitions by using the WebSphere Message Broker Toolkit.

Different perspectives in the WebSphere Message Broker Toolkit are used to develop message flows, message sets, and other related resources; see “WebSphere Message Broker Toolkit” on page 31.

You can use a repository to provide access control and version control of your development resources. A repository also allows multiple developers to work on the same resources in parallel; see “Development repository” on page 45.

Your applications can communicate with the broker by using a range of protocols that includes WebSphere MQ, JMS 1.1, HTTP and HTTPS, Web Services (SOAP and REST), File, Enterprise Information Systems (including SAP and Siebel), and TCP/IP. For more information about connecting applications, see “Nodes for connectivity” on page 1028.

Deploy applications to the runtime environment

When you have created and configured your message flows, message sets, and associated resources by using the Broker Application Development perspective of the WebSphere Message Broker Toolkit, you can deploy the executable data to one or more brokers; see “Packaging and deployment overview” on page 3210.

You can deploy data in the following ways:

- From the Brokers view of the WebSphere Message Broker Toolkit
- From the stand-alone administrative interface, the WebSphere Message Broker Explorer
- From the Test Client environment in the WebSphere Message Broker Toolkit

- By using a command
- By creating applications that use the Administration API application programming interface

When you deploy message flows and message sets, they are compiled and enveloped in a broker archive (BAR) file, and sent to the target broker; see "Packaging and deployment overview" on page 3210. The BAR file has configurable system properties. You can override properties such as queue and database names, without the need to change source files or redevelop the message flow. This configuration makes it easier to move definitions between systems.

The broker opens the BAR file, removes the contents, makes a record of the information that it has received, and discards the envelope. It retains the information in its local storage area within the computer file system, so that it can restore the application resources and restart messages flows if and when required.

Publish/Subscribe

Publish/subscribe is a style of messaging for which WebSphere Message Broker provides limited support; in WebSphere Message Broker Version 7.0 this support was transferred to WebSphere MQ. If you have been connecting publish/subscribe applications to brokers in previous versions, see "Migrating publish/subscribe information to WebSphere MQ" on page 141.

Further information

For a basic introduction to WebSphere Message Broker, see the IBM Redbooks publication WebSphere Message Broker Basics.

Related concepts:

"WebSphere Message Broker Toolkit" on page 31
The WebSphere Message Broker Toolkit is an integrated development environment and graphical user interface based on the Eclipse platform.

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message sets overview" on page 1162
A message set is a container for grouping messages and associated message resources (elements, types, groups).

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

"Execution groups" on page 53
An execution group is a named grouping of message flows that have been assigned to a broker. The broker enforces a degree of isolation between message flows in distinct execution groups by ensuring that they run in separate address spaces, or as unique processes.

"Publish/Subscribe" on page 2215
Publish/subscribe is a style of messaging application in which the providers of information (publishers) are decoupled from the consumers of that information (subscribers).

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Generate WSDL" on page 1274
A *Web Services Description Language* (WSDL) document specifies the interface to a web service, and enables a web service client to start it. A WSDL document that is generated from a message model defines web service requests and responses in terms of the messages that you have defined in that message model.

Related reference:

"General industry standards supported by WebSphere Message Broker" on page 3607
WebSphere Message Broker supports general industry standards that are associated with message processing.

WebSphere Message Broker Toolkit

The WebSphere Message Broker Toolkit is an integrated development environment and graphical user interface based on the Eclipse platform.

Application developers work in separate instances of the WebSphere Message Broker Toolkit to develop resources associated with message flows. The WebSphere Message Broker Toolkit connects to one or more brokers to which the message flows are deployed.

You can install the WebSphere Message Broker Toolkit only on Windows and Linux on x86. You can only view and interact with brokers that you have created in WebSphere Message Broker Version 7.0.

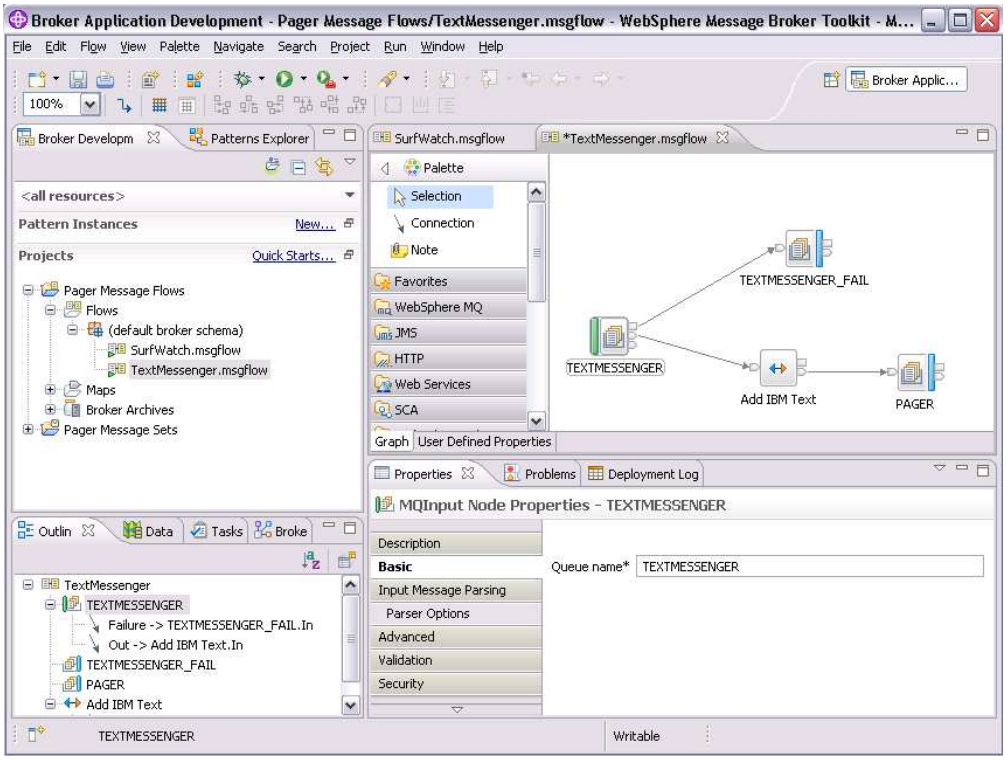
The WebSphere Message Broker Toolkit

When you start the WebSphere Message Broker Toolkit, a single window is displayed. This window is the WebSphere Message Broker Toolkit, which contains one or more perspectives.

A perspective is a collection of views and editors that you use to complete a specific task, or work with specific types of resource. The two significant perspectives in the WebSphere Message Broker Toolkit are the Broker Application Development perspective for application development, and the Debug perspective for debugging message flows. The first time that you start the WebSphere Message Broker Toolkit, the Broker Application Development perspective is displayed.

An additional stand-alone component, the WebSphere Message Broker Explorer, is supplied for advanced administrative users, and enables additional administration tasks that you cannot perform in the WebSphere Message Broker Toolkit.

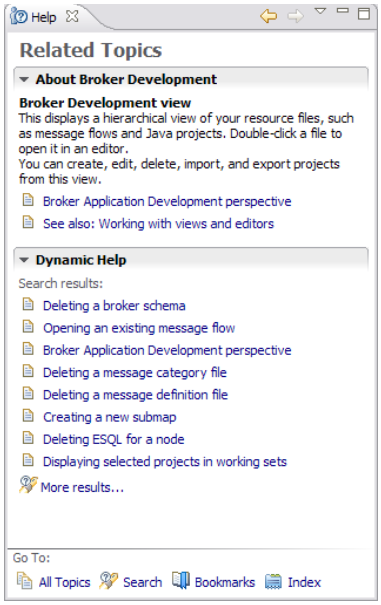
The following figure shows the Broker Application Development perspective with a message flow open in the Message Flow editor on Windows.



Accessing context-sensitive help

The Help view provides context-sensitive help throughout the WebSphere Message Broker Toolkit. You can display the Help view for most aspects of the user interface (for example, on the Broker Development view, the Message Flow editor, or a properties page) by bringing focus to the object and pressing F1 (on Windows) or SHIFT+F1 (on Linux). The Related Topics page shows description and help topics that are related to the selected object. The About section shows context help that is specific to your current context, and the Dynamic Help section shows some search results that might be related.

The following figure shows the Related Topics page of the Help view that is displayed when you press F1 in the Broker Development view.



Use the other pages in the Help view to view and search the contents of the information center. The All Topics page shows the table of contents of all the books in the information center. The Index provides an index of keywords of all the books in the information center. You can enter a keyword in the text field on the Index page to highlight the best match in the list of keywords. You can use the Search page to locate topics, samples, and remote documents using keywords in a search query. You can bookmark topics and other documents of interest, and view them in the Bookmarks page.

For a basic introduction to using the WebSphere Message Broker Toolkit, see the IBM Redbooks publication WebSphere Message Broker Basics.

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

"WebSphere Message Broker Toolkit perspectives" on page 34
A perspective is a group of views and editors that shows various aspects of the resources in the WebSphere Message Broker Toolkit.

"Editors" on page 35
An editor is a component of the WebSphere Message Broker Toolkit. Editors are typically used to edit or browse resources, which are the files, folders, and projects that exist in the workbench.

"Resources" on page 36
The projects, folders, and files that you work with in the WebSphere Message Broker Toolkit workspace are called *resources*. By default, these resources are stored with their metadata in the workspace directory in your local file system. The workspace directory is created the first time that you start the WebSphere Message Broker Toolkit.

"Development repository" on page 45
Use a development repository to benefit from features such as version control and access control of files, which make it easier for teams to work on shared resources.

Related information:

Workbench User Guide - Perspectives

Workbench User Guide - Views

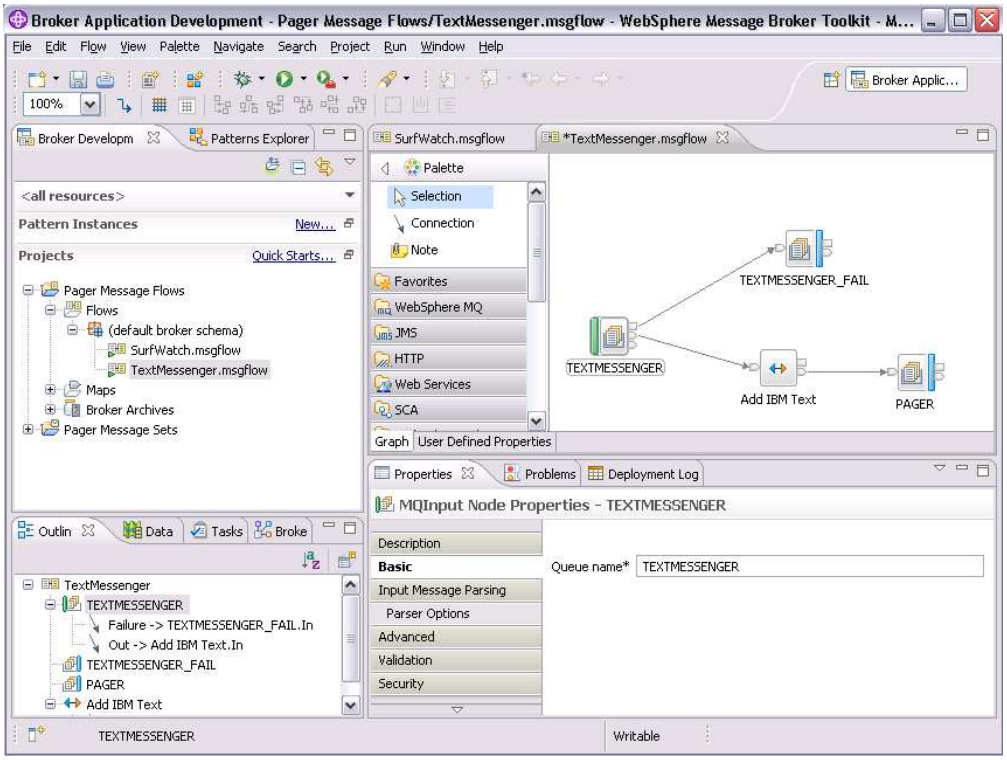
Workbench User Guide - Resources

WebSphere Message Broker Toolkit perspectives

A perspective is a group of views and editors that shows various aspects of the resources in the WebSphere Message Broker Toolkit.

You can switch perspectives, depending on the task at hand, and customize the layout of views and editors. Switch between perspectives by clicking **Window** > **Open Perspective** > **Other**, then clicking the perspective to which you want to switch.

The following figure shows the Broker Application Development perspective in the WebSphere Message Broker Toolkit.



The WebSphere Message Broker Toolkit offers the following perspectives:

Broker Application Development perspective

The Broker Application Development perspective is the default perspective that is displayed the first time that you start the WebSphere Message Broker Toolkit.

Application developers work in this perspective to develop and modify message sets, message flows, and other associated resources. You can also import relational database schemas for ESQL content assist and validation, and interact with databases by using the Data Project Explorer view and Data Source Explorer view.

The preceding figure shows the Broker Application Development perspective with a message flow open in the Message Flow editor.

You can use the Brokers view in the Broker Application Development perspective to create new brokers and deploy resources to connected brokers. Some of the administrative tasks that are available through the WebSphere Message Broker Explorer, which is supplied as a separate component that you can install on computers on which you intend to perform only administrative tasks, are also supported by the Brokers view.

Debug perspective

The Debug perspective is where application developers test and debug message flows.

Plug-in Development perspective

The Plug-in Development perspective is where application developers develop plug-ins for user-defined extensions.

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

"Editors"
An editor is a component of the WebSphere Message Broker Toolkit. Editors are typically used to edit or browse resources, which are the files, folders, and projects that exist in the workbench.

Related reference:

"Editors in the WebSphere Message Broker Toolkit" on page 6793

"Broker Application Development perspective" on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

"Perspectives in the WebSphere Message Broker Toolkit" on page 6783

Related information:

Workbench User Guide - Working with Perspectives

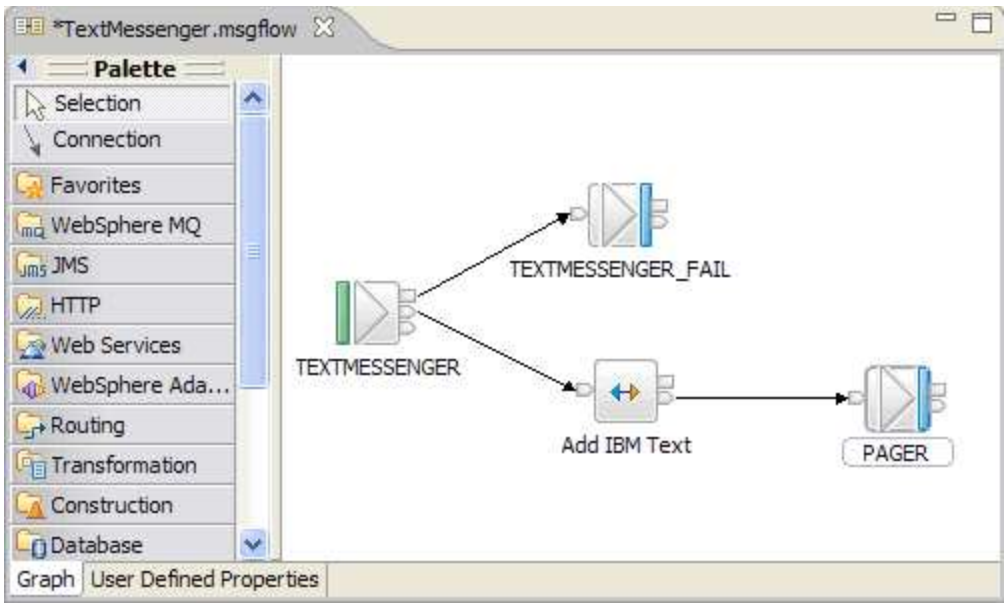
Editors

An editor is a component of the WebSphere Message Broker Toolkit. Editors are typically used to edit or browse resources, which are the files, folders, and projects that exist in the workbench.

When you open a file for editing, for example by double-clicking it in the Broker Development view, the default editor associated with that file opens in the editor area of the current perspective. By default, the editor area is in the upper-right corner of the WebSphere Message Broker Toolkit window.

Open resources with the default editor because other editors might not validate the changes correctly.

The following diagram shows the `TextMessenger.msgflow` file from the Pager sample opened in the Message Flow editor, which is part of the Broker Application Development perspective.



You can open any number of editors at the same time, but only one editor is active at a time. The main menu bar and main toolbar display the operations that apply to the active editor. By default, editors are stacked in the editor area, but you can tile them to view source files simultaneously. Tabs in the editor area indicate the names of the resources that are open for editing. An asterisk (*) indicates that an editor has unsaved changes. If you attempt to close the editor or exit the WebSphere Message Broker Toolkit with unsaved changes, you are prompted to save the changes.

To find out about the Pager sample, click the following link:

- Pager

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Related concepts:

"WebSphere Message Broker Toolkit" on page 31
The WebSphere Message Broker Toolkit is an integrated development environment and graphical user interface based on the Eclipse platform.

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related reference:

"Editors in the WebSphere Message Broker Toolkit" on page 6793
Workbench User Guide - Tiling editors

Resources

The projects, folders, and files that you work with in the WebSphere Message Broker Toolkit workspace are called *resources*. By default, these resources are stored

with their metadata in the workspace directory in your local file system. The workspace directory is created the first time that you start the WebSphere Message Broker Toolkit.

The default locations for the workspace are in the following places:

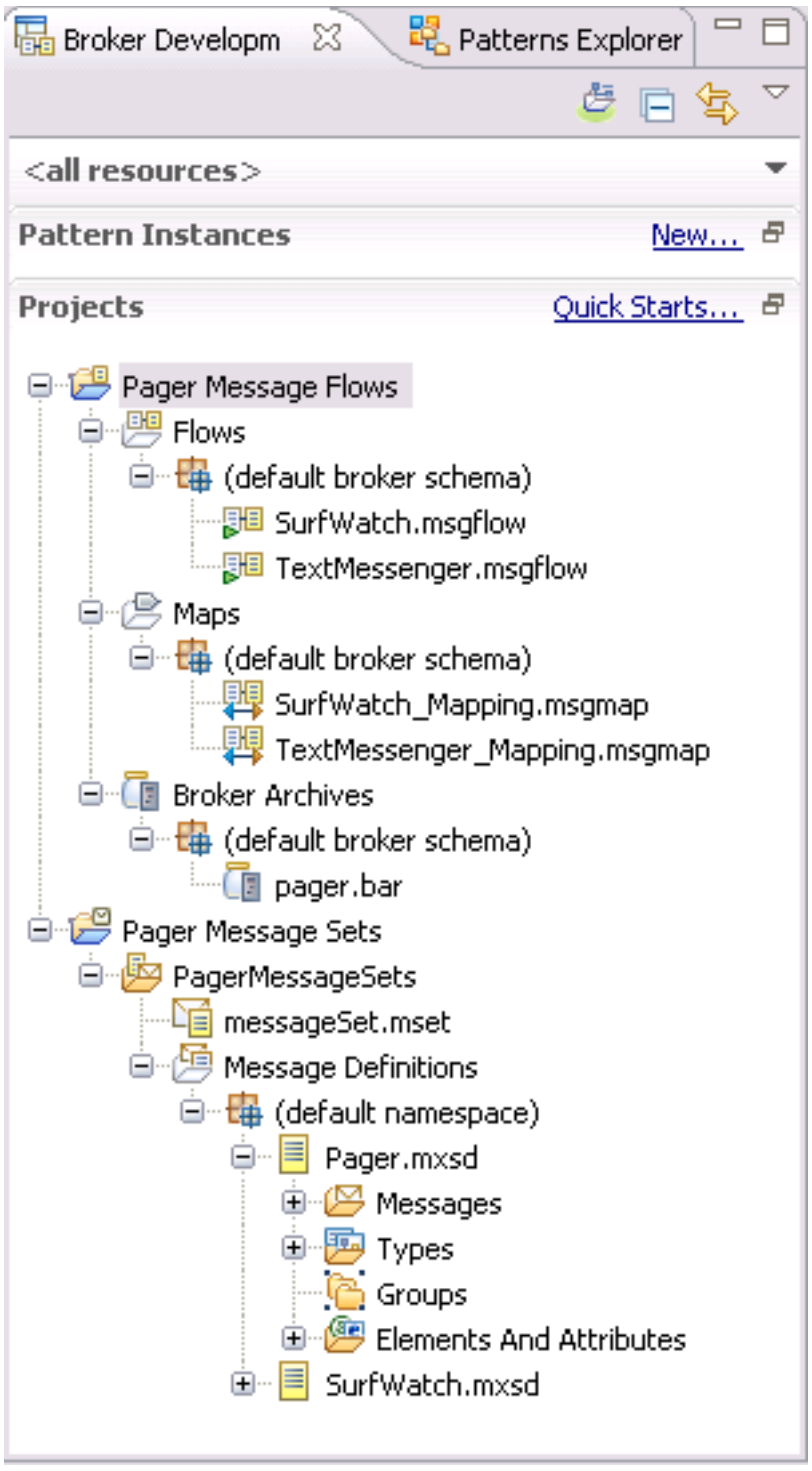
- On Linux on x86, the default workspace directory is created at `/home/user_ID/IBM/wmbt70/workspace`.
- On Windows XP and Windows Server 2003, the default workspace directory is created at `C:\Documents and Settings\user_ID\IBM\wmbt70\workspace`.
- On Windows Vista and Windows Server 2008, the default workspace directory is created at `C:\Users\user_ID\IBM\wmbt70\workspace`.

  where *user_ID* is the user name with which you are logged on.

You can create projects in other directories in addition to the workspace directory. You can maintain multiple workspaces by specifying a new location at the prompt when you start your WebSphere Message Broker Toolkit session.

Typically, you edit and view WebSphere Message Broker Toolkit resources in the Broker Development view in the Broker Application Development perspective.

The following figure shows a message flow project and a message set project in the Broker Development view of the Broker Application Development perspective.



Resource editors do not automatically reflect the changes that you make in one window in additional windows that you have opened to view the same resource. Close and reopen additional windows each time that you update a resource in an editor session.

Types of resource: You can create and work with three basic types of resource:

Source files
Source files are included in your workspace and are accessible from the

Broker Application Development perspective. When you create source files, they are grouped by file type in *folders*, in the same way as the directories of a file system. You can also group folders or files in *projects*. Projects are used for building, version control, sharing, and resource organization.

The following source files can exist:

- `.msgflow` files contain the message nodes and connections that you create to form a message flow. These files are displayed in the Broker Application Development perspective in a folder called Flows. The Flows folder is always created and displayed.
- `.esql` files are optional and contain ESQL code that performs specific processing steps within a message node. You can code ESQL to tailor message processing in Compute, Database, and Filter nodes. If you have created ESQL files, they are displayed in the Broker Application Development perspective in a folder called ESQLs.
- `.msgmap` files are optional and map the relationships between different data sources (typically database tables and messages) that you have created by using the graphical mapping tools. You can create maps to tailor message processing in DataDelete, DataInsert, DataUpdate, and Warehouse nodes. If you have created mapping files, they are displayed in the Broker Application Development perspective in a folder called Maps.
- `.category` files are optional and contain a group of messages, either related to WSDL operations, or to a user-specific purpose or message flow. If you have created category files, they are displayed in the Broker Application Development perspective in a folder called Message Categories.
- `.mxsd` files are optional and contain definitions of the messages you have modeled. If you have created message models, they are displayed in the Broker Application Development perspective in a folder called Message Definitions.
- `.wsdl` files are optional and contain WSDL definitions which you have imported into the workspace to use as a source for message definitions. If you have created WSDL files, they are displayed in the Broker Application Development perspective in a folder called Deployable WSDL, and are grouped by namespace.
- `.insca` and `.outsca` files are optional Broker SCA definition files. A `.insca` file contains an SCA import component and is used to configure SCAInput and SCAReply nodes. A `.outsca` file contains an SCA export component and is used to configure SCAAsyncRequest, SCAAsyncResponse, and SCARequest nodes. They are displayed in the Broker Application Development perspective in a folder called Broker SCA Definitions, in the message set project.

You can create resources from a pattern more than once to give unique pattern instances with different configurations, see “Patterns” on page 1310. The resources for each pattern instance are contained within a single pattern instance project. The pattern instance project contains links to all projects containing the resources that are created as a result of generating a pattern instance from your configuration, such as message flows, Java classes for JavaCompute nodes, ESQL modules, message maps, test client, XML files, and style sheet files.

Helper files

Helper files maintain information that supports other activities:

- `.broker` files contain definitions of broker connections.
- `.bar` files contain deployable and other files that you have chosen to send to a broker.
- `.mbtest` files contain the steps that define a test that you use with the Test Client to debug your applications.

Deployable files

Deployable files are included in a broker archive (BAR) file and deployed to the broker where they are involved in some way in the processing of messages. A BAR file might contain the following files:

- A `.cmf` file for each message flow. This file is a compiled version of the message flow. You can have any number of these files within your BAR file.
- A `.dictionary` file for each message set dictionary. You can have any number of these files within your BAR file.
- One or more XSD compressed files (`.xsdzip`), if XML schema and WSDL are defined within a message set.
- A `broker.xml` file. This file is called the *broker deployment descriptor*. You can have only one of these files within your BAR file. This file, in XML format, is contained in the `META-INF` folder of the compressed file and can be modified by using a text editor or shell script.
- One or more XML files (`.xml`), style sheets (`.xsl`), and XSLT files (`.xlst`), if required by nodes in the message flows you have added to this BAR file. The XSLTransform node is one that might require these files.
- One or more JAR files, if required by JavaCompute nodes in the message flows you have added to this BAR file.
- One or more inbound or outbound adapter files (`.inadapter` or `.outadapter`), if required by WebSphere Adapter nodes (for example, the SiebelInput node) in the message flows you have added to this BAR file.
- One or more PHP script files (`.php`), if required by PHPCompute nodes in the message flows you have added to this BAR file.
- Other files that you might want to associate with this BAR file. For example, you might want to include Java source files, `.msgflow` files, or `.wsdl` files for future reference. BAR files can contain all files types.

References between files:

Source files can refer to content in other files. For example, a message flow can require an ESQL file. The files on which the message flow depends must be present when that message flow is compiled; see "By name linking" on page 43.

If you are considering changing a resource, you can see a list of other resources that refer to it. To search for these resources, you must first enable indexing by following the instructions in "Enabling and disabling indexing" on page 1454. Then you can right-click a resource in the Broker Development view and click **Show all references**. The search results view shows, in a tree structure, all references of your resource, including its location, such as a node property or the line number in an ESQL file. You can double-click a resource to open it in an appropriate editor, or you can right-click a resource and click **Open with** to choose the editor in which to open it. Note that dependent files can be found in another project. See "Project references" on page 44.

To see which files are dependent on which other files, see "Showing resource references" on page 1447.

If you are considering renaming or moving a resource and you know the new name or location, you can run an impact analysis. For more information, see “Impact analysis: analyzing the effects of planned changes to your applications” on page 1150.

:

Related concepts:

“Impact analysis: analyzing the effects of planned changes to your applications” on page 1150
When developing an application, you might want to change the names of some artifacts, or move them. Impact analysis reports the artifacts that are likely to be affected by a particular change.

“Working sets” on page 42
A *working set* is a logical collection of projects, which you can use to limit the number of resources that are displayed in the Broker Development view. By creating and using a working set, you can reduce the visual complexity of what is displayed in the Broker Development view, making it easier to manage and work with your projects.

“Using patterns” on page 1312
Use patterns that are supplied with the WebSphere Message Broker Toolkit to create resources that are used to solve a specific business problem.

“WebSphere Message Broker Toolkit” on page 31
The WebSphere Message Broker Toolkit is an integrated development environment and graphical user interface based on the Eclipse platform.

“Project references” on page 44
When a projects refers to other projects in the workspace, this is called a *project reference*. When one project references another, the files in the referenced project are available for use by the referring project. For example to configure certain nodes, to create libraries of reusable message flows, and to enable Content Assist in the ESQL editor.

“By name linking” on page 43
You identify objects using a combination of a *namespace* and a name, referred to as a *fully qualified name*. The use of fully qualified names, called *by name linking*, makes it easy to identify and locate objects, and to correct broken references.

“WebSphere Message Broker Explorer” on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

“The broker environment” on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related tasks:

“Showing resource references” on page 1447
If you are considering changing a resource, you can see a list of other resources that would be affected by that change.

“Working with patterns in the Broker Development view” on page 1314
Using the Broker Development view to create patterns.

“Choosing a pattern” on page 1313
Select a pattern in the Patterns Explorer view to create resources to solve a specific business problem.

“Creating a working set” on page 575
Create a working set to limit the number of resources that are displayed in the Broker Development view.

Related reference:

"Broker Application Development perspective" on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

"Resource types in the WebSphere Message Broker Toolkit" on page 6821
Links to information on projects and resource files.

"Rules for naming workspace objects" on page 6827

Related information:

Workbench User Guide - Resources

Workbench User Guide - Working with projects, folders, and resources

Working sets:

A *working set* is a logical collection of projects, which you can use to limit the number of resources that are displayed in the Broker Development view. By creating and using a working set, you can reduce the visual complexity of what is displayed in the Broker Development view, making it easier to manage and work with your projects.

The *active working set* is the current working set of projects that you choose to display. If you do not create any working sets, the default active working set contains all your resources and are contained in the <all resources> section of the Broker Development view, which displays all of your projects.

The Broker Development view has three sections: the working set selection list, Pattern Instances (initially collapsed), and Projects (initially expanded). Pattern instance projects are shown only in the Pattern Instances section, and all other project types are shown in the Projects section. If no projects exist, the Projects section contains a list of Quick Start Wizard links.

You can create a new working set in the Broker Development view in one of the following three ways:

- You can create a new working set and add existing projects to it.
- You can use a Quick Start wizard to set up the basic resources that are required to develop a message flow. The wizard provides the option of creating a new working set for the resources.
- You can create a pattern instance project, and a working set of the same name is created automatically.

When you use the Broker Application Development perspective to create a file or new project, the new file or project is automatically added to the current active working set. If you have not identified an active working set, the new file or project is displayed in the <all resources> section of the Broker Development view, but is not added to a working set.

Related concepts:

"Resources" on page 36
The projects, folders, and files that you work with in the WebSphere Message Broker Toolkit workspace are called *resources*. By default, these resources are stored with their metadata in the workspace directory in your local file system. The workspace directory is created the first time that you start the WebSphere Message Broker Toolkit.

"Using patterns" on page 1312
Use patterns that are supplied with the WebSphere Message Broker Toolkit to

create resources that are used to solve a specific business problem.

"Project references" on page 44
When a projects refers to other projects in the workspace, this is called a *project reference*. When one project references another, the files in the referenced project are available for use by the referring project. For example to configure certain nodes, to create libraries of reusable message flows, and to enable Content Assist in the ESQL editor.

"By name linking"
You identify objects using a combination of a *namespace* and a name, referred to as a *fully qualified name*. The use of fully qualified names, called *by name linking*, makes it easy to identify and locate objects, and to correct broken references.

Related tasks:

"Working with patterns in the Broker Development view" on page 1314
Using the Broker Development view to create patterns.

"Creating a working set" on page 575
Create a working set to limit the number of resources that are displayed in the Broker Development view.

Related reference:

"Broker Application Development perspective" on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

"Resource types in the WebSphere Message Broker Toolkit" on page 6821
Links to information on projects and resource files.

"Rules for naming workspace objects" on page 6827

Related information:

Workbench User Guide - Resources

Workbench User Guide - Working with projects, folders, and resources

By name linking:

You identify objects using a combination of a *namespace* and a name, referred to as a *fully qualified name*. The use of fully qualified names, called *by name linking*, makes it easy to identify and locate objects, and to correct broken references.

For example, if you rename an object, all references to that object are broken. If you substitute another object with the same name, all the broken references are corrected.

This concept is important when you are working in a team environment. With by name linking, you can share files in a repository and concurrently modify, add, and delete objects in your message flow application. When you integrate the various parts of the message flow application, you can detect and resolve broken references to objects that have been moved, renamed, or deleted.

Related concepts:

"Resources" on page 36
The projects, folders, and files that you work with in the WebSphere Message Broker Toolkit workspace are called *resources*. By default, these resources are stored with their metadata in the workspace directory in your local file system. The workspace directory is created the first time that you start the WebSphere Message Broker Toolkit.

"Project references"
When a projects refers to other projects in the workspace, this is called a *project reference*. When one project references another, the files in the referenced project are available for use by the referring project. For example to configure certain nodes, to create libraries of reusable message flows, and to enable Content Assist in the ESQL editor.

"Namespaces in the message model" on page 1201
Use namespaces to qualify message model object names.

Related reference:

"Rules for naming workspace objects" on page 6827

Project references:

When a projects refers to other projects in the workspace, this is called a *project reference*. When one project references another, the files in the referenced project are available for use by the referring project. For example to configure certain nodes, to create libraries of reusable message flows, and to enable Content Assist in the ESQL editor.

The following scenarios in the WebSphere Message Broker Toolkit give examples of project references:

- When configuring a Mapping node in a message flow, you must select source and target messages for the map. The messages are contained within one or more message sets. In order to select messages, the message flow project must have a reference to the message sets. You must also have a reference to a data design project if you want to map to or from a database.
- When configuring a node, such as an MQInput node, you can define the message template for the message type that the node processes. If you have chosen MRM, SOAP, XMLNSC, DataObject, or IDOC as the `Message domain` property of the node, you must also specify the name of the message sets that contains the message model. To pre-populate the list of message sets in the `Message set` property, the message flow project must have a reference to the message sets.
- You might want to create a library of reusable ESQL subroutines in a message flow project, or create a library of message flows to reuse in other flows. A message flow that you want to use these subroutines or message flows in, must have a reference to the message flow project from its parent message flow project.
- You can also use project references to enable Content Assist in the ESQL editor. ("Content Assist" is context-sensitive help that displays valid ways in which a code statement can be completed.) If you set up a project reference from a message flow project containing ESQL code to a message sets, the ESQL editor is able to display a list of valid message references.
- If you use the Patterns Explorer to generate a pattern instance from a pattern specification, the generated pattern instance project contains references to all the other projects generated from the pattern. See "Using patterns" on page 1312.

To create or remove a project reference manually, right-click the project name in the Broker Development view and select the **Properties** menu item. Select **Project References** from the Properties window, and a list of all regular projects in the workspace is displayed from which you can select or clear project references. Message flow, message sets and pattern instance projects have an additional menu item **Add or Remove Project References** that launches the Add or Remove Project References window where you can select or clear project references.

If you later close or delete a referenced project, or delete an object within it, it is no longer available to the referencing project and an error is generated. You can correct the error by opening the closed project or adding the missing object, with the correct name, and saving.

Related concepts:

"Namespaces in the message model" on page 1201
Use namespaces to qualify message model object names.

"By name linking" on page 43
You identify objects using a combination of a *namespace* and a name, referred to as a *fully qualified name*. The use of fully qualified names, called *by name linking*, makes it easy to identify and locate objects, and to correct broken references.

Related reference:

"Resource types in the WebSphere Message Broker Toolkit" on page 6821
Links to information on projects and resource files.

"Rules for naming workspace objects" on page 6827

Development repository

Use a development repository to benefit from features such as version control and access control of files, which make it easier for teams to work on shared resources.

The WebSphere Message Broker Toolkit is based on the Eclipse platform, therefore you can access Eclipse-supported repositories directly from the WebSphere Message Broker Toolkit. You can use all repositories that are supported by Eclipse with WebSphere Message Broker.

You can take the version numbers of resources from the repository, and associate that version number with the message flows and message sets when they are deployed. This association allows you to display which version of the flow is deployed in the WebSphere Message Broker Toolkit. An alternative method of assigning a version number to a message flow is to specify one in the WebSphere Message Broker Toolkit when the message flow is created; see "Message flow version and keywords" on page 1445.

- For information about how to set up the WebSphere Message Broker Toolkit to run with CVS, see "Configuring CVS to run with the WebSphere Message Broker Toolkit" on page 573.
- For information about how to integrate the Rational Team Concert client, see "Integrating the Rational Team Concert client with the WebSphere Message Broker Toolkit" on page 576.
- For information on how to enable Rational ClearCase®, see "Configuring the WebSphere Message Broker Toolkit to run Rational ClearCase" on page 574.
- For information about other repositories, read about other team repositories that are supported by Eclipse.

Related concepts:

"Resources" on page 36
The projects, folders, and files that you work with in the WebSphere Message Broker Toolkit workspace are called *resources*. By default, these resources are stored with their metadata in the workspace directory in your local file system. The workspace directory is created the first time that you start the WebSphere Message Broker Toolkit.

"Message flow version and keywords" on page 1445
When you are developing a message flow, you can define the version of the message flow as well as other key information that you want to be associated with it.

Related tasks:

"Configuring CVS to run with the WebSphere Message Broker Toolkit" on page 573
Install CVS as a normal program by following the usual prompts. Not all versions of *CVSNT* are supported by Eclipse.

"Configuring the WebSphere Message Broker Toolkit to run Rational ClearCase" on page 574
To use Rational ClearCase with the WebSphere Message Broker Toolkit, enable the capability in the Preferences page.

Related reference:

"Resource types in the WebSphere Message Broker Toolkit" on page 6821
Links to information on projects and resource files.

The broker environment

A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Create more than one broker, on one or more computers, to support your applications; creating more than one broker can provide load balancing, or a division of responsibilities. For example, you might have one broker that handles all your financial applications, and another that handles your order processing and fulfillment.

Application programs connect to and send messages to the broker, and receive messages from the broker. Code your applications to use one of the supported protocols for interacting with the broker; for example, WebSphere MQ queues and connections, Web services, or WebSphere Adapters. The broker routes each message by using the rules that you have defined in message flows and message sets, and transforms the data into the structure required by the receiving application.

You can install the broker component on one or more of the supported platforms, which are listed in "Operating system requirements" on page 3590. You can create a broker only on the computer on which you have installed the broker component. You can use the WebSphere Message Broker Explorer, the WebSphere Message Broker Toolkit, or the command line to create local brokers.

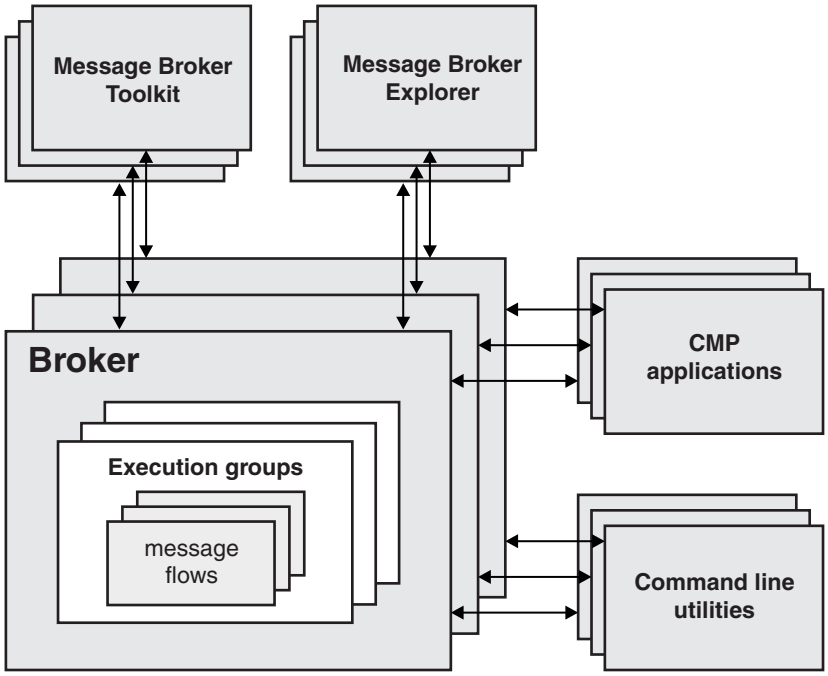
When you create a broker, it operates in one of a number of modes: `enterprise`, `adapter`, `starter`, or `entry`. You must run the broker in the mode that matches the license that you have purchased; see "Operation modes" on page 48.

Administer the broker by using the WebSphere Message Broker Explorer, the Broker view in the WebSphere Message Broker Toolkit, or the product commands. Alternatively, you can write your own programs to use the Administration API for WebSphere Message Broker (also known as the CMP API).

Manage the application resources of the broker, which include message flows and message sets, by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer; these two applications connect to the broker by using a WebSphere MQ server connection, which is defined to the broker queue manager when you create the broker.

The following figure shows the relationship between the resources that exist at run time, and how they interact with the WebSphere Message Broker Explorer and

WebSphere Message Broker Toolkit.



Resources associated with a broker

When you create a broker, the following resources are also defined and created:

- A WebSphere MQ queue manager, if one does not exist.
- A set of fixed-name queues that are defined to the WebSphere MQ queue manager.
- A default WebSphere MQ SVRCONN channel with a fixed name SYSTEM.BKR.CONFIG, which is used by the WebSphere Message Broker Toolkit, the WebSphere Message Broker Explorer, and applications that use the Administration API (CMP API).

Related concepts:

"Operation modes" on page 48
The operation mode that you use for your broker is determined by the license that you purchase.

"Execution groups" on page 53
An execution group is a named grouping of message flows that have been assigned to a broker. The broker enforces a degree of isolation between message flows in distinct execution groups by ensuring that they run in separate address spaces, or as unique processes.

"The Administration API for WebSphere Message Broker" on page 54
The Administration API for WebSphere Message Broker is a programming interface that your applications can use to control brokers and their resources through a remote interface.

"WebSphere Message Broker Toolkit" on page 31
The WebSphere Message Broker Toolkit is an integrated development environment and graphical user interface based on the Eclipse platform.

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the

Eclipse platform for administering your brokers.

Related tasks:

Chapter 7, “Configuring brokers for test and production environments,” on page 579
Create one or more brokers on one or more computers, and configure them on your test and production systems to process messages that contain your business data.

Chapter 8, “Administering brokers and broker resources,” on page 899
Administering brokers and associated broker resources includes the tasks that you perform frequently to activate and manage those resources. Choose the method you prefer to administer your brokers and associated resources.

Related reference:

“System management interfaces” on page 52
The brokers provide a service for independent system management agents.

Operation modes

The operation mode that you use for your broker is determined by the license that you purchase.

The following modes are supported:

- “Trial Edition mode” on page 49. All features are enabled, but you can use the product for only 90 days after installation.
- “Entry Edition mode” on page 49. Entry node features are enabled and the types of node that you can use, and the number of execution groups that you can create, are limited.
- “Starter Edition mode” on page 49. All features are enabled for use with a single execution group. The number of message flows that you can deploy are unlimited.
- “Enterprise mode” on page 49. All features are enabled and no restrictions or limits are imposed. This mode is the default mode, unless you have the Trial Edition.
- “Remote Adapter Deployment mode” on page 50. Only adapter-related features are enabled, and the types of node that you can use, and the number of execution groups that you can create, are limited.

You must ensure that your brokers are running in the operation mode for which you have purchased a license. You can set the operation mode when you create a broker by using the **mqsicreatebroker** command.

If you have purchased a license for the full package, the Starter Edition, or the Remote Adapter Deployment mode, the broker is automatically created in `enterprise` mode unless you specify the correct mode for your license.

Change the mode of your broker to conform to your license if necessary; see “Changing the operation mode of your broker” on page 655. You can also report the current mode of your broker; see “Checking the operation mode of your broker” on page 657.

The WebSphere Message Broker Toolkit remains the same in all modes. All the capabilities of the WebSphere Message Broker Toolkit are available in all modes. If you try to deploy too many message flows or execution groups for the mode, or try to use a node that is not valid in the mode, the operation is rejected, and an error message is displayed indicating the reason for the failure; see “Restrictions

that apply in each operation mode" on page 3657. Node restrictions for a given mode also apply to the use of message flows generated by WebSphere Message Broker patterns.

Fix packs are identical for all modes, and can be applied without affecting the validity of the mode. The expiry of a Trial Edition is not affected by applying service; in Trial Edition, by default new brokers continue to be in Trial Edition mode.

Trial Edition mode

In `trial` mode, the broker operates with all features enabled. You can use all available function, and are not limited in the number of resources that you create and maintain. All capability is available for 90 days after installation.

You cannot revert to Trial Edition from any other mode.

You can download Trial Edition at no charge, from the following website: WebSphere Message Broker Trial package.

Entry Edition mode

In `entry` mode, the broker operates with a limited set of nodes available for deployed flows. You are also limited to one execution group. For a list of nodes that can be deployed to a broker running in Entry Edition mode, see "Restrictions that apply in each operation mode" on page 3657. If you attempt to exceed the limits of this mode, the deployment is rejected. Use this edition for simple use cases only.

Because the functions that are enabled and the number of execution groups that you can create are limited, not all samples work in Entry Edition mode. To run samples, see "Development and unit test" on page 50.

Starter Edition mode

In `starter` mode, the broker operates with all features enabled. Use this edition if you expect to use all or most of the features that are available, but intend to configure a limited environment because of low capacity requirements.

You can use all the available functions, but are limited in the number of resources that you can create and maintain. You are limited to creating one execution group; for more information, see "Restrictions that apply in each operation mode" on page 3657. If you attempt to exceed the limits of this mode, the deployment is rejected.

You cannot use all the samples when your broker is in `starter` mode, because of the preceding restrictions. If you want to run samples, see "Development and unit test" on page 50.

Enterprise mode

In `enterprise` mode, the broker operates with all features enabled, and no operational limits on the creation of execution groups or on the number of flows deployed to an individual execution group are enforced. If you want to set up a full broker environment that uses most or all the features available, your brokers must operate in this mode, and you therefore require the full license. If you do not

specify another mode, your brokers have the mode set to the default value `enterprise`.

Remote Adapter Deployment mode

In `adapter` mode, the broker operates with a limited set of nodes available for deployed flows. Use this edition if you expect your typical use of the broker to be integration with Enterprise Information Systems (EIS). This edition supports the subset of development resources that provide EIS interaction. For a list of nodes that can be deployed to a broker running in Remote Adapter Deployment mode, see "Restrictions that apply in each operation mode" on page 3657.

You can create up to two execution groups, with no limit on the number of deployed message flows in each of these execution groups; see "Restrictions that apply in each operation mode" on page 3657. If you attempt to exceed the limits of this mode, the deployment is rejected.

You cannot use all the patterns and samples when your broker is in `adapter` mode, because of the preceding restrictions. If you want to run samples, see "Development and unit test."

Development and unit test

Your license also covers use of the product for development and unit test purposes, but check the license to ensure that you conform to any restrictions for development and unit test. All developers in your organization who are working on resources and applications for the WebSphere Message Broker environment can install all components on their computer. They can create and configure a broker environment without any functional or resource restrictions, but they can use only a single broker per WebSphere Message Broker Toolkit. Installation of the WebSphere Message Broker Toolkit limits this use to Windows and Linux on x86 computers. Developers can create and use development and unit test brokers in `enterprise` mode.

You can also install the supplied WebSphere MQ and DB2 products on the computers on which your developers perform their development and unit test, regardless of the license agreement that you have purchased.

If you want to run samples to explore and understand the features of the product, install them on your development and unit test computers.

Integration with Tivoli License Manager

If you use IBM Tivoli License Manager to control and manage your licensed software products, you must ensure that you choose the correct license for the WebSphere Message Broker edition that you have purchased. For more information, see "Installing Tivoli License Manager" on page 301.

Related tasks:

"Checking the operation mode of your broker" on page 657
Use the **mqsimode** command to find out the operation mode of your broker.

"Changing the operation mode of your broker" on page 655
Change the operation mode in which your broker is working by using the **mqsimode** command.

Related reference:

"Restrictions that apply in each operation mode" on page 3657
The operation mode in which your broker is working defines how many execution groups you can use, and which nodes are available.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsimode** command" on page 3899
Use the **mqsimode** command to configure and retrieve operation mode information.

Controlling the functional level of WebSphere Message Broker

You can control the functional level of your broker to enable new functionality added in WebSphere Message Broker fix packs.

About this task

The default functional level of the broker represents the level for WebSphere Message Broker Version 7.0.0.0. At this level, some functionality added by later fix packs is not supported by the broker.

Nodes and parsers that are added in later fix packs are available in the WebSphere Message Broker Toolkit, and you can include these nodes and parsers in a message flow. You can deploy the message flow to a broker only if you have set the functional level of that broker to the value that represents the fix pack in which the nodes and parsers were first delivered.

You can control the functional level of each broker, therefore you can try out new nodes on test brokers without affecting the operation of your production brokers. When you are satisfied that the nodes provide the functions that you require, and work as you expect, you can set the functional level of other brokers in your domain when appropriate.

To change the function level of a broker, use the **mqsichangebroker** command and set the **-f** parameter to the appropriate value. For more information about the use of the **mqsichangebroker** command, see "**mqsichangebroker** command" on page 3723.

You can check the functional level of a broker by using the **mqsireportbroker** command (see "**mqsireportbroker** command" on page 3919) or through the WebSphere Message Broker Explorer.

If you enable a certain functional level, you also enable all the functions added at lower levels. For example, if you enable new fix pack 3 functions, you automatically enable new functions for fix packs 1 and 2.

You can enable all new functionality by setting the function level to all.

The list of functions that are enabled for each function level is described in the following sections.

Version 7.0.0.1

You can install Version 7.0.0.1 as a full, generally available version, or as a fix pack. Version 7.0.0.1 has the functions enabled for V7.0.0.1 regardless of the installation route chosen.

Version 7.0.0.2

- JDEdwardsInput node (see "JDEdwardsInput node" on page 4519)
- JDEdwardsRequest node (see "JDEdwardsRequest node" on page 4524)
- EmailInput node (see "EmailInput node" on page 4394)
- FileRead node (see "FileRead node" on page 4444)
- JSON parser (see "JSON parser and domain" on page 1128)
- Execution group profiles (see "Execution group-specific command environment: Windows systems" on page 309 and "Execution group-specific command environment: Linux and UNIX systems" on page 312)

Related concepts:

"New function added in Version 7.0 fix packs" on page 15
Some fix packs and other maintenance packs deliver new functions.

Related tasks:

"Checking the broker operation mode and function level" on page 298
You must ensure that your production brokers conform to the terms of your license. You can also change the function level to enable the use of nodes that are supplied in the latest fix pack.

Related reference:

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

"**mqsireportbroker** command" on page 3919
Use the **mqsireportbroker** command to display broker registry entries.

System management interfaces

The brokers provide a service for independent system management agents.

This service enables a central management facility to access information about a network that includes one or more brokers. Therefore, you can extend your existing system management agents to include WebSphere Message Broker resources.

Brokers publish event messages, using fixed topics, in response to configuration changes, state changes, and user actions such as subscription registrations. Brokers also use architected messages to publish events related to their operational status, and changes in that status. These messages are published using the reserved topic root `$SYS` in code page 1208.

An example of a fixed topic is:

```
$SYS/Broker/<brokerName>/Status/ExecutionGroup/<executionGroupName>
```

The topic structure is fixed in this case, but obviously `<brokerName>` and `<executionGroupName>` are replaced with the appropriate values.

An example of the actual message data for this publication is:

```
<Broker uuid="12345678-1234-1234-1234-123456789012">
        <ExecutionGroup uuid="12345678-1234-1234-1234-123456789012">
                <Stop>
                        <AllMessageFlows/>
                </Stop>
        </ExecutionGroup>
</Broker>
```

A system management agent can subscribe to these topics, or to a subset of these topics, to receive the detailed information about activity and state changes in the WebSphere Message Broker components.

The event messages have a fixed structure, defined in *XML (Extensible Markup Language)*. The format of these messages, constructed in XML, is detailed in "The XML message body" on page 4262. The messages cover configuration changes, state changes, error notifications, and detailed subscription and topic information (for example, a subscription registration).

You can develop or purchase system management adapters or customized administrative applications. These applications subscribe to the system management topics generated by WebSphere Message Broker to receive information about the activity of its resources.

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related tasks:

"Installing Tivoli License Manager" on page 301
IBM Tivoli License Manager (ITLM) enables you to monitor the use of IBM (and other) software products. WebSphere Message Broker includes support for ITLM Version 2.1.

Related reference:

"WebSphere Message Broker event reports" on page 6883
Event messages are published by a broker in response to certain conditions that occur while the broker is active.

"WebSphere Message Broker event reports: general architecture" on page 6884
Brokers publish messages on reserved topics after significant events within the broker. By subscribing to these topics, a client can be informed when these events occur.

Execution groups

An execution group is a named grouping of message flows that have been assigned to a broker. The broker enforces a degree of isolation between message flows in distinct execution groups by ensuring that they run in separate address spaces, or as unique processes.

Each execution group is started as a separate operating system process, providing an isolated runtime environment for a set of deployed message flows. Within an execution group, the assigned message flows run in different thread pools. You can specify the size of the thread pool (that is, the number of threads) that are assigned for each message flow by specifying the number of additional instances of each message flow.

The mode that your broker is working in can affect the number of execution groups that you can use; see "Restrictions that apply in each operation mode" on page 3657.

A single default execution group is set up ready for use when you create a reference to a broker in the WebSphere Message Broker Toolkit. By setting up additional execution groups, you can isolate message flows that handle sensitive data such as payroll records, or security information, or unannounced product information, from other non-sensitive message flows.

If you create additional execution groups, you must give each group a name that is unique within the broker, and assign and deploy one or more message flows to each one.

You can create and deploy execution groups either in the WebSphere Message Broker Toolkit, or using commands.

An execution group process is also known as a DataFlowEngine (DFE); this term is typically used in problem determination scenarios (trace contents, diagnostic messages, and so on). A DFE is created as an operating system process, and has a one-to-one relationship with the named execution group. If more than one message flow runs within an execution group, multiple threads are created within the DFE process.

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Packaging and deployment overview" on page 3210
Deployment is the process of transferring data to an execution group on a broker so that it can take effect in the broker. Message flows and associated resources are packaged in broker archive (BAR) files for deployment.

Related tasks:

"Creating an execution group using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 937
Use the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer to create execution groups on your broker.

"Enabling ODBC connections to the databases" on page 668
Set up the resources and environment that the broker requires for Open Database Connectivity (ODBC) connections to user databases on distributed systems.

"Using WebSphere MQ trusted applications" on page 613
Configure a broker to run as a WebSphere MQ trusted application.

Related reference:

"Supported databases" on page 3591
You can optionally configure databases to contain data that is accessed by your message flows. Databases from IBM and other suppliers are supported at specific versions on supported operating systems.

"Support for 32-bit and 64-bit platforms" on page 3589
WebSphere Message Broker operates in 32-bit mode or 64-bit mode, on supported operating systems.

The Administration API for WebSphere Message Broker

The Administration API for WebSphere Message Broker is a programming interface that your applications can use to control brokers and their resources through a remote interface.

The Administration API for WebSphere Message Broker is also known as the Configuration Manager Proxy, or CMP API

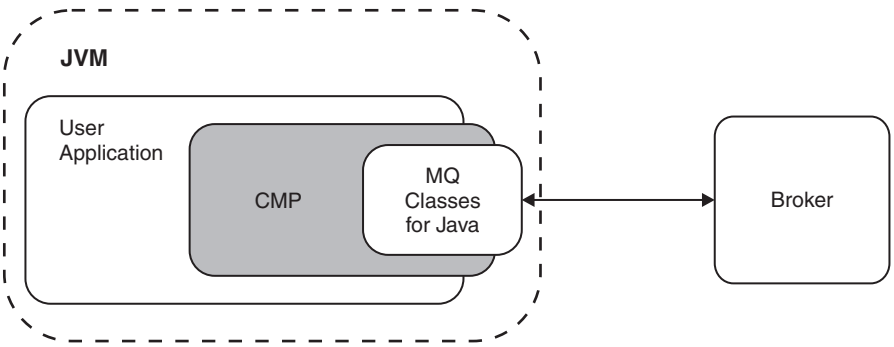
The Configuration Manager has been removed from Version 7, and the full name of the API has changed. However, the terms CMP application and

CMP API have been retained, and are used in this information center to refer to the Administration API, for continuity and consistency with the JAR file `ConfigManagerProxy.jar` that supplies all the classes.

The Administration API for WebSphere Message Broker (CMP API) consists solely of a Java implementation, and is referred to as the Message Broker Java API. Your applications have complete access to the broker functions and resources through the set of Java classes that constitute the CMP. Use the CMP API to interact with the broker to perform the following tasks:

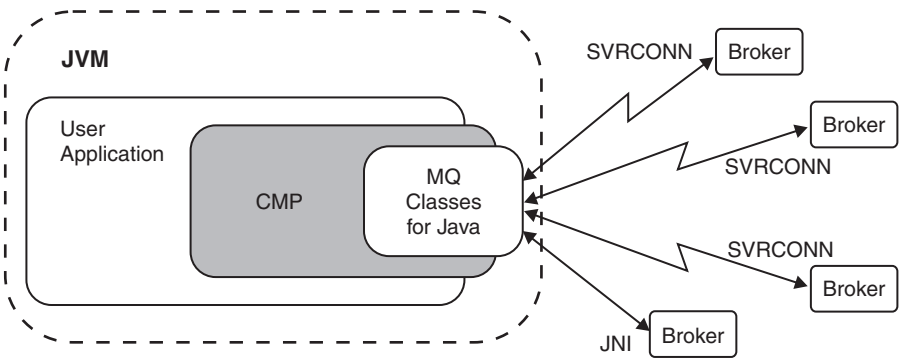
- Deploy BAR files
- Change the broker configuration properties
- Create, modify, and delete execution groups
- Inquire and set the status of the broker and its associated resources, and to be informed if status changes
  - Execution groups
  - Deployed message flows
  - Deployed files used by the message flows (for example, JAR files)
- View the Administration log

Interaction between CMP applications and the broker: The Java classes sit logically between the user application and the broker, inside the Java Virtual Machine (JVM) of the user application. The API requires the WebSphere MQ Classes for Java for connectivity, as shown in the following diagram.



The CMP application can be on the same physical computer as the broker, connected by a JNI (Java Native Interface) connection to the queue manager that uses the WebSphere MQ Java Bindings transport. If appropriate, you can distribute your applications over an Internet Protocol (TCP/IP) network, and connected to the broker by using a WebSphere MQ SVRCONN channel through the WebSphere MQ Java Client transport.

You can use the CMP API to communicate with more than one broker from within the same application, as shown in the following diagram.



Migrating from earlier versions of WebSphere Message Broker: If you are migrating to Version 7.0 from Version 6.1, the CMP is largely compatible, but you must take the following actions before you run your applications in a Version 7.0 environment:

- Check the source of your existing applications against the CMP Javadoc information that is provided with Version 7.0. Although a significant number of the classes and methods have been deprecated, they are still tolerated by the CMP, and appropriate action is taken if possible. For example, if your code connects to a Configuration Manager, the same connection properties are used by the CMP to connect to a broker.

  If appropriate action is not available, the behavior is undefined. Rework your applications to remove such classes and methods.
- Recompile your applications against the Version 7.0 libraries, even if you have not changed the source, to ensure that your code can take advantage of improvements in the updated classes.

You can work only with Version 7.0 brokers from your CMP API applications; earlier versions are not compatible and are not supported.

For further information about new and deprecated classes and methods, see "Administration API" on page 3672.

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

"Execution groups" on page 53
An execution group is a named grouping of message flows that have been assigned to a broker. The broker enforces a degree of isolation between message flows in distinct execution groups by ensuring that they run in separate address spaces, or as unique processes.

"WebSphere Message Broker Toolkit" on page 31
The WebSphere Message Broker Toolkit is an integrated development environment and graphical user interface based on the Eclipse platform.

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

"The Administration API samples" on page 958
Explore the samples to learn the basic features that are provided by the Administration API (also known as the CMP API). Run the samples to deploy a BAR file or manage a broker, or use the CMP API Exerciser to implement various

tasks.

Related tasks:

"Developing applications that use the Administration API" on page 956
Develop Java applications that use the Administration API (also known as the CMP API) to communicate with, deploy to, and manage brokers and their associated resources.

Related information:

Administration API for WebSphere Message Broker (CMP API)

WebSphere Message Broker Explorer

The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

The WebSphere Message Broker Explorer is an extension to the WebSphere MQ Explorer.

You can install the WebSphere Message Broker Explorer only on Windows and Linux on x86. You can view and interact only with brokers that you have created in WebSphere Message Broker Version 7 or later.

The WebSphere Message Broker Explorer

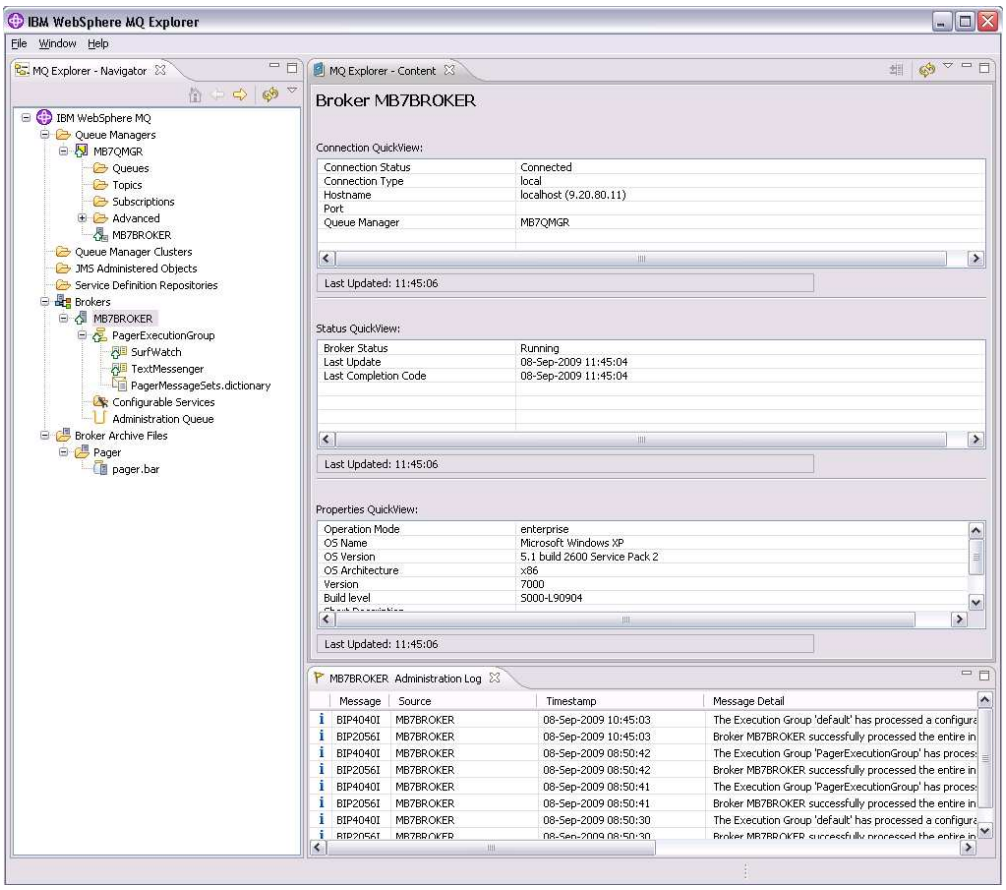
To use the WebSphere Message Broker Explorer, you must start the WebSphere MQ Explorer. The WebSphere Message Broker Explorer adds the Brokers and Broker Archive Files folders to the MQ Explorer - Navigator view:

- Use the Brokers folder to create, view, and modify brokers
- Use the Broker Archive Files folder to import, view, and modify broker archive files before deploying them to your brokers

If you cannot see these two folders in your MQ Explorer session, you have not installed the broker-specific plug-ins that are provided by WebSphere Message Broker Explorer. Close your MQ Explorer session, follow the instructions to install the WebSphere Message Broker Explorer, then start the MQ Explorer again.

The WebSphere Message Broker Explorer provides several QuickViews that you can use to view the properties of brokers and their resources. These QuickViews are automatically displayed when you click the resource in the Brokers folder in the MQ Explorer - Navigator view. A QuickView is also available for viewing the details of broker archive files that you have imported into the WebSphere Message Broker Explorer.

The following figure shows the QuickViews that are displayed when a broker is selected in the WebSphere Message Broker Explorer.



The following views and editors are provided for working with brokers in the WebSphere Message Broker Explorer:

Broker Archive editor
Use the Broker Archive editor to create and manage broker archive (BAR) files.

Broker Statistics and Broker Statistics Graph views
Use the Broker Statistics and Broker Statistics Graph views to view snapshot accounting and statistics data as it is produced by the broker.

Policy Sets and Policy Set Bindings editor
Use the Policy Sets and Policy Set Bindings editor to edit, save, import, and export policy sets or bindings.

Security Profiles editor
Use the Security Profiles editor to create a security profile for use with Lightweight Directory Access Protocol (LDAP) or Tivoli Federated Identity Manager (TFIM).

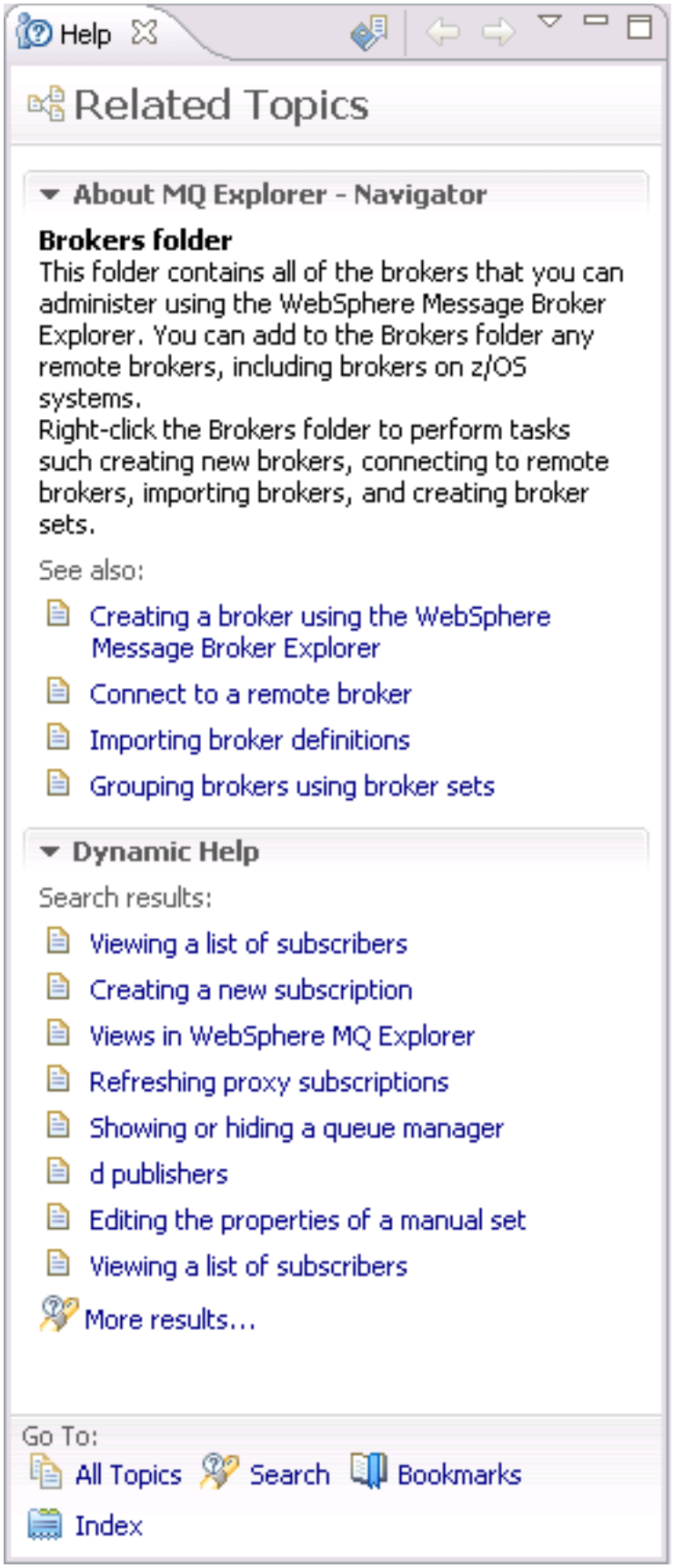
DataPower Security wizard
Use the DataPower Security wizard to configure an external DataPower appliance to handle the WS-Security Policy for your HTTP, HTTPS, and SOAP nodes within your message flow.

Administration Log view
Use the Administration Log view to view the results of deployment actions on brokers.

Accessing context-sensitive help

The Help view provides context-sensitive help throughout the WebSphere Message Broker Explorer. You can display the Help view for most aspects of the user interface by bringing focus to the object and pressing F1 (on Windows) or SHIFT+F1 (on Linux). The Related Topics page shows description and help topics that are related to the selected object. The About section shows context help that is specific to your current context, and the Dynamic Help section shows some search results that might be related.

The following figure shows the Related Topics page of the Help view that is displayed when you press F1 when the Brokers folder is selected in the WebSphere Message Broker Explorer.


Help
Related Topics
About MQ Explorer - Navigator
Brokers folder
This folder contains all of the brokers that you can administer using the WebSphere Message Broker Explorer. You can add to the Brokers folder any remote brokers, including brokers on z/OS systems.
Right-click the Brokers folder to perform tasks such creating new brokers, connecting to remote brokers, importing brokers, and creating broker sets.
See also:
Creating a broker using the WebSphere Message Broker Explorer
Connect to a remote broker
Importing broker definitions
Grouping brokers using broker sets
Dynamic Help
Search results:
Viewing a list of subscribers
Creating a new subscription
Views in WebSphere MQ Explorer
Refreshing proxy subscriptions
Showing or hiding a queue manager
d publishers
Editing the properties of a manual set
Viewing a list of subscribers
More results...
Go To:
All Topics
Search
Bookmarks
Index

Use the other pages in the Help view to view and search the contents of the information center. The All Topics page shows the table of contents of all the books in the information center. The Index provides an index of keywords of all the books in the information center. You can enter a keyword in the text field on the Index page to highlight the best match in the list of keywords. You can use the Search page to locate topics, samples, and remote documents using keywords in a search query. You can bookmark topics and other documents of interest, and view them in the Bookmarks page.

Related concepts:

"WebSphere Message Broker Toolkit" on page 31
The WebSphere Message Broker Toolkit is an integrated development environment and graphical user interface based on the Eclipse platform.

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related tasks:

"Creating a security profile" on page 433
You can create a security profile for use with Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2. Support is also provided for TFIM V6.1, for compatibility with previous versions of WebSphere Message Broker. You can create the security profile by using either the **mqsicreateconfigurableservice** command or an editor in the WebSphere Message Broker Explorer.

"Viewing message flow accounting and statistics data" on page 3300
You can use the Broker Statistics and Broker Statistics Graph views in the WebSphere Message Broker Explorer to view snapshot accounting and statistics data as it is produced by the broker.

Related reference:

"WebSphere Message Broker Explorer views" on page 6838
The WebSphere Message Broker Explorer can be used to manage and administer your brokers and deployed resources.

"Broker Archive editor" on page 6794
Use the Broker Archive editor to create and manage broker archive (BAR) files.

"Administration Log view" on page 6840
The Administration Log view shows administration requests and their results, changes made to objects, and the result of deployment actions on brokers. The Administration Log view can also describe configuration changes that have been automatically applied as a result of moving the broker from one version to another; for example, after applying maintenance.

"Policy Sets and Policy Set Bindings editor" on page 6841
The Policy Sets and Policy Set Bindings editor is the default editor for editing, saving, importing, and exporting a policy set or binding. You can define all the WS-Security policies for a single node, or a set of nodes. The policy set can be associated by the administrator with either a message flow or a node.

External systems and resources

You can configure WebSphere Message Broker resources to interact with a wide range of external systems and resources, such as WebSphere Process Server, and databases.

The products and standards listed here are for guidance only; for details of supported versions, you must check the WebSphere Message Broker Requirements Web page.

External systems

WebSphere Message Broker interfaces with other IBM products, and with products from other software vendors, to provide services that enhance message processing in the broker:

- IBM WebSphere Service Registry and Repository
- IBM WebSphere Process Server
- IBM WebSphere Integration Developer
- IBM WebSphere Business Monitor
- IBM WebSphere Transformation Extender
- IBM WebSphere MQ File Transfer Edition
- Enterprise Information Systems
  - SAP
  - Siebel
  - Peoplesoft
- Security
  - Tivoli Federated Identity Manager (TFIM)
  - Lightweight Directory Access Protocol (LDAP)
  - RACF and other External Security Managers, on z/OS only
- Management
  - Tivoli License Manager
- Citrix Presentation Server

External resources

WebSphere Message Broker interfaces with other IBM resources, and with resources from other software vendors, to provide extensions to message processing:

- Databases and ODBC support
  - IBM DB2
  - Oracle
  - Sybase
  - Microsoft SQL Server
  - IBM Informix®
- Databases and JDBC support
  - IBM DB2 Driver for JDBC and SQLJ
  - Microsoft SQL Server 2005 JDBC driver
  - Oracle JDBC Driver
  - Sybase jConnect for JDBC
  - IBM Informix JDBC
- File systems
  - FTP
  - SFTP
- Email systems
- Development repositories
  - Rational ClearCase
  - CVS
  - Other repositories supported by Eclipse

- Compilers (C and Java)
- Browsers
  - Microsoft Internet Explorer
  - Mozilla
- Adobe Flash Player for Quick Tour
- Adobe Acrobat for reading PDF files

External standards

Products and applications that adhere to certain specifications can also interact with WebSphere Message Broker:

- Java and JMS providers
- XSD
- WSDL
- SOAP
- XSLT
- WS-Addressing
- WS_Security

For further information about standards, see "General industry standards supported by WebSphere Message Broker" on page 3607.

WebSphere Message Broker business scenario

WebSphere Message Broker manages the flow of information in your business, applying message processing rules to route, store, retrieve, and transform the information as required by the different systems in your business and the systems of your customers, suppliers, partners, and service providers.

The following scenario outlines an example of how WebSphere Message Broker can be used to solve IT infrastructure problems in a business.

Mergers and acquisitions scenario

This scenario describes how WebSphere Message Broker is used by a fictitious insurance company to manage two disparate IT infrastructures after a small, Internet-based insurance company is acquired by a large, more traditional, insurance company. The description focuses on what happens when a potential customer requests a motor insurance quotation by using the merged company Web site. This scenario is based on a larger, more complex scenario that was published on developerWorks. To read the full scenario see the links at the end of the scenario description.

Background

Company A is a motor and general insurance company that has been in business for approximately 50 years and currently has approximately 5 million policyholders. The company uses agents and a call center to communicate with customers. The company has a large established IT infrastructure, which includes CICS Transaction Server for z/OS and IBM DB2 Universal Database™ on z/OS.

Company B is small Internet-based motor insurance company, which currently has less than 1 000 000 policyholders, and is expanding. The IT infrastructure managed

by the company includes WebSphere Application Server on Microsoft Windows Server 2003, and Oracle Enterprise and IBM DB2 Universal Database on Windows XP.

The problems

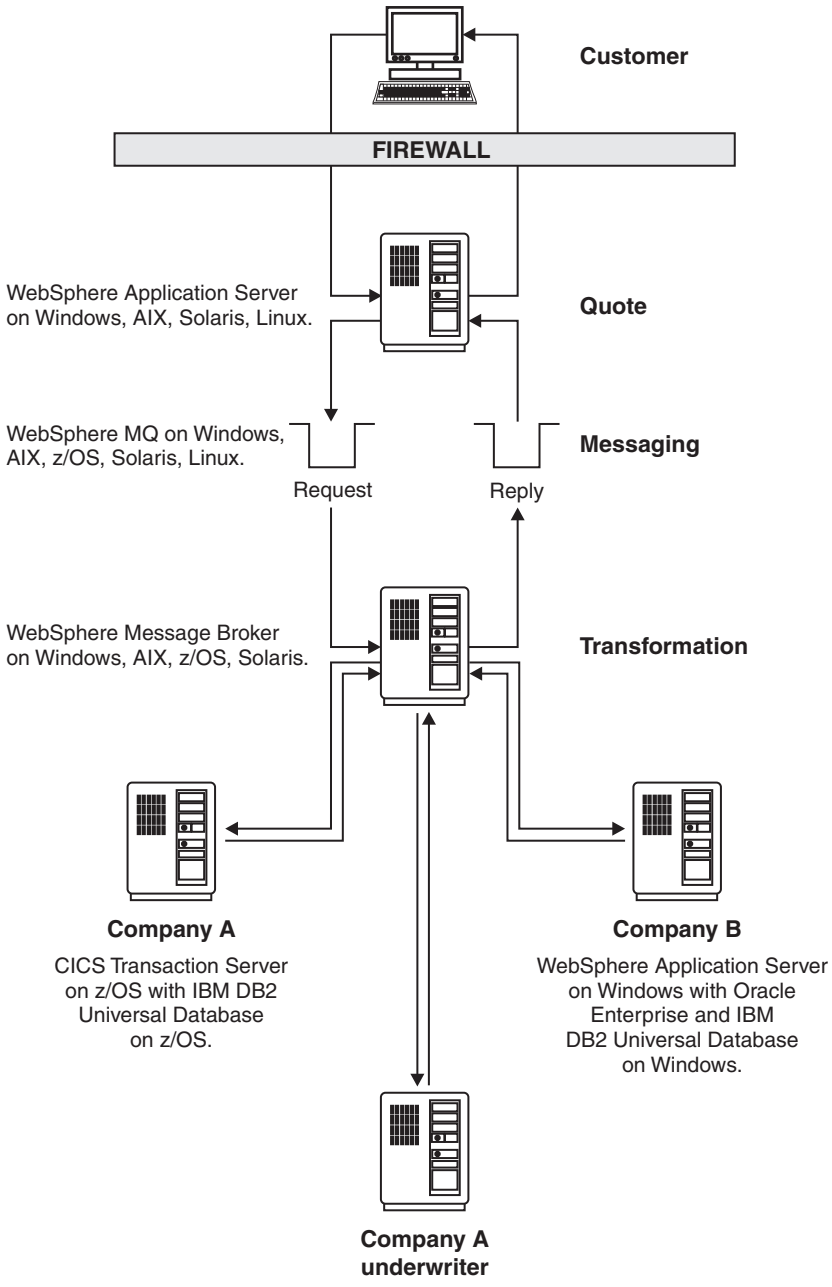
Company A acquired Company B to gain access to the Internet-based insurance market and to use the Internet-based skills and IT infrastructure established in Company B. The two companies have customer and policy data of different formats but, for legal reasons, the data from the separate companies cannot be merged. However, the administration costs of managing the separate IT infrastructures are high. Also, customers, agents, and call center staff need a single administration process to interact with the company data.

The solution

Now that the two companies have merged, users can request an insurance quotation by giving some basic personal information in a form on the Web site of the new company. WebSphere Application Server, on which the Web site runs, forwards the request in XML format to WebSphere Message Broker using the request queue in a WebSphere MQ cluster. WebSphere Message Broker transforms the XML request to the COMMAREA format that is used by Company A systems, then routes the request to those systems. WebSphere Message Broker also routes the request, in XML format, to Company B systems. Both systems return a quotation to WebSphere Message Broker.

Logic within WebSphere Message Broker also requests a risk assessment from the internal underwriter and applies the returned risk to the quotations from systems within Company A and Company B. The broker detects that, in this instance, the best or lowest quotation for the customer has been generated by Company A systems. Therefore, the broker transforms the quotation from Company A from COMMAREA to XML, and routes the quotation back to WebSphere Application Server to a reply queue in the WebSphere MQ cluster, where the quotation is stored for up to 14 days. WebSphere Application Server returns the quotation to the customer.

The following diagram shows the flow of information in this scenario.



Welcome to WebSphere Message Broker

New users: Use this topic to help you navigate through the WebSphere Message Broker information center:

- If you are new to WebSphere Message Broker, look at the Getting started section first.
- If you are ready to start developing message flows for WebSphere Message Broker, look at the Developing message flow applications section.
- If you are ready to design and configure your brokers, look at the WebSphere Message Broker administration section.
- The Additional information section includes links that might be of interest to all users.

Use the links in the following sections to go to start here topics that contain useful links into the information center or to external information. You can recognize the start here topics by the blue background, as shown in this paragraph. To return to this page, click **Navigation view** ( ) to display the contents of the information center, then select **Start here**. To find out where the page that you are currently viewing is located in the contents of the information center, click **Show in Table of Contents** ( ). You can use **Back** ( ) to go back through the pages you have previously viewed.

Some of the links in the start here topics work only if you are accessing this information center from the WebSphere Message Broker Toolkit or the WebSphere Message Broker Explorer. The links that work only in the WebSphere Message Broker Toolkit are displayed with this icon . Some of the links in the start here topics work only if you have a connection to the internet, and these links are displayed with this icon .

WebSphere iconGetting started

WebSphere Message Broker provides an advanced enterprise service bus, delivering universal connectivity and data transformation. The following introductory information describes the high-level concepts of the product and some basic tasks, including how to use this information center and how to use the WebSphere Message Broker Toolkit.

- "How do I use the information center?" on page 67 Find out how to use this information center, including navigation and searching techniques.
- "Where can I get an overview of WebSphere Message Broker?" on page 69 Get links to introductory information about WebSphere Message Broker.
- "How do I use the WebSphere Message Broker Toolkit?" on page 70 Find out how to use the WebSphere Message Broker Toolkit.
- "How can I check that my installation was successful?" on page 90 Find out how to check that your installation was successful.

WebSphere iconDeveloping message flows

Use the links in this section to help you to start developing your own message flows. Links are also included to help you to understand WebSphere Message Broker concepts that might be useful when you develop your own message flows. You can also use the Samples to help you to develop your own message flows. Most samples provide working examples of message flows, and demonstrate the features that are available in WebSphere Message Broker. To run samples, or deploy and test your own message flows, you must first create a broker.

You can use the Default Configuration wizard available in the WebSphere Message Broker Toolkit or the WebSphere Message Broker Explorer to create a basic broker configuration for testing message flows, see "Creating the Default Configuration" on page 106.

- "What do I need to know to start developing applications?" on page 72 Find out about the WebSphere Message Broker concepts you might want to know before developing your own message flows.
- "How do I design and develop applications?" on page 72 Start to develop your own message flows and associated resources.

- "How do I deploy and test message flows?" on page 86 Find out how to deploy your message flows to a broker and how to test them.
- How do I debug and troubleshoot my applications? Find out how to debug message flows and troubleshoot problems.

WebSphere iconWebSphere Message Broker Administration

To test and run your message flows, you must create and configure your brokers. Use the links in this section to find out how to configure and administer your brokers and message flows:

- "How do I plan and configure brokers?" on page 91 Find out how to plan and configure brokers for your business.
- "How do I deploy and configure message flow applications?" on page 94 Find out how to configure your message flows and get them running.
- "How do I administer and monitor brokers?" on page 93 Find out about day to day administrative tasks and about monitoring the performance of your message flows.

WebSphere iconAdditional information

Use the following links to get additional information about using WebSphere Message Broker:

- Where can I find out about the accessibility functions of WebSphere Message Broker? Find out about accessibility features in WebSphere Message Broker and shortcut keys to use in the information center.
- How can I troubleshoot any problems? Find out how to debug message flows and troubleshoot problems.
- "What information is available for users of previous versions?" on page 91 Find out about migration and what is new for users of previous versions of the product.

How do I use the information center?

New users: Use this topic to find out how to use the WebSphere Message Broker information center.

To view most of the following links you must be running the information center from within the WebSphere Message Broker Toolkit or the WebSphere Message Broker Explorer.

- What is the information center?

  Find out what the information center is, and the alternative ways that you can use to access it.
- How do I navigate the information center?

  Learn how to find your way around the information center.
- What are the accessibility features and keyboard shortcuts in the information center?

  Find out about accessibility in the information center and how to navigate using keyboard shortcuts.
- How do I search the information center?

  Learn how to search the information center, limiting your search scope and making your searches more effective.
- How do I set bookmarks and print topics?

Find out how to bookmark information to make it easier to find later. Also find out how to print topics for reading offline.

- How do I view information in different languages?

  Find out how to view the information center in different languages.

- Where can I find other sources of help information?

  Find out how the documentation is structured, and what other sources of help are provided in the information center and the WebSphere Message Broker Toolkit or the WebSphere Message Broker Explorer.

Where can I find other sources of help information?

New users: Use this topic to find out about the resources available in the information center. You can also find out about other ways that you can get help when using WebSphere Message Broker.

What resources are available in the information center?

The Contents pane in the WebSphere Message Broker information center lists high-level categories of information based on tasks that you might want to perform, such as configuring, administering, and developing applications. In each of the high-level categories, concept topics provide definitions and background information to help you understand the tasks in the product. Task topics provide the actions or steps that you complete for the associated tasks. You can find reference topics in the Reference category that provide supporting information to help you complete the tasks, for example lists of options and parameters for commands.

Use the following links as alternative ways to help you find the information that you want:

- Where can I find a glossary of terms and abbreviations?

  The glossary contains a list of terms and abbreviations that are used in the WebSphere Message Broker information center.

- Where can I find the information center index?

  Search for information about particular subjects or terms by using the Index.

- Where can I find support and technical documents on IBM Web sites?

  Search for WebSphere Message Broker support and technical documents on IBM Web sites. You must have an Internet connection to use the links on this page.

- Where can I find information about diagnostic messages?

  Search for information about diagnostic messages that might be generated when you use WebSphere Message Broker.

What other sources of help are there in WebSphere Message Broker?

User assistance is provided in WebSphere Message Broker, in the form of messages, context-sensitive help on wizards, and information provided on the product interfaces, including the WebSphere Message Broker Toolkit and in the command-line environment.

- How do I get context-sensitive help in the WebSphere Message Broker Toolkit?

  Context-sensitive help and links into the information center are available in the WebSphere Message Broker Toolkit in the Help view. The Help view can be launched by selecting an object in the WebSphere Message Broker Toolkit and pressing the F1 key (on Windows) or SHIFT+F1 (on Linux).

- How do I get context-sensitive help in the WebSphere Message Broker Explorer?

Context-sensitive help and links into the information center are available in the WebSphere Message Broker Explorer in the Help view. The Help view can be launched by selecting an object in the WebSphere Message Broker Explorer and pressing the F1 key (on Windows) or SHIFT+F1 (on Linux).

- **How do I get assistance when I am developing my applications?**

  You can get assistance when you develop your applications by using content assist. Content assist provides a list of available code options, for example the XPath functions to use in a message map, or a list of available references, for example to assist with constructing message references in ESQL. Content assist is available in the following editors when developing applications:

  – ESQL Editor
  – Java Editor
  – Message Mapping Editor

  You can also use content assist to help construct XPath expressions in the properties of some message flow nodes. Content assist can be displayed in the properties fields, and in the XPath Expression Builder.

  To access content assist, either select **Edit** > **Content Assist** or press Ctrl+Space to display a list of available options. To use content assist to help with constructing messages, you must have defined the message to the WebSphere Message Broker Toolkit and linked it to your message flow project; see "Project references" on page 44.

- **How do I get help when I am using commands?**

  You can get usage information for the commands by entering the command without any parameters, or alternatively entering the command followed by `/?`, for example:

  ```
  mqsilist /?
  ```

Where can I get an overview of WebSphere Message Broker?

New users: Use the links in this topic to get introductory information about WebSphere Message Broker.

- Where can I get a quick tour of the product?

  Learn about the basic concepts and features of WebSphere Message Broker with this animated tour.

- Where can I get a technical overview of the product?

  Learn about the capabilities and components of WebSphere Message Broker with this technical overview.

- Where can I find a WebSphere Message Broker example scenario?

  Learn how WebSphere Message Broker can be used to solve a business problem with this example scenario.

I am a developer; what tasks am I interested in?

If you are a developer, you will be involved in the tasks to create, test, debug, and run message flow applications.

Before you start

New users: Read the following introductory topics to the product, and the information center, before you start application development:

- "How do I use the information center?" on page 67
- "Where can I get an overview of WebSphere Message Broker?"

- "How do I use the WebSphere Message Broker Toolkit?"

Returning users: Find out what new features are supported for application development:

- "What's new in Version 7.0?" on page 7

WebSphere iconDeveloping message flow applications

Use the links in this section to help you to start developing your own message flow applications. Links are also included to help you to understand WebSphere Message Broker concepts that might be useful when you develop your own message flow applications.

You can also use the Samples to help you to develop your own message flow applications. Most samples provide working examples of message flow applications, and demonstrate the features that are available in WebSphere Message Broker. To run samples, or deploy and test your own message flow applications, you must first create a broker; use the Default Configuration wizard available in the WebSphere Message Broker Toolkit or the WebSphere Message Broker Explorer, to create a basic broker configuration for testing message flow applications. For more information about the Default Configuration wizard, see "Creating the Default Configuration" on page 106.

- "What do I need to know to start developing applications?" on page 72 Find out about the WebSphere Message Broker concepts you might want to know before developing your own message flow applications.
- "How do I design and develop applications?" on page 72 Start to develop your own message flow applications and associated resources.
- "How do I deploy and test message flows?" on page 86 Find out how to deploy your message flow applications to a broker and how to test them.

What else might I be interested in?

When you develop message flow applications, you might also be interested in the following questions:

- How can I troubleshoot problems? Find out how to debug message flow applications and troubleshoot problems.
- Where can I find out about the accessibility functions of WebSphere Message Broker? Find out about accessibility features in WebSphere Message Broker and shortcut keys to use in the information center.

How do I use the WebSphere Message Broker Toolkit?

New users: Use the links in this topic to help you get started using the WebSphere Message Broker Toolkit.

The WebSphere Message Broker Toolkit is an integrated development environment and graphical user interface based on the Eclipse platform. You can use the WebSphere Message Broker Toolkit for developing, testing, and debugging your applications.

Some of the links in this topic work only if you are accessing this information center from the WebSphere Message Broker Toolkit, or from the WebSphere Message Broker Explorer.

How do I navigate and customize the WebSphere Message Broker Toolkit?

Use the links in this section to find out how to navigate the WebSphere Message Broker Toolkit. You can also find information about the different resources that you can create in the WebSphere Message Broker Toolkit.

- What are the perspectives in the WebSphere Message Broker Toolkit?

  A perspective defines the initial set and layout of windows in the workbench. Each perspective provides a set of functionality aimed at accomplishing a specific type of task or working with specific types of resources. Follow this link to find out about the WebSphere Message Broker Toolkit perspectives.
- What are the editors in the WebSphere Message Broker Toolkit?

  Follow this link to find out about the different editors in the WebSphere Message Broker Toolkit.
- What are the views in the WebSphere Message Broker Toolkit?

  Use this link to find out how views are used in the Eclipse workbench. You can use the following links to find out more about the views that are used in association with different editors in the WebSphere Message Broker Toolkit:
  - Project Explorer view
  - Problems view
  - Tasks view
  - Outline view
  - Properties view
- What types of resource can I create in the WebSphere Message Broker Toolkit?

  Find out about the types of resources and files that you can create and develop in the WebSphere Message Broker Toolkit.
- What are the keyboard shortcuts in the WebSphere Message Broker Toolkit?

  Find out how to navigate by using keyboard shortcuts in the WebSphere Message Broker Toolkit.

How can I customize the Eclipse workbench?

You can use the instructions in these links to learn how to use any application running in the Eclipse workbench, including the WebSphere Message Broker Toolkit and Rational Application Developer.

- How do I work with perspectives?

  Find out how to work with perspectives.
- How do I work with views and editors?

  Find out how to work with views and editors such as rearranging views and opening files for editing.
- How do I customize the workbench?

  Find out how to work to further customize the Eclipse workbench such as rearranging the main toolbar and changing the placement of the tabs.

How can I change the WebSphere Message Broker Toolkit preferences?

You can change a number of settings in the WebSphere Message Broker Toolkit to suit your requirements. Use these links to get more information about these settings and how to change them.

- What WebSphere Message Broker Toolkit preferences can I change?

  Use this link to find out about WebSphere Message Broker Toolkit preferences that you can change, and get links to information about preferences that might be useful when you develop applications.
- How do I change the WebSphere Message Broker Explorer preferences?

  Follow this link to get an overview of how you change preferences to alter the behavior of the WebSphere Message Broker Explorer.
- How do I change the standard Eclipse preferences?

  Follow this link to find out about the standard preferences in the Eclipse workbench.

How do I design and develop applications?

New users: Use the links in this topic to get information about how to design and develop message flow applications and related resources.

- "What do I need to know to start developing applications?"

  Use this link to get information that it is useful to know before you start developing your message flow applications.
- How do I design a message flow?

  Follow this link to get a comprehensive list of the decisions that you need to make when designing a message flow.
- "How do I construct message flows?" on page 76

  Find out how to construct message flows.
- "How do I program message flows?" on page 78

  Find out how to program you message flow nodes by using ESQL, Java, message mappings, and XML transformations.
- "How do I design and develop a message model?" on page 83

  Find out how to develop message models.
- "How do I develop publish/subscribe applications?" on page 86

  Find out how to design and develop publish/subscribe applications.
- How do I develop user-defined extensions?

  Find out how to develop your own user-defined extensions such as nodes and parsers.

What do I need to know to start developing applications?:

New users: Before you start developing your applications, use the links in this topic to get information about concepts that you must understand.

This topic contains the following sections:

- "Basic application development concepts"
- "Advanced application development concepts" on page 76

Basic application development concepts:
What types of resources can I develop?

You can develop the following kinds of resources:

- Message flows

  A message flow is a sequence of processing steps that run in the broker when an input message is received. You define a message flow by including a number of

message flow nodes, each of which represents a set of actions that define a processing step. For more information about message flow nodes, see message flow nodes.

- Message models

  You can define the structure of messages for use with your message flows.

What is WebSphere Message Broker typically used for?

Use the links in this section to find out about the tasks that WebSphere Message Broker is commonly used for. You can also find out about the alternative ways to implement message flows for these tasks.

- **How can I use WebSphere Message Broker to route messages?**

  You can select from a number of ways of routing messages using message flows:

  – Using a Filter node

    You can route messages through a message flow based on the content of the message using a Filter node. The nodes that are connected to the different terminals of the Filter node can be used to apply different processing and routing to messages based on their content.

  – Using a Route node

    You can route messages through different paths in a message flow, based on the content of the message using a Route node. The Route node uses XPath expressions to control processing.

  – Using the destination list to route messages

    You can create a destination list to define the recipients of output messages using a Compute, Mapping, PHPCompute, or JavaCompute node. This list can then be used to route messages using RouteToLabel and Label nodes. Alternatively, a single message can be sent to many locations using a destination list for the destination mode, on some output nodes.

  You can see examples of how message flows can be used for routing in the following samples.

  – Message Routing
  – Airline Reservations
  – Simplified Database Routing

  You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

- **How can I use WebSphere Message Broker to transform and enrich messages?**

  You can select from a number of different ways of transforming and enriching messages using different nodes in message flows. Messages can be enriched with content from databases, applications, and files. The following methods of programming nodes in your message flows can be used to transform messages:

  – Databases

    WebSphere Message Broker supports various database managers so that you con configure your brokers to interact with databases on behalf of your message flows. Follow this link to get instructions on how to create and configure databases and connections.

  – ESQL

Extended Structured Query Language (ESQL) is a programming language based on Structured Query Language (SQL) that is typically used to work with databases. ESQL is extended to access and manipulate data in messages and databases. You can code ESQL to transform and enrich messages using the Compute node.

- Java

  Use the JavaCompute node to add valid Java code to your message flow. You can access, create, and transform messages in your message flows by using Java.

- Message mappings

  Message mappings use a drag-and-drop interface to transform messages. You can use conditional logic, ESQL functions, Java functions, and XPath functions to create complex mappings and transformations. You can also use message mapping to select and manipulate data in database tables. You can create message mappings to transform and enrich messages using the Mapping node.

- PHP

  Use the PHPCompute node to add valid PHP code to your message flow. You can access, create, and transform messages in your message flows by using PHP.

- XML transformation

  You can use the XSLTransform node to transform from one type of XML message to another, based on rules provided by an Extensible Stylesheet Language (XSL) style sheet.

The Airline Reservations demonstrates message enrichment, and the following samples demonstrate message transformation.

- Video Rental
- XSL Transform

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

- **How can I use WebSphere Message Broker with Web services?**

  WebSphere Message Broker provides a number of ways to work with Web services:

  - "WebSphere Message Broker and Web services" on page 1602

    Find out how WebSphere Message Broker can be used to integrate applications using Web services.

  - Web services scenarios

    Find out about the different Web services scenarios that WebSphere Message Broker supports.

  - "Processing Web service messages" on page 1601

    Find out more about working with Web services in WebSphere Message Broker.

  - Web services using HTTP nodes
  - SOAP Nodes

How can I connect my applications?

Use the links in this section to find out how you can connect your applications to WebSphere Message Broker.

- What are the application communication models?

  Find out about the differences between the two types of application communication model that WebSphere Message Broker supports.
- What application programming interfaces are supported?

  Find out about the many programming interfaces supported by WebSphere Message Broker.
- Connecting applications

  Find out about the communication transports and protocols that WebSphere Message Broker supports.
- "Publish/Subscribe" on page 2215

  Publish/subscribe is a style of messaging in which the providers of information (publishers) are decoupled from the consumers of that information (subscribers).

What is a logical message tree?

Use the links in this section to find out about the logical message tree and the internal representation of messages within message flows. When a message is received by a message flow, a logical structure is created, known as the logical message tree or the message assembly. The message assembly contains a message tree, which is the internal representation of the physical message, and a number of other trees that are used to store information during the processing of the message in the message flow.

- What is the logical message tree?

  Use this link to get more information about the logical message tree and how it is used in application development. You can get information about the four subtrees from the following links:
  - "Message tree structure" on page 1045
  - "Environment tree structure" on page 1055
  - "Local environment tree structure" on page 1056
  - "Exception list tree structure" on page 1066
- How is the message tree populated?

  Follow this link to get detailed information about how the message tree is populated when a message enters a message flow, and how the structure of the message tree is affected by the transport protocol that the message is received on.

What are physical message formats?

Use the links in this section to get information about the physical messages received by message flows, and to find out how the different structures and formats are handled by your applications.

- What are the differences between predefined and self-defining messages?

  Find out about the differences between predefined and self-defining messages in WebSphere Message Broker. The design of your applications can be influenced by your decision to use predefined or self-defining messages. If you use predefined messages, you must create a message model to define the logical

message structure to be used by the nodes in your message flows, but you can use self-defining messages instead, or as well.

- "Why model messages?" on page 1158

  Find out about the benefits of modeling messages, even if you use self-defining messages in your system.

- Where can I find an introduction to message modeling?

  Follow this link to get introductory information about message modeling.

- What is a parser?

  A parser is used to interpret the physical structure of an incoming message and create an internal representation of the message in a tree structure - the logical message tree. Follow this link to get an introduction to parsers.

- Which message domain and format can I use?

  A message model has one or more domains that determine which parser is used to construct the logical tree structure from the physical message. Follow this link for guidance on selecting a domain and parser for your message model.

- How can I define a physical format for my message?

  Follow this link to find out how you can define the physical structure for your messages. The following links provide information about specific types of message format:

  – What is a Custom Wire Format?
  – What is an XML Wire Format?
  – What is a Tagged/Delimited String Format?

Advanced application development concepts:
What other resources can I develop?

- "Developing message flows that use WebSphere Adapters" on page 2033

  You can develop applications that connect to an Enterprise Information System (EIS) using WebSphere Adapters.

- "User-defined extensions overview" on page 2971

  A user-defined extension is an optional component that is designed by the user to extend the functions provided by WebSphere Message Broker. A user-defined extension can be either a node or a parser.

- Administration API applications

  The Administration API for WebSphere Message Broker (also known as the CMP API) is an application programming interface (API) that you can program to control brokers by using a remote interface to the appropriate broker.

How do I construct message flows?:

New users: when you have considered the various factors involved in designing a message flow you are ready to create one.

Use the links in this topic to learn how to construct message flows and work with related resources.

How do I create message flow resources?

- How do I create a message flow project?

  A message flow project is a specialized container in which you create and maintain all the resources associated with one or more message flows. You can group together related message flows and resources in a single message flow project to provide an organizational structure to your message flow resources.

You must create a project before you can create a message flow. Use this link to learn how to create a message flow project.

- How do I create a broker schema?

  If you want to organize your message flow project resources, and to define the scope of resource names to ensure uniqueness, you can create broker schemas. When you first create a message flow project, a default broker schema is created in the project. Use this link to learn how to create a broker schema.

- How do I create a message flow?

  You create a message flow to specify how to process messages in the broker. You can create any number of message flows and deploy them to one or more brokers. Use this link to learn how to create a message flow.

How do I construct a message flow?

- How do I add a node?

  When you create a message flow, the first action to take to define its function is to add nodes. A message flow node is a processing step in a message flow. A message flow node can be a built-in node, a user-defined node, or a subflow node; see “Message flow nodes” on page 1024. Select the nodes to add to your message flow from the node palette, see “Message flow node palette” on page 1027.

- How do I rename a message flow node?

  To make your message flows easier to understand and to maintain, you can change the name of any type of node that you have added to your message flow. For example, you might change the name of an MQInput node to match the input queue name defined in the node.

-

  When you have included an instance of a node in your message flow, you can customize its function. Each node has a set of properties that are specific to the function of that node. For a list of nodes that are provided with WebSphere Message Broker, see “Built-in nodes” on page 4293. Select the required node from this list to view details of the terminals, and properties, and how to configure the node. For more information about nodes that can be programmed, such as the JavaCompute, Compute, and Mapping nodes, see “How do I program message flows?” on page 78.

- How do I connect nodes?

  You connect the nodes in your message flow to indicate how the flow of control passes from input to output, and the route that messages can take through the message flow. A message flow node has a fixed number of input and output points known as terminals. You can connect the failure or catch terminals of nodes to add error handling to your message flows, see “Handling errors in message flows” on page 2823.

- How do I align and arrange nodes in a message flow?

  You can change the way that the nodes in your message flow are displayed and arranged, to make them easier to read. You can add a bend point to make the flow of control easier to follow where node connections cross each other, see “Adding a bend point” on page 1527.

- How do I save a message flow?

  Save the message flow when you want to do any of the following actions:

  – Close the workbench
  – Work with another resource
  – Validate the contents of the message flow

How do I manage my message flows?

- How do I delete a message flow node?

  Learn how to remove a node from your message flow.
- How do I delete a node connection?

  Learn how to remove a node connection between nodes in your message flow.
- How do I delete a message flow?

  Learn how to delete your message flow.
- How do I rename a message flow?

  Learn how to rename your message flow.
- How do I delete a message flow project?

  Learn how to delete your message flow project.
- How do I move a message flow?

  Learn how to move your message flow between broker schemas or message flow projects.
- How do I add a subflow?

  After you have created message flows, you can embed them in other message flows as subflows. You can use subflows to reuse function across message flow projects, reduce development time, and increase the maintainability of your message flows.
- How do I back up my message flow projects and related resources?

  Your message flow projects and other resources are stored in the WebSphere Message Broker Toolkit workspace and other locations in your file system. You can use the export function in Eclipse, or take copies of the directories on your file system, to back up your resources. Alternatively, you can set up the WebSphere Message Broker Toolkit to work with a repository, see "Development repository" on page 45.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

How do I program message flows?:

New users: Use the links in this topic to get information about the concepts and tasks associated with programming message flows.

This topic contains the following sections:

How do I use message mappings?: Message mappings use a drag-and-drop interface to create and transform messages. You can use conditional logic, ESQL functions, and XPath functions to create complex mappings and transformations. You can also use message mapping to select and manipulate data in database

tables. You must create a message definition for any messages that you include in a message mapping. Message map files are stored in message flow projects.

- How do I create a message map file?

  Follow this link for instructions on how to create a message map in the Broker Application Development perspective. You can also create a message map from the nodes that support message mapping including:

  – Mapping node
  – DataInsert node
  – DataUpdate node
  – DataDelete node

- What does the Message Mapping editor look like?

  You create and modify message mappings in the Message Mapping editor. Use this link to discover information about the Message Mapping editor.

- How do I configure message mappings?

  When you configure message mappings, you can drag content from a source to a target. The source can be a message, a database, or both, and the target can be messages, database tables, or both. If your target is a database, you can select the database operation (insert, update, or delete) that you want to perform on the table. You can set the value for your target to be a constant, or you can use a function or expression to produce the value. Additionally, you can configure conditional mappings to set different values for targets based on the content of the sources, and to handle repeating elements in sources and targets. Follow this link to discover more about message mappings.

- **Message mapping syntax**

  When you use an expression to set the value of a target in a message map, the expression must be in XPath format.

  – What syntax is used in mapping nodes?

    Use this link to discover more about the XPath syntax used in message maps.

  – How do I use expressions in my message maps?

    Use this link to discover more about the type of functions that you can use in expressions in your message maps.

  – How can I find out more about XPath query syntax?

    To discover more about XPath, follow this link to the W3C recommended XPath 1.0 query syntax reference document. This link works only if you have an active internet connection.

- How do I create and call submaps and routines?

  You can reuse message maps between different message flows and message flow projects by using a submap. You can also use a submap to create message mappings for a wildcard source so that you can select replacement elements, for example to select the appropriate body element from a SOAP message. In addition to calling a submap from a message map, you can call a submap from ESQL. You can also call ESQL routines from a message map. Use this link to discover more information about creating and using submaps and ESQL routines.

How do I use ESQL?: Extended Structured Query Language (ESQL) is a programming language based on Structured Query Language (SQL), which is commonly used with relational databases such as DB2. ESQL extends the constructs of the SQL language to provide support for you to work with both message and database content. ESQL can be used with the Compute, Database, and Filter nodes.

Many of the WebSphere Message Broker "Samples" on page 98 show how to use ESQL in message flows.

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

The following samples contain example ESQL code:

- Airline Reservations
- Large Messaging
- Message Routing

Use the following links to discover how to use ESQL in your message flows:

- How do I create an ESQL file?

  The ESQL code for each of your nodes is contained within a module in an ESQL file. Use this topic to discover how to create an empty ESQL file in your message flow project. Alternatively, you can select **Open ESQL** to create an ESQL file when you configure the first node in a message flow that uses ESQL. This action creates an ESQL file with skeleton ESQL code for a module associated with the selected node. You can also select an ESQL module for a node from a different message flow project by creating "Project references" on page 44.
- What is the ESQL editor?

  You can create and modify your ESQL code in the ESQL editor. You can change the way that code is displayed in the ESQL editor, and modify the way in which the ESQL editor validates code, by changing your ESQL preferences.
- How do I create ESQL for a node?

  Follow this link to get an overview of how to create ESQL for your node, including more information about the differences between the skeleton ESQL code generated for the modules associated with the Compute, Database, and Filter nodes.
- How do I modify ESQL for a node?

  Follow this link to discover how to modify the skeleton ESQL module code.
- How do I save an ESQL file?

  Discover how to save your ESQL file.
- How do I write ESQL code?

  Follow this link to get introductory information about writing ESQL code for your message flows. Discover how to manipulate the message tree, transform data, access databases, and work with messages from different message domains using ESQL.
- **ESQL language**

  Use the following the links to get concept and reference information about the ESQL language:

  – "ESQL data types" on page 2373
  – "ESQL variables" on page 2374
  – "ESQL operators" on page 2382
  – "ESQL field references" on page 2381
  – "ESQL statements" on page 2383
  – "ESQL functions" on page 2385

- "ESQL procedures" on page 2386
- "Special characters, case sensitivity, and comments in ESQL" on page 5305
- "ESQL modules" on page 2388
- "ESQL reserved keywords" on page 5307
- "ESQL non-reserved keywords" on page 5307

How do I use Java?: You can create a Java class file for a JavaCompute node and code Java functions to tailor the behavior of the node. You can add any valid Java code to your JavaCompute nodes and use the Java user-defined node API to process messages. You manage Java files through the Java perspective.

The following sample provides a collection of message flows that show how to use the JavaCompute node:

- JavaCompute Node

  You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.
- How do I use a JavaCompute node?

  You can use the JavaCompute node to examine the content of an input message, transform a message, and build new messages. Follow this link to discover how to use and configure the JavaCompute node in your message flows.
- How do I create Java code for a JavaCompute node?

  Discover how to create a Java class file using the JavaCompute node wizard. The JavaCompute node wizard creates a Java class with skeleton Java code that is based on the options that you select in the wizard. You can then modify the skeleton code to perform your own processing.
- How do I open an existing Java file?

  You can modify existing Java code that you have created in a Java Project.
- How do I save a Java file?

  Discover how to save your Java file.
- How do I write Java for message flow applications?

  Get introductory information about writing Java code for your message flow applications, including how to manipulate the message tree, access databases, handle errors, and access broker properties.
- What views and editors do I use when programming Java?

  Get a list of concept, task, and reference topics that relate to editors and views for working with Java.
- Where can I get assistance when programming Java?

  You can use code assist to provide a list of available command completions that you can select to insert into the editor. You can also use command assist to access Javadoc information about code in the Java editor.
- How do I add Java code dependencies?

  Discover how to include references to other Java projects and JAR files in your JavaCompute node code.
- Where can I find the Java user-defined node API?

  Follow this link to the Java API for the WebSphere Message Broker classes for creating a Java user-defined node, which you can also use to code your JavaCompute node.

How do I use XML transformations?: You can use the XSLTransform node to transform an XML message into another form of XML according to the rules provided by an XSL (Extensible Stylesheet Language) style sheet. You can specify the location of the style sheet to apply to this transformation in three ways:

- By using the content of the XML data within the message itself, which transforms the message according to a style sheet that the message itself defines.
- By setting a value within the LocalEnvironment folder.
- By using node properties, which ensures that the transformation that is defined by this single style sheet is applied to every message that is processed by this node.

You can discover links to information about using XML transformation in this section.

- Where can I find a sample that shows XML transformations?

  If you have installed the WebSphere Message Broker Toolkit, you can use this link to display the XSL Transform sample. The XSL Transform sample is a message flow sample that shows how to use a message flow to transform an XML message to another form of XML message according to the rules provided by an XSL stylesheet.

  You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.
- How do I use the XSLTransform node?

  Follow this link to discover how to use and configure the XSLTransform node to transform an XML message into another form of XML.
- Where can I find out more about XSL Transformations?

  To discover more about XML Transformations, follow this link to the W3C specification of the syntax and semantics of the XSL Transformations language for transforming XML documents into other XML documents.

How do I use PHP?: PHP is a scripting language that you can code to route and transform messages. You can use PHP in the PHPCompute node, which is supported only on Windows.

- Where can I get an overview of PHP?

  Follow this link for an overview of PHP, and links to further information about how you can use this scripting language.
- Where can I find a sample that shows how to use PHP?

  If you have installed the WebSphere Message Broker Toolkit, you can use this link to display the PHP sample. The PHP sample is a message flow sample that shows how to use PHP code to transform an XML message.

  You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.
- How do I use a PHPCompute node?

  Follow this link to discover how to use and configure the PHPCompute node in your message flows.
- How do I create PHP code for a PHPCompute node?

Discover how to create a PHP script file for the PHPCompute node in your message flows.

How do I use XPath?: The XML Path Language (XPath) is a language used to uniquely identify or address parts of an XML document. An XPath expression can be used to search through an XML document, and extract information from the nodes (any part of the document, such as an element or attribute) in it. XPath expressions can be used in message maps, and in the properties of some of the nodes. Typically, you might use XPath when you are using Java, XML, or PHP.

- Where can I get an overview of XPath?

  Follow this link for an overview of XPath, and how you can use XPath with nodes.
- How do I use the XPath Expression Builder?

  You can use the XPath Expression Builder to add XPath expressions to your node properties. Follow this link for information about how to use the XPath Expression Builder.

How do I design and develop a message model?:

New users: how to design and create message models.

If the format of the messages that you want to use with your applications is not self-defining, you must create a message model that defines the structure of your messages. If your messages are self-defining, you might want to create a message model to take advantage of runtime validation of messages, reuse of messages, automatic generation of documentation, and code completion on the message structure when you use ESQL. If you want to use message mapping, you must also create a message model for your messages.

You can obtain prebuilt models for common industry standard message formats such as SWIFT, EDIFACT, X12, FIX, HL7, and TLOG to use with WebSphere Message Broker. You can also create message models from C header files, COBOL copybooks, XML Schema and DTDs, and WSDL files. Alternatively, you can use the Message Definition editor to create your own message models.

For further information, read the logical and physical message structure sections in "What do I need to know to start developing applications?" on page 72.

What are the components of a message model?

Use the links in this section to learn how to create and configure message set projects and message set files.

- Use the following links to learn about the different components that are used to build a message model:
  - What is a message set project?

    A message set project is a container in which you create and maintain all of the resources that are associated with a single message set. A message set can contain one or more message models.
  - What is a message set?

    A message set is a logical grouping of messages and the objects that comprise them (elements, types, and groups). A message set can contain one message set file, message definition files, and message category files.
  - What is a message definition file?

A message definition file contains the messages, elements, types, and groups that make up a message model. The message definition file contains the logical model and associated physical model in XML Schema form for a group of related messages.

– What is a message model object?

  Get an overview of the objects that make up a message model.

– What is a message category?

  You can use message categories to group your messages for documentation purposes.

– What is a multipart message?

  You can define a message that contains one or more embedded messages within its structure.

– How can I control the number of occurrences of an element or attribute in a message?

  You can define a message that contains repeating, optional, and mandatory elements.

How do I create and configure a message model?

Use the links in this section to learn how to create and configure message set projects and message set files. The following sample provides step-by-step instructions about how to create a simple message model. The sample also demonstrates message transformation between three different message formats.

- Video Rental

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

- How can I use the Quick Start wizards to help create resources?

  You can use the Quick Start wizards to help you to create a message model and to set up the resources that you require to develop a WebSphere Message Broker application.

- How do I create a message set?

  You must create a message set before you can add any content to your message model. When you create a message set, a message set project is also created.

- **How do I configure a message set file?**

  After you have created a message set, you must configure the logical and physical properties of the message set. You can use the Message set editor to configure the properties of a message set. See “Message set editor” on page 6819.

  – How do I configure the logical properties of my message set?

    Learn how to configure logical properties of a message set.

  – How do I configure the physical formats of my message set?

    Learn how to add and configure different physical format layers in a message set.

- How do I create a message definition file?

  You must create a message definition file before you can create the message model objects. You can use the New message definition file wizard to create an empty message definition file, or you can populate a message definition by

using existing application message formats by importing data structures. See "New message definition file wizards" on page 6360 and "Working with data structures" on page 2930.

- How do I configure the properties of my message definition files?

  When you have created a message definition file, you can configure the message definition file properties described in this topic by using the Message Definition editor. See "Message Definition editor" on page 6804.

- **How do I add and configure message model objects?**

  When you have created a message definition file you can add message model objects to your message definition file, to define the structure of your message. You can also add objects to existing message definition files. You must then configure the logical, physical, and other properties of the message model objects.

  – How do I add message model objects?

    Follow this link to learn how to add different message model objects to a message definition file.

  – How do I configure message model objects?

    Follow this link to learn how to configure message model objects.

  – What properties do message model objects have?

    Follow this link to get reference information about logical, physical, and documentation properties for all types of message model objects.

  – How do I create a multipart message?

    You can create a message model that includes a multipart (embedded) message. Use this link to learn how to create a multipart message.

  – How do I link from one message definition file to another?

    Follow this link to learn how to link one message definition file to another.

- How are namespaces used in the message model?

  Objects in the message model such as elements, attributes, types, and groups are identified by their name. No two objects in the same scope are allowed to have the same name. If namespaces are enabled for a message set, each message definition file within it can specify a namespace. Global objects within namespaces can share the same name, therefore namespaces provide a way to avoid name clashes between objects.

How do I manage my message models?

Learn about how to manage your message models:

- **How do I generate model representations from message models?**

  When you have created and populated a message set, you can generate a message model in different representations for use both by WebSphere Message Broker and your applications. Use the following links to learn about the types of model representations that you can generate from your message models:

  – "Generate message dictionaries" on page 1271

  – "Generate XML schema" on page 1272

  – "Generate WSDL" on page 1274

- How do I configure message set preferences?

  Follow this link to learn how to make changes to preferences that relate to message set processing.

- How do I change my viewing preferences for message model editors?

You can change your viewing preferences for the Message Set editor and the Message Definition editor, to make it easier to view message definition files.

How do I develop publish/subscribe applications?:

New users: use the links in this topic to get concept and reference information that is useful when you develop publish/subscribe applications.

Publish/subscribe is a style of messaging application in which the providers of information (publishers) are decoupled from the consumers of that information (subscribers) using a broker. The following samples are examples of simple publish/subscribe message flows, and include associated user applications. You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

- Pager
- Scribble

A publish/subscribe application can consist of a network of brokers connected together. By connecting brokers together, publications can be received by a client on any broker in the network.

- Where can I find an overview of publish/subscribe?

  For an overview of publish/subscribe, see the following topics:
  - "Publishers" on page 2218
  - "Publications" on page 2219
  - "Subscription points" on page 2222
  - "Subscribers" on page 2220
- How do I use a publication node?

  Use the Publication node in your message flows to filter output messages and transmit them to subscribers who have registered an interest in a particular set of topics.
- What are command messages?

  Different types of command messages can be sent from an application to instruct the broker to perform the required task; for example to publish a message, or register a subscriber.

How do I deploy and test message flows?:

New users: Use this topic to get information about deploying and testing your message flows during development. When you have developed your message flows, message models, and other resources, you are ready to deploy them to a broker.

You can use features provided in the WebSphere Message Broker Toolkit to help you test your message flows. Read "How do I deploy and configure message flow applications?" on page 94 for information about deploying and configuring message flows for use in a production environment.

Before you can deploy and test your message flows, you must create and configure at least one broker.

You can create the Default Configuration to create a broker environment that is suitable for testing message flows during development, see "Creating the Default

Configuration" on page 106. You can create the Default Configuration only if you are accessing the information center from the WebSphere Message Broker Toolkit or the WebSphere Message Broker Explorer.

How do I deploy message flows and related resources?

Use the links in this section to get instructions on how to deploy your message flows, message sets, and related resources to a broker. Read "Message flow application deployment" on page 3213 for an overview of the concepts of deployment of message flow resources.

- How do I create a broker archive?

  A broker archive file is a compressed file that contains compiled message flows, message dictionaries, and other application development resources. Use this link for instructions about how to create a broker archive file. You can use the "Broker Archive editor" on page 6794 to add or remove message flows and message sets to, or from, your broker archive. You can also use the Broker Archive editor to edit the configurable properties in your broker archive.
- How do I add files to a broker archive?

  You can add message flows and message sets to a broker archive file. The message flows, message sets, and related resources are compiled when they are added to the broker archive file. You can also choose to add the source files to the broker archive file. You cannot add resources to a broker archive file from a project that contains an error.
- How do I deploy a broker archive?

  Find out about the different ways that you can deploy a broker archive file to a broker. For deploying message flows for development to a test environment, the simplest method is to use the WebSphere Message Broker Toolkit.
- How do I check the results of the deployment?

  Find out about the ways that you can check the results of deployment. When you deploy message flows to a test environment, the simplest method is to use the WebSphere Message Broker Toolkit. You can also use the "**mqsilist** command" on page 3882 with the name of the broker and execution group to which you deployed your broker archive file.
- How do I edit configurable properties?

  System objects that are defined in message flows have configurable properties that you can update within the broker archive before deployment. By changing configurable properties, you can customize a broker archive for a new domain, without needing to edit and rebuild the message flows or other resources.
- How do I refresh the contents of a broker archive?

  You can refresh the contents of a broker archive by removing resources from it and, having made required changes, add them back again. Alternatively, you can use the Refresh option in the Broker Archive editor.

How can I test my message flows?

Use the links in this section provide information about the different features provided in the WebSphere Message Broker Toolkit to help you to test your message flows.

- How do I test a message flow using the Test Client?

  You can test message flows that use WebSphere MQ or HTTP input and output nodes. If you have a message sets defined for the input nodes in your message

flow, the message flow test tool can provide an input message template in XML format, that you can use to develop test messages.

- How can I put a test message onto a WebSphere MQ queue?

  Get instructions about how to create and use an enqueue file to put a test message on to a WebSphere MQ queue.
- How can I get a test message from a WebSphere MQ queue?

  Get instructions about how to read a message from a WebSphere MQ queue by using the Dequeue Message dialog box.
- How can I test my message flows?

  You can use the flow debugger to help test your message flows by tracking the path a message takes through your message flow. You can also step through the ESQL, Java, and message mappings in your message flows, to view the output messages that are being constructed.
- "How can I diagnose problems?" on page 96

  If you are having difficulty deploying your message flows and other resources, or receiving unexpected results when you test your message flows, look at the problem determination information in this topic.

I am an administrator; what tasks am I interested in?

If you are an administrator, you will be involved in the tasks to configure brokers and associated resources, and support your application developers.

Before you start

New users: Read the following introductory topics to the product, and the information center, before you start configuration:

- "How do I use the information center?" on page 67
- "Where can I get an overview of WebSphere Message Broker?" on page 69
- "How do I use the WebSphere Message Broker Explorer and WebSphere Message Broker Toolkit?" on page 89

Returning users: Find out what new features are supported for configuration and administration:

- "What's new in Version 7.0?" on page 7

WebSphere iconWebSphere Message Broker Administration

To test and run your message flow applications, you must create and configure your brokers. Use the links in this section to find out how to configure and administer your brokers and message flow applications:

- "How can I check that my installation was successful?" on page 90 Find out if your installation has been successful.
- "What information is available for users of previous versions?" on page 91 Find out what you need to know if you are a user of a previous version or release.
- "How do I plan and configure brokers?" on page 91 Find out how to plan and configure brokers for your business.
- "How do I administer and monitor brokers?" on page 93 Find out about day to day administrative tasks and about monitoring the performance of your message flow applications.
- "How do I deploy and configure message flow applications?" on page 94 Find out how to configure your message flow applications and get them running.

What else might I be interested in?

When you configure and administer your brokers, you might also be interested in the following questions:

- How can I troubleshoot problems? Find out how to debug message flow applications and troubleshoot problems.
- Where can I find out about the accessibility functions of WebSphere Message Broker? Find out about accessibility features in WebSphere Message Broker and shortcut keys to use in the information center.

How do I use the WebSphere Message Broker Explorer and WebSphere Message Broker Toolkit?

New users: use the links in this topic to help you get started using the WebSphere Message Broker Explorer and WebSphere Message Broker Toolkit.

The WebSphere Message Broker Explorer is an administration environment and graphical user interface based on the Eclipse platform, and integrated with the WebSphere MQ Explorer. You can configure, administer, view, and monitor your brokers, and their associated resources, by using the options in this workbench.

The WebSphere Message Broker Toolkit enables you to complete some basic administration functions for testing and debugging your message flow applications.

Some of the links in this topic work only if you are accessing this information center from the WebSphere Message Broker Explorer or the WebSphere Message Broker Toolkit.

How do I navigate and customize the WebSphere Message Broker Explorer?

Use the links in this section to find out how to navigate the WebSphere Message Broker Explorer.

- What are the keyboard shortcuts?

  Find out how to navigate by using keyboard shortcuts in the WebSphere Message Broker Explorer.

How can I customize the Eclipse workbench?

You can use the instructions in these links to learn how to use any application running in the Eclipse workbench, including the WebSphere Message Broker Explorer and Rational Application Developer.

- How do I work with perspectives?

  Find out how to work with perspectives.
- How do I work with views and editors?

  Find out how to work with views and editors such as rearranging views and opening files for editing.
- How do I customize the workbench?

  Find out how to work to further customize the Eclipse workbench such as rearranging the main toolbar and changing the placement of the tabs.

How can I change the WebSphere Message Broker Toolkit preferences?

You can change a number of settings in the WebSphere Message Broker Toolkit to suit your requirements. Use these links to get more information about these settings and how to change them.

- What WebSphere Message Broker Toolkit preferences can I change?

  Use this link to find out about WebSphere Message Broker Toolkit preferences that you can change, and get links to information about preferences that might be useful when you develop message flow applications.
- How do I change WebSphere Message Broker Explorer preferences?

  Follow this link to get an overview of how you change preferences to alter the behavior of the WebSphere Message Broker Explorer.
- How do I change the standard Eclipse preferences?

  Follow this link to find out about the standard preferences in the Eclipse workbench.

How can I check that my installation was successful?

New users: Use the links in this topic to help you to check that your installation of WebSphere Message Broker was successful.

Most links in this topic work only if you are accessing this information center from the WebSphere Message Broker Toolkit or the WebSphere Message Broker Explorer.

- "Creating the default configuration" on page 564

  Use the Default Configuration wizard from the WebSphere Message Broker Toolkit or the WebSphere Message Broker Explorer to create a broker to run the WebSphere Message Broker samples. The default configuration also provides a useful environment to test and debug your own message flow applications. You can run a wizard to remove the default configuration when you have finished with it.
- Where can I find samples?

  The WebSphere Message Broker samples are a set of sample message flow applications and associated resources that demonstrate some of the capabilities of WebSphere Message Broker. Use the Where can I find samples? link to view a description of all the samples. You must create the default configuration before you can run the samples.

  You can use the following samples to quickly check that your installation was successful:

  – Pager

  – Scribble

  You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

  The Starter Edition and Entry Edition modes of operation restrict you to a single execution group. Also, the Entry Edition mode restricts the set of nodes you can use. As a result, not all samples can be used in these modes of operation. For more information, see "Restrictions that apply in each operation mode" on page 3657.

For more information about how to check that your installation of the WebSphere Message Broker or the WebSphere Message Broker Explorer was successful, see "Verifying your WebSphere Message Broker installation" on page 290. This topic describes how to verify your installation on Linux on x86, Linux on x86-64, or Windows by using either the WebSphere Message Broker Toolkit or the WebSphere Message Broker Explorer.

What information is available for users of previous versions?

If you are a user of a previous version of WebSphere Message Broker, use the links in this topic to get information about what is new in WebSphere Message Broker Version 7.0, and how to migrate from previous versions.

What's new?

Use the links in this section to find out about the main enhancements and changes in this release. You can find out more about the new functions and capabilities of this release by looking at the new WebSphere Message Broker Version 7.0 samples.

- "What's new in Version 7.0?" on page 7

  Find out about the main new function in this release, and new features added at Version 7.0.
- Where can I find sample applications?

  Look at the new samples introduced in Version 7.0 that demonstrate WebSphere Message Broker capabilities. Follow this link to get a description of all the samples and a direct link to each one. You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

What do I need to know about migration and coexistence?

Use the links in this section to find out how to migrate to WebSphere Message Broker Version 7.0 from previous versions of the product.

- How does WebSphere Message Broker Version 7.0 coexist with previous versions?

  WebSphere Message Broker Version 7.0 can coexist with previous versions. You can therefore perform a staged migration when you want to migrate from a previous version. See this topic for details about coexistence on a single computer.
- How do I migrate from Version 6.1 products?

  If you are migrating from Version 6.1 products, see this topic for information about planning for migration, and the steps involved in carrying out a migration from your specific product.
- How do I migrate from Version 6.0 products?

  If you are migrating from Version 6.0 products, see this topic for information about planning for migration, and the steps involved in carrying out a migration from your specific product.

How do I plan and configure brokers?

New users: Use the links in this topic to find out how to plan and configure brokers.

You can create a broker for testing purposes on your WebSphere Message Broker Toolkit computer by using the Default Configuration wizard. When you are ready

to create brokers for testing on platforms other than Windows or Linux on x86, or for production purposes, you must plan your configuration carefully, by using the information provided in the WebSphere Message Broker information center.

What do I need to know about brokers?

Use the links in this section to find information that you might find useful before planning and designing your brokers.

- What is a broker?

  A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in-flight messages. Follow this link to get information about brokers, and the resources that you need to create and configure your brokers.

- What is an operation mode?

  The operation mode that you use is determined by the license that you purchased when you bought the product. The mode can either give you the full operating capabilities of your broker, that is, `enterprise` mode, or restrict the operating capabilities of your broker, that is, `starter`, `adapter`, or `entry` modes.

  You can set the operation mode when you create a broker, however, brokers are created, or migrated from previous versions, in `enterprise` mode by default. Follow this link to get information about the various modes that are available.

- What do I need to know about databases?

  You can configure your message flows to access data in databases; you can read from, write to, and update the information in a supported database. Follow this link to find out more about databases and how to create connections to them.

What do I need to know about authorization and security?

You must set up access and authorization credentials for some of the resources that you use with WebSphere Message Broker; for example, databases. Follow the links in this section to get information about authorization, access control, and security for your broker.

- What authorizations are needed for configuration tasks?

  Authorization is the process of granting or denying access to a system resource. Use this link to get examples of the tasks in WebSphere Message Broker that require authorization. Use the following links to get more details about the security requirements for administrative tasks on the different platforms:

  - "Security requirements for Linux and UNIX platforms" on page 3648
  - "Security requirements for Windows systems" on page 3651
  - "Security requirements for z/OS" on page 3655

- What authorizations are required for access to runtime resources?

  If you want to set up administrative authority for the resources managed by the broker (including the broker itself), you must set up WebSphere MQ authorizations for brokers and execution groups.

- Where can I find out about security for publish/subscribe?

  Follow this link to get an overview of publish/subscribe security.

How do I plan brokers and configure prerequisites?

Use the following links to find out how to plan your brokers. You can also find out about tasks you must complete before you can configure your brokers.

- How do I plan brokers?

Follow this link to find out what you must consider when you plan your brokers, and your WebSphere MQ infrastructure.

- How do I set up broker security?

  Follow this link for information about how to set up security for your brokers. Each link comprises a list of reminders or questions about the security tasks to consider for your broker configuration. The answers to the questions provide the security information that you need to configure your brokers, and also give you further information about other security controls that you might want to deploy.

How do I create and configure brokers?

You can create one or more brokers on one or more computers. The use of multiple brokers can help you to spread the load of the message processing that you want to achieve, or to allocate different functions to different brokers or computers. For example, you might set up a broker to run personnel applications, and another to run financial or order systems. To support each broker, you must create a WebSphere MQ queue manager, and you might need to create a WebSphere MQ infrastructure of channels or a WebSphere MQ cluster to connect the brokers together.

You might also create several execution groups on each broker to manage the message flow applications that you deploy to the broker.

To create a broker, complete the following tasks in order:

1. How do I create a broker?

   On Windows and Linux on x86, you can create brokers in the WebSphere Message Broker Toolkit. On all platforms, you can create the brokers by using the command-line commands. Create as many brokers as you require.
2. How do I verify that the broker has been created successfully?

   You can use the **mqsilist** command to check that your brokers were created successfully
3. How do I start a broker?

   Follow this link to get instructions on how to start a broker.

How do I administer and monitor brokers?

New users: Use the links in this topic to get information about administering brokers.

How do I administer my brokers?

Use the links in this section to find out about the tasks that you operate frequently to activate and run your brokers.

- How do I start and stop a broker?

  Use this link to find out how to start and stop a broker on the different platforms. You can stop and start a broker by using the command line.
- How do I connect to a broker?

  Use the following links to find out how to connect to your broker:
  - Using the WebSphere Message Broker Toolkit or the WebSphere Message Broker Explorer
  - Using the Administration API (CMP API)
  - On z/OS
- How do I start and stop message flows?

Use the following links to find out how to start and stop message flows:
  - Using the WebSphere Message Broker Explorer in stand-alone mode (link to follow)
  - Start and Stop message flows using the command line
  - Using the CMP API
- How do I deploy a broker archive file?

  Use this link to find out how to deploy a broker archive file to a broker by using the WebSphere Message Broker Toolkit, the WebSphere Message Broker Explorer, the **mqsideploy** command, and the CMP API.
- How do I submit batch requests using the CMP API?

  You can use the CMP API to group multiple requests destined for the same broker together, and submit them as a single unit of work.
- How do I modify my broker?

  You might want to change the configuration of your broker. See How do I modify a broker? and follow the link for your broker platform.
- How do I delete my broker?

  You might want to delete your broker. See How do I delete a broker? for a link to your platform.
- How do I back up my resources?

  Follow this link to find out how to back up your broker and application resources.

How can I monitor my brokers?

Use the links in this section to find out how you can monitor the events and performance of your brokers.

- How can I view administration log information?

  The "Administration Log view" on page 6840 in the WebSphere Message Broker Toolkit contains information about events that occur during operation. These events can be information, errors, or warnings, and relate to your own actions.
- How can I view message flow accounting and statistics data?

  Message flow accounting and statistics data is the information that can be collected by a broker to record performance and operating details of message flow execution. Follow this link to get an overview of message flow accounting and statistics, and links to further information.
- What are the system management interfaces?

  The brokers provide a service for independent system management agents so that a central management facility can access information about a network that includes one or more brokers. Use this link to find out how you can use the system management publications from WebSphere Message Broker to develop or use system management adapters or customized administrative applications to receive information about broker activity.

How do I deploy and configure message flow applications?

New users: Use the links in this topic to get information about deploying and configuring message flow applications and related resources.

What do I need to know about deployment?

Deployment is the process of transferring data to an execution group that belongs to a broker, so that it can take effect in the broker. Use the following links to find out more about deployment.

- Where can I get an overview of deployment?

  Use this link to find out about the WebSphere Message Broker Toolkit, the WebSphere Message Broker Explorer, the **mqsideploy** command, and the Administration API (also known as the CMP API) as environments for deployment.
- What do I need to know about message flow application deployment?

  Use this link to find out more about deploying message flow applications and related resources.
- What is a broker archive?

  You add message flow applications and related resources to a broker archive to deploy the resources to an execution group. Use this link to find out about the files that can be associated with a broker archive file.
- What are configurable properties of broker archives?

  System objects that are defined in message flows can have configurable properties that you can update within the broker archive file before deployment. You can use configurable properties to update target-dependent properties, such as queue names, queue manager names, and database connections.
- How do I view version and keyword information for deployable resources?

  Follow this link to find out how to view the version and keyword information of deployable objects. You can use version and keyword information to track resources that have been deployed to a broker.

How do I deploy message flow applications?

You must create, configure, and start at least one broker to deploy message flow applications to an execution group. Use the links in this section to find out how to deploy your message flow applications and related resources to an execution group.

- **How do I add an execution group?**

  An execution group is a named grouping of message flows that have been assigned to a broker. The broker enforces a degree of isolation between message flows in distinct execution groups by ensuring that they run in separate address spaces, or as unique processes. You can create more execution groups in a number of different ways:

  – Using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer
  – Using the command line
  – Using the CMP API
- How can I create a broker archive?

  A broker archive file (BAR file) is a compressed file that contains compiled message flows, message dictionaries, and other development resources. Use this link to find out how to create a broker archive. You can use the “Broker Archive editor” on page 6794 to add or remove message flows and message sets to or from your broker archive, and to edit the configurable properties within your BAR file.
- How can I add resources to a broker archive?

You can add message flows and message sets to a broker archive file. The message flows, message sets, and related resources are compiled when they are added to the broker archive file. You can also choose to add the source files to the broker archive file. You cannot add resources to a broker archive file from a project that contains an error.

- How can I deploy a broker archive file?

  Find out about the different ways that you can deploy a broker archive file to a broker. For deploying message flow applications during development to a test environment, the simplest method is to use the WebSphere Message Broker Toolkit. When you deploy to a broker, you select an execution group to deploy to.

- How do I check the results of deployment?

  Use this link to find out how you can check the results of a deployment.

- How do I refresh the contents of a broker archive?

  You can refresh the contents of a broker archive by removing resources from it and, having made required changes, add them back again. Alternatively, you can use the Refresh option in the Broker Archive editor.

How can I configure message flow applications?

Use the links in this section to find out how to configure your message flow applications for use in a production environment.

- How do I edit configurable properties?

  System objects that are defined in message flows can have configurable properties that you can update within the broker archive file before deployment. By changing configurable properties, you can customize a broker archive file for a new broker, without needing to edit and rebuild the message flows or other resources.

- How do I configure message flows at deployment time using user-defined properties?

  User-defined properties (UDPs) give you the opportunity to configure message flows at deployment time, without modifying program code.

- How can I optimize message flow throughput?

  Use this link to find out how you can optimize the throughput of messages in your message flows using parallel processing.

- How do I add multiple instances of a message flow to a broker archive?

  You can edit the name of your files in the broker archive so that you can deploy multiple instances of a message flow with different values for the configurable properties.

- How do I configure global coordination of transactions?

  To ensure data integrity during transactions, you can globally coordinate message flow transactions with a WebSphere MQ queue manager. Follow this link to find out how to configure your databases, queue managers, and message flow applications for global coordination.

How can I diagnose problems?

New users: Use the links in this topic to find out how to debug message flow applications and get help with troubleshooting problems.

How can I debug and troubleshoot my message flow applications?

A number of tools and sources of information are available to help you solve problems with your message flow applications. If you can see errors or warnings in the Problems view in the WebSphere Message Broker Toolkit, your message flow applications and related resources might have problems that prevent you from adding them to a broker archive file. Also, your message flow applications might not work as you expect when they are deployed.

Use the information in the error or warning messages to fix the problem. If you have difficulty deploying message flow application resources, see **How can I troubleshoot problems when deploying message flow applications?** in this topic for more information. If you have successfully deployed your message flow applications, you can use the following links to debugging and troubleshooting topics to help you solve any problems.

- How do I use the flow debugger?

  You can test and debug your message flow applications by using the debugger in the WebSphere Message Broker Toolkit. By adding breakpoints to your message flows, you can use the debugger to view the path that messages take through your message flow. You can also view the contents of the message passing through the flow, and step through code to view the output messages being constructed. The debugger is launched from the "Debug perspective" on page 6789.
- Where can I find useful logs?

  Find out where to find useful logs that are used in WebSphere Message Broker.
- How do I use trace?

  User trace can be used to get further information about problems with messages passing through your message flows. You can use the following sequence of tasks to get user trace to troubleshoot a problem with a message flow:

  1. "Starting user trace" on page 3197
  2. "Retrieving user trace" on page 3204
  3. "Formatting trace" on page 3543
  4. "Interpreting trace" on page 3546
- How can I resolve problems when developing message flows?

  Follow this link for information about some common problems when developing message flows.
- How can I resolve problems when developing message models?

  Follow this link for information about some common problems when developing message models.

How can I troubleshoot problems when deploying message flow applications?

If your message flow applications or related resources contain an error, they cannot be added to a broker archive file. Sometimes message flow applications that are successfully added to a broker archive file can fail to be deployed, for example, because of a problem with the broker or with the message flow applications. Use the following links to find out how to check the results of deployment and get help solving problems.

- How can I check the results of deployment?

  Find out how to check the results of a deployment by using the "Administration Log view" on page 6840, the **mqsideploy** command, and the Administration API (also known as the CMP API).
- How can I resolve problems when deploying message flows or message sets?

Follow this link to find tips on resolving problems when deploying message flows and message sets.

Where can I get other help on troubleshooting and solving problems?

Use the links in this section to find out more information about troubleshooting and how to get support.

- What initial checks can I make?

  Follow this link to get more in-depth advice on checks to make if you have problems with WebSphere Message Broker.
- How do I deal with problems?

  Use the links in this topic to find out about common problems that you might see when you use WebSphere Message Broker. Check to see if one of these problems match the symptoms you are observing.
- Where can I get more help with problems?

  View this topic for links to more techniques for diagnosing problems with WebSphere Message Broker, and links to information about getting support.

Samples

The WebSphere Message Broker Toolkit provides samples that show the features that are available in WebSphere Message Broker, and how to use them. This topic provides links to the information about the individual samples.

Use the samples to learn how to use WebSphere Message Broker.

- You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.
- Not all samples work in all modes. If you try to use a sample in a mode that is restricted, you receive a message indicating the reason; see "Restrictions that apply in each operation mode" on page 3657. You can run all samples in a development and unit test broker; see "Development and unit test" on page 50.
- You might receive the following error:

  ```
  SEVERE: Could not find the connection files.
  ```

  If you receive this error, copy the `.broker` file for your default broker from the runtime workspace in which you created your default configuration to your current runtime workspace.
- To ensure that all available samples are displayed in the **Samples and Tutorials** tab in the WebSphere Message Broker Toolkit, in the "More samples" panel in the **Samples and Tutorials** tab click **Retrieve** and select the sample categories that you want to display.

The samples are categorized as either Application or Technology samples.

Application samples

The Application samples are small end-to-end WebSphere Message Broker message flow applications that show how to transform and route messages through message flows.

- Application samples

Technology samples

The Technology samples are small WebSphere Message Broker message flow applications that each show a specific feature of WebSphere Message Broker.

- Controlling and Routing samples
- File Processing samples
- Industry samples
- Message Formats samples
- Message Transformation samples
- Monitoring samples
- Security samples
- Transports and Connectivity samples
- Web Service samples

Before you can use the samples you must create the Default Configuration, see "Creating the Default Configuration" on page 106.

The following table lists the Application samples that are available in WebSphere Message Broker.

Sample name	Description
Airline Reservations	This sample shows how to use a range of nodes, including nodes for aggregation, routing, tracing, filtering, and updating database tables.
Coordinated Request Reply	This sample shows how two applications with different message formats communicate with each other through the use of WebSphere MQ messages in a request-reply processing pattern, coordinated by the use of an MQGet node.
DatabaseInput Node	This sample demonstrates how to take data from a database, as it is being updated, and process the data within WebSphere Message Broker.
Data Warehouse	This sample contains a message flow that archives data, such as sales data, to a database.
Error Handler	This sample contains a message flow and a subflow to show error handling in message flow applications.
Large Messaging	This sample shows how to process messages that contain repeating structures, and how to minimize the virtual storage requirements for the message flow.
Message Routing	This sample shows how to use a message flow to route messages to different WebSphere MQ queues based on data stored in a database table or a file.
Pager	This sample shows simple point-to-point messaging and publish/subscribe messaging. Use graphical interfaces to send text messages to a pager application, or to subscribe to publications about the surf on selected beaches.
Scribble	This sample is a small, graphical whiteboard application on which you draw by using your mouse pointer. Depending on the options you choose, you can see the effects of message transformation by using WebSphere MQ transport or RealTime transport.

Sample name	Description
Solar Pattern Authoring	This sample shows how to build a WebSphere Message Broker pattern. The sample provides an example message flow project that calculates the sunrise and sunset times in a PHPCompute node. The sample also provides a pattern authoring project that configures a pattern.
User-defined Extension	These two samples show the use of user-defined nodes written in the C and Java programming languages.
Video Rental	This sample shows message transformation between three different formats: XML, Custom Wire Format (CWF), and Tagged/Delimited String (TDS) Format.
WBI JDBC Adapter Migration	This sample re-creates a scenario of migrating a JDBC adapter to invoke a message flow, by using an MQInput node with the built-in DatabaseInput node.
Web Service Aggregation	This sample demonstrates how to invoke a number of web services and amalgamate the results by using WebSphere Message Broker aggregation nodes. The sample illustrates how aggregation can be used for transports other than WebSphere MQ, and highlights issues to be aware of. The sample also shows how you can use message flow monitoring to audit data across the aggregation fan-out and fan-in by using Collector node techniques.

The following tables list the Technology samples that are available in WebSphere Message Broker.

Control and Routing samples

Sample name	Description
Aggregation	This sample shows how to use the Aggregation nodes to perform a basic four-way aggregation operation, with simple fan-out and fan-in message flows.
Collector Node	This sample shows how to configure the Collector node to gather input from different input sources. It also shows some alternative methods for completing collections.
Simplified Database Routing	This sample shows how to use the following range of simplified (non-programming) message flow nodes: Route, DatabaseRoute, and DatabaseRetrieve. The sample illustrates how to access databases by using JDBC, and how to use values held in an acquired result set, gathered from a database query, to either dynamically route messages or update the content of the messages.
Timeout Processing	This sample shows how to use the timeout nodes to add timeouts to message flows.

File Processing samples

Sample name	Description
Batch Processing	This sample shows how to use the FileInput and FileOutput nodes to read different input files and append them to one output file. It also shows you how to read a file "as is" from one local input directory and write the file to a different local output directory.
File Output	This sample shows how a FileOutput node can write a message to a file during a message flow. The sample shows a flow updating a SOAP message, the FileOutput node writing this updated message to a file, and the message being sent back to the sender by using HTTP as the transport mechanism.
Managed File Transfer	This sample shows how the FTEOutput node writes a message to a file by using WebSphere MQ File Transfer Edition to manage the transfer of the file. The sample shows a Retail HQ to Branch product provisioning scenario. The flow receives a stream of WebSphere MQ messages with product data, and a file is created with an FTEOutput node. The built-in FTEAgent operates with the FTEOutput node to manage the transfer of the file to a remote location.
WildcardMatch	This sample shows how to access the LocalEnvironment.Wildcard.WildcardMatch variable set in the FileInput node. The sample then gives an example of how, by using this variable, you can dynamically override the output file name and directory properties set in the FileOutput node.

Industry samples

Sample name	Description
Healthcare	This sample is a development accelerator, decreasing the time an integration developer within the healthcare industry requires to deliver integration solutions. This sample provides a number of Healthcare assets that solve key healthcare-specific integration problems.
TLOG Processor	This sample is a set of customized message sets, subflows, message flows, and style sheets that process transaction log (TLOG) data from retail stores. These samples provide sample TLOG input messages that are generated by various IBM Retail applications to test the TLOG message flows. Customers and service teams can customize or extend the TLOG Processor samples by modifying the message sets and flows, or by recombining these components in alternative ways.

Message Formats samples

Sample name	Description
Comma Separated Value (CSV)	This sample shows how to model common CSV message variants, and how to transform the sample CSV messages to and from XML.
EDIFACT	This sample shows an industry-standard message set for Edifact messages.

Sample name	Description
FIX	This sample shows an industry-standard message set for Fix messages.
SWIFT	This sample shows an industry-standard message set for Swift messages.
X12	This sample shows an industry-standard message set for X12 messages.
XMLNSC Namespaces	This sample shows how to change the existing namespace of a message, remove a namespace from a message, and add a namespace to a message.
XMLNSC Validation	This sample shows the capability of the XMLNSC parser to validate messages against an XML schema.

Message Transformation samples

Sample name	Description
JavaCompute Node	This sample shows how to use the JavaCompute node to perform tasks such as calling an external service and propagating a new message based on the results of the call.
Message Map	This sample shows how to author message maps in the WebSphere Message Broker Toolkit.
PHPCompute Node	This sample shows how to use the PHPCompute node in a message flow to transform an XML message.
XSL Transform	This sample shows how to use a message flow to transform an XML message to another form of XML message according to the rules provided by an XSL stylesheet.

Monitoring samples

Sample name	Description
WebSphere Business Monitor	This sample provides resources to help you use the monitoring events produced by your message flows for business process modeling by using WebSphere Business Monitor.

Security samples

Sample name	Description
Security Identity Propagation	This sample shows how to use Identity Security features to extract the security credentials from the messages on the MQInput and HTTPInput nodes. So that the sample can run stand-alone, the sample does not include security validation with an external security provider system, such as LDAP or TFIM. The sample also shows how to manipulate the security credentials by using ESQL, then how to propagate the identity to the MQOutput and HTTPRequest nodes.
Security Policy Enforcement Point (PEP)	This sample demonstrates how to use the SecurityPEP node as the Policy Enforcement Point in a message flow.

Transports and Connectivity samples

Sample name	Description
Browsing WebSphere MQ Queues	This sample shows how a message flow can browse WebSphere MQ messages that are in a queue, therefore retrieving the messages non-destructively. This sample also shows how to examine the contents of the browsed message to determine whether to get the message. Getting the message is a destructive process that removes the message from the queue.
CICS Transaction Server for z/OS Connectivity	This sample is based on a scenario in which a business wants to retrieve a record from a file resource on CICS Transaction Server for z/OS. The sample demonstrates how to use the CICSRequest node. With this node, you can run CICS applications, and retrieve data from CICS regions.
CICS Transaction Server for z/OS Channel Connectivity	This sample demonstrates how to call a channel-based CICS program. A CICS channel structure can be represented in WebSphere Message Broker by a message collection. This sample demonstrates how to create and populate a message collection for the CICSRequest node and how to process the collection after the call.
CORBA nodes	This sample is based on a product warehouse scenario where a stock administrator wants to manage a stock control management system hosted on a CORBA server. The sample demonstrates how to use the CORBARequest node to invoke CORBA server applications.
Email	This sample consists of three message flow applications that show the use of sending and receiving emails. The emailform message flow provides an HTML input form to construct and submit an email message. The sendemail message flow receives the message and processes all the details that are associated with the email message. The recipients specified on the form receive the message as an email in the appropriate format, with any attachments. The getemail message flow processes the email that is sent and filters the email either to a WebSphere MQ queue, or saves the attachment to a file by using the FileOutput node.
HTTPHeader node	This sample consists of three message flows that show the different ways in which you can use an HTTPHeader node. The three message flows are: • Single WebService in MQ flow sample. This sample message flow shows how to create an interface between a WebSphere MQ application and web-based applications by using the HTTPHeader and MQHeader nodes. • Multiple WebService requests sample. This sample shows how to create and reset HTTP headers by using the HTTPHeader node. • Set Cookie HTTP reply sample. This sample shows how to add an HTTPReply header by using the HTTPHeader node in a request-reply session.

Sample name	Description
IMS Synchronous Request	This sample shows how to call an IBM Information Management System (IMS) transaction synchronously from within a message flow. The sample uses the IMSRequest node to make the synchronous calls by using IMS Connect. This sample uses the IMS sample transaction DSPALLI (Display All Invoices), which is typically available on all IMS systems. The DSPALLI transaction can call a REXX or COBOL program, although REXX is the default that is typically installed on IMS.
JD Edwards Connectivity	This sample consists of a message flow application that demonstrates the use of the JDEdwardsRequest node. This sample message flow uses the JD Edwards business function call "retrieve" to fetch a record from a JD Edwards EnterpriseOne server. The record is then put on a WebSphere MQ queue.
JMS Nodes	This sample shows how to use the JMS nodes as a JMS Consumer and Producer to an external JMS provider.
JMSHeader node	This sample shows how to use the JMSHeader node in a JMS coordinated request-reply scenario.
MQHeader node	This message flow sample shows how to use the MQHeader node to add and remove an MQMD header.
SAP callout to a synchronous system	This sample consists of a single message flow application that demonstrates the use of the SAPInput node with the SAPReply node to enable a message flow to act as a synchronous BAPI. The message flow is used to service requests for four different BAPIs that create, update, retrieve, and delete customer details.
SAP callout to an asynchronous system	This sample consists of three message flow applications that demonstrate the use of the SAPInput node with the SAPReply node to enable a message flow to act as a synchronous BAPI that wraps an asynchronous application. The message flows are used to service requests for four different BAPIs that create, update, retrieve, and delete customer details.
SAP Connectivity	This sample consists of two message flow applications that show the use of the SAPInput node and the SAPRequest node. The SAPInput node scenario shows how to use a message flow to receive IDocs from the SAP Material Master, then send the data to a WebSphere MQ output queue for processing by another message flow or application. The SAPRequest node scenario shows how to use a message flow to create a customer in SAP, then update and retrieve the customer details.

Sample name	Description
SCA nodes	This sample shows how to use the SCAInput, SCAReply, SCAAsyncRequest, and SCAAsyncResponse nodes to exchange message requests and responses with a business process in WebSphere Process Server. The sample re-creates a scenario in which a savings account is linked to a current account, and money can be transferred between the two accounts. In the outbound scenario, WebSphere Message Broker passes money transfer requests to WebSphere Process Server, which hosts the savings account. This sample can be extended to include an inbound scenario where WebSphere Process Server passes requests onto WebSphere Message Broker, which hosts the current account.
TCPIP Client Nodes	This sample consists of three message flows that show both synchronous and asynchronous communication from the WebSphere Message Broker to a TCP/IP server. It also includes a simple message flow to simulate the TCP/IP server.
TCPIP Handshake	This sample shows how to implement an application-level handshake protocol for a synchronous request reply model of communication between a client and a server. The sample also includes two other message flows to emulate the client and server applications. You can replace these applications by external applications which use the same interfaces.
Twineball Example EIS Adapter	This sample shows how to use the WebSphere Adapter nodes by using the Twineball adapter, a self contained EIS, to synchronize a C system with an EIS.

Web Service samples

Sample name	Description
Address Book	This sample shows how to use the SOAPInput, SOAPReply, and SOAPRequest nodes to provide and consume a web service. Two sets of example input messages are provided: one set to call the consumer flow which in turn calls the provider flow, and one set to call the provider flow directly. This sample can also be extended to show how to set up WS-Security for existing message flows for both a provider and a consumer.
Asynchronous Consumer	This sample shows how to use the asynchronous SOAP nodes when you call a web service. The web service simulates an order service, and the client shows how existing WebSphere MQ interfaces can be extended to make web service requests.

Sample name	Description
RESTful Web Service Using JSON	This sample shows how to front an existing service as a RESTful web service providing a JSON message format interface. The sample also shows how to consume the RESTful web service from a message flow.
SOAP Nodes	This sample shows the use of SOAP nodes to both provide and consume a web service.
Web services using HTTP nodes	This sample shows how to use WebSphere Message Broker to front an existing application as a web service.
Web Services Gateway	This sample demonstrates how to use the SOAP nodes in a Web Services Gateway mode, which allows WebSphere Message Broker to handle generic SOAP request/response and one-way messages when used as a web services provider or consumer.
WebSphere Service Registry and Repository Connectivity	This sample shows how to retrieve documents by using the WebSphere Service Registry and Repository nodes. You can use these nodes to query Service Registry information, and to use this information at run time. You can also use these nodes to acquire WSDL or other generic descriptions of available services.

Related tasks:

"Creating the Default Configuration"
You can create the Default Configuration of WebSphere Message Broker by using the Default Configuration wizard. You can also remove the Default Configuration by using the link provided.

"Resolving problems when running samples" on page 3366
Use the advice given here to help you to resolve common problems that can arise when you run or remove samples.

Related reference:

"What the Default Configuration wizard creates" on page 107
A table of the components that are created by the wizard, details of how to resolve problems, and how to view errors.

Creating the Default Configuration

You can create the Default Configuration of WebSphere Message Broker by using the Default Configuration wizard. You can also remove the Default Configuration by using the link provided.

About this task

The Default Configuration wizard creates all the components that you need to import and deploy the WebSphere Message Broker samples, and to build your own samples. For more information about the Default Configuration wizard and the authorities your user account must have to successfully create the Default Configuration, see "What the Default Configuration wizard creates" on page 107.

You can use the following links only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the WebSphere Message Broker Explorer.

Procedure

Click the following link to start the Default Configuration wizard:

- Start the Default Configuration wizard

What to do next

When you have finished using the Default Configuration, you can remove it by clicking the following link:

- Remove the Default Configuration wizard

Related concepts:

"Samples" on page 98
The WebSphere Message Broker Toolkit provides samples that show the features that are available in WebSphere Message Broker, and how to use them. This topic provides links to the information about the individual samples.

Related tasks:

"Creating a default configuration" on page 564
Use the Default Configuration wizard to create and test a basic broker configuration.

Related reference:

"What the Default Configuration wizard creates"
A table of the components that are created by the wizard, details of how to resolve problems, and how to view errors.

What the Default Configuration wizard creates

A table of the components that are created by the wizard, details of how to resolve problems, and how to view errors.

Purpose

The components that are created by the Default Configuration wizard are listed in the following table.

Component	Name
broker	MB7BROKER
queue manager	MB7QMGR

- WebSphere Message Broker automatically scans for a free queue manager port starting at 2414.
- The HTTP listener port that is automatically used is 7080.

Target environment of the wizard

The target environment contains the default components that are created by the Default Configuration wizard.

```
Target Environment of Wizard
------------------------------------------------
Summarized resource updates listed below will be
applied to installation
[C:\Program Files\IBM\MQSI\7.0]
```

```
All actions are logged to file
[DefaultConfigurationWizard.log]
in the workspace directory
[C:\Documents and Settings\Administrator\IBM\wmbt70\
workspace\.metadata]

All actions are applied under account:
LocalSystem
Queue manager name: MB7QMGR
Queue manager port: 2414

Default broker details
Broker name: MB7BROKER
Queue manager name: MB7QMGR

HTTP listener port: 7080
```

Resolving problems when creating and running the Default Configuration

If you have problems when you run the Default Configuration wizard, consider whether the following issues apply to you:

- If you have any existing brokers that are using the default HTTP listener port of 7080, you must ensure that the existing brokers are stopped before you create and deploy to the Default Configuration. If a sample does not run as you expect, check the Event Log for errors such as BIP3144:

  ```
  BIP3144
  (MB8BROKER.HTTPListener) An error has occurred during HTTP
  listener startup:
  the specified TCPIP port ('7080') is already in use.

  The HTTP listener needs to bind to a TCPIP port for correct
  operation to be possible.
  The broker-specific TCPIP port number '7080' is in use by
  another application.

  Stop other applications from using the specified port,
  or change the broker-specific port.
  ```

- Linux On Linux: Ensure that your user account belongs to the mqbrkrs group.

Viewing errors

To view errors generated by the Default Configuration wizard look at the `DefaultConfigurationWizard.log` file:

- Windows On Windows: This file is, by default, in `C:\Documents and settings\`*`user_name`*`\IBM\wmbt70\workspace\.metadata\DefaultConfigurationWizard.log`
- Linux On Linux: This file is, by default, in `/home/`*`user_name`*`/IBM/wmbt70/workspace/.metadata/DefaultConfigurationWizard.log`

To view Eclipse errors that are caused by the Default Configuration wizard look at the Eclipse Error log, see “Viewing the Eclipse error log” on page 3532. The Error Log view opens in the perspective in which you are currently working.

Related concepts:

"Samples" on page 98
The WebSphere Message Broker Toolkit provides samples that show the features that are available in WebSphere Message Broker, and how to use them. This topic provides links to the information about the individual samples.

Related tasks:

"Creating a default configuration" on page 564
Use the Default Configuration wizard to create and test a basic broker configuration.

"Creating the Default Configuration" on page 106
You can create the Default Configuration of WebSphere Message Broker by using the Default Configuration wizard. You can also remove the Default Configuration by using the link provided.

"Viewing the Eclipse error log" on page 3532
The Eclipse error log captures internal errors that are caused by the operating system or your code.

Legal information for WebSphere Message Broker

Read the information that describes the legal statements for this product.

- Notices
- Trademarks

This information center provides links or references to non-IBM Web sites and resources. IBM makes no representations, warranties, or other commitments whatsoever about any non-IBM Web sites or third-party resources (including any Lenovo Web site) that may be referenced, accessible from, or linked to any IBM site. A link to a non-IBM Web site does not mean that IBM endorses the content or use of such Web site or its owner. In addition, IBM is not a party to or responsible for any transactions you may enter into with third parties, even if you learn of such parties (or use a link to such parties) from an IBM site. Accordingly, you acknowledge and agree that IBM is not responsible for the availability of such external sites or resources, and is not responsible or liable for any content, services, products, or other materials on or available from those sites or resources. When you access a non-IBM Web site, even one that may contain the IBM-logo, be aware that it is independent from IBM, and that IBM does not control the content on that Web site. It is up to you to take precautions to protect yourself from viruses, worms, trojan horses, and other potentially destructive programs, and to protect your information as you deem appropriate.

Notices for WebSphere Message Broker

Read the legal notices for WebSphere Message Broker.

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this information in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this information. The furnishing of this information does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the information. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this information at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM United Kingdom Laboratories,
Mail Point 151,
Hursley Park,
Winchester,
Hampshire,
England
SO21 2JN

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Programming License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information includes examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Privacy policy considerations

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is described below.

For WebSphere Message Broker, the Software Offering does not use cookies or other technologies to collect personally identifiable information.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at http://www.ibm.com/privacy and IBM's Online Privacy Statement at http://www.ibm.com/privacy/details/us/en

sections entitled "Cookies, Web Beacons and Other Technologies" and "Software Products and Software-as-a Service".

COPYRIGHT LICENSE

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

(C) (*your company name*) (*year*). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. *_enter the year or years_*. All rights reserved.

Trademarks in the WebSphere Message Broker Information Center

Review the trademark information for WebSphere Message Broker.

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at Copyright and trademark information at `www.ibm.com/legal/copytrade.shtml`.

Adobe is a registered trademark of Adobe Systems Incorporated in the United States and/or other countries.

Intel, Itanium, and Pentium are trademarks of Intel Corporation in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliate.

Other company, product, or service names may be trademarks or service marks of others.

Glossary of terms and abbreviations

This glossary defines WebSphere Message Broker terms and abbreviations that are used in this online information center.

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

A

active working set
The logical collection of application projects that is currently displayed in the Broker Application Development perspective. See also working set.

adapter
An intermediary software component that allows two other software components to communicate with one another.

Administration API (CMP)
An application programming interface that your applications can use to control brokers through a remote interface. For continuity and consistency, this API is referred to as the CMP, its name in previous versions.

aggregation
See message element aggregation.

AMI See Application Messaging Interface.

Application Messaging Interface (AMI)
The programming interface, provided by WebSphere MQ, that defines a high-level interface to message queuing services. See also Message Queue Interface (MQI) and Java Message Service (JMS). Applications that use the AMI connect to the broker using WebSphere MQ Enterprise Transport.

attribute
A characteristic or trait of an entity that describes the entity; for example, the telephone number of an employee is one of the employee attributes.

An attribute may have a type, which indicates the range of information given by the attribute, and a value, which is within that range. In XML, for example, an attribute consists of a name-value pair within a tagged element, that modifies features of the element.

attribute group
A set of attributes that can appear in a complex type.

TOP

B

BAR file
See broker archive file.

bend point
A point that is introduced in a connection between two message flow nodes at which the line that represents the connection changes direction. A bend point can be used to make node alignment and processing logic clearer and more effectively displayed.

binary large object (BLOB)
A block of bytes of data (for example, the body of a message) that has no discernible meaning, but is treated as one entity that cannot be interpreted.

BLOB See binary large object.

BLOB domain
The message domain that includes all messages with content that cannot be interpreted or subdivided into smaller sections of information. Messages in this domain are processed by the BLOB parser. See also DataObject domain, IDOC domain, JMS domain, MIME domain, MRM domain, SOAP domain, XML domain, XMLNS domain, and XMLNSC domain.

BLOB parser
A program that interprets a message that belongs to the BLOB domain, and generates the corresponding tree from the bit stream on input, or the bit stream from the tree on output.

broker
A set of execution processes that host one or more message flows. Also known as a message broker.

broker archive file
The unit of deployment to the broker; also known as a BAR file. The broker archive file contains a number of different files, including compiled message flows (`.cmf`) and message sets (`.dictionary` and `.xsdzip` files), that are used by the broker at run time. It can also contain additional user-provided files that your message flows might need at run time, if the file extension does not overlap with extensions that are used by the broker.

broker schema
A symbol space that defines the scope of uniqueness of the names of resources that are defined within it. The resources include message flows, ESQL files, and mapping files.

built-in node
A message flow node that is supplied by the product. Some of the supplied nodes provide basic processing such as input and output.

built-in pattern
A pattern that covers a set of commonly encountered message flow scenarios and that is packaged and released with WebSphere Message Broker.

business component
This term is specific to the WebSphere Adapters. A component that defines the structure, behavior, and information that is displayed by a particular subject, such as a product, contact, or account, in Siebel Business Applications.

business graph
This term is specific to the WebSphere Adapters. A wrapper that is added around a simple business object or a hierarchy of business objects to provide additional capabilities. For example, a wrapper might carry change summary and event summary information related to the business objects in the business graph. See also business object.

business object
A software entity that represents a business entity, such as an invoice. A business object includes persistent and nonpersistent attributes, actions that can be performed on the business object, and rules that the business object is governed by. See also business graph.

TOP

C

callback function
See implementation function.

cardinality
See mapping cardinality.

certificate authority
A trusted third-party organization or company that issues the digital certificates. The certificate authority typically verifies the identity of the individuals who are granted the unique certificate.

cmf See compiled message flow.

compiled message flow (cmf)
A message flow that has been compiled to prepare it for deployment to the broker. A `cmf` file is sent to the broker within a BAR file.

complex element
A named structure that contains simple elements within the message. Complex elements can contain other complex elements, and can also contain groups. The content of a complex element is defined by a complex type. See also simple element.

complex type
A type that can contain elements, attributes, and groups organized into a hierarchy.

A complex type structure within a message contains elements, attributes, and groups organized into a hierarchy. See also simple type.

component
A set of processes that perform a specific set of functions. WebSphere Message Broker consists of two components, the broker and the WebSphere Message Broker Toolkit.

component directory
In z/OS, the root directory of the component's runtime environment.

component name
The external name of a component. Each component requires a name, which is used, for example, in the WebSphere Message Broker Toolkit and in commands.

component PDSE
In a z/OS environment, a PDSE that contains jobs to define resources to WebSphere MQ, DB2, and the broker started task. See partitioned data set.

connection
See message flow node connection. For broker-to-broker connections, see publish/subscribe topology.

connection factory
A set of configuration values that produces connections that enable a Java EE component to access a resource. Connection factories provide on-demand connections from an application to an enterprise information system (EIS).

content-based filter
In publish/subscribe, an expression that is included as part of a subscription to determine whether a publication message is received based on its content. The expression can include wildcards.

Custom Wire Format (CWF)
The physical representation of a message in the MRM domain that is composed of a number of fixed-format data structures or elements. In CWF, these structures are not separated by delimiter characters.

CWF See Custom Wire Format.

TOP

D

data element separation
For a complex type, defines to the MRM parser TDS physical format which method of identifying data elements is to be used, and how the data elements are constructed. The following separation types are supported: data pattern separation, delimited separation, fixed-length separation, and tagged separation

DataFlowEngine (DFE)
See execution group.

datagram
A form of asynchronous messaging in which an application sends a message, but does not want a response. Also known as send-and-forget. Contrast with request/reply.

DataObject domain
The message domain that includes all messages that are exchanged between the broker and enterprise information system applications such as SAP, PeopleSoft, and Siebel. Messages in this domain are processed by the DataObject parser. You must create a message model for messages that you process in this domain. See also BLOB domain, IDOC domain, JMS domain, MIME domain, MRM domain, SOAP domain, XML domain, XMLNS domain, and XMLNSC domain.

DataObject parser
A program that interprets a message that belongs to the DataObject domain, and generates the corresponding tree from the business object on input, or the business object from the tree on output.

debugger
See flow debugger.

deploy
To place files or install software into an operational environment.

The process of transferring data to an execution group on a broker.

For deploying message flows and associated resources, the data is packaged in a broker archive (BAR) file before being sent to the broker, which unpacks and stores the data.

destination list
See local environment.

digital certificate
An electronic document used to identify an individual, a system, a server, a company, or some other entity, and to associate a public key with the entity. A digital certificate is issued by a certificate authority and is digitally signed by that authority.

digital signature
Information that is encrypted with a private key and is appended to a message or object to assure the recipient of the authenticity and integrity of the message or object. The digital signature proves that the message or object was signed by the entity that owns, or has access to, the private key or shared-secret symmetric key.

distribution list
A list of WebSphere MQ queues to which a message can be put with a single statement.

document type definition (DTD)
The rules that specify the structure for a particular class of SGML or XML documents. The DTD defines the structure with elements, attributes, and notations, and it establishes constraints for how each item can be used within the particular class of documents. A DTD is analogous to a database schema in that the DTD completely describes the structure for a particular markup language.

DOM See Document Object Model.

DTD See document type definition.

TOP

E

editor area
The area in the WebSphere Message Broker Toolkit where files are opened for editing.

EIS See Enterprise Information System.

element
A named piece of information, or a field, within a message, with a business meaning agreed by the applications that create and process the message. See also simple element and complex element.

embedded message
See multipart message.

EMD See Enterprise Metadata Discovery.

endpoint
A JCA application or other client consumer of an event from the enterprise information system.

Enterprise Information System (EIS)
The applications that comprise an existing enterprise system for handling company-wide information. An enterprise information system offers a well-defined set of services that are exposed as local or remote interfaces or both. (Sun)

Enterprise Metadata Discovery (EMD)
A specification that defines how you can examine an Enterprise Information System (EIS) and get details of business object data structures and APIs. An EMD stores the definitions as XML schemas by default, and builds components that can access the EIS.

environment
A structure within the message tree that is user-defined and can contain variable information that is associated with a message while it is being processed by a message flow.

ESM See external security manager.

ESQL See Extended SQL.

ESQL data type
A characteristic of an item of data that determines how that data is processed. ESQL supports six data types (Boolean, datetime, null, numeric, reference, and string). Data that is retrieved from a database or is defined in a message model is mapped to one of these basic ESQL types when it is processed in ESQL expressions.

ESQL field reference
A sequence of values, separated by periods, that identify a specific field (which might be a structure) within a message tree or a database table. An example of a field reference is `Body.Invoice.InvoiceNo`.

ESQL function
A single ESQL expression that calculates a resultant value from a number of specified input values. The function can take input parameters but has no output parameters; it returns to the caller the value that results from the implementation of the expression. The ESQL expression can be a compound expression such as BEGIN END.

ESQL module
A sequence of declarations that define MODULE-scope variables and their initialization, and a sequence of subroutine (function and procedure) declarations that define a specific behavior for a message flow node. A module must begin with the `CREATE node_type MODULE` statement and end with an `END MODULE` statement. The `node_type` must be one of Compute, Database, or Filter. The entry point of the ESQL code is the module scope procedure named MAIN.

ESQL procedure
A subroutine that has no return value. It can accept input parameters from and return output parameters to the caller.

ESQL variable
A local temporary field that is used to assist in the processing of a message.

event A change to a state, such as the completion or failure of an operation, business process, or human task, that can trigger a subsequent action, such as persisting the event data to a data repository or invoking another business process.

event store
A persistent cache where event records are saved until a polling adapter can process them.

exception list
A list of exceptions, with supporting information, that has been generated during the processing of a message.

execution group
A named grouping of message flows that have been assigned to a broker.

The broker enforces a degree of isolation between message flows in distinct execution groups by ensuring that they execute in separate address spaces, or as unique processes.

An execution group process is also known as a DataFlowEngine (DFE). This term is typically used in problem determination scenarios; for example, trace contents or diagnostic messages). A DFE is created as an operating system process, and has a one-to-one relationship with the named execution group. If more than one message flow runs within an execution group, multiple threads are created within the DFE process.

exemplar
A project that contributes most of its content to a pattern. An exemplar contains message flows and other resources, such as source code.

Extended SQL (ESQL)
A specialized set of SQL functions and statements that are based on regular SQL, and extended with functions and statements that are unique to WebSphere Message Broker.

Extensible Markup Language (XML)
A standard metalanguage for defining markup languages that is based on Standard Generalized Markup Language (SGML).

Extensible Stylesheet Language (XSL)
A language for specifying style sheets for XML documents. Extensible Stylesheet Language Transformation (XSLT) is used with XSL to describe how an XML document is transformed into another document.

External Security Manager (ESM)
In a z/OS environment, a security product that performs security checking on users and resources. RACF is an example of an ESM.

TOP

F

field reference
See ESQL field reference.

file splitting
The division of an event file, based on a delimiter or based on size, to separate individual business objects within the file and send them as if they are each an event file to reduce memory requirements.

filter An ESQL expression that is applied to the content of a message to determine whether the message matches certain criteria.

For example, a Filter node uses a filter to determine how a message is to be processed. Alternatively, a filter is applied to the content of a publication message to determine whether it is to be passed to a subscriber.

flow debugger
A facility to debug message flows that is provided in the Debug perspective in the WebSphere Message Broker Toolkit.

TOP

G

graphical user interface (GUI)
A type of computer interface that presents a visual metaphor of a real-world scene, often of a desktop. A GUI combines high-resolution graphics, pointing devices, menu bars and other menus, overlapping windows, icons, and the object-action relationship.

GUI See graphical user interface.

TOP

I

IBM Runtime Environment for Java
A subset of the IBM Developer Kit for the Java Platform that contains the core executable files and other files that constitute the standard Java platform. The IBM Runtime Environment includes the Java virtual machine (JVM), core classes, and supporting files.

IBM Software Developer Kit for Java
A software package that can be used to write, compile, debug, and run Java applets and applications.

IDOC domain
The message domain that includes all messages that are exchanged between the broker and SAP R3 clients by the MQSeries® link for R/3. Messages in this domain are processed by the IDOC parser. See also BLOB domain, DataObject domain, JMS domain, MIME domain, MRM domain, SOAP domain, XML domain, XMLNS domain, and XMLNSC domain.

The IDOC domain is deprecated; use the MRM domain for new messages.

IDOC parser
A program that interprets a message that belongs to the IDOC domain, and generates the corresponding tree from the bit stream on input, or the bit stream from the tree on output.

implementation function
A function written for a user-defined node or message parser; also known as a callback function.

inbound processing
The process by which changes to business information in an enterprise information system (EIS) are detected, processed, and delivered to a runtime environment by a JCA Adapter. An adapter can detect EIS changes by polling an event table or by using an event listener.

input node
A message flow node that represents a source of messages for a message flow or subflow. See also output node.

installation directory
In a z/OS environment, a file system into which all product data is installed, and from which it is referenced and retrieved during the customization phase.

TOP

J

J2EE See Java 2 platform, Enterprise Edition.

Java 2 Platform, Enterprise Edition (J2EE)
An environment for developing and deploying enterprise applications, defined by Sun Microsystems Inc. The Java EE platform consists of a set of services, application programming interfaces (APIs), and protocols that provide the functionality for developing multi-tiered, Web-based applications. (Sun)

Java Database Connectivity (JDBC)
An industry standard for database-independent connectivity between the Java platform and a wide range of databases. The JDBC interface provides a call-level API for SQL-based and XQuery-based database access. See also Open Database Connectivity.

Java EE Connector Architecture (JCA)
A standard architecture for connecting the J2EE platform to heterogeneous enterprise information systems (EIS).

Java Message Service (JMS)
An application programming interface that provides Java language functions for handling messages. See also Application Messaging Interface (AMI) and Message Queue Interface (MQI).

Java virtual machine (JVM)
A software implementation of a processor that runs compiled Java code (applets and applications).

JCA See Java EE Connector Architecture.

JCL See Job Control Language

JDBC See Java Database Connectivity.

JMS See Java Message Service.

JMS domain
The message domain that includes all messages that are produced by the WebSphere MQ implementation of the Java Message Service standard. These messages, which have a message type of either JMSMap or JMSStream, are supported in the same way as messages in the XML domain, and are parsed by the XML parser. See also BLOB domain, DataObject domain, IDOC domain, MIME domain, MRM domain, SOAP domain, XML domain, XMLNS domain, and XMLNSC domain.

Job Control Language (JCL)
Job Control Language (JCL) comprises a set of Job Control Statements that are used to define work requests called jobs. JCL tells the operating system what program to run, and defines its inputs and outputs.

TOP

K

keystore
In security, a storage object, either a file or a hardware cryptographic card, where identities and private keys are stored, for authentication and encryption purposes. Some keystores also contain trusted, or public, keys.

TOP

L

LIL See loadable implementation library.

loadable implementation library (LIL)
The implementation module for a node or parser written in C. This library file is implemented in the same way as a dynamic link library, but has a file extension of `.lil` not `.dll`.

local environment
A structure within the message tree that contains broker and, optionally, user information associated with a message while it is being processed by a message flow.

In previous releases, the local environment structure was known as the Destination list; the latter term is retained for compatibility.

local error log
A generic term that refers to the logs to which WebSphere Message Broker writes records on the local system. Also known as the system log.

TOP

M

map (1) A complete transformation that has source objects that define the structure of the inputs and target objects that define the structure of the outputs. A map is represented as a `.msgmap` file.

(2) To associate a source to a target in a message map.

mapping
A target value expression.

mapping cardinality
The granularity of the way in which message elements are mapped from message source to message target. For example:

- One-to-one: associates a single source with a single target
- One-sided: associates a value with a target
- Many-to-one: associates multiple sources with a single target

message
Data that is passed from one application to another. Each message must have a structure and format that is compatible with both the sending and receiving applications.

message broker
See broker.

message category
An optional grouping of messages that are related in some way. For example, messages that relate to a particular application might be included in a single category.

message definition
An annotated XML Schema model of a message format. A message definition is a structured collection of elements, types, and groups.

message definition file
A file in a message set that contains one or more message definitions.

message dictionary
A data structure that describes all the messages in a message set in a form suitable for use by the MRM parser.

message domain
A grouping of messages that share certain characteristics. A message domain has an associated parser that interprets messages that are received and generated by a broker. WebSphere Message Broker supports messages in the BLOB domain, DataObject domain, IDOC domain, JMS domain, MIME domain, MRM domain, SOAP domain, XML domain, XMLNS domain, and XMLNSC domain. You can create additional parsers known as user-defined parsers to support messages that do not conform to the supported domains.

message element aggregation
A mapping in which all the repeatable elements in one instance are mapped to another instance. It is not possible to map the repeatable elements themselves, only the instances. This aggregation is useful when mapping all possible inputs to one or more outputs, and can be used for copying an array, or for assigning a scalar, such as a summation. Use message element aggregation when the following conditions are met:
- A single source and target are selected.
- Source and target are of simple numeric type.
- The source repeats.

message flow
A sequence of processing steps that run in the broker when an input message is received. A message flow is created in the WebSphere Message Broker Toolkit by including a number of message flow nodes that each represents a set of actions that define a processing step. The connections in the flow determine which processing steps are carried out, in which order, and under which conditions. A message flow must include an input node that provides the source of the messages that are processed. Message flows are then ready to deploy to a broker for execution. See also subflow.

message flow node
A processing step in a message flow, also called a message processing node. A message flow node can be a built-in node, a user-defined node, or a subflow node.

message flow node connection
An entity that connects an output terminal of one message flow node to an input terminal of another. A message flow node connection represents the flow of control and data between two message flow nodes.

message format
The definition of the internal structure of a message, in terms of the fields and the order of those fields. When a message format is self-defining, the message is interpreted dynamically when it is read.

message group
A list of elements with information about how those elements can appear in a message. Message groups can be ordered, unordered, or selective.

message model
See message definition.

message parser
A program that interprets an incoming message and creates an internal representation of the message in a tree structure, and that regenerates a bit stream for an outgoing message from the internal representation.

message processing node
See message flow node.

Message Queue Interface (MQI)
The programming interface that is provided by WebSphere MQ queue managers. Application programs use the programming interface to access message queuing services. See also Application Messaging Interface (AMI) and Java Message Service (JMS). Applications that use the MQI, connect to the broker using WebSphere MQ Enterprise Transport.

message set
A folder in a message set project that contains one or more message definition files. It can be deployed to a broker in a broker archive file.

message set project
The eclipse container for a message set.

message template
A means of identifying a message format within the broker. It consists of four parts: message domain, message set, message type, and physical format.

message tree
The logical tree structure that represents the content and structure of a message in the broker. The message tree is created by a message parser from the input message received by a message flow, according to a message template.

message type
The name given to a message definition in a message definition file.

metadata
The data that describes the characteristic of stored data.

metadata tree
A list in a tree structure that presents all the objects discovered from the enterprise information system (EIS) by the enterprise service discovery process. These objects can be selected from the tree to generate a business object.

MIME See Multipurpose Internet Mail Extensions.

MIME domain
The message domain that includes all messages that conform to the MIME standard. See also BLOB domain, DataObject domain, IDOC domain, JMS domain, MRM domain, SOAP domain, XML domain, XMLNS domain, and XMLNSC domain.

MIME parser
A program that interprets a message that belongs to the MIME domain, and generates the corresponding tree from the bit stream on input, or the bit stream from the tree on output.

MQI See Message Queue Interface.

MQRFH
An architected message header that is used to provide metadata for the processing of a message. This header is supported by the WebSphere MQ (MQSeries) Publish/Subscribe SupportPac.

MQRFH2
An extended version of MQRFH, providing enhanced function in message processing.

MRM domain
The message domain that can parse, and write, a wide variety of message

formats. This domain is primarily intended for non-XML message formats, but it can also parse and write XML messages. Message models are created in the WebSphere Message Broker Toolkit, with one or more physical formats. Messages in the MRM domain are processed by the MRM parser. See also BLOB domain, DataObject domain, IDOC domain, JMS domain, MIME domain, SOAP domain, XML domain, XMLNS domain, and XMLNSC domain.

MRM parser
A program that interprets a message that belongs to the MRM domain, and generates the corresponding tree from the bit stream on input, or the bit stream from the tree on output. Its interpretation depends on the physical format that you have associated with the input or output message.

multilevel wildcard
A wildcard that can be specified in subscriptions to match any number of levels in a topic.

multipart message
A message that contains one or more other messages within its structure. The contained message is sometimes referred to as an embedded message.

Multipurpose Internet Mail Extensions
An Internet standard that defines different forms of data, including video, audio, or binary data, that can be attached to email without requiring translation into ASCII text.

TOP

N

namespace
In XML and XQuery, a uniform resource identifier (URI) that provides a unique name to associate with the element, attribute, and type definitions in an XML schema, or with the names of elements, attributes, types, functions, and errors in XQuery expressions.

XML instance documents, XML Schemas, and message definitions can use namespaces.

node (1) An endpoint or junction used in a message flow. See message flow node.

(2) Any element in a tree. See tree node.

TOP

O

ODBC
See Open Database Connectivity.

Open Database Connectivity (ODBC)
A standard application programming interface (API) for accessing data in both relational and non-relational database management systems. By using this API, database applications can access data stored in database management systems on various computers, even if each database management system uses a different data storage format and programming interface.

operation mode
A property of a broker that determines what operations it can perform.

outbound processing
The process by which a calling client application uses the adapter to update or retrieve data in an enterprise information system (EIS). The adapter uses operations such as create, update, delete, and retrieve to process the request.

output node
A message flow node that represents a point at which messages leave the message flow or subflow. See also input node.

TOP

P

package group
A group of one or more packages that are designed to work together and can be installed to one directory. In WebSphere Message Broker, this term refers to the group of products that are installed and maintained by Installation Manager, which includes the WebSphere Message Broker Toolkit. Products that are installed into a package group share common files and resources. You can create multiple package groups on a single computer.

parser See message parser.

partitioned data set (PDS, PDSE)
In a z/OS environment, a data set in direct-access storage that is divided into partitions, which are called members. A partitioned data set (extended) (PDSE) is an extension to a PDS that contains an indexed directory in addition to the members.

pattern
A reusable solution that encapsulates a tested approach to solving a common architecture, design, or deployment task in a particular context.

pattern archive
An archive file that contains all the installable pattern resources. The pattern archive can be distributed by using a pattern community site.

pattern author
The developer that creates a pattern to meet a business or technical requirement.

pattern authoring
The process of configuring one or more regular projects to turn them into a pattern.

pattern authoring project
A project that contains the information used to create a pattern.

pattern categories
Categories that are based on pattern classification and that structure the display in the Patterns Explorer view.

pattern community site
An application that supports uploading, browsing, searching, and downloading pattern archives. The application can be either a website or a shared file system directory.

pattern instance
The implementation of a pattern, consisting of a pattern instance project and one or more regular WebSphere Message Broker projects that implement the pattern. A pattern instance is generated by providing appropriate customization values to the parameters available in the pattern.

pattern instance project
A project that contains project references to all other projects in the workspace, relating to a specific pattern instance. A pattern instance project also contains a pattern instance configuration file that stores the pattern parameter values.

pattern parameter
A parameter that customizes and configures a pattern. For example, a queue name from which messages are read.

pattern user
A user who configures a pattern that the pattern author has created. The pattern is available to a pattern user in the Patterns Explorer view.

PDS, PDSE
See partitioned data set.

perspective
A group of views that show various aspects of the resources in the WebSphere Message Broker Toolkit. See also view.

physical format
The physical representation of a message within the bit stream. The supported physical formats are Custom Wire Format, XML Wire Format, and Tagged/Delimited String Format. Physical format information is used only by the MRM parser and the IDOC parser.

point-to-point
A style of messaging application in which the sending application knows the destination of the message. Contrast with publish/subscribe.

predefined element and message
An element or message for which a matching definition exists in the message model. See also self-defining element and message.

principal
An individual user ID (for example, a login ID) or a group. A group can contain individual user IDs and other groups, to the level of nesting that is supported by the underlying facility.

property
A characteristic of an object that describes the object. A property can be changed or modified. Properties can describe the name, type, value, or behavior of an object, and various other characteristics.

Resources that are created and maintained in the WebSphere Message Broker Toolkit and components have properties; for example, message flow nodes, deployed message flows, and brokers.

publication
In publish/subscribe messaging, a piece of information about a specified topic that is available to a queue manager, for delivery to subscribed applications.

publication node
An end point of a specific path through a message flow to which a client application subscribes, identified to the client by its subscription point.

publisher
An application that makes information about a specified topic available to a broker in a publish/subscribe system.

publish/subscribe
A style of messaging application in which the providers of information (publishers) are de-coupled from the consumers of that information (subscribers) using a broker. See also topic. Contrast with point-to-point messaging.

publish/subscribe topology
The brokers, and the connections between them, that support publish/subscribe applications.

TOP

Q

queue A WebSphere MQ object to which message queuing applications can put messages, and from which message queuing applications can get messages.

queue manager
A system program that provides queuing services to applications. A queue manager provides an application programming interface (the MQI) that enables programs to access messages on the queues that the queue manager owns.

TOP

R

request/reply
A type of messaging application in which a request message is used to request a reply from another application. Contrast with datagram.

Resource Recovery Services (RRS)
A z/OS facility that provides two-phase sync point support by participating resource managers.

retained publication
A published message that is kept at the broker for propagation to clients that subscribe in the future.

RRS See Resource Recovery Services.

TOP

S

SCADA
See Supervisory, Control, And Data Acquisition.

SCA See Service Component Architecture (SCA).

schema
See XML Schema.

self-defining element and message
An element or message for which no matching definition exists in a message model, but which can be parsed without reference to a model. For example, a message that is coded in XML can be self-defining. See also predefined element and message.

send-and-forget
See datagram.

Service Component Architecture (SCA)
An architecture in which all elements of a business transaction, such as access to Web services, Enterprise Information System (EIS) service assets, business rules, workflows, databases, and so on, are represented in a service-oriented way.

shared resources directory
The directory that contains software files or plug-ins that are shared by packages. In WebSphere Message Broker, this term refers to the directory that contains Eclipse plug-ins and features, and other common files and resources, that are used by all products on the computer that are installed and maintained by Installation Manager, which include the WebSphere Message Broker Toolkit. The contents of this directory are used by all products in all the package groups that are defined on the computer.

simple element
A field in a message that is based on a simple type. A simple element can repeat, and it can define a default or a fixed value. See also complex element.

simple type
A characteristic of a simple element that defines the type of data within a message (for example, string, integer, or float). A simple type can have value constraints which place limits on the values of any simple elements based on that simple type. See also complex type.

single-level wildcard
A wildcard that can be specified in subscriptions to match a single level in a topic.

SOAP A lightweight, XML-based protocol for exchanging information in a decentralized, distributed environment. SOAP can be used to query and return information and invoke services across the Internet.

SOAP domain
The message domain that includes all messages that conform to the SOAP standard. You must create a message model for messages that you process in this domain. See also BLOB domain, DataObject domain, IDOC domain, JMS domain, MIME domain, MRM domain, XML domain, XMLNS domain, and XMLNSC domain.

SOAP parser
A program that interprets a message that belongs to the SOAP domain, and generates the corresponding tree from the bit stream on input, or the bit stream from the tree on output. The bit stream is a representation of an XML file.

SQL See Structured Query Language.

SQLJ A Java extension that supports static Structured Query Language statements embedded within Java code.

stream
A method of topic partitioning that is used by applications that connect to WebSphere MQ Publish/Subscribe brokers.

Structured Query Language (SQL)
A standardized programming language that is used to define and manipulate data in a relational database. ESQL, the language that is used by WebSphere Message Broker, is based on SQL, and has many similar constructs.

style sheet
A specification of formatting instructions that, when applied to structured information, provides a particular rendering of that information (for example, online or printed). Different style sheets can be applied to the same piece of structured information to produce different presentations of the information.

subflow
A sequence of processing steps, implemented by message flow nodes, that is designed to be embedded in a message flow or in another subflow. A subflow must include at least one Input or Output node. A subflow can be started by a broker only as part of the message flow in which it is embedded, and therefore cannot be deployed.

subflow node
A message flow node that represents a subflow.

subscriber
A publish/subscribe application that requests information about a topic.

subscription
A record that contains the information that a subscriber passes to its local broker to describe the publications that it wants to receive.

subscription filter
A predicate that specifies the subset of messages that are to be delivered to a particular subscriber.

subscription point
The name that a subscriber uses to request publications from a particular set of publication nodes. It is the property of a publication node that differentiates that publication node from other publication nodes in the same message flow.

substitution group
An XML Schema feature that provides a means of substituting one element for another in an XML message. A substitution group contains a list of global elements that can appear in place of another global element, called the head element.

Supervisory, Control, And Data Acquisition (SCADA)
A term used to describe any form of remote telemetry system that is used to gather data from remote sensor devices (for example, flow rate meters on an oil pipeline) and for the near real-time control of remote equipment (for example, pipeline valves).

system log
See local error log.

TOP

T

Tagged/Delimited String (TDS) Format
The physical representation of a message in the MRM domain that has a number of data elements separated by tags and delimiters.

target property
A message flow property that is selected by the pattern author to be configured by the pattern.

TDS Format
See Tagged/Delimited String Format.

terminal
The point at which one node in a message flow is connected to another node. You can connect terminals to control the route that a message takes, dependent on the outcome of the operation that is performed on that message by the node.

topic A character string that describes the nature of the data that is published in a publish/subscribe system.

topic-based subscription
A subscription specified by a subscribing application that includes a topic that filters publications.

tree node
An element in a mapping tree; a container for the mapping type such as a message, database table, a column, or a basic element.

truststore
In security, a storage object, either a file or a hardware cryptographic card, where public keys are stored in the form of trusted certificates, for authentication purposes in Web transactions. In some applications, these trusted certificates are moved into the application keystore to be stored with the private keys.

type A characteristic of a message element that describes its data content. See also simple type and complex type.

TOP

U

Unicode Transformation Format, 8-bit encoding form (UTF-8)
A transformation format that is designed for ease of use with existing ASCII-based systems. UTF-8 is an encoding of Unicode character strings that optimizes the encoding of ASCII characters in support of text-based communication.

uniform resource identifier (URI)
An encoded address that represents any resource, such as an HTML document, image, video clip, or program, on the Web; a URI is an abstract superclass compared with a Uniform resource locator or a Uniform resource name, which are concrete entities.

uniform resource locator (URL)
The unique address of an information resource that is accessible in a network such as the Internet. The URL includes the abbreviated name of the protocol used to access the information resource and the information used by the protocol to locate the information resource.

A Web server typically maps the request portion of the URL to a path and file name. Also known as universal resource locator.

uniform resource name (URN)
A name that uniquely identifies a Web service to a client.

URI See uniform resource identifier.

URL See uniform resource locator.

URN See uniform resource name.

user-defined extension
An optional component that is designed by the user to extend the functions of WebSphere Message Broker. A user-defined extension can be either a node or a message parser. See also user-defined node and user-defined parser.

user-defined node
An extension to the broker that provides a new message flow node in addition to the nodes that are supplied with the product. See also implementation function and utility function.

user-defined parser
An extension to the broker that provides a new message parser in addition to the parsers that are supplied with the product. See also implementation function and utility function.

user-defined pattern
A pattern that is created by a pattern author.

UTF-8 See Unicode Transformation Format.

utility function
A function provided by the broker that can be used by developers who write user-defined nodes or parsers.

TOP

V

value constraint
A limit that sets a restriction on the values that a simple type can represent.

view In Eclipse-based user interfaces, a pane that is outside the editor area, which can be used to look at or work with the resources in the WebSphere Message Broker Toolkit. For example, you can view and edit your project files in the Broker Development view (previously called the Resource Navigator view). See also perspective.

TOP

W

warehouse
A persistent, data store for historical events (or messages). The Warehouse node within a message flow supports the recording of information in a database for subsequent retrieval and processing by other applications.

Web service
A self-contained, self-describing modular application that can be published, discovered, and invoked over a network using standard network protocols.

Typically, XML is used to tag the data, SOAP is used to transfer the data, WSDL is used for describing the services available, and UDDI is used for listing what services are available.

Web Services Description Language (WSDL)
An XML-based specification for describing networked services as a set of endpoints that operate on messages that contain either document-oriented or procedure-oriented information. A WSDL document enables a Web services client to invoke a Web service using the messages defined in a message definition.

WebSphere Adapters
WebSphere Adapters facilitate the exchange of business objects between Enterprise Information Systems (such as SAP Software, PeopleSoft Enterprise, and Siebel Business Application systems) and other applications.

WebSphere Message Broker Explorer
A graphical user interface based on the Eclipse platform for administering your brokers.

WebSphere Message Broker pattern
A pattern in the WebSphere Message Broker Toolkit that exposes one or more pattern parameters for a pattern user to complete.

WebSphere Message Broker Toolkit
A graphical user interface built on Eclipse that is used to provide integration and connectivity solutions by developing resources associated with message flows.

WebSphere MQ Enterprise Transport
A transport protocol supported by WebSphere Message Broker that enables WebSphere MQ application clients to connect to brokers.

WebSphere MQ Everyplace®
A generally available WebSphere MQ product that provides proven WebSphere MQ reliability and security for mobile and wireless devices. WebSphere MQ Everyplace applications connect to the broker using WebSphere MQ Mobile Transport.

WebSphere MQ Mobile Transport
A transport protocol supported by WebSphere Message Broker that enables WebSphere MQ Everyplace application clients to connect to brokers.

WebSphere MQ Web Services Transport
A transport protocol supported by WebSphere Message Broker that enables HTTP-compliant application clients to connect to brokers.

wildcard
A character that can be specified in subscriptions to match a range of topics. See also multilevel wildcard and single-level wildcard.

work_path
The location in the local file system in which the component stores internal and working data. For example, the default location on Windows XP systems is `C:\Documents and Settings\All Users\Application Data\IBM\MQSI\`.

working set
A logical collection of application projects, that you can use to limit the number of resources that are displayed in the Broker Application Development perspective. See also active working set.

World Wide Web Consortium (W3C)
An international industry consortium set up to develop common protocols to promote the evolution and interoperability of the World Wide Web.

WSDL
See Web Services Description Language.

W3C See World Wide Web Consortium.

TOP

X

XML See Extensible Markup Language.

XML domain
The message domain that includes all messages that conform to the W3C XML standard. Messages in this domain are processed by the XML parser. See also BLOB domain, DataObject domain, IDOC domain, JMS domain, MIME domain, MRM domain, SOAP domain, XMLNS domain, and XMLNSC domain.

The XML domain is deprecated; use the XMLNSC domain for new messages.

XML parser
A program that interprets a message that belongs to the XML domain and JMS domains, and generates the corresponding tree from the bit stream on input, or the bit stream from the tree on output. The bit stream is a representation of an XML file.

XMLNS domain
An extension of the XML domain that contains messages that conform to the W3C XML standard, and that can also use the namespaces specification. Messages in this domain are processed by the XMLNS parser. See also BLOB domain, DataObject domain, IDOC domain, JMS domain, MIME domain, MRM domain, SOAP domain, XML domain, and XMLNSC domain.

XMLNS parser
A program that interprets a message that belongs to the XMLNS domain, and generates the corresponding tree from the bit stream on input, or the bit stream from the tree on output. The bit stream is a representation of an XML file.

XMLNSC domain
An extension of the XML domain that provides high-performance XML parsing and offers optional XML Schema validation. Messages in this domain are processed by the XMLNSC parser. You can create a message model for messages that you process in this domain, but a model is required only if you want to validate the message. See also BLOB domain, DataObject domain, IDOC domain, JMS domain, MIME domain, MRM domain, SOAP domain, XML domain, and XMLNS domain.

XMLNSC parser
A program that interprets a message that belongs to the XMLNSC domain, and generates the corresponding tree from the bit stream on input, or the bit stream from the tree on output. The bit stream is a representation of an XML file.

XML Path Language (XPath)
A language designed to uniquely identify or address parts of a source XML document, for use with XSLT. XPath provides basic facilities for the manipulation of strings, numbers, and Boolean values in message flow resources. For example, it can be used by Java programs within a JavaCompute node, or as the expression language within a Mapping node, or by the properties of several other nodes.

XML Schema
An international standard that defines a language for describing the structure of XML documents. An XML Schema formally describes and constrains the content of XML documents by indicating which elements are valid and in which combinations. (An XML Schema is an alternative to a document type definition (DTD), and can be used to extend functionality in the areas of data typing, inheritance, and presentation.) The XML Schema language is ideally suited to describing the messages that flow between business applications, and is widely used in the business community for this purpose. Message definitions are annotated XML Schema.

XML Schema Definition Language (XSD)
A language for describing XML files that contain XML Schema.

XML Wire Format
The physical representation of a message in the MRM domain that can be parsed as XML.

XPath See XML Path Language.

XSD See XML Schema Definition Language (XSD).

XSL See Extensible Stylesheet Language.

TOP

Accessibility features for WebSphere Message Broker

Accessibility features help users who have a physical disability, such as restricted mobility or limited vision, to use information technology products successfully.

Accessibility features

The following list includes the major accessibility features in WebSphere Message Broker. You can use screen-reader software to hear what is displayed on the screen.

- Supports keyboard-only operation
- Supports interfaces commonly used by screen readers

Tip: This information center, and its related publications, are accessibility-enabled for the IBM Home Page Reader. You can operate all features using the keyboard instead of the mouse.

If you are reading a PDF file with a screen reader, the default reading option typically returns the best results. In some cases, the way in which the PDF file has been generated might require you to select one of the other reading options; for example, `Use reading order in raw print stream`.

Keyboard navigation

This product uses standard Linux and Microsoft Windows navigation keys.

Visit the IBM Accessibility Center for more information about the commitments that IBM makes towards accessibility.

Related reference:

"WebSphere Message Broker Explorer and WebSphere Message Broker Toolkit keyboard shortcuts" on page 6828
You can navigate all interfaces in the WebSphere Message Broker Explorer and WebSphere Message Broker Toolkit by using the keyboard.

Workbench User Guide - Keyboard shortcuts for the Workbench, Java development tools, and the debugger

Chapter 3. Migrating and upgrading

To migrate a broker domain to WebSphere Message Broker Version 7.0, plan your migration strategy, perform pre-migration tasks, migrate your domain components, and then complete post-migration tasks.

Before you begin

Migration information is regularly updated on the WebSphere Message Broker support web page with the latest details available. Click **Troubleshoot** and look for the document "Problems when migrating".

You can migrate your existing components and resources to WebSphere Message Broker Version 7.0 from previous versions of both WebSphere Message Broker and WebSphere Event Broker.

The architecture of WebSphere Message Broker Version 7.0 has been simplified:

- The broker has no requirement for a database. When you migrate from a previous version, the following changes apply:
  - Because a database is no longer a mandatory requirement for a broker, DB2 and DB2 Runtime Client (the Derby database supported only on Windows systems) are not supplied when you purchase WebSphere Message Broker Version 7.0.
  - The **mqsicreatebroker** and **mqsichangebroker** commands have been updated to remove the specification of database parameters.
  - The DatabaseInstanceMgr that was available on Windows systems to manage Derby and DB2 databases has been removed for Version 7.0. Commands that control the DatabaseInstanceMgr and the Derby database have been removed:

    mqsicreatedb

    mqsideletedb

    mqsichangedbimgr
- The broker security model is implemented by using WebSphere MQ queues. The Configuration Manager component has been removed and you cannot migrate existing components of this type. All operational data maintained by the Configuration Manager components is retrieved when you migrate your brokers.

  The commands that controlled this component have been removed:

  mqsichangeconfigmgr
  mqsicreateconfigmgr
  mqsideleteconfigmgr
  mqsireportconfigmgr
- All topic-based publish/subscribe operations are handled by WebSphere MQ. Because access control for publishers and subscribers is managed by WebSphere MQ, the User Name Server component is no longer required, and has been removed; you cannot migrate existing components of this type. The following commands are no longer available:

  mqsichangeusernameserver
  mqsicreateusernameserver
  mqsideleteusernameserver

After you have migrated, you can control brokers only from the WebSphere Message Broker Explorer or the WebSphere Message Broker Toolkit Version 7.0.

You can migrate from Version 6.1 or Version 6.0 only to a full edition of Version 7.0, Remote Adapter Deployment, Entry Edition, or Starter Edition. Do not migrate brokers from an earlier version to the Trial Edition. If you choose to migrate brokers by using the **mqsimigratecomponents** command, the brokers are migrated in enterprise mode. If you have purchased the Remote Adapter Deployment, Entry Edition, or Starter Edition, you must change the mode of all your migrated brokers to comply with the terms of your license. For more information, see "Changing the operation mode of your broker" on page 655.

Procedure

1. Check that your current installation of WebSphere Message Broker (Version 6.1 or Version 6.0) or WebSphere Event Broker (Version 6.0 only) is at a supported level for migration. Details are provided in "Supported migration paths" on page 3579.
2. Understand the options that are available to install Version 7.0 components on the same computer as components from previous versions. These options are described in "Coexistence with previous versions and other products" on page 139.
3. You must migrate your queue managers to a supported version of WebSphere MQ before you can migrate your brokers. For the latest details of all supported levels of hardware and software, visit the WebSphere Message Broker Requirements website.
4. If your configuration includes publish/subscribe applications, support has transferred from WebSphere Message Broker to WebSphere MQ. You must take additional steps to migrate publish/subscribe information from your current WebSphere Message Broker or WebSphere Event Broker configuration. Details are provided in "Migrating publish/subscribe information to WebSphere MQ" on page 141.
5. If you are currently working with WebSphere Message Broker or WebSphere Message Broker with Rules and Formatter Extension , follow the instructions in "Migrating from Version 6.1 products" on page 163.
6. If you are currently working with WebSphere Message Broker Version 6.0 or WebSphere Message Broker with Rules and Formatter Extension Version 6.0, follow the instructions in "Migrating from Version 6.0 products" on page 183.
7. If you are currently working with WebSphere Event Broker Version 6.0, and you only use publish/subscribe functions, you can migrate to WebSphere MQ. This migration path is documented in the *Migration* section of the WebSphere MQ Version 7 Information Center online.

   If you currently use, or plan to use, additional broker functions, and you want to migrate to WebSphere Message Broker Version 7.0, follow the instructions in "Migrating from Version 6.0 products" on page 183. The tasks that you must complete are the same as those tasks documented for WebSphere Message Broker Version 6.0.
8. When you have migrated your components and resources, check if the post-migration tasks apply to your configuration. Guidance is provided in "Post-migration tasks" on page 204.

Related concepts:

"Operation modes" on page 48
The operation mode that you use for your broker is determined by the license that you purchase.

"Coexistence with previous versions and other products"
WebSphere Message Broker Version 7.0 supports restricted coexistence with previous versions and with other products.

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

"Migrating publish/subscribe information to WebSphere MQ" on page 141
These tasks give an overview of how to migrate publish/subscribe information from WebSphere Message Broker or WebSphere Message Broker Version 6.0 to WebSphere MQ.

"Migrating from Version 6.1 products" on page 163
Migrate your components and resources to WebSphere Message Broker Version 7.0.

"Migrating from Version 6.0 products" on page 183
Migrate your components and resources to WebSphere Message Broker Version 7.0.

"Post-migration tasks" on page 204
After you have migrated to Version 7.0, finish setting up your environment.

Related reference:

"Migration and upgrade" on page 3579
Consider the factors involved in the migration of components and resources from Version 6.0 or Version 6.1 to Version 7.0.

"Supported migration paths" on page 3579
You can migrate to WebSphere Message Broker Version 7.0 from previous versions of the product.

Related information:

WebSphere Message Broker Requirements

WebSphere Message Broker Support

WebSphere MQ Version 7 Information Center online

Coexistence with previous versions and other products

WebSphere Message Broker Version 7.0 supports restricted coexistence with previous versions and with other products.

WebSphere Message Broker Version 7.0 requires a specific minimum supported version of WebSphere MQ. For systems other than z/OS that allow only one version of WebSphere MQ on a single computer, you must first install a supported version of WebSphere MQ and migrate your WebSphere MQ resources to this version. You can then install WebSphere Message Broker Version 7.0. For the latest details of all supported levels of hardware and software, visit the WebSphere Message Broker Requirements website.

Coexistence with previous versions of the product installed on the same computer

WebSphere Message Broker Version 7.0 can coexist with a Version 6.1 product, a Version 6.0 product, or both, on the same computer.

Component names must be unique on the computer, regardless of version.

When you migrate from a Version 6.1 or Version 6.0 product to Version 7.0, you do not have to uninstall the Version 6.1 or Version 6.0 product before installing Version 7.0. You can install Version 7.0 in a different location on the same computer, migrate your components and resources to Version 7.0, and uninstall the Version 6.1 or Version 6.0 product later, when you are sure that you no longer need it.

For details of default locations for installations of each version, see "Coexistence and migration" on page 239.

If you attempt to install the broker into a directory that already contains an installation of the broker at a previous version, you are prompted to confirm that you want to continue with the installation, because it will overwrite the existing installation. Cancel the installation and select a different directory to preserve your existing configuration.

If you install more than one version on a single computer, and you create multiple components, check that sufficient memory and disk space are available.

When you have installed Version 7.0, you can test your configuration.

Coexistence of Version 7.0 components with components from previous versions

The following restrictions apply:

- The WebSphere Message Broker Explorer and the WebSphere Message Broker Toolkit work only with brokers that you have migrated to, or created in, Version 7.0. If you try to connect to brokers at an earlier version, the connection attempt fails.
- You can run all programs that you have written to the CMP API to interact with a CMP API at Version 6.1 or Version 6.0 to interact with brokers that you have migrated to, or have created in, Version 7.0. Programs that start, stop, delete, and monitor deployed resources continue to work with Version 7.0.
- You can deploy BAR files that you have created in an earlier version of the WebSphere Message Broker Toolkit from the WebSphere Message Broker Toolkit Version 7.0 to a broker at Version 7.0, without change.
- You cannot deploy resources that you have created in the WebSphere Message Broker Toolkit Version 7.0 to brokers that are at Version 6.1 or Version 6.0.

Coexistence with SupportPac IS02

The SupportPac IS02, WebSphere Message Broker Explorer Plug-in, cannot coexist with the WebSphere Message Broker Explorer in WebSphere Message Broker Version 7.0. If you have previously installed SupportPac IS02, you must uninstall the SupportPac before installing the WebSphere Message Broker Explorer. To remove SupportPac IS02, delete or move the directory to which you extracted the `IS02 zip` file supplied with the SupportPac. Delete the `BrokerExplorer.link` file which you copied into the `C:\Program Files\IBM\WebSphere MQ\eclipseSDK33\eclipse\links\` directory.

Related tasks:

"Migrating from Version 6.1 products" on page 163
Migrate your components and resources to WebSphere Message Broker Version 7.0.

"Migrating from Version 6.0 products" on page 183
Migrate your components and resources to WebSphere Message Broker Version 7.0.

"Post-migration tasks" on page 204
After you have migrated to Version 7.0, finish setting up your environment.

"Resolving problems that occur when you start resources" on page 3371
Use the advice given here to help you to resolve common problems that can occur when you start resources.

"Installing WebSphere Message Broker Explorer" on page 280
To use WebSphere Message Broker Explorer only, without installing the complete WebSphere Message Broker Toolkit, use the WebSphere Message Broker Explorer installation wizard to install the WebSphere Message Broker Explorer.

Related reference:

"Memory and disk space requirements" on page 3584
Check the memory and disk space that is required for your installation.

"Supported migration paths" on page 3579
You can migrate to WebSphere Message Broker Version 7.0 from previous versions of the product.

Migrating publish/subscribe information to WebSphere MQ

These tasks give an overview of how to migrate publish/subscribe information from WebSphere Message Broker or WebSphere Message Broker Version 6.0 to WebSphere MQ.

Before you begin

Before you start

- Read about publish/subscribe on WebSphere MQ.
- For the latest details of all supported levels of hardware and software, visit the WebSphere Message Broker Requirements website.
- Plan your migration, and pay specific attention to your Access Control List (ACL) authorities.

  The migration tool accepts and modifies only your positive ACL entries. Therefore, you must inspect your ACL entries and modify all that do not meet the appropriate criteria; see "Access Control List (ACL) migration - publish/subscribe" on page 146
- Back up your Version 6.1 resources.

To migrate publish/subscribe information to WebSphere Message Broker Version 7.0 you must have WebSphere MQ installed on your system.

About this task

To migrate your existing publish/subscribe information to WebSphere MQ, complete the following steps.

Procedure

1. Install WebSphere MQ on your system. This step upgrades your queue managers; for instructions about how to install this product, see the WebSphere MQ documentation.

2. Install WebSphere Message Broker Version 7.0. This installation must be alongside your existing WebSphere Message Broker Version 6.1 or Version 6.0 system.
3. Explicitly enable content-based filtering if you are using this option. For further information, see "Enabling content-based filtering with publish/subscribe" on page 2224.
4. Use a WebSphere MQ command prompt to run the **migmbbrk** command. This command copies data from your WebSphere Message Broker Version 6.1 or Version 6.0 system to your WebSphere MQ system.
5. Use a WebSphere Message Broker Version 7.0 command prompt to run the **mqsimigratecomponents** command; for further information, see "**mqsimigratecomponents** command" on page 3894.
6. If you have existing MQRFH format retained publications in WebSphere Message Broker Version 6.0 or Version 6.1, see "Retained publications with headers in MQRFH format" on page 149 for further migration considerations.

What to do next

Next: Follow the instructions detailed in "Migrating publish/subscribe from WebSphere Message Broker Version 6.0 or WebSphere Message Broker to WebSphere MQ" on page 146

Related concepts:

"Access Control List (ACL) migration - publish/subscribe" on page 146
The function that migrates publish/subscribe configuration data from WebSphere Event Broker Version 6.0 and WebSphere Message Broker to WebSphere MQ produces a file containing suggested security commands, and creates topic objects as required.

Related tasks:

"Migrating publish/subscribe from WebSphere Message Broker Version 6.0 or WebSphere Message Broker to WebSphere MQ" on page 146
You can migrate publish/subscribe configuration data from WebSphere Message Broker Version 6.0 or WebSphere Message Broker to WebSphere MQ.

Related reference:

"**mqsimigratecomponents** command" on page 3894
Use the **mqsimigratecomponents** command to migrate a component from a previously installed version of the product to another version on the same computer.

WebSphere MQ migmbbrk command

Migrate the publish/subscribe information from WebSphere Message Broker Version 6.0 or WebSphere Message Broker to WebSphere MQ

Purpose

Use the WebSphere MQ **migmbbrk** command to migrate the publish/subscribe information from a WebSphere Message Broker Version 6.0 or WebSphere Message Broker broker to a WebSphere MQ queue manager. The command runs a migration process that migrates the following publish/subscribe information to the queue manager that is associated with the named broker:

- Subscriptions
- Subscription points (subscription points are supported only when RFH2 messages are used)
- Streams

- Retained publications

When you run the **migmbbrk** command, you must use the profile for the version of WebSphere Message Broker from which you are migrating.

The **migmbbrk** command does not migrate the Access Control List (ACL). Instead, running the migration with the **-t** or **-r** parameters produces a file containing suggested **setmqaut** commands to set up a security environment in the queue manager that is equivalent to the security environment that existed in the broker. You must review and modify the security command file as needed and run the commands to set up a security environment in the queue manager, equivalent to the one that existed in the broker, before you run the migration with the **-c** parameter to complete the migration.

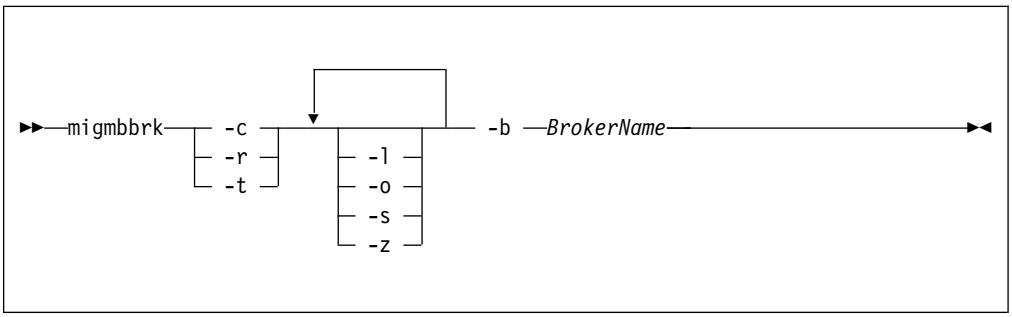
Note: On UNIX systems, all authorities are held by user groups internally, not by principals, which has the following implications:

- If you use the **setmqaut** command to grant an authority to a principal, the authority is granted to the primary user group of the principal. Therefore, the authority is effectively granted to all members of that user group.
- If you use the **setmqaut** command to revoke an authority from a principal, the authority is revoked from the primary user group of the principal. Therefore the authority is effectively revoked from all members of that user group.

You must issue the **migmbbrk** command from a command environment that can process both WebSphere MQ and WebSphere Message Broker commands successfully. Typically, you can issues these commands from a WebSphere Message Broker command console.

The WebSphere Message Broker Version 6.0, WebSphere Event Broker Version 6.0, or WebSphere Message Broker publish/subscribe configuration data, which is stored in the subscription database tables, is not deleted by the migration process. This configuration data is therefore available to use until you explicitly delete it.

Syntax



Parameters

-b *BrokerName*
(Required) The name of the broker that is the source of the publish/subscribe information that is to be migrated. The queue manager to which the publish/subscribe information is migrated is the queue manager that is associated with the named broker.

-c

(Required) Complete the migration of the publish/subscribe configuration data. The completion phase of the migration uses the topic objects that are created in the initial **-t** phase. The broker state might have changed since the initial phase was run, and new additional topic objects might now be required. If this is the case, the completion phase creates these new topic objects as necessary. The completion phase does not delete any topic objects that have become unnecessary; you might need to delete any topic objects that you do not require.

Before you complete the migration you must review and modify the security command file produced in the **-r** or **-t** phase as required and run the commands to set up a security environment in the queue manager, equivalent to the one that existed in the broker.

Before you run this completion phase you must run the initial **-t** phase. You cannot use the **-c** parameter with the **-r** parameter or the **-t** parameter. This phase also creates a migration log.

-r

(Required) Rehearse the migration process but do not change anything. You can use this before running the migration with the **-t** parameter, to create a migration log, including any errors, so that you can observe what the result of the migration process would be, but without changing the current configurations.

Rehearsing the migration also produces a file containing suggested **setmqaut** commands to set up a security environment in the queue manager that is equivalent to the security environment that existed in the broker. Before you complete the migration with the **-c** parameter you must review and modify the security command file as required and run the commands to set up a security environment in the queue manager, equivalent to the one that existed in the broker.

You cannot use the **-r** parameter with the **-c** parameter or the **-t** parameter.

-t

(Required) Create topic objects that might be needed in the queue manager, based on the ACL entries that are defined in the broker.

Use of the **-t** parameter also produces a file containing suggested **setmqaut** commands to set up a security environment in the queue manager that is equivalent to the security environment that existed in the broker. The topic objects are created in anticipation of you running the security commands to create ACLs for the topic objects. Before you complete the migration with the **-c** parameter you must review and modify the security command file as required and run the commands to set up a security environment in the queue manager, equivalent to the one that existed in the broker.

You must run this phase before you run the completion phase with the **-c** parameter. You cannot use the **-t** parameter with the **-c** parameter or the **-r** parameter. This phase also creates a migration log.

-l

(Optional) Leave the broker running. If you do not specify this parameter, the broker is shut down by default at the end of the migration process.

-o

(Optional) Overwrite subscription or retained publication that already exists in the queue manager and that has the same name as a subscription or retained

publication that is being migrated from the broker, with the publish/subscribe information that was retrieved from the broker. The **-o** parameter has no effect if you use it with the **-r** parameter.

-s
(Optional) Discard intermediate information that was retained from a previous instance of the migration process that failed or was interrupted. The migration process populates private queues with temporary data. If the migration process completes successfully, the temporary data is deleted. If you do not specify this parameter and the migration process fails or is interrupted, the temporary data is retained and is used by the migration process if you restart it, so that the process resumes at the point where it previously failed or was interrupted.

-z
(Optional) Run the migration process, regardless of whether it has previously run to a successful completion. If you do not specify this parameter and the migration process has previously run to a successful completion, the process recognizes this fact and exits. You can use the **-o** parameter with the **-z** parameter, but this is not mandatory. A previous rehearsal of the migration using the **-r** parameter does not count as a successful completion.

Return codes

0	Migration completed successfully
20	An error occurred during processing

Output files

The migration process writes two output files to the current directory:

amqmigrateacl.txt
A file containing a list of **setmqaut** commands, for you to review, change, and run if appropriate, to help you to reproduce your ACLs.

amqmigmbbrk.log
A log file containing a record of the details of the migration.

Examples

This command migrates the publish/subscribe information of broker BRK1 into its associated queue manager and specifies that the migration process runs regardless of whether it has previously run to a successful completion. It also specifies that any subscription or retained publication that already exists in the queue manager, that has the same name as a subscription or retained publication that is being migrated from the broker, must be overwritten.

```
migmbbrk -r -z -o -b BRK1
```

Supported operating systems

The **migmbbrk** command is supported only on the following platforms that support WebSphere Event Broker Version 6.0 or WebSphere Message Broker :

Microsoft Windows XP Professional with SP2, 32–bit versions only
Solaris on x86-64: Solaris 10
Solaris on SPARC: Solaris 9 (64–bit)
AIX Version 5.2 or later, 64–bit only
HP-Itanium: HP-UX 11i
Linux on IBM z Systems (64-bit)

Linux on POWER® (64–bit)
Linux on x86
Linux on x86-64

On z/OS the equivalent function to the migmbbrk command is provided by the CSQUMGMB utility.

For the latest details of all supported levels of hardware and software, visit the WebSphere Message Broker Requirements website.

Access Control List (ACL) migration - publish/subscribe

The function that migrates publish/subscribe configuration data from WebSphere Event Broker Version 6.0 and WebSphere Message Broker to WebSphere MQ produces a file containing suggested security commands, and creates topic objects as required.

On WebSphere Event Broker Version 6.0 and WebSphere Message Broker , the default behavior is for all user IDs to have access to any topic unless the ACL explicitly restricts access. In WebSphere MQ the default behavior is for no user ID to have access to any topic unless the ACL explicitly authorizes access, and it is not possible to explicitly restrict access. Because of this difference in security approaches, the migration process cannot directly migrate WebSphere Event Broker Version 6.0 and WebSphere Message Broker ACLs to the WebSphere MQ queue manager.

If the rehearsal or initial phase of the migration finds an ACL entry that denies access, the process cannot produce a WebSphere MQ equivalent command. Instead, it reports it in the security command file and advises that the ACL migration must be performed manually.

You can modify the:

- Broker security settings to match the WebSphere MQ security approach, and run the rehearsal or initial phase of the migration again to produce a new security command file, or
- Security command file as needed. You must set up a security environment in the queue manager, equivalent to the one that existed in the broker, before you run the completion phase of the migration.

WebSphere Event Broker Version 6.0 and WebSphere Message Broker provide the capability to define topic trees, but there is no capability to set specific attributes for a particular individual topic in a topic tree. WebSphere MQ supports the concept of topic objects that allow you to set specific, nondefault attributes for a topic. An Access Control List is a property of a topic object. The initial phase of the migration creates topic objects speculatively, based on the ACL entries that are defined in the broker and in anticipation of you subsequently running the security commands to create ACLs for the topic objects. When you have resolved what security settings you need, you might need to delete the topic objects that you do not require.

Migrating publish/subscribe from WebSphere Message Broker Version 6.0 or WebSphere Message Broker to WebSphere MQ

You can migrate publish/subscribe configuration data from WebSphere Message Broker Version 6.0 or WebSphere Message Broker to WebSphere MQ.

Before you begin

Before you start:

For the latest details of all supported levels of hardware and software, visit the WebSphere Message Broker Requirements website.

This topic assumes the following prerequisite steps:

- You have installed WebSphere MQ.
- You want to migrate publish/subscribe support from WebSphere Message Broker Version 6.0 or WebSphere Message Broker to WebSphere MQ.
- You want to upgrade from WebSphere Message Broker Version 6.0 or WebSphere Message Broker to WebSphere Message Broker Version 7.0. You are currently using the broker as an interface for functions other than, but in addition to, publishing and subscribing, and you want to continue to use those other functions after you migrate publish/subscribe information to WebSphere MQ.
- On distributed systems, you have set up and initialized a command environment in which WebSphere MQ and WebSphere Message Broker commands can run.
- The WebSphere MQ queue manager is not currently handling any publish or subscribe messages.
- The queue manager attribute **PSMODE** is set to COMPAT; run the command **ALTER QMGR PSMODE(COMPAT)**.

Note: WebSphere MQ publish/subscribe functionality might alter the header chains in your published message. The message body is not affected, but avoid assumptions about the value of the **CodedCharSetId** field in MQRFH2 headers when subscribing.

About this task

This task upgrades your enterprise from WebSphere Message Broker Version 6.0 or WebSphere Message Broker to WebSphere Message Broker Version 7.0.

Procedure

1. Uninstall WebSphere MQ Version 6.X from your system.
2. Install WebSphere MQ on your system.
3. Install WebSphere Message Broker Version 7.0 on your system. Do not remove your existing WebSphere Message Broker installation until you have migrated your current publish/subscribe data.
4. Start the broker on your WebSphere Message Broker Version 6.0 or Version 6.1 system.
5. Run the migration process with the **-r** parameter.

   For more information, see "WebSphere MQ migmbbrk command" on page 142. This option rehearses the migration of the publish/subscribe configuration data from the broker to its underlying queue manager without changing either of the configurations. For example, on supported systems other than z/OS, use the following command to rehearse the migration from a broker named BRK1:

   ```
   migmbbrk -r -b BRK1
   ```

   On z/OS use the equivalent parameters with the **CSQUMGMB** utility.
6. Run the following command:

```
migmbbrk -t  -b brokername
```

then run the security commands that set up the security environment. If you do not run these commands, migration fails.

7. To check what is to be migrated in a real migration, review the contents of the log file.
8. Back up the file that contains the security commands, and create your own copy of the file.
9. Review and, if necessary, edit the commands in your copy of the security commands file to ensure that they create a security environment that is like your broker security environment.
10. Run the migration process with the **-o** parameter.

    The migration process migrates the publish/subscribe configuration data to the queue manager, and creates a log file and a new security commands file.

    For example, on supported systems other than z/OS, use the following command:

    ```
    migmbbrk -c -o -b BRK1
    ```

    This command completes the following actions:

    - Migrates the publish/subscribe configuration data from broker BRK1
    - Overwrites any subscription or retained publication that exists in the queue manager
    - Creates a file that has the same name as a migrating subscription or retained publication

    On z/OS use the equivalent parameters with the CSQUMGMB utility.
11. Stop the broker and check the Administration log to confirm that the broker has stopped.

    Note: After you have stopped the broker, the queue manager is in COMPAT mode. As a result, the publish/subscribe state cannot be changed in WebSphere MQ or WebSphere Message Broker, allowing the migration to proceed.
12. Check the contents of the new security commands file against your backup copy to make sure that nothing related to the ACL has changed since you rehearsed the migration. If anything has changed, you might need to edit your copy of the security commands file.
13. Run the commands that are in your copy of the security commands file.
14. Optional: Remove WebSphere Message Broker Version 6.0 or WebSphere Message Broker from your system if necessary. Carry out this step only if you are sure that you are not going to require your WebSphere Message Broker Version 6.0 or WebSphere Message Broker system in the future.

    To reverse the migration process, see “Migrating publish/subscribe information from WebSphere MQ to WebSphere Message Broker Version 6.0 or WebSphere Message Broker ” on page 150.
15. Run the **mqsimigratecomponents** command; for more information, see “**mqsimigratecomponents** command” on page 3894.
16. To set the **PSMODE** queue manager attribute to ENABLED, run the following command:

    ```
    ALTER QMGR PSMODE(ENABLED)
    ```

This command starts the queued publish/subscribe interface so that the queue manager deals with all subsequent publish/subscribe processing.

17. If appropriate, enable content-based filtering on at least one execution group. For more information, see "Enabling content-based filtering with publish/subscribe" on page 2224.
18. Restart the broker.

    When the broker restarts, it continues to provide message services other than publish and subscribe functions.

Related tasks:

"Migrating publish/subscribe information to WebSphere MQ" on page 141
These tasks give an overview of how to migrate publish/subscribe information from WebSphere Message Broker or WebSphere Message Broker Version 6.0 to WebSphere MQ.

"Migrating publish/subscribe information from WebSphere MQ to WebSphere Message Broker Version 6.0 or WebSphere Message Broker " on page 150
Complete these tasks to migrate publish/subscribe configuration data from WebSphere MQ to WebSphere Message Broker Version 6.0 or WebSphere Message Broker .

Related reference:

"**mqsimigratecomponents** command" on page 3894
Use the **mqsimigratecomponents** command to migrate a component from a previously installed version of the product to another version on the same computer.

"WebSphere MQ migmbbrk command" on page 142
Migrate the publish/subscribe information from WebSphere Message Broker Version 6.0 or WebSphere Message Broker to WebSphere MQ

Retained publications with headers in MQRFH format

Retained publications in MQRFH format might lose data when migrated to WebSphere MQ.

WebSphere Message Broker applications that communicate with one another using publish/subscribe can do so regardless of the message format that they use. WebSphere Message Broker delivers the message in the format of the subscription and provides automatic conversion to ensure that a subscriber receives messages in the requested format.

WebSphere Message Broker applications generally use the MQRFH2 message header, but it is possible that an application might have used the MQRFH format.

The migration of publish/subscribe information from WebSphere Message Broker to WebSphere MQ requests messages in MQRFH2 format. It is rare for WebSphere Message Broker client applications to use messages in MQRFH format. However, if an application does use retained messages in MQRFH format, it is possible that some truncation of data might occur upon migration. In particular, data passed using the MQPSStringData and MQPSIntData name/value pairs is not migrated.

The migration function checks for two conditions in the data that is returned from the broker:

1. That there is at least one retained message stored in the broker
2. That there is at least one MQRFH subscription

If both these conditions are true, the migration function displays a warning message and writes a warning message in the migration log stating that MQRFH retained publications have been migrated with a possible loss of data.

MQRFH2 retained publications are migrated correctly.

Migrating publish/subscribe information from WebSphere MQ to WebSphere Message Broker Version 6.0 or WebSphere Message Broker

Complete these tasks to migrate publish/subscribe configuration data from WebSphere MQ to WebSphere Message Broker Version 6.0 or WebSphere Message Broker .

Before you begin

Before you start:

For the latest details of all supported levels of hardware and software, visit the WebSphere Message Broker Requirements website.

This task assumes the following prerequisites:

- You want to migrate publish/subscribe support from WebSphere MQ back to WebSphere Message Broker Version 6.0 or WebSphere Message Broker .
- On distributed systems, you have set up and initialized a command environment in which WebSphere MQ and WebSphere Message Broker commands can run.
- The WebSphere MQ queue manager is not currently handling any publish or subscribe messages.

Procedure

1. Run **mqsimigratecomponents** on WebSphere Message Broker Version 7.0, specifying the broker that you want to migrate. For further information about running this command, see "**mqsimigratecomponents** command" on page 3894.

   This step resets the information to your original specifications in WebSphere Message Broker Version 6.0 or WebSphere Message Broker .
2. Use the command: **ALTER QMGR PSMODE(COMPAT)** to set the **PSMODE** queue manager attribute to COMPAT. This step starts the queued publish/subscribe interface so that the queue manager deals with all subsequent publish/subscribe processing.
3. Restart the broker.

Related tasks:

"Migrating publish/subscribe information to WebSphere MQ" on page 141
These tasks give an overview of how to migrate publish/subscribe information from WebSphere Message Broker or WebSphere Message Broker Version 6.0 to WebSphere MQ.

Related reference:

"**mqsimigratecomponents** command" on page 3894
Use the **mqsimigratecomponents** command to migrate a component from a previously installed version of the product to another version on the same computer.

"WebSphere MQ migmbbrk command" on page 142
Migrate the publish/subscribe information from WebSphere Message Broker

Migrating publish/subscribe collectives from WebSphere Message Broker Version 6.0 or WebSphere Message Broker to WebSphere MQ

Complete these tasks to migrate the configuration data from publish/subscribe collectives in WebSphere Message Broker Version 6.0 or WebSphere Message Broker to WebSphere MQ queue-manager clusters.

Before you begin

Before you start:

For the latest details of all supported levels of hardware and software, visit the WebSphere Message Broker Requirements website.

This task assumes the following prerequisites:

- You have installed WebSphere MQ
- You want to migrate publish/subscribe collectives from WebSphere Message Broker Version 6.0 or WebSphere Message Broker to WebSphere MQ queue manager clusters.
- You want to upgrade from WebSphere Message Broker Version 6.0 or WebSphere Message Broker to WebSphere Message Broker Version 7.0.
- On distributed systems, you have set up and initialized a command environment in which WebSphere MQ and WebSphere Message Broker commands can run.
- The WebSphere MQ queue manager is not currently handling any publish or subscribe messages.

You must migrate each broker separately, and if you have a large number of brokers to migrate, you can administer the process from a WebSphere Message Broker WebSphere Message Broker Toolkit, using a Configuration Manager to a mixed environment of WebSphere Message Broker brokers and migrated WebSphere Message Broker Version 7.0 brokers.

This process is for migration only. You cannot administer new WebSphere Message Broker Version 7.0 brokers or update, for example, publish/subscribe topologies, or Access Control Lists

If you inadvertently initiate a publish/subscribe topology deploy, any brokers already migrated to WebSphere Message Broker Version 7.0 ignore this part of the deployment and issue a BIP2082 warning message to the system log.

If you use this mixed environment, you must use the “WebSphere MQ migmbbrk command” on page 142 as part of the migration for each broker.

To migrate each broker separately:

Procedure

1. Remove each broker from the collective.
2. Migrate each broker, as described in “Migrating publish/subscribe from WebSphere Message Broker Version 6.0 or WebSphere Message Broker to WebSphere MQ” on page 146.

3. Restart all the brokers. When you have migrated all the brokers you can place the brokers in a queue manager cluster. See "Setting up a new queue-manager cluster" for more information on how you carry out this procedure.

Related concepts:

"Setting up a new queue-manager cluster"
Initial points to consider when setting up a new queue-manager cluster

Related tasks:

"Migrating publish/subscribe information to WebSphere MQ" on page 141
These tasks give an overview of how to migrate publish/subscribe information from WebSphere Message Broker or WebSphere Message Broker Version 6.0 to WebSphere MQ.

Setting up a new queue-manager cluster

Initial points to consider when setting up a new queue-manager cluster

Scenario:

- You are setting up a new WebSphere MQ network for a chain store. The store has two branches, one in London and one in New York. The data and applications for each store are hosted by systems running separate queue managers. The two queue managers are called LONDON and NEWYORK.
- The inventory application runs on the system in New York, connected to queue manager NEWYORK. The application is driven by the arrival of messages on the INVENTQ queue, hosted by NEWYORK.
- The two queue managers, LONDON and NEWYORK, are to be linked in a cluster called INVENTORY so that they can both put messages to the INVENTQ.
- Examples are given using TCP/IP only.
- Example UNIX systems commands are shown for AIX only.

Carry out the following tasks to set up a new cluster:

1. "Decide on the organization of the cluster and its name" on page 153
2. "Determine which queue managers should hold full repositories" on page 153
3. "Alter the queue-manager definitions to add repository definitions" on page 154
4. "Define the CLUSRCVR channels" on page 154
5. "Define the CLUSSDR channels" on page 155
6. "Define the cluster queue INVENTQ" on page 156
7. "Verify and test the cluster" on page 158

When you have set up your initial cluster you can add further queue managers; see "Adding a new queue manager to a cluster" on page 159 for further information.

For further information about setting up queue-manager clusters, refer to the WebSphere MQ information center.

Note: You can also use one of the wizards supplied with WebSphere MQ Explorer to create a new cluster similar to the one created by this task. Right-click the Queue Manager Clusters folder, then click **New** > **Queue Manager Cluster**, and follow the instructions given in the wizard.

Related tasks:

"Migrating publish/subscribe collectives from WebSphere Message Broker Version 6.0 or WebSphere Message Broker to WebSphere MQ" on page 151
Complete these tasks to migrate the configuration data from publish/subscribe collectives in WebSphere Message Broker Version 6.0 or WebSphere Message Broker to WebSphere MQ queue-manager clusters.

Decide on the organization of the cluster and its name

Start by linking two queue managers into a cluster. You can use this as a basis for expansion at a later date.

You have decided to link the two queue managers, LONDON and NEWYORK, into a cluster. A cluster with only two queue managers offers only marginal benefit over a network that is to use distributed queuing, but is a good way to start and provides scope for future expansion. When you open new branches of your store, you will be able to add the new queue managers to the cluster easily and without any disruption to the existing network. "Adding a new queue manager to a cluster" on page 159 describes how to do this.

For the time being the only application you are running is the inventory application. The cluster name is INVENTORY.

Related concepts:

"Adding a new queue manager to a cluster" on page 159
How to add a queue manager to the cluster you have set up.

"Setting up a new queue-manager cluster" on page 152
Initial points to consider when setting up a new queue-manager cluster

Related tasks:

"Migrating publish/subscribe collectives from WebSphere Message Broker Version 6.0 or WebSphere Message Broker to WebSphere MQ" on page 151
Complete these tasks to migrate the configuration data from publish/subscribe collectives in WebSphere Message Broker Version 6.0 or WebSphere Message Broker to WebSphere MQ queue-manager clusters.

Determine which queue managers should hold full repositories

In any cluster you need to nominate at least one queue manager, or preferably two, to hold full repositories.

See "Selecting queue managers to hold full repositories" on page 161 for more information. In this example there are only two queue managers, LONDON and NEWYORK, both of which hold full repositories.

Note:

1. You can perform the remaining steps in any order.
2. As you proceed through the steps, warning messages might be written to the queue-manager log or the z/OS system console if you have yet to make some expected definitions.

```
Examples of the responses to the commands are shown in a box
like this after each step in this task.
These examples show the responses returned by WebSphere MQ for AIX.
The responses vary on other platforms.
```

3. Before proceeding with these steps, make sure that the queue managers are started.

Related concepts:

"Setting up a new queue-manager cluster" on page 152
Initial points to consider when setting up a new queue-manager cluster

Related tasks:

"Migrating publish/subscribe collectives from WebSphere Message Broker Version 6.0 or WebSphere Message Broker to WebSphere MQ" on page 151
Complete these tasks to migrate the configuration data from publish/subscribe collectives in WebSphere Message Broker Version 6.0 or WebSphere Message Broker to WebSphere MQ queue-manager clusters.

Alter the queue-manager definitions to add repository definitions

On each queue manager that is to hold a full repository, you need to alter the queue-manager definition.

To alter the queue-manager definition, use the ALTER QMGR command and specifying the REPOS attribute:

```
ALTER QMGR REPOS(INVENTORY)
```

If you enter:

```
     1 : ALTER QMGR REPOS(INVENTORY)
AMQ8005: Websphere MQ queue manager changed.
```

1. C:\..runmqsc LONDON
2. ALTER QMGR REPOS(INVENTORY) (as shown above)

LONDON will be changed to a full repository.

Note: If you just runmqsc and enter the ALTER QMGR command, the local queue manager will be changed.

Related concepts:

"Setting up a new queue-manager cluster" on page 152
Initial points to consider when setting up a new queue-manager cluster

Related tasks:

"Migrating publish/subscribe collectives from WebSphere Message Broker Version 6.0 or WebSphere Message Broker to WebSphere MQ" on page 151
Complete these tasks to migrate the configuration data from publish/subscribe collectives in WebSphere Message Broker Version 6.0 or WebSphere Message Broker to WebSphere MQ queue-manager clusters.

Define the CLUSRCVR channels

On every queue manager in a cluster you need to define a cluster-receiver channel on which the queue manager can receive messages.

This definition defines the queue manager's connection name and the CLUSTER keyword shows the queue manager's availability to receive messages from other queue managers in the cluster. The queue manager's connection name is stored in the repositories, where other queue managers can refer to it.

Using transport protocol TCP/IP to define the channel:

On the LONDON queue manager, define:

```
DEFINE CHANNEL(TO.LONDON) CHLTYPE(CLUSRCVR) TRPTYPE(TCP)
CONNAME(LONDON.CHSTORE.COM) CLUSTER(INVENTORY)
DESCR('TCP Cluster-receiver channel for queue manager LONDON')
```

```
    1 : DEFINE CHANNEL(TO.LONDON) CHLTYPE(CLUSRCVR) TRPTYPE(TCP)
        CONNAME(LONDON.CHSTORE.COM) CLUSTER(INVENTORY)
        DESCR('TCP Cluster-receiver channel for queue manager LONDON')
AMQ8014: Websphere MQ channel created.
07/09/98  12:56:35  No repositories for cluster 'INVENTORY'
```

In this example the channel name is TO.LONDON, and the connection name (CONNAME) is the network address of the machine the queue manager resides on, which is LONDON.CHSTORE.COM. The network address can be entered as an alphanumeric DNS hostname, or an IP address in either in IPv4 dotted decimal form (for example 9.20.9.30) or IPv6 hexadecimal form (for example fe80:43e4:0204:acff:fe97:2c34:fde0:3485). Do not allow the CONNAME to specify a generic name. The port number is not specified, so the default port (1414) is used.

On the NEWYORK queue manager, define:

```
DEFINE CHANNEL(TO.NEWYORK) CHLTYPE(CLUSRCVR) TRPTYPE(TCP)
CONNAME(NEWYORK.CHSTORE.COM) CLUSTER(INVENTORY)
DESCR('TCP Cluster-receiver channel for queue manager NEWYORK')
```

Related concepts:

"Setting up a new queue-manager cluster" on page 152
Initial points to consider when setting up a new queue-manager cluster

Related tasks:

"Migrating publish/subscribe collectives from WebSphere Message Broker Version 6.0 or WebSphere Message Broker to WebSphere MQ" on page 151
Complete these tasks to migrate the configuration data from publish/subscribe collectives in WebSphere Message Broker Version 6.0 or WebSphere Message Broker to WebSphere MQ queue-manager clusters.

Define the CLUSSDR channels

On every queue manager in a cluster you need to define one cluster-sender channel on which the queue manager can send messages to one of the full repository queue managers.

In this case there are only two queue managers, both of which hold full repositories. They must each have a CLUSSDR definition that points to the CLUSRCVR channel defined at the other queue manager. Note that the channel names given on the CLUSSDR definitions must match those on the corresponding CLUSRCVR definitions.

Using transport protocol TCP/IP:

On the LONDON queue manager, define:

```
DEFINE CHANNEL(TO.NEWYORK) CHLTYPE(CLUSSDR) TRPTYPE(TCP)
CONNAME(NEWYORK.CHSTORE.COM) CLUSTER(INVENTORY)
DESCR('TCP Cluster-sender channel from LONDON to repository at NEWYORK')
```

```
    1 : DEFINE CHANNEL(TO.NEWYORK) CHLTYPE(CLUSSDR) TRPTYPE(TCP)
        CONNAME(NEWYORK.CHSTORE.COM) CLUSTER(INVENTORY)
        DESCR('TCP Cluster-sender channel from LONDON to repository at NEWYORK')
AMQ8014: Websphere MQ channel created.
07/09/98  13:00:18    Channel program started.
```

On the NEWYORK queue manager, define:

```
DEFINE CHANNEL(TO.LONDON) CHLTYPE(CLUSSDR) TRPTYPE(TCP)
CONNAME(LONDON.CHSTORE.COM) CLUSTER(INVENTORY)
DESCR('TCP Cluster-sender channel from NEWYORK to repository at LONDON')
```

Related concepts:

"Setting up a new queue-manager cluster" on page 152
Initial points to consider when setting up a new queue-manager cluster

Related tasks:

"Migrating publish/subscribe collectives from WebSphere Message Broker Version 6.0 or WebSphere Message Broker to WebSphere MQ" on page 151
Complete these tasks to migrate the configuration data from publish/subscribe collectives in WebSphere Message Broker Version 6.0 or WebSphere Message Broker to WebSphere MQ queue-manager clusters.

Define the cluster queue INVENTQ

Define the INVENTQ queue on the NEWYORK queue manager, specifying the CLUSTER keyword.

```
DEFINE QLOCAL(INVENTQ) CLUSTER(INVENTORY)
```

```
    1 : DEFINE QLOCAL(INVENTQ) CLUSTER(INVENTORY)
AMQ8006: Websphere MQ queue created.
```

The CLUSTER keyword causes the queue to be advertised to the cluster. As soon as the queue is defined it becomes available to the other queue managers in the cluster. They can send messages to it without having to make a remote-queue definition for it.

Now that you have completed all the definitions, if you have not already done so start the channel initiator on WebSphere MQ for z/OS and, on all platforms, start a listener program on each queue manager. The listener program listens for incoming network requests and starts the cluster-receiver channel when it is needed.

All cluster queue managers need a channel initiator to monitor the system-defined initiation queue SYSTEM.CHANNEL.INITQ and a channel listener program on each queue manager. A channel listener program 'listens' for incoming network requests and starts the appropriate receiver channel when it is needed; see "Channel listener" on page 157 for more information.

Channel initiator: This is the initiation queue for all transmission queues including the cluster transmission queue.

WebSphere MQ for z/OS

There is one channel initiator for each queue manager and it runs as a separate address space. You start it using the MQSC START CHINIT command, which you issue as part of your queue manager startup.

Platforms other than z/OS

When you start a queue manager, a channel initiator is automatically started if the queue manager attribute SCHINIT is set to QMGR. Otherwise it can be started using the MQSC START CHINIT command or the runmqchi control command.

Channel listener:

The implementation of channel listeners is platform specific, however there are some common features. On all WebSphere MQ platforms, the listener can be started using the MQSC command START LISTENER. On Windows systems and UNIX systems, you can make the listener start automatically with the queue manager by setting the CONTROL attribute of the LISTENER object to QMGR or STARTONLY.

Platform-specific details follow:

WebSphere MQ for z/OS

Use the channel listener program provided by WebSphere MQ. To start a WebSphere MQ channel listener, use the MQSC command START LISTENER, which you issue as part of your channel initiator startup. For example:

```
START LISTENER PORT(1414) TRPTYPE(TCP)
```

As well as a listener for each queue manager, members of a queue-sharing group can make use of a shared listener. Do not use shared listeners in conjunction with clusters. Specifically, do not make the CONNAME of the queue manager's CLUSRCVR channel the address of the queue sharing group's shared listener. If you do, queue managers might receive messages for queues for which they do not have a definition.

WebSphere MQ for Windows

Use either the channel listener program provided by WebSphere MQ, or the facilities provided by the operating system.

To start the WebSphere MQ channel listener use the RUNMQLSR command. For example:

```
RUNMQLSR -t tcp -p 1414 -m QM1
```

WebSphere MQ on UNIX systems

Use either the channel listener program provided by WebSphere MQ, or the facilities provided by the operating system (for example, inetd for TCP communications).

To start the WebSphere MQ channel listener use the runmqlsr command. For example:

```
runmqlsr -t tcp -p 1414 -m QM1
```

To use inetd to start channels, configure two files:

1. Edit the file /etc/services. (To do this you must be logged in as a superuser or root.) If you do not have the following line in that file, add it as shown:

   ```
   MQSeries       1414/tcp      # Websphere MQ channel listener
   ```

   where 1414 is the port number required by WebSphere MQ. You can change this, but it must match the port number specified at the sending end.

2. Edit the file /etc/inetd.conf. If you do not have the following line in that file, add it as shown:

   For AIX:

   ```
   MQSeries stream tcp nowait mqm /usr/mqm/bin/amqcrsta amqcrsta
   -m queue.manager.name
   ```

   The updates become active after inetd has reread the configuration files. Issue the following commands from the root user ID:

   On AIX:

   ```
   refresh -s inetd
   ```

You need to delete the remote queue definition at LONDON for the INVENTQ queue.

As the INVENTQ is defined to the cluster, the queue managers no longer need remote-queue definitions for the INVENTQ. At every queue manager, issue the command:

```
DELETE QREMOTE(INVENTQ)
```

Until you do this, the remote-queue definitions will continue to be used and you will not get the benefit of using clusters.

Related concepts:

"Setting up a new queue-manager cluster" on page 152
Initial points to consider when setting up a new queue-manager cluster

Related tasks:

"Migrating publish/subscribe collectives from WebSphere Message Broker Version 6.0 or WebSphere Message Broker to WebSphere MQ" on page 151
Complete these tasks to migrate the configuration data from publish/subscribe collectives in WebSphere Message Broker Version 6.0 or WebSphere Message Broker to WebSphere MQ queue-manager clusters.

Verify and test the cluster

Issue some DISPLAY commands to verify the cluster that you have set up.

The responses you see should be similar to those shown in the examples that follow.

From the NEWYORK queue manager, issue the command:

```
dis clusqmgr(*)
```

```
     1 : dis clusqmgr(*)
AMQ8441: Display Cluster Queue Manager details.
   CLUSQMGR(NEWYORK)              CLUSTER(INVENTORY)
   CHANNEL(TO.NEWYORK)
AMQ8441: Display Cluster Queue Manager details.
   CLUSQMGR(LONDON)               CLUSTER(INVENTORY)
   CHANNEL(TO.LONDON)
```

Now issue the corresponding DISPLAY CHANNEL STATUS command:

```
dis chstatus(*)
```

```
     1 : dis chstatus(*)
AMQ8417: Display Channel Status details.
   CHANNEL(TO.NEWYORK)           XMITQ( )
   CONNAME(9.20.40.24)           CURRENT
   CHLTYPE(CLUSRCVR)             STATUS(RUNNING)
   RQMNAME(LONDON)
AMQ8417: Display Channel Status details.
   CHANNEL(TO.LONDON)            XMITQ(SYSTEM.CLUSTER.TRANSMIT.QUEUE)
   CONNAME(9.20.51.25)           CURRENT
   CHLTYPE(CLUSSDR)              STATUS(RUNNING)
   RQMNAME(LONDON)
```

Because the INVENTQ queue has been advertised to the cluster there is no need for remote-queue definitions. Applications running on NEWYORK and applications running on LONDON can put messages to the INVENTQ queue. They can receive responses to their messages by providing a reply-to queue and specifying its name when they put messages.

At every queue manager, issue the command:

```
DELETE QREMOTE(INVENTQ)
```

Note: The definition for the local queue LONDON_reply does not need the CLUSTER attribute. NEWYORK replies to this queue by explicitly specifying the queue manager name.

Related concepts:

"Setting up a new queue-manager cluster" on page 152
Initial points to consider when setting up a new queue-manager cluster

Related tasks:

"Migrating publish/subscribe collectives from WebSphere Message Broker Version 6.0 or WebSphere Message Broker to WebSphere MQ" on page 151
Complete these tasks to migrate the configuration data from publish/subscribe collectives in WebSphere Message Broker Version 6.0 or WebSphere Message Broker to WebSphere MQ queue-manager clusters.

Adding a new queue manager to a cluster

How to add a queue manager to the cluster you have set up.

Scenario:

- The INVENTORY cluster has been set up as described in "Setting up a new queue-manager cluster" on page 152. It contains two queue managers, LONDON and NEWYORK, which both hold full repositories.
- A new branch of the chain store is being set up in Paris and you want to add a queue manager called PARIS to the cluster.
- Queue manager PARIS will send inventory updates to the application running on the system in New York by putting messages on the INVENTQ queue.
- Network connectivity exists between all three systems.
- The network protocol is TCP.

Procedure

Follow these steps to add a new queue manager to a cluster.

1. Determine which full repository PARIS should refer to first

Every queue manager in a cluster must refer to one or other of the full repositories in order to gather information about the cluster and so build up its own partial repository. Choose either of the repositories, because as soon as a new queue manager is added to the cluster it immediately learns about the other repository as well. Information about changes to a queue manager is sent directly to two repositories. In this example we choose to link PARIS to the queue manager LONDON, purely for geographical reasons.

Note: Perform the remaining steps in any order, after queue manager PARIS is started.

2. Define a CLUSRCVR channel on queue manager PARIS

Every queue manager in a cluster needs to define a cluster-receiver channel on which it can receive messages. On PARIS, define:

```
DEFINE CHANNEL(TO.PARIS) CHLTYPE(CLUSRCVR) TRPTYPE(TCP)
CONNAME(PARIS.CHSTORE.COM) CLUSTER(INVENTORY)
DESCR('Cluster-receiver channel for queue manager PARIS')
```

This advertises the queue manager's availability to receive messages from other queue managers in the cluster INVENTORY. There is no need to make definitions on other queue managers for a sending end to the cluster-receiver channel TO.PARIS. These will be made automatically when needed.

3. Define a CLUSSDR channel on queue manager PARIS

Every queue manager in a cluster needs to define one cluster-sender channel on which it can send messages to its initial full repository. On PARIS, make the following definition for a channel called TO.LONDON to the queue manager whose network address is LONDON.CHSTORE.COM.

```
DEFINE CHANNEL(TO.LONDON) CHLTYPE(CLUSSDR) TRPTYPE(TCP)
CONNAME(LONDON.CHSTORE.COM) CLUSTER(INVENTORY)
DESCR('Cluster-sender channel from PARIS to repository at LONDON')
```

Now that you have completed all the definitions, if you have not already done so, start the channel initiator on WebSphere MQ for z/OS and, on all platforms, start a listener program on queue manager PARIS. The listener program listens for incoming network requests and starts the cluster-receiver channel when it is needed.

By making only two definitions, a CLUSRCVR definition and a CLUSSDR definition, the queue manager PARIS has been added to the cluster.

Now the PARIS queue manager learns, from the full repository at LONDON, that the INVENTQ queue is hosted by queue manager NEWYORK. When an application hosted by the system in Paris tries to put messages to the INVENTQ, PARIS automatically defines a cluster-sender channel to connect to the cluster-receiver channel TO.NEWYORK. The application can receive responses when its queue-manager name is specified as the target queue manager and a reply-to queue is provided.

4. Issue the REFRESH CLUSTER command

If you are adding a queue manager to a cluster that has previously been removed from the same cluster, you must issue the **REFRESH CLUSTER** command on the queue manager you are adding. This completes the task of adding the queue manager.

```
REFRESH CLUSTER(INVENTORY) REPOS(YES)
```

Related concepts:

"Setting up a new queue-manager cluster" on page 152
Initial points to consider when setting up a new queue-manager cluster

Related tasks:

"Migrating publish/subscribe collectives from WebSphere Message Broker Version 6.0 or WebSphere Message Broker to WebSphere MQ" on page 151
Complete these tasks to migrate the configuration data from publish/subscribe collectives in WebSphere Message Broker Version 6.0 or WebSphere Message Broker to WebSphere MQ queue-manager clusters.

Selecting queue managers to hold full repositories

In each cluster you must select at least one, preferably two, or possibly more of the queue managers to hold full repositories.

A cluster can work quite adequately with only one full repository but using two improves availability. You interconnect the full repository queue managers by defining cluster-sender channels between them.

- The most important consideration is that the queue managers chosen to hold full repositories need to be reliable and well managed. For example, it would be far better to choose queue managers on a stable z/OS system than queue managers on a portable personal computer that is frequently disconnected from the network.
- You should also consider the planned outages of the systems hosting your full repositories, and ensure that they do not have coinciding outages.
- You might also consider the location of the queue managers and choose ones that are in a central position geographically or perhaps ones that are located on the same system as a number of other queue managers in the cluster.
- Another consideration might be whether a queue manager already holds the full repositories for other clusters. Having made the decision once, and made the necessary definitions to set up a queue manager as a full repository for one cluster, you might choose to rely on the same queue manager to hold the full repositories for other clusters of which it is a member.

When a queue manager sends out information about itself or requests information about another queue manager, the information or request is sent to two full repositories. A full repository named on a CLUSSDR definition handles the request whenever possible, but if the chosen full repository is not available another full repository is used. When the first full repository becomes available again it collects the latest new and changed information from the others so that they keep in step.

If all of the full repository queue managers go out of service at the same time, queue managers continue to work using the information they have in their partial repositories. The repositories are limited to using the information that they have. New information and requests for updates cannot be processed. When the full repository queue managers reconnect to the network, messages are exchanged to bring all repositories (both full and partial) back up-to-date.

The full repositories republish the publications they receive through the manually-defined CLUSSDR channels, which must point to other full repositories in the cluster. You must make sure that a publication received by any full repository ultimately reaches all of the other full repositories. This is done by manually defining CLUSSDR channels between the full repositories. Having more interconnection of full repositories will make the cluster more robust.

Having only two full repositories is sufficient for all but very exceptional circumstances

Related concepts:

"Setting up a new queue-manager cluster" on page 152
Initial points to consider when setting up a new queue-manager cluster

Related tasks:

"Migrating publish/subscribe collectives from WebSphere Message Broker Version 6.0 or WebSphere Message Broker to WebSphere MQ" on page 151
Complete these tasks to migrate the configuration data from publish/subscribe collectives in WebSphere Message Broker Version 6.0 or WebSphere Message Broker to WebSphere MQ queue-manager clusters.

Migrating existing z/OS applications

You must migrate your existing z/OS applications to be 64-bit before you can use them in WebSphere Message Broker Version 7.0.

You must recompile and link your C code and user exits on z/OS using XPLINK, ensuring that you use the:

- `LP64` flag
- IEEE Floating Point format

Therefore, the options you require are:

- LP64
- FLOAT (IEEE) - the default when you use LP64
- XPLINK - the default when you use LP64

When you make WebSphere MQ calls from within your C code or user exits, you need to link against the following stub:

```
++WMQHLQ++.SCSQDEFS(CSQBMB2X)
```

where `++WMQHLQ++` refers to the high level qualifier for the WebSphere MQ datasets.

You must use the `cmqcmb.h` header file sipped in WebSphere Message Broker Version 7.0 to recompile code that uses WebSphere MQ structures.

Using the `cmqc.h` header file provided by WebSphere MQ results in compilation errors, as `cmqc.h` explicitly disallows 64-bit compilation.

Note, that compilation using the `cmqcmb.h` header file, and linking against `CSQBMB2X` are supported only for user extensions running within the WebSphere Message Broker Version 7.0 address space.

See "Compiling a C user-defined extension" on page 3047 for examples of 64-bit compilation.

Related tasks:

Chapter 3, “Migrating and upgrading,” on page 137
To migrate a broker domain to WebSphere Message Broker Version 7.0, plan your migration strategy, perform pre-migration tasks, migrate your domain components, and then complete post-migration tasks.

“Compiling a C user-defined extension” on page 3047
Compile user-defined extensions in C for all supported operating systems.

Migrating from Version 6.1 products

Migrate your components and resources to WebSphere Message Broker Version 7.0.

Before you begin

Before you start:

Read “Preparing for migration from Version 6.1” on page 165.

About this task

You can migrate to WebSphere Message Broker Version 7.0 from the following products:

- WebSphere Message Broker
- WebSphere Message Broker with Rules and Formatter Extension

For full product version and release levels, see “Supported migration paths” on page 3579.

For the latest details of all supported levels of hardware and software, visit the WebSphere Message Broker Requirements website.

The instructions in this section apply to all operating systems that are supported by Version 6.1.

You can migrate to WebSphere Message Broker Version 7.0 only if you do not use the additional features provided by Rules and Formatter, or you choose not to use them after migration. If you want to use the additional features provided by Rules and Formatter, you can migrate to WebSphere Message Broker Version 7.0.0.6 with WebSphere Message Broker with Rules and Formatter Extension Version 7.0.

Complete the following migration tasks in the order shown to reduce the possibility of problems:

1. Back up your components and resources to ensure that you can return to your previous version if necessary. See the information center for your current product for more information and instructions.

   Consider backing up the following resources:

   - The internal configuration repository maintained by the Configuration Manager.
   - The broker database.
   - Other critical databases that are accessed by your message flows.
   - All your development resources; for example, message flows and message sets.

If you use a repository to manage these resources, check that the product you use provides sufficient features for you to recover your resources at a specific version.

2. Migrate your WebSphere MQ installation to a supported version.
3. Optional: In WebSphere Message Broker Version 7.0, publish/subscribe is controlled by WebSphere MQ. If your applications use publish/subscribe functions, and you have message flows that include one or more Publication nodes, you must migrate your subscriptions to WebSphere MQ before you migrate your broker.

   Information about how to run the **migmqbrk** command to migrate and complete other tasks that might be required, is provided in "Migrating publish/subscribe information to WebSphere MQ" on page 141.

   When you complete this step, all your subscriptions are migrated to WebSphere MQ; subsequently, you must use WebSphere MQ facilities to change subscriptions. A subscription related to the SYSTEM.BROKER.ADMIN.REPLY queue has to be removed manually.

   If you are using cloned WebSphere Message Broker support in Version 6.1, you must change to use WebSphere MQ Clustering for Publish and Subscribe.
4. Migrate the WebSphere Message Broker Toolkit.

   If the users of your toolkit operate in a team environment and share resources with each other, upgrade all users to Version 7.0 at the same time to ensure continued access to all resources. Because toolkit resources are stored in a different format when they are first saved in Version 7.0, your users might experience compatibility problems in some circumstances if some of their colleagues are using Version 7.0 and they are still working with a previous version.
5. Migrate the broker.
6. Optional: If you are migrating from a secure domain, you cannot directly migrate security settings for brokers. You must set up equivalent security by using the facilities provided by Version 7.0. To set up administration security for a broker:
   a. Activate broker administration security.
   b. Set up your security based on existing ACLs.
7. Start the broker by using the **mqsistart** command.
8. Review the changes of behavior that are introduced in Version 7.0.
9. Consider the list of post-migration tasks, and follow the guidance provided if these tasks apply to your environment.

Related concepts:

"Coexistence with previous versions and other products" on page 139
WebSphere Message Broker Version 7.0 supports restricted coexistence with previous versions and with other products.

Related tasks:

"Preparing for migration from Version 6.1" on page 165
Plan the order and extent of the migration of components and resources to Version 7.0.

"Migrating publish/subscribe information to WebSphere MQ" on page 141
These tasks give an overview of how to migrate publish/subscribe information from WebSphere Message Broker or WebSphere Message Broker Version 6.0 to WebSphere MQ.

"Post-migration tasks" on page 204
After you have migrated to Version 7.0, finish setting up your environment.

Related reference:

"Supported migration paths" on page 3579
You can migrate to WebSphere Message Broker Version 7.0 from previous versions of the product.

"**mqsistart** command" on page 3965
Use the **mqsistart** command to start the specified broker if all initial verification tests complete successfully.

Related information:

WebSphere Message Broker Information Center online

Preparing for migration from Version 6.1

Plan the order and extent of the migration of components and resources to Version 7.0.

Before you begin

Before you start: Check that your current installation of WebSphere Message Broker Version 6.1 is at a supported level for migration. Details are provided in "Supported migration paths" on page 3579.

For the latest details of all supported levels of hardware and software, visit the WebSphere Message Broker Requirements website.

About this task

Complete the following steps:

Procedure

1. Decide how you want to migrate the WebSphere Message Broker Version 6.1 product components:
   a. Find out what is new in Version 7.0, and learn about new and changed function. These changes might affect how you want to use your migrated components in the future.
   b. Plan your migration of WebSphere MQ to a supported version. It is a requirement that a supported version of WebSphere MQ is installed before installing WebSphere Message Broker Version 7.0.

      Publish/subscribe functions are no longer supported in the broker; WebSphere MQ provides this support. If your applications use the publish/subscribe communication model, you must migrate your subscriptions to a supported version of WebSphere MQ before you migrate your brokers.
   c. Check the requirements for other products on which Version 7.0 components might depend. If you have configured your message flows to use external resources, such as databases, or event monitoring applications, you might have to modify your configuration. You can find details of supported versions of optional products on the WebSphere Message Broker Requirements web page.
   d. Decide where you will migrate the product components; you can migrate them to a different location on the same computer or to a second computer. For example, you might want to migrate components to another location to maintain availability during the migration.

e. Decide when to migrate the product components. You might want to preserve some components at the Version 6.1 level for now, and migrate them later. Each migration task includes steps for installing WebSphere Message Broker Version 7.0, so you might decide to carry out the tasks in parallel, rather than completing one task before starting the next task.

f. Decide the order in which you will migrate your components. No required order is defined, but your specific circumstances might mean that you decide to migrate components in a particular order.

   A typical order might be:

   1) Install the WebSphere Message Broker Toolkit Version 7.0 on a single computer.
   2) Create a Version 7.0 broker, or migrate an existing broker to Version 7.0.
   3) Migrate further brokers and toolkits.

"Coexistence with previous versions and other products" on page 139 explains how WebSphere Message Broker Version 7.0 can coexist on the same computer with previous versions of the product. It also describes the extent to which Version 7.0 components can operate with components from previous versions.

2. Decide how you will use your existing development resources with WebSphere Message Broker Version 7.0.

   You do not have to perform specific tasks to migrate your development and deployment resources, such as message flow files, message set definition files, ESQL files, XML Schema files, and broker archive files. You can start using these resources with WebSphere Message Broker Version 7.0 immediately.

   However, some migration actions are performed automatically when you open or rebuild resources in the WebSphere Message Broker Toolkit; see "Migrating the WebSphere Message Broker Toolkit development resources from Version 6.1 to Version 7.0" on page 170 for details.

3. Decide what testing you will do to ensure a successful migration.

   The purpose of testing your migration is to identify problems that might arise during migration. For example, if problems arise you might need to restore some migrated resources to the Version 6.1 level that you backed up before you started the migration; all post-migration changes to these resources are lost in this situation. If you migrate your development and test domains before you migrate your production domain, you can identify potential problems and develop a strategy for dealing with further problems.

   Each new release can include changes to address product defects that affect external behavior. If your resources depend on undocumented or incorrect behavior (for example, the ESQL code in a Compute node), you might need to make changes and test these resources to understand the implications in your business scenarios. Read the guidance provided in "Reviewing technical changes in Version 7.0" on page 205 to see if your configuration is affected.

4. Optional: When you are ready to migrate, run the **mqsimigratecomponents** command with the **-c** parameter. Use this form of the command to run a premigration check against the Version 6.1 components to ensure that they can be migrated. The premigration check identifies potential problems so that you can correct them before you continue with migration.

5. Optional: Consider adding time to your migration plan to update development resources in response to changes in product behavior. If you have identified changes that you must make to one or more of your resources in step 3, you must allow time during your migration schedule to make the necessary changes and test those applications.

What to do next

Next: After you have planned your migration, back up your resources.

Related concepts:

"Coexistence with previous versions and other products" on page 139
WebSphere Message Broker Version 7.0 supports restricted coexistence with previous versions and with other products.

Related tasks:

"Migrating from Version 6.1 products" on page 163
Migrate your components and resources to WebSphere Message Broker Version 7.0.

"Post-migration tasks" on page 204
After you have migrated to Version 7.0, finish setting up your environment.

Related reference:

"Supported migration paths" on page 3579
You can migrate to WebSphere Message Broker Version 7.0 from previous versions of the product.

"**mqsimigratecomponents** command" on page 3894
Use the **mqsimigratecomponents** command to migrate a component from a previously installed version of the product to another version on the same computer.

Related information:

- WebSphere Message Broker Requirements
- WebSphere Message Broker Support
- WebSphere MQ Version 7 Information Center online

Backing up WebSphere Message Broker resources

Back up your resources before you start to migrate components to Version 7.0.

Before you begin

Before you start:

Read "Preparing for migration from Version 6.1" on page 165 to plan your migration strategy.

About this task

Before you carry out migration tasks, back up your WebSphere Message Broker resources by completing the following steps:

Procedure

1. Back up the broker database tables. On z/OS, customize and submit the JCL member BIPBUDB. On other platforms, use the documented procedures that are provided by your database supplier.
2. Optional: If your message flows access user databases through an ODBC connection, back up the ODBC files that you use for these connections. Take a copy of these files and store them safely in a different location.
3. Optional: If you have created or configured configurable services for your message flows, record the properties of all your configurable services. For example, a message flow might access user databases through a JDBC

connection, for which you have set up a JDBCProvider configurable service. Run the **mqsireportproperties** for these services, take a copy of the output, and store it safely in a different location.

4. Back up your WebSphere Message Broker Toolkit workspace and resources; for example, message flow files, message set definition files, Java files, ESQL files, mapping files, XML Schema files, and broker archive (BAR) files.
   - Export all your projects from your WebSphere Message Broker Toolkit Version 6.1.
   - Archive your workspace resources:
     - If you manage your workspace resources in a shared repository, for example CVS, follow standard backup procedures for safeguarding versions. Create a version for storing Version 7.0 resources.
     - If you maintain your workspace resources on a local or shared disk, copy your workspace directory to a different location.

Results

For detailed instructions on how to back up these resources, see the WebSphere Message Broker information center.

What to do next

Next: After you have backed up your WebSphere Message Broker resources, update your ODBC definitions.

Related tasks:

"Migrating from Version 6.1 products" on page 163
Migrate your components and resources to WebSphere Message Broker Version 7.0.

Related information:

WebSphere Message Broker Information Center online

Updating ODBC definitions when migrating

Before you migrate a broker, create ODBC definitions for user databases that specify appropriate database drivers for WebSphere Message Broker Version 7.0.

About this task

The database drivers that are supported by Version 7.0 are at a later version than the drivers used by Version 6.1 and Version 6.0.

Complete this update before you run the **mqsimigratecomponents** command for the broker that uses these ODBC connections.

Follow the instructions provided for your operating system:

Windows systems

To change the ODBC connection definitions:

1. Open the ODBC Data Source Administrator window.
2. Open the System DSN page.
3. For each Oracle and Sybase database that is accessed by the broker, associate the data source name with the new ODBC driver:

a. Delete the data source by clicking **Remove**.
b. Re-create the data source with the new ODBC driver by clicking **Add**.

The following table displays the name of the new ODBC driver for each database management system (DBMS).

DBMS	New ODBC driver
Oracle	WebSphere Message Broker DataDirect Technologies 6.00 32-BIT Oracle Wire Protocol
Sybase	WebSphere Message Broker DataDirect Technologies 6.00 32-BIT Sybase Wire Protocol

To change the XA resource manager definitions:

1. Open the **Properties** window of the broker queue manager using the WebSphere MQ Services snap-in.
2. Open the **Resources** page.
3. For each Oracle and Sybase database that participates in a global unit of work, coordinated by the broker queue manager, change the contents of the **SwitchFile** field. For changes to the switch file configuration to take effect, you must restart the broker queue manager.

   The following table specifies what you must change for each database management system (DBMS). *WBIMB* represents the fully qualified path name of the directory in which you have installed WebSphere Message Broker.

DBMS	Change ...	To ...
Oracle	*WBIMB*\bin\ukor8dtc20.dll or*WBIMB*\bin\ukor8dtc22.dll or*WBIMB*\bin\ukor8dtc23.dll	*WBIMB*\bin\ukora24.dll
Sybase	*WBIMB*\bin\ukase20.dll or*WBIMB*\bin\ukase22.dll or*WBIMB*\bin\ukase23.dll	*WBIMB*\bin\ukase24.dll

Linux and UNIX systems

To change the ODBC connection definitions:

Create an ODBC definitions file by following the instructions in "Connecting to a database from Linux and UNIX systems using the DataDirect drivers" on page 674. Before you run the commands at the new service level, check that your ODBCINI environment variable points to the new file and not to the existing file.

To change the XA resource manager definitions:

To change the XA resource manager definitions, edit the queue manager configuration file (`qm.ini`) of the queue manager that is associated with the broker. The `qm.ini` file is located at `/var/mqm/qmgrs/`*queue_manager_name*`/qm.ini`, where *queue_manager_name* is the name of the queue manager that is associated with the broker.

In the *XAResourceManager* stanza for each Oracle and Sybase database that participates in a global unit of work that is coordinated by the broker queue manager, change the entry for the

switch file. For changes to the switch file configuration to take effect, you must restart the broker queue manager.

The following table specifies what you must change for each broker operating system and database management system (DBMS).

DBMS	Change ...	To ...

Oracle	`SwitchFile=UKor8dtc20.so`
	`orSwitchFile=UKor8dtc22.so`
	`orSwitchFile=UKoradtc22.so`
	`orSwitchFile=UKor8dtc23.so`
	`orSwitchFile=UKoradtc23.so`
		`SwitchFile=UKoradtc24.so`

Sybase (not supported on Linux on IBM z Systems)	`SwitchFile=UKasedtc20.so`
	`orSwitchFile=UKasedtc22.so`
	`orSwitchFile=UKasedtc23.so`
		`SwitchFile=UKasedtc24.so`

To check that your ODBC environment is set up correctly on Linux and UNIX systems, run the **mqsicvp** command. This command also validates the connection to all data sources that are listed in the `odbc.ini` file that have been associated with a broker by using the **mqsisetdbparms** command. For more information, see "**mqsicvp** command" on page 3857.

Results

If you revert to a previous version of WebSphere Message Broker, you must reverse the changes that you make to the ODBC definitions. Update your ODBC files after you have run the **mqsimigratecomponents** command, but before you restart the broker at the earlier version.

Related tasks:

"Restoring components and resources to Version 6.1" on page 225
Restore components and resources that you have migrated from Version 6.1 products to their original state.

"Restoring components and resources to Version 6.0" on page 227
Restore components and resources that you have migrated from Version 6.0 products to their original state.

"Enabling ODBC connections to the databases" on page 668
Set up the resources and environment that the broker requires for Open Database Connectivity (ODBC) connections to user databases on distributed systems.

Migrating the WebSphere Message Broker Toolkit development resources from Version 6.1 to Version 7.0

You cannot migrate the WebSphere Message Broker Toolkit component from WebSphere Message Broker Version 6.1, but you can install Version 7.0 to coexist with Version 6.1, and migrate the development resources that you have created in your workspace.

About this task

- You cannot deploy resources from the WebSphere Message Broker Toolkit Version 7.0 to brokers at a previous version that you have not migrated to Version 7.0.
- If any of the following actions have caused an error in the project:
  - Creating the metadata information for the user-defined node project
  - Correcting the plug-in identifier if it does not match the project name
  - Ensuring all the user-defined nodes are in the same category

  The WebSphere Message Broker Toolkit flags the error, and you can use a Quick Fix to rectify the error, see "Applying a Quick Fix to a task list error" on page 2862.However, if you have different user-defined nodes that depend on different resources that have identical names, the broker archive (BAR) file compiler produces an error indicating a naming conflict in the dependent resources. Quick Fix cannot be used to resolve BAR file compiler errors.
- You cannot migrate a deployable JAR file for a user-defined node that was created in Version 7.0, or before Version 7.0.

To migrate your WebSphere Message Broker Toolkit workspace to Version 7.0, complete the following steps:

Procedure

1. Install WebSphere Message Broker Toolkit Version 7.0 in a different location from WebSphere Message Broker Toolkit Version 6.1 or Version 6.0. For detailed instructions, see "Installing the WebSphere Message Broker Toolkit" on page 276, or refer to the "Installation Guide" on page 233.
2. When you start the WebSphere Message Broker Toolkit Version 7.0 for the first time, you are prompted to enter a workspace location. Enter the directory that contains the WebSphere Message Broker Toolkit Version 6.1 or Version 6.0 workspace that you want to migrate and click **OK**. The workspace and resources are now available in the WebSphere Message Broker Toolkit Version 7.0.
3. In the Broker Application Development perspective, select **Window** > **Reset Perspective** to ensure that all views and menus are updated for Version 7.0.
4. You must migrate any user-defined node source projects that were developed in Version 7.0, or before Version 7.0. An error is displayed on all projects that require migration. You can use a Quick Fix to clear these errors, see "Applying a Quick Fix to a task list error" on page 2862. After fixing the errors, you must package the user-defined node source projects and redistribute to the users, see "Packaging and distributing a user-defined node project" on page 3121.

Results

You can now view and modify existing resources, and create resources. You can deploy your workspace resources to Version 7.0 brokers.

What to do next

When you have migrated the resources associated with the WebSphere Message Broker Toolkit, you must migrate your brokers, see "Migrating a broker from WebSphere Message Broker to WebSphere Message Broker Version 7.0" on page 172.

When you have completed these tasks, for information about tasks that you might want to perform after migration, see "Post-migration tasks" on page 204.

Related concepts:

"WebSphere Message Broker Toolkit" on page 31
The WebSphere Message Broker Toolkit is an integrated development environment and graphical user interface based on the Eclipse platform.

Related tasks:

"Migrating from Version 6.1 products" on page 163
Migrate your components and resources to WebSphere Message Broker Version 7.0.

"Migrating a broker from WebSphere Message Broker to WebSphere Message Broker Version 7.0"
Migrate one or more brokers to Version 7.0.

"Post-migration tasks" on page 204
After you have migrated to Version 7.0, finish setting up your environment.

Migrating a broker from WebSphere Message Broker to WebSphere Message Broker Version 7.0

Migrate one or more brokers to Version 7.0.

Before you begin

Before you start: Read "Preparing for migration from Version 6.1" on page 165.

About this task

To migrate a broker from WebSphere Message Broker to WebSphere Message Broker Version 7.0, see the appropriate topic for your operating system:

- "Migrating a Version 6.1 broker to Version 7.0 on distributed operating systems" on page 173
- "Migrating a Version 6.1 broker to Version 7.0 on z/OS" on page 177

Results

You do not have to redeploy resources to a broker that you have migrated. When you start the broker, it starts all existing message flows.

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

"Coexistence with previous versions and other products" on page 139
WebSphere Message Broker Version 7.0 supports restricted coexistence with previous versions and with other products.

Related tasks:

"Migrating from Version 6.1 products" on page 163
Migrate your components and resources to WebSphere Message Broker Version 7.0.

"Post-migration tasks" on page 204
After you have migrated to Version 7.0, finish setting up your environment.

Related reference:

"Supported migration paths" on page 3579
You can migrate to WebSphere Message Broker Version 7.0 from previous versions of the product.

Migrating a Version 6.1 broker to Version 7.0 on distributed operating systems

Migrate a broker to use the enhanced facilities available in Version 7.0.

Before you begin

Before you start:

- Back up your broker resources.
- For the latest details of all supported levels of hardware and software, visit the WebSphere Message Broker Requirements website.
- Install WebSphere MQ and migrate your queue managers and other resources, following the instructions provided by WebSphere MQ.
- Optional: If your applications use publish/subscribe functions, and you have message flows that include one or more Publication nodes, you must migrate your subscriptions to WebSphere MQ before you migrate the broker.
- Migrate the WebSphere Message Broker Toolkit from Version 6.1 to Version 7.0.
- Check that you have no aggregations in progress on this broker. When you migrate a broker to Version 7.0, all live data that is being stored for aggregations in progress is lost.
- If the broker runs in a locale that is not listed in "Locales" on page 3629, check that the code page is one of the supported code pages, and that the locale is set up correctly.

About this task

If you stop the broker, you can migrate it immediately to the new version on the same computer. If you prefer not to stop the broker to avoid problems for your business applications, or if you want to reproduce the broker function on another computer, you can associate the application logic on your Version 6.1 broker with a separate Version 7.0 broker.

Select the topic that is appropriate to your environment:

- "Migrating a Version 6.1 broker to Version 7.0"
- "Migrating application logic to a Version 7.0 broker" on page 174

Migrating a Version 6.1 broker to Version 7.0:
About this task

To migrate a Version 6.1 broker on a distributed operating system to Version 7.0 on the same computer, complete the following steps:

Procedure

1. Install WebSphere Message Broker Version 7.0 on the same computer as Version 6.1. Install at least the Broker component; other components are optional. You must specify a new location for this installation.
2. Open the WebSphere Message Broker Toolkit Version 6.1, and remove the broker from the domain configuration.
3. Stop all channels that are connected to the Version 6.1 broker.

4. Open a Version 6.1 command environment, and stop the Version 6.1 broker by using the **mqsistop** command.
5. Optional: If your message flows access user databases by using ODBC connections, update the ODBC definitions to Version 7.0 format.
6. Set up the correct Version 7.0 command environment:
   - Linux UNIX On Linux and UNIX systems, open a new shell and run the environment profile **mqsiprofile** for this Version 7.0 installation.
   - Windows On Windows, click **Start**, and open the Command Console that is associated with this Version 7.0 installation.

     On Windows 7 and Windows Server 2008 systems, you must open a command console with elevated privileges. To open a command console with elevated privileges, use the **mqsicommandconsole** command. For more information, see "**mqsicommandconsole** command" on page 3830.
7. Run the **mqsimigratecomponents** command to migrate the broker. For example:

   ```
   mqsimigratecomponents Broker1
   ```

   All configuration data is retrieved from the Version 6.1 broker database. If you had set a default user ID and password for your Version 6.1 broker by using the **-u** and **-p** options on the **mqsicreatebroker** command, these values are migrated with the broker. You can change these values for your Version 7.0 broker by using the **mqsisetdbparms** command.
8. On the AIX, Linux on x86-64, and Solaris on SPARC platforms, support has changed from both 32-bit and 64-bit support to 64-bit support only. 32-bit execution groups are converted to 64-bit. These platforms now require 64-bit libraries and 64-bit ODBC configurations for databases.
   a. Recompile C/C++ user-defined nodes, parsers, and exits to 64-bit, if the existing ones are 32-bit only.
   b. Configure the broker to point at these recompiled extensions by using the following command:

      ```
      mqsichangebroker -l userLilPath -x userExitPath
      ```

      where *userLilPath* defines one or more paths to your LIL files, and *userExitPath* defines one or more paths to your user exit programs.

   The following changes are made to environment variables, for all platforms:

Previous version	Version 7
MQSI_LILPATH64 and MQSI_LILPATH32	MQSI_LILPATH
MQSI_USER_EXIT_PATH64 and MQSI_USER_EXIT_PATH	MQSI_USER_EXIT_PATH
MQSI_SECURITY_PROVIDER_PATH64 and MQSI_SECURITY_PROVIDER_PATH32	MQSI_SECURITY_PROVIDER_PATH

   Update all custom profiles that set these environment variables.
9. Copy all additional custom environment settings from your previous environment into your Version 7.0 environment; for example, MQSI_FILENODES_ROOT_DIRECTORY.
10. Start the Version 7.0 broker by using the **mqsistart** command.

Migrating application logic to a Version 7.0 broker:

About this task

To migrate application logic from a Version 6.1 broker on a distributed operating system to a different Version 7.0 broker on the same computer, or on a different computer, complete the following steps:

Procedure

1. Install WebSphere Message Broker Version 7.0 on the same computer as Version 6.1, or on a different computer. Install at least the Broker component; other components are optional. If you are installing on the same computer, you must specify a different location.
2. Optional: If your message flows access user databases by using ODBC connections, update the ODBC definitions to Version 7.0 format.
3. Set up the correct Version 7.0 command environment:
   - Linux UNIX On Linux and UNIX systems, open a new shell and run the environment profile **mqsiprofile** for this Version 7.0 installation.
   - Windows On Windows, click **Start**, and open the Command Console that is associated with this Version 7.0 installation.

     On Windows 7 and Windows Server 2008 systems, you must open a command console with elevated privileges. To open a command console with elevated privileges, use the **mqsicommandconsole** command. For more information, see "**mqsicommandconsole** command" on page 3830.
4. Create a Version 7.0 broker by using the **mqsicreatebroker** command; give it a name and a queue manager name that are different from the names of the Version 6.1 broker and queue manager. If you have installed one or both of these components on this computer, you can also create a local broker by using the WebSphere Message Broker Explorer or the WebSphere Message Broker Toolkit.
5. Start the Version 7.0 broker by using the **mqsistart** command. If you have installed one or both of these components on this computer, you can also start a local broker by using the WebSphere Message Broker Explorer or the WebSphere Message Broker Toolkit.
6. Write a list of the execution groups that you have on the Version 6.1 broker, and create these same execution groups on the Version 7.0 broker. Use the WebSphere Message Broker Explorer, the Version 7.0 WebSphere Message Broker Toolkit, or the **mqsicreateexecutiongroup** command to complete this step. On the AIX, Linux on x86-64, and Solaris on SPARC platforms, you can create only 64-bit execution groups.
7. Deploy the message flows and message sets that are in use by the Version 6.1 broker to the Version 7.0 broker from the Version 7.0 WebSphere Message Broker Toolkit. You cannot complete this step unless you have already migrated the WebSphere Message Broker Toolkit resources.
8. Configure all other relevant properties of the Version 6.1 broker on the Version 7.0 broker.
9. If you want to delete your Version 6.1 broker at this time, or later:
   a. In a Version 6.1 command environment, stop the Version 6.1 broker by using the **mqsistop** command.
   b. Remove the Version 6.1 broker from the Version 6.1 WebSphere Message Broker Toolkit.
   c. Redeploy the Version 6.1 topology from the Version 6.1 WebSphere Message Broker Toolkit to update the configuration data held by the Configuration Manager.

d. In a Version 6.1 command environment, delete the Version 6.1 broker by using the **mqsideletebroker** command.

What to do next

When you have completed these tasks, see the post-migration tasks for information about tasks that you might want to perform after migration.

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

"Coexistence with previous versions and other products" on page 139
WebSphere Message Broker Version 7.0 supports restricted coexistence with previous versions and with other products.

Related tasks:

"Migrating from Version 6.1 products" on page 163
Migrate your components and resources to WebSphere Message Broker Version 7.0.

"Post-migration tasks" on page 204
After you have migrated to Version 7.0, finish setting up your environment.

"Backing up WebSphere Message Broker resources" on page 167
Back up your resources before you start to migrate components to Version 7.0.

"Creating a broker" on page 611
You can create brokers on every platform that is supported by WebSphere Message Broker. The broker runs as a 64-bit application on all platforms except Linux on x86 and Windows on x86.

"Starting and stopping a broker" on page 921
Run the appropriate command to start or stop a broker.

"Creating an execution group using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 937
Use the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer to create execution groups on your broker.

"Creating an execution group using the **mqsicreateexecutiongroup** command" on page 939
Use the **mqsicreateexecutiongroup** command to create execution groups on your broker.

Related reference:

"Supported migration paths" on page 3579
You can migrate to WebSphere Message Broker Version 7.0 from previous versions of the product.

"**mqsicommandconsole** command" on page 3830
Use the **mqsicommandconsole** command to launch an elevated command console from which commands that require elevation on Windows can be run.

"**mqsideletebroker** command" on page 3863
Use the **mqsideletebroker** command to delete a named broker. The command also deletes the queues on the associated queue manager (created when the broker was created). You can also specify that the queue manager is to be deleted.

"**mqsimigratecomponents** command" on page 3894
Use the **mqsimigratecomponents** command to migrate a component from a previously installed version of the product to another version on the same computer.

"**mqsistart** command" on page 3965
Use the **mqsistart** command to start the specified broker if all initial verification tests complete successfully.

"**mqsistop** command" on page 3972
Use the **mqsistop** command to stop the specified component.

Migrating a Version 6.1 broker to Version 7.0 on z/OS

Migrate a broker to use the enhanced facilities available in Version 7.0.

Before you begin

Before you start:

- Backup your broker resources.
- For the latest details of all supported levels of hardware and software, visit the WebSphere Message Broker Requirements website.
- Install WebSphere MQ and migrate your queue managers and other resources, following the instructions provided by WebSphere MQ.
- Optional: If your applications use publish/subscribe functions, and you have message flows that include one or more Publication nodes, you must migrate your subscriptions to WebSphere MQ before you migrate the broker.
- Ensure that you are familiar with the steps involved in creating a broker on z/OS.
- Ensure that you are familiar with the **mqsimigratecomponents** command which the JCL (Job Control Language) file uses to migrate a broker.
- Check that you have no aggregations in progress. When you migrate a broker to Version 7.0, any live data that is being stored for aggregations in progress is lost.
- If the broker runs in a locale that is not listed in "Locales" on page 3629, check that the code page is one of the supported code pages and that the locale is set up correctly.

About this task

To migrate a WebSphere Message Broker Version 6.1 broker to Version 7.0 on z/OS:

Procedure

1. Stop the Version 6.1 broker and the Version 6.1 Configuration Manager.
2. Create a new broker PDSE.
3. Copy all broker JCL files from the Version 7.0 installed SBIPPROC and SBIPSAMP PDSEs to the new broker PDSE. You can then customize these files. Some customization is required for migration, as described in the following sub-steps. You can update these and other files at a later date, if required. See "Customizing the broker JCL" on page 625 for more information.

   You must take a backup copy of your Version 6.1 `ENVFILE` file before you submit the `BIPGEN` job from the Version 7.0 component data set. The `ENVFILE` file is stored in the directory referenced by the ++*HOME*++ JCL variable. Subsequently, if you want to migrate back to Version 6.1, restore the `ENVFILE` file, either from the backup file location, or by submitting the `BIPGEN` job from the Version 6.1 component data set, before you start the Version 6.1 broker.

   a. Customize the `BIPEDIT` file by using the values that are defined in the Version 6.1 `BIPEDIT` file.

b. Copy all additional changes that you have made to the Version 6.1 version of the BIPBPROF and BIPDSNAO files to these files in the Version 7.0 component data set. Submit the BIPGEN job to create the environment file ENVFILE.

c. Customize and submit the BIPMGCMP (**mqsimigratecomponents**) job. This job migrates the registry, queue, and broker database; connection to the broker database of the previous broker is required to complete this action.

   After migration, all information in the broker database is stored in an internal repository in Version 7.0; the Version 7.0 broker does not require a database.

4. Copy the renamed started task JCL file BIPBRKP to the procedures library. When you copy the started task, keep a second copy of the original in a safe place for backup purposes.
5. Start the Version 7.0 broker. The verification program checks the configuration of the broker.

What to do next

When you have completed these tasks, see the post-migration tasks for information about tasks that you might want to perform after migration.

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

"Coexistence with previous versions and other products" on page 139
WebSphere Message Broker Version 7.0 supports restricted coexistence with previous versions and with other products.

Related tasks:

"Migrating from Version 6.1 products" on page 163
Migrate your components and resources to WebSphere Message Broker Version 7.0.

"Post-migration tasks" on page 204
After you have migrated to Version 7.0, finish setting up your environment.

"Backing up WebSphere Message Broker resources" on page 167
Back up your resources before you start to migrate components to Version 7.0.

"Creating a broker on z/OS" on page 620
Create the broker component and the other resources on which it depends.

"Creating the broker PDSE" on page 622
This is part of the larger task of creating a broker on z/OS.

"Customizing the broker JCL" on page 625
This subtask is part of the larger task of creating a broker on z/OS.

"Copying the broker started task to the procedures library" on page 629
This is part of the larger task of creating a broker on z/OS.

"Starting and stopping a broker on z/OS" on page 924
Run the appropriate command from SDSF to start or stop a broker.

Related reference:

"Supported migration paths" on page 3579
You can migrate to WebSphere Message Broker Version 7.0 from previous versions of the product.

"Sample BIPEDIT file" on page 4009
The sample BIPEDIT file that is shipped with WebSphere Message Broker is

included here for your reference.

"**mqsimigratecomponents** command" on page 3894
Use the **mqsimigratecomponents** command to migrate a component from a previously installed version of the product to another version on the same computer.

Migrating Configuration Manager ACLs

If you are migrating from WebSphere Message Broker , WebSphere Message Broker Version 6.0, or WebSphere Event Broker Version 6.0, you can use the Access Control Lists (ACLs) that you set up in the Configuration Manager as the basis for your security model in Version 7.0.

Before you begin

Before you start:

- Migrate your broker by following the instructions in "Migrating a broker from WebSphere Message Broker to WebSphere Message Broker Version 7.0" on page 172 or "Migrating a broker from WebSphere Message Broker Version 6.0 to WebSphere Message Broker Version 7.0" on page 193.
- Activate broker administration security by following the instructions in "Enabling broker administration security" on page 368.

Administration security for a broker in Version 7.0 is disabled by default. Because no security details are migrated, when you migrate a broker from a previous version security is disabled, even if you had set up security in the previous version.

About this task

The Configuration Manager has been removed as a result of the architectural and security model changes in Version 7.0, therefore the commands that controlled this component and its ACLs have been removed:

mqsichangeconfigmgr
mqsicreateconfigmgr
mqsideleteconfigmgr
mqsireportconfigmgr
mqsicreateaclentry
mqsideleteaclentry
mqsilistaclentry

Although you can use the ACLs that you set up in previous versions as the basis for your security model in Version 7.0, you might choose to reconsider your security model, and set up different levels of control that match the facilities available in this version.

If you choose to use your existing ACLs as a basis for your security model in Version 7.0, consider the following factors:

- You can obtain a list of existing ACLs maintained by your Configuration Manager by using the **mqsilistaclentry** command.
- You can migrate existing groups, users, or both, subject to the following factors:
  - You must define the user ID on the same computer as the queue manager associated with the Version 7.0 broker.
  - On Linux and UNIX systems, you can grant authority only to the primary group of a user. All users that you have defined with the same primary group

automatically get the same level of security access. You must therefore consider the membership of all your groups to ensure that you give the required level of control to each user.

- You can grant only broker and execution level authorities for Version 7.0 brokers.
- If you grant read, write, and execute authority to a user ID for a Version 7.0 broker, this permission is equivalent to full control access in previous versions.
- If you grant read authority to a user ID for a Version 7.0 broker, this permission is equivalent to view access in previous versions.
- Check the authority that write and execute permissions grant for a Version 7.0 broker to determine the best match for edit and deploy access levels in previous versions.
- Although you can set up access for a particular computer or domain name in previous versions, you can use only user IDs and groups in Version 7.0. If you want to establish a more secure environment, consider the use of WebSphere MQ security exits and SSL.

To set up authorization for your Version 7.0 broker, complete the following steps.

Procedure

1. Review the authorizations that are required for specific tasks and commands. Details are provided in "Tasks and authorizations for broker administration security" on page 3645 and "Commands and authorizations for broker administration security" on page 3646.
2. Grant the authorities that your users require.
   - For details of this task on distributed systems, with examples, see "Granting and revoking authority on Linux, UNIX, and Windows systems" on page 374.
   - For details of this task on z/OS, with examples, see "Granting and revoking authority on z/OS systems" on page 376.

What to do next

Next: Start the broker in the WebSphere Message Broker Explorer, or run the **mqsistart** command.

Related concepts:

"Security overview" on page 351
When you are designing a WebSphere Message Broker application it is important to consider the security measures that are needed to protect the information in the system.

Related tasks:

"Enabling broker administration security" on page 368
Enable broker administration security on a broker to control which users can complete specific tasks against that broker and its resources.

"Authorizing users for broker administration" on page 371
Grant authority to one or more groups or users to authorize them to complete specific tasks against a broker and its resources.

"Enabling SSL on the WebSphere MQ Java Client" on page 540
The WebSphere MQ Java Client supports SSL-encrypted connections over the server-connection (SVRCONN) channel between an application and the queue manager. Configure SSL support for connections between applications that use the CMP API (including the WebSphere Message Broker Toolkit and the WebSphere

Message Broker Explorer) and a broker.

Related reference:

"Commands" on page 3672
All WebSphere Message Broker Toolkit and runtime commands that are provided on distributed systems are listed, grouped by function, with references to command details.

Related information:

WebSphere MQ Version 7 Information Center online

Migrating a Microsoft Windows WebSphere Message Broker that is configured using Microsoft Cluster Services (MSCS)

You must complete a specific set of steps to migrate your MSCS cluster to WebSphere Message Broker Version 7.0.

Before you begin

Read the topics in the section "Migrating from Version 6.1 products" on page 163, and the topic "Using a broker with an existing high availability manager" on page 843.

About this task

Use the following procedure to migrate a WebSphere Message Broker that is configured using MSCS.

Migrating from WebSphere Message Broker to WebSphere Message Broker Version 7.0 .

Procedure

1. Install WebSphere Message Broker Version 7.0 onto each node. You must specify a new location for this installation.
2. Stop all channels that are connected to the WebSphere Message Broker broker.
3. In MSCS bring the broker resource offline, keeping the cluster disk and WebSphere MQ resource online on your primary node
4. In a WebSphere Message Broker Version 7.0 command console run the following command:

   ```
   mqsimigratecomponents BROKER1 -m
   ```

   where BROKER1 is the name of your broker
5. Move the broker cluster group to the secondary node. Note that the broker resource should remain stopped.
6. In a WebSphere Message Broker Version 7.0 command console run the following command:

   ```
   mqsimigratecomponents BROKER1 -m -1
   ```
7. Copy all additional custom environment settings from your previous environment into your WebSphere Message Broker Version 7.0 environment; for example: MQSI_FILENODES_ROOT_DIRECTORY.
8. Bring the cluster disk and WebSphere MQ resource offline.
9. Start the WebSphere Message Broker cluster group on its primary node.

What to do next

Migrating from WebSphere Message Broker Version 7.0 to WebSphere Message Broker .

1. In MSCS bring the broker resource offline, keeping the cluster disk and WebSphere MQ resource online on your primary node.
2. In a WebSphere Message Broker Version 7.0 command console run the following command:

   ```
   mqsimigratecomponents BROKER1 -s 7.0 -t 6.1
   ```
3. In a WebSphere Message Broker command console run the following command:

   ```
   mqsistart BROKER1
   ```

   This is required to update the Windows service to point at the WebSphere Message Broker binary files.
4. In a WebSphere Message Broker command console run the following command:

   ```
   mqsistop BROKER1
   ```
5. Move the broker cluster group to the secondary node. Note that the broker resource should remain stopped.
6. In a WebSphere Message Broker Version 7.0 command console run the following command:

   ```
   mqsimigratecomponents BROKER1 -m -s 6.1 -t 7 -1
   ```
7. In a WebSphere Message Broker Version 7.0 command console run the following command:

   ```
   mqsimigratecomponents BROKER1 -s 7.0 -t 6.1 -1
   ```
8. In a WebSphere Message Broker command console run the following command:

   ```
   mqsistart BROKER1
   ```

   This is required to update the Windows service to point at the WebSphere Message Broker binary files
9. In a WebSphere Message Broker command console run the following command:

   ```
   mqsistop BROKER1
   ```
10. Bring the cluster disk and WebSphere MQ resource offline.
11. Start the broker cluster group on its primary node.

Due to changes in the file security model used in WebSphere Message Broker Version 7.0, you might need to alter the file permissions of the shared resource.

If necessary, create a domain group, add the `userID` of your broker to this group, and give the group full access to the shared files.

Related tasks:

"Migrating from Version 6.1 products" on page 163
Migrate your components and resources to WebSphere Message Broker Version 7.0.

"Using a broker with an existing high availability manager" on page 843
You can use WebSphere Message Broker Version 7.0 with an existing high availability manager, for example HACMP, HA/XD, VCS, or HP-UX Serviceguard.

Migrating from Version 6.0 products

Migrate your components and resources to WebSphere Message Broker Version 7.0.

About this task

You can migrate to WebSphere Message Broker Version 7.0 products from the following products:

- WebSphere Event Broker Version 6.0
- WebSphere Message Broker Version 6.0
- WebSphere Message Broker with Rules and Formatter Extension Version 6.0

For full product version and release levels, see "Supported migration paths" on page 3579.

For the latest details of all supported levels of hardware and software, visit the WebSphere Message Broker Requirements website.

The instructions in this section apply to all operating systems that are supported by Version 6.0. The instructions are the same for both WebSphere Event Broker Version 6.0 and WebSphere Message Broker Version 6.0; in the interests of readability, only WebSphere Message Broker Version 6.0 is shown.

If your current installation is WebSphere Message Broker with Rules and Formatter Extension Version 6.0, you can migrate to WebSphere Message Broker Version 7.0 only if you do not use the additional features provided by Rules and Formatter, or you choose not to use them after migration. If you want to use the additional features provided by Rules and Formatter, you need to migrate to WebSphere Message Broker Version 7.0.0.1 or later, with WebSphere Message Broker with Rules and Formatter Extension Version 7.0.

Some of the migration tasks that you must complete are dependent on the order in which you start them. Complete these tasks in the order shown, to reduce the possibility of problems:

1. Read "Preparing for migration from Version 6.0" on page 185.
2. Back up your components and resources to ensure that you can return to your previous version, if necessary. See the information center for your current product for more information and instructions.

   Consider backing up the following resources:

   - The internal configuration repository maintained by the Configuration Manager.
   - The broker database.
   - Other critical databases that are accessed by your message flows.
   - All your development resources; for example, message flows and message sets.

     If you use a repository to manage these resources, check that the product you use provides sufficient features for you to recover your resources at a specific version.
3. If you have installed SupportPac IA9Q, you must uninstall it before you install WebSphere Message Broker Version 7.0; see "Migrating from SupportPac IA9Q" on page 190.
4. Migrate your WebSphere MQ installation to a supported version.

5. Optional: If your applications use publish/subscribe functions, and you have message flows that include one or more Publication nodes, you must migrate your subscriptions to WebSphere MQ before you migrate your broker.

   In WebSphere Message Broker Version 7.0, publish/subscribe is controlled by WebSphere MQ.

   Information about how to run the **migmqbrk** command to perform this migration and other tasks that might be required, is provided in "Migrating publish/subscribe information to WebSphere MQ" on page 141.

   When you have completed this step, all your subscriptions have been migrated to WebSphere MQ; in future, you must use WebSphere MQ facilities to change subscriptions.
6. Migrate the WebSphere Message Broker Toolkit.

   If your toolkit users operate in a team environment, and share resources with each other, upgrade all users to Version 7.0 at the same time to ensure continued access to all resources. Because toolkit resources are stored in a different format when they are first saved in Version 7.0, your users might experience compatibility problems in some circumstances if some of their colleagues are using Version 7.0 and they are still working with a previous version.
7. Migrate the broker.
8. Optional: If you are migrating from a secure domain, you cannot directly migrate security settings for brokers. You must set up equivalent security by using the facilities provided by Version 7.0. To set up administration security for a broker:
   a. Activate broker administration security.
   b. Set up your security based on existing ACLs.
9. Start the broker by using the **mqsistart** command.
10. Review the changes of behavior that are introduced in Version 7.0.
11. Consider the list of post-migration tasks, and follow the guidance provided if these tasks apply to your environment.

Related concepts:

"Coexistence with previous versions and other products" on page 139
WebSphere Message Broker Version 7.0 supports restricted coexistence with previous versions and with other products.

Related tasks:

"Preparing for migration from Version 6.0" on page 185
Plan the order and extent of the migration of components and resources to Version 7.0.

"Migrating publish/subscribe information to WebSphere MQ" on page 141
These tasks give an overview of how to migrate publish/subscribe information from WebSphere Message Broker or WebSphere Message Broker Version 6.0 to WebSphere MQ.

"Post-migration tasks" on page 204
After you have migrated to Version 7.0, finish setting up your environment.

Related reference:

"Supported migration paths" on page 3579
You can migrate to WebSphere Message Broker Version 7.0 from previous versions of the product.

"**mqsistart** command" on page 3965
Use the **mqsistart** command to start the specified broker if all initial verification tests complete successfully.

Related information:

WebSphere Message Brokers Version 6.0 Information Center online

Preparing for migration from Version 6.0

Plan the order and extent of the migration of components and resources to Version 7.0.

Before you begin

Before you start: Check that your current installation of WebSphere Message Broker or WebSphere Event Broker Version 6.0 is at a supported level for migration. Details are provided in "Supported migration paths" on page 3579.

For the latest details of all supported levels of hardware and software, visit the WebSphere Message Broker Requirements website.

About this task

Complete the following steps:

Procedure

1. Decide how you want to migrate the WebSphere Message Broker Version 6.0 product components:
   a. Find out what is new in Version 7.0 and learn about new and changed function. These changes might affect how you want to use your migrated components in the future.
   b. Plan your WebSphere MQ migration. A supported version of WebSphere MQ is required; you must install it before you install WebSphere Message Broker Version 7.0.

      Publish/subscribe functions are no longer supported in the broker; WebSphere MQ provides this support. If your applications use the publish/subscribe communication model, you must migrate your subscriptions to WebSphere MQ before you migrate your brokers.
   c. Check the requirements for other products on which Version 7.0 components might depend. If you have configured your message flows to use external resources, such as databases, or event monitoring applications, you might have to modify your configuration. You can find details of supported versions of optional products on the WebSphere Message Broker Requirements Web page.
   d. Decide where you will migrate the product components. Migrate them to a different location on the same computer or to a second computer. For example, you might want to migrate components to another location to maintain availability during the migration.
   e. Decide when to migrate the product components. You might want to preserve some components at the Version 6.0 level for now, and migrate them later. Each migration task includes steps for installing WebSphere Message Broker Version 7.0, so you might decide to carry out the tasks in parallel, rather than completing one task before starting the next task.
   f. Decide the order in which you will migrate your components. You can migrate components in any order, but your specific circumstances might mean that you need to migrate components in a particular order.

      A typical order might be:

1) Install a Version 7.0 toolkit.
2) Create a Version 7.0 broker, or migrate an existing broker to Version 7.0.
3) Migrate further brokers and toolkits.

"Coexistence with previous versions and other products" on page 139 explains how WebSphere Message Broker Version 7.0 can coexist on the same computer with previous versions of the product. It also describes the extent to which Version 7.0 components can operate with components from previous versions.

2. Decide how you will use your existing resources with WebSphere Message Broker Version 7.0.

   You do not have to perform specific tasks to migrate your development and deployment resources, such as message flow files, message set definition files, ESQL files, XML Schema files, and broker archive files. You can start using these resources with WebSphere Message Broker Version 7.0 immediately.

   However, some migration actions are performed automatically when you open or rebuild resources in the WebSphere Message Broker Toolkit; see "Migrating the WebSphere Message Broker Toolkit development resources from Version 6.0 to Version 7.0" on page 192 for details.

3. Decide what testing you will do to ensure a successful migration.

   The purpose of testing your migration is to identify any problems that might arise during migration. For example, if problems arise you might need to restore some migrated resources to the Version 6.0 level that you backed up before you started the migration, and all post-migration changes to these resources are lost. If you migrate your development and test domains before you migrate your production domain, you can identify potential problems and develop a strategy for dealing with further problems.

   Each new release can include changes to address product defects that affect external behavior. If your resources depend on undocumented or incorrect behavior (for example, the ESQL code in a Compute node), you might need to change and test these resources to understand the implications in your business scenarios. Read the guidance provided in "Reviewing technical changes in Version 7.0" on page 205 to see if your configuration is affected.

4. Optional: When you are ready to migrate, run the **mqsimigratecomponents** command with the **-c** parameter. Use this form of the command to run a premigration check against the Version 6.0 components to ensure that they can be migrated. The premigration check identifies potential problems so that you can correct them before you continue with migration.

5. Optional: Consider adding time to your migration plan to update resources in response to changes in product behavior. If you have identified changes that you must make to one or more of your resources in step 3, you must allow time during your migration schedule to make the necessary changes and test those applications.

What to do next

Next: After you have planned your migration, back up your resources.

Related concepts:

"Coexistence with previous versions and other products" on page 139
WebSphere Message Broker Version 7.0 supports restricted coexistence with previous versions and with other products.

Related tasks:

"Migrating from Version 6.0 products" on page 183
Migrate your components and resources to WebSphere Message Broker Version 7.0.

"Post-migration tasks" on page 204
After you have migrated to Version 7.0, finish setting up your environment.

Related reference:

"Supported migration paths" on page 3579
You can migrate to WebSphere Message Broker Version 7.0 from previous versions of the product.

"**mqsimigratecomponents** command" on page 3894
Use the **mqsimigratecomponents** command to migrate a component from a previously installed version of the product to another version on the same computer.

Related information:

WebSphere Message Broker Requirements

WebSphere Message Broker Support

WebSphere MQ Version 7 Information Center online

Backing up WebSphere Message Broker Version 6.0 resources

Back up your resources before you start to migrate components to Version 7.0.

Before you begin

Before you start:

Read "Preparing for migration from Version 6.0" on page 185 to determine your migration strategy.

About this task

Before you carry out migration tasks, back up your WebSphere Message Broker Version 6.0 resources by completing the following steps:

Procedure

1. Back up the configuration repository. Use the **mqsibackupconfigmgr** command.
2. Back up the broker database tables. Use the documented procedures that are provided by your database supplier.
3. Optional: If your message flows access user databases through an ODBC connection, back up the ODBC files that you use for these connections. Take a copy of these files and store them safely in a different location.
4. Optional: If you have created or configured configurable services for your message flows, record the properties of all your configurable services. For example, a message flow might include a JMSInput node, for which you have set up a JMSProviders configurable service. Run the **mqsireportproperties** for these services, take a copy of the output, and store it safely in a different location.
5. Back up your WebSphere Message Broker Toolkit workspace and resources; for example, message flow files, message set definition files, Java files, ESQL files, mapping files, XML Schema files, and broker archive (BAR) files.
   - Export all your projects from your WebSphere Message Broker Toolkit Version 6.0.
   - Archive your workspace resources:

- If you manage your workspace resources in a shared repository, for example CVS, follow standard backup procedures for safeguarding versions. Create a version for storing Version 7.0 resources.
- If you maintain your workspace resources on a local or shared disk, copy your workspace directory to a different location.

Results

For detailed instructions on how to back up these resources, see the WebSphere Message Broker Version 6.0 information center.

What to do next

Next: After you have backed up your WebSphere Message Broker Version 6.0 resources, update your ODBC definitions.

Related tasks:

"Migrating from Version 6.0 products" on page 183
Migrate your components and resources to WebSphere Message Broker Version 7.0.

Related information:

WebSphere Message Brokers Version 6.0 Information Center online

Updating your ODBC definitions when migrating

As part of migrating a broker, create ODBC definitions for user databases that specify appropriate database drivers for Version 7.0.

About this task

The database drivers that are supported by Version 7.0 are at a later version than the drivers used by Version 6.1 and Version 6.0.

Complete this update before you run the **mqsimigratecomponents** command for the broker that uses these ODBC connections.

Follow the instructions provided for your operating system:

Windows systems

To change the ODBC connection definitions:

1. Open the ODBC Data Source Administrator window.
2. Open the System DSN page.
3. For each Oracle and Sybase database that is accessed by the broker, associate the data source name with the new ODBC driver:
   a. Delete the data source by clicking **Remove**.
   b. Re-create the data source with the new ODBC driver by clicking **Add**.

   The following table displays the name of the new ODBC driver for each database management system (DBMS).

DBMS	New ODBC driver
Oracle	WebSphere Message Broker DataDirect Technologies 6.00 32-BIT Oracle Wire Protocol
Sybase	WebSphere Message Broker DataDirect Technologies 6.00 32-BIT Sybase Wire Protocol

To change the XA resource manager definitions:

1. Open the **Properties** window of the broker queue manager using the WebSphere MQ Services snap-in.
2. Open the **Resources** page.
3. For each Oracle and Sybase database that participates in a global unit of work, coordinated by the broker queue manager, change the contents of the **SwitchFile** field. For changes to the switch file configuration to take effect, you must restart the broker queue manager.

   The following table specifies what you must change for each database management system (DBMS). *WBIMB* represents the fully qualified path name of the directory in which you have installed WebSphere Message Broker.

DBMS	Change ...	To ...
Oracle	*WBIMB*\bin\ukor8dtc20.dll or*WBIMB*\bin\ukor8dtc22.dll or*WBIMB*\bin\ukor8dtc23.dll	*WBIMB*\bin\ukora24.dll
Sybase	*WBIMB*\bin\ukase20.dll or*WBIMB*\bin\ukase22.dll or*WBIMB*\bin\ukase23.dll	*WBIMB*\bin\ukase24.dll

Linux and UNIX systems

To change the ODBC connection definitions:

Create an ODBC definitions file by following the instructions in "Connecting to a database from Linux and UNIX systems using the DataDirect drivers" on page 674. Before you run the commands at the new service level, check that your ODBCINI environment variable points to the new file and not to the existing file.

To change the XA resource manager definitions:

To change the XA resource manager definitions, edit the queue manager configuration file (`qm.ini`) of the queue manager that is associated with the broker. The `qm.ini` file is located at `/var/mqm/qmgrs/`*queue_manager_name*`/qm.ini`, where *queue_manager_name* is the name of the queue manager that is associated with the broker.

In the *XAResourceManager* stanza for each Oracle and Sybase database that participates in a global unit of work that is coordinated by the broker queue manager, change the entry for the switch file. For changes to the switch file configuration to take effect, you must restart the broker queue manager.

The following table specifies what you must change for each broker operating system and database management system (DBMS).

DBMS	Change ...	To ...
Oracle	SwitchFile=UKor8dtc20.so orSwitchFile=UKor8dtc22.so orSwitchFile=UKoradtc22.so orSwitchFile=UKor8dtc23.so orSwitchFile=UKoradtc23.so	SwitchFile=UKoradtc24.so
Sybase (not supported on Linux on IBM z Systems)	SwitchFile=UKasedtc20.so orSwitchFile=UKasedtc22.so orSwitchFile=UKasedtc23.so	SwitchFile=UKasedtc24.so

To check that your ODBC environment is set up correctly on Linux and UNIX systems, run the **mqsicvp** command. This command also validates the connection to all data sources that are listed in the `odbc.ini` file that have been associated with a broker by using the **mqsisetdbparms** command. For more information, see "**mqsicvp** command" on page 3857.

Results

If you revert to a previous version of WebSphere Message Broker, you must reverse the changes that you make to the ODBC definitions. Update your ODBC files after you have run the **mqsimigratecomponents** command, but before you restart the broker at the earlier version.

Related tasks:

"Enabling ODBC connections to the databases" on page 668
Set up the resources and environment that the broker requires for Open Database Connectivity (ODBC) connections to user databases on distributed systems.

Migrating from SupportPac IA9Q

If you installed SupportPac IA9Q, which introduced the SRRetrieveITService and SRRetrieveEntity nodes, you must remove this SupportPac before you can upgrade to WebSphere Message Broker Version 7.0.

About this task

If you do not remove SupportPac IA9Q, you might encounter problems when you deploy or use the EndpointLookup and RegistryLookup nodes that replace the SRRetrieveITService and SRRetrieveEntity nodes.

Before you install WebSphere Message Broker Version 7.0, complete the following steps:

Procedure

1. Write down the properties of your SRRetrieveITService and SRRetrieveEntity nodes so that you can use the same values when you create the EndpointLookup and RegistryLookup nodes in WebSphere Message Broker Version 7.0.

2. Remove message flows that contain the SRRetrieveITService and SRRetrieveEntity nodes from your WebSphere Message Broker Version 6.0 brokers.
3. Remove the dynamic Cache Notification component (the Cache Synchronization WebSphere Message Broker BAR file), if you deployed it.
4. Edit the broker classpath in `bin/mqsiprofile.cmd` to remove the following files:
   - `sibc.jms.jar`
   - `sibc.jndi.jar`
   - `ibm-jaxrpc-client.jar`
   - Solaris `sibc.orb.jar` (on the Solaris platform only)

   When you have edited the broker classpath, delete the files from your system.
5. Remove the `ServiceRegistryMessageBrokerNodes_1.0.0.par` file.
6. Remove the following files from the shared classes directory *MQSI_working_path*`/shared-classes`:
   - `wsrr.properties`
   - `sdo-int.jar`
   - `ServiceRegistryClient.jar`
   - `WMBc4WSRR_1.0.0.jar`
7. Remove the WebSphere Message Broker Toolkit plugin directory and files from the toolkit. The directory was created when you unzipped SupportPac IA9Q; for example, `C:\IBM\MessageBrokersToolkit\6.0\evtoolkit\eclipse\plugins\com.ibm.sr.mb.nodes_1.0.1`

What to do next

Next: When you have removed SupportPac IA9Q, migrate the WebSphere Message Broker Toolkit.

Related tasks:

"Migrating from Version 6.0 products" on page 183
Migrate your components and resources to WebSphere Message Broker Version 7.0.

Related reference:

"Installation Guide" on page 233
Installation information for WebSphere Message Broker is provided in the *Installation Guide* that is supplied as a PDF file with your product package.

"EndpointLookup node" on page 4407
Use the EndpointLookup node to retrieve service endpoint information held in the WebSphere Service Registry and Repository (WSRR). The entities that match the specified search criteria are stored in the local environment. Depending on the node configuration, the Web service URL destination used by the SOAP and HTTP request nodes can also be set. The input message is not modified.

"RegistryLookup node" on page 4646
Use the RegistryLookup node to retrieve any type of entity held in the WebSphere Service Registry and Repository (WSRR). The entities that match the specified search criteria are stored in the local environment. The input message is not modified.

Migrating the WebSphere Message Broker Toolkit development resources from Version 6.0 to Version 7.0

You cannot migrate the WebSphere Message Broker Toolkit component from WebSphere Message Broker Version 6.0, but you can install Version 7.0 to coexist with Version 6.0, and migrate the development resources that you have created in your workspace.

About this task

- You cannot deploy resources from the WebSphere Message Broker Toolkit Version 7.0 to brokers at a previous version that you have not migrated to Version 7.0.
- If any of the following actions have caused an error in the project:
  - Creating the metadata information for the user-defined node project
  - Correcting the plug-in identifier if it does not match the project name
  - Ensuring all the user-defined nodes are in the same category

  The WebSphere Message Broker Toolkit flags the error, and you can use a Quick Fix to rectify the error, see "Applying a Quick Fix to a task list error" on page 2862.However, if you have different user-defined nodes that depend on different resources that have identical names, the broker archive (BAR) file compiler produces an error indicating a naming conflict in the dependent resources. Quick Fix cannot be used to resolve BAR file compiler errors.
- You cannot migrate a deployable JAR file for a user-defined node that was created in Version 7.0, or before Version 7.0.

To migrate your WebSphere Message Broker Toolkit workspace to Version 7.0, complete the following steps:

Procedure

1. Install WebSphere Message Broker Toolkit Version 7.0 in a different location from WebSphere Message Broker Toolkit Version 6.1 or Version 6.0. For detailed instructions, see "Installing the WebSphere Message Broker Toolkit" on page 276, or refer to the "Installation Guide" on page 233.
2. When you start the WebSphere Message Broker Toolkit Version 7.0 for the first time, you are prompted to enter a workspace location. Enter the directory that contains the WebSphere Message Broker Toolkit Version 6.1 or Version 6.0 workspace that you want to migrate and click **OK**. The workspace and resources are now available in the WebSphere Message Broker Toolkit Version 7.0.
3. In the Broker Application Development perspective, select **Window** > **Reset Perspective** to ensure that all views and menus are updated for Version 7.0.
4. You must migrate any user-defined node source projects that were developed in Version 7.0, or before Version 7.0. An error is displayed on all projects that require migration. You can use a Quick Fix to clear these errors, see "Applying a Quick Fix to a task list error" on page 2862. After fixing the errors, you must package the user-defined node source projects and redistribute to the users, see "Packaging and distributing a user-defined node project" on page 3121.

Results

You can now view and modify existing resources, and create resources. You can deploy your workspace resources to Version 7.0 brokers.

What to do next

When you have migrated the resources associated with the WebSphere Message Broker Toolkit, you must migrate your brokers, see "Migrating a broker from WebSphere Message Broker to WebSphere Message Broker Version 7.0" on page 172.

When you have completed these tasks, for information about tasks that you might want to perform after migration, see "Post-migration tasks" on page 204.

Related concepts:

"WebSphere Message Broker Toolkit" on page 31
The WebSphere Message Broker Toolkit is an integrated development environment and graphical user interface based on the Eclipse platform.

Related tasks:

"Migrating from Version 6.0 products" on page 183
Migrate your components and resources to WebSphere Message Broker Version 7.0.

"Migrating a broker from WebSphere Message Broker Version 6.0 to WebSphere Message Broker Version 7.0"
Migrate one or more brokers to Version 7.0.

"Post-migration tasks" on page 204
After you have migrated to Version 7.0, finish setting up your environment.

Migrating a broker from WebSphere Message Broker Version 6.0 to WebSphere Message Broker Version 7.0

Migrate one or more brokers to Version 7.0.

Before you begin

Before you start: Read "Preparing for migration from Version 6.0" on page 185.

About this task

To migrate a broker from WebSphere Message Broker Version 6.0 to WebSphere Message Broker Version 7.0, see the appropriate topic for your operating system:

- "Migrating a Version 6.0 broker to Version 7.0 on distributed operating systems" on page 194
- "Migrating a Version 6.0 broker to Version 7.0 on z/OS" on page 198

Results

You do not have to redeploy resources to a broker that you have migrated. When you start the broker, it starts all existing message flows.

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

"Coexistence with previous versions and other products" on page 139
WebSphere Message Broker Version 7.0 supports restricted coexistence with previous versions and with other products.

Related tasks:

"Migrating from Version 6.0 products" on page 183
Migrate your components and resources to WebSphere Message Broker Version 7.0.

"Post-migration tasks" on page 204
After you have migrated to Version 7.0, finish setting up your environment.

Related reference:

"Supported migration paths" on page 3579
You can migrate to WebSphere Message Broker Version 7.0 from previous versions of the product.

Migrating a Version 6.0 broker to Version 7.0 on distributed operating systems

Migrate a broker to use the enhanced facilities available in Version 7.0.

Before you begin

Before you start:

- Back up your broker resources.
- For the latest details of all supported levels of hardware and software, visit the WebSphere Message Broker Requirements website.
- Install a supported version of WebSphere MQ and migrate your queue managers and other resources, following the instructions provided by WebSphere MQ.
- Optional: If your applications use publish/subscribe functions, and you have message flows that include one or more Publication nodes, you must migrate your subscriptions to WebSphere MQ before you migrate the broker.
- Migrate the WebSphere Message Broker Toolkit from Version 6.0 to Version 7.0.
- Check that you have no aggregations in progress on this broker. When you migrate a broker to Version 7.0, all live data that is being stored for aggregations in progress is lost.
- If the broker runs in a locale that is not listed in "Locales" on page 3629, check that the code page is one of the supported code pages, and that the locale is set up correctly.

About this task

If you stop the broker, you can migrate it immediately to the new version on the same computer. If you prefer not to stop the broker to avoid problems for your business applications, or if you want to reproduce the broker function on another computer, you can associate the application logic on your Version 6.0 broker with a separate Version 7.0 broker.

Select the topic that is appropriate to your environment:

- "Migrating a Version 6.0 broker to Version 7.0"
- "Migrating application logic to a Version 7.0 broker" on page 196

Migrating a Version 6.0 broker to Version 7.0:
About this task

To migrate a Version 6.0 broker on a distributed operating system to Version 7.0, complete the following steps:

Procedure

1. Install WebSphere Message Broker Version 7.0 on the same computer as Version 6.0. Install at least the broker component; other components are optional. You must specify a new location for this installation.
2. Open a Version 6.0 command environment, and stop the Version 6.0 broker by using the **mqsistop** command.
3. Stop all channels that are connected to the Version 6.0 broker.
4. Optional: If your message flows access user databases by using ODBC connections, update the ODBC definitions to Version 7.0 format.
5. Set up the correct Version 7.0 command environment:
   - Linux UNIX On Linux and UNIX systems, open a new shell and run the environment profile **mqsiprofile** for this Version 7.0 installation.
   - Windows On Windows, click **Start**, and open the Command Console that is associated with this Version 7.0 installation.

     On Windows 7 and Windows Server 2008 systems, you must open a command console with elevated privileges. To open a command console with elevated privileges, use the **mqsicommandconsole** command. For more information, see "**mqsicommandconsole** command" on page 3830.
6. Enter the **mqsimigratecomponents** command to migrate the broker. For example:

   ```
   mqsimigratecomponents Broker1
   ```

   All configuration data is retrieved from the Version 6.0 broker database. If you had set a default user ID and password for your Version 6.0 broker by using the **-u** and **-p** options on the **mqsicreatebroker** command, these values are migrated with the broker. You can change these values for your Version 7.0 broker by using the **mqsisetdbparms** command.
7. If the Version 7.0 support for your platform has changed from both 32 bit and 64-bit support to 64-bit support only, make the following changes to your environment:
8. On the AIX, Linux on x86-64, and Solaris on SPARC platforms, support has changed from both 32-bit and 64-bit support to 64-bit support only. 32-bit execution groups are converted to 64-bit. These platforms now require 64-bit libraries and 64-bit ODBC configurations for databases.
   a. Recompile C/C++ user-defined nodes, parsers, and exits to 64-bit, if the existing ones are 32-bit only.
   b. Configure the broker to point at these recompiled extensions by using the following command:

      ```
      mqsichangebroker -l userLilPath -x userExitPath
      ```

      where *userLilPath* defines one or more paths to your LIL files, and *userExitPath* defines one or more paths to your user exit programs.

   The following changes are made to environment variables, for all platforms:

Previous version	Version 7
MQSI_LILPATH64 and MQSI_LILPATH32	MQSI_LILPATH
MQSI_USER_EXIT_PATH64 and MQSI_USER_EXIT_PATH	MQSI_USER_EXIT_PATH
MQSI_SECURITY_PROVIDER_PATH64 and MQSI_SECURITY_PROVIDER_PATH32	MQSI_SECURITY_PROVIDER_PATH

Update all custom profiles that set these environment variables.

9. Copy all additional custom environment settings from your previous environment into your Version 7.0 environment; for example, MQSI_FILENODES_ROOT_DIRECTORY.
10. Start the Version 7.0 broker by using the **mqsistart** command.

Migrating application logic to a Version 7.0 broker:
About this task

To migrate application logic from a Version 6.0 broker on a distributed operating system to a different Version 7.0 broker on the same computer, or on a different computer, complete the following steps:

Procedure

1. Install WebSphere Message Broker Version 7.0 on the target computer. Install at least the broker component; other components are optional. If you are installing on the same computer, you must specify a different location.
2. Optional: If your message flows access user databases by using ODBC connections, update the ODBC definitions to Version 7.0 format.
3. Set up the correct Version 7.0 command environment:
   - Linux UNIX On Linux and UNIX systems, open a new shell and run the environment profile **mqsiprofile** for this Version 7.0 installation.
   - Windows On Windows, click **Start**, and open the Command Console that is associated with this Version 7.0 installation.

     On Windows 7 and Windows Server 2008 systems, you must open a command console with elevated privileges. To open a command console with elevated privileges, use the **mqsicommandconsole** command. For more information, see "**mqsicommandconsole** command" on page 3830.
4. Create a Version 7.0 broker by using the **mqsicreatebroker** command; give it a name and a queue manager name that are different from the names of the Version 6.0 broker and queue manager. You can also create a local broker by using the WebSphere Message Broker Explorer or the WebSphere Message Broker Toolkit, providing you have installed one or both of these components on this computer.
5. Start the Version 7.0 broker by using the **mqsistart** command. You can also start a local broker by using the WebSphere Message Broker Explorer or the WebSphere Message Broker Toolkit, providing you have installed one or both of these components on this computer.
6. Write a list of the execution groups that you have on the Version 6.0 broker, and create these same execution groups on the Version 7.0 broker. Use the WebSphere Message Broker Explorer, the Version 7.0 WebSphere Message Broker Toolkit, or the **mqsicreateexecutiongroup** command to complete this step. On the AIX, Linux on x86-64, and Solaris on SPARC platforms, you can create only 64-bit execution groups.
7. Deploy the message flows and message sets that are in use by the Version 6.0 broker to the Version 7.0 broker from the Version 7.0 WebSphere Message Broker Toolkit. You cannot complete this step unless you have already migrated the WebSphere Message Broker Toolkit.
8. Configure all other relevant properties of the Version 6.0 broker on the Version 7.0 broker.
9. When you are ready to delete your Version 6.0 broker:

a. In a Version 6.0 command environment, stop the Version 6.0 broker by using the **mqsistop** command.
b. Remove the Version 6.0 broker from the Version 6.0 WebSphere Message Broker Toolkit.
c. Redeploy the Version 6.0 topology from the Version 6.0 WebSphere Message Broker Toolkit to update the configuration data held by the Configuration Manager.
d. In a Version 6.0 command environment, delete the Version 6.0 broker by using the **mqsideletebroker** command.

What to do next

When you have completed these tasks, see the post-migration tasks for information about tasks that you might want to perform after migration.

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

"Coexistence with previous versions and other products" on page 139
WebSphere Message Broker Version 7.0 supports restricted coexistence with previous versions and with other products.

Related tasks:

"Migrating from Version 6.0 products" on page 183
Migrate your components and resources to WebSphere Message Broker Version 7.0.

"Post-migration tasks" on page 204
After you have migrated to Version 7.0, finish setting up your environment.

"Creating a broker" on page 611
You can create brokers on every platform that is supported by WebSphere Message Broker. The broker runs as a 64-bit application on all platforms except Linux on x86 and Windows on x86.

"Starting and stopping a broker" on page 921
Run the appropriate command to start or stop a broker.

"Creating an execution group using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 937
Use the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer to create execution groups on your broker.

"Creating an execution group using the **mqsicreateexecutiongroup** command" on page 939
Use the **mqsicreateexecutiongroup** command to create execution groups on your broker.

Related reference:

"Supported migration paths" on page 3579
You can migrate to WebSphere Message Broker Version 7.0 from previous versions of the product.

"**mqsicommandconsole** command" on page 3830
Use the **mqsicommandconsole** command to launch an elevated command console from which commands that require elevation on Windows can be run.

"**mqsideletebroker** command" on page 3863
Use the **mqsideletebroker** command to delete a named broker. The command also deletes the queues on the associated queue manager (created when the broker was created). You can also specify that the queue manager is to be deleted.

"**mqsimigratecomponents** command" on page 3894
Use the **mqsimigratecomponents** command to migrate a component from a previously installed version of the product to another version on the same computer.

"**mqsistart** command" on page 3965
Use the **mqsistart** command to start the specified broker if all initial verification tests complete successfully.

"**mqsistop** command" on page 3972
Use the **mqsistop** command to stop the specified component.

Migrating a Version 6.0 broker to Version 7.0 on z/OS

Migrate a broker to use the enhanced facilities available in Version 7.0.

Before you begin

Before you start:

- Back up your broker resources.
- For the latest details of all supported levels of hardware and software, visit the WebSphere Message Broker Requirements website.
- Install WebSphere MQ and migrate your queue managers and other resources, following the instructions provided by WebSphere MQ.
- Optional: If your applications use publish/subscribe functions, and you have message flows that include one or more Publication nodes, you must migrate your subscriptions to WebSphere MQ before you migrate the broker.
- Ensure that you are familiar with the steps involved in creating a broker on z/OS.
- Ensure that you are familiar with the **mqsimigratecomponents** command which the JCL (Job Control Language) file uses to migrate a broker.
- Check that you have no aggregations in progress. When you migrate a broker to Version 7.0, any live data that is being stored for aggregations in progress is lost.
- If the broker runs in a locale that is not listed in "Locales" on page 3629, check that the code page is one of the supported code pages and that the locale is set up correctly.

About this task

To migrate a WebSphere Message Broker Version 6.0 broker to Version 7.0 on z/OS:

Procedure

1. Stop the Version 6.0 broker.
2. Create a new broker PDSE.
3. Copy all broker JCL files from the Version 7.0 installed SBIPPROC and SBIPSAMP PDSEs to the new broker PDSE and customize them all. See "Customizing the broker JCL" on page 625 for more information.

   You must take a backup copy of your Version 6.0 `ENVFILE` file before you submit the `BIPGEN` job from the Version 7.0 component data set. The `ENVFILE` file is stored in the directory referenced by the ++*HOME*++ JCL variable. Subsequently, if you want to migrate back to Version 6.0, restore the `ENVFILE` file, either from the backup file location, or by submitting the `BIPGEN` job from the Version 6.0 component data set, before you start the Version 6.0 broker.

a. Customize the BIPEDIT file by using the values that are defined in the Version 6.0 BIPEDIT file.
b. Copy all additional changes that you have made to the Version 6.1 version of the BIPBPROF and BIPDSNAO files to these files in the Version 7.0 component data set. Submit the BIPGEN job to create the environment file ENVFILE.
c. Customize and submit the BIPMGCMP (**mqsimigratecomponents**) job. This job migrates the registry, queues, and broker database. Connection to the database of the broker that you are migrating is required to run this command. After migration, all information is stored in an internal repository in Version 7.0; the Version 7.0 broker does not require a database.

4. Copy the renamed started task JCL BIPBRKP to the procedures library. When you copy the started task, keep a second copy of the original in a safe place for backup purposes.
5. The verification program runs when you start the Version 7.0 broker.

What to do next

Next: See the post-migration tasks for information about tasks that you might want to perform after migration.

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

"Coexistence with previous versions and other products" on page 139
WebSphere Message Broker Version 7.0 supports restricted coexistence with previous versions and with other products.

Related tasks:

"Migrating from Version 6.0 products" on page 183
Migrate your components and resources to WebSphere Message Broker Version 7.0.

"Post-migration tasks" on page 204
After you have migrated to Version 7.0, finish setting up your environment.

"Backing up WebSphere Message Broker Version 6.0 resources" on page 187
Back up your resources before you start to migrate components to Version 7.0.

"Creating a broker on z/OS" on page 620
Create the broker component and the other resources on which it depends.

"Creating the broker PDSE" on page 622
This is part of the larger task of creating a broker on z/OS.

"Customizing the broker JCL" on page 625
This subtask is part of the larger task of creating a broker on z/OS.

"Copying the broker started task to the procedures library" on page 629
This is part of the larger task of creating a broker on z/OS.

"Starting and stopping a broker on z/OS" on page 924
Run the appropriate command from SDSF to start or stop a broker.

Related reference:

"Supported migration paths" on page 3579
You can migrate to WebSphere Message Broker Version 7.0 from previous versions of the product.

"Sample BIPEDIT file" on page 4009
The sample BIPEDIT file that is shipped with WebSphere Message Broker is

included here for your reference.

Migrating Configuration Manager ACLs

If you are migrating from WebSphere Message Broker , WebSphere Message Broker Version 6.0, or WebSphere Event Broker Version 6.0, you can use the Access Control Lists (ACLs) that you set up in the Configuration Manager as the basis for your security model in Version 7.0.

Before you begin

Before you start:

- Migrate your broker by following the instructions in "Migrating a broker from WebSphere Message Broker to WebSphere Message Broker Version 7.0" on page 172 or "Migrating a broker from WebSphere Message Broker Version 6.0 to WebSphere Message Broker Version 7.0" on page 193.
- Activate broker administration security by following the instructions in "Enabling broker administration security" on page 368.

Administration security for a broker in Version 7.0 is disabled by default. Because no security details are migrated, when you migrate a broker from a previous version security is disabled, even if you had set up security in the previous version.

About this task

The Configuration Manager has been removed as a result of the architectural and security model changes in Version 7.0, therefore the commands that controlled this component and its ACLs have been removed:

mqsichangeconfigmgr
mqsicreateconfigmgr
mqsideleteconfigmgr
mqsireportconfigmgr
mqsicreateaclentry
mqsideleteaclentry
mqsilistaclentry

Although you can use the ACLs that you set up in previous versions as the basis for your security model in Version 7.0, you might choose to reconsider your security model, and set up different levels of control that match the facilities available in this version.

If you choose to use your existing ACLs as a basis for your security model in Version 7.0, consider the following factors:

- You can obtain a list of existing ACLs maintained by your Configuration Manager by using the **mqsilistaclentry** command.
- You can migrate existing groups, users, or both, subject to the following factors:
  - You must define the user ID on the same computer as the queue manager associated with the Version 7.0 broker.
  - On Linux and UNIX systems, you can grant authority only to the primary group of a user. All users that you have defined with the same primary group automatically get the same level of security access. You must therefore consider the membership of all your groups to ensure that you give the required level of control to each user.

- You can grant only broker and execution level authorities for Version 7.0 brokers.
- If you grant read, write, and execute authority to a user ID for a Version 7.0 broker, this permission is equivalent to full control access in previous versions.
- If you grant read authority to a user ID for a Version 7.0 broker, this permission is equivalent to view access in previous versions.
- Check the authority that write and execute permissions grant for a Version 7.0 broker to determine the best match for edit and deploy access levels in previous versions.
- Although you can set up access for a particular computer or domain name in previous versions, you can use only user IDs and groups in Version 7.0. If you want to establish a more secure environment, consider the use of WebSphere MQ security exits and SSL.

To set up authorization for your Version 7.0 broker, complete the following steps.

Procedure

1. Review the authorizations that are required for specific tasks and commands. Details are provided in "Tasks and authorizations for broker administration security" on page 3645 and "Commands and authorizations for broker administration security" on page 3646.
2. Grant the authorities that your users require.
   - For details of this task on distributed systems, with examples, see "Granting and revoking authority on Linux, UNIX, and Windows systems" on page 374.
   - For details of this task on z/OS, with examples, see "Granting and revoking authority on z/OS systems" on page 376.

What to do next

Next: Start the broker in the WebSphere Message Broker Explorer, or run the **mqsistart** command.

Related concepts:

"Security overview" on page 351
When you are designing a WebSphere Message Broker application it is important to consider the security measures that are needed to protect the information in the system.

Related tasks:

"Enabling broker administration security" on page 368
Enable broker administration security on a broker to control which users can complete specific tasks against that broker and its resources.

"Authorizing users for broker administration" on page 371
Grant authority to one or more groups or users to authorize them to complete specific tasks against a broker and its resources.

"Enabling SSL on the WebSphere MQ Java Client" on page 540
The WebSphere MQ Java Client supports SSL-encrypted connections over the server-connection (SVRCONN) channel between an application and the queue manager. Configure SSL support for connections between applications that use the CMP API (including the WebSphere Message Broker Toolkit and the WebSphere Message Broker Explorer) and a broker.

Related reference:

"Commands" on page 3672
All WebSphere Message Broker Toolkit and runtime commands that are provided on distributed systems are listed, grouped by function, with references to command details.

Related information:

WebSphere MQ Version 7 Information Center online

Migrating a WebSphere Message Broker that is configured by using Microsoft Cluster Services (MSCS)

You must complete a specific set of steps to migrate your Microsoft Cluster Services (MSCS) cluster to WebSphere Message Broker Version 7.0.

Before you begin

Read the topics within the section "Migrating from Version 6.0 products" on page 183, and the topic "Using a broker with an existing high availability manager" on page 843.

About this task

Use the following procedure to migrate a WebSphere Message Broker that is configured by using MSCS.

Migrating a WebSphere Message Broker Version 6.0 to WebSphere Message Broker Version 7.0

Procedure

1. Install WebSphere Message Broker Version 7.0 onto each node. You must specify a new location for this installation.
2. Stop all channels that are connected to the WebSphere Message Broker Version 6.0 broker.
3. In MSCS bring the broker resource offline, keeping the cluster disk and WebSphere MQ resource online on your primary node
4. In a WebSphere Message Broker Version 7.0 command console, run the following command:

   ```
   mqsimigratecomponents BROKER1 -m
   ```

   where BROKER1 is the name of your broker.
5. Move the broker cluster group to the secondary node. The broker resource must remain stopped.
6. In a WebSphere Message Broker Version 7.0 command console, run the following command:

   ```
   mqsimigratecomponents BROKER1 -m -1
   ```
7. Copy all additional custom environment settings from your previous environment into your WebSphere Message Broker Version 7.0 environment. For example: MQSI_FILENODES_ROOT_DIRECTORY.
8. Bring the cluster disk and WebSphere MQ resource offline.
9. Start the WebSphere Message Broker cluster group on its primary node.

What to do next

Migrating a WebSphere Message Broker Version 7.0 to WebSphere Message Broker Version 6.0

Due to changes in the file security model that are used in WebSphere Message Broker Version 7.0, you might be required to alter the file permissions of the shared resource.

If necessary, create a domain group, add the `userID` of your broker to this group, and give the group full access to the shared files.

1. In MSCS bring the broker resource offline, keeping the cluster disk and WebSphere MQ resource online on your primary node.
2. In a WebSphere Message Broker Version 6.0 command console, run the following command:

   ```
   mqsimigratecomponents BROKER1 -s 7.0 -t 6.0
   ```
3. In a WebSphere Message Broker Version 6.0 command console, run the following command:

   ```
   mqsistart BROKER1
   ```

   This command is required to update the Windows service to point at the WebSphere Message Broker Version 6.0 binary files.
4. In a WebSphere Message Broker Version 6.0 command console, run the following command:

   ```
   mqsistop BROKER1
   ```
5. Move the broker cluster group to the secondary node. The broker resource must remain stopped.
6. In a WebSphere Message Broker Version 7.0 command console, run the following command:

   ```
   mqsimigratecomponents MSCSV61BROKER -m -s 6.0 -t 7 -1
   ```
7. In a WebSphere Message Broker Version 7.0 command console, run the following command:

   ```
   mqsimigratecomponents MSCSV61BROKER -s 7.0 -t 6.0 -1
   ```
8. In a WebSphere Message Broker Version 6.0 command console, run the following command:

   ```
   mqsistart BROKER1
   ```

   This command is required to update the Windows service to point at the WebSphere Message Broker Version 6.0 binary files.
9. In a WebSphere Message Broker Version 6.0 command console, run the following command:

   ```
   mqsistop BROKER1
   ```
10. Bring the cluster disk and WebSphere MQ resource offline.
11. Start the broker cluster group on its primary node.

Related tasks:

"Migrating from Version 6.0 products" on page 183
Migrate your components and resources to WebSphere Message Broker Version 7.0.

"Using a broker with an existing high availability manager" on page 843
You can use WebSphere Message Broker Version 7.0 with an existing high availability manager, for example HACMP, HA/XD, VCS, or HP-UX Serviceguard.

Post-migration tasks

After you have migrated to Version 7.0, finish setting up your environment.

About this task

Test the WebSphere Message Broker Version 7.0 broker resources and components to verify that you experience no loss or unexpected change of functionality. Some changes in behavior might be caused by defects that have been fixed between versions.

The following topics describe further tasks that you can complete after migration:

Procedure

- "Reviewing technical changes in Version 7.0" on page 205
- "Setting up a command environment" on page 213
- "Migrating a flow containing HTTPRequest nodes" on page 214
- "Migrating a flow containing XMLTransformation nodes" on page 215
- "Migrating a flow containing data definitions" on page 217
- "Migrating CMP applications" on page 220
- "Updating error processing routines" on page 221

What to do next

Next: Complete the following tasks as required:

- Delete the Version 6.1 or Version 6.0 components (Configuration Manager, User Name Server, and brokers that you have replaced but not migrated).

  If the broker that you have migrated shares a queue manager with the Configuration Manager, the default server connection channel SYSTEM.BKR.CONFIG, which was created when you created the Configuration Manager, is deleted when you delete the Configuration Manager. The Version 7.0 broker requires this channel for connections from the WebSphere Message Broker Explorer, the WebSphere Message Broker Toolkit, and CMP applications. You must re-create this channel on the broker queue manager after you have deleted the Configuration Manager.
- Remove the installed code for Version 6.1 or Version 6.0.
- Delete the broker database. You can also remove the installed database product, if you do not use it for other purposes.
- If you are migrating from Version 6.0, delete the configuration repository. You can also remove the installed database product, if you do not use it for other purposes.

Access the Version 6.1 or Version 6.0 information center for details of these tasks (or use the links provided).

Related concepts:

"Coexistence with previous versions and other products" on page 139
WebSphere Message Broker Version 7.0 supports restricted coexistence with previous versions and with other products.

Related tasks:

"Migrating publish/subscribe information to WebSphere MQ" on page 141
These tasks give an overview of how to migrate publish/subscribe information from WebSphere Message Broker or WebSphere Message Broker Version 6.0 to

WebSphere MQ.

"Migrating from Version 6.1 products" on page 163
Migrate your components and resources to WebSphere Message Broker Version 7.0.

"Migrating from Version 6.0 products" on page 183
Migrate your components and resources to WebSphere Message Broker Version 7.0.

Related reference:

"Supported migration paths" on page 3579
You can migrate to WebSphere Message Broker Version 7.0 from previous versions of the product.

Related information:

WebSphere Message Broker Information Center online

WebSphere Message Brokers Version 6.0 Information Center online

Reviewing technical changes in Version 7.0

Some minor changes in behavior are present in WebSphere Message Broker Version 7.0; for example, those changes caused by defects that have been fixed between versions.

If you are migrating from Version 6.0 or Version 6.1, read the following sections to understand the potential effects on your broker and message flows:

- "Changes to user ID and password requirements"
- "Changes to location of description properties" on page 206
- "Default Configuration wizard and database usage on Windows" on page 206
- "Default execution group when creating brokers" on page 206
- "Starting and stopping execution groups" on page 206
- "Using SOAPAsyncRequest, SOAPInput, and SOAPRequest nodes" on page 207
- "Using HTTPS with HTTPInput and HTTPReply nodes" on page 207
- "Monitoring message flows" on page 207
- "ESQL field references with an index of zero" on page 208
- "Using RegistryLookup nodes" on page 208
- "Interfaces in the WebSphere Message Broker Toolkit" on page 208

If you are migrating from Version 6.0, and you have created message sets, review the changes in behavior in "Message set migration" on page 209.

Changes to user ID and password requirements

User ID and password management in Version 7.0 has been updated:

- The requirement for a service user ID and password has been removed on all systems except Windows. These parameters (**-i, -a**) are no longer used when you migrate your brokers to Version 7.0.

  If you restore a Windows broker to an earlier version, the password value is restored with the broker. If you have changed the password by using the **mqsichangebroker** command, the updated value is set in the previous version.
- Because the Version 7.0 broker has no requirement for a database, the parameters to define and change database user IDs and passwords have been removed.

If you had set these values for the brokers that you are migrating, these parameters (**-u, -p**) are migrated with the broker, and are used as default values for data sources (user databases) for which you do not set explicit values. If you had not set **-u, -p**, that values for **-i, -a** are migrated. In Version 7.0, you can manage these user IDs and passwords for your user databases by using the **mqsisetdbparms** command.

- The requirement for the broker service user ID to be a member of the mqm group has been removed.
- If you change the database user ID and password by using the **mqsisetdbparms** command, you no longer need to restart the broker. You can instead restart the affected execution group by using the **mqsireload** command.

Changes to location of description properties

Long and short description properties of deployed Message Broker artifacts were not held in the deployed execution group repository, so they will not have been migrated to the Version 7.0 broker.

If the following fields have been used to hold keywords:

```
$MQSI name = value MQSI$
```

they will not be displayed in the migrated artefacts. To correct this behavior, redeploy the artefacts directly to the Version 7.0 broker.

For more information about defining keywords, see "Guidance for defining keywords" on page 4017.

Default Configuration wizard and database usage on Windows

Some of the sample programs use a database; for example, the Airline sample. If you had used the Default Configuration wizard to set up a default configuration on Windows, and deploy samples to the default broker, the samples that require a database use the Derby database that is embedded in the broker. Version 7.0 does not ship or support the Derby database. You must reconfigure your database samples by following the updated instructions in the samples documentation.

Default execution group when creating brokers

When you create a broker by using the **mqsicreatebroker** command, a default execution group is no longer created.

If you use either the WebSphere Message Broker Toolkit or the WebSphere Message Broker Explorer to create a broker, you can select an option to create a default execution group with the name `default` (unless you specify another name).

You can also create execution groups by using the **mqsicreateexecutiongroup** command.

Starting and stopping execution groups

Starting and stopping execution group behavior has been updated in Version 7.0. When you start or stop an execution group using the **mqsistartmsgflow** or **mqsistopmsgflow** commands without the **-m** parameter, the execution group process is stopped or started. When you stop the execution group in this way, or using the WebSphere Message Broker Toolkit, or the WebSphere Message Broker Explorer,

the run state of the message flows deployed to the execution group is remembered. When you next start the execution group only those message flows that were running when the execution group was stopped are restarted, unless you specifically request all flows to be started, or use the **-j** parameter on the command.

Using SOAPAsyncRequest, SOAPInput, and SOAPRequest nodes

The Failure action property of the SOAPAsyncRequest, SOAPInput, and SOAPRequest nodes has changed to be not configurable. If you have set this property, for example in a BAR file, the setting is ignored.

Using HTTPS with HTTPInput and HTTPReply nodes

The Version 7.0 broker checks for required SSL configuration when you run the **mqsistart** command.

If you have deployed a message flow that includes HTTPInput or HTTPReply nodes to a Version 6.1 or Version 6.0 broker, and you migrate the broker to Version 7.0 and start the broker again, you might see the following error message generated. (Message lines are continuous, but have been split to improve readability).

```
BIP3135S: An exception occurred while starting the servlet engine connector.
Exception text is HTTP Listener LifecycleException:
 Protocol handler start failed: java.io.FileNotFoundException: /home/leed/.keystore
 (No such file or directory)
 at org.apache.coyote.tomcat5.CoyoteConnector.start(CoyoteConnector.java:1529)
 at com.ibm.broker.httplistener.ConnectorWrapper.start(ConnectorWrapper.java:166)
 at com.ibm.broker.httplistener.TomcatWrapper.startSecureHTTPSConnector
 (TomcatWrapper.java:146)
 at com.ibm.broker.httplistener.HTTPListenerManager.ensureServletContainer
 (HTTPListenerManager.java:290)
 at com.ibm.broker.httplistener.HTTPListenerManager.run(HTTPListenerManager.java:153)
 at java.lang.Thread.run(Thread.java:735)  :
 DANBRK.httplistener: /build/S000_P/src/DataFlowEngine/NativeTrace/ImbNativeTrace.cpp: 732:
 ensureServletContainer: :
Oct 13 13:47:16 partick user:err|error WebSphere Broker v7000[303572]:
 (DANBRK.default)[1]BIP2275E: Error loading message flow 'ef2a0606-2401-0000-0080-984a4915984c'. :
 DANBRK.de427601-2401-0000-0080-d525e90f1528: /build/S000_P/src/DataFlowEngine/ImbDataFlowDirector.cpp:
 2957: ImbDataFlowDirector::loadAllDataFlowsFromDatabase:
 ExecutionGroup: de427601-2401-0000-0080-d525e90f1528
```

This error is generated because the Version 7.0 broker has detected that you have configured the HTTP nodes in the message flow to use HTTPS, but you have not set up the required SSL configuration; the broker does not load the message flow. In previous versions, this check was not performed and no error generated.

To resolve this error, configure your HTTP nodes to use SSL, and redeploy the message flow. For SSL configuration information, see “Configuring HTTPInput and HTTPReply nodes to use SSL (HTTPS)” on page 535.

Monitoring message flows

The default behavior for publishing monitoring events has changed. In versions before Version 7.0, monitoring events are emitted out of sync point. Now, the default for all events except transaction rollback is that events are emitted only if the message flow commits its unit of work successfully. By default, transaction rollback events are emitted in a second unit of work, independent of the main unit of work.

These changes mean that you no longer see events that have been backed out because of a failed message flow; you see only the transaction start event and the transaction rollback event, if these events are defined. You also see all other events that are defined to be in an independent unit of work. See “Monitoring basics” on page 3320 for more information.

A sequence number has been added to the eventSequence element of the monitoring event. Because both the creation time and sequence number are always emitted in the monitoring event, the Sequence tab has been removed from the monitoring tab in the WebSphere Message Broker Toolkit.

ESQL field references with an index of zero

The validity of using a field reference index of zero has been corrected. If you have statements in your ESQL modules that include an index of zero, error BIP3226E is generated when you deploy the message flow.

For example, if you have code that contains the statement:

```
SET OutputRoot.XMLNSC.Top.A[0].B = 42;
```

You must update the code to contain the following content:

```
SET OutputRoot.XMLNSC.Top.A[1].B = 42;
```

Using RegistryLookup nodes

The default for the Depth Policy property of the RegistryLookup node has been changed from the value Return matched showing immediate relationships (For compatibility only) in Version 6.1 to the value Return matched only (Depth = 0) in Version 7.0.

If you do not explicitly set this property on a RegistryLookup node, it uses the default value Return matched only (Depth = 0) to determine the depth of the WSRR query and the contents of the entity data to be returned.

If you want to use the node in deprecated mode in Version 7.0, you must explicitly set the Depth Policy property to the value Return matched showing immediate relationships (For compatibility only), and rebuild the BAR file.

For more information about the RegistryLookup node and its properties, see “RegistryLookup node” on page 4646.

Interfaces in the WebSphere Message Broker Toolkit

The following changes are present in the WebSphere Message Broker Toolkit:

Problems view

In WebSphere Message Broker Toolkit Version 6.1, you can configure the list of problems shown in the Problems view by clicking either the icon on the Problems view pane bar, or the down arrow next to the icon, and selecting **Configure filter** from the list of options displayed. In WebSphere Message Broker Toolkit Version 7.0, the icon is no longer shown. Click the down arrow shown at the right end of the bar, and select **Configure contents**.

Broker Development view
In WebSphere Message Broker Toolkit Version 7.0, the Broker Development view shows pattern instance projects in a separate pane, in addition to other projects in your workspace.

Broker administration perspective
In WebSphere Message Broker Toolkit Version 6.1, you can connect, configure, and deploy to brokers by using the Broker administration perspective in the WebSphere Message Broker Toolkit. In Version 7.0, the Broker Administration perspective has been removed, and you can now connect, configure and deploy to brokers by using the Brokers view in the Broker Application Development perspective. For more advanced configuration tasks, you can use the WebSphere Message Broker Explorer.

Event Log viewer
In WebSphere Message Broker Toolkit Version 6.1, deployment responses and messages from the broker are displayed in the Event Log viewer. In Version 7.0, deployment messages from your instance of the WebSphere Message Broker Toolkit are displayed in the Deployment Log view, in the Broker Application Development perspective.

Command assistants
In WebSphere Message Broker Toolkit Version 6.1 and Version 6.0, you can use the command assistants to create, change, and delete components such as brokers on your local system. In Version 7.0, you can use the Brokers view to create and delete components. Alternatively, you can use the WebSphere Message Broker Explorer to create, change, and delete brokers on your local system.

XPath Expression Builder
In WebSphere Message Broker Toolkit Version 6.1, the Data Types Viewer shows two top-level categories, Data Types and Variables. In Version 7.0, you can find the variables under the single top-level category Data Types.

Related tasks:

"Migrating from Version 6.1 products" on page 163
Migrate your components and resources to WebSphere Message Broker Version 7.0.

"Migrating from Version 6.0 products" on page 183
Migrate your components and resources to WebSphere Message Broker Version 7.0.

"Post-migration tasks" on page 204
After you have migrated to Version 7.0, finish setting up your environment.

Message set migration

When you migrate a message flow that uses message sets, review the changes in behavior in WebSphere Message Broker Version 7.0.

Before you begin

Review this information about message set changes, and potential effects, if you are migrating from Version 6.0; no message set migration is required from Version 6.1.

You do not have to run migration commands; WebSphere Message Broker Toolkit Version 7.0 can read a Version 6.0 message set project and automatically convert it to Version 7.0 format when you modify and save it for the first time.

For guidance about changes to the behavior of parsers, and for general migration information, see the following sections:

- "Behavioral changes in the XMLNSC parser"
- "Behavioral changes in the MRM parser" on page 211
- "General migration information" on page 211

Behavioral changes in the XMLNSC parser:
About this task

The following changes are implemented in Version 7.0:

- In Version 6.0, the attributes of an XML element are displayed in the message tree that is created by the XMLNSC parser in the order in which they are displayed on the element in the XML document.

  In Version 7.0, the XMLNSC parser always displays namespace attributes ahead of other attributes when it creates the message tree. However, XML attributes are inherently unordered, therefore do not write message flow logic that assumes attribute position.
- In Version 6.0, the XMLNSC parser accepts the EBCDIC NEL as an XML line feed character on z/OS only.

  In Version 7.0, NEL is accepted on all platforms.
- In Version 6.0, the XMLNSC parser ignores a node's Validate property setting.

  In Version 7.0, the XMLNSC parser honors the Validate property setting. If you have set this property to Content or Content And Value, the parser validates the message against the message set defined by the Message Set property.

  Therefore, if you have accidentally set the Validate property in Version 6.0 to Content or Content And Value, the message flow throws an exception after migration to Version 7.0.

  To correct this situation, set the Validate property to None.
- In Version 6.0. the XMLNSC parser uses the default ESQL format "YYYY-MM-DD hh:mm:ss" when it writes a field of type TIMESTAMP or GMTTIMESTAMP. This format does not comply with the XML Schema 1.0 specification.

  In Version 7.0, the XMLNSC parser uses the format "YYYY-MM-DD'T'hh:mm:ss.sss", which complies with the XML Schema 1.0 specification. If you require another format to be written, you must cast the TIMESTAMP or GMTTIMESTAMP to a CHARACTER. (If the element was originally parsed by the XMLNSC parser and had a time zone indicator, the time zone indicator is also written to the output message field.)
- In Version 6.0, the XMLNSC parser uses the default ESQL strings "TRUE" and "FALSE" when it writes a field of type BOOLEAN. This use does not comply with the XML Schema 1.0 specification.

  In Version 7.0, the XMLNSC parser writes "true" and "false", which complies with the XML Schema 1.0 specification. If you require strings other than "true and "false" to be written, cast the BOOLEAN field to a CHARACTER.
- In Version 6.0, the XMLNSC parser does not validate XML documents against XML Schema generated from a message set.

  In Version 7.0, the XMLNSC parser offers this function. When you add a message set to a broker archive (BAR) file, if the Message Domain property of the message set is set to XMLNSC, an XML Schema is generated for each message definition file in the message set for potential use by the XMLNSC parser. These XML Schema are checked by the XMLNSC parser when the BAR file is deployed.

However, the checks are stricter than the checks that are carried out by the message definition file editor, and the deployment might fail. If the deployment fails, you must correct the message definition file to comply with XML Schema rules. If the message set contains one of the IBM supplied SOAP Envelope message definition files, reimport the message definition file by using the New Message Definition File From IBM supplied Message wizard.

Behavioral changes in the MRM parser:
About this task

The following changes are implemented in Version 7.0:

- In Version 6.0, the MRM parser TDS physical format Data Element Separations Tagged Fixed Length, Tagged Delimited, and Tagged Encoded Length write out any xsi:type attributes they encounter in the message tree, using a tag name of "type".

  In Version 7.0, the TDS physical format assumes that all xsi:type attributes are XML-specific, and does not write them out.
- In Version 6.0, the MRM TDS physical format Data Element Separations Fixed Length, Fixed Length AL3, and Tagged Fixed Length treat an element with a TDS Length of zero as fixed length when parsing, but as variable length when writing. Therefore, all data in the message tree for such an element is written to the bit stream without an exception being thrown.

  In Version 7.0, the element is treated as fixed length when parsing and writing. Therefore, all data in the message tree for such an element causes message BIP5436E to be generated, unless you have set the TDS property Truncate on output, in which case nothing is written.
- In Version 6.0, message BIP5374E is generated when a missing mandatory repeating element (minOccurs > 1) is detected during validation of an MRM message.

  In Version 7.0, two different conditions are detected during validation:
  - A mandatory element that is repeating but has no instances in the message tree. In this case, message BIP5378E is generated, which provides details of the elements that exist before and after the expected location of the missing element. Because of this change, you might have to change or enhance automated error processes that you have in place.
  - A mandatory element that is repeating and has the incorrect number of instances in the message tree. In this case, message BIP5374E is generated as before, giving details of the existing and the expected instances of the element.

General migration information:
About this task

The following changes are implemented in Version 7.0:

- A single message set can contain models for MRM or IDOC domains, and also for other domains such as XMLNSC. In Version 6.0, a `.dictionary` file for use by the MRM parser is always generated when the message set is added to a BAR file.

  In Version 7.0, a `.dictionary` file is generated only if you have either specified MRM or IDOC as the default message domain, or added MRM or IDOC to Supported message domains in the message set file. If a `.dictionary` file is no longer generated, add MRM or IDOC to Supported message domains.

- The XML parser is deprecated and might be withdrawn in a future release. Existing message flows that use the XML parser continue to operate in Version 7.0. Use the XMLNSC domain for most new message flows that need to parse or write XML documents. For more information, see "Which XML parser should you use?" on page 1080.
- The IDOC domain and parser are deprecated, and might be withdrawn in a future release. Existing message flows that use the IDOC parser continue to operate in Version 7.0. Use the MRM domain TDS physical format for new message flows that need to parse or write SAP ALE IDocs in text format that originate from the WebSphere MQ link for R3.
- The location and labeling of some MRM TDS properties have been updated; no behavioral changes are associated with these changes.
  - The Repeating Element Delimiter property has moved into a box labeled "Occurrences", to keep all repeating properties together.
  - The Virtual Decimal Point and Precision properties are moved below the "Sign" group of properties, to align them with CWF properties.
  - The Length Reference property was not used for local groups, group references, and complex elements, and is removed from their property sheets.
  - The local group property sheet now has two boxes labeled "Field Identification", to distinguish the Data Pattern property from the rest, because Data Pattern applies not to the group itself, but to its inclusion in its parent.
  - The message set Boolean representation properties are now displayed with prefix "Text" to distinguish them from new binary Boolean representation properties.
  - The Physical Type property enumeration value Character is renamed Text.
  - The Physical Type property enumeration value Messaging Standard Alternate is renamed TLOG Specific.
  - The Trim Fixed Len String property is renamed Trim On Input, and has moved into a new box labeled "Fixed length strings".
  - The Derive default length from logical type property has moved into a new box labeled "General settings".
- The location and labeling of some MRM CWF properties have been updated; no behavioral changes are associated with these changes.
  - The Format property has moved to be last in the "Physical representation" section, to align it with TDS properties.
  - The element property sheet now has a section labeled "Numeric representation" section which contains the Sign group of properties and Virtual Decimal Point, to align these properties with TDS properties.
  - The Length Count property is renamed Length, to align this property with TDS properties.
  - The String Justification property is renamed Justification, to align this property with TDS properties.
  - The Sign EBCDIC Custom property is renamed Sign EBCDIC Custom Overpunched, to align this property with TDS properties.
  - The Sign Orientation property enumeration values Leading and Trailing are renamed Leading Overpunched and Trailing Overpunched, to align these properties with TDS properties.
  - The message set byte order properties are now displayed with the prefix "Default" to align these properties with TDS properties.

Related concepts:

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

"Identifying an embedded message by using a Message Identity" on page 1193
You can identify an embedded message by using the Message Identity. This technique is used by the MRM domain, and replaces the use of the Message Key.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Migrating from Version 6.0 products" on page 183
Migrate your components and resources to WebSphere Message Broker Version 7.0.

Input message tree modification

You might see a change in behavior when you migrate message flows that modify input message trees.

When you migrate message flows from Version 6.0 of WebSphere Message Broker that have been incorrectly modifying input message trees in ESQL nodes or user-defined nodes, you might see a change in behavior because the changes are now reflected in a bit stream that is written from that tree.

Related concepts:

"Message tree structure" on page 1045
The message tree is a part of the logical message tree in which the broker stores its internal representation of the message body.

Related tasks:

"Migrating from Version 6.1 products" on page 163
Migrate your components and resources to WebSphere Message Broker Version 7.0.

"Migrating from Version 6.0 products" on page 183
Migrate your components and resources to WebSphere Message Broker Version 7.0.

Chapter 3, "Migrating and upgrading," on page 137
To migrate a broker domain to WebSphere Message Broker Version 7.0, plan your migration strategy, perform pre-migration tasks, migrate your domain components, and then complete post-migration tasks.

"Manipulating message body content" on page 2418
The message body is always the last child of root, and is identified by its parser name, for example XML or MRM. You can refer to, modify, and create message body data.

"Manipulating other parts of the message tree" on page 2452
You can access message tree headers, the properties tree, the local environment tree, the environment tree and the exception list tree.

Setting up a command environment

After you have installed the product on one of the distributed systems, you must initialize the environment before you can use a runtime component or command.

About this task

Also complete this task if you have migrated to WebSphere Message Broker Version 7.0 from an earlier version. A profile is provided to help you set up the environment.

If appropriate, you can extend the initialization performed by this profile; for example, for user databases, or for other products that you want to use within the broker.

Ensure that you use this environment each time you run an administrative command, or start a broker.

For information about setting up your command and runtime environment on a Windows system, see "Command environment: Windows systems" on page 306.

For information on setting up your command and runtime environment on Linux and UNIX systems, see "Command environment: Linux and UNIX systems" on page 310.

For information about setting up your command and runtime environment for execution groups on Linux and UNIX systems, see "Execution group-specific command environment: Linux and UNIX systems" on page 312.

For information about setting up your command and runtime environment for execution groups on Windows systems, see "Execution group-specific command environment: Windows systems" on page 309.

Related tasks:

"Migrating from Version 6.1 products" on page 163
Migrate your components and resources to WebSphere Message Broker Version 7.0.

"Migrating from Version 6.0 products" on page 183
Migrate your components and resources to WebSphere Message Broker Version 7.0.

"Enabling ODBC connections to the databases" on page 668
Set up the resources and environment that the broker requires for Open Database Connectivity (ODBC) connections to user databases on distributed systems.

"Command environment: Windows systems" on page 306
Set up the Windows environment to run WebSphere Message Broker commands.

"Command environment: Linux and UNIX systems" on page 310
Set up the Linux or UNIX environment to run WebSphere Message Broker commands.

Related reference:

"Runtime commands" on page 3715
The topics in this section describe the WebSphere Message Broker runtime commands.

Migrating a flow containing HTTPRequest nodes

When you migrate a message flow that contains HTTPRequest nodes from Version 6.0, check how Version 7.0 validates messages in these nodes.

About this task

You must complete this task if you are migrating from Version 6.0; no migration is required from Version 6.1.

In WebSphere Message Broker Version 7.0, message validation for HTTPRequest nodes has changed. The validation options that you can specify on an HTTPRequest node apply only to the response message that is received by the node from the remote server when an HTTP request is made. In Version 6.0, these options were also applied to the message received by the In terminal of the node.

The message received at the In terminal is no longer validated; however, the outbound (request) message can still be validated. Set validation options on a preceding Compute node, or include a Validate node. The HTTPRequest node uses the validation options associated with the message that is received at the In terminal to validate that message when it is sent out.

Related tasks:

"Migrating from Version 6.0 products" on page 183
Migrate your components and resources to WebSphere Message Broker Version 7.0.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"HTTPRequest node" on page 4488
Use the HTTPRequest node to interact with a web service.

"Validate node" on page 4959
Use the Validate node to check that the message that arrives on its input terminal is as expected. You can use this node to check that the message has the expected message template properties, and to check that the content of the message is correct by selecting message validation.

Migrating a flow containing XMLTransformation nodes

When you migrate a message flow that contains XMLTransformation nodes, check how WebSphere Message Broker Version 7.0 uses the node properties and search priorities.

About this task

You must complete this task if you are migrating from Version 6.0; no migration is required from Version 6.1.

The XMLTransformation node in Version 6.0 is replaced by the XSLTransform node in Version 7.0. When you migrate a flow that contains one or more XMLTransformation nodes, consider the following enhancements:

Procedure

- You can no longer set the style sheet search and character set information search priorities in the node properties in the WebSphere Message Broker Toolkit. The following fixed search priorities are adopted. However, search priorities set on existing flows are still respected.
  - The node searches for the name of the style sheet to be used by interrogating each of the following items in turn:

1. The input message. The node searches the message XML data for stylesheet location information. For example, if the XML data contains this entry:

   ```
   <?xml-stylesheet type="text/xsl" href="aaa.xsl"?>
   ```

   the node uses "aaa.xsl" as the stylesheet name.
2. The local environment. If no stylesheet name is found in the input message, the node searches the local environment associated with the current message for stylesheet information that is stored in an element called XSL.StyleSheetName.

   In Version 6.0, element ComIbmXslXmltStylesheetname is used for the name of the style sheet, therefore the current node checks both elements. If both are present, the value in XSL.StyleSheetName takes precedence.
3. The properties of the node. If no stylesheet name is found in the input message or local environment, the node uses the `Stylesheet Name` and `Stylesheet Directory` properties to determine the correct values.

– The node searches for the character set to use for the output message by interrogating each of he following items in turn:

1. The local environment. The node searches the local environment associated with the current message for character set information that is stored in an element called XSL.OutputCharSet; for example, to encode the output of the transformation as UTF-16, enter the value `1200` as a string in this element.

   In Version 6.0, element ComIbmXslXmltOutputcharset is used for the output character set, therefore the current node checks both elements. If both are present, the value in XSL.OutputCharSet takes precedence.
2. The properties of the node. If no character set information is found in the local environment, the node uses the `Output Character Set` property to determine the correct value.

   If the node cannot determine the output character set from either of these two sources, either because no value is set or the selection priorities are set to zero, the default value of `1208` (UTF-8) is used.

- You can set additional node properties, or use additional local environment overrides, to specify the message domain, message set, message type, and message format of the output message. If you have included a ResetContentDescriptor node to set these values before passing the message to the XSLTransform node, you can now optimize the message flow by removing the ResetContentDescriptor node.

Related tasks:

"Migrating from Version 6.0 products" on page 183
Migrate your components and resources to WebSphere Message Broker Version 7.0.

Related reference:

"ResetContentDescriptor node" on page 4663
Use the ResetContentDescriptor node to request that the message is reparsed by a different parser.

"XSLTransform node" on page 4968
Use the XSLTransform node to transform an XML message to another form of message, according to the rules provided by an XSL (Extensible Stylesheet Language) style sheet, and to set the `Message domain`, `Message set`, `Message type`, and `Message format` for the generated message.

Migrating a flow that contains File nodes

If you migrate message flows that contain File nodes to WebSphere Message Broker Version 7.0, ensure that your NFS server appropriately supports file locking.

About this task

If you use an NFS server, and have File nodes in different execution groups in WebSphere Message Broker Version 7.0 that access the same directory on the NFS server, ensure that you are using NFS version 4 to correctly support file locking.

Related tasks:

"Working with files" on page 1807

"Problems when developing message flows with file nodes" on page 3402
Use the advice given here to help you to resolve some common problems that can arise when you develop message flows that contain file nodes.

Related reference:

"FileInput node" on page 4415
Use the FileInput node to process messages that are read from files.

"FileOutput node" on page 4430
Use the FileOutput node to write messages to files.

"FileRead node" on page 4444
Use the FileRead node to read one record, or the entire contents of a file, from within a message flow.

Migrating a flow containing data definitions

When you migrate a message flow that uses data definitions from Version 6.0, Version 7.0 converts and replaces those definitions.

About this task

You must complete this task if you are migrating from Version 6.0; no migration is required from Version 6.1.

WebSphere Message Broker Version 7.0 manages data definitions in a data design project. A single file in the project (a `.dbm` file) represents the database. This file replaces the set of `.xmi` files that you created for each data definition in the Version 6.0 workbench.

- If you are migrating from Version 6.0, migration is performed only when you rebuild the message flow project in the WebSphere Message Broker Toolkit. Therefore, you must start a Version 7.0 WebSphere Message Broker Toolkit session to migrate the data definitions. See "Migrating data definitions in the workbench" on page 218.

Because the project files are changed by the migration process, and the old format files are deleted, back up your resources before you start to use existing resources in the Version 7.0 WebSphere Message Broker Toolkit.

When migration is complete, you can view data design projects in the Broker Application Development perspective, and you can view the contents of the `.dbm` files in the Data Project Explorer and Data Source Explorer views.

If a data definition is not migrated successfully, the database might no longer be supported. Check the version of the database that is referenced by the data

definition against the “Supported databases” on page 3591, and upgrade your database if necessary.

Migrating data definitions in the workbench

About this task

Complete the following steps:

Procedure

1. Start the Version 7.0 WebSphere Message Broker Toolkit.
2. Import the message flow that contains data definitions in `.xmi` files. You are asked to confirm that you want migration to be performed.
3. Select **Do not show this message again** if you want all projects to be migrated automatically when you import or rebuild them, and click **Yes** to continue with migration. The project is rebuilt, all data definitions are converted to the `.dbm` file format, and all database references from ESQL modules and mappings are replaced.

   To reset the option that suppresses the confirmation dialog, click **Windows** > **Preferences**, open **Broker Development**, and click **Database Definition**. Change the setting on this panel.
4. Clean all your projects after migration:
   a. Click **Project** > **Clean**.
   b. Select **Clean all projects**.
   c. Click **OK**.
5. Check the results of the migration. Errors and warnings are displayed in the Problems view.

Results

If you decide not to continue with migration, some errors might be generated in ESQL modules (`.esql` files) and in mappings (`.msgmap` files) that refer to databases. You must migrate the database resources in the project before you can deploy the project. You can start migration later in two ways:

- Rebuild the message flow project.
- Right-click the Database Connections folder and select **Database Definition Migration**. (The Database Connections folder appears in the toolkit only if you have not migrated all your database definitions, and it is removed when all migration has completed.)

How the migration works

About this task

Changes are made to the workspace resources as follows:

Procedure

- A new data design project is created if database definitions are found in the project. The first data design project name is set to the message flow project name, with `_DDP` appended; for example, `FlowProject1_DDP`.

  Each database definition is converted to Version 7.0 format. That is, the set of `.xmi` files is converted and stored as a single file called *`database_name`*`.dbm`. Multiple unique database definitions are converted and stored in a single data design project.

  You can rename the `.dbm` file, or the data design project.

- If you have created duplicate database definitions for the same database in the message flow project, another data design project is created for each definition.

  In Version 6.0, you can create multiple database definitions for the same database within a single message flow project. In Version 7.0 you can have only one definition, for each database, to ensure that mappings are unambiguous.

  You can create separate data definitions for the same database in separate data design projects, but you can refer to only one of these definitions from a single message flow project.

  The `.xmi` files are converted to a single `.dbm` file and each duplicate definition is stored in a separate data design project. The names of additional data design projects are also based on the name of the message flow project, and are appended with `_DDP2`, `_DDP3`, and so on.

  You might have to modify your message flow projects to ensure that all database references are to the same definition of the database.
- All old database definition files are deleted from the message flow project and from the file system.
- The first data design project that is created is added as a project reference to the message flow project.

  If data design projects have been created for duplicate database definitions, you cannot refer to these projects from the same message flow project.

  If you change the name of the data design project, you must modify the project reference in the message flow project.
- During conversion, information and errors are recorded in a log. Errors are also displayed in a dialog to indicate that migration has failed for a definition, and the cause of the failure. After migration, view the log file `migratedbdefinitions.log` in the `.metadata/.plugins/com.ibm.etools.mft.rdb` directory.

  The log contains details of the data design projects that have been created, the `.dbm` files that have been added to those projects, and the `.xmi` files that have been deleted.

  If you migrate these resources at a later date, for example when you import a different message flow project, the log records are appended to the existing content.

Related concepts:

"Resources" on page 36
The projects, folders, and files that you work with in the WebSphere Message Broker Toolkit workspace are called *resources*. By default, these resources are stored with their metadata in the workspace directory in your local file system. The workspace directory is created the first time that you start the WebSphere Message Broker Toolkit.

Related tasks:

"Migrating from Version 6.0 products" on page 183
Migrate your components and resources to WebSphere Message Broker Version 7.0.

"Adding a database as a source or target" on page 2274
Add a database as a source, and database tables as targets, to message maps that support database mappings.

"Adding database definitions to the WebSphere Message Broker Toolkit" on page 2278
Use the New Database Definition File wizard to add database definitions to the WebSphere Message Broker Toolkit.

"Modifying databases using message mappings" on page 2277
Create message mappings to read, update, and write to databases.

Related reference:

"Supported databases" on page 3591
You can optionally configure databases to contain data that is accessed by your message flows. Databases from IBM and other suppliers are supported at specific versions on supported operating systems.

Migrating a flow supporting ?wsdl queries

If you want WSDL and XML Schema information to be made available for existing SOAPInput-SOAPReply flows that implement web services, you must explicitly set the SOAPInput node property `Enable support for ?wsdl` and redeploy the flow.

About this task

In the unlikely event you have implemented an HTTPInput-HTTPReply flow to support `?wsdl` queries related to a SOAPInput-SOAPReply flow, you should now deprecate the HTTP flow. By sending `?wsdl` requests directly to the endpoint exposed by the SOAPInput node, this capability will continue to work even if the internal WSDL deployment details change in the future.

If you migrate your HTTP flow to use the embedded HTTP listener, the HTTPInput URL will clash with the SOAPInput URL, and a warning is written to the event log. In this case the `?wsdl` request is always serviced correctly, but web service requests could be sent to either the SOAPInput or the HTTPInput node.

Related concepts:

"Using WSDL to configure message flows" on page 1664
You can use WSDL to configure message flows.

Related reference:

"SOAPInput node" on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

"HTTPInput node" on page 4474
Use the HTTPInput node to receive an HTTP message from an HTTP client for processing by a message flow.

Migrating CMP applications

If you have written applications that use the CMP API, check that they access the correct resources.

Procedure

1. Update your CMP applications to use the file that is supplied by Version 7.0. You cannot use the `ConfigManagerProxy.jar` file from an earlier version to work with a Version 7.0 broker.

   You do not have to change the methods that the application uses; the Version 7.0 CMP API includes all deprecated classes and takes appropriate action.
2. Update the connection details that are used by your CMP applications to connect to the appropriate broker.

   If the application connects to a broker that is running on the same queue manager that was previously used by the Configuration Manager to which it connected, no change is required.

Related concepts:

"The Administration API for WebSphere Message Broker" on page 54
The Administration API for WebSphere Message Broker is a programming interface that your applications can use to control brokers and their resources through a remote interface.

"The Administration API samples" on page 958
Explore the samples to learn the basic features that are provided by the Administration API (also known as the CMP API). Run the samples to deploy a BAR file or manage a broker, or use the CMP API Exerciser to implement various tasks.

Related tasks:

"Developing applications that use the Administration API" on page 956
Develop Java applications that use the Administration API (also known as the CMP API) to communicate with, deploy to, and manage brokers and their associated resources.

"Configuring an environment for developing and running CMP applications" on page 969
Prepare the environment in which you want to run your CMP applications.

Updating error processing routines

Update your message processing to handle new or reused messages, and to delete obsolete messages.

About this task

WebSphere Message Broker Version 7.0 components generate new diagnostic messages (BIP messages), and no longer generates others that were reported in previous versions. The text of some messages have also been updated to reflect change in behavior and function in Version 7.0, although the meaning has been retained. One message, BIP8663, has been reused and has different content and meaning.

If you have applications that check for specific messages, you must update these routines. For example, you might have programs that automate error reporting, or identify the occurrence of specific error conditions.

For details of the content of all messages generated by Version 7.0, see Diagnostic messages.

Version 7.0 generates the following new messages:

- **Configuration:** messages in the range 1000 - 1999:

  ```
  BIP1014, BIP1054, BIP1060, BIP1115, BIP1117
  BIP1181 to BIP1191
  BIP1251 to BIP1269
  BIP1280 to BIP1295
  BIP1810
  ```

- **Broker:** messages in the range 2000 - 2999:

  ```
  BIP2082, BIP2289, BIP2319, BIP2358
  BIP2372, BIP2676, BIP2829, BIP2830
  BIP2850 to BIP2866
  BIP2870 to BIP2872
  BIP2880 to BIP2883
  BIP2890 to BIP2894
  ```

- **Built-in nodes:** messages in the range 3000 - 3999:

  BIP3226
  BIP3432 to BIP3439
  BIP3451 to BIP3463
  BIP3466 to BIP3471
  BIP3524 to BIP3526
  BIP3574, BIP3635, BIP3687
  BIP3747 to BIP3760
  BIP3769 to BIP3831
  BIP3915 to BIP3925

- **Built-in nodes:** messages in the range 4000 - 4999:

  BIP4015 to BIP4017
  BIP4071 to BIP4076
  BIP4251 to BIP4254
  BIP4359
  BIP4513 to BIP4516
  BIP4678
  BIP4821 to BIP4829
  BIP4833 to BIP4865

- **Parsers:** messages in the range 5000 - 5999:

  BIP5031, BIP5637

- **WebSphere MQ parsers:** messages in the range 6000 - 6999:

  BIP6069, BIP6265, BIP6266

- **Publish/subscribe:** messages in the range 7000 - 7999:

  BIP7036 to BIP7038
  BIP7042 to BIP7045
  BIP7099
  BIP7120 to BIP7123

- **Commands:** messages in the range 8000 - 8999:

  BIP8157, BIP8200
  BIP8231 to BIP8239
  BIP8769 to BIP8783
  BIP8919 to BIP8923
  BIP8934 to BIP8940
  BIP8990 to BIP8993

- **z/OS:** messages in the range 9000 - 9999:

  BIP9284
  BIP9286 to BIP9290

Version 7.0 reuses the message BIP8663.

Version 7.0 components no longer generate the following messages:

- **Configuration:** messages in the range 1000 - 1999:

  BIP1002 to BIP1013
  BIP1015 to BIP1016
  BIP1020 to BIP1023

```
BIP1029, BIP1032, BIP1037, BIP1040
BIP1042 to BIP1045
BIP1055 to BIP1057
BIP1059, BIP1070
BIP1072 to BIP1088
BIP1090, BIP1094
BIP1096 to BIP1107
BIP1109 to BIP1112
BIP1126, BIP1128, BIP1131, BIP1132, BIP1145, BIP1146
BIP1151 to BIP1165
BIP1170 to BIP1173
BIP1201 to BIP1211
BIP1213 to BIP1217
BIP1221 to BIP1230
BIP1301 to BIP1306
BIP1350 to BIP1353
BIP1355 to BIP1363
BIP1401 to BIP1415
BIP1449, BIP1450, BIP1461, BIP1462
BIP1502 to BIP1528
BIP1530 to BIP1552
BIP1557 to BIP1571
BIP1581 to BIP1589
BIP1702 to BIP1703
BIP1705 to BIP1708
BIP1711 to BIP1712
BIP1751 to BIP1760
BIP1762, BIP1764, BIP1765
BIP1767 to BIP1773
BIP1775 to BIP1781
BIP1806 to BIP1807
```

- **Broker:** messages in the range 2000 - 2999:

```
BIP2012, BIP2013
BIP2025 to BIP2028
BIP2032, BIP2035, BIP2036, BIP2040
BIP2048, BIP2049, BIP2053
BIP2072, BIP2073, BIP2090
BIP2092 to BIP2098
BIP2220, BIP2221, BIP2223, BIP2224, BIP2317
BIP2814 to BIP2816
```

- **Built-in nodes:** messages in the range 4000 - 4999:

```
BIP4046
```

- **Parsers:** messages in the range 5000 - 5999:

```
BIP5315
```

- **Publish/subscribe:** messages in the range 7000 - 7999:

```
BIP7003, BIP7007, BIP7011, BIP7012
BIP7028, BIP7029, BIP7060, BIP7090
```

- **Commands:** messages in the range 8000 - 8999:

```
BIP8058, BIP8060, BIP8061, BIP8066, BIP8077, BIP8078
BIP8088, BIP8089, BIP8090
```

```
BIP8102, BIP8103, BIP8106, BIP8107
BIP8109, BIP8110, BIP8112
BIP8118, BIP8119, BIP8148, BIP8180
BIP8201 to BIP8210, BIP8214,
BIP8250 to BIP8263, BIP8286
BIP8301 to BIP8311
BIP8380 to BIP8382
BIP8390 to BIP8394
BIP8617, BIP8618, BIP8620
BIP8664 to BIP8668
BIP8672, BIP8674, BIP8675, BIP8678, BIP8795, BIP8796
BIP8829 to BIP8832
BIP8834 to BIP8839
BIP8842 to BIP8845
BIP8925
BIP8930 to BIP8933
BIP8995
```

- **z/OS:** messages in the range 9000 - 9999:

```
BIP9144 to BIP9146
BIP9801 to BIP9842
```

Related tasks:

Chapter 3, “Migrating and upgrading,” on page 137
To migrate a broker domain to WebSphere Message Broker Version 7.0, plan your migration strategy, perform pre-migration tasks, migrate your domain components, and then complete post-migration tasks.

Restoring migrated components to previous versions

You can restore migrated brokers to previous versions of WebSphere Message Broker.

About this task

You can only restore brokers to the state they were in before you migrated them; changes that you have made to them after migration, such as updated properties, are lost.

You might want to restore components if you encounter a problem after you have migrated your brokers to Version 7.0. For example, if a message flow does not work as expected in Version 7.0, restore that broker until you can resolve the problem.

You can restore brokers to a previous version only if they were migrated from that version originally. You cannot create a new Version 6.1 broker, then restore it to a previous level. Nor can you create a Version 6.0 broker, migrate it to Version 7.0, then restore it to Version 6.1.

The topics in this section explain how to restore brokers and resources that you have migrated from Version 6.1 or Version 6.0 products back to their original state:

- “Restoring components and resources to Version 6.1” on page 225
- “Restoring components and resources to Version 6.0” on page 227

Because you do not migrate Configuration Manager or User Name Server components, you do not have to restore these. If you have not deleted them, they

retain their existing configuration, and can be reused with brokers that you restore to a previous version.

Related tasks:

Chapter 3, "Migrating and upgrading," on page 137
To migrate a broker domain to WebSphere Message Broker Version 7.0, plan your migration strategy, perform pre-migration tasks, migrate your domain components, and then complete post-migration tasks.

"Backing up WebSphere Message Broker resources" on page 167
Back up your resources before you start to migrate components to Version 7.0.

"Backing up WebSphere Message Broker Version 6.0 resources" on page 187
Back up your resources before you start to migrate components to Version 7.0.

"Uninstalling" on page 331
Remove the Broker component, the WebSphere Message Broker Toolkit, or the WebSphere Message Broker Explorer from your computer.

Restoring components and resources to Version 6.1

Restore components and resources that you have migrated from Version 6.1 products to their original state.

About this task

If you have migrated from WebSphere Message Broker Version 6.1 to Version 8.0, you can restore your installation to Version 6.1, subject to the following restrictions and procedures.

When you use the **mqsimigratecomponents** command, you must be logged in to a Version 8.0Version 7.0 command environment.

See the **mqsimigratecomponents** command description for usage information and details of parameters and the format to use when specifying version numbers.

Restrictions

Message flows that you have deployed to a broker after migration to Version 7.0 might no longer work when you restore that broker to Version 6.1; always check logs to search for errors or warnings about message flows that the broker cannot start.

Source files in the WebSphere Message Broker Toolkit Version 7.0 are maintained in a different format from that used in Version 6.1. The files are migrated to the new format when you save them in the WebSphere Message Broker Toolkit Version 7.0. After you have saved them, you can no longer use the resources with the WebSphere Message Broker Toolkit Version 6.1.

Changes that you made to brokers, the WebSphere Message Broker Toolkit, and development resources after migration to Version 7.0 are not retained when you restore your resources back to Version 6.1.

Migrating resources back to Version 6.1

The following sections describe how to restore the WebSphere Message Broker Toolkit and your brokers to Version 6.1. You can reuse your original Version 6.1 Configuration Manager and User Name Server components with the brokers that you restore to Version 6.1.

Restoring the WebSphere Message Broker Toolkit to Version 6.1

1. Close all WebSphere Message Broker Toolkit Version 7.0 sessions.
2. Restore the Version 6.1 workspace from the backup that you took before migration.

   Ensure that the workspace directories include the `.metadata` directory, which contains information that is specific to the release of Eclipse on which the toolkit is based.
3. Restart WebSphere Message Broker Toolkit Version 6.1.

Restoring brokers to Version 6.1

Use the **-s** and **-t** parameters of the **mqsimigratecomponents** command to migrate brokers from Version 7.0 to Version 6.1.

- Specify the installed level of Version 7.0 for the source version parameter (**-s**), for example `7.0.0.0` for the GA (general availability) level, or allow this to take the default value.
- Specify the appropriate level of Version 6.1 for the target version parameter (**-t**), for example `6.1.0.9` for Version 6.1, Fix Pack 9.

See the **mqsimigratecomponents** command for detailed information about these parameters and the format to use when specifying version numbers.

Restoring brokers on distributed systems

1. Stop the Version 7.0 broker by using the **mqsistop** command.
2. Restore the broker to Version 6.1 using the **mqsimigratecomponents** command, as shown in the following example:

   ```
   mqsimigratecomponents Broker -t 6.1.0.9
   ```
3. Reverse the changes that you made to the ODBC definitions when you migrated to Version 7.0; for more details, see "Updating your ODBC definitions when migrating" on page 188.
4. Start the Version 6.1 WebSphere Message Broker Toolkit, and add the broker to the configuration.
5. Open a Version 6.1 command window, and restart the broker by using the **mqsistart** command.

If you migrate to Version 7.0, deploy a message set to the Version 7.0 broker, and then migrate back to Version 6.1, Version 6.1 is unable to recognize the message set that was deployed by Version 7.0. In this case, all message sets that Version 6.1 is unable to use are deleted and the warning message BIP8688W is displayed for each message set, prompting you to redeploy it to Version 6.1 following successful migration.

Restoring brokers on z/OS

1. Stop the Version 7.0 broker by using one of the available options.
2. Submit the BIPMGCMP job to call the **mqsimigratecomponents** command, specifying the **-s** and **-t** parameters, as described for distributed systems.
3. Start the Version 6.1 WebSphere Message Broker Toolkit, and add the broker to the configuration.

4. Replace the started task JCL file in USER.PROCLIB with the Version 6.1 copy that you backed up.

Related tasks:

Chapter 3, "Migrating and upgrading," on page 137
To migrate a broker domain to WebSphere Message Broker Version 7.0, plan your migration strategy, perform pre-migration tasks, migrate your domain components, and then complete post-migration tasks.

"Backing up WebSphere Message Broker resources" on page 167
Back up your resources before you start to migrate components to Version 7.0.

"Uninstalling" on page 331
Remove the Broker component, the WebSphere Message Broker Toolkit, or the WebSphere Message Broker Explorer from your computer.

Related reference:

"**mqsimigratecomponents** command" on page 3894
Use the **mqsimigratecomponents** command to migrate a component from a previously installed version of the product to another version on the same computer.

Restoring components and resources to Version 6.0

Restore components and resources that you have migrated from Version 6.0 products to their original state.

About this task

If you have migrated from WebSphere Message Broker Version 6.0 to Version 7.0, you can restore your installation to Version 6.0, subject to the following restrictions and procedures.

When you use the **mqsimigratecomponents** command, you must be logged in to a Version 7.0 command environment.

See the **mqsimigratecomponents** command description for usage information and details of parameters and the format to use when specifying version numbers.

Restrictions

Message flows that you have deployed to a broker after migration to Version 7.0 might no longer work when you restore that broker to Version 6.0; always check logs to search for errors or warnings about message flows that the broker cannot start.

Source files in WebSphere Message Broker Toolkit Version 7.0 are maintained in a different format to that used in Version 6.0. The files are migrated to the new format when you save them in the WebSphere Message Broker Toolkit Version 7.0. After you have saved them, you can no longer use the resources in WebSphere Message Broker Toolkit Version 6.0. For detailed information, see "Conditions for using migrated resources with previous versions of the WebSphere Message Broker Toolkit" on page 229.

Changes that you made to brokers, the WebSphere Message Broker Toolkit, and development resources after migration to Version 7.0 are not retained.

Migrating resources back to Version 6.0

The following sections describe how to restore the WebSphere Message Broker Toolkit and your brokers to Version 6.0. You can reuse your original

Version 6.0 Configuration Manager and User Name Server components with the brokers that you restore to Version 6.0.

Restoring the WebSphere Message Broker Toolkit to Version 6.0

1. Close all WebSphere Message Broker Toolkit Version 7.0 sessions.
2. Restore the Version 6.0 workspace from the backup that you took before migration.

   Ensure that the workspace directories include the `.metadata` directory, which contains information that is specific to the release of Eclipse on which the toolkit is based.
3. Restart WebSphere Message Broker Toolkit Version 6.0.

Restoring brokers to Version 6.0

Use the **-s** and **-t** parameters of the **mqsimigratecomponents** command to migrate brokers from Version 7.0 to Version 6.0.

- Specify the installed level of Version 7.0 for the source version parameter (**-s**), for example `7.0.0.0` for the GA (general availability) level, or accept the default value.
- Specify the appropriate level of Version 6.0 for the target version parameter (**-t**), for example `6.0.0.9` for Version 6.0, Fix Pack 9.

See the **mqsimigratecomponents** command for detailed information about these parameters and the format to use when specifying version numbers.

Restoring brokers on distributed systems

1. Stop the Version 7.0 broker by using the **mqsistop** command.
2. Restore the broker to Version 6.0 by using the **mqsimigratecomponents** command, as shown in the following example:

   ```
   mqsimigratecomponents Broker -t 6.0.0.9
   ```
3. Reverse the changes that you made to the ODBC definitions when you migrated to Version 7.0.
4. Open a Version 6.0 command window, and restart the broker by using the **mqsistart** command.

If you migrate to Version 7.0, deploy a message set to the Version 7.0 broker, and then migrate back to Version 6.0, Version 6.0 is unable to recognize the message set that was deployed by Version 7.0. In this case, all message sets that Version 6.1 is unable to use are deleted and the warning message BIP8688W is displayed for each message set, prompting you to redeploy it to Version 6.1 following successful migration.

Restoring brokers on z/OS

1. Stop the Version 7.0 broker by using one of the available options.
2. Submit the BIPMGCMP job to call the **mqsimigratecomponents** command, specifying the **-s** and **-t** parameters as described previously.
3. Replace the started task JCL file in USER.PROCLIB with the Version 6.0 copy that you backed up.

Related concepts:

"Conditions for using migrated resources with previous versions of the WebSphere Message Broker Toolkit"
WebSphere Message Broker Version 7.0 can interoperate with components of previous versions. Conditions apply when you use migrated development and deployed resources with previous versions of the WebSphere Message Broker Toolkit.

Related tasks:

Chapter 3, "Migrating and upgrading," on page 137
To migrate a broker domain to WebSphere Message Broker Version 7.0, plan your migration strategy, perform pre-migration tasks, migrate your domain components, and then complete post-migration tasks.

"Backing up WebSphere Message Broker Version 6.0 resources" on page 187
Back up your resources before you start to migrate components to Version 7.0.

"Uninstalling" on page 331
Remove the Broker component, the WebSphere Message Broker Toolkit, or the WebSphere Message Broker Explorer from your computer.

Related reference:

"**mqsimigratecomponents** command" on page 3894
Use the **mqsimigratecomponents** command to migrate a component from a previously installed version of the product to another version on the same computer.

Conditions for using migrated resources with previous versions of the WebSphere Message Broker Toolkit

WebSphere Message Broker Version 7.0 can interoperate with components of previous versions. Conditions apply when you use migrated development and deployed resources with previous versions of the WebSphere Message Broker Toolkit.

Version 6.1 and Version 6.0 resources

Import Version 6.1 and Version 6.0 resources into your Version 7.0 workspace to view and change them. After you have changed and saved resources, you can no longer use them in your Version 6.1 or Version 6.0 WebSphere Message Broker Toolkit.

You do not have to perform any explicit migration tasks on imported resources.

Restrictions after migration

After you have migrated WebSphere Message Broker Toolkit resources to Version 7.0, your use of these resources in the Version 6.1 and Version 6.0 WebSphere Message Broker Toolkit is restricted.

Development resources

Resources in Version 6.1 and Version 6.0 formats can coexist in the Version 7.0 workspace.

Resources remain in their original format until you save them in the Version 7.0 WebSphere Message Broker Toolkit. After saving, all resources are stored in the Version 7.0 format.

You can no longer use these saved resources in earlier versions of the WebSphere Message Broker Toolkit. Therefore you cannot share development resources between the Version 7.0 and previous versions of the WebSphere Message Broker Toolkit.

If you expect to continue development of these resources, you must retain a WebSphere Message Broker Toolkit at the version that you require.

Deployment resources

You can deploy broker archive (BAR) files that you edit or create in a Version 7.0 WebSphere Message Broker Toolkit only to Version 7.0 brokers.

You can import resources that you created in previous versions into your Version 7.0 workspace, but when you include these resources in a BAR file, you can deploy that BAR file only to Version 7.0 brokers.

The following table summarizes which of the files that you can include in a BAR file are deployable and accepted by the broker at each version. A broker rejects broker archive files that contain file types that are supported only by later brokers.

File type	Version 6.0	Version 6.1	Version 7.0
Compiled message flows (`.cmf`)	Yes	Yes	Yes
Message dictionaries (`.dictionary`)	Yes	Yes	Yes
XSL style sheets (`.xsl`, `.xslt`, `.xml`)	Yes	Yes	Yes
Java archives (`.jar`)	Yes	Yes	Yes
WebSphere TX maps (`.mar`)	No	Yes	Yes
XSD archives (`.xsdzip`)	No	Yes	Yes
Inbound adapter configuration (`.inadapter`)	No	Yes	Yes
Outbound adapter configuration (`.outadapter`)	No	Yes	Yes
Inbound SCA definitions	No	No	Yes
Outbound SCA definitions	No	No	Yes

The Version 7.0 WebSphere Message Broker Toolkit can coexist with the Version 6.1 and Version 6.0 WebSphere Message Broker Toolkit on the same computer. However, because you cannot share development resources between the Version 7.0 WebSphere Message Broker Toolkit and previous versions of the WebSphere Message Broker Toolkit, you might find that it is appropriate to create and maintain a different workspace for the Version 7.0 WebSphere Message Broker Toolkit.

Related concepts:

"Coexistence with previous versions and other products" on page 139
WebSphere Message Broker Version 7.0 supports restricted coexistence with previous versions and with other products.

Related tasks:

Chapter 3, "Migrating and upgrading," on page 137
To migrate a broker domain to WebSphere Message Broker Version 7.0, plan your migration strategy, perform pre-migration tasks, migrate your domain components, and then complete post-migration tasks.

Chapter 4. Installing and uninstalling

Install and uninstall WebSphere Message Broker components and service.

Find out where you can see the latest information about installation of WebSphere Message Broker components, and how to apply service. You can also view information about complementary products. This section also describes how to remove service (not available on all operating systems), and how to uninstall components.

- "Installing"
- "Uninstalling" on page 331

Related tasks:

Chapter 7, "Configuring brokers for test and production environments," on page 579
Create one or more brokers on one or more computers, and configure them on your test and production systems to process messages that contain your business data.

Chapter 3, "Migrating and upgrading," on page 137
To migrate a broker domain to WebSphere Message Broker Version 7.0, plan your migration strategy, perform pre-migration tasks, migrate your domain components, and then complete post-migration tasks.

Related reference:

"Installation" on page 3581
Use the reference information in this section to understand installation requirements, installation options, and how they affect your computer.

Installing

Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

About this task

See "Finding the latest information" on page 232 for details about how to check that you have access to the most recent information available.

The "Installation Guide" on page 233 defines hardware and software requirements for WebSphere Message Broker and its corequisite and prerequisite products. It describes the tasks that you must complete to prepare for installation, to install WebSphere Message Broker, and to verify the installation. When you have completed installation, use this information center to create and configure your components and brokers.

If product fixes or updates are made available, refer to the following topics for information about how to apply these changes:

- "Applying service to the Broker component" on page 314
- "Applying service to the WebSphere Message Broker Toolkit" on page 325
- "Applying service to the WebSphere Message Broker Explorer" on page 329

You do not have to apply service to the runtime components and the WebSphere Message Broker Toolkit at the same time; all Version 7.0 fix pack levels are compatible with all other fix pack levels. However, if a fix pack delivers additional nodes, you must ensure that the WebSphere Message Broker Toolkit and the brokers to which you deploy the message flows that include these nodes are at the same fix pack level, and that you have set the correct function level of those brokers by using the **-f** flag on the **mqsichangebroker** command.

To install complementary products, see "Installing complementary products" on page 300.

Related tasks:

Chapter 7, "Configuring brokers for test and production environments," on page 579
Create one or more brokers on one or more computers, and configure them on your test and production systems to process messages that contain your business data.

"Uninstalling" on page 331
Remove the Broker component, the WebSphere Message Broker Toolkit, or the WebSphere Message Broker Explorer from your computer.

"Resolving problems when installing" on page 3517
Work through the advice provided to help you to deal with problems that can arise when the product is installed.

"Finding the latest information"
Access the latest information for WebSphere Message Broker.

Related reference:

"Installation" on page 3581
Use the reference information in this section to understand installation requirements, installation options, and how they affect your computer.

"Support for 32-bit and 64-bit platforms" on page 3589
WebSphere Message Broker operates in 32-bit mode or 64-bit mode, on supported operating systems.

Related information:

WebSphere Message Broker Library web page

Finding the latest information

Access the latest information for WebSphere Message Broker.

About this task

The following information is provided:

Requirements website
: For the latest details of hardware and software requirements on all supported platforms, visit the WebSphere Message Broker Requirements website.

readme.html
: The product readme file is frequently updated and includes information about last minute changes and known problems and workarounds. The latest version is always on the product readmes web page; always check to see that you have the latest copy. The version of file that is included on the product media, and which is installed when you install product components, contains a link to the latest version on the product readmes web page.

Installation Guide
The Installation Guide is provided as a PDF file with your product. See "Installation Guide" for details about how to access the guide online and where to find installation information in this information center.

Information center
The information center is installed with the WebSphere Message Broker Toolkit and the WebSphere Message Broker Explorer, and updates are typically included when you apply service to those components.

The information center is periodically updated independently of the code, and you can install the latest level from within the toolkit. For instructions about installing code and documentation updates, see "Applying service to the WebSphere Message Broker Toolkit" on page 325.

The information center is also available online at WebSphere Message Broker Library web page.

Support information
The WebSphere Message Broker support web page is regularly updated with the latest product support information. For example, if you are migrating from an earlier version, look under the heading "Solve a problem" for the document "Problems and solutions when migrating".

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

"Applying service to the Broker component" on page 314
Apply maintenance updates and program fixes to the Broker component.

"Applying service to the WebSphere Message Broker Toolkit" on page 325
Apply maintenance updates and program fixes to the WebSphere Message Broker Toolkit.

Related reference:

"Installation Guide"
Installation information for WebSphere Message Broker is provided in the *Installation Guide* that is supplied as a PDF file with your product package.

"Software requirements" on page 3588
View the operating system, database, and other software requirements.

"Hardware requirements" on page 3582
View the processor and related hardware requirements on all platforms.

Installation Guide

Installation information for WebSphere Message Broker is provided in the *Installation Guide* that is supplied as a PDF file with your product package.

The installation information provided by the *Installation Guide* is also contained in the information center:

See "Installation" on page 3581 for reference information about system requirements, including operating systems, hardware, and additional software and memory requirements.

The *Installation Guide* provides general preparation, planning, and security information for all platforms. It also provides installation instructions for distributed systems, common problems and solutions, and the names and locations of the installation log files. Verification programs on Linux on x86 and Windows are also provided.

If you are installing on z/OS, the equivalent installation instructions are provided in the Program Directory that is supplied in hardcopy with your product media.

The *Installation Guide* does not cover configuration or customization tasks on any operating system; it describes only how you install the product components onto your computers. When you have completed installation on distributed systems, you must initialize the local environment before you can create or configure resources. This task is described in "Setting up a command environment" on page 213. Environment initialization is not required on z/OS. For information about designing and configuring your broker environment on all systems, see Chapter 7, "Configuring brokers for test and production environments," on page 579.

The *Installation Guide* PDF file is provided as a convenience for printing and offline reading. You can find the *Installation Guide* PDF file in the following locations after a major release of the information center, such as Version 7.0:

- In English, on the WebSphere Message Broker customer FTP site. To access the PDF file on the customer FTP site, see WebSphere Message Broker Installation Guide.
- In English, from the WebSphere Message Broker Library web page.
- In English and supported languages, on the Quick Start CD.
- In English and supported languages, with your product package.

If significant changes are accumulated between major releases, the *Installation Guide* PDF file is refreshed and provided in English, on the customer FTP site, and the Library web page.

The PDF file content is the same as the content that is provided in the information center at the time of publish. However, the PDF file documentation is updated less frequently than the information center. For the latest information, see the "Installation" on page 3581 and "Installing" on page 231 sections in the online information center.

You can view, search, and print PDF files by using Adobe Reader. To download Adobe Reader, see the Adobe Systems Inc. website. For more information about printing the *Installation Guide* PDF file, see Printing information center topics and the Installation Guide PDF file.

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

"Resolving problems when installing" on page 3517
Work through the advice provided to help you to deal with problems that can arise when the product is installed.

"Setting up a command environment" on page 213
After you have installed the product on one of the distributed systems, you must

initialize the environment before you can use a runtime component or command.

Chapter 7, “Configuring brokers for test and production environments,” on page 579
Create one or more brokers on one or more computers, and configure them on your test and production systems to process messages that contain your business data.

“Applying service to the Broker component” on page 314
Apply maintenance updates and program fixes to the Broker component.

“Applying service to the WebSphere Message Broker Toolkit” on page 325
Apply maintenance updates and program fixes to the WebSphere Message Broker Toolkit.

“Applying service to the WebSphere Message Broker Explorer” on page 329
You can apply maintenance or fixes to the WebSphere Message Broker Explorer.

“Uninstalling” on page 331
Remove the Broker component, the WebSphere Message Broker Toolkit, or the WebSphere Message Broker Explorer from your computer.

“Resolving problems when uninstalling” on page 3525
Work through the advice provided to help you to deal with problems that can arise when the product is uninstalled.

“Finding the latest information” on page 232
Access the latest information for WebSphere Message Broker.

Related reference:

“Installation” on page 3581
Use the reference information in this section to understand installation requirements, installation options, and how they affect your computer.

Preparing for installation

Use the instructions in this tutorial for an overview of the installation tasks, and to prepare for installation of WebSphere Message Broker.

About this task

The tasks that you perform to complete installation are listed here; each task indicates whether it is required or optional. A summary of each task is provided, along with pointers to later chapters or sections which describe that task in more detail.

Procedure

1. Required: Make sure that you have acquired the product packages that you need for installation.

   Physical and electronic packages are available for WebSphere Message Broker Version 7.0.

   For more information about available packages and their contents, see “Installation packages” on page 3608. If you have ordered the product for electronic delivery from IBM Passport Advantage®, check that you have downloaded all the images that you need for all components and all platforms.

   Both DVDs and CDs are supplied in the physical package, where as only CD images are available from IBM Passport Advantage. The CD images that are available from IBM Passport Advantage contain the same content as the physical DVDs, but might be greater in number.

For example, a physical DVD might contain installation files for the Broker component, WebSphere Message Broker Toolkit, and WebSphere MQ, where as IBM Passport Advantage might provide a CD image for the Broker component, three images for the WebSphere Message Broker Toolkit, and another image for WebSphere MQ.

Find out how to access, download, and extract product images in "Accessing CDs and DVDs" on page 252.

For instructions about downloading and applying service updates, see "Applying service" on page 313.

2. Required: Make sure that you have access to the product documentation that you need for installation.
   - The product readme file `readme.html` contains the latest available information.
   - The information center describes planning and preparation on all platforms. The sections of the information center you need for installation are also available as a PDF file on the Quick Start CD.
   - This information center describes installation procedures for all components on all distributed platforms. The sections of the information center that you need for installation are also available as a PDF file on the Quick Start CD.
   - The *Program Directory for WebSphere Message Broker for z/OS* describes installation procedures for all components on z/OS.

   Find out how to get these documents in "Finding the latest information" on page 232.
3. Required: Decide which components you want to install on your computers.

   The following information provides a minimum level of detail about WebSphere Message Broker components; read "Choosing what to install" on page 260 to find out more about the components.
   - The WebSphere Message Broker Toolkit. You must install this component on at least one computer. You can install this component only on Windows 32-bit, Windows 64-bit, Linux on x86, and Linux on x86-64.

     Use the WebSphere Message Broker Toolkit to create, manage, deploy, and delete message flows and associated resources in a development environment.
   - The WebSphere Message Broker Explorer. You can install this component only on Windows 32-bit, Windows 64-bit, Linux on x86, and Linux on x86-64.

     Use the WebSphere Message Broker Explorer to administer your brokers in a production environment.
   - The Broker component. You must install this component on at least one computer.

     You can create multiple brokers on a single computer. Deploy your message flow resources to one or more brokers to process your application messages.
4. Required: Decide which platform you want to install your chosen components onto.

   The following table shows the components that you can install on the supported platforms.

Table 1. Summary of component and platform support

Component	Platform
WebSphere Message Broker Toolkit	• Linux on x86 • Linux on x86-64 • Windows 32-bit • Windows 64-bit
WebSphere Message Broker Explorer	• Linux on x86 • Linux on x86-64 • Windows 32-bit • Windows 64-bit
Broker component	All supported platforms

5. Required: Prepare each computer on which you are installing one or more components.
   a. Check that your target computers meet the initial hardware, storage, and software requirements.

      The requirements vary depending on what computers you want to install WebSphere Message Broker on, and what components you are installing; read the details in "Hardware requirements" on page 3582 and "Operating system requirements" on page 3590.

      The supported hardware and software environments are updated occasionally. To view the latest requirements, see the product requirements website:

      `www.ibm.com/software/integration/wbimessagebroker/requirements/`
   b. Complete the appropriate tasks for your computer to set up security and UNIX kernel configuration, and get ready to access the installation media.

      All these tasks are described in "Preparing the system" on page 245.
   c. Check that your user ID has the required authority to complete installation:
      - AIX systems: Use the user ID root for installation.
      - Linux and other UNIX systems: Use either the user ID root, or another ID and become root.
      - Windows systems: Your user ID must be a member of the group Administrators.
      - z/OS systems: Use a user ID that has suitable RACF privileges to perform installation.

      This information is a summary only; more details are provided in "Setting up security" on page 245.
6. Required: Check the programs that you use to perform component installation.

   The following table lists the programs that are available.

Table 2. Summary of available installation methods

Platform	Tools
Windows only	The Windows Launchpad. This program installs prerequisite products if they are not already installed, and identifies prerequisite products that are not at the supported level. See "Installing by using the Windows Launchpad" on page 262.

Table 2. Summary of available installation methods (continued)

Platform	Tools
Linux, UNIX, and Windows systems	Installation wizards on each supported platform have unique names, which are listed in "Installation wizard names" on page 3626. • To install the Broker component, see "Installing the Broker component" on page 267. • To install the WebSphere Message Broker Toolkit, see "Installing the WebSphere Message Broker Toolkit" on page 276. • To install the WebSphere Message Broker Explorer, see "Installing WebSphere Message Broker Explorer" on page 280.
z/OS only	SMP/E To install runtime components, see the *Program Directory for WebSphere Message Broker for z/OS*.

7. Required: Install additional products that are required by WebSphere Message Broker.

   WebSphere Message Broker requires other software products to work successfully. The order in which you install these products is not important. However, you must install all required products before you can configure and start WebSphere Message Broker components.

   The following table gives a summary of these requirements.

Table 3. Summary of prerequisite products

Component	Prerequisite products
WebSphere Message Broker Toolkit	• A web browser to view the information center.
WebSphere Message Broker Explorer	• Install WebSphere MQ Explorer. • A web browser to view the information center.
Broker component	• WebSphere MQ to communicate with other components. • A Java Runtime Environment (JRE).

   Full details of all these requirements are provided in "Additional software requirements" on page 3598:

   - For web browsers, see "Browsers" on page 3600.
   - For details of supported WebSphere MQ versions, see "WebSphere MQ" on page 3598.
   - For further information about JREs, see "JRE" on page 3599.

8. Optional: Configure a minimum broker domain and verify its operation.
   a. To create a minimum broker domain, install the WebSphere Message Broker Toolkit and the Broker component on a single computer. Because the WebSphere Message Broker Toolkit is required to complete this task, you must choose a Linux on x86, Linux on x86-64, or Windows computer.
   b. Use the Default Configuration wizard, which you can start from the WebSphere Message Broker Toolkit, to create the required components after installation.
   c. Use this broker domain to create message flow resources, verify your installation, investigate how the product works, and explore the product samples.

The verification process is described in "Verifying your WebSphere Message Broker installation" on page 290.

9. Optional: Change the broker operation mode.

   When you install WebSphere Message Broker and create brokers, they are configured with an operation mode set either to `trial` (if you have installed the Trial Edition) or `enterprise` (all other editions). You must configure your brokers to conform to the license that you have purchased. Therefore, if you have purchased the Starter Edition, Entry Edition, or the Remote Adapter Deployment, you must set the operation mode of all your brokers to the correct value.

   See "Checking the broker operation mode and function level" on page 298 for more details.

10. Optional: Change the broker function level.

    If new message flow nodes are delivered in a fix pack, they show in the WebSphere Message Broker Toolkit, but are not enabled in the runtime broker environment. If you deploy a BAR file that includes a message flow that uses a new node, the deployment fails.

    If you want to use and test the new nodes, you can enable them on an individual broker basis. See "Checking the broker operation mode and function level" on page 298 for more details.

Results

You have completed the tutorial.

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

Related reference:

"Installation" on page 3581
Use the reference information in this section to understand installation requirements, installation options, and how they affect your computer.

Related information:

WebSphere Message Broker Requirements

Coexistence and migration

WebSphere Message Broker Version 7.0 can coexist with previous versions.

You can install WebSphere Message Broker Version 7.0 on a computer on which you have installed previous versions, but each version must be installed into its own directory, referred to as *install_dir*. Different versions can coexist and can run independently, and you can migrate brokers from one version to another, if and when appropriate. For the Broker component and WebSphere Message Broker Toolkit, you can install multiple instances of the same version on the same computer, each in its own separate directory. For example, the Broker component can coexist with different fix pack levels, such as V7.0.0.0 with V7.0.0.1. You can install multiple instances of the same version of the Broker component and WebSphere Message Broker Toolkit on the same computer, however only one version of WebSphere Message Broker Explorer is allowed.

The following sections provide further details:
- "Coexistence"
- "Migration" on page 244

Coexistence

WebSphere Message Broker Version 7.0 can coexist with previous versions as follows:

- The Broker component can coexist with runtime components at Version 6.0, and Version 6.1.
- The WebSphere Message Broker Toolkit can coexist with the toolkit at Version 6.0, and Version 6.1.

The following sections describe how to achieve coexistence, and the restrictions that apply.

Broker component on distributed systems

When you install the Broker component on distributed systems, the default action taken by the installation wizard is to complete a `typical installation`, which installs the Broker component into a default directory. The default directory for a typical installation is fixed and you cannot change it.

If you accepted the default location during installation, this directory is as follows. The default directory includes the version and release of the product that you are installing in the format v.r (version.release):

Linux `/opt/ibm/mqsi/v.r`

UNIX `/opt/IBM/mqsi/v.r`

Windows 32-bit
`C:\Program Files\IBM\MQSI\v.r`

Windows 64-bit
`C:\Program Files\IBM\MQSI\v.r` for the 64-bit version of WebSphere Message Broker

`C:\Program Files (x86)\IBM\MQSI\v.r` for the 32-bit version of WebSphere Message Broker

These locations define the default value of *install_dir* on each platform.

Each unique version and release of the product is therefore installed into a different default location.

The installation wizard differentiates only at version and release level; it does not differentiate between different modification levels and fix pack levels. The current modification level is 0 (Version 7.0.0). If a later modification level is made available, it installs into the same default location, and therefore upgrades the Version 7.0.0 to the higher modification level (for example, Version 7.0.1).

The installation wizard installs one fix pack over a previous fix pack, but prohibits you from installing a previous modification or fix pack over a more recent one.

You can install the product at the same version and release more than once; these installations can be at the same modification or fix pack level, or at different levels.

To achieve concurrent installations, you must select the `custom installation` option and specify a unique directory for each installation (one of which can be the default directory).

You can also use the custom installation to install into a non-default directory.

If you have never completed a typical installation of the product on the computer on which you have selected a custom installation, the directory is initially set to the default directory, but you can change this default value to your chosen value.

If you attempt to install the Broker component into a directory that already contains an installation of the Broker component at a previous version, you are prompted to confirm that you want to continue with the installation, because it overwrites the existing installation. Cancel the installation and select a different directory to preserve your existing configuration.

If you install the same version and release more than once, the native installer support cannot manage these installations in the normal way. For more information, see "How to uninstall multiple installations of the Broker component" on page 3620.

You can use multiple installations at different modification or fix pack levels to test out fixes or new functions, or to stage your adoption of a new fix pack level. For more information, see "Applying service" on page 313.

During and after installation, files are also stored in the working directory, which is associated with the user ID with which you are currently logged on. The location depends on the operating system:

Linux and UNIX
`/var/mqsi`

Windows
`%ALLUSERSPROFILE%\Application Data\IBM\MQSI`

The environment variable `%ALLUSERSPROFILE%` defines the system working directory. The default directory depends on the operating system:

- On Windows XP and Windows Server 2003: `C:\Documents and Settings\All Users\Application Data\IBM\MQSI`
- On **Windows** Vista and later operating systems: `C:\ProgramData\IBM\MQSI`

The actual location might be different on your computer.

If you have multiple installations on a single computer, you can review the contents of the file `install.properties`, which is stored in the root of the working directory. For each installation at Version 6.1 and above, the file is updated with the location and the level.

This example shows the contents of `install.properties` on a Windows 32-bit operating system on which a single installation has completed:

```
C\:\\Program\ Files\\IBM\\MQSI\\7.0=7.0.0.0
```

(The backslash character \ is interpreted as an escape character. It is inserted before each non-alphabetic and non-numeric character in the string to preserve the character. A colon, a space, and several backslash characters are escaped in this example.)

If you want to revert from your latest installation to a previous level for any reason, you must uninstall the current version and install the previous level of the product. Before you uninstall, back up any resources that you want to return to a previous state.

Because the version and release are included in the directory structure when you complete a typical installation, you can also install Version 7.0 and later releases on the computer on which you have already installed either Version 6.0, or Version 6.1. The Version 7.0 installation can coexist with the existing installation; you can operate the two configurations independently.

If you use custom installations for Version 7.0 and later releases, you can specify a unique installation directory for each release, and therefore achieve coexisting releases on a single computer.

The number of installations of Version 7.0, or later, is limited only by the availability of system resources.

Because different versions and releases can coexist, you can migrate to Version 7.0 from an earlier version in a controlled manner, and you do not have to migrate all brokers at the same time. For more information, see "Migration" on page 244.

Broker component on z/OS

On z/OS, you can install multiple copies of the Broker component on the same computer if you specify a different installation location for each copy. The installations can run independently of each other. The code can be at the same or different version and release levels; Version 6.0, Version 6.1, and Version 7.0. The number of installations is restricted only by the availability of system resources.

For more details about locations, libraries, and file system paths, see the *Program Directory for WebSphere Message Broker for z/OS* on the WebSphere Message Broker Library web page.

Coexistence with components of the same version

Although you can install multiple versions of the Broker component on the same system, you should not install both an x86 and an x86-64 version, of the same version number, on the same system. Some data files used by the Broker component are specific to each version, and using files of the wrong version with a broker could cause unpredictable results. This warning applies for all versions of the product and for all platforms.

WebSphere Message Broker Toolkit on Linux on x86, Linux on x86-64, and Windows

Linux When you install the WebSphere Message Broker Toolkit, the default action taken by the installation wizard is to install Installation Manager files, shared files, and product-specific files into the following directories:

- Installation Manager installation directory:
  `/opt/IBM/InstallationManager`
- Shared resources directory:
  `/opt/IBM/SDPShared/`
- Package group directory:
  `/opt/IBM/WMBT700`

This location defines the default value of *install_dir* on this platform.

For a description of these directories, see "Installing the WebSphere Message Broker Toolkit" on page 276.

You can install multiple instances of the WebSphere Message Broker Toolkit Version 7.0 at the same modification or fix pack level, or at different levels, on a single computer. Each installation must be in a separate package group; package groups are described in more detail in "IBM Installation Manager" on page 3600.

Windows

When you install the WebSphere Message Broker Toolkit, the default action taken by the installation wizard is to install Installation Manager files, shared files, and product-specific files into the following directories:

- Installation Manager installation directory:
  `C:\Program Files\IBM\InstallationManager` for the 32-bit version of WebSphere Message Broker
  `C:\Program Files (x86)\IBM\InstallationManager` for the 64-bit version of WebSphere Message Broker
- Shared resources directory:
  `C:\Program Files\IBM\SDPShared\` for the 32-bit version of WebSphere Message Broker
  `C:\Program Files (x86)\IBM\SDPShared\v.r` for the 64-bit version of WebSphere Message Broker
- Package group directory:
  `C:\Program Files\IBM\WMBT700` for the 32-bit version of WebSphere Message Broker
  `C:\Program Files (x86)\IBM\WMBT700` for the 64-bit version of WebSphere Message Broker

  This location defines the default value of *install_dir* on this platform.

For a description of these directories, see "Installing the WebSphere Message Broker Toolkit" on page 276.

You can install multiple instances of the WebSphere Message Broker Toolkit Version 7.0 at the same modification or fix pack level, or at different levels, on a single computer. Each installation must be in a separate package group; package groups are described in more detail in "IBM Installation Manager" on page 3600.

If you install the WebSphere Message Broker Toolkit on Windows and you specify your own directory location, be aware of the file system limit of 256 characters imposed by Windows file systems. This limit can cause restrictions in path specification to resources (for example, message flows), and might cause access problems if the combination of path and resource name exceeds this limit. Keep installation locations and resource names short to avoid problems associated with this restriction.

The WebSphere Message Broker Toolkit Version 7.0 can coexist with the WebSphere Message Broker Toolkit Version 6.0 or Version 6.1. Only one instance of the WebSphere Message Broker Toolkit Version 6.0 can be installed on a single computer.

The WebSphere Message Broker Toolkit Version 7.0 can coexist with multiple installations of the Broker component, subject to the restrictions described for the Broker component.

Setting the environment for an installation

Because you can have more than one installation on a single computer, you must ensure that the commands that you issue on that computer are directed to the correct version of installed code.

- On Linux and UNIX systems, you must run the profile file **mqsiprofile** to set up the correct environment before you run other WebSphere Message Broker commands, such as **mqsicreatebroker**. The profile file is stored in *install_dir*/bin.

  If you add the profile file to your system logon profile, it is run automatically whenever you log on.
- On Windows systems, a command console is available for each installation. So you must run commands in the correct window for a particular installation.

  If you prefer, you can run the mqsiprofile.cmd file, which is stored in *install_dir*\bin.

If you have installed an earlier version of this product on the same computer, check that the earlier profile is not set for the current user ID. The two profiles are incompatible and might cause unpredictable results. Consider using a different user ID for each version and associate the correct profile with each user ID to avoid potential problems.

This requirement is not applicable on z/OS systems.

For more details about **mqsiprofile**, see "Setting up a command environment" on page 213.

WebSphere Message Broker Explorer

Only one copy of WebSphere Message Broker Explorer can be installed at any time on a single system, so you must ensure the latest level of WebSphere Message Broker Explorer is installed. For example, if you have WebSphere Message Broker V7.0.0.0 and WebSphere Message Broker V7.0.0.1 installed on the same system, you must ensure that WebSphere Message Broker Explorer is installed at the later V7.0.0.1 level. This only applies to Windows, Linux on x86, and Linux on x86-64 systems.

Migration

Because you can install WebSphere Message Broker Version 7.0 on the same computer as previous versions and other installations of Version 7.0, you are not required to complete any migration tasks before you install Version 7.0.

Because WebSphere Message Broker Version 7.0 requires a later version of WebSphere MQ, you must update any existing WebSphere Message Broker Version 6.0 and WebSphere Message Broker installations to use a supported version of WebSphere MQ before installing WebSphere Message Broker Version 7.0. For the latest details of all supported levels of hardware and software, visit the WebSphere Message Broker Requirements website.

For details about migrating to WebSphere Message Broker Version 7.0, see Chapter 3, "Migrating and upgrading," on page 137.

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

Chapter 3, "Migrating and upgrading," on page 137
To migrate a broker domain to WebSphere Message Broker Version 7.0, plan your migration strategy, perform pre-migration tasks, migrate your domain components, and then complete post-migration tasks.

Preparing the system

On some operating systems, you must complete several tasks before you install WebSphere Message Broker.

About this task

You might also want to complete other tasks, depending on your installation intentions.

Read the following sections before installing:

Procedure

1. "Setting up security."
2. "Configuring temporary space on distributed systems" on page 251.
3. If you are installing on a distributed system, see "Accessing CDs and DVDs on the local system" on page 253.
4. If you are installing on Linux or UNIX systems, see "Checking the kernel configuration on Linux and UNIX systems" on page 259.

What to do next

When you have completed these tasks, follow the installation instructions for distributed systems in the appropriate chapter:

- On distributed systems, choose which WebSphere Message Broker components to install. For more information, see "Choosing what to install" on page 260.
- On z/OS, refer to the *Program Directory for WebSphere Message Broker for z/OS* on the WebSphere Message Broker Library web page.

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

Related information:

WebSphere Message Broker Requirements

Setting up security

Set up the required security before you install WebSphere Message Broker.

About this task

This section describes security requirements for installing the product and verifying your installation.

After installation, check the topics under the Security heading in the information center to review and implement the security requirements for additional users doing other tasks.

Security control of WebSphere Message Broker components, resources, and tasks depends on the definition of users and groups of users (principals) to the security subsystem of the operating system. Check that you have the correct authority, and that the required principals are in place, before you install WebSphere Message Broker.

User ID restrictions: some operating systems and other products impose restrictions on user IDs:

- On Windows systems, user IDs can be up to 12 characters long, but on Linux, UNIX, and z/OS systems, they are restricted to eight characters. Database products, for example DB2, might also restrict user IDs to eight characters. If you have a mixed environment, ensure that the user IDs that you use within the broker environment are limited to a maximum of eight characters.
- Ensure that the case (upper, lower, or mixed) of user IDs in your broker environment is consistent. In some environments, uppercase and lowercase user IDs are considered the same, but in other environments, user IDs of different case are considered unique. For example, on Windows the user IDs 'tester' and 'TESTER' are identical, but on Linux and UNIX systems they are recognized as different user IDs.
- Check the validity of spaces and special characters in user IDs to ensure that, if used, these characters are accepted by all relevant systems and products in your broker environment.

If your user ID does not conform to these restrictions, you might have problems with installation or verification. If so, use an alternative user ID, or create a new one, to complete installation and verification.

Set up the security appropriate to the operating systems that you are using:

- If you are installing on Linux or UNIX systems, go to “Security on Linux and UNIX systems” on page 247.
- If you are installing on Windows, go to “Security on Windows systems” on page 248.
- If you are installing on z/OS, go to “Security on z/OS systems” on page 251.

Related tasks:

“Installing” on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

“Preparing the system” on page 245
On some operating systems, you must complete several tasks before you install WebSphere Message Broker.

Related information:

WebSphere Message Broker Requirements

Security on Linux and UNIX systems:

Set up the required security on Linux and UNIX systems before you install WebSphere Message Broker.

About this task

Use the security facilities provided by your operating system to complete these tasks; for example, the Systems Management Interface Tool (SMIT) on AIX, or the System Administration Manager on HP-Itanium.

Complete the following actions:

Procedure

1. Log into the system.

   On AIX, you must log in as root. On Linux and on other UNIX computers, your user ID must have root authority to complete installation. Follow your local security guidelines to acquire root authority; either log in as root, or log in as another user and become root.

   The use of a user ID other than root itself has some advantages; it provides an audit trail of the user ID that installs the product and it limits the scope of root authority to tasks performed in a single session. The use of a user ID other than root might also be mandatory if you are logging in from a remote system.

   If you are using a Linux on x86 or a Linux on x86-64 system and are not planning to install the Broker component, continue with step 6.
2. Create a security group called *mqbrkrs*.
3. Add your current logon ID to the group *mqbrkrs*.

   If you are installing on a system that runs as a production server (with the Broker component installed), create an additional new user ID for use only with product components and add it to the *mqbrkrs* group.

   On a Linux on x86 or a Linux on x86-64 system that you are running as a development or test system, you can use the ID that you logged in with to complete installation.
4. If you have already installed WebSphere MQ on this system, a group called *mqm* and a user called *mqm* are defined. If you have not yet installed WebSphere MQ, you must create this group and user.
5. Add to the group *mqm* the user ID that you logged in with, the new user ID (if you created one), and the user ID *mqm*.

   On some systems, you must log off and log on again for these new group definitions (*mqbrkrs* and *mqm*) to be recognized.
6. Verification procedures are provided for Linux on x86 and Linux on x86-64. To complete verification, you do not require root authority. If you do not want to complete verification with root authority, log off when you have completed installation. Log in with the same or a different user ID, but do not become root.

   If you log in with another user ID, and have not already added this ID to the groups *mqbrkrs* and *mqm*, do so before you open the WebSphere Message Broker Toolkit.

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

"Preparing the system" on page 245
On some operating systems, you must complete several tasks before you install WebSphere Message Broker.

"Setting up security" on page 245
Set up the required security before you install WebSphere Message Broker.

Related information:

WebSphere Message Broker Requirements

Security on Windows systems:

Set up the required security on Windows systems before you install WebSphere Message Broker.

About this task

Before you install the Broker component, the WebSphere Message Broker Toolkit, or the WebSphere Message Broker Explorer, log on with a user ID that has Administrator authority.

If you are installing the Broker component, the installation wizard calls the **mqsisetsecurity** command which completes the following tasks:

- Creates a new security group called *mqbrkrs*.
- Adds your current (logged on) user ID to the group *mqbrkrs*.
- Adds your current user ID to the group *mqm*, if that group exists.

  The *mqm* group exists if you have already installed WebSphere MQ on this system. If you have not, call the **mqsisetsecurity** command when you have completed WebSphere MQ installation. If you use the Windows Launchpad (described in "Installing by using the Windows Launchpad" on page 262), it completes WebSphere MQ installation first.

If you prefer to create principals before you install WebSphere Message Broker, use the security facilities provided by the Windows Control Panel.

If you are running Terminal Services on this computer, change user mode to ensure that actions taken during installation are completed correctly; for example, the creation of `.ini` files and other related files in the default system directory `C:\Windows`. If you do not change user mode, files might be written to other locations and, although the installation might complete successfully, the product might not work as expected.

- Before you install any product components, enter the following command to change user mode:

  ```
  change user /install
  ```

- When installation is complete, enter the following command to restore the original user mode:

  ```
  change user /execute
  ```

To complete verification, your user ID must have Administrator authority. If you log in with a different user ID from the ID with which you perform installation, you must add that user ID to the groups *mqbrkrs* and *mqm*. Use either the Windows security facilities or the **mqsisetsecurity** command (run this command after you have logged on with that different ID) to complete these additions.

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

"Preparing the system" on page 245
On some operating systems, you must complete several tasks before you install WebSphere Message Broker.

"Setting up security" on page 245
Set up the required security before you install WebSphere Message Broker.

"Security in a Windows domain environment"
Set up the required security configuration in a Windows domain environment.

Related information:

WebSphere Message Broker Requirements

Security in a Windows domain environment:

Set up the required security configuration in a Windows domain environment.

About this task

You can use Windows domain groups to organize different levels of authorization to selective WebSphere Message Broker resources across your domain. To design and implement this domain group topology, add each domain group to the relevant local security groups on the domain workstations. You can now manage authorities by adding domain user accounts to the appropriate domain groups. For information about the group membership required to administer WebSphere Message Broker resources, see "Security requirements for Windows systems" on page 3651.

Procedure

1. Design your authorization group categories, and define domain groups on the domain controller system that correspond to these authorization categories, by using Windows security. For example, suppose you have a single domain containing three distinct sets of systems, used in development, testing, and production. Within your organization, various user roles require different levels of authorization to WebSphere MQ and WebSphere Message Broker resources on those systems.

   Here is an example of how those authorization categories could map to domain groups:

Domain group	Description
ADM-MQprd	WebSphere MQ administrator authorities on production machines
ADM-MQuat	WebSphere MQ administrator authorities on test machines

Domain group	Description
ADM-MQdev	WebSphere MQ administrator authorities on development machines
ADM-MBprd	WebSphere Message Broker administrator authorities on production machines
ADM-MBuat	WebSphere Message Broker administrator authorities on test machines
ADM-MBdev	WebSphere Message Broker administrator authorities on development machines

2. Define and configure domain user accounts on the domain controller, by using Windows security. Add each user account to one or more domain groups to determine the authorizations granted that account. For example:

Table 4.

Domain user account	Role	Group membership
MQadmPRD	WebSphere MQ administrator for production systems	ADM-MQprd
MQadmUAT	WebSphere MQ administrator for test systems	ADM-MQuat
MQadmDEV	WebSphere MQ administrator for development systems	ADM-MQdev
MBadmPRD	WebSphere Message Broker administrator for production systems	ADM-MBprd
MBadmUAT	WebSphere Message Broker administrator for test systems	ADM-MBuat
MBadmDEV	WebSphere Message Broker administrator for development systems	ADM-MBdev
john.smith	WebSphere MQ and WebSphere Message Broker administrator for production environments	ADM-MQprd, ADM-MBprd

3. Install and configure WebSphere Message Broker on domain workstations.
   a. Install WebSphere Message Broker on the workstation.
   b. Add your domain groups to local groups *mqm* or *mqbrkrs* as appropriate. In our example, if a particular workstation is to serve as a development machine, add domain group *ADM-MQdev* to local group *mqm*, and domain group *ADM-MBdev* to local group *mqbrkrs*.

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

"Preparing the system" on page 245
On some operating systems, you must complete several tasks before you install WebSphere Message Broker.

"Setting up security" on page 245
Set up the required security before you install WebSphere Message Broker.

"Security on Windows systems" on page 248
Set up the required security on Windows systems before you install WebSphere Message Broker.

Related information:


WebSphere Message Broker Requirements

Security on z/OS systems:

User ID security required for z/OS system product installation.

About this task

The user ID that you use to install the product must be no more than eight characters in length. It must also have suitable RACF privileges to perform SMP/E installation in your environment. The user ID must have a valid OMVS segment, because the product installs into the file system paths specified during the SMP/E APPLY processing.

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

"Preparing the system" on page 245
On some operating systems, you must complete several tasks before you install WebSphere Message Broker.

"Setting up security" on page 245
Set up the required security before you install WebSphere Message Broker.

Related information:


WebSphere Message Broker Requirements

Configuring temporary space on distributed systems

Configure temporary directory space to enable WebSphere Message Broker Java components to operate correctly.

About this task

WebSphere Message Broker contains Java components. Some of these components require temporary directory space on the file system in order to extract class files from node packages (PAR files) and Java packages (JAR files).

The extracted files might not be visible to interactive users on the system, however, they consume space in the temporary directory.

The Java Runtime Environment also creates files in the temporary directory to support debug facilities.

To enable correct operation, allow at least 50 MB of space per execution group in the file system temporary directory for WebSphere Message Broker components. This space is required in addition to any space required for operation of user developed artifacts. Use of JavaCompute nodes, Java user-defined nodes, or additional plug-in nodes implemented in Java might require further temporary directory space.

On Linux, UNIX, and z/OS computers, the TMPDIR directory is typically `/tmp`; on Windows computers, it is `c:\temp`. If this directory is not large enough to hold the JAR files, the broker does not start.

To change the location of the temporary directory, use one of the following methods:

Procedure

- Use the environment variable `TMPDIR`.
- Set the system property `java.io.tmpdir`.

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

"Preparing the system" on page 245
On some operating systems, you must complete several tasks before you install WebSphere Message Broker.

Related information:

WebSphere Message Broker Requirements

Accessing CDs and DVDs

Accessing CDs and DVDs to install WebSphere Message Broker.

About this task

When you install or upgrade WebSphere Message Broker, you can access CDs or DVDs on the local system, or you can set up a shared drive and access the shared resource from multiple computers.

The information in the following sections is relevant to both CDs and DVDs. All references are to CDs; DVD behavior is identical. DVDs are available for Linux on x86, Linux on x86-64, Windows 32-bit, and Windows 64-bit only.

You can also install or upgrade from installation images that you have obtained from Passport Advantage, if you are registered with this scheme:

Procedure

1. Read the instructions that are provided with the packages.
2. Download the images that you require for the operating systems in your environment.
3. Extract the contents of the images. Specify a short path for the directory to which you are extracting; the depth of directory structure and the directory

names might cause problem if restrictions on some operating systems are reached. For example, the limit of 256 characters on Windows might be exceeded.

4. Set up local or remote access to these images in the same way that you do for a CD or DVD. For local installations, see "Accessing CDs and DVDs on the local system"; for remote installations, see "Accessing CDs and DVDs on a remote system" on page 256.

What to do next

If you are installing on Windows, you cannot enter a Universal Naming Convention (UNC) path (`\\server\drive`) to access the installation program; you must map the drive, otherwise the Java process times out. If you cannot map the drive, or choose not to map the drive, copy the contents of the DVD to a local drive and install from that drive.

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

"Preparing the system" on page 245
On some operating systems, you must complete several tasks before you install WebSphere Message Broker.

Related information:

WebSphere Message Broker Requirements

Accessing CDs and DVDs on the local system:

If you want to install product components from a local CD or DVD, or a local downloaded image, complete this task.

About this task

Always consult your operating system documentation for exact details.

AIX

1. Log in as root. You cannot complete installation successfully if you have logged in as another ID and become root.
2. Complete the security setup described in "Security on Linux and UNIX systems" on page 247.
3. Create a CD mount point directory:
   ```
   mkdir /cdbroker
   ```
   where */cdbroker* is the mount point.
4. Insert the CD into the drive of the computer on to which you want to install product components.
5. Use SMIT to mount the CD, or use the following command:
   ```
   mount -r -v cdrfs /dev/cd0 /cdbroker
   ```
   where */dev/cd0* is the CD device and */cdbroker* is the mount point.

You are now ready to install the product that is supplied on this CD.

HP-Itanium

The HP-Itanium CDs have the format ISO 9660, with Rockridge extensions enabled. If volume management software is in use, the CD mounts automatically when you insert it into the CD drive. Alternatively, you can mount the CD as described in the following procedure.

If the CD is mounted incorrectly, some of the files cannot be read and the installation fails with a corrupted directory. You must mount the CD with Rockridge extensions enabled.

1. Log in and ensure that your user ID has root authority.
2. Complete the security setup described in "Security on Linux and UNIX systems" on page 247.
3. Create a CD mount point directory and grant read-only access to all users:

   ```
   mkdir /cdbroker
   chmod 775 /cdbroker
   ```

   where */cdbroker* is the mount point.
4. Insert the CD into the drive of the computer on to which you want to install product components.
5. Mount the CD by using the following command:

   ```
   mount -F cdfs /dev/dsk/device /cdbroker
   ```

   where */device* is the CD device, for example `/c0t0d0` and */cdbroker* is the mount point.

You are now ready to install the product that is supplied on this CD.

Linux on x86 and Linux on x86-64

1. Log in and ensure that your user ID has root authority.
2. Complete the security setup described in "Security on Linux and UNIX systems" on page 247.
3. Create a DVD mount point directory:

   ```
   mkdir /dvdbroker
   ```

   where */dvdbroker* is the mount point.
4. Insert the DVD into the drive of the computer on to which you want to install product components.
5. Run the following command:

   ```
   mount -o ro -t iso9660 /dev/dvdrom /dvdbroker
   ```

   where */dev/dvdrom* is the name of your DVD device (for example, `/dev/hdc`) and */dvdbroker* is the mount point.

You are now ready to install the product that is supplied on this DVD.

Linux, other than Linux on x86 and Linux on x86-64

1. Log in and ensure that your user ID has root authority.
2. Complete the security setup described in "Security on Linux and UNIX systems" on page 247.
3. Create a CD mount point directory:

   ```
   mkdir /cdbroker
   ```

   where */cdbroker* is the mount point.

4. Insert the CD into the drive of the computer on to which you want to install product components.
5. Run the following command:

   ```
   mount -o ro -t iso9660 /dev/cdrom /cdbroker
   ```

   where */dev/cdrom* is the name of your CD device (for example, `/dev/hdc`) and */cdbroker* is the mount point.

You are now ready to install the product that is supplied on this CD.

Solaris

1. Log in and ensure that your user ID has root authority.
2. Complete the security setup described in "Security on Linux and UNIX systems" on page 247.
3. Insert the CD into the drive of the computer on to which you want to install product components.
4. Enter the following command to check whether the Volume Manager is running on your system:

   ```
   /usr/bin/ps -ef | /bin/grep vold
   ```

   If the Volume Manager is running, the CD is mounted on `/cdrom/vol_label` automatically, where *vol_label* is the volume label of the current CD; for example, `wmb6_sol` for Runtime Disk 1.
5. If the Volume Manager is not started, run the following commands to mount the CD:

   ```
   mkdir -p /cdbroker
   mount -F hsfs -o ro /dev/dsk/cdrom /cdbroker
   ```

   where */dev/dsk/cdrom* is the CD location (for example, `c0t0d0`) and */cdbroker* is the mount point directory.

   Check where your CD is located by using the command `iostat -En`. Alternatively, use the **`volcheck`** command to mount a CD device automatically.

You are now ready to install the product that is supplied on this CD.

Windows

1. Log on with a user ID that has Administrator authority.
2. Complete the security setup described in "Security on Windows systems" on page 248.
3. Insert the DVD into the drive of the computer on to which you want to install product components. The Launchpad opens.

You are now ready to install the product that is supplied on this DVD.

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

"Preparing the system" on page 245
On some operating systems, you must complete several tasks before you install WebSphere Message Broker.

"Accessing CDs and DVDs" on page 252
Accessing CDs and DVDs to install WebSphere Message Broker.

Related information:

WebSphere Message Broker Requirements

Accessing CDs and DVDs on a remote system:

If you want to install product components from a remote (server) CD or DVD, complete this task.

About this task

Always consult your operating system documentation for exact details of this task.

If you want to perform more than one installation of one or more components, you might find that a remote server setup provides some performance benefits, particularly for the WebSphere Message Broker Toolkit which is the largest component. You might also find this method more convenient if you want to run installations by using the silent interface.

If you want to install the WebSphere Message Broker Toolkit by using the silent interface, and you cannot install from DVD, you must copy the installation images onto a disk drive, as described here, to avoid the requirement to swap CDs during the process.

To enable a remote installation, you must complete tasks on both the server (the computer on which the CD, DVD, or shared drive is mounted) and each target system (on which you want to install the product). For details of the commands used in these examples, refer to the operating system documentation.

To set up a server, see "Setting up the server."

To set up a target system, see "Setting up the target system" on page 258.

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

"Preparing the system" on page 245
On some operating systems, you must complete several tasks before you install WebSphere Message Broker.

"Accessing CDs and DVDs" on page 252
Accessing CDs and DVDs to install WebSphere Message Broker.

Related information:

WebSphere Message Broker Requirements

Setting up the server:

You can either share the CD drive on the server, or copy the installation images onto a disk and share the directory on that disk.

About this task

You can share a CD drive on Linux or UNIX with any other supported Linux or UNIX system, but not with Windows. You can share a Windows CD drive only with other Windows systems.

1. If you want to share a copy of the installation image, create the copy:
   a. Create a directory on the server to store the installation images:

      Linux and UNIX

      Enter the following command:

      ```
      mkdir /instbroker
      ```

      where *instbroker* is the directory into which you copy the product files.

      Windows

      Enter the following command:

      ```
      md m:\instbroker
      ```

      where *m* is the drive on which you want to store the installation images and *instbroker* is the directory on that drive.

      If you are creating directories for the WebSphere Message Broker Toolkit on Linux on x86, Linux on x86-64 or Windows, you must create all three disk subdirectories in the same directory, for example:

      ```
      /instbroker/disk1
      /instbroker/disk2
      /instbroker/disk3
      ```

      Where each subdirectory, for example `disk1`, is the root level of the corresponding CD image.

      This structure ensures that the installation program does not prompt for location, and does not fail because it cannot find the right images.

   b. Insert and mount the appropriate CD in the drive as described in "Accessing CDs and DVDs on the local system" on page 253. The installation programs for the runtime components and the WebSphere Message Broker Toolkit are on separate CDs; insert the correct CD for the components that you want to install from this server.

      If you have inserted a runtime or toolkit CD on Windows and autorun is enabled, the Launchpad is started. When the initial window opens, click **Cancel** to close it.

   c. Copy the complete contents of the CD to the new directory.

      Linux and UNIX

      Enter the following command:

      ```
      cp -rf /cdrom/. /instbroker
      ```

      Windows

      Enter the following command:

      ```
      xcopy f:\*.* m:\instbroker /e
      ```

      where *f* is the CD drive.

2. Grant users access to the drive that contains the product code. These instructions are the same for a disk drive on which you have copied the CD contents, and for the CD drive itself.

   AIX Either type **smit** and click **Communications Applications and Services** > **NFS** > **Network File System (NFS)** > **Add a Directory to Exports**

List, or enter the fast path command **smitty mknfsexp**. Complete the fields as appropriate and press Enter.

HP-UX and Linux

Use the **exportfs** command. The following example gives all users read-only access using NFS:

```
exportfs -i -o ro /instbroker
exportfs -a
```

where */instbroker* represents the CD drive or the directory that contains the CD copy.

Solaris

Use the **share** and **exportfs** commands. The following example gives all users read-only access using NFS:

```
share -F nfs -o ro -d "Broker LAN server" /instbroker
exportfs -a
```

where "*Broker LAN server*" is an optional description and */instbroker* represents the server CD drive or directory containing the CD copy.

Windows

Open Windows Explorer and right-click the drive that you want to share. Click **Sharing** and follow the instructions on the Properties dialog box.

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

"Preparing the system" on page 245
On some operating systems, you must complete several tasks before you install WebSphere Message Broker.

"Accessing CDs and DVDs" on page 252
Accessing CDs and DVDs to install WebSphere Message Broker.

Related information:

WebSphere Message Broker Requirements

Setting up the target system:

Set up a target system to access CDs and DVDs from a server.

About this task

1. On Linux and UNIX systems, create a new directory on which to mount the shared directory. Enter the following command:

   ```
   mkdir /remotebroker
   ```

   where *remotebroker* is the name of the new directory.
2. Access the remote directory:

   Linux and UNIX

   Enter the following command:

   ```
   mount server name:instbroker /remotebroker
   ```

where *server name* is the name of the server on which you created the CD or DVD copy.

Windows

Connect to the appropriate drive and folder by using the **net use** command at a command prompt on the target system, for example:

```
net use x: \\server_name\instbroker
```

where *x*: is the required mapped drive on the target system.

If your shared installation directory name contains spaces (for example, Broker Image), enclose it in double quotation marks.

If your server is protected, you might need to specify a user ID and password on this command (see the Windows online help for more information about the **net use** command). Alternatively, use Windows Explorer or an alternative method to map the shared resource to a drive letter.

You cannot enter a UNC path (`\\server\drive`) to access the installation program; you must map the drive, as shown, otherwise the Java process times out. If you cannot map the drive, or choose not to map the drive, copy the contents of the DVD onto a local drive and install from that drive. In addition, you cannot enter a UNC path when the installation wizard requests a path as input; the wizard cannot interpret a UNC path.

3. Change to the remote image directory. You are now ready to run the Launchpad (on Windows only) or the installation wizard to install the product from the remote directory to your local system.

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

"Preparing the system" on page 245
On some operating systems, you must complete several tasks before you install WebSphere Message Broker.

"Accessing CDs and DVDs" on page 252
Accessing CDs and DVDs to install WebSphere Message Broker.

Related information:

WebSphere Message Broker Requirements

Checking the kernel configuration on Linux and UNIX systems

Check the kernel configuration parameters on Linux and UNIX systems for prerequisite and corequisite products.

About this task

WebSphere Message Broker has no specific requirements for kernel configuration parameters; however, other products might require particular settings. If you do not tune your kernel parameters to suit the products that you have installed, you might see unexpected results or a deterioration in performance.

Follow these steps to configure your kernel parameters:

Procedure

1. Check the documented values for the following products:
   - WebSphere MQ
   - DB2 (if it is installed)
   - Other software that you have installed to work with WebSphere Message Broker, including other databases.

   You can access the relevant information for IBM products through the IBM product information centers web page.
2. Take the highest value for each parameter and compare it to the corresponding value in your kernel configuration.
3. If the current value is lower than the highest documented value, update the current setting by using the appropriate tooling that is supplied by the operating system provider. If the current value is higher, leave it unchanged.
4. On Solaris, increase the maximum number of concurrent open file descriptors on your system to at least 256.
5. If you have changed any kernel values, you might need to restart your system for these changes to take effect. Check the documentation for your operating system for further information about these parameters.

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

"Preparing the system" on page 245
On some operating systems, you must complete several tasks before you install WebSphere Message Broker.

Related information:

WebSphere Message Broker Requirements

Choosing what to install

Choose which components and products to install.

Before you begin

On some operating systems, you must complete several tasks before you install WebSphere Message Broker. For more information, see "Preparing the system" on page 245.

About this task

WebSphere Message Broker consists of three components; the Broker component, the WebSphere Message Broker Explorer, and the WebSphere Message Broker Toolkit. You can also install other products, such as the WebSphere Message Broker ODBC Database Extender:

- The Broker component
- The WebSphere Message Broker Explorer
- The WebSphere Message Broker Toolkit

- The WebSphere Message Broker ODBC Database Extender

You can install the Broker component on all supported operating systems. You can install the WebSphere Message Broker Explorer and the WebSphere Message Broker Toolkit on Windows 32-bit, Windows 64-bit, Linux on x86, and Linux on x86-64.

The Broker component

The Broker component is a set of execution processes that provides message processing facilities that interact with various application clients that use both point-to-point and publish/subscribe communications. The message flows that you create are hosted by the broker. A broker can host many message flows, in one or more execution groups, and can support many clients.

You define how messages are received, processed, and delivered to receiving applications or subscribers:

- You can customize some message processing nodes in a message flow with mappings, ESQL, Java, PHP, and XSL style sheets.
- You can create message models to define message structures determined by C and COBOL data structures, industry standards such as SWIFT or EDIFACT, and XML DTD or schema.
- You can develop user-defined extensions (nodes and parsers) to support message processing options that are not provided by the supplied nodes and parsers.
- You can debug message flows and step through processing to check paths and results.
- You can use message flow aggregation to manage multiple requests and responses that are generated by a single input message.

You can install more than one WebSphere Message Broker component on any system. For more details about how different installations can coexist, see “Coexistence and migration” on page 239.

For installation of the Broker component, you can choose between a typical installation and a custom installation. These installation options are explained in “Coexistence and migration” on page 239.

WebSphere Message Broker Explorer

The WebSphere Message Broker Explorer is a stand-alone administration environment that is based on the Eclipse platform which communicates with one or more brokers. Administrators use the WebSphere Message Broker Explorer to manage the resources associated with these brokers. Install the WebSphere Message Broker Explorer on computers on which you intend to perform only administrative tasks.

WebSphere Message Broker Toolkit

The WebSphere Message Broker Toolkit is an integrated development environment and graphical user interface that is based on the Eclipse platform and the Rational framework.

Application developers work in separate instances of the WebSphere Message Broker Toolkit to develop message flows, message sets, and user-defined nodes and parsers. You can access a shared repository (for example, CVS) to store resources and make them accessible in a secure manner to multiple users.

WebSphere Message Broker ODBC Database Extender

The WebSphere Message Broker ODBC Database Extender is required when using WebSphere Message Broker to interface with an ODBC data source that is not supported through the DataDirect ODBC drivers.

What to do next

Choose the interface you want to use to install WebSphere Message Broker. For more information, see "IBM Installation Manager" on page 3600

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

Related information:

WebSphere Message Broker Requirements

Installing by using the Windows Launchpad

Use the Windows Launchpad to install the WebSphere Message Broker components and the prerequisite products.

Before you begin

Choose which WebSphere Message Broker components to install. For more information, see "Choosing what to install" on page 260.

About this task

On Windows only, use the Launchpad for additional help with installing:

- The WebSphere Message Broker Toolkit.
- The WebSphere Message Broker Explorer.
- The Broker component.
- Prerequisite products for the Broker component.

If you use the Launchpad, you can install everything that you need, and do not have to follow the procedures described in other chapters for installing the Broker component, the WebSphere Message Broker Toolkit, and the WebSphere Message Broker Explorer.

The Launchpad works with both physical media (the DVD) and with electronic images that you have downloaded from IBM Passport Advantage; however, the Launchpad depends on a file structure identical to that on the DVD, therefore you must not make any changes during or after download.

Multiple installations on a single computer

About this task

You can use the Launchpad to install only one instance of each component on a single computer. If you have selected and installed the broker or toolkit components, you cannot use the Launchpad to install these components again in a different location.

To install additional instances, run the appropriate installation wizard directly. For more information about completing these tasks, see “Installing the Broker component” on page 267, “Installing the WebSphere Message Broker Toolkit” on page 276, and “Installing WebSphere Message Broker Explorer” on page 280.

The Launchpad also manages only one installation of WebSphere MQ on a single computer. Refer to the relevant documentation for these products if you want to install multiple instances.

Installation summary

About this task

The following steps summarize the actions that you must take to complete the installation.

Procedure

1. Check the `readme.html` file for any updates to these installation instructions.
2. Check that you have enough memory and disk space; refer to “Memory and disk space requirements” on page 3584.
3. Decide whether you want to install from a server, or install locally on each system. These choices are described in “Accessing CDs and DVDs” on page 252.
4. Start the Windows Launchpad to install WebSphere Message Broker and its prerequisite product; WebSphere MQ. For more information about using the Launchpad, see “Starting the Windows Launchpad.”

Related tasks:

“Installing” on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

“Resolving problems when installing” on page 3517
Work through the advice provided to help you to deal with problems that can arise when the product is installed.

“Finding the latest information” on page 232
Access the latest information for WebSphere Message Broker.

Related information:

WebSphere Message Broker Requirements

Starting the Windows Launchpad

Use the instructions to start the Windows Launchpad.

About this task

The Launchpad is available on every DVD and downloaded image from which product components or prerequisite products can be installed.

If you are using physical product media, the Launchpad starts automatically if autorun is enabled. If autorun is not enabled, or if you are installing from a downloaded image, navigate to root directory of the DVD or image, and double-click the file `mqsilaunchpad.exe` or type `mqsilaunchpad` in a command window and press **Enter**.

The **Installation** window displays. On the **Installation** window, you can install the set of products that are required for a default configuration of WebSphere Message Broker, including WebSphere MQ, if it is not already installed. For more information, see "Installing a default configuration by using the Installation window" on page 265.

Access further information from the left pane:

- Click **Installation Guide** to launch a PDF copy of the Installation Guide in Adobe Reader.
- Click **Readme** to view the readme file `readme.html` in a new web browser window.
- Click **Quick Tour** to take a tour around the product. See Chapter 2, "WebSphere Message Broker overview," on page 5 for further information.

The Launchpad might have to search for an installation wizard for some of your selections. If you are installing from DVD, all the required products are available, but if you are installing from a downloaded image, the program might be on another downloaded image, or might not be in the expected location. If necessary, the Launchpad prompts you to take the appropriate action to find the file. The following table shows, for each supplied product, the program names and their locations on the downloaded images.

Table 5. Installation wizard names and locations used by the Windows Launchpad

Product	Installation wizard name	Directory	Downloaded image
WebSphere Eclipse Platform V3.3[1]	`IBM WebSphere Eclipse Platform V3.3.msi`	`\WebSphere_MQ_V7.0.1\Prereqs\IES\MSI`	Runtime Disk 2
WebSphere MQ V7.0.1[1]	`IBM WebSphere MQ.msi`	`\WebSphere_MQ_V7.0.1\MSI`	Runtime Disk 2
Broker component	`setup.exe`	`\` (root directory)	Runtime Disk 1
WebSphere Message Broker Explorer	`install.exe`	`\MBExplorer`	Runtime Disk 1
WebSphere Message Broker Toolkit	`install.exe`	`\IBMInstallationManager`[2]	Toolkit Disk 1

Notes:

1. WebSphere Message Broker Trial Edition does not include this product. See "Installation packages" on page 3608 for further information.
2. The Launchpad starts Installation Manager, which installs itself (if required), and starts the WebSphere Message Broker Toolkit installation.

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

"Finding the latest information" on page 232
Access the latest information for WebSphere Message Broker.

"Installing by using the Windows Launchpad" on page 262
Use the Windows Launchpad to install the WebSphere Message Broker components and the prerequisite products.

Related information:

WebSphere Message Broker Requirements

Installing a default configuration by using the Installation window:

Use the Windows Launchpad Installation window to install the set of products that are required for a default configuration of WebSphere Message Broker.

Before you begin

Read "Installing by using the Windows Launchpad" on page 262 for background information.

For the latest details of all supported levels of hardware and software, visit the WebSphere Message Broker Requirements website.

About this task

1. Start the Windows Launchpad. See "Starting the Windows Launchpad" on page 263 for more information.

   When the Launchpad starts, the Installation window opens and displays the following minimum set of products, along with version numbers, that are required for a default configuration of WebSphere Message Broker:

   - WebSphere MQ
   - WebSphere Message Broker
   - WebSphere Message Broker Toolkit
   - WebSphere Message Broker Explorer
2. Check the initial installation status that is shown for each listed product:
   - **Required** indicates that the product is not installed but is required for a minimum configuration, or you have cleared the associated check box to indicate that you do not intend to install the product, or the Launchpad cannot install it (and no check box is shown).
   - **Pending** indicates that the product is not installed but is required for a minimum configuration, or the check box is selected to show that this product is to be installed.
   - **Installed** indicates that the product is already installed at a level that is supported by WebSphere Message Broker. The installed version is shown and no check box is displayed.
   - **Partial Installation** indicates that the product is installed, but not all components required to ensure that the successful operation of a minimum configuration broker domain are present on the system. The associated check box is selected to show that additional components are installed.
3. Click the plus sign to the left of each listed product in turn. The Launchpad displays more information about the product, which you can use to decide if you want it installed. The additional information also provides an estimate of the time taken to complete each product installation, if appropriate.
4. Check the Installation has installed all the WebSphere MQ components that you require. WebSphere Message Broker requires only the server, WebSphere MQ Explorer, and the Java Messaging component, and only these components are installed.

   If you want additional components, use the WebSphere MQ installer.

5. WebSphere MQ Explorer requires the WebSphere Eclipse Platform to be installed; when you select WebSphere MQ, the Eclipse Platform is automatically installed.
6. If you do not want to install a listed product, clear the check box that is associated with the product. Its status is changed to **Required**, because you cannot configure and verify your installation without all the listed products. If the product you have cleared is a prerequisite for another product, that other product is also cleared. You can complete installation of any remaining products, and install the cleared products at a later time.

   If you are installing WebSphere Message Broker Trial Edition, you must install WebSphere MQ and the Eclipse platform before using the Launchpad. These products are not supplied with the Trial Edition. You must obtain prerequisite software from other sources to support the WebSphere Message Broker Trial Edition. See "Installation packages" on page 3608 for more information.
7. Click **Launch Installation for WebSphere Message Broker**.

   If you have cleared one or more of the required products, you are asked to confirm your choices.

   The Launchpad installs the products that you have selected in the order shown. You cannot change the Installation window after you have started the installation process. When the Launchpad starts each installation, it updates status from **Pending** to **In Progress**.

   If you have the incorrect WebSphere MQ level, for example V6.1, you might get the following warning:

   ```
   One or more of the selected installs will upgrade an existing version
   of the installed software. Please be aware that the old version may be
   removed with no further prompting.
   Click OK to continue.
   ```

   The Launchpad uninstalls the incorrect WebSphere MQ level, and then installs the correct level of WebSphere MQ.

   - If you have selected WebSphere MQ, the Launchpad starts the installation wizard silent interface for WebSphere Eclipse Platform V3.3; default values are used for all options. A progress bar is displayed. When the installation completes, the Launchpad starts the installation wizard silent interface for WebSphere MQ; default values are used for all options. A progress bar is displayed so that you can check on progress.
   - If you have selected WebSphere Message Broker (the Broker component), the Launchpad starts the installation wizard graphical interface. You must supply the input that is required by the installation wizard.

     The installation wizard guides you through a series of windows where you can make choices about where to install the component.

     You must also read and accept the Software License Agreement that is displayed.

     The license agreement covers your use of WebSphere MQ. Be aware it is only licensed for use with WebSphere Message Broker and must not be used for other purposes.

     You are asked if you want to open a command console when the wizard terminates; select **Yes** to open a console window that is initialized with the correct environment for command invocation. The command console is explained in "Command environment: Windows systems" on page 306. If you do not want to enter any commands at this time, select **No**.
   - If you have selected the WebSphere Message Broker Toolkit, the Launchpad starts the installation wizard graphical interface.

The process is controlled by Installation Manager, which installs itself if it is not already installed on this computer. For further information about Installation Manager, see "Additional software requirements" on page 3598.

The installation wizard guides you through a series of windows where you can make choices about where to install the component, which package group to install it in, and which language support you want to install. You must supply all input that is required by the installation wizard. You must also accept the Software License Agreement that is displayed.

If you want to launch the WebSphere Message Broker Toolkit when its installation wizard is complete, select **WebSphere Message Broker Toolkit** in the list of installed products that is displayed on the completion window. When you click **Finish**, the wizard ends and returns control to the Launchpad, and the WebSphere Message Broker Toolkit is started.

- If you have selected WebSphere Message Broker Explorer, the Launchpad starts the installation wizard graphical interface. You must supply the input that is required by the installation wizard.

  The installation wizard guides you through a series of windows where you can make choices about where to install the component.

  You must also read and accept the Software License Agreement that is displayed.

The status of each product changes to **Installed** when the Launchpad completes its installation.

When the Launchpad has installed all your selected products, it returns control to the Installation window.

8. Click **Refresh** to check the status of each product listed.
9. Click **Exit Launchpad** to end the program.

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

"Finding the latest information" on page 232
Access the latest information for WebSphere Message Broker.

"Installing by using the Windows Launchpad" on page 262
Use the Windows Launchpad to install the WebSphere Message Broker components and the prerequisite products.

Related information:

WebSphere Message Broker Requirements

Installing the Broker component

Use the installation wizard to install the Broker component.

Before you begin

Read the topic on "Preparing for installation" on page 235 and perform the tasks described in "Preparing the system" on page 245.

About this task

Complete the following tasks to install the Broker component on all supported operating systems.

If you are installing the Broker component on Windows, you can use the LaunchPad to complete this task. For more information, see "Installing by using the Windows Launchpad" on page 262. If you do not want to use the LaunchPad, complete the tasks in this topic instead.

The following list identifies the choices that you have for installing the Broker component, and the actions that you must take to complete your chosen task:

Procedure

1. Check the `readme.html` for any updates to these installation instructions. The readme file location is shown in "System requirements" on page 3581.
2. Check that you have enough memory and disk space; see "Memory and disk space requirements" on page 3584.
3. If you do not already have WebSphere MQ installed, install it before you install the Broker component.

   Although you can install WebSphere MQ after you have installed the Broker component, the installation wizard checks that you have the supported level of WebSphere MQ, or later, installed. If this check fails when you are using the graphical or console interface, the installation wizard displays a warning that lists potential problems. If you decide to continue, you must complete the installation of WebSphere MQ before you create or start any brokers.

   If you start the installer by using the silent interface and WebSphere MQ is not installed, or WebSphere MQ is installed but it is not at the prerequisite level, the check for WebSphere MQ fails. If you have not modified the default behavior by specifying a tailored response file, the wizard terminates without taking any further action. If you have modified the response file to ignore this check, the installation wizard continues.
4. Decide if you want to install from a remote server, or to install locally on each system. These choices are described in "Accessing CDs and DVDs" on page 252 for CDs, DVDs, and for installation images that you can download from Passport Advantage. The instructions here do not differentiate between CDs, DVDs, and downloaded images because the behavior is the same.
5. Decide whether to use the graphical installation, a console installation, or a silent installation. For more information about these interfaces, see "How to install and uninstall the Broker component" on page 3618.
   - To use the graphical installation, continue with these instructions.
   - To use a console installation, see "Installing the Broker component in console mode" on page 270.
   - To use a silent installation, see "Installing the Broker component in silent mode" on page 272.
6. Determine the installation wizard name for your operating system. For more information, see "Installation wizard names" on page 3626.
7. Start the installation wizard graphical interface locally or remotely.
   a. To install locally, load the product CD or DVD, then complete the following steps:

Linux and UNIX

Open a command prompt and navigate to the root directory of the CD or DVD. Type the installation wizard name with no options, and press **Enter**.

Windows

Take one of the following actions:

- If autorun is enabled, the Launchpad is started immediately. To use the Launchpad, see "Installing by using the Windows Launchpad" on page 262. To cancel the Launchpad, click **Exit Launchpad**.
- In Windows Explorer, navigate to the root directory of the CD or DVD. Locate the installation wizard and double-click the wizard to start it.
- Open a command prompt and navigate to the root directory of the CD or DVD. Type the installation wizard name with no options, and press **Enter**.

b. To install from a remote server, access the remote CD or DVD drive or network drive on which the product media is available. Find the installation wizard on the CD, DVD, or network drive and start it as previously described.

8. When the wizard starts, navigate through the windows and provide input when requested. You must also read and accept the Software License Agreement.

   If the directory that you specify for installation already contains a previous version of WebSphere Message Broker, for example Version 6.1.0.9, the installation wizard prevents you from installing Version 7.0 in this location. You must specify a different location. You can then migrate components to Version 7.0 from the previous version, when appropriate.

   Linux and UNIX only

   At the end of this installer, a panel asks if you would like to start the installation of the WebSphere Message Broker ODBC Database Extender (IE02) SupportPac. If selected, the IE02 SupportPac installer automatically launches in GUI mode. For more information, see the WebSphere Message Broker ODBC Database Extender (IE02) SupportPac installation instructions, which are located within the IE02 directory contained on the WebSphere Message Broker installation image.

9. When the summary window displays, check your choices and click **Next** to complete the installation. A progress bar displays so that you can check on progress.
10. If you want to install the WebSphere Message Broker ODBC Database Extender (IE02) package, select **Yes** when the window appears at the end of the installation process. See "Installing the WebSphere Message Broker ODBC Database Extender (IE02)" on page 273 for further details.
11. If you experience problems during installation, see "Resolving problems when installing" on page 3517.

What to do next

When you have completed installation, see the following topics:

- "Verifying your WebSphere Message Broker installation" on page 290

This topic describes how to verify your installation on Linux on x86, Linux on x86-64, or Windows by using either the WebSphere Message Broker Toolkit or the WebSphere Message Broker Explorer.

- "Checking the broker operation mode and function level" on page 298
- "Start and main menu updates after installation" on page 3631

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

"Resolving problems when installing" on page 3517
Work through the advice provided to help you to deal with problems that can arise when the product is installed.

"Uninstalling" on page 331
Remove the Broker component, the WebSphere Message Broker Toolkit, or the WebSphere Message Broker Explorer from your computer.

"Resolving problems when uninstalling" on page 3525
Work through the advice provided to help you to deal with problems that can arise when the product is uninstalled.

"Finding the latest information" on page 232
Access the latest information for WebSphere Message Broker.

Related information:

WebSphere Message Broker Requirements

Installing the Broker component in console mode

Install the Broker component by using the installation wizard in console mode.

Before you begin

On Linux and UNIX systems, ensure that you set up the required security before you install the Broker component. See "Security on Linux and UNIX systems" on page 247 for more information.

About this task

The following list identifies the choices that you have for installing the Broker component in console mode, and the actions that you must take to complete your chosen task:

Procedure

1. Determine the installation wizard name for your operating system. See "Installation wizard names" on page 3626.
2. Locate the installation wizard in the root directory of the local or remote CD or DVD, or the network drive.
3. If you are installing on HP-Itanium, `/usr/sbin/` must be included in the PATH statement.
4. Enter the following command at a command prompt for default invocation:

   `installer` `-console` (where *installer* is the installation wizard name).

   If you start the installation from a directory other than the one in which the wizard exists, include the absolute or relative path with the command name.

5. When the wizard starts, navigate through the windows and provide input when requested. You must also read and accept the Software License Agreement.

   The installation process specifies a default installation location that, if accepted, is version specific; for example: `C:\Program Files\IBM\MQSI\v.r`. The default directory includes the version and release of the product, in the format v.r (version.release). You do have the option of specifying a different directory location, but the installation process does not proceed if the location that you specify already contains a previous version of the Broker component. If the location that you specify already contains a previous version of the Broker component, you must specify a different location. You can then migrate components to Version 7.0 from the previous version, when appropriate.

   Linux and UNIX only

   At the end of this installer, a panel asks if you would like to start the installation of the WebSphere Message Broker ODBC Database Extender (IE02) SupportPac. If selected, the IE02 SupportPac installer automatically launches in console mode. For more information, see "Installing the WebSphere Message Broker ODBC Database Extender (IE02)" on page 273.
6. When the summary window is displayed, check your choices and enter 1 to complete installation.
7. If you experience problems during installation, see "Resolving problems when installing" on page 3517.

What to do next

When you have completed installation, see the following topics:

- "Verifying your WebSphere Message Broker installation" on page 290

  This topic describes how to verify your installation on Linux on x86, Linux on x86-64, or Windows by using either the WebSphere Message Broker Toolkit or the WebSphere Message Broker Explorer.
- "Checking the broker operation mode and function level" on page 298
- "Start and main menu updates after installation" on page 3631

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

"Uninstalling" on page 331
Remove the Broker component, the WebSphere Message Broker Toolkit, or the WebSphere Message Broker Explorer from your computer.

"Installing the Broker component" on page 267
Use the installation wizard to install the Broker component.

"Finding the latest information" on page 232
Access the latest information for WebSphere Message Broker.

"Resolving problems when installing" on page 3517
Work through the advice provided to help you to deal with problems that can arise when the product is installed.

Related information:

Installing the Broker component in silent mode

Install the Broker component by using the installation wizard in silent mode.

About this task

The following list identifies the choices that you have for installing the Broker component in silent mode, and the actions that you must take to complete your chosen task.

Procedure

1. Determine the installation wizard name for your operating system. See "Installation wizard names" on page 3626.
2. Locate the installation wizard in the root directory of the local or remote CD or DVD, or the network drive.
3. If you are installing on HP-Itanium, `/usr/sbin/` must be included in the PATH statement.
4. Enter the following command at a command prompt for a typical installation with default settings.

   If you start the installation from a directory other than the one in which the installation wizard exists, include the absolute or relative path with the command name.

   Linux and UNIX

   `installer` `-silent -G licenseAccepted=true` (where *installer* is the installation wizard name).

   Windows

   Start the installation wizard within a **`start`** command with parameter `/w` to ensure that the installation completes before it returns to the command prompt:

   ```
   start /w setup.exe -silent -G licenseAccepted=true
   ```

   If you want to specify non-default settings, include a response file on the invocation. If you want to install the Broker component in a customized location, you must specify a response file. For more information about using response files, creating response files, and editing response files to define your requirements, see "Using response files with the Broker component" on page 3621.
5. The installation wizard completes without any user interaction.

   Linux and UNIX

   Note: The WebSphere Message Broker ODBC Database Extender (IE02) installer cannot be automatically started when running the install in silent mode. For more information, see "Installing the WebSphere Message Broker ODBC Database Extender (IE02)" on page 273.

   If you experience problems during installation, see "Resolving problems when installing" on page 3517.

What to do next

When you have completed installation, see the following topics:

- "Verifying your WebSphere Message Broker installation" on page 290

This topic describes how to verify your installation on Linux on x86, Linux on x86-64, or Windows by using either the WebSphere Message Broker Toolkit or the WebSphere Message Broker Explorer.

- "Checking the broker operation mode and function level" on page 298
- "Start and main menu updates after installation" on page 3631

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

"Uninstalling" on page 331
Remove the Broker component, the WebSphere Message Broker Toolkit, or the WebSphere Message Broker Explorer from your computer.

"Installing the Broker component" on page 267
Use the installation wizard to install the Broker component.

"Finding the latest information" on page 232
Access the latest information for WebSphere Message Broker.

"Resolving problems when installing" on page 3517
Work through the advice provided to help you to deal with problems that can arise when the product is installed.

Related information:

WebSphere Message Broker Requirements

Installing the WebSphere Message Broker ODBC Database Extender (IE02)

WebSphere Message Broker ODBC Database Extender encapsulates the unixODBC driver manager, which is an implementation of the Open DataBase Connectivity interface for UNIX systems. This topic describes how you install WebSphere Message Broker ODBC Database Extender.

Before you begin

Before you start:

Read the information about the unixODBC Project and the IBM solidDB product family.

Install WebSphere Message Broker Version 7.0.0.1 or later.

About this task

WebSphere Message Broker ODBC Database Extender is required when using WebSphere Message Broker to interface with an ODBC data source that is not supported through the DataDirect ODBC drivers.

Procedure

1. To install the WebSphere Message Broker ODBC Database Extender, download the package for your platform to the directory of your choice, or select the option to install the WebSphere Message Broker ODBC Database Extender from the product disk.

To download the package externally, select WebSphere Message Broker on the WebSphere MQ SupportPacs web page and locate the package file.

The package file is called `install-ie02.bin` on all platforms.

This package must be run with the same privileges (typically root) that you used when installing WebSphere Message Broker. The default IE02 installation directory is `/opt/ibm/IE02/`.

2. Run the installer in one of the following modes:
   - Graphical
   - Console
   - Silent

   a. **Graphical**. From within the directory where you downloaded the package file, run the following command:

      ```
      ./install-ie02.bin
      ```

      The installer launches within a separate window and guides you through the installation process. However, if the installer cannot launch in Graphical mode, it launches in console mode.

   b. **Console**. From within the directory where you downloaded the package file, run the following command:

      ```
      ./install-ie02.bin –i console
      ```

      The installer launches, and guides you through the installation process from within the same console as the command was run.

   c. **Silent**. When specifying a silent installation, a response file is required to define the required responses to the options provided by the installer. See following example file:

      ```
      # This file was built by the Replay feature of InstallAnywhere.
      # It contains variables that were set by Panels, Consoles or Custom
      # Code.

      #Has the license been accepted
      #-----------------------------
      LICENSE_ACCEPTED=TRUE

      #Choose Install Folder
      #---------------------
      USER_INSTALL_DIR=/opt/ibm/IE02/
      ```

      Note the above example specifies that the license has been accepted. By using this example script you are accepting the terms of the license agreement.

      The response file must be named `installer.properties` and must be located in the same directory as the WebSphere Message Broker ODBC Database Extender installer.

      Run the following command from within the directory where you downloaded the WebSphere Message Broker ODBC Database Extender:
      `./install-ie02.bin –i silent`

      After the installation has completed, review the installation log that is located under the installation path specified during the installation. The file name of the log is:
      `WebSphere_Message_Broker_ODBC_Database_Extender_InstallLog.log`.

      If you are installing over a previous installation (within the same installation directory), the previous installation is silently uninstalled before the new version is installed.

What to do next

Next, Configure the WebSphere Message Broker Database Extender.

Related tasks:

"Configuring the WebSphere Message Broker ODBC Database Extender (IE02)"
WebSphere Message Broker ODBC Database Extender encapsulates the unixODBC driver manager, which is an implementation of the Open DataBase Connectivity interface for UNIX systems, and this topic describes how you configure it.

"Connecting to a database from Linux and UNIX systems using the WebSphere Message Broker ODBC Database Extender (IE02)" on page 682
WebSphere Message Broker ODBC Database Extender encapsulates the unixODBC driver manager and this topic describes how you set up and configure the broker to use it.

Related reference:

"Sample WebSphere Message Broker ODBC Database Extender (IE02) configuration files" on page 3596
How you use WebSphere Message Broker ODBC Database Extender (IE02) to load the correct data source driver.

Configuring the WebSphere Message Broker ODBC Database Extender (IE02):

WebSphere Message Broker ODBC Database Extender encapsulates the unixODBC driver manager, which is an implementation of the Open DataBase Connectivity interface for UNIX systems, and this topic describes how you configure it.

Before you begin

Before you start:

Install the WebSphere Message Broker ODBC Database Extender.

About this task

To allow WebSphere Message Broker to take advantage of the additional database support, it needs to know where you have installed the SupportPac. The location of the SupportPac needs to be set by the value of the environment variable `IE02_PATH`.

This environment variable is automatically set within the WebSphere Message Broker profile during the installation of the SupportPac, by the creation of the script called `IE02.sh` which is placed within the directory `/var/mqsi/common/profiles`. See following example file:

```
#!/usr/bin/sh
# This file was created as part of the IBM WebSphere Message
Broker ODBC
# Database Extender SupportPac (IE02) install
export IE02_PATH=/opt/ibm/IE02/
```

Note: If you installed the SupportPac in silent mode, you must create a script in the directory `${MQSI_WORKPATH}/common/profiles` to set the environment variable `IE02_PATH`.

If you alter the location of your `$MQSI_WORKPATH` and, therefore, alter the location where WebSphere Message Broker dynamically runs additional scripts while

loading its profile, you must either copy the existing file, or create a new file containing the required contents within your new `${MQSI_WORKPATH}/common/profiles` directory.

You must not put the `lib` directory of the WebSphere Message Broker ODBC Database Extender into your library path (`LD_LIBRARY_PATH` or equivalent). If you do so, the WebSphere Message Broker ODBC Database Extender is not correctly loaded by WebSphere Message Broker and unpredictable results can occur.

What to do next

Configure the database that you are going to use; see "Connecting to a database from Linux and UNIX systems using the WebSphere Message Broker ODBC Database Extender (IE02)" on page 682

Related tasks:

"Connecting to a database from Linux and UNIX systems using the WebSphere Message Broker ODBC Database Extender (IE02)" on page 682
WebSphere Message Broker ODBC Database Extender encapsulates the unixODBC driver manager and this topic describes how you set up and configure the broker to use it.

Related reference:

"Sample WebSphere Message Broker ODBC Database Extender (IE02) configuration files" on page 3596
How you use WebSphere Message Broker ODBC Database Extender (IE02) to load the correct data source driver.

Installing the WebSphere Message Broker Toolkit

Use the installation wizard graphical interface to install the WebSphere Message Broker Toolkit on Windows and Linux on x86.

About this task

If you are installing the WebSphere Message Broker Toolkit on Windows, you can use the LaunchPad. For more information about using the Windows Launchpad, see "Installing by using the Windows Launchpad" on page 262.

If you do not want to use the Windows LaunchPad, complete the following steps.

Procedure

1. Check the product readme file, `readme.html`, for updates to these installation instructions. The product readme file includes information pertinent to all components and platforms, and is maintained in US English on the product readmes web page. Translated readme files are available on the documentation FTP site.
2. Check that your computer has enough memory and disk space for your installation. Requirements for memory and disk space depend on the installation operating system, and on the WebSphere Message Broker components that you are installing. For more information, see "Memory and disk space requirements" on page 3584.
3. Before installing WebSphere Message Broker Toolkit on Linux, check that the default permissions are set correctly by running the following command under a user ID with root authority:

   ```
   umask
   ```

A value of *0022* should be returned, indicating the permissions are correctly set. If any other value is returned, set the correct permissions by running the following command:

```
umask 0022
```

4. Decide whether you want to install the WebSphere Message Broker Toolkit from a server or if you want to install the WebSphere Message Broker Toolkit locally. These choices are described in "Accessing CDs and DVDs" on page 252 for CDs, DVDs, and for installation images that you can download from Passport Advantage.
5. Decide whether you want to use the installation wizard graphical interface to install the WebSphere Message Broker Toolkit or if you want to complete a silent installation. For more information about these interfaces, see "How to install and uninstall the WebSphere Message Broker Toolkit" on page 3623. If you decide to use the installation wizard graphical interface, continue with these instructions. Alternatively, to use a silent installation, see "Installing the WebSphere Message Broker Toolkit in silent mode" on page 279.

   In either scenario, the installation of the WebSphere Message Broker Toolkit is controlled by the IBM Installation Manager. If the Installation Manager is not already installed, it is installed before the WebSphere Message Broker Toolkit is installed.
6. Start the installation wizard graphical interface by navigating to the `/Message_Broker_Toolkit_V7.0/disk1` directory and running the batch file `installToolkit.sh` as root on Linux, or `installToolkit.bat` on Windows. The installation wizard checks your system locale setting. If the locale setting listed in "Multicultural support" on page 3628 is supported, the wizard continues in this locale. If the current setting is not supported, the wizard continues in US English. This language is used for installation only, and does not affect other processes on your computer.

   The Installation Manager starts and the Install Packages window opens. The installation wizard is preconfigured to install both the Installation Manager and the WebSphere Message Broker Toolkit, therefore the WebSphere Message Broker Toolkit packages are already selected in this window. If the Installation Manager has not been installed on this computer, its packages are also selected and cannot be cleared.
7. Click **Next** to continue. The Software License Agreement window opens.
8. Read the license agreement, select **I accept the terms in the license agreements**, and click **Next**.

   If you do not accept the license, the installation wizard ends. If you have not installed the Installation Manager on this computer, or if you have installed the Installation Manager but have not yet installed any product that is managed by the Installation Manager, the Shared Directory window opens. Continue with step 9. Otherwise, because the shared resources directory has already been defined, the Package group directory window opens, so you can continue with step 10 on page 278.
9. If the WebSphere Message Broker Toolkit is not the first product that is installed by the Installation Manager, you might not be prompted to specify the shared resources directory. If you are prompted, specify the location of the shared resources directory that is used by all the products that are managed by the Installation Manager. The following directories show the default locations:
   - Linux on x86: `/opt/IBM/SDPShared/`.
   - Windows: `C:\Program Files\IBM\SDPShared\` for 32-bit editions, or `C:\Program Files (x86)\IBM\SDPShared\`*`v.r`* for 64-bit editions.

To specify a different location, type over the default location, or click **Browse**.

The shared resources directory must not contain another installation of WebSphere Message Broker Toolkit, or other files or products.

If the Installation Manager is not yet installed, you must also specify its installation directory. The following directories show the default locations:

- Linux on x86: `/opt/IBM/InstallationManager/`.
- Windows: `C:\Program Files\IBM\InstallationManager\` for 32-bit editions, or `C:\Program Files (x86)\IBM\InstallationManager\` for 64-bit editions.

To specify a different location, type over the default location, or click **Browse**.

Click **Next**. The Package group directory window opens. You can create a new package group for the WebSphere Message Broker Toolkit, which requires you to specify an installation directory, or you can choose an existing package group to share the resources.

For more information about installing the WebSphere Message Broker Toolkit into a package group, and for further reference information about Package groups, see "IBM Installation Manager" on page 3600.

10. If you want to use the WebSphere Message Broker Toolkit in a locale other than US English, select additional support from the list presented. English is always selected and installed; you cannot clear this selection. If you select one or more alternative locales, documentation and properties files for all supported languages are installed. Click **Next** to continue. The Summary window opens.
11. Check your choices and click **Back** if you want to make further changes to your responses on any of the previous windows. This window displays guidance information for the space required for the packages that you are about to install and indicates that your disk has sufficient space.

    Click **Next** to start installation. The Install Progress window opens.
12. The features that you are installing, their associated directories, and the locales that you have selected display for information. A progress bar displays, which you can use to check the status of the installation. When the installation has finished, the Completion window opens.
13. The wizard displays an indication of success or failure, and lists the products and options that have been installed. Click **View Log File** to check the results of the installation.

    On Windows, you can indicate that you want the WebSphere Message Broker Toolkit to launch when you click **Finish** to close the wizard. This option is not available on Linux on x86 because you might want to complete verification while logged on as a different user ID that does not have root authority.
14. If you experience problems during installation, see "Resolving problems when installing" on page 3517.

What to do next

When you have completed the installation, see "Verifying your WebSphere Message Broker installation" on page 290, "Checking the broker operation mode and function level" on page 298, and "Start and main menu updates after installation" on page 3631.

If you do not install optional locales at this time, you can install them later in the following way:

- On Linux on x86, navigate to the `/eclipse` directory within the Installation Manager installation directory, and start the Installation Manager program `IBMIM`.

(You cannot use the main menu entries unless you are already logged on as root; the menu item does not provide an option to become root, and root authority is required for all installation tasks.)

- On Windows, click **Start** > **Programs** > **IBM Installation Manager** > **IBM Installation Manager** to launch the Installation Manager, and click **Modify Packages** to change your installation.

  If you prefer to use the command line, navigate to the `\eclipse` directory within the Installation Manager installation directory, and start the Installation Manager program `IBMIM.exe`.

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

"Resolving problems when installing" on page 3517
Work through the advice provided to help you to deal with problems that can arise when the product is installed.

"Uninstalling" on page 331
Remove the Broker component, the WebSphere Message Broker Toolkit, or the WebSphere Message Broker Explorer from your computer.

"Finding the latest information" on page 232
Access the latest information for WebSphere Message Broker.

"Applying service to the WebSphere Message Broker Toolkit" on page 325
Apply maintenance updates and program fixes to the WebSphere Message Broker Toolkit.

Related information:

WebSphere Message Broker Requirements

Installing the WebSphere Message Broker Toolkit in silent mode

Install the WebSphere Message Broker Toolkit by using the installation wizard in silent mode.

About this task

To perform a silent installation, complete the following steps:

Procedure

1. Determine the installation wizard name for your operating system. See "Installation wizard names" on page 3626.
2. Locate the installation wizard in the `/Message_Broker_Toolkit_V7.0/disk1` directory of the local or remote DVD, or the network drive.
3. Enter the following command at a command prompt for an installation with all default settings:

   Linux on x86

   ```
   ./installToolkit-silent.sh
   ```

   Windows

   ```
   installToolkit-silent.bat
   ```

The `installToolkit-silent.sh` and `installToolkit-silent.bat` files use the `/Message_Broker_Toolkit_V7.0/disk1/IBMInstallationManager/mbtoolkit-silent.xml` response file that contains all the default settings for installation.

If you want to specify non-default settings, include a different response file on the invocation. To create a different response file, complete the first installation by using the graphical interface, specifying the `-record` option. The installation wizard records a response file that includes all your chosen selections. For more information about how to record and use response files, see "Using response files with the WebSphere Message Broker Toolkit" on page 3625.

4. The installation wizard completes without any user interaction. Check the log for success or failure of the installation process.
5. If you experience problems during installation, refer to "Resolving problems when installing" on page 3517.

What to do next

When you have completed installation, see "Verifying your WebSphere Message Broker installation" on page 290, "Checking the broker operation mode and function level" on page 298, and "Start and main menu updates after installation" on page 3631.

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

"Uninstalling" on page 331
Remove the Broker component, the WebSphere Message Broker Toolkit, or the WebSphere Message Broker Explorer from your computer.

"Installing the WebSphere Message Broker Toolkit" on page 276
Use the installation wizard graphical interface to install the WebSphere Message Broker Toolkit on Windows and Linux on x86.

"Finding the latest information" on page 232
Access the latest information for WebSphere Message Broker.

"Resolving problems when installing" on page 3517
Work through the advice provided to help you to deal with problems that can arise when the product is installed.

Related information:

WebSphere Message Broker Requirements

Installing WebSphere Message Broker Explorer

To use WebSphere Message Broker Explorer only, without installing the complete WebSphere Message Broker Toolkit, use the WebSphere Message Broker Explorer installation wizard to install the WebSphere Message Broker Explorer.

Before you begin

Before you start:

- Alternatively, to install the complete WebSphere Message Broker Toolkit, see "Installing the WebSphere Message Broker Toolkit" on page 276.

- The WebSphere Message Broker Explorer installation searches the Windows registry for a version of WebSphere MQ Explorer. If registry access is disabled by a local security policy such as "Prevent access to registry editing tools", the installation cannot find WebSphere MQ Explorer. Before you install WebSphere Message Broker Explorer, remove this restriction temporarily and reinstate it after the installation.

About this task

To install WebSphere Message Broker Explorer on Windows or Linux on x86 and Linux on x86-64, complete the following tasks.

One version only of the WebSphere Message Broker Explorer can be active at any one time because one version only of the WebSphere MQ Explorer can be installed on any one system.

If you already have WebSphere Message Broker Explorer installed, and you decide to install a later version of WebSphere Message Broker Explorer, you must upgrade the existing version.

If you add another installation of WebSphere MQ after you install WebSphere Message Broker Explorer, you must install WebSphere Message Broker Explorer again to enable it in all WebSphere MQ installations.

To upgrade the existing version, complete the following steps.

1. Uninstall WebSphere Message Broker Explorer. For more information, see "Uninstalling the WebSphere Message Broker Explorer" on page 346. You can uninstall the WebSphere Message Broker Explorer only if you are the administration user (for example, root on Linux on x86 and Linux on x86-64, or Administrator on Windows).
2. Check that no files exist in the previous installation directory.
3. Install WebSphere Message Broker Explorer.

After you install the new version of the WebSphere Message Broker Explorer, if you are upgrading from an existing version of the WebSphere Message Broker Explorer, you must complete the following steps:

1. To initialize the new version of the WebSphere Message Broker Explorer, run the following command from a command line in which that **mqsiprofile** command has not run:

   ```
   strmqcfg -i
   ```

   To run this command, you must be the administration user.
2. Start WebSphere Message Broker Explorer either by using the WebSphere Message Broker Explorer Windows shortcut or by running the following command from a command line in which the **mqsiprofile** command has not run:

   ```
   strmqcfg -c -d
   ```

Windows On Windows, the **Add/Remove** program panels point to the new installation location. To remove the older version, navigate to the old location on the file system and run the uninstaller that is located there.

Windows If you are installing WebSphere Message Broker Explorer on Windows, you can use the Launchpad to complete this task. If you do not want to use the Launchpad, complete the tasks in this topic instead.

To use the installation wizard for the WebSphere Message Broker Explorer with a Java compatible screen reader for accessibility reasons, see "Installing the WebSphere Message Broker Explorer by using console mode with a screen reader" on page 285.

Linux To install WebSphere Message Broker Explorer on Linux on x86 and Linux on x86-64, you must have write access to the locations that you want to use for the installation and data (configuration) directories. If you use an installation directory in your home directory rather than the default installation directory, do not use a tilde character (~) as a path prefix. Specify the directory path name in full. An InstallAnywhere error occurs if you use a tilde character, which means the product is not installed and you must repeat the installation process. You can install to a private location if you do not have root access to the default installation location.

If you encounter problems during the installation process, you can view the installation log `MBExplorer_install.log`. The installation log is created in the installation directory, for example `C:\Program Files\IBM\MBExplorer\MBExplorer_install.log`.

Note: The installation log is not created until the installation wizard is complete and you click **Done** to exit the wizard.

If you previously installed the IS02: WebSphere Message Broker Explorer plug-in SupportPac, you must complete the following steps before you install WebSphere Message Broker Explorer:

1. Move or delete the location to which you extracted the IS02 SupportPac.
2. Delete the `BrokerExplorer.link` file, which you copied into the `links` directory. On Windows, the directory name is `C:\Program Files\IBM\WebSphere MQ\eclipseSDK33\eclipse\links\`. On Linux on x86 and Linux on x86-64, the directory name is `/opt/mqm/eclipseSDK33/eclipse/links`.
3. Start WebSphere Message Broker Explorer either by using the WebSphere Message Broker Explorer Windows shortcut or by running the following command from a command line in which the **mqsiprofile** command has not run:

   ```
   strmqcfg -c -d
   ```

4. Optional: If you want a new installation of your WebSphere Message Broker Explorer, complete one of the following steps:
   - Move or delete the WebSphere Message Broker Explorer `metadata` directory. On Windows, the directory name is `C:\Documents and Settings\<user>\Application Data\IBM\MQ Explorer\.metadata`. On Linux on x86 and Linux on x86-64, the directory name is `/home/user/.mqdata/.metadata`.
   - Alternatively, switch your Eclipse workspace.

   Moving or deleting the `metadata` directory, or switching your Eclipse workspace, resets your WebSphere Message Broker Explorer to its initial startup state. You must then reconnect to any remote queue managers. Any local queue manager is discovered automatically.

The following list identifies the choices that you have for installing the WebSphere Message Broker Explorer, and the actions that you must take to complete your chosen task.

Procedure

1. Check the `readme.html` for any updates to these installation instructions. The readme file location is shown in "System requirements" on page 3581.
2. Check that you have enough memory and disk space. For more information, see "Memory and disk space requirements" on page 3584.
3. If you do not already have WebSphere MQ installed, install it before you install the WebSphere Message Broker Explorer.
4. Decide whether you want to install from a server, or to install locally on each system. These choices are described in "Accessing CDs and DVDs" on page 252 for both CDs and DVDs, and for images that you downloaded from Passport Advantage (if you are registered with the scheme). The instructions here do not differentiate between CDs, DVDs, and downloaded images; their behavior is the same.
5. Decide whether to use the graphical installation, a console installation, or a silent installation.
   - To use the graphical installation, continue with these instructions.
   - To use the installation wizard with a screen reader, see "Installing the WebSphere Message Broker Explorer by using console mode with a screen reader" on page 285.
   - Windows To use a console installation on Windows, see "Installing the WebSphere Message Broker Explorer in console mode on Windows" on page 285.
   - Linux To use a console installation on Linux, see "Installing the WebSphere Message Broker Explorer in console mode on Linux" on page 287.
   - To use a silent installation, see "Installing the WebSphere Message Broker Explorer in silent mode" on page 288.
6. Open the installation wizard by using the executable file or binary file. On Windows, this file is `install.exe`. On Linux on x86 and Linux on x86-64, this file is `install.bin`. These files are in the `\MBExplorer` directory on the DVD.
7. When you open the wizard, continue with the installation process by working through each of the following panels. The installation wizard itself contains help information about selected panels.
8. Select the language to use for the installation process, then click **OK**.
9. On the Introduction panel, click **Next**.
10. Read the software license agreement, select the option to accept the terms of the license, then click **Next**.
11. Enter (or browse for) a product installation directory for the WebSphere Message Broker Explorer, or accept the default location. On Windows, the default installation directory for WebSphere Message Broker Explorer is `C:\Program Files\IBM\MBExplorer`. On Linux on x86 and Linux on x86-64, the default installation directory is `/opt/IBM/MBExplorer`.

    If the product installation directory exists because you previously installed WebSphere Message Broker Explorer, either click **Refresh** to refresh the existing installation or click **Select** to select a new product directory.
12. Click **Next**.
13. Read the summary panel. If necessary, click **Previous** to go back and modify earlier panels.
14. Click **Install** and wait while the files are installed.
15. On the Install complete panel, click **Done**.

Results

You can now use the WebSphere Message Broker Explorer. To start the WebSphere Message Broker Explorer, complete one of the following steps:

- On Windows:

  Click **Start** > **All Programs** > **IBM WebSphere Message Broker 7.0** > **IBM WebSphere Message Broker Explorer**.
- On Linux on x86 and Linux on x86-64:

  Open a command shell in which the **mqsiprofile** command has not run and enter the `strmqcfg` command, or run `/usr/bin/strmqcfg`.

What to do next

When you complete installation, see "Verifying your WebSphere Message Broker installation" on page 290.

Related tasks:

"Setting up a command environment" on page 213
After you have installed the product on one of the distributed systems, you must initialize the environment before you can use a runtime component or command.

Chapter 7, "Configuring brokers for test and production environments," on page 579
Create one or more brokers on one or more computers, and configure them on your test and production systems to process messages that contain your business data.

"Installing the WebSphere Message Broker Explorer by using console mode with a screen reader" on page 285
You can install the WebSphere Message Broker Explorer by using console mode with screen reader software for accessibility reasons.

"Installing the WebSphere Message Broker Explorer in console mode on Windows" on page 285
You can install WebSphere Message Broker Explorer using the installation wizard in console mode on Windows and Linux.

"Installing the WebSphere Message Broker Explorer in console mode on Linux" on page 287
You can install WebSphere Message Broker Explorer using the installation wizard in console mode on Windows and Linux.

"Installing the WebSphere Message Broker Explorer in silent mode" on page 288
You can install WebSphere Message Broker Explorer by using the installation wizard in silent mode.

"Resolving problems when installing" on page 3517
Work through the advice provided to help you to deal with problems that can arise when the product is installed.

"Verifying your WebSphere Message Broker installation" on page 290
You can verify your installation of WebSphere Message Broker by using the WebSphere Message Broker Toolkit or the WebSphere Message Broker Explorer.

"Applying service to the WebSphere Message Broker Explorer" on page 329
You can apply maintenance or fixes to the WebSphere Message Broker Explorer.

"Installing the WebSphere Message Broker Toolkit" on page 276
Use the installation wizard graphical interface to install the WebSphere Message Broker Toolkit on Windows and Linux on x86.

"Uninstalling" on page 331
Remove the Broker component, the WebSphere Message Broker Toolkit, or the

WebSphere Message Broker Explorer from your computer.

"Finding the latest information" on page 232
Access the latest information for WebSphere Message Broker.

Installing the WebSphere Message Broker Explorer by using console mode with a screen reader

You can install the WebSphere Message Broker Explorer by using console mode with screen reader software for accessibility reasons.

About this task

Complete the following steps to use console mode with screen reader software:

Procedure

1. Install a screen reader program, such as JAWS.
2. Install the WebSphere Message Broker Explorer by using console mode. For more information, see "Installing the WebSphere Message Broker Explorer in console mode on Windows" or "Installing the WebSphere Message Broker Explorer in console mode on Linux" on page 287.

Related tasks:

"Installing the WebSphere Message Broker Explorer in console mode on Windows"
You can install WebSphere Message Broker Explorer using the installation wizard in console mode on Windows and Linux.

"Installing the WebSphere Message Broker Explorer in console mode on Linux" on page 287
You can install WebSphere Message Broker Explorer using the installation wizard in console mode on Windows and Linux.

"Uninstalling" on page 331
Remove the Broker component, the WebSphere Message Broker Toolkit, or the WebSphere Message Broker Explorer from your computer.

"Finding the latest information" on page 232
Access the latest information for WebSphere Message Broker.

Installing the WebSphere Message Broker Explorer in console mode on Windows

You can install WebSphere Message Broker Explorer using the installation wizard in console mode on Windows and Linux.

About this task

Use the following steps to install the WebSphere Message Broker Explorer using the console mode on Windows. If you encounter any problems during the installation process, you can view the install log `MBExplorer_install.log`.

Procedure

1. Type the following command on a command line to run a console install:
   `<dvd_rom>/MBExplorer/install.exe -i console`
2. Select the language that you want to use for the installation process by typing the number next to the language and press **Enter**. The installer uses the term "locale" rather than "language". Alternatively, press **Enter** to accept the default language.
3. Read the console install instructions, and press **Enter** to continue.

4. Read the software license agreement and type 1 to accept the terms of the license. Press **Enter**.
5. Type the name of the product installation directory for the WebSphere Message Broker Explorer and press **Enter**. Alternatively, press **Enter** to accept the default location. The default installation directory for the WebSphere Message Broker Explorer on Windows is `C:\Program Files\IBM\MBExplorer`.
6. Read the Pre-Installation Summary information and press **Enter** to install the WebSphere Message Broker Explorer. Wait while the files are installed.
7. On the Installation Complete panel, press **Enter** to exit the console installer.

Results

You can now use the WebSphere Message Broker Explorer. To use the WebSphere Message Broker Explorer you must start the WebSphere MQ Explorer. To start the WebSphere MQ Explorer, click **Start** > **All Programs** > **IBM WebSphere MQ** > **WebSphere MQ Explorer**, or open a command line in which the **mqsiprofile** command has not been run and enter the `strmqcfg` command.

What to do next

When you have completed installation, see "Verifying your WebSphere Message Broker installation" on page 290.

Related tasks:

"Setting up a command environment" on page 213
After you have installed the product on one of the distributed systems, you must initialize the environment before you can use a runtime component or command.

"Installing the WebSphere Message Broker Explorer in console mode on Linux" on page 287
You can install WebSphere Message Broker Explorer using the installation wizard in console mode on Windows and Linux.

Chapter 7, "Configuring brokers for test and production environments," on page 579
Create one or more brokers on one or more computers, and configure them on your test and production systems to process messages that contain your business data.

"Installing WebSphere Message Broker Explorer" on page 280
To use WebSphere Message Broker Explorer only, without installing the complete WebSphere Message Broker Toolkit, use the WebSphere Message Broker Explorer installation wizard to install the WebSphere Message Broker Explorer.

"Installing the WebSphere Message Broker Explorer by using console mode with a screen reader" on page 285
You can install the WebSphere Message Broker Explorer by using console mode with screen reader software for accessibility reasons.

"Installing the WebSphere Message Broker Explorer in silent mode" on page 288
You can install WebSphere Message Broker Explorer by using the installation wizard in silent mode.

"Verifying your WebSphere Message Broker installation" on page 290
You can verify your installation of WebSphere Message Broker by using the WebSphere Message Broker Toolkit or the WebSphere Message Broker Explorer.

"Uninstalling" on page 331
Remove the Broker component, the WebSphere Message Broker Toolkit, or the WebSphere Message Broker Explorer from your computer.

"Finding the latest information" on page 232
Access the latest information for WebSphere Message Broker.

Installing the WebSphere Message Broker Explorer in console mode on Linux

You can install WebSphere Message Broker Explorer using the installation wizard in console mode on Windows and Linux.

About this task

Use the following steps to install the WebSphere Message Broker Explorer using the console mode on Linux. If you encounter any problems during the installation process, you can view the install log `MBExplorer_install.log`.

Procedure

1. Type the following command on a command line to run a console install:
   `<cd_rom>/MBExplorer/install.bin -i console`
2. Select the language that you want to use for the installation process by typing the number next to the language and press **Enter**. The installer uses the term "locale" rather than "language". Alternatively, press **Enter** to accept the default language.
3. Read the software license agreement and type 1 to accept the terms of the license. Press **Enter**.
4. Type the name of the product installation directory for the WebSphere Message Broker Explorer and press **Enter**. Alternatively, press **Enter** to accept the default location. The default installation directory for the WebSphere Message Broker Explorer on Linux systems is `/opt/IBM/MBExplorer`. If the product installation directory already exists because you have previously installed the WebSphere Message Broker Explorer, you can either refresh the existing installation or select a new product directory.
5. Confirm that the install location is correct, and press **Enter** to continue.
6. Read the Pre-Installation Summary information and press **Enter** to install the WebSphere Message Broker Explorer. Wait while the files are installed.
7. On the Installation Complete panel, press **Enter** to exit the console installer.

Results

You can now use the WebSphere Message Broker Explorer. To use the WebSphere Message Broker Explorer you must start the WebSphere MQ Explorer. To start the WebSphere MQ Explorer, open a command shell in which the **`mqsiprofile`** command has not been run and enter the `strmqcfg` command, or run `/usr/bin/strmqcfg`.

What to do next

When you have completed installation, see "Verifying your WebSphere Message Broker installation" on page 290.

Related tasks:

"Setting up a command environment" on page 213
After you have installed the product on one of the distributed systems, you must initialize the environment before you can use a runtime component or command.

"Installing the WebSphere Message Broker Explorer in console mode on Windows" on page 285
You can install WebSphere Message Broker Explorer using the installation wizard in console mode on Windows and Linux.

Chapter 7, "Configuring brokers for test and production environments," on page 579
Create one or more brokers on one or more computers, and configure them on your test and production systems to process messages that contain your business data.

"Installing WebSphere Message Broker Explorer" on page 280
To use WebSphere Message Broker Explorer only, without installing the complete WebSphere Message Broker Toolkit, use the WebSphere Message Broker Explorer installation wizard to install the WebSphere Message Broker Explorer.

"Installing the WebSphere Message Broker Explorer by using console mode with a screen reader" on page 285
You can install the WebSphere Message Broker Explorer by using console mode with screen reader software for accessibility reasons.

"Installing the WebSphere Message Broker Explorer in silent mode"
You can install WebSphere Message Broker Explorer by using the installation wizard in silent mode.

"Verifying your WebSphere Message Broker installation" on page 290
You can verify your installation of WebSphere Message Broker by using the WebSphere Message Broker Toolkit or the WebSphere Message Broker Explorer.

"Uninstalling" on page 331
Remove the Broker component, the WebSphere Message Broker Toolkit, or the WebSphere Message Broker Explorer from your computer.

"Finding the latest information" on page 232
Access the latest information for WebSphere Message Broker.

Installing the WebSphere Message Broker Explorer in silent mode

You can install WebSphere Message Broker Explorer by using the installation wizard in silent mode.

About this task

Before you can run the installation wizard in silent mode, you must create a response file that contains the installation options. You can create a response file during an installation using the graphical installation wizard, or you can use a sample response file supplied in the `samples-scripts` directory in the root directory of the installation media. To install silently, you specify the path to the response file as an argument to the installation command.

Use the following steps to create the response file and run the installation wizard in silent mode:

Procedure

1. Create a response file using the graphical installation wizard or by manually editing a response file:
   - Create a response file during an installation using the graphical installation wizard. You must provide a full path for the response file, or the response file is not created by the installation wizard.

     Enter the following commands into a command prompt:

– On Windows:

```
install.exe -r <filepath>
```

– On Linux:

```
install.bin -r <filepath>
```

This command starts the GUI installation as normal, except that it also records all your answers and saves them in the file you specified, for example, `c:\temp\mbx-response.properties`. You can then use this response file with the silent installer for subsequent installations of WebSphere Message Broker Explorer.

- Alternatively, use the supplied template in the `samples-scripts` directory in the root directory of the local or remote CD or DVD, or the network drive. Or use the following template to create a response file manually:

```
# Thu Jul 09 16:44:28 BST 2009
# Replay feature output
# ---------------------
# This file was built by the Replay feature of InstallAnywhere.
# It contains variables that were set by Panels, Consoles or Custom Code.

#Has the license been accepted
#-----------------------------
LICENSE_ACCEPTED=TRUE

#Choose Install Folder
#---------------------
USER_INSTALL_DIR=C:\\Program Files\\IBM\\MBExplorer
```

2. To run the installation wizard in silent mode with the response file, run the following command:
   - On Windows:

     ```
     install.exe -i silent -f <filename>
     ```

   - On Linux:

     ```
     install.bin -i silent -f <filename>
     ```

   where *<filename>* is the path to the response file, for example: `d:\messagebroker_runtime1\sample-scripts\mbx-response.properties`.

Results

You can now use the WebSphere Message Broker Explorer. To use the WebSphere Message Broker Explorer you must start the WebSphere MQ Explorer. To start the WebSphere MQ Explorer, complete one of the following steps:

- On Windows:

  Click **Start** > **All Programs** > **IBM WebSphere MQ** > **WebSphere MQ Explorer**, or, on a command line in which the **mqsiprofile** command has not been run, enter the `strmqcfg` command.

- On Linux:

  Open a command shell in which the **mqsiprofile** command has not been run and enter the `strmqcfg` command, or run `/usr/bin/strmqcfg`.

What to do next

When you have completed installation, see "Verifying your WebSphere Message Broker installation" on page 290.

Related tasks:

"Setting up a command environment" on page 213
After you have installed the product on one of the distributed systems, you must initialize the environment before you can use a runtime component or command.

Chapter 7, "Configuring brokers for test and production environments," on page 579
Create one or more brokers on one or more computers, and configure them on your test and production systems to process messages that contain your business data.

"Installing WebSphere Message Broker Explorer" on page 280
To use WebSphere Message Broker Explorer only, without installing the complete WebSphere Message Broker Toolkit, use the WebSphere Message Broker Explorer installation wizard to install the WebSphere Message Broker Explorer.

"Installing the WebSphere Message Broker Explorer by using console mode with a screen reader" on page 285
You can install the WebSphere Message Broker Explorer by using console mode with screen reader software for accessibility reasons.

"Installing the WebSphere Message Broker Explorer in console mode on Windows" on page 285
You can install WebSphere Message Broker Explorer using the installation wizard in console mode on Windows and Linux.

"Verifying your WebSphere Message Broker installation"
You can verify your installation of WebSphere Message Broker by using the WebSphere Message Broker Toolkit or the WebSphere Message Broker Explorer.

"Uninstalling" on page 331
Remove the Broker component, the WebSphere Message Broker Toolkit, or the WebSphere Message Broker Explorer from your computer.

"Finding the latest information" on page 232
Access the latest information for WebSphere Message Broker.

Verifying your WebSphere Message Broker installation

You can verify your installation of WebSphere Message Broker by using the WebSphere Message Broker Toolkit or the WebSphere Message Broker Explorer.

About this task

This section describes how to verify your installation on Linux on x86, Linux on x86-64, or Windows by using either the WebSphere Message Broker Toolkit or the WebSphere Message Broker Explorer.

- "Verifying your installation by using the WebSphere Message Broker Toolkit" on page 291
- "Verifying your installation using the WebSphere Message Broker Explorer" on page 295

After verification, you can keep all the components to carry out further development and unit test. Development and test environments are restricted to Linux on x86, Linux on x86-64, and Windows computers on which one copy of each component is installed on each computer. You can also create additional components and resources on your test computers to investigate the ways in which your business requirements can be met by this product.

If you have purchased WebSphere Message Broker Remote Adapter Deployment, Starter Edition, or Entry Edition, you must modify the operation mode of the broker after you have completed the procedures in this chapter (unless you are

running a test environment). The broker operation mode is always set to the default `enterprise` when you install the Broker component from the full product packages. If you intend to keep and use the default broker that you create in this chapter, you are required to modify its mode to conform to the terms of your license. See "Checking the broker operation mode and function level" on page 298 for details of this task.

Verifying your installation by using the WebSphere Message Broker Toolkit

Use the instructions in this tutorial to verify your installation of WebSphere Message Broker and learn how to run samples with the WebSphere Message Broker Toolkit.

Before you begin

You must have installed the WebSphere Message Broker Toolkit to run this tutorial.

About this task

To verify your installation by using the WebSphere Message Broker Toolkit, complete the following tasks:

- Create a Default Configuration.
- Run the Pager samples.
- (Optional) Start the Samples Preparation wizard.
- Remove the samples.

Before you can run sample programs, you must use the Default Configuration wizard to create a broker, which has a fixed name and properties that the samples depend on.

The Default Configuration wizard requires that the following conditions are met:

- You have installed the Broker component and the WebSphere Message Broker Toolkit.
- None of the Default Configuration wizard resources exist (the resources are listed in the table that is included later in this section).
- This configuration is required for test and evaluation purposes only.
- Your current user ID has the following characteristics on Windows only:
  - It is a member of groups mqbrkrs and mqm.
  - It has Administrator authority.
  - It is a local ID, not a domain ID.

  For more information about these security requirements, see "Broker component security" on page 497.

You cannot complete the configuration and verification described here if the previously described conditions are not met.

Use the following instructions to complete these tasks:

Procedure

1. Start the WebSphere Message Broker Toolkit:
   - On Windows:

On Windows, you cannot complete verification unless you have Administrator authority; carry out verification with the same user ID that you used to complete installation.

If you did not launch the WebSphere Message Broker Toolkit from the installation wizard, launch it from the **Start** menu, or run the script file provided. On a command line, navigate to the root directory of the package group and enter the following command:

```
mb.exe
```

The script file runs the following command; if you prefer you can use this command yourself:

```
eclipse.exe -product com.ibm.etools.msgbroker.tooling.ide
```

- On Linux:

  On Linux on x86 and Linux on x86-64, you do not need root authority to complete verification. You cannot launch the WebSphere Message Broker Toolkit from the installation wizard because you might experience problems during operation if you were to create resources such as brokers when you are logged in as root, and this option is therefore unavailable.

  Log off from the user ID with which you have installed the product. Log in as the same ID (if ID is not root), or log in as another ID, but do not become root.

  Launch the WebSphere Message Broker Toolkit from the main menu or run the script file provided. On a command line, navigate to the root directory of the package group and enter the following command:

  ```
  ./launcher
  ```

  If you prefer, you can also run the application directly:

  ```
  ./eclipse -product com.ibm.etools.msgbroker.tooling.ide
  ```

  However, you must set the *LD_LIBRARY_PATH* before running the application. For details about how to set *LD_LIBRARY_PATH*, see the launcher script.

When you first launch the WebSphere Message Broker Toolkit, you are asked to specify the location of your workspace. This directory exists on your local drive, and is where the WebSphere Message Broker Toolkit stores all the resources that you create. You can accept the default directory shown, or you can specify your own choice either by typing it in, or by clicking **Browse** to specify the location. Select **Use this as the default and do not ask again** to inhibit the display of the workspace dialog next time you launch the WebSphere Message Broker Toolkit.

The WebSphere Message Broker Toolkit opens and the Welcome page is displayed.

2. Click the **Get Started** icon  to begin the configuration and verification process.

   The Get Started page opens, from which you can start the Quick Tour or create a default configuration that is used by a sample program to verify that your installation is successful.

3. Create the Default Configuration:

   a. On the Get Started page, click the link to **Create the Default Configuration**.

       The "Creating the Default Configuration" page opens.

   b. Click **Start the Default Configuration wizard**.

Follow the guidance provided by the wizard to navigate through its pages.

The wizard creates a default broker that can be used by a sample program to verify that your installation is successful.

The wizard displays a progress bar that shows which task it is currently performing. It also reports on all the actions that it takes by writing progress information into a scrollable text window from which you can copy and paste some or all of the information displayed.

The information in the text window is also written to a log file in your workspace directory structure. The default workspace directory is shown in the example, but you can choose another location when you start the WebSphere Message Broker Toolkit.

Linux on x86 and Linux on x86-64

```
user_home_dir/IBM/wmbt70/workspace/.metadata/
DefaultConfigurationWizard.log
```

Windows

```
user_home_dir\IBM\wmbt70\workspace\.metadata\
DefaultConfigurationWizard.log
```

If the wizard encounters an error in processing, it informs you of what has happened and returns any error information, for example a return code from a command. If you know why the error has occurred from the error text, and can correct the situation, you can do so now. Return to the error message display and click **Yes** to continue the wizard.

If you do not understand the error, and do not know how to fix it, click **No**. If the wizard can, it rolls back all the actions that it has taken so far, so that when it completes, your system is in the same state as it was before you started the wizard. The text window shows you exactly what the wizard has, and has not, done.

Click **Open Log File** to access the log from the summary page of the wizard; this option is available whether the wizard has succeeded or failed.

The wizard creates the resources shown in the following table.

Table 6. Resources created by the Default Configuration wizard

Name	Type
MB7BROKER	Broker
MB7QMGR	WebSphere MQ queue manager that hosts the broker. The queue manager has a listener at the first available port greater than or equal to 2414.

It also starts the broker so that it is ready to process a sample.

c. On the final page, ignore the option to start the Samples Preparation wizard; you start this wizard later in these instructions.

d. Click **Finish** to close the wizard. When the wizard completes, it opens the Broker Application Development perspective and displays the resources that the wizard has created.

4. To verify your installation, click **Help** > **Samples and Tutorials** > **WebSphere Message Broker Toolkit - Message Broker** to open the Samples and Tutorials panel. The Samples and Tutorials panel can also be opened from the Welcome page.

   a. Expand **Application Samples**, and click **Pager Samples** to open the Pager samples page. The following options are displayed:

      - **Import and deploy the Pager samples**

This option starts the Samples Preparation wizard, which helps you to import the samples into your workspace, and to deploy the samples and associated resources (for example, message flows) to the default broker.

- **Run the Pager samples**

  This option opens the help page that contains a description of each of three sample programs, and icons that you can click to start each one.

- **Find out what the Pager samples do**

  This option opens a page that describes in detail what the Pager samples do and how they work. You can examine the message flows that implement the sample function, and the messages that are handled by those flows.

b. Click **Import and deploy the Pager samples**. The Samples Preparation wizard starts and displays its first page. The option to import and deploy to the default broker is preselected.

c. Click **Next** and follow the guidance provided by the wizard to navigate through its pages.

   The wizard displays a progress bar that shows which task it is currently performing. It also reports on all the actions that it takes by writing progress information into a scrollable text window.

   You can copy and paste some or all of the information that is reported in this text window. This information is also written to the following log file:

   Linux on x86 and Linux on x86-64

   ```
   user_home_dir/IBM/wmbt70/workspace/.metadata/
   samplePreparationWizard.log
   ```

   Windows

   ```
   user_home_dir\eclipse\workspace\.metadata\
   samplePreparationWizard.log
   ```

   If the wizard encounters an error in processing, it informs you of what has happened and returns any error information, for example a return code from a command. If you know why the error has occurred from the error text, and can correct the situation, you can do so now. Return to the error message display and click **Yes** to continue the wizard.

   If you do not understand the error, and do not know how to fix it, click **No**. If the wizard can, it rolls back all the actions that it has taken so far, so that when it completes, your system is in the same state as it was in before you started the wizard. The text window shows you exactly what the wizard has, and has not, done.

   The wizard displays information messages to show that the Pager samples and associated resources are deployed and ready to run.

d. Click **Next** when you have read the messages about the actions that have been completed by the wizard. The confirmation page is displayed.

e. Click **Finish** to close the wizard. The "Pager samples" page (from which you launched the wizard) is redisplayed.

f. Click **Run the Pager samples**. On the page that opens, click **How to use the applications** to read about the Text Messenger and Surf report publisher applications. When you understand what the applications do, and how to use them, click the icon that represents the application that you want to run.

   If you want more detailed information about the contents of these applications, and how the message flows work, click **Find out what the Pager samples do**.

g. When you have sent and received messages successfully, you have verified that your installation is complete. You can now close your Pager applications and the Samples Gallery.

5. (Optional) You can start the Samples Preparation wizard to create the resources and start other supplied sample programs. Click **File** > **New** > **Other** > **Message Broker - Default Configuration and Samples** in the WebSphere Message Broker Toolkit, and select **Prepare the Samples**. The Samples Preparation wizard opens, and lists other samples that are available.

   You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

6. To remove the sample or samples when you have finished with them, run the Samples Preparation wizard again and remove the samples that you have added. This action removes the samples from the broker, and removes the sample resources from your workspace.

   When you have completed your verification tests, run the Default Configuration wizard to remove all the default resources. Use the same workspace and the same user ID that you used to create the resources. To start the wizard from the WebSphere Message Broker Toolkit, click **File** > **New** > **Other** and expand Message Broker - Default Configuration and Samples. Select **Create the Default Configuration** and click **Next**.

Results

You have completed the tutorial.

Related tasks:

"Creating a default configuration" on page 564
Use the Default Configuration wizard to create and test a basic broker configuration.

"Dealing with problems" on page 3363
Learn how to resolve some of the typical problems that can occur.

Verifying your installation using the WebSphere Message Broker Explorer

Use the instructions in this tutorial to verify your installation of WebSphere Message Broker and learn how to deploy broker archive files with the WebSphere Message Broker Explorer.

Before you begin

You must have installed the WebSphere Message Broker Explorer to run this tutorial. If you are using the WebSphere Message Broker Toolkit you can verify your installation using one of the samples. For a list of samples, see "Samples" on page 98.

About this task

To verify your installation using the WebSphere Message Broker Explorer you can complete the following tasks:

- Create a Default Configuration
- Deploy a broker archive file
- Check the results of a deployment

- Test the message flow

Use the following instructions to complete these tasks:

Procedure

1. To use the WebSphere Message Broker Explorer you must start the WebSphere MQ Explorer. To start the WebSphere MQ Explorer, complete one of the following steps:
   - On Windows:

     Click **Start** > **All Programs** > **IBM WebSphere Message Broker 7.0** > **IBM WebSphere Message Broker Explorer**, or open a command line in which the **mqsiprofile** command has not been run and enter the `strmqcfg` command.
   - On Linux:

     Enter the `strmqcfg` command on a command line in which the **mqsiprofile** command has not been run, or run `/usr/bin/strmqcfg`.
2. Create the Default Configuration:
   - Start the Create the Default Configuration wizard using the following link:

     Start the Default Configuration wizard

     You can use this link only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the WebSphere Message Broker Explorer.
   - Alternatively, you can use the following instructions to start the Create the Default Configuration wizard:
     a. In the **WebSphere MQ Explorer - Navigator** view, click the **Brokers** folder.
     b. In the **WebSphere MQ Explorer - Content** view, click the **Create the Default Configuration** button. The Create the Default Configuration wizard is displayed.

   a. You can click **Cancel** at any time to cancel the creation of the default configuration.
   b. The Default Configuration Summary page lists the resources created. Click **Next** to continue.
   c. Click **Finish**.

   The default broker is created and started. The broker is also added to the **WebSphere MQ Explorer - Navigator** view.
3. Deploy the broker archive file to the broker:
   a. Expand the **Broker Archive Files** folder.
   b. Expand the **Getting Started** folder.
   c. Right-click the `pager.bar` file, and click **Deploy File**.
   d. Select the MB7BROKER broker, and click **Finish**. The broker archive file is deployed to the broker.
4. Check the results of the deployment:
   a. In the **WebSphere MQ Explorer - Navigator** view, expand the **Brokers**.
   b. Click MB7BROKER to select the broker.
   c. View messages from the broker in the Administration Log view. If the deployment was successful, message BIP2881I is displayed in the Administration Log view. You can also see the TextMessenger message flow and the PagerMessageSets message set deployed to the Default execution group under the MB7BROKER broker.

5. Test the message flow:
   a. In the **WebSphere MQ Explorer - Navigator** view, expand MB7QMGR in the **Queue Managers** folder.
   b. Right-click the **Queues** folder, and click **New** > **Local Queue**.
   c. Enter TEXTMESSENGER as the name for the queue, and click **Finish**. This queue is the input queue for the Pager message flow.
   d. Click **OK**.
   e. Right-click the **Queues** folder, and click **New** > **Local Queue**.
   f. Enter TEXTMESSENGER_FAIL as the name for the queue, and click **Finish**. This queue is the failure queue for the Pager message flow.
   g. Click **OK**.
   h. Right-click the **Queues** folder, and click **New** > **Local Queue**.
   i. Enter PAGER as the name for the queue, and click **Finish**. This queue is the output queue for the Pager message flow.
   j. Click **OK**.
   k. In the **WebSphere MQ Explorer - Content** view right-click the TEXTMESSENGER queue, and click **Put Test Message**. The Put test message dialog is displayed.
   l. Enter the following message in the message data field, and click **Put message** to put the test message to the input queue for the Pager message flow:

      ```
      <Pager><text>This is my message to the pager.</text></Pager>
      ```

      The test message is passed through the message flow, transformed, and the output message is put to the Pager queue.
   m. Click **Close** to close the Put test message dialog.
   n. Right-click the Pager queue, and click **Browse Messages**. The Message browser dialog opens and displays any messages on the queue. If the test is successful, the queue shows a message like the following:

      ```
      <?xml version="1.0"?><!DOCTYPE Pager>
      <!--MRM Generated XML Output on :Sat Jun 20 10:38:56 2009-->
      <Pager><Text> Powered by IBM.</Text></Pager>
      ```
6. Click **Close** to close the Message browser dialog.

Results

You have completed the tutorial.

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

Related tasks:

"Creating a default configuration" on page 564
Use the Default Configuration wizard to create and test a basic broker configuration.

Chapter 8, "Administering brokers and broker resources," on page 899
Administering brokers and associated broker resources includes the tasks that you perform frequently to activate and manage those resources. Choose the method you prefer to administer your brokers and associated resources.

"Dealing with problems" on page 3363
Learn how to resolve some of the typical problems that can occur.

Checking the broker operation mode and function level

You must ensure that your production brokers conform to the terms of your license. You can also change the function level to enable the use of nodes that are supplied in the latest fix pack.

About this task

The following tasks are described in this topic:

- Configure your brokers to conform to your license
- Change the function levels of your brokers

For a full description of the broker operation mode, and behavior associated with each mode, see "Operation modes" on page 48.

Configuring your brokers to conform to your license

Procedure

- **Upgrading from the Trial Edition**

  If you installed WebSphere Message Broker Trial Edition, and have now purchased the product, you can keep the components and all the associated resources that you have already created and configured. You do not have to uninstall the Trial Edition and reinstall the purchased packages. However, if you do not reinstall, the operation mode of your existing brokers, and the default operation mode of all new brokers that you create, has the value `trial`. To change the value, complete the following steps.

  – If you intend to keep the brokers that you created for your trial domain for further development and unit test by your developers, change the operation mode of each broker to `enterprise`, as described in "Changing the operation mode of your broker" on page 655.

    After this change, your brokers are no longer restricted by the trial period, but check the license agreement file to ensure that your configuration conforms to any restrictions for development and unit test. Development and unit test conditions are described in "License requirements" on page 3606.

  – If you intend to use the brokers that you created for your trial domain for production purposes, change the operation mode to conform to the license that you have purchased:

    - If you have purchased Remote Adapter Deployment, change the operation mode to `adapter`.
    - If you have purchased Starter Edition, change the operation mode to `starter`.
    - If you have purchased the full (unrestricted) license, change the operation mode to `enterprise`.
    - If you have purchased Entry Edition, change the operation mode to `entry`.

  If you do reinstall the Broker component with the new physical or electronic packages for your purchased product, the default operation mode of all new brokers that you create has the value `enterprise`. If you have purchased Starter Edition, Remote Adapter Deployment, or Entry Edition, you must always change the mode to conform to the license that you have purchased.

  If you also installed a trial version of WebSphere MQ, you are now entitled to install the restricted license version that is supplied as part of WebSphere Message Broker. If the supplied products meet your requirements and you intend to use them only with WebSphere Message Broker, you do not need to make further purchases.

To upgrade your trial version of WebSphere MQ, complete the following steps.

1. Uninstall the trial version.
2. Install a licensed version of WebSphere MQ:
   - A version of WebSphere MQ is included with WebSphere Message Broker for use only with WebSphere Message Broker. Use either the supplied media or the images that you have downloaded to install this version. If you are installing on Windows, you can use the Launchpad to install this product.
   - To use WebSphere MQ for applications that are not related to your use of WebSphere Message Broker, you must purchase a separate license.

     To use additions or enhancements that are included in WebSphere MQ, you can also purchase a separate license.

- **Remote Adapter Deployment, Starter Edition, or Entry Edition**

  If you have purchased WebSphere Message Broker Remote Adapter Deployment, WebSphere Message Broker Starter Edition, or WebSphere Message Broker Entry Edition and installed components from the full runtime package, read the following information.

  - All brokers that you have created (for example, by completing the verification procedures) have an operation mode set to `enterprise`, which is the default setting for this installation. You can keep these brokers for further development and unit test, subject to any restrictions that apply for unit test environments, as indicated in your license. Development and unit test conditions are described in "License requirements" on page 3606.
  - If you intend to use these brokers for production purposes, follow the instructions in "Changing the operation mode of your broker" on page 655, to conform to the license that you have purchased.
    - If you have purchased Remote Adapter Deployment, change the operation mode to `adapter`.
    - If you have purchased Starter Edition, change the operation mode to `starter`.
    - If you have purchased Entry Edition, change the operation mode to `entry`.
  - When you create new brokers, you must set the operation mode to conform to the license that you have purchased, because by default the operation mode is set to the value `enterprise`. You can use the **-o** parameter on the **mqsicreatebroker** command to override this default value by specifying an alternative value of `adapter`, `starter`, or `entry`. Alternatively, you can change the mode by using the **mqsimode** command for the created broker.

- **Full (unrestricted) license for WebSphere Message Broker**

  If you have purchased a full (unrestricted) license and installed components from the full runtime package, all brokers that you create have an operation mode set to the default value of `enterprise`, which is the correct setting for your license.

  Whenever you create a new broker, on existing installations or on different installations or computers, the operation mode is set to `enterprise`, and you do not need to change this value.

  You can continue to work with all the components and associated resources that you have already created. For example, after you have completed verification, you might keep the resources that you have created for further development and unit test (subject to any restrictions that apply for unit test environments, as indicated in your license). Development and unit test conditions are described in "License requirements" on page 3606.

Changing the function levels of your brokers

About this task

Note: Version 7.0.0.1 can be installed as a full generally available version, or as a fix pack. Version 7.0.0.1 has the function enabled for Version 7.0.0.1 regardless of the installation route chosen. Subsequent Version 7.0 fix packs use the following procedure.

When a fix pack is delivered, it might include new nodes that you can add to your message flows to provide specific functions.

The default function level of the broker is not set to a specific value; the default value is equivalent to the value `7.0.0.0`, which represents the level for Version 7.0. At this level, nodes that might be added by later fix packs are not supported by the broker.

Nodes that are added in later fix packs are available in the WebSphere Message Broker Toolkit, and you can include these nodes in a message flow. You can deploy the message flow to a broker only if you have set the function level of that broker to the value that represents the fix pack in which the nodes are first delivered.

Because you can control the function level of each broker, you can try out new nodes on test brokers without affecting the operation of your production brokers. When you are satisfied that the nodes provide the function that you require, and work as you expect, you can set the function level of other brokers in your domain, when appropriate.

To change the function level of a broker, use the **mqsichangebroker** command, specifying the **-f** flag with the appropriate value.

For more information about nodes added in fix packs, and the use of the **mqsichangebroker** command, see "**mqsichangebroker** command" on page 3723.

Related concepts:

"Operation modes" on page 48
The operation mode that you use for your broker is determined by the license that you purchase.

Related tasks:

"Changing the operation mode of your broker" on page 655
Change the operation mode in which your broker is working by using the **mqsimode** command.

"Controlling the functional level of WebSphere Message Broker" on page 51
You can control the functional level of your broker to enable new functionality added in WebSphere Message Broker fix packs.

Related reference:

"License requirements" on page 3606
Use the reference information in this section to understand license requirements.

"**mqsimode** command" on page 3899
Use the **mqsimode** command to configure and retrieve operation mode information.

Installing complementary products

WebSphere Message Broker works with several other products to provide complementary services.

About this task

If you want to use these optional services in your WebSphere Message Broker environment, refer to the following installation information:

- IBM Tivoli License Manager
- Eclipse plug-ins from another vendor
- "Publishing in a Citrix XenApp environment" on page 302

Related tasks:

"Finding the latest information" on page 232
Access the latest information for WebSphere Message Broker.

Related reference:

"Software requirements" on page 3588
View the operating system, database, and other software requirements.

"Hardware requirements" on page 3582
View the processor and related hardware requirements on all platforms.

"Installation and uninstallation interfaces" on page 3617
You can use different interfaces for installation and uninstallation of the Broker component and the WebSphere Message Broker Toolkit.

Installing Tivoli License Manager

IBM Tivoli License Manager (ITLM) enables you to monitor the use of IBM (and other) software products. WebSphere Message Broker includes support for ITLM Version 2.1.

About this task

Use ITLM to perform the following software auditing functions:

- Monitor the licenses used by different machines.
- Help keep unnecessary licenses to a minimum.
- Guard against software license compliance problems.

Ensure that you choose the correct ITLM license for the WebSphere Message Broker edition that you have purchased. See "Operation modes" on page 48.

If you are using one or more of the WebSphere Adapters with WebSphere Message Broker, you must activate ITLM to include those adapters, for example in monitoring activities. If you require this support, follow the instructions in "Activating IBM Tivoli License Manager for WebSphere Adapters" on page 2036.

To find out more about using ITLM to monitor usage of WebSphere Message Broker and other IBM products, or to purchase ITLM, see the IBM Tivoli License Manager website.

For information about installing this product, see the IBM Tivoli License Manager Information Center.

Related tasks:

"Installing complementary products" on page 300
WebSphere Message Broker works with several other products to provide complementary services.

"Activating IBM Tivoli License Manager for WebSphere Adapters" on page 2036
If you want to use IBM Tivoli License Manager (ITLM), you must activate it for each of the WebSphere Adapters.

Related reference:

"System management interfaces" on page 52
The brokers provide a service for independent system management agents.

"Software requirements" on page 3588
View the operating system, database, and other software requirements.

"Hardware requirements" on page 3582
View the processor and related hardware requirements on all platforms.

Installing plug-ins using Eclipse Update Manager

Use Eclipse Update Manager to install and update plug-ins provided by partners and other software vendors.

About this task

Do not use the Eclipse Update Manager to update or install plug-ins that are managed by IBM Installation Manager. Always use Installation Manager to work with products that are installed with Installation Manager. For other products, use Eclipse Update Manager, but follow the restrictions and guidance provided:

- Do not delete the `platform.xml` file to prompt Eclipse to regenerate the file. The regenerated contents will be incomplete and therefore invalid.
- Always save a backup copy of your `platform.xml` file before you make any changes to your configuration, for example by using Eclipse Update Manager to install or update plug-ins that are provided by another software vendor.
- When you use Eclipse Update Manager (graphical interface or command line), always specify the package group directory as the destination; never specify the shared resource directory. The shared resource directory is managed by Installation Manager and is reserved for its use.
- Do not use Eclipse Update Manager to search for updates for features that you have installed using Installation Manager.
- Do not select **Clean up all configuration errors** in the Eclipse Update Manager interface; this option might lose information that is maintained by Installation Manager.

Related tasks:

"Installing complementary products" on page 300
WebSphere Message Broker works with several other products to provide complementary services.

Related reference:

"System management interfaces" on page 52
The brokers provide a service for independent system management agents.

"Software requirements" on page 3588
View the operating system, database, and other software requirements.

"Hardware requirements" on page 3582
View the processor and related hardware requirements on all platforms.

Publishing in a Citrix XenApp environment

Supply application location and user details to Citrix to publish a WebSphere Message Broker command console or WebSphere Message Broker Toolkit.

About this task

Read the Citrix documentation for general information about publishing applications. Publishing does not involve any tasks within WebSphere Message Broker; you complete the following tasks in the Citrix Presentation Server console.

"Publishing a WebSphere Message Broker command console"

"Publishing a WebSphere Message Broker Toolkit"

"Publishing applications using the CMP API"

Publishing a WebSphere Message Broker command console:
About this task

The following steps show how to set the Application Location and User information to publish a WebSphere Message Broker command console.

Procedure

1. Set the **Application Type** to `Application`.
2. Set the **Command Line** to the following value: This example assumes that you have installed a 32-bit operating system.

   ```
   C:\WINDOWS\system32\cmd.exe /k title IBM WebSphere Message Brokers 7.0
        &&"install_dir\MQSI\7.0\bin\mqsiprofile.cmd"
   ```

   On Windows, you can install the Broker component in multiple locations. Each one can have a different level of service (fix pack) applied and can support a certain set of brokers. Each installation has its own command console executable. To publish command consoles corresponding to different runtime environments, modify the path to `mqsiprofile.cmd` appropriately.
3. Set the **Working Directory** to *install_dir*`\MQSI\7.0`.
4. Define the users who are authorized to use the WebSphere Message Broker command console. This must be a subset of users or groups who are authorized to use the command console locally, on the server.

   Attention: Users who have access to the WebSphere Message Broker command console through Citrix can use the console to run other commands that are unrelated to the broker.

Publishing a WebSphere Message Broker Toolkit:
About this task

To publish a WebSphere Message Broker Toolkit, set the Application Location and User information as follows:

Procedure

1. Set the **Application Type** to `Application`.
2. Set the **Command Line** to the package group directory for WebSphere Message Broker Toolkit that your system uses, for example:

   ```
   E:\WMBT700\eclipse.exe
   ```
3. Set the **Working Directory** to the appropriate workspace, for example:

   ```
   E:\workspace\wmbt70\workspace1
   ```

   When the WebSphere Message Broker Toolkit is run from Citrix, a window inviting the user to select a workspace is displayed. This is the same window that you see when you run the WebSphere Message Broker Toolkit locally. At this stage you can specify a different workspace, for which you have the necessary Windows read and write file permissions on the server.
4. Define the users who are authorized to use the WebSphere Message Broker command console. This must be a subset of users or groups who are authorized to use the command console locally, on the server.

Publishing applications using the CMP API:

About this task

You can publish a Java application that uses the CMP API programming interface. The steps are similar to publishing a WebSphere Message Broker Toolkit or command console.

Example

For example, to publish the CMP API Exerciser sample, publish this file:

install_dir\sample\ConfigManagerProxy\StartConfigManagerProxyExerciser.bat

Related concepts:

"Users and Citrix"
Review the categories of user that might want to use with Citrix, and how to configure for concurrent multi-users.

Related tasks:

"Developing applications that use the Administration API" on page 956
Develop Java applications that use the Administration API (also known as the CMP API) to communicate with, deploy to, and manage brokers and their associated resources.

Related reference:

"System requirements for Citrix XenApp" on page 3605
This topic gives information about licensing issues, and the software and hardware that you need to use WebSphere Message Broker in a Citrix XenApp environment.

Users and Citrix:

Review the categories of user that might want to use with Citrix, and how to configure for concurrent multi-users.

Categories of user

The following categories of user exist:

Application developers
: These users do not need to run mqsi commands, but must be placed in an access control list (ACL) before they can use a broker. Put these users only in the user list for publishing WebSphere Message Broker Toolkit.

Administrators
: If you use Citrix to publish the WebSphere Message Broker Toolkit and the WebSphere Message Broker command console, put all administrators in the users lists for both applications. For information about the permissions that administrators need, see "Security requirements for administrative tasks" on page 3644.

You can create Windows groups that correspond to the application developer and administrator categories, put the groups in the appropriate user lists, then allocate users to the relevant Windows group. Alternatively, you can give appropriate permissions directly to the user.

Make all users members of the Remote Desktop Users group, so that they can use Terminal Server.

Attention: Users who have access to the WebSphere Message Broker command console through Citrix can use the console to run other commands that are unrelated to the broker.

For information about the rights of groups such as mqm, mqbrkrs, and Administrators, see "Security requirements for Windows systems" on page 3651.

Concurrent multi-users

Allowing multiple concurrent users to access WebSphere Message Broker by Citrix on a Windows server requires the same planning as allowing multiple concurrent users to access the WebSphere Message Broker on a Linux or UNIX server. For example:

- All application developers should have their own execution group, so that they can deploy message flows and start and stop execution groups without conflicting with each other during development and testing. However, an exection group might be administered by more than one administrator. In these cases, administrators should coordinate their activities so that they do not conflict with each other. For example, when one administrator is starting an execution group, no one else should be trying to stop it.
- All users should have their own workspace. The Windows system administrator should give each user the appropriate read and write permissions for the directory that corresponds to their workspace.
- The WebSphere Message Broker Toolkit user can create the default configuration from the Samples; the product samples are deployed to the default configuration. Because you can create only one default configuration, only one user should use it at one time.

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Related tasks:

"Publishing in a Citrix XenApp environment" on page 302
Supply application location and user details to Citrix to publish a WebSphere Message Broker command console or WebSphere Message Broker Toolkit.

"Creating user IDs" on page 498
When you plan the administration of your broker configuration, you might have to define one or more user IDs for the tasks associated with particular roles.

Related reference:

"System requirements for Citrix XenApp" on page 3605
This topic gives information about licensing issues, and the software and hardware that you need to use WebSphere Message Broker in a Citrix XenApp environment.

"Security requirements for Windows systems" on page 3651
Security requirements depend on the administrative task that you want to perform.

Setting up a command environment

After you have installed the product on one of the distributed systems, you must initialize the environment before you can use a runtime component or command.

About this task

Also complete this task if you have migrated to WebSphere Message Broker Version 7.0 from an earlier version. A profile is provided to help you set up the environment.

If appropriate, you can extend the initialization performed by this profile; for example, for user databases, or for other products that you want to use within the broker.

Ensure that you use this environment each time you run an administrative command, or start a broker.

For information about setting up your command and runtime environment on a Windows system, see "Command environment: Windows systems."

For information on setting up your command and runtime environment on Linux and UNIX systems, see "Command environment: Linux and UNIX systems" on page 310.

For information about setting up your command and runtime environment for execution groups on Linux and UNIX systems, see "Execution group-specific command environment: Linux and UNIX systems" on page 312.

For information about setting up your command and runtime environment for execution groups on Windows systems, see "Execution group-specific command environment: Windows systems" on page 309.

Related tasks:

"Migrating from Version 6.1 products" on page 163
Migrate your components and resources to WebSphere Message Broker Version 7.0.

"Migrating from Version 6.0 products" on page 183
Migrate your components and resources to WebSphere Message Broker Version 7.0.

"Enabling ODBC connections to the databases" on page 668
Set up the resources and environment that the broker requires for Open Database Connectivity (ODBC) connections to user databases on distributed systems.

"Command environment: Windows systems"
Set up the Windows environment to run WebSphere Message Broker commands.

"Command environment: Linux and UNIX systems" on page 310
Set up the Linux or UNIX environment to run WebSphere Message Broker commands.

Related reference:

"Runtime commands" on page 3715
The topics in this section describe the WebSphere Message Broker runtime commands.

Command environment: Windows systems

Set up the Windows environment to run WebSphere Message Broker commands.

About this task

On Windows systems, the components run as services, therefore they do not inherit the environment that is set for the command prompt from which you start them. The components run the **mqsiprofile** command when they start, which completes the environment initialization.

The **mqsiprofile** command is located in the directory *install_dir*\bin.

You must not change the location of the **mqsiprofile** command, or make user modifications to the command, because it might be replaced if you install service, or an update, to the product.

If you need to run your own settings, add one or more command files called *your_file_name*.cmd to the directory *work_path*\common\profiles, where *work_path* identifies the broker's working directory.

The default working directory is %ALLUSERSPROFILE%\Application Data\IBM\MQSI where %ALLUSERSPROFILE% is the environment variable that defines the system working directory. The default directory depends on the operating system:

- On Windows XP and Windows Server 2003: C:\Documents and Settings\All Users\Application Data\IBM\MQSI
- On Windows Vista and Windows Server 2008: C:\ProgramData\IBM\MQSI

The actual value might be different on your computer.

If you are unsure of the broker's work path, enter the following command in a command console:

```
echo %MQSI_WORKPATH%
```

When **mqsiprofile** runs, it automatically calls any additional user-written scripts in this directory.

A typical reason for adding scripts is "Running database setup scripts" on page 308.

When you have configured any additional setup, you can use one of the following methods to initialize the runtime environment for components and commands:

Procedure

1. If you have only one installation of the runtime product, open a command console by clicking **Start** > **All Programs** > **IBM WebSphere Message Broker 7.0** > **Command Console**. The **mqsiprofile** command initializes the environment and invokes any additional scripts in the common\profiles directory.
2. If you have one or more installations of the runtime product, open a command prompt window. Locate and run the mqsiprofile.cmd script in the directory in which you installed the appropriate product. The **mqsiprofile** command initializes the environment and invokes any additional scripts in the common\profiles directory.

What to do next

Check whether the following conditions apply to your environment:

- If you have a previous version of the product on this system, ensure that you run the correct profile before using Version 7.0. The **mqsiprofile** command places the Version 7.0 commands and libraries at the front of your search path, and invokes any user profiles that you have supplied which can override any combination of PATH, CLASSPATH, or library PATH.
- If you use the same user ID, and you run multiple profiles (from multiple different installations or versions), you might get unexpected results. Log off and log on again before you run the specific profile that you require.

Running database setup scripts:
About this task

A broker might require access to user databases from deployed message flows.

When you install a database product on Windows, the relevant settings are typically made to the system environment. However, some database managers provide a profile to perform this setup, or provide details of actions that you must take in their documentation. Always check the database product documentation for environment setup details; the information provided here is for general guidance only.

If a profile is provided for the database that you are using, complete the following steps:

Procedure

1. If you can update the profile to provide permanent values for the details that are required (for example, the database server name or the installation directory):
   a. Complete the changes to the database profile.
   b. Copy the profile file to the directory *work_path*\common\profiles.
2. If you cannot update the profile permanently, but need to make changes each time, you must run it independently of the **mqsiprofile** command before you start the component.

What to do next

When your environment has been set up, see "Working with databases" on page 2109 for information about setting up your databases for use with the broker.

Related tasks:

"Working with databases" on page 2109
Create and configure databases to use with your message flow applications.

"Setting up a command environment" on page 213
After you have installed the product on one of the distributed systems, you must initialize the environment before you can use a runtime component or command.

"Changing the location of the work path" on page 1011
The work path directory is the location where a component stores internal data, such as installation logs, component details, and trace output. The shared-classes directory is also located in the work path directory and is used for deployed Java code. If the work path directory does not have enough capacity, redirect the directory to another file system that has enough capacity.

"Command environment: Linux and UNIX systems" on page 310
Set up the Linux or UNIX environment to run WebSphere Message Broker commands.

Related reference:

"Additional software requirements" on page 3598
WebSphere Message Broker requires additional software products to run successfully.

"Runtime commands" on page 3715
The topics in this section describe the WebSphere Message Broker runtime commands.

Execution group-specific command environment: Windows systems

Extend or change the Windows environment that is used when running a specific WebSphere Message Broker.

About this task

To enable function that becomes available in WebSphere Message Broker fix packs, use the **-f** parameter on the **mqsichangebroker** command. For more information, see "**mqsichangebroker** command" on page 3723.

To run your own additional environment settings for a broker, add a script (or scripts) in the *%MQSI_WORKPATH%\config\<my_broker_name>\<my_eg_label>\profiles* directory.

Ensure that the broker name, and any execution groups that you create, contain only characters that are valid on your file system. You might also have to create the required directory structure.

All files placed into the *%MQSI_WORKPATH%\config\<my_broker_name>\<my_eg_label>\profiles* directory that have a `.cmd` extension, are processed when the execution group *<my_eg_label>* on broker *<my_broker_name>* starts.

Scripts in the *%MQSI_WORKPATH%\config\<my_broker_name>\<my_eg_label>\profiles* directory are run after the **mqsiprofile** script, and any scripts placed in the *common\profiles* directory have been run.

To diagnose any problems, a log file is written that lists the profile scripts that have been read, and the environment used by this broker. The log file location is under the work path of the broker and is located at *%MQSI_WORKPATH%\common\log\<my_broker_name>.<my_eg_label>.profilelog*

Execution group profile scripts are not removed automatically when an execution group is deleted. If you no longer need the profile files that you have created, delete them manually.

Related tasks:

"Setting up a command environment" on page 213
After you have installed the product on one of the distributed systems, you must initialize the environment before you can use a runtime component or command.

"Command environment: Windows systems" on page 306
Set up the Windows environment to run WebSphere Message Broker commands.

"Controlling the functional level of WebSphere Message Broker" on page 51
You can control the functional level of your broker to enable new functionality added in WebSphere Message Broker fix packs.

Command environment: Linux and UNIX systems

Set up the Linux or UNIX environment to run WebSphere Message Broker commands.

About this task

When you start a runtime component on Linux and UNIX systems, it inherits the environment from where you issue the **mqsistart** command.

You must therefore initialize the environment before you start a component; the command **mqsiprofile** located in the directory *install_dir*/bin, performs this initialization. If you are starting a broker, you might also need to initialize the environment for any databases that are accessed by the broker.

You must not change the location of the **mqsiprofile** command, or make user modifications to the command, because it might be replaced if you install service, or an update, to the product.

If you want to run your own additional environment settings, add a script called *your_file_name*.sh to the brokers *work_path* directory which contains the:

```
/common/profiles
```

subdirectory.

When you run **mqsiprofile** again, the command automatically calls the additional user-written scripts in this location.

You must log out and log back in, to pick up the new files in the /common/profiles directory, before you run **mqsiprofile** again

The newly added script is not picked up if you run a **mqsistart** command from an existing initialized command shell.

work_path identifies the working directory defined for the broker; if you are unsure of the work path, enter the following command:

```
echo $MQSI_WORKPATH
```

A typical reason for adding scripts is "Running database setup scripts" on page 311.

When you have configured any additional setup, you can initialize the runtime environment for components and commands:

Procedure

1. Issue the **mqsiprofile** command:

   ```
   . install_dir/bin/mqsiprofile
   ```

   You must include the period and space preceding the location for this invocation to work correctly. Add this command to your login profile if you want it to be run at the start of every session.

   This command accesses additional scripts that you have copied to the common/profiles directory, therefore the environment is initialized for runtime components and other resources such as databases.
2. Start the components that you want to run on this system by means of the **mqsistart** command.

What to do next

Check if the following conditions apply to your environment:

- If you have a previous version of the product on the system, ensure that you run the correct profile before using WebSphere Message Broker Version 7.0. The **mqsiprofile** command places the Version 7.0 commands and libraries at the front of your search path, and calls any user profiles that you have supplied that can override any combination of PATH, CLASSPATH, or library PATH.
- If you use the same user ID, and you run multiple profiles (from multiple different installations or versions), you might get unexpected results. Log off and log on again before you run the specific profile that you require.
- ODBC settings on Linux and UNIX systems are found in a text file defined by the ODBCUOINI environment variable. Set ODBCUOINI to point to a copy of the sample file install_dir/ODBC/V6.0/odbc.ini .

  You can check that your ODBC environment is configured correctly by running the **mqsicvp** command. This command also validates the connection to all data sources that are listed in the odbc.ini file that have been associated with a broker by using the **mqsisetdbparms** command. For more information, see "**mqsicvp** command" on page 3857.

Running database setup scripts:
About this task

A broker might require access to user databases from deployed message flows.

When you install a database product on Linux and UNIX systems, some database managers provide a profile to perform the environment setup that the database requires, or provide details of actions that you must take in their documentation. Always check the database product documentation for environment setup details; the information provided here is for general guidance only and might not be complete.

Procedure

1. If you can update the profile to provide permanent values for the details that are required (for example, the database server name or the installation directory):
   a. Complete the changes to the profile.
   b. Copy the profile file to the directory *work_path*\common\profiles.
2. If you cannot update the profile permanently, but have to change it each time, you must run it independently of the **mqsiprofile** command:
   a. Run the appropriate profiles to initialize the environment for the database:
      - For DB2, issue the following command for the relevant DB2 instance:

        . *db2_instance_directory*/sqllib/db2profile
      - For Oracle, export ORACLE_HOME and ORACLE_SID, then issue the following command:

        . ${ORACLE_HOME}/bin/oraenv
      - For Sybase, issue the following command, specifying your installation directory:

        . *Sybase_installation_directory*/SYBASE.sh
      - For Informix, check the documentation for the client on your broker system for details of the actions that you must take. For example, you might have to specify the following environment variables:

```
export INFORMIXDIR=/installation_directory_of_informix_client_software
export PATH=${INFORMIXDIR}/bin:${PATH}
export INFORMIXSERVER=server_name
export INFORMIXSQLHOSTS=${INFORMIXDIR}/etc/sqlhosts
export TERMCAP=${INFORMIXDIR}/etc/termcap
export TERM=vt100
export LIBPATH=${INFORMIXDIR}/lib:${INFORMIXDIR}/lib/esql:
               ${INFORMIXDIR}/lib/cli:$LIBPATH
```

where *server_name* is defined in the file `sqlhosts` (the required value is typically the machine name), and the location of the file `sqlhosts` is set up as part of the installation process.

To configure your system to run this setup at the start of every session, add these statements to the login profile of the user that is going to run the broker.

What to do next

When your environment has been set up, see "Working with databases" on page 2109 for information about setting up your databases for use with the broker.

Related tasks:

"Working with databases" on page 2109
Create and configure databases to use with your message flow applications.

"Setting up a command environment" on page 213
After you have installed the product on one of the distributed systems, you must initialize the environment before you can use a runtime component or command.

"Changing the location of the work path" on page 1011
The work path directory is the location where a component stores internal data, such as installation logs, component details, and trace output. The shared-classes directory is also located in the work path directory and is used for deployed Java code. If the work path directory does not have enough capacity, redirect the directory to another file system that has enough capacity.

"Command environment: Windows systems" on page 306
Set up the Windows environment to run WebSphere Message Broker commands.

Related reference:

"Additional software requirements" on page 3598
WebSphere Message Broker requires additional software products to run successfully.

"Runtime commands" on page 3715
The topics in this section describe the WebSphere Message Broker runtime commands.

Execution group-specific command environment: Linux and UNIX systems

Extend or change the Linux or UNIX environment used when running a specific execution group.

About this task

To enable function that becomes available in WebSphere Message Broker fix packs, use the **-f** parameter on the **mqsichangebroker** command. For more information, see "**mqsichangebroker** command" on page 3723.

If you want to run your own additional environment settings, add a script (or scripts) in the *$MQSI_WORKPATH/config/<my_broker_name>/<my_eg_label>/profiles* directory.

Ensure that the broker name, and any execution groups that you create, contain only characters that are valid on your file system. You might also have to create the required directory structure.

All files placed into the *$MQSI_WORKPATH/config/<my_broker_name>/<my_eg_label>/profiles* directory that have a `.sh` extension, are processed when the execution group *<my_eg_label>* on broker *<my_broker_name>* starts.

Scripts in the *$MQSI_WORKPATH/config/<my_broker_name>/<my_eg_label>/profiles* directory are run after the **mqsiprofile** script, and any scripts placed in the *common/profiles* directory have been run.

To diagnose any problems, a log file is written that lists the profile scripts that have been read, and the environment used by this execution group. The log file location is under the work path of the broker and is located at:

```
$MQSI_WORKPATH/common/log/<my_broker_name>.<my_eg_uuid>.profilelog
```

Execution group profile scripts are not removed automatically when an execution group is deleted. If you no longer need the profile files that you have created, deleted them manually.

Related tasks:

"Setting up a command environment" on page 213
After you have installed the product on one of the distributed systems, you must initialize the environment before you can use a runtime component or command.

"Command environment: Linux and UNIX systems" on page 310
Set up the Linux or UNIX environment to run WebSphere Message Broker commands.

"Controlling the functional level of WebSphere Message Broker" on page 51
You can control the functional level of your broker to enable new functionality added in WebSphere Message Broker fix packs.

Applying service

Apply service to the WebSphere Message Broker Toolkit, the WebSphere Message Broker Explorer, and the Broker component.

Service updates and other fixes are delivered occasionally in the form of Program Temporary Fixes (PTFs) or fix packs. You can download these fixes from the web and update your existing installations. Check regularly for updates and recommended fixes on the WebSphere Message Broker support web page.

For information on obtaining the function level that your enterprise is using, see the **mqsichangebroker** command.

For information about installation directories for fix packs, and installing more than one fix pack at the same or different levels on a single computer, see "Coexistence and migration" on page 239.

- "Applying service to the Broker component" on page 314
- "Applying service to the WebSphere Message Broker Toolkit" on page 325
- "Applying service to the WebSphere Message Broker Explorer" on page 329

Related tasks:

"Controlling the functional level of WebSphere Message Broker" on page 51
You can control the functional level of your broker to enable new functionality added in WebSphere Message Broker fix packs.

Chapter 7, “Configuring brokers for test and production environments,” on page 579
Create one or more brokers on one or more computers, and configure them on your test and production systems to process messages that contain your business data.

“Uninstalling service for the Broker component” on page 338
On some platforms, you can remove fixes that you have applied.

“Uninstalling service from the WebSphere Message Broker Toolkit” on page 344
Remove maintenance or fixes from the WebSphere Message Broker Toolkit.

“Uninstalling service from the WebSphere Message Broker Explorer” on page 350
You must reinstall the WebSphere Message Broker Explorer to revert to a previous service level.

Related reference:

“**mqsichangebroker** command” on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

Applying service to the Broker component

Apply maintenance updates and program fixes to the Broker component.

Before you begin

Before you apply service, check that you have backed up all associated resources. You might also choose to back up installation and work path directories. For details of this task, see “Backing up resources” on page 1013.

About this task

You can download maintenance updates for all components of WebSphere Message Broker from a website, in the form of a Program Temporary Fix (PTF), also known as a fix pack. Fix packs are cumulative; therefore, if multiple fix packs are available, you do not have to install a previous fix pack before you install the latest available. However, you must first install the GA (general availability) code to ensure that you comply with your license agreement. You can find the latest information about current fixes by following the link **Recommended fixes** in the Download section of the WebSphere Message Broker support web page.

In some circumstances, fix packs are also provided as a media refresh; CDs and DVDs are re-created, and electronic images on IBM Passport Advantage are replaced. Check with your IBM representative if you want a media refresh.

If you have installed the Trial Edition, you can apply service to this installation, if you are within your 90 day trial period.

Fix packs for WebSphere Message Broker are installed by using the same technology as the GA release. Therefore product files are consistently tracked and updated appropriately. For information about the default directories for a typical installation, see “Coexistence and migration” on page 239.

If you have applied one or more interim fixes to your existing installation, contact your IBM Service representative for instructions and possible updated interim fixes for the fix pack level that you are installing.

Because you can install more than one copy of the Broker component on a single computer, you can choose how to apply service:

1. Apply service to an existing installation.

   When you apply the fix pack, the new level of the product overwrites the existing level. The installer prohibits you from installing a previous modification or fix pack over a more recent one. You must stop all brokers before you apply service. All the brokers and resources that you have defined are retained. When the fix pack is successfully installed, restart the brokers.
2. Install the fix pack level of the product at a new location.

   Fix packs are supplied as complete installations. You can install the product in a new location on your computer by using the fix pack packages. You can install only within the terms of your license:

   - You must have installed a GA level of the product on this computer.
   - You must conform to the license restrictions for the number of installations on a single computer; your license might permit you to install more than one copy of the product, but allow you to run only one installation at a time.

   This option provides several advantages:

   - You do not have to stop brokers to complete the installation.
   - You can choose which broker runs at which service level by running the appropriate **mqsiprofile** before you start the component.
   - You can back out a broker to the previous service level by using the **mqsiprofile** from the older code level, and restarting the broker.

   If you already have more than one installation on the computer, read the additional information in "Applying service on computers with multiple installations" on page 316.

Service is not affected by the operation mode in which you broker is working.

What to do next

If you want to remove service that you have applied, see "Uninstalling service for the Broker component" on page 338.

Downloading a fix pack:
About this task

Download the latest fix pack from the WebSphere Message Broker support web page:

Procedure

1. Click **Downloads** to open the page of available download packages.
2. Click **Recommended fixes**, then click the appropriate product and version. Available fixes are displayed.
3. Click the fix that you want to download. Details about the fix, and instructions for download and installation, are displayed.
4. Click **Problems Fixed** to find out what PMRs, APARs, and defects have been fixed in the fix pack.
5. Select your chosen operating system and click **FC** in the **Download package** section to start the download process.
6. If you haven't already signed in, sign in with your IBM ID and password and click Submit. If you do not have an ID, you can register on this page.
7. Select the fixes you want to download.
8. Select the download method to be used to download fixes.

9. Accept the download terms and conditions.
10. Click **Download now**.
11. When the download completes, follow the instructions to apply the maintenance updates for this fix.
    a. If you are applying service to an existing installation, the installation wizard uses this current installation directory as the default directory for typical installations in graphical, console, and silent modes.

       This location might therefore be different from the default directory that is set for an initial typical installation on a computer on which the product has not been installed. The default installation directories are described in "Coexistence and migration" on page 239, "Uninstalling the Broker component by using the silent interface" on page 336, or "Uninstalling the Broker component by using the graphical interface" on page 333. If you have previously installed components into non-default locations, check carefully that you are installing the fix pack into the required location.
       - Windows
       - Linux and UNIX
       - z/OS

    b. If you are applying service by installing an additional copy of the product, follow the instructions provided in the "Installation Guide" on page 233, specifying a new installation location.

Applying service on computers with multiple installations:
About this task

If you have installed the Broker component more than once on a single computer, you can choose to apply service to one or more of these installations in any order. You might therefore choose to install a service level on one instance initially, and complete some tests, before you apply the fixes to additional installations.

You must ensure that you comply with the terms of your license at all circumstances.

Distributed systems

You specify which installation you want a broker to work with by running the **mqsiprofile** command that is associated with that installation. Any resources that you have defined are associated with the installation for which you have run **mqsiprofile**. When the **mqsiprofile** command has completed, restart the broker.

If you decide to return to a previous level, stop the broker, run the **mqsiprofile** command that is associated with the installation at the previous service level, and restart the broker.

For further details about working with multiple installations, see "Command environment: Windows systems" on page 306 and "Command environment: Linux and UNIX systems" on page 310.

z/OS systems

You can create component profiles to work with one of multiple installations. Details are provided in "Installing service on z/OS" on page 322.

For further information about components, see "z/OS customization overview" on page 592.

If you have a single installation on a computer, and you want to apply service, you must stop all brokers, apply service, and restart the brokers. The service is applied to the product code, and you can start the brokers immediately to run against the new service level.

Related concepts:

"Operation modes" on page 48
The operation mode that you use for your broker is determined by the license that you purchase.

Related tasks:

"Finding the latest information" on page 232
Access the latest information for WebSphere Message Broker.

"Uninstalling the Broker component" on page 332
You can uninstall the Broker component on distributed systems in a number of ways.

"Uninstalling service for the Broker component" on page 338
On some platforms, you can remove fixes that you have applied.

Related reference:

"Installation Guide" on page 233
Installation information for WebSphere Message Broker is provided in the *Installation Guide* that is supplied as a PDF file with your product package.

"Installation and uninstallation interfaces" on page 3617
You can use different interfaces for installation and uninstallation of the Broker component and the WebSphere Message Broker Toolkit.

"Installation and uninstallation authorization" on page 3628
Check the authorization requirements for the user ID that you use to install or uninstall the Broker component and the WebSphere Message Broker Toolkit.

Installing service on Windows:

Apply maintenance updates and program fixes to the broker component.

Before you begin

Before you start:

- Download the required update package for the WebSphere Message Broker component, as described in "Applying service to the Broker component" on page 314.
- Check that your user ID has the correct authority to complete this task; see "Installation and uninstallation authorization" on page 3628 for details.

About this task

Fix packs are supplied as self-extracting executable programs. Complete the following steps to install the updates that you have downloaded:

Procedure

1. Check that your user ID has the correct authority to install service for the broker component. The requirements are defined in "Installation and uninstallation authorization" on page 3628.
2. Ensure that all WebSphere Message Broker function is stopped:

a. Stop all brokers on this computer by using the "**mqsistop** command" on page 3972, or by stopping the Windows services for these components from **Start** > **Settings** > **Control Panel** > **Administrative Tools** > **Services**.
b. Close all instances of the WebSphere Message Broker Explorer.
c. Close all instances of the WebSphere Message Broker Toolkit.
d. Check that all files associated with WebSphere Message Broker are closed.

3. Close all Windows programs before applying maintenance to the WebSphere Message Broker component to ensure that data is not lost.
4. Run the fix pack self-extracting program from either Windows Explorer or the command line.

   The program is referred to as *FixPackLauncher* and its name is in the form *V.R.M-prod-platform-fixpack*.exe. Where *V.R.M-prod* represents the version, release, and modification level of the product, *-platform* identifies the platform, and *-fixpack* identifies the fix pack. For example, the file name 7.0.0-WS-MB-WINIA32-FP0001.exe identifies the file for Fix Pack 1 on Version 7.0.0 of WebSphere Message Broker on Windows.

   - If you start this program from Windows Explorer, or you start it from the command line with no options, the *FixPackLauncher* runs with default options. It extracts the updates from the fix pack file, and starts the graphical interface of the WebSphere Message Broker installation wizard.
   - If you run the wizard from the command line with options, you can choose the interface that you want to use, and how the WebSphere Message Broker installation wizard runs.

     For further details about how you can install, and other options that you can specify, see "Installation and uninstallation interfaces" on page 3617 and "How to install and uninstall the Broker component" on page 3618.

     If you start this wizard from a directory other than the one in which the file is saved, you must include the fully qualified path of the file and its name.

     Graphical interface

     Use the following command format:

     ```
     FixPackLauncher -a -gui
     ```

     -gui specifies that you want to use the graphical interface of the installation program. The graphical interface is used by default if you specify only *FixPackLauncher*.

     For example, you can use either of the following commands:

     ```
     7.0.0-WS-MB-WINIA32-FP0001.exe
     7.0.0-WS-MB-WINIA32-FP0001.exe -a -gui
     ```

     Console interface

     Use the following command format:

     ```
     FixPackLauncher -a -console
     ```

     -console specifies that you want to use the console interface of the installation program. You must specify this option if you want the installation to use this interface.

     For example:

     ```
     7.0.0-WS-MB-WINIA32-FP0001.exe -a -console
     ```

     Silent interface

     Use the following command format:

```
FixPackLauncher -s -a -options "path" -silent
```

`-s` specifies that you want the extraction to use its silent interface. You must specify this option if you want the extraction to be completed without interaction.

`-a` specifies that you want to pass options to the installation program.

`-options "path"` specifies the path to the silent installation response file.

`-silent` specifies that you want to use the silent interface of the installation program. You must specify this option if you want the installation to use this interface.

If you do not specify a response file, the default options are used from the supplied response file. If you want to tailor the options to suit your installation requirements, create your own response file and specify its location on the command.

For example, to run both the extractor program and the WebSphere Message Broker installation program silently with a response file, enter the following command:

```
FixPackLauncher -s -a -options "C:\response1.txt" -silent
```

5. If you use the graphical or console interface of the WebSphere Message Broker installation program, follow the prompts given and provide any input that is required to complete the installation.
6. If you run the fix pack installation program or the WebSphere Message Broker installation program silently, check the installation log to ensure that the process was successful. File `mqsi7_install.log` is stored in your installation directory.

   If you accepted the default location during installation, this directory is as follows. The default directory includes the version and release of the product that you are installing in the format v.r (version.release):

   Linux `/opt/ibm/mqsi/v.r`

   UNIX `/opt/IBM/mqsi/v.r`

   Windows 32-bit
   `C:\Program Files\IBM\MQSI\v.r`

   Windows 64-bit
   `C:\Program Files\IBM\MQSI\v.r` for the 64-bit version of WebSphere Message Broker

   `C:\Program Files (x86)\IBM\MQSI\v.r` for the 32-bit version of WebSphere Message Broker

   These locations define the default value of *install_dir* on each platform.
7. When installation has successfully completed, review the release notes that are supplied in the directory `readmes`, and complete any manual post-installation tasks that are required.
8. When you have completed any required ODBC changes, restart the components by using the **mqsistart** command.

Related tasks:

"Applying service to the WebSphere Message Broker Toolkit" on page 325
Apply maintenance updates and program fixes to the WebSphere Message Broker

Toolkit.

"Uninstalling service for the Broker component" on page 338
On some platforms, you can remove fixes that you have applied.

"Finding the latest information" on page 232
Access the latest information for WebSphere Message Broker.

Related reference:

"How to install and uninstall the Broker component" on page 3618
You can install and uninstall the Broker component by using one of three interfaces.

"Installation and uninstallation interfaces" on page 3617
You can use different interfaces for installation and uninstallation of the Broker component and the WebSphere Message Broker Toolkit.

"Installation and uninstallation authorization" on page 3628
Check the authorization requirements for the user ID that you use to install or uninstall the Broker component and the WebSphere Message Broker Toolkit.

"Using response files with the Broker component" on page 3621
You can use a response file to define the behavior of an installation or uninstallation wizard that is running the silent interface.

"**mqsistart** command" on page 3965
Use the **mqsistart** command to start the specified broker if all initial verification tests complete successfully.

"**mqsistop** command" on page 3972
Use the **mqsistop** command to stop the specified component.

Installing service on Linux and UNIX:

Apply maintenance updates and program fixes to the Broker component.

Before you begin

Before you start:

- Download the required update package for the Broker component, as described in "Applying service to the Broker component" on page 314.
- Check that your user ID has the correct authority to complete this task; see "Installation and uninstallation authorization" on page 3628 for details.

About this task

To install the updates that you have downloaded:

Procedure

1. Check that your user ID has the correct authority to uninstall the Broker component. The requirements are defined in "Installation and uninstallation authorization" on page 3628.
2. Log in to the system. On AIX, you must log in as root. On Linux and on other UNIX computers, your user ID must have root authority to complete installation. Follow your local security guidelines to acquire root authority; either log in as root, or log in as another user and become root.
3. Stop all brokers that are running on this computer by using the **mqsistop** command. If you are applying service on a computer that is running Linux on x86, close all instances of the WebSphere Message Broker Toolkit and the WebSphere Message Broker Explorer.

Check that all files associated with WebSphere Message Broker are closed.

4. Change to the directory where you downloaded the fix pack file. The file name is in the form *V.R.M-prod-platform-fixpack*.tar.z, where *V.R.M-prod* represents the version, release, and modification level of the product, *-platform* identifies the platform, and *-fixpack* identifies the fix pack. For example, the file name 7.0.0-WS-MB-AIXPPC64-FP0001.tar.z identifies the file for Fix Pack 1 on Version 7.0.0 of WebSphere Message Broker on AIX.

   Note: When using AIX, if you selected HTTP for the download instead of the Download Director, the file name extension is .tar.tgz instead of .tar.z.

5. Replace the .z file with a .tar file, by entering the following command:

   uncompress -fv *V.R.M-prod-platform-fixpack*.tar.z

6. Untar the image and extract all the directories, subdirectories, and files required to apply the update, by entering the following command:

   tar -xvf *V.R.M-prod-platform-fixpack*.tar

7. Decide which interface you want to use for the installation; -graphical (the default option if none is specified), -console, or -silent. For further details of how you can run and installation, and other options that you can specify, see "Installation and uninstallation interfaces" on page 3617 and "How to install and uninstall the Broker component" on page 3618.

8. Depending on the choice you have selected, run one of the following commands. Where *installer* is the appropriate installation program for your platform, as listed in the following table.

   - To use the installation wizard graphical interface, type the *installer* name with no options at a command prompt, and press Enter. Include the absolute or relative path names if you start installation from a directory other than the one in which the wizard exists. When you start the installation wizard, it guides you through a series of panels where you can make choices about where to install the components, and which components you want to install.
   - To use the installation wizard console interface, enter *installer* -console. Include the absolute or relative path names if you start installation from a directory other than the one in which the wizard exists. When you start the installation wizard, it prompts you to enter input for all options, including where to install the components, and which components you want to install.
   - To use the installation wizard silent interface, enter *installer* -silent. Include the absolute or relative path names if you start installation from a directory other than the one in which the wizard exists. If you want to specify non-default settings, include a response file on the command. For more details of how to specify a response file, and how to create and edit a response file to define your requirements, see "Using response files with the Broker component" on page 3621.

Platform	Installation program
AIX	setupaix
HP-Itanium	setuphpia64
Linux on POWER	setuplinuxppc
Linux on x86	setuplinuxia32
Linux on x86-64	setuplinuxx64
Linux on IBM z Systems	setuplinux390

Platform	Installation program
Solaris on SPARC	`setupsolaris`
Solaris on x86-64	`setupsolarisx64`

The installer extracts the update files and runs according to the arguments supplied.

9. Follow the instructions presented by the installation program and provide all input that is required to complete the installation.

   When installation has successfully completed, review the release notes that are supplied in the directory `readmes`, and complete all manual post-installation tasks that are required.

10. If you have defined ODBC connections for brokers on this computer, you must update the `odbc.ini` files before you restart your brokers.
11. When you have completed any required ODBC changes, restart the brokers by using the **mqsistart** command.

Related tasks:

"Applying service to the WebSphere Message Broker Toolkit" on page 325
Apply maintenance updates and program fixes to the WebSphere Message Broker Toolkit.

"Uninstalling service for the Broker component" on page 338
On some platforms, you can remove fixes that you have applied.

"Finding the latest information" on page 232
Access the latest information for WebSphere Message Broker.

Related reference:

"Installation and uninstallation interfaces" on page 3617
You can use different interfaces for installation and uninstallation of the Broker component and the WebSphere Message Broker Toolkit.

"Installation and uninstallation authorization" on page 3628
Check the authorization requirements for the user ID that you use to install or uninstall the Broker component and the WebSphere Message Broker Toolkit.

"Using response files with the Broker component" on page 3621
You can use a response file to define the behavior of an installation or uninstallation wizard that is running the silent interface.

"**mqsistart** command" on page 3965
Use the **mqsistart** command to start the specified broker if all initial verification tests complete successfully.

"**mqsistop** command" on page 3972
Use the **mqsistop** command to stop the specified component.

Installing service on z/OS:

Apply maintenance updates to the broker.

Before you begin

Before you start:

- Download the required update package for WebSphere Message Broker, as described in "Applying service to the Broker component" on page 314.
- Check that your user ID has the correct authority to complete this task; see "Installation and uninstallation authorization" on page 3628 for details.

About this task

Use this three stage process to install service on z/OS:

1. Store the updates on your target system.
2. Install and test the fix pack. If testing is not satisfactory, you can remove this fix pack at this stage.
3. After testing, confirm the fix pack as your current base level of installation.

Follow these instructions to complete the process:

Procedure

1. Receive the updates. Use SMP/E RECEIVE to transfer the contents of the fix pack from the tape or the downloaded package to your system at the specified location.
2. Apply the updates. Use SMP/E APPLY to install the fix pack which completes the link edit steps and generates a runtime environment in your chosen location.

   The installation process completes updates to USS and to four product data sets (SBIPSAMP, SBIPINST, SBIPPROC, and SBIPAUTH).

   If your SMP/E target system is not on the same file system as your production system, copy the data sets (including the data set that represents your USS mount point) to the system where they are to be used, overwriting the current content. When you complete this action, all runtime components use the updated data sets when they next restart.

   If you do not want all your brokers to use the updated installation, copy the data sets to a different location and start selected brokers against that copy. Further details are provided in "Multiple concurrent installations at different levels."

   You can specify the same destination for receiving the fix pack contents (the SMP/E target system) and for installing (the production system). If you specify a different location, you must copy the product data sets to the appropriate location.

   When you have installed the fix pack, test it to ensure that it is working correctly in your environment. If you find problems at this stage, you can remove the fix pack by using SMP/E RESTORE. The command restores your installation to the level that you last accepted (SMP/E ACCEPT).
3. Confirm the update. Use SMP/E ACCEPT to commit the changes and to establish this fix pack as the current level. When you have completed this step, you cannot restore your system to a previous service level. If you want to use an earlier service level, you must reinstall that level.

Multiple concurrent installations at different levels:
About this task

If you want to install a later fix pack level, and retain your existing installation so that you can use both service levels, install the second level in a different location. You can then set up additional customization so that one or more of your existing brokers can use the second service level. By retaining the product files in the first location, you can switch a broker back to the first level, if required.

The following steps are a summary of the tasks that you must complete to run a broker at one service level concurrently with a broker at another service level. These steps are identical to those you take for setting up an initial installation, except where indicated.

For a full description of installation and customization for brokers, see "Creating a broker on z/OS" on page 620.

Procedure

1. To install a second service or fix pack level:
   a. Install the second level of the broker by using the instructions provided with that level.
   b. Copy all the template data sets and the main files from the HFS file system to a unique location.
2. To prepare a broker for use with the second installation:
   a. Create and customize a new broker component data set. Complete the steps described in "Customizing the broker component data set" on page 624.
   b. Copy the template JCL from the second location to the new dataset.
   c. Customize the JCL to work with the second location. For more details about this step, see "Customizing the broker JCL" on page 625.
3. To change a broker to work with the second installation:
   a. Submit the `BIPGEN` job to update the environment file `ENVFILE` for the broker. This task is described in "Creating the environment file" on page 626.
   b. Copy the customized started task from the new data set into the procedures library, for example `USER.PROCLIB`.
   c. Start the broker; the broker is now associated with the second location.
4. To roll back to the previous version:
   a. Open the original broker component data set, that is associated with the first location.
   b. Resubmit the `BIPGEN` job from that data set.
   c. Copy the original `PROC` into the procedures library `USER.PROCLIB`.
   d. Stop and restart the broker.

Related tasks:

"Installing service on Windows" on page 317
Apply maintenance updates and program fixes to the broker component.

"Installing service on Linux and UNIX" on page 320
Apply maintenance updates and program fixes to the Broker component.

"Finding the latest information" on page 232
Access the latest information for WebSphere Message Broker.

"Applying service to the WebSphere Message Broker Toolkit" on page 325
Apply maintenance updates and program fixes to the WebSphere Message Broker Toolkit.

Related reference:

"Customization tasks and roles on z/OS" on page 3984
System programmers do most of the customization of WebSphere Message Broker.

"Installation and uninstallation authorization" on page 3628
Check the authorization requirements for the user ID that you use to install or uninstall the Broker component and the WebSphere Message Broker Toolkit.

Applying service to the WebSphere Message Broker Toolkit

Apply maintenance updates and program fixes to the WebSphere Message Broker Toolkit.

Before you begin

Before you apply a fix pack or other service pack to the WebSphere Message Broker Toolkit, you might want to preserve the resources that you have created or imported into Version 7.0. For details of this task, see "Backing up the WebSphere Message Broker Explorer and WebSphere Message Broker Toolkit workspace" on page 1016.

Check that your user ID has the correct authority to complete this task; see "Installation and uninstallation authorization" on page 3628 for details.

About this task

The WebSphere Message Broker Toolkit uses IBM Installation Manager to install service updates. Occasionally, an update is also required to Installation Manager; if an update is required, it is downloaded automatically and you are prompted to apply service to Installation Manager before you can apply service to the WebSphere Message Broker Toolkit.

Occasionally, updates to the information center are also made available. These updates are integrated with component updates when they are made available, and are therefore automatically included when you apply any updates using these instructions.

If you have installed the Trial Edition, you can apply service to this installation, if you are within your 90 day trial period.

You can install multiple versions of the WebSphere Message Broker Toolkit at the same modification or fix pack level, or at different levels, on a single computer. Each installation must be in a separate package group; package groups are described in more detail in "IBM Installation Manager" on page 3600.

For information about current fixes, follow the link **Recommended fixes** in the Download section of the WebSphere Message Broker support web page. Click the link for Version 7.0, and select the updates you want to apply. Service release notes, the list of problems fixed in a fix pack or other maintenance release, and the instructions you must follow to complete the service upgrade are available. The instructions provide full details of how to apply each service fix, and include information about fixes to Installation Manager, if required. Always use those instructions in preference to the information in this topic, because they are maintained regularly and provide the latest information.

If you want Installation Manager to retrieve updates from a different location, see "Changing the update repositories for Installation Manager" on page 327. If you use FTP or HTTP proxies to allow applications to access the Internet, see "Setting FTP and HTTP proxies for Installation Manager" on page 329.

Use Installation Manager for all service updates, including those updates to Installation Manager itself. In most cases, the following instructions are sufficient; occasionally, further steps are necessary, therefore you must always check with the instructions provided with each fix on the support website.

Procedure

1. Start Installation Manager by using one of the following two methods:
   - From the Linux main menu or the Windows Start menu:
     - Linux On Linux: You can use this option only if you are logged in as root. If you have logged in as another user, you cannot use the main menu links, even if you become root.
       - On Red Hat: Click **System** > **IBM Installation Manager**.
       - On SUSE: Click **System Tools** > **IBM Installation Manager**.
     - Windows On Windows: Click **Start** > **Programs** > **IBM Installation Manager** > **IBM Installation Manager**.
   - From the command line:

     Navigate to the `eclipse` subdirectory in the directory in which Installation Manager is installed and run the following command:
     - Linux `IBMIM`
     - Windows `IBMIM.exe`

   Installation Manager opens and displays its initial panel.
2. Click **Update Packages**, then select the package group in which you have installed the WebSphere Message Broker Toolkit.
3. Click **Next**. Installation Manager searches the selected update site or sites, and displays a list of available updates for the products and features that you have installed.

   Initially **Show recommended** (below the list) is selected, and the latest updates only are shown. To include earlier updates as well, select **Show all**.

   For information about how you can change the locations that are searched, see "Changing the update repositories for Installation Manager" on page 327.
4. Select the updates that you want to apply and click **Update**. The updates are applied.
5. To return to the Installation Manager initial panel, click **OK**.
6. To close Installation Manager, click **File** > **Exit**. Verify that the changes have worked; see "Verifying your installation by using the WebSphere Message Broker Toolkit" on page 291.

Results

If you want to remove service that you have applied, see "Uninstalling service from the WebSphere Message Broker Toolkit" on page 344.

Related tasks:

"Changing the update repositories for Installation Manager" on page 327
If you have a large number of WebSphere Message Broker Toolkit users, or your users do not have regular access to the Internet, you might find it more convenient to download updates to a local site or server, and modify Installation Manager to access the updates at that alternative site.

"Setting FTP and HTTP proxies for Installation Manager" on page 329
If your environment has a firewall, or uses proxies to control access to the Internet, you can change the Installation Manager settings to enhance your access security.

"Applying service to the Broker component" on page 314
Apply maintenance updates and program fixes to the Broker component.

"Backing up the WebSphere Message Broker Explorer and WebSphere Message Broker Toolkit workspace" on page 1016
The WebSphere Message Broker Explorer and WebSphere Message Broker Toolkit workspaces contain your personal settings and data, such as message flow and message set resources. You can have multiple workspaces in different locations, and you can also have references to projects that are in other locations, therefore consider all these locations when you back up your resources.

"Uninstalling service from the WebSphere Message Broker Toolkit" on page 344
Remove maintenance or fixes from the WebSphere Message Broker Toolkit.

"Uninstalling the WebSphere Message Broker Toolkit" on page 340
Choose the method that you want to use to uninstall the WebSphere Message Broker Toolkit and follow the instructions to remove the component.

"Finding the latest information" on page 232
Access the latest information for WebSphere Message Broker.

"Verifying your installation by using the WebSphere Message Broker Toolkit" on page 291
Use the instructions in this tutorial to verify your installation of WebSphere Message Broker and learn how to run samples with the WebSphere Message Broker Toolkit.

Related reference:

"Installation and uninstallation interfaces" on page 3617
You can use different interfaces for installation and uninstallation of the Broker component and the WebSphere Message Broker Toolkit.

"Installation and uninstallation authorization" on page 3628
Check the authorization requirements for the user ID that you use to install or uninstall the Broker component and the WebSphere Message Broker Toolkit.

Changing the update repositories for Installation Manager:

If you have a large number of WebSphere Message Broker Toolkit users, or your users do not have regular access to the Internet, you might find it more convenient to download updates to a local site or server, and modify Installation Manager to access the updates at that alternative site.

About this task

For example, if you want to limit Internet access to a restricted set of computers, or if you want to reduce download times, or both, you can set up local repositories that your users can access to apply updates.

To set up alternative repositories:

Procedure

1. Start Installation Manager from the Start menu (on Windows) or main menu (on Linux on x86), or from the command line. Installation Manager opens and displays its initial panel.
2. Click **File** > **Preferences**. The Preferences dialog is displayed.
3. Click **Repositories** in the left pane. The Repositories page is displayed.
4. Click **Add Repository**. If you know the name and location of the update repositories that Installation Manager is to search, enter the URL or fully qualified location of the update file, including the repository name; for example, `repository.config`.

To search for repositories, which can be on a local or a remote computer, click **Browse** and navigate to the correct location. The field **File of type** lists the values `repository.config`, `diskTag.inf`, `.jar`, and `.zip`, and repositories with these names and types are searched for by default. The identity of an update depends on what type of update it is (for example, disk images are named `diskTag.inf`),

If you are not sure what updates you might want, leave the contents of field **File of type** unchanged; all available updates are shown. Click the required repository to select it, then click **Open**. The repository is added to the displayed list of repositories. If the repository is not connected, a red cross is displayed in the **Connection** column. You might need to check your network connections, or try again at a later time.

When it is added to the list, the repository is initially shown as selected, to indicate that its contents will be included when you run Installation Manager to update packages.

Repeat this step to add additional repositories if appropriate from the same or another location. You can have one or more locations selected at any time; Installation Manager searches all selected repositories listed on the Repositories page.

5. If you want Installation Manager to search only the repositories that you have added, clear **Search service repositories during installation and update**. The service repositories are those locations on the product support Web sites that host updates. If you clear this option, the default service repositories are not searched. The service repositories location is fixed and predefined; you cannot change it.
6. Click **OK** on the Browse panel if you opened it, and click **OK** on the Repositories page.
7. Click **Apply** to confirm the update, then click **OK** to close the Preferences dialog box.

What to do next

To remove a repository from the selected list, clear its selection. If you want to delete a repository in the list, select the repository and click **Remove Repository**. To clear selection of all listed repositories, click **Clear Credentials**.

Related tasks:

"Applying service to the WebSphere Message Broker Toolkit" on page 325
Apply maintenance updates and program fixes to the WebSphere Message Broker Toolkit.

"Applying service to the Broker component" on page 314
Apply maintenance updates and program fixes to the Broker component.

"Backing up the WebSphere Message Broker Explorer and WebSphere Message Broker Toolkit workspace" on page 1016
The WebSphere Message Broker Explorer and WebSphere Message Broker Toolkit workspaces contain your personal settings and data, such as message flow and message set resources. You can have multiple workspaces in different locations, and you can also have references to projects that are in other locations, therefore consider all these locations when you back up your resources.

"Uninstalling service from the WebSphere Message Broker Toolkit" on page 344
Remove maintenance or fixes from the WebSphere Message Broker Toolkit.

"Uninstalling the WebSphere Message Broker Toolkit" on page 340
Choose the method that you want to use to uninstall the WebSphere Message Broker Toolkit and follow the instructions to remove the component.

Related reference:

"Installation and uninstallation interfaces" on page 3617
You can use different interfaces for installation and uninstallation of the Broker component and the WebSphere Message Broker Toolkit.

Setting FTP and HTTP proxies for Installation Manager:

If your environment has a firewall, or uses proxies to control access to the Internet, you can change the Installation Manager settings to enhance your access security.

About this task

You do not have to complete this task if you have changed the Installation Manager repository location to a local site.

Complete the following steps:

Procedure

1. Start Installation Manager from the Start menu (on Windows) or main menu (on Linux on x86), or from the command line. Installation Manager opens and displays its initial panel.
2. Click **File** > **Preferences**. The Preferences dialog is displayed.
3. Expand the **Internet** tree in the left pane, and select **FTP Proxy** or **HTTP Proxy**.
4. Select **Enable proxy server** and enter the details in the entry fields that are required for the protocol that you selected.
5. Click **Apply** to confirm the updates, then click **OK** to close the Preferences dialog box.

Related tasks:

"Applying service to the WebSphere Message Broker Toolkit" on page 325
Apply maintenance updates and program fixes to the WebSphere Message Broker Toolkit.

"Applying service to the Broker component" on page 314
Apply maintenance updates and program fixes to the Broker component.

"Backing up the WebSphere Message Broker Explorer and WebSphere Message Broker Toolkit workspace" on page 1016
The WebSphere Message Broker Explorer and WebSphere Message Broker Toolkit workspaces contain your personal settings and data, such as message flow and message set resources. You can have multiple workspaces in different locations, and you can also have references to projects that are in other locations, therefore consider all these locations when you back up your resources.

"Uninstalling service from the WebSphere Message Broker Toolkit" on page 344
Remove maintenance or fixes from the WebSphere Message Broker Toolkit.

"Uninstalling the WebSphere Message Broker Toolkit" on page 340
Choose the method that you want to use to uninstall the WebSphere Message Broker Toolkit and follow the instructions to remove the component.

Related reference:

"Installation and uninstallation interfaces" on page 3617
You can use different interfaces for installation and uninstallation of the Broker component and the WebSphere Message Broker Toolkit.

Applying service to the WebSphere Message Broker Explorer

You can apply maintenance or fixes to the WebSphere Message Broker Explorer.

About this task

Download the latest fix pack from the WebSphere Message Broker support web page:

Procedure

1. Click **Recommended fixes** in the Download section of the support page.
2. Click your WebSphere Message Broker product. Available fixes are displayed.
3. Click the fix that you want to download. Details about the fix, and instructions for download and installation, are displayed.
4. Click **Problems fixed** to find out what PMRs, APARs, and defects have been fixed in the fix pack.
5. Click **Release notes** to read more detail about the fix pack contents.
6. Click **FC** in the **Download package** section to start the download process. Accept the download terms and conditions and sign in to the download site with your IBM ID. If you do not have an ID, you can register on this page. When the download completes, extract the executable file or binary file.
7. Decide whether to use the graphical installation, a console installation, or a silent installation.
   - To use the graphical installation, see Installing the WebSphere Message Broker Explorer.
   - To use the installation wizard with a screen reader, see Installing the WebSphere Message Broker Explorer by using console mode with a screen reader.
   - To use a console installation on Windows, see Installing the WebSphere Message Broker Explorer in console mode on Windows.
   - To use a console installation on Linux, see Installing the WebSphere Message Broker Explorer in console mode on Linux.
   - To use a silent installation, see Installing the WebSphere Message Broker Explorer in silent mode.

   To apply service to any Broker component on Windows using the *fixpacklauncher* see Installing service on Windows.
8. Launch the installation wizard by running the executable file or binary file.

   The name of this file is in the form `V.R.M-prod-platform-fixpack.exe` on Windows and `V.R.M-prod-platform-fixpack.bin` on Linux platforms. Where *V.R.M* represents the version, release, and modification level of the product, *prod* identifies the product, *platform* identifies the platform, and *fixpack* identifies the fix pack. For example, the file name `7.0.0-WS-MBX-WINIA32-FP0004.exe` identifies the file for Fix Pack 4 on Version 7.0.0 of WebSphere Message Broker Explorer on Windows.

   On Linux, you must make this file executable by using the **chmod** command. For example, use the following command: `chmod u+x 7.0.0-WS-MBX-LINUXIA32-FP0004.bin`
9. When you have launched the wizard, continue with the installation process by working through each of the panels. The installation wizard itself contains help information about selected panels. You can apply service to a new location on your system.

Related tasks:

"Applying service to the WebSphere Message Broker Toolkit" on page 325
Apply maintenance updates and program fixes to the WebSphere Message Broker Toolkit.

"Applying service to the Broker component" on page 314
Apply maintenance updates and program fixes to the Broker component.

"Uninstalling service from the WebSphere Message Broker Explorer" on page 350
You must reinstall the WebSphere Message Broker Explorer to revert to a previous service level.

"Installing WebSphere Message Broker Explorer" on page 280
To use WebSphere Message Broker Explorer only, without installing the complete WebSphere Message Broker Toolkit, use the WebSphere Message Broker Explorer installation wizard to install the WebSphere Message Broker Explorer.

"Finding the latest information" on page 232
Access the latest information for WebSphere Message Broker.

Uninstalling

Remove the Broker component, the WebSphere Message Broker Toolkit, or the WebSphere Message Broker Explorer from your computer.

Before you begin

Before you start:

Check that your user ID has the correct authority to complete this task. The requirements for each platform are defined in "Installation and uninstallation authorization" on page 3628.

About this task

Follow the appropriate link for the task that you want to complete:

- "Uninstalling the Broker component" on page 332
- "Uninstalling the WebSphere Message Broker Toolkit" on page 340
- "Uninstalling the WebSphere Message Broker Explorer" on page 346
- "Uninstalling the WebSphere Message Broker ODBC Database Extender (IE02)" on page 339

What to do next

You might also want to uninstall complementary products. See the documentation provided by those products to complete this task:

- IBM DB2: Access the appropriate information for your installation:
  - DB2 V9.1 Information Center (distributed systems)
  - DB2 V9.5 Information Center (distributed systems)
  - DB2 Information Center (z/OS) (Versions 8 and 9)
- IBM Tivoli License Manager: access the IBM Tivoli License Manager Information Center.

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

"Finding the latest information" on page 232
Access the latest information for WebSphere Message Broker.

"Resolving problems when uninstalling" on page 3525
Work through the advice provided to help you to deal with problems that can arise when the product is uninstalled.

"Uninstalling the WebSphere Message Broker ODBC Database Extender (IE02)" on page 339
How you uninstall the WebSphere Message Broker ODBC Database Extender.

Related reference:

"Installation" on page 3581
Use the reference information in this section to understand installation requirements, installation options, and how they affect your computer.

"Installation and uninstallation authorization" on page 3628
Check the authorization requirements for the user ID that you use to install or uninstall the Broker component and the WebSphere Message Broker Toolkit.

Uninstalling the Broker component

You can uninstall the Broker component on distributed systems in a number of ways.

About this task

This section describes this task for the following systems:
- AIX
- HP-UX
- Linux
- Solaris
- Windows

If you are uninstalling on Linux or UNIX systems, make sure that the directory `/var` is not full; the uninstallation wizard requires space in this directory while it is running.

Procedure

1. Check that your user ID has the correct authority to uninstall the Broker component, or to uninstall service that you have applied. The requirements are defined in "Installation and uninstallation authorization" on page 3628.
2. Decide which mode of uninstallation you want to use. The choices are explained in "How to install and uninstall the Broker component" on page 3618.
3. If you have multiple installations of the Broker component on your system, see "How to uninstall multiple installations of the Broker component" on page 3620.
4. Follow the uninstallation instructions provided in the appropriate topic:
   - "Uninstalling the Broker component by using the graphical interface" on page 333
   - "Uninstalling the Broker component by using the console interface" on page 335
   - "Uninstalling the Broker component by using the silent interface" on page 336
   - "Uninstalling service for the Broker component" on page 338

Results

If, in exceptional circumstances, the uninstallation wizard fails to remove the product successfully, you can remove the product manually. Contact your IBM Service Center for assistance if a failure occurs.

For information about removing service updates, see "Uninstalling service for the Broker component" on page 338.

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

"Applying service to the Broker component" on page 314
Apply maintenance updates and program fixes to the Broker component.

"Finding the latest information" on page 232
Access the latest information for WebSphere Message Broker.

"Resolving problems when uninstalling" on page 3525
Work through the advice provided to help you to deal with problems that can arise when the product is uninstalled.

Related reference:

"Installation" on page 3581
Use the reference information in this section to understand installation requirements, installation options, and how they affect your computer.

"Installation and uninstallation authorization" on page 3628
Check the authorization requirements for the user ID that you use to install or uninstall the Broker component and the WebSphere Message Broker Toolkit.

Uninstalling the Broker component by using the graphical interface

Remove the Broker component by using the graphical interface.

Before you begin

Always use the WebSphere Message Broker uninstallation wizard to remove the Broker component, unless otherwise stated. Do not use the uninstallation program that is provided by the operating system; for example, **geninstall** on AIX, or **swremove** on HP-UX.

On HP-UX Version 11.23, and on Linux on POWER, a known restriction prevents the creation of the wizard. If this situation occurs, use the alternative command that is specified in these instructions. Also, check the `readme.html` file to ensure that no additional operating systems are affected by this restriction.

About this task

For information about the graphical interface and alternative interfaces available, see "How to install and uninstall the Broker component" on page 3618.

To start the graphical interface of the uninstaller, complete the following steps:

Procedure

1. Stop all running processes that are associated with brokers before you uninstall. Use the "**mqsilist** command" on page 3882 to check which brokers are running. Use the "**mqsistop** command" on page 3972 to stop each broker.
2. On Windows, Linux on x86, and Linux on x86-64, end all active sessions of the WebSphere Message Broker Toolkit.
3. If you do not plan to reinstall brokers on this computer, delete all the brokers that are associated with this installation by using the "**mqsideletebroker** command" on page 3863. When you delete a broker, all its associated data and resources are deleted.
4. Run the following command from outside the installation directory, specifying the full path. This method of invocation ensures that the full directory structure is removed during uninstallation. Where *install_dir* is the home directory of your WebSphere Message Broker installation:
   - On HP-UX systems running Version 11.23 only, and on Linux on POWER systems:

```
/install_dir/_uninst_runtime_jvm/bin/java -jar
/install_dir/_uninst_runtime/uninstall.jar
```

   - On all other operating systems:

     install_dir/_uninst_runtime/*uninstaller*

   If you accepted the default location during installation, this directory is as follows. The default directory includes the version and release of the product that you are installing in the format v.r (version.release):

   Linux /opt/ibm/mqsi/*v.r*

   UNIX /opt/IBM/mqsi/*v.r*

   Windows 32-bit
   C:\Program Files\IBM\MQSI*v.r*

   Windows 64-bit
   C:\Program Files\IBM\MQSI*v.r* for the 64-bit version of WebSphere Message Broker

   C:\Program Files (x86)\IBM\MQSI*v.r* for the 32-bit version of WebSphere Message Broker

   These locations define the default value of *install_dir* on each platform.

   Alternatively, if you have a single installation of the Broker component on Windows, you can navigate to **Start** > **Settings** > **Control Panel** > **Add/Remove Programs**. Select WebSphere Message Broker and click **Change/Remove**. The uninstaller graphical interface opens.
5. Follow the prompts to complete uninstallation.

Related tasks:

"Uninstalling the Broker component by using the silent interface" on page 336
Remove the Broker component without user interaction by using the silent interface.

"Uninstalling the Broker component by using the console interface" on page 335
Remove the Broker component by using the console interface.

"Resolving problems when uninstalling" on page 3525
Work through the advice provided to help you to deal with problems that can arise when the product is uninstalled.

Related reference:

"**mqsideletebroker** command" on page 3863
Use the **mqsideletebroker** command to delete a named broker. The command also deletes the queues on the associated queue manager (created when the broker was created). You can also specify that the queue manager is to be deleted.

"**mqsilist** command" on page 3882
Use the **mqsilist** command to list installed brokers and their associated resources.

"**mqsistop** command" on page 3972
Use the **mqsistop** command to stop the specified component.

Uninstalling the Broker component by using the console interface

Remove the Broker component by using the console interface.

Before you begin

Always use the WebSphere Message Broker uninstallation wizard to remove the Broker component, unless otherwise stated. Do not use the uninstallation program that is provided by the operating system; for example, **geninstall** on AIX, or **swremove** on HP-UX.

On HP-UX Version 11.23, and on Linux on POWER, a known restriction prevents the creation of the wizard. If this situation occurs, use the alternative command specified in these instructions. Also check the `readme.html` file to ensure that no additional operating systems are affected by this restriction.

About this task

For information about the console interface and alternative interfaces available, see "How to install and uninstall the Broker component" on page 3618.

To start the console interface of the uninstaller, complete the following steps:

Procedure

1. Stop all running processes that are associated with brokers before you uninstall. Use the "**mqsilist** command" on page 3882 to check which brokers are running. Use the "**mqsistop** command" on page 3972 to stop each broker.
2. On Windows, Linux on x86, and Linux on x86-64, end any active sessions of the WebSphere Message Broker Toolkit.
3. If you do not plan to reinstall brokers on this computer, delete all the brokers that are associated with this installation by using the "**mqsideletebroker** command" on page 3863. When you delete a broker, all its associated data and resources are deleted.
4. Run the uninstallation program from outside the installation directory, specifying the full path to ensure that the folders are removed during uninstallation, where *install_dir* is the home directory of your WebSphere Message Broker installation:
   - On some HP-UX systems that are running Version 11.23:

```
/install_dir/_uninst_runtime_jvm/bin/java -jar
/install_dir/_uninst_runtime/uninstall.jar -console
```

   - On some Linux on POWER systems:

```
/install_dir/_uninst_runtime_jvm/jre/bin/java -jar
/install_dir/_uninst_runtime/uninstall.jar -console
```

   - On all other operating systems:

```
install_dir/_uninst_runtime/uninstaller -console
```

If you accepted the default location during installation, this directory is as follows. The default directory includes the version and release of the product that you are installing in the format v.r (version.release):

Linux `/opt/ibm/mqsi/v.r`

UNIX `/opt/IBM/mqsi/v.r`

Windows 32-bit
`C:\Program Files\IBM\MQSI\v.r`

Windows 64-bit
`C:\Program Files\IBM\MQSI\v.r` for the 64-bit version of WebSphere Message Broker

`C:\Program Files (x86)\IBM\MQSI\v.r` for the 32-bit version of WebSphere Message Broker

These locations define the default value of *install_dir* on each platform.

5. When started, the uninstaller presents text-based screens on the console. Navigate between these screens using the following options:
   - 1 next
   - 2 previous
   - 3 cancel
   - 4 redisplay

   The uninstaller asks you to confirm that you want to uninstall the product and that the location it is uninstalling from is correct. When you have responded to these questions, the uninstallation starts.

Related tasks:

"Uninstalling the Broker component by using the graphical interface" on page 333
Remove the Broker component by using the graphical interface.

"Uninstalling the Broker component by using the silent interface"
Remove the Broker component without user interaction by using the silent interface.

"Uninstalling the Broker component by using the console interface" on page 335
Remove the Broker component by using the console interface.

"Resolving problems when uninstalling" on page 3525
Work through the advice provided to help you to deal with problems that can arise when the product is uninstalled.

Related reference:

"**mqsideletebroker** command" on page 3863
Use the **mqsideletebroker** command to delete a named broker. The command also deletes the queues on the associated queue manager (created when the broker was created). You can also specify that the queue manager is to be deleted.

"**mqsilist** command" on page 3882
Use the **mqsilist** command to list installed brokers and their associated resources.

"**mqsistop** command" on page 3972
Use the **mqsistop** command to stop the specified component.

Uninstalling the Broker component by using the silent interface

Remove the Broker component without user interaction by using the silent interface.

Before you begin

Always use the WebSphere Message Broker uninstallation wizard to remove the Broker component, unless otherwise stated. Do not use the uninstallation program that is provided by the operating system; for example, **geninstall** on AIX, or **swremove** on HP-UX.

On HP-UX Version 11.23, and on Linux on POWER, a known restriction prevents the creation of the wizard. If this situation occurs, use the alternative command specified in these instructions. Also check the `readme.html` file to ensure that no additional operating systems are affected by this restriction.

If you use the silent interface to uninstall the Broker component, the wizard always uninstalls the Broker component from the last known Version 7.0 installation location (that is, the most recent installation), regardless of the location of the uninstallation wizard that you start. If you want to remove the Broker component from an earlier Version 7.0 installation, use the console or graphical interface.

About this task

You can remove the Broker component without user interaction. This process is called unattended (or silent) removal. This topic describes how to run the uninstallation program silently without a response file; the program assumes all the default values.

If you want to complete a silent uninstallation, but have non-default requirements, you can use a response file. A response file is a text file that contains values for the options that you select when you uninstall WebSphere Message Broker. For more details of how to create, edit, and employ a response file to specify your requirements, see Using response files.

For information about the silent interface and alternative interfaces available, see "How to install and uninstall the Broker component" on page 3618.

To start the silent interface of the uninstaller, complete the following steps:

Procedure

1. Stop all running processes that are associated with brokers before you uninstall. Use the "**mqsilist** command" on page 3882 to check which brokers are running. Use the "**mqsistop** command" on page 3972 to stop each broker.
2. On Windows, Linux on x86, and Linux on x86-64, end any active sessions of the WebSphere Message Broker Toolkit.
3. If you do not plan to reinstall brokers on this computer, delete all the brokers that are associated with this installation by using the "**mqsideletebroker** command" on page 3863. When you delete a broker, all its associated data and resources are deleted.
4. Run the following command from outside the installation directory, specifying the full path (this method of invocation ensures that the full directory structure is removed during uninstallation), where *install_dir* is the home directory of your WebSphere Message Broker installation:
   - On some HP-UX systems that are running Version 11.23 only:

```
/install_dir/_uninst_runtime_jvm/bin/java -jar
/install_dir/_uninst_runtime/uninstall.jar -silent
```

   - On some Linux on POWER systems:

```
/install_dir/_uninst_runtime_jvm/jre/bin/java -jar
/install_dir/_uninst_runtime/uninstall.jar -silent
```

- On all other operating systems:

  install_dir/_uninst_runtime/*uninstaller -silent*

If you accepted the default location during installation, this directory is as follows. The default directory includes the version and release of the product that you are installing in the format v.r (version.release):

Linux /opt/ibm/mqsi/*v.r*

UNIX /opt/IBM/mqsi/*v.r*

Windows 32-bit
C:\Program Files\IBM\MQSI*v.r*

Windows 64-bit
C:\Program Files\IBM\MQSI*v.r* for the 64-bit version of WebSphere Message Broker

C:\Program Files (x86)\IBM\MQSI*v.r* for the 32-bit version of WebSphere Message Broker

These locations define the default value of *install_dir* on each platform. The uninstallation program runs without interaction.

Related tasks:

"Uninstalling the Broker component by using the graphical interface" on page 333
Remove the Broker component by using the graphical interface.

"Uninstalling the Broker component by using the console interface" on page 335
Remove the Broker component by using the console interface.

"Resolving problems when uninstalling" on page 3525
Work through the advice provided to help you to deal with problems that can arise when the product is uninstalled.

Related reference:

"**mqsideletebroker** command" on page 3863
Use the **mqsideletebroker** command to delete a named broker. The command also deletes the queues on the associated queue manager (created when the broker was created). You can also specify that the queue manager is to be deleted.

"**mqsilist** command" on page 3882
Use the **mqsilist** command to list installed brokers and their associated resources.

"**mqsistop** command" on page 3972
Use the **mqsistop** command to stop the specified component.

Uninstalling service for the Broker component

On some platforms, you can remove fixes that you have applied.

About this task

Distributed systems

You cannot remove individual fixes that you have applied to the Broker component on distributed systems. If you want to restore a system to a previous service level, you must uninstall the product before you reinstall the required level of service.

If you have a single installation on your computer, create and retain a backup image, or keep previous GA or fix pack images or media, in case you experience problems after you have installed service. If you have multiple installations, each one can be at a different service level and you

can transfer the broker to work with an alternative installation at a newer or older level of service. This option is described in "Applying service to the Broker component" on page 314.

When you uninstall the product, the components that you have created are not lost, and you can continue to use these components and associated resources with the reinstalled product. However, you might want to take a backup of the installation and work path directories before you start this procedure.

To restore a system on which you have a single installation to a previous service level:

1. Uninstall the entire product.
2. Reinstall the GA level of the product. This step is required to ensure that you comply with your license agreement.
3. Install the required level of fix pack.

z/OS On z/OS systems, you can uninstall service levels under some circumstances during the installation phase; for further information, see "Installing service on z/OS" on page 322.

Related concepts:

"Coexistence with previous versions and other products" on page 139
WebSphere Message Broker Version 7.0 supports restricted coexistence with previous versions and with other products.

Related tasks:

"Applying service to the Broker component" on page 314
Apply maintenance updates and program fixes to the Broker component.

"Uninstalling the Broker component" on page 332
You can uninstall the Broker component on distributed systems in a number of ways.

"Resolving problems when uninstalling" on page 3525
Work through the advice provided to help you to deal with problems that can arise when the product is uninstalled.

Uninstalling the WebSphere Message Broker ODBC Database Extender (IE02)

How you uninstall the WebSphere Message Broker ODBC Database Extender.

Procedure

To uninstall the WebSphere Message Broker ODBC Database Extender, change your directory to `Uninstall_IE02` within the installation path `IE02 ($IE02_PATH)`, and run the following command:

```
./Uninstall_IE02
```

This command launches the uninstaller in the same mode that it was previously installed. For example, in Graphical, Console, or Silent mode.
The uninstaller does not delete the `/var/mqsi/common/profiles/IE02.sh` file. You can remove this file manually after you have removed the SupportPac, although no problems are caused if the file is not removed.

Related tasks:

"Configuring the WebSphere Message Broker ODBC Database Extender (IE02)" on page 275
WebSphere Message Broker ODBC Database Extender encapsulates the unixODBC driver manager, which is an implementation of the Open DataBase Connectivity

interface for UNIX systems, and this topic describes how you configure it.

"Installing the WebSphere Message Broker ODBC Database Extender (IE02)" on page 273
WebSphere Message Broker ODBC Database Extender encapsulates the unixODBC driver manager, which is an implementation of the Open DataBase Connectivity interface for UNIX systems. This topic describes how you install WebSphere Message Broker ODBC Database Extender.

Uninstalling the WebSphere Message Broker Toolkit

Choose the method that you want to use to uninstall the WebSphere Message Broker Toolkit and follow the instructions to remove the component.

About this task

Complete the following steps to uninstall the WebSphere Message Broker Toolkit:

Procedure

1. Check that your user ID has the correct authority to uninstall the WebSphere Message Broker Toolkit or to uninstall service that you have applied. The requirements are defined in "Installation and uninstallation authorization" on page 3628.
2. If you are uninstalling the WebSphere Message Broker Toolkit component, decide which mode of uninstallation you want to use. The choices are explained in "How to install and uninstall the WebSphere Message Broker Toolkit" on page 3623.
3. Follow the instructions provided in the appropriate topic:
   - "Uninstalling the WebSphere Message Broker Toolkit in graphical mode" on page 341
   - "Uninstalling the WebSphere Message Broker Toolkit in silent mode" on page 343
   - "Uninstalling service from the WebSphere Message Broker Toolkit" on page 344

Results

When you uninstall the WebSphere Message Broker Toolkit component, entries are removed from the system menus:

Linux If you have installed the WebSphere Message Broker Toolkit more than once on Linux on x86, the main menu shows a single entry, regardless of the number of installations that you have completed; the entry always accesses the last installation that you completed. When you uninstall this component, the single entry is removed, and you cannot access remaining installations using the main menu. Navigate to the correct package group installation directory and start the WebSphere Message Broker Toolkit from the command line.

Windows Entries in the Windows Start menu are grouped by package group. When you uninstall this component, the entry within the appropriate package group is correctly removed.

Related tasks:

"Finding the latest information" on page 232
Access the latest information for WebSphere Message Broker.

"Uninstalling" on page 331
Remove the Broker component, the WebSphere Message Broker Toolkit, or the WebSphere Message Broker Explorer from your computer.

"Applying service to the WebSphere Message Broker Toolkit" on page 325
Apply maintenance updates and program fixes to the WebSphere Message Broker Toolkit.

"Uninstalling service from the WebSphere Message Broker Toolkit" on page 344
Remove maintenance or fixes from the WebSphere Message Broker Toolkit.

Related reference:

"Installation Guide" on page 233
Installation information for WebSphere Message Broker is provided in the *Installation Guide* that is supplied as a PDF file with your product package.

"Installation and uninstallation authorization" on page 3628
Check the authorization requirements for the user ID that you use to install or uninstall the Broker component and the WebSphere Message Broker Toolkit.

Uninstalling the WebSphere Message Broker Toolkit in graphical mode

The graphical interface guides you through the process of uninstalling the WebSphere Message Broker Toolkit.

About this task

For information about the modes that are available, see "How to install and uninstall the WebSphere Message Broker Toolkit" on page 3623.

Complete the following steps:

Procedure

1. Log on to the computer on which WebSphere Message Broker Toolkit is installed with a user ID that has the correct authority. The authority requirements are defined in "Installation and uninstallation authorization" on page 3628.
2. Start the IBM Installation Manager.
   - Linux Choose one of the following options:
     - The main menu. You can use this option only if you are logged in as root. If you logged in as another user, you cannot use the main menu links, even if you become root.
       - On Red Hat, click **System** > **IBM Installation Manager**.
       - On SUSE, click **System Tools** > **IBM Installation Manager**.
     - Navigate to the `eclipse` subdirectory in the directory in which Installation Manager is installed and run the command `IBMIM`.
     - On the command line, start the WebSphere Message Broker Toolkit uninstallation wizard (where *package_group_dir* is the directory in which you installed the package group):

       ```
       package_group_dir/uninstall/uninstall.sh
       ```

       For example:

       ```
       /opt/IBM/WMBT700/uninstall/uninstall.sh
       ```
   - Windows Choose one of the following options:
     - Click **Start** > **Programs** > **IBM Installation Manager** > **IBM Installation Manager**.

- Click **Start** > **Settings** > **Control Panel** > **Add/Remove Programs**. Open the package group and select the package group in which you installed WebSphere Message Broker Toolkit, for example **IBM Software Development Platform_1**.
- Navigate to the `eclipse` subdirectory in the directory in which Installation Manager is installed and run the command `IBMIM.exe`.
- On the command line, start the WebSphere Message Broker Toolkit uninstallation wizard (where *package_group_dir* is the directory in which you installed the package group):

  package_group_dir`\uninstall\uninstall.bat`

  For example, on Windows XP:

  `C:\Program Files\IBM\WMBT700\uninstall\uninstall.bat`

3. If you started Installation Manager from the Start or main menu, or by running `IBMIM` or `IBMIM.exe`, click **Uninstall Packages**. The Uninstall Package window displays.
4. Select the WebSphere Message Broker Toolkit, then click **Next**. The Uninstall Summary window is displayed.
5. Check the list of items. Click **Back** to change items, or click **Uninstall** to continue. The uninstallation process starts.
6. Follow the guidance through the series of windows, providing input and responses as required.
7. When the process completes, check the log file for errors. The file `YYYYMMDD_TIME.xml` is written to the following directory:
   - Linux `/var/ibm/InstallationManager/logs`
   - Windows `%ALLUSERSPROFILE%\Application Data\IBM\Installation Manager\logs`, where `%ALLUSERSPROFILE%` is the environment variable that defines the system working directory. The default directory depends on the operating system:
     - On Windows XP and Windows Server 2003: `C:\Documents and Settings\All Users\Application Data\IBM\Installation Manager\logs`.
     - On Windows Vista and later operating systems: `C:\ProgramData\IBM\Installation Manager\logs`.

   The actual value might be different on your computer.
8. Restart your computer to complete the uninstallation of the WebSphere Message Broker Toolkit.

Related tasks:

"Uninstalling the WebSphere Message Broker Toolkit in silent mode" on page 343
Uninstall the WebSphere Message Broker Toolkit without user interaction.

"Uninstalling service from the WebSphere Message Broker Toolkit" on page 344
Remove maintenance or fixes from the WebSphere Message Broker Toolkit.

Related reference:

"How to install and uninstall the WebSphere Message Broker Toolkit" on page 3623
Install and uninstall the WebSphere Message Broker Toolkit by using one of two interfaces.

"Installation and uninstallation authorization" on page 3628
Check the authorization requirements for the user ID that you use to install or uninstall the Broker component and the WebSphere Message Broker Toolkit.

Uninstalling the WebSphere Message Broker Toolkit in silent mode

Uninstall the WebSphere Message Broker Toolkit without user interaction.

About this task

For information about alternative modes that are available, see "How to install and uninstall the WebSphere Message Broker Toolkit" on page 3623.

Complete the following steps:

Procedure

1. Log on to the computer on which WebSphere Message Broker Toolkit is installed with a user ID that has the correct authority. The authority requirements are defined in "Installation and uninstallation authorization" on page 3628.
2. To silently uninstall the WebSphere Message Broker Toolkit, start IBM Installation Manager by using the supplied script files.
   a. Navigate to the uninstallation directory and open the script file in a text editor:
      - Linux *package_group_dir*/uninstall/uninstall.sh

        For example:

        /opt/IBM/WMBT700/uninstall/uninstall.sh
      - Windows *package_group_dir*\uninstall\uninstall.bat

        For example, on Windows XP:

        C:\Program Files\IBM\WMBT700\uninstall\uninstall.bat
   b. Modify the invocation of the IBMIM executable file as follows:
      - Linux If you plan to run the script file from the command line in the uninstallation directory:

        IM_dir/IBMIM -nosplash -input uninstall.xml -silent

        Otherwise:

        IM_dir/IBMIM -nosplash -input *package_group_dir*/uninstall/uninstall.xml -silent
      - Windows If you plan to run the script file from the command line in the uninstallation directory:

        IM_dir\IBMIMc -nosplash -input uninstall.xml -silent

        Otherwise:

        IM_dir\IBMIMc -nosplash -input *package_group_dir*\uninstall\uninstall.xml -silent

      Replace *IM_dir/*, or *IM_dir*\, depending on your platform, with the full path for Installation Manager on your operating system. If you have installed Installation Manager in the default location, the full path is:
      - Linux /opt/IBM/InstallationManager/eclipse/
      - Windows On Windows 32-bit editions: C:\Program Files\IBM\InstallationManager\eclipse\
      - Windows On Windows 64-bit editions: C:\Program Files(x86)\IBM\InstallationManager\eclipse\

        Note, that on Windows you must enclose the full path name in double quotation marks. For example:

```
"C:\Program Files(x86)\IBM\InstallationManager\eclipse\IBMIMc"
 -nosplash -input package_group_dir\uninstall\uninstall.xml
 -silent
```

3. Run the file that you just edited. The product is uninstalled silently; control is returned to the command line when uninstallation is finished.
4. When the process completes, check the log file for errors. The file `YYYYMMDD_TIME.xml` is written to the following directory:
   - Linux `/var/ibm/InstallationManager/logs`.
   - Windows `%ALLUSERSPROFILE%\Application Data\IBM\Installation Manager\logs` where `%ALLUSERSPROFILE%` is the environment variable that defines the system working directory. The default directory depends on the operating system:
     – On Windows XP and Windows Server 2003: `C:\Documents and Settings\All Users\Application Data\IBM\Installation Manager\logs`.
     – On Windows Vista and later operating systems: `C:\ProgramData\IBM\Installation Manager\logs`.

   The actual value might be different on your computer.
5. Restart your computer to complete the uninstallation of the WebSphere Message Broker Toolkit.
6. Delete the uninstallation directory if it still exists.

Related tasks:

"Uninstalling the WebSphere Message Broker Toolkit in graphical mode" on page 341
The graphical interface guides you through the process of uninstalling the WebSphere Message Broker Toolkit.

"Uninstalling service from the WebSphere Message Broker Toolkit"
Remove maintenance or fixes from the WebSphere Message Broker Toolkit.

Related reference:

"How to install and uninstall the WebSphere Message Broker Toolkit" on page 3623
Install and uninstall the WebSphere Message Broker Toolkit by using one of two interfaces.

"Installation and uninstallation authorization" on page 3628
Check the authorization requirements for the user ID that you use to install or uninstall the Broker component and the WebSphere Message Broker Toolkit.

Uninstalling service from the WebSphere Message Broker Toolkit

Remove maintenance or fixes from the WebSphere Message Broker Toolkit.

Before you begin

Linux To remove service from the WebSphere Message Broker Toolkit on Linux on x86 and Linux on x86-64, your user ID must have root authority. Follow your local security guidelines to acquire root authority; either log in as root, or log in as another user and become root.

Windows To remove service from the WebSphere Message Broker Toolkit on Windows, your user ID must have Administrator authority.

About this task

You can remove WebSphere Message Broker interim fix and service updates from the toolkit using IBM Installation Manager.

To uninstall service:

Procedure

1. Close all sessions of the WebSphere Message Broker Toolkit on this computer.
2. Start Installation Manager in one of the following ways:
   - From the Linux on x86 and Linux on x86-64 main menu, or the Windows Start menu:
     - On Linux, you can use this option only if you are logged in as root. If you have logged in as another user, you cannot use the main menu links, even if you become root.
       - On Red Hat, click **System** > **IBM Installation Manager**.
       - On SUSE, click **System Tools** > **IBM Installation Manager**.
     - On Windows, click **Start** > **Programs** > **IBM Installation Manager** > **IBM Installation Manager** .
   - Navigate to the `eclipse` subdirectory in the directory in which Installation Manager is installed and run the following command:
     - Linux `IBMIM`
     - Windows `IBMIM.exe`

   Installation Manager starts and displays its initial panel.
3. Click **Roll Back Packages**, then select the package group in which you have installed the WebSphere Message Broker Toolkit.
4. Click **Next**. The Roll Back Packages panel displays the products that you have installed in the selected group. The service updates that you have applied to any item in the list are displayed under the item to which they apply.
5. If updates are installed, select the updates that you want to remove from each particular installation. A summary of the selected updates is displayed. Click **more details...** to open the readme file associated with the fix.
6. Click **Next** to continue. Installation Manager checks that no WebSphere Message Broker Toolkit sessions are running. If the check fails, an error message is displayed and you cannot continue with the operation. Close all active sessions, then return to this step and click **Next** again.

   Installation Manager displays the exact identification of the fixes that you selected; check that what is listed is correct. If you want to change your choices, click **Back** to view the list of fixes again.
7. Click **Roll Back**. Installation Manager starts to remove the selected fixes. A progress bar is displayed so that you can monitor the request. A completion message is displayed; click **View log** to read the report that is generated by the removal.
8. If the fix has been removed successfully, click **Finish** to return to the initial panel. If the removal failed, examine the contents of the log file to determine the cause of the error.
9. Click **File** > **Exit** to close Installation Manager. Verify that the changes have worked; see “Verifying your installation by using the WebSphere Message Broker Toolkit” on page 291.

Related tasks:

"Applying service to the WebSphere Message Broker Toolkit" on page 325
Apply maintenance updates and program fixes to the WebSphere Message Broker Toolkit.

"Uninstalling the WebSphere Message Broker Toolkit" on page 340
Choose the method that you want to use to uninstall the WebSphere Message Broker Toolkit and follow the instructions to remove the component.

"Verifying your installation by using the WebSphere Message Broker Toolkit" on page 291
Use the instructions in this tutorial to verify your installation of WebSphere Message Broker and learn how to run samples with the WebSphere Message Broker Toolkit.

Uninstalling the WebSphere Message Broker Explorer

You can uninstall the WebSphere Message Broker Explorer by using the graphical installation wizard, by using the console mode of the installation wizard, or by using the installation wizard in silent mode.

To uninstall the WebSphere Message Broker Explorer by using the graphical installation wizard, click the link for your operating system and follow the steps in the topic. To uninstall the WebSphere Message Broker Explorer by using the console mode, click the appropriate link and follow the steps in the topic. The steps for console mode are applicable on Windows, Linux on x86, and Linux on x86-64. You can also uninstall the WebSphere Message Broker Explorer by using the installation wizard in silent mode.

- "Uninstalling the WebSphere Message Broker Explorer on Windows" on page 347
- "Uninstalling the WebSphere Message Broker Explorer on Linux" on page 347
- "Uninstalling the WebSphere Message Broker Explorer in console mode" on page 348
- "Uninstalling the WebSphere Message Broker Explorer in silent mode" on page 349

To uninstall service from the WebSphere Message Broker Explorer, you must reinstall the WebSphere Message Broker Explorer to revert to a previous service level. For more information about uninstalling service, see "Uninstalling service from the WebSphere Message Broker Explorer" on page 350.

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

"Finding the latest information" on page 232
Access the latest information for WebSphere Message Broker.

Related reference:

"Installation" on page 3581
Use the reference information in this section to understand installation requirements, installation options, and how they affect your computer.

"Installation and uninstallation authorization" on page 3628
Check the authorization requirements for the user ID that you use to install or uninstall the Broker component and the WebSphere Message Broker Toolkit.

Uninstalling the WebSphere Message Broker Explorer on Windows

You can uninstall the WebSphere Message Broker Explorer by using the graphical installation wizard.

About this task

Complete the following steps to uninstall the WebSphere Message Broker Explorer by using the graphical installation wizard on Windows.

Procedure

1. From the Windows task bar, click **Start** > **Control Panel**, or **Start** > **Settings** > **Control Panel**.
2. Click **Add or Remove Programs**.
3. Select **IBM WebSphere Message Broker Explorer** and click **Change/Remove**. The WebSphere Message Broker Explorer uninstaller opens.
4. Click **Uninstall** to uninstall the WebSphere Message Broker Explorer. The uninstaller lists any items that cannot be uninstalled. You can delete these items manually.
5. When the uninstall process has completed, click **Done** to close the uninstaller.

Related concepts:

"Uninstalling the WebSphere Message Broker Explorer" on page 346
You can uninstall the WebSphere Message Broker Explorer by using the graphical installation wizard, by using the console mode of the installation wizard, or by using the installation wizard in silent mode.

Related tasks:

"Uninstalling" on page 331
Remove the Broker component, the WebSphere Message Broker Toolkit, or the WebSphere Message Broker Explorer from your computer.

"Uninstalling the WebSphere Message Broker Explorer on Linux"
You can uninstall the WebSphere Message Broker Explorer by using the graphical installation wizard.

Uninstalling the WebSphere Message Broker Explorer on Linux

You can uninstall the WebSphere Message Broker Explorer by using the graphical installation wizard.

About this task

Complete the following steps to uninstall the WebSphere Message Broker Explorer by using the graphical installation wizard on Linux. To uninstall successfully on Linux systems, you must have write permissions to the installation directory. You might also need root administrator privileges to complete this task.

Procedure

1. Navigate to the uninstall directory. The default location on Linux is `/opt/IBM/MBExplorer/Uninstall_Message_Broker_Explorer`. You can use your native file explorer to run the file in this directory.
2. Run the `./Uninstall_Message_Broker_Explorer` file. The uninstaller launches.
3. Click **Uninstall** to uninstall the WebSphere Message Broker Explorer. The uninstaller lists any items that cannot be uninstalled. You can delete these items manually.

4. When the uninstall process is completed, click **Done** to close the uninstaller.

Related concepts:

"Uninstalling the WebSphere Message Broker Explorer" on page 346
You can uninstall the WebSphere Message Broker Explorer by using the graphical installation wizard, by using the console mode of the installation wizard, or by using the installation wizard in silent mode.

Related tasks:

"Uninstalling" on page 331
Remove the Broker component, the WebSphere Message Broker Toolkit, or the WebSphere Message Broker Explorer from your computer.

"Uninstalling the WebSphere Message Broker Explorer on Windows" on page 347
You can uninstall the WebSphere Message Broker Explorer by using the graphical installation wizard.

Uninstalling the WebSphere Message Broker Explorer in console mode

You can uninstall the WebSphere Message Broker Explorer using the console mode of the installation wizard.

About this task

Use the following steps to uninstall the WebSphere Message Broker Explorer using the console mode of the installation wizard on Windows and Linux. To uninstall successfully on Linux systems, you need write permissions to the installation directory. You are likely to require root administrator privileges for this task.

Procedure

1. Navigate to the Uninstall directory:
   - The default location on Windows is `C:\Program Files\IBM\MBExplorer\Uninstall_Message_Broker_Explorer`
   - The default location on Linux is `/opt/IBM/MBExplorer/Uninstall_Message_Broker_Explorer`. You can also use your native file explorer to run this file.
2. Run the following command: A new uninstaller command window launches:
   - On Windows, the command to uninstall the product from the console is: `Uninstall_Message_Broker_Explorer.exe -i console`.
   - On Linux, the command to to uninstall the product from the console is: `./Uninstall_Message_Broker_Explorer -i console`.

   The uninstaller launches.
3. Press Enter to uninstall the WebSphere Message Broker Explorer. The uninstaller lists any items that could not be uninstalled. You can delete these items manually.
4. When the uninstallation has ended, press **Enter** to exit the uninstaller.

Related concepts:

"Uninstalling the WebSphere Message Broker Explorer" on page 346
You can uninstall the WebSphere Message Broker Explorer by using the graphical installation wizard, by using the console mode of the installation wizard, or by using the installation wizard in silent mode.

Related tasks:

"Uninstalling" on page 331
Remove the Broker component, the WebSphere Message Broker Toolkit, or the

WebSphere Message Broker Explorer from your computer.

"Uninstalling the WebSphere Message Broker Explorer on Windows" on page 347
You can uninstall the WebSphere Message Broker Explorer by using the graphical installation wizard.

"Uninstalling the WebSphere Message Broker Explorer on Linux" on page 347
You can uninstall the WebSphere Message Broker Explorer by using the graphical installation wizard.

Uninstalling the WebSphere Message Broker Explorer in silent mode

You can uninstall the WebSphere Message Broker Explorer using the installation wizard in silent mode.

About this task

Use the following steps to uninstall the WebSphere Message Broker Explorer using the silent mode on Windows and Linux.

Procedure

1. On a command line, navigate to the uninstallation directory for WebSphere Message Broker Explorer in the file system. The default uninstallation directory for the WebSphere Message Broker Explorer on Windows is `C:\Program Files\IBM\MBExplorer\Uninstall_Message_Broker_Explorer`. The default uninstallation directory for the WebSphere Message Broker Explorer on Linux systems is `/opt/IBM/MBExplorer/Uninstall_Message_Broker_Explorer`.
2. Enter the following command on a command line to start the uninstaller in silent mode: `Uninstall_Message_Broker_Explorer.exe -i silent` If you run the silent uninstallation command directly from a command line, the command prompt returns immediately, and the installation wizard completes the uninstallation as a background task. If you run the silent uninstallation command as part of a batch file or script, control is returned to the batch file or script after the uninstallation has completed.

Results

The WebSphere Message Broker Explorer files have been removed from your system.

Related concepts:

"Uninstalling the WebSphere Message Broker Explorer" on page 346
You can uninstall the WebSphere Message Broker Explorer by using the graphical installation wizard, by using the console mode of the installation wizard, or by using the installation wizard in silent mode.

Related tasks:

"Uninstalling" on page 331
Remove the Broker component, the WebSphere Message Broker Toolkit, or the WebSphere Message Broker Explorer from your computer.

"Uninstalling the WebSphere Message Broker Explorer on Windows" on page 347
You can uninstall the WebSphere Message Broker Explorer by using the graphical installation wizard.

"Uninstalling the WebSphere Message Broker Explorer on Linux" on page 347
You can uninstall the WebSphere Message Broker Explorer by using the graphical installation wizard.

Uninstalling service from the WebSphere Message Broker Explorer

You must reinstall the WebSphere Message Broker Explorer to revert to a previous service level.

About this task

You cannot remove individual fixes that you have applied to the WebSphere Message Broker Explorer. If you want to restore a system to a previous service level, you must reinstall the desired level of service. If you have a single installation on your computer, create and retain a backup image, or keep previous GA or fix pack images or media, in case of problems after you have installed service. To restore a Windows or Linux system on which you have an installation of the WebSphere Message Broker Explorer to a previous service level, you have two options:

- Uninstall the current level, then reinstall the previous service level of the WebSphere Message Broker Explorer.
- Install the previous service level of the WebSphere Message Broker Explorer to a new location on your system.

To uninstall the current level of service from the WebSphere Message Broker Explorer, and return to a previous level use the following instructions:

Procedure

1. Uninstall the WebSphere Message Broker Explorer.
2. Reinstall the GA level of the WebSphere Message Broker Explorer.
3. Install the desired level of fix pack.

Related concepts:

"Uninstalling the WebSphere Message Broker Explorer" on page 346
You can uninstall the WebSphere Message Broker Explorer by using the graphical installation wizard, by using the console mode of the installation wizard, or by using the installation wizard in silent mode.

Related tasks:

"Applying service to the WebSphere Message Broker Explorer" on page 329
You can apply maintenance or fixes to the WebSphere Message Broker Explorer.

"Uninstalling" on page 331
Remove the Broker component, the WebSphere Message Broker Toolkit, or the WebSphere Message Broker Explorer from your computer.

"Uninstalling the WebSphere Message Broker Explorer on Windows" on page 347
You can uninstall the WebSphere Message Broker Explorer by using the graphical installation wizard.

"Uninstalling the WebSphere Message Broker Explorer on Linux" on page 347
You can uninstall the WebSphere Message Broker Explorer by using the graphical installation wizard.

Chapter 5. Security

Security is an important consideration for both developers of WebSphere Message Broker applications, and for system administrators configuring WebSphere Message Broker authorities.

When you are designing a WebSphere Message Broker application, it is important to consider the security measures that are needed to protect the information in the system.

The "Security overview" introduces the concepts that you need to understand before you set up security for WebSphere Message Broker.

The following topics explain the different aspects of security that you need to consider when you are designing your applications:
- "Broker administration security" on page 361
- "Message flow security" on page 382
- "Broker component security" on page 497
- "Setting up z/OS security" on page 556
- "Activating broker administration security for WebSphere MQ Version 7.1, or later" on page 381

Related concepts:

"What's new in Version 7.0?" on page 7
Learn about the main new functions in IBM WebSphere Message Broker Version 7.0.

"WS-Security" on page 765
Web Services Security (WS-Security) describes enhancements to SOAP messaging to provide quality of protection through message integrity, message confidentiality, and single message authentication. WS-Security mechanisms can be used to accommodate a wide variety of security models and encryption technologies.

Related reference:

"Security requirements for administrative tasks" on page 3644
You can configure access permissions to govern which users and groups can manipulate objects in the broker network. Security requirements for administrative tasks depend on the platform that you use.

Security overview

When you are designing a WebSphere Message Broker application it is important to consider the security measures that are needed to protect the information in the system.

An important aspect of securing an enterprise system is the ability to protect the information that is passed from third parties. This capability includes restricting access to WebSphere MQ and JMS queues. For information about the steps involved, refer to the documentation supplied by your transport provider. If you are using HTTPS, you need to set specific properties in the HTTP nodes. For information about this option, see "HTTPInput node" on page 4474, "HTTPRequest node" on page 4488, and "HTTPReply node" on page 4484.

In addition to securing the transport, you can secure individual messages based on their identity. For more information about securing messages in a message flow, see “Message flow security” on page 382.

Some security configuration is required to enable WebSphere Message Broker to work correctly and to protect the information in the system. For example, you can secure the messaging transport with SSL connections, restrict access to queues, apply WS-Security to Web services, and secure access to message flows.

In addition, system administrators need WebSphere Message Broker authorities that allow them to perform customization and configuration tasks, run utilities, perform problem determination, and collect diagnostic materials.

The following topics introduce the concepts that you need to understand before you set up security for WebSphere Message Broker:

- “Planning for security when you install WebSphere Message Broker” on page 353
- “Broker administration security overview” on page 362
- “Message flow security overview” on page 383
- “WS-Security” on page 765
- “Authorization for configuration tasks” on page 353
- “Security exits” on page 354
- “Public key cryptography” on page 354
- “Digital certificates” on page 356
- “Digital signatures” on page 360

Additionally, if WebSphere MQ Version 7.1, or later, has been selected for the queue manager and channel auth security is required to be enabled, see “Activating broker administration security for WebSphere MQ Version 7.1, or later” on page 381.

Related concepts:

“Message flow security overview” on page 383
WebSphere Message Broker provides a security manager, which enables you to control access to individual messages in a message flow, using the identity of the message.

“WS-Security” on page 765
Web Services Security (WS-Security) describes enhancements to SOAP messaging to provide quality of protection through message integrity, message confidentiality, and single message authentication. WS-Security mechanisms can be used to accommodate a wide variety of security models and encryption technologies.

“What's new in Version 7.0?” on page 7
Learn about the main new functions in IBM WebSphere Message Broker Version 7.0.

Related tasks:

“Broker component security” on page 497
You must consider several security aspects when you are setting up brokers running on Windows, Linux, or UNIX platforms.

“Setting up z/OS security” on page 556
On z/OS, you must complete several security configuration tasks before WebSphere Message Broker can work correctly.

“Setting up broker administration security” on page 368
Control the actions that users can request against a broker and its resources.

Related reference:

"Security requirements for administrative tasks" on page 3644
You can configure access permissions to govern which users and groups can manipulate objects in the broker network. Security requirements for administrative tasks depend on the platform that you use.

Planning for security when you install WebSphere Message Broker

The Installation Guide describes the security tasks that you must complete before, during, and after installation.

About this task

On Linux and UNIX systems, you must complete security tasks before you install WebSphere Message Broker; these tasks are described in the Installation Guide. On Windows systems, security tasks are completed during and after installation.

Always refer to the Installation Guide for the latest information about installation tasks.

After installation, refer to "Creating user IDs" on page 498 for further security considerations.

For an introduction to various aspects of security, see "Security overview" on page 351.

Related reference:

"Installation Guide" on page 233
Installation information for WebSphere Message Broker is provided in the *Installation Guide* that is supplied as a PDF file with your product package.

Authorization for configuration tasks

Authorization is the process of granting or denying access to a system resource.

For WebSphere Message Broker, authorization is concerned with controlling who has permission to access WebSphere Message Broker resources, and ensuring that users who attempt to work with those resources have the necessary authorization to do so.

Examples of tasks that require authorization are:

- Configuring a broker using, for example, the "**mqsicreatebroker** command" on page 3831.
- Accessing queues, for example, putting a message to the input queue of a message flow.
- Taking actions within the WebSphere Message Broker Toolkit, for example, deploying a message flow to an execution group.

Related tasks:

"Broker component security" on page 497
You must consider several security aspects when you are setting up brokers running on Windows, Linux, or UNIX platforms.

Related reference:

"Security requirements for administrative tasks" on page 3644
You can configure access permissions to govern which users and groups can manipulate objects in the broker network. Security requirements for administrative tasks depend on the platform that you use.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

Security exits

Use security exit programs to verify that the partner at the other end of a connection is genuine.

When you connect from the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer to a broker on another computer, a security exit is not started by default to monitor the connection. If you want to protect access to the broker from client programs, you can use the WebSphere MQ security exit facility.

You can enable a security exit at each end of the connection between your client session and the broker:

- Set up a security exit on the channel at the broker end. This security exit has no special requirements; you can provide a standard security exit.
- Set up a security exit in the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer. Identify the security exit properties when you connect to the broker. The security exit is a standard WebSphere MQ security exit, written in Java.

For an overview of security exits and details of their implementation, see "Channel security exit programs" in the *Intercommunication* section of the WebSphere MQ Version 7 Information Center online.

Related concepts:

"Security overview" on page 351
When you are designing a WebSphere Message Broker application it is important to consider the security measures that are needed to protect the information in the system.

Related tasks:

"Using security exits" on page 555
Define a security exit on the WebSphere MQ channel when you create a broker connection.

"Broker component security" on page 497
You must consider several security aspects when you are setting up brokers running on Windows, Linux, or UNIX platforms.

Related reference:

"Security requirements for administrative tasks" on page 3644
You can configure access permissions to govern which users and groups can manipulate objects in the broker network. Security requirements for administrative tasks depend on the platform that you use.

Related information:

WebSphere MQ Version 7 Information Center online

Public key cryptography

All encryption systems rely on the concept of a key. A key is the basis for a transformation, usually mathematical, of an ordinary message into an unreadable message. For centuries, most encryption systems have relied on private key encryption. Public key encryption is the only challenge to private key encryption that has appeared in the last 30 years.

Private key encryption

Private key encryption systems use a single key that is shared between the sender and the receiver. Both must have the key; the sender encrypts the message by using the key, and the receiver decrypts the message with the same key. Both the sender and receiver must keep the key private to keep their communication private. This kind of encryption has characteristics that make it unsuitable for widespread general use:

- Private key encryption requires a key for every pair of individuals who need to communicate privately. The necessary number of keys rises dramatically as the number of participants increases.
- Keys must be shared between pairs of communicators, therefore the keys must be distributed to the participants. The need to transmit secret keys makes them vulnerable to theft.
- Participants can communicate only by prior arrangement. You cannot send a usable encrypted message to someone spontaneously. You and the other participant must make arrangements to communicate by sharing keys.

Private key encryption is also called symmetric encryption because the same key is used to encrypt and decrypt the message.

Public key encryption

Public key encryption uses a pair of mathematically-related keys. A message that is encrypted with the first key must be decrypted with the second key, and a message that is encrypted with the second key must be decrypted with the first key.

Each participant in a public key system has a pair of keys. One key is nominated as the private key and is kept secret. The other key is distributed to anyone who wants it; this key is the public key.

Anyone can encrypt a message by using your public key, but only you can read it. When you receive the message, you decrypt it by using your private key.

Similarly, you can encrypt a message for anyone else by using their public key, and they decrypt it by using their private key. You can then send the message safely over an unsecured connection.

This kind of encryption has characteristics that make it very suitable for general use:

- Public key encryption requires only two keys per participant.
- The need for secrecy is more easily met: only the private key needs to be kept secret, and because it does not need to be shared, it is less vulnerable to theft in transmission than the shared key in a symmetric key system.
- Public keys can be published, which eliminates the need for prior sharing of a secret key before communication. Anyone who knows your public key can use it to send you a message that only you can read.

Public key encryption is also called asymmetric encryption, because the same key cannot be used to encrypt and decrypt the message. Instead, one key of a pair is used to undo the work of the other.

With symmetric key encryption, beware of stolen or intercepted keys. In public key encryption, where anyone can create a key pair and publish the public key, the challenge is in verifying the identity of the owner of the public key. Nothing prevents a user from creating a key pair and publishing the public key under a false name. The listed owner of the public key cannot read messages that are encrypted with that key because the owner does not have the corresponding private key. If the creator of the false public key can intercept these messages, that person can decrypt and read messages that are intended for someone else. To counteract the potential for forged keys, public key systems provide mechanisms for validating public keys and other information with digital certificates and digital signatures.

Public Key Infrastructure (PKI)

PKI is an infrastructure that uses public key technology to allow applications to interact securely. PKI uses public key encryption to provide privacy. In practice, only a small amount of data is encrypted in this way. Typically, a *session key* is used with a symmetric algorithm to transmit the bulk of the data efficiently.

In business transactions, trust is even more important than privacy. PKI uses the private key to allow an application to sign a document. For the recipient to authenticate the sender, it needs a reliable way to obtain the public key for the sender. This public key is provided in the form of a digital certificate, which is mediated by a trusted third party certificate authority (CA).

Related concepts:

"Digital certificates"
Certificates provide a way of authenticating users. Instead of requiring each participant in an application to authenticate every user, third-party authentication relies on the use of digital certificates.

"Digital signatures" on page 360
A digital signature is a number that is attached to a document. For example, in an authentication system that uses public-key encryption, digital signatures are used to sign certificates.

"Policy sets" on page 774
Policy sets and bindings define and configure your WS-Security requirements, supported by WebSphere Message Broker, for the SOAPInput, SOAPReply, SOAPRequest, SOAPAsyncRequest, and SOAPAsyncResponse nodes.

"WS-Security" on page 765
Web Services Security (WS-Security) describes enhancements to SOAP messaging to provide quality of protection through message integrity, message confidentiality, and single message authentication. WS-Security mechanisms can be used to accommodate a wide variety of security models and encryption technologies.

"WS-Security mechanisms" on page 768
The WS-Security specification provides three mechanisms for securing Web services at the message level: authentication, integrity, and confidentiality.

Digital certificates

Certificates provide a way of authenticating users. Instead of requiring each participant in an application to authenticate every user, third-party authentication relies on the use of digital certificates.

A digital certificate is equivalent to an electronic ID card. The certificate serves two purposes:

- Establishes the identity of the owner of the certificate

- Distributes the owner's public key

Certificates are issued by trusted third parties, called certificate authorities (CAs). These authorities can be commercial ventures or local entities, depending on the requirements of your application. The CA is trusted to adequately authenticate users before issuing certificates. A CA issues certificates with digital signatures. When a user presents a certificate, the recipient of the certificate validates it by using the digital signature. If the digital signature validates the certificate, the certificate is recognized as intact and authentic. Participants in an application need to validate certificates only; they do not need to authenticate users. The fact that a user can present a valid certificate proves that the CA has authenticated the user. The designation, "trusted third parties", indicates that the system relies on the trustworthiness of the CAs.

The certificates and private keys are stored in files called *keystores* and *truststores*.

- A keystore holds the private keys and public key certificates for an application.
- A truststore contains the CA certificates required to authenticate certificates that are presented by another application.

Contents of a digital certificate

A certificate contains several pieces of information, including information about the owner of the certificate and the issuing CA. Specifically, a certificate includes:

- The distinguished name (DN) of the owner. A DN is a unique identifier, a fully qualified name including not only the common name (CN) of the owner but also the owner's organization and other distinguishing information.
- The public key of the owner.
- The date on which the certificate is issued.
- The date on which the certificate expires.
- The distinguished name of the issuing CA.
- The digital signature of the issuing CA. The message-digest function creates a signature based upon all the previously listed fields.

The idea of a certificate is that a CA takes the public key of the owner, signs the public key with its own private key, and returns the information to the owner as a certificate. When the owner distributes the certificate to another party, it signs the certificate with its private key. The receiver can extract the certificate that contains the CA signature with the public key of the owner. By using the CA public key and the CA signature on the extracted certificate, the receiver can validate the CA signature. If valid, the public key that is used to extract the certificate is considered good. The owner signature is validated, and if the validation succeeds, the owner is successfully authenticated to the receiver.

The additional information in a certificate helps an application to determine whether to honor the certificate. With the expiration date, the application can determine if the certificate is still valid. With the name of the issuing CA, the application can check that the CA is considered trustworthy by the site.

An application that needs to authenticate itself must provide its personal certificate, the one containing its public key, and the certificate of the CA that signed its certificate, called a signer certificate. In cases where chains of trust are established, several signer certificates can be involved.

Requesting certificates

To get a certificate, send a certificate request to the CA. The certificate request includes:

- The distinguished name of the owner or the user for whom the certificate is requested
- The public key of the owner
- The digital signature of the owner

The message digest function creates a signature based on all the previously listed fields.

The CA verifies the signature with the public key in the request to ensure that the request is intact and authentic. The CA then authenticates the owner. Exactly what the authentication consists of depends on a prior agreement between the CA and the requesting organization. If the owner in the request is authenticated successfully, the CA issues a certificate for that owner.

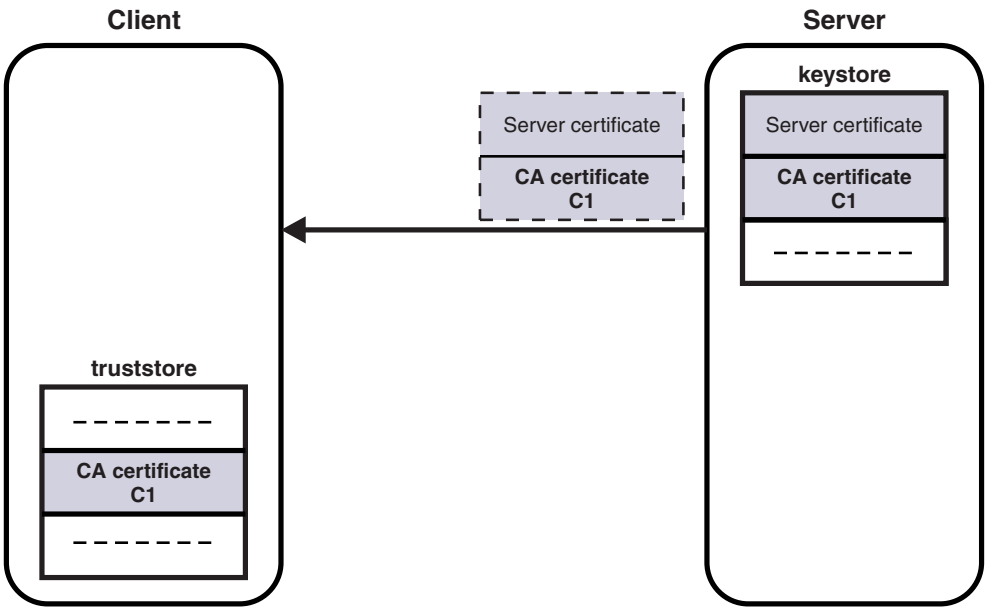
Using certificates: Chain of trust and self-signed certificate

To verify the digital signature on a certificate, you must have the public key of the issuing CA. Public keys are distributed in certificates, therefore you must have a certificate for the issuing CA that is signed by the issuer. One CA can certify other CAs, so a chain of CAs can issue certificates for other CAs, all of whose public keys you need. Eventually, you reach a root CA that issues itself a self-signed certificate. To validate a user certificate, you need certificates for all of the intervening participants back to the root CA. You then have the public keys that you need to validate each certificate, including the user certificate.

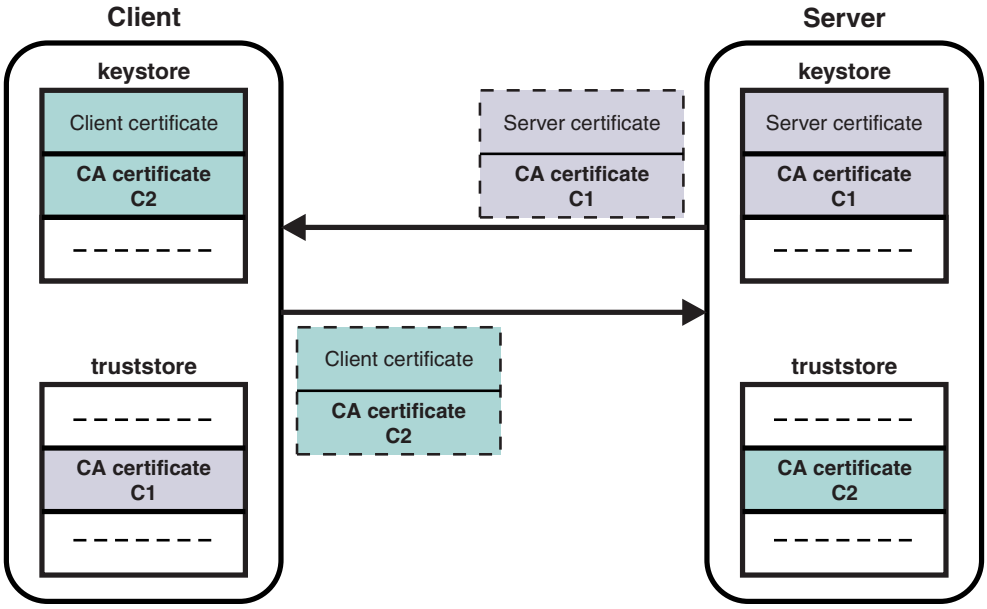
A self-signed certificate contains the public key of the issuer and is signed with the private key. The digital signature is validated like any other, and if the certificate is valid, the public key it contains is used to check the validity of other certificates issued by the CA. However, anyone can generate a self-signed certificate. In fact, you can probably generate self-signed certificates for testing purposes before installing production certificates. The fact that a self-signed certificate contains a valid public key does not mean that the issuer is a trusted certificate authority. To ensure that self-signed certificates are generated by trusted CAs, such certificates must be distributed by secure means; for example, hand-delivered on floppy disks, downloaded from secure sites, and so on.

Applications that use certificates store these certificates in a keystore file. This file typically contains the necessary personal certificates, its signing certificates, and its private key. The private key is used by the application to create digital signatures. Servers always have personal certificates in their keystore files. A client requires a personal certificate only if the client must authenticate to the server when mutual authentication is enabled.

To allow a client to authenticate a server, a server keystore file contains the private key and the certificate of the server and the certificates of its CA. A client truststore file must contain the signer certificates of the CAs of each server, which the client must authenticate. The following diagram illustrates how a client authenticates a server.



If mutual authentication is needed, the client keystore file must contain the client private key and certificate. The server truststore file requires a copy of the certificate of the client CA. The following diagram illustrates mutual authentication.



Related concepts:

"Public key cryptography" on page 354
All encryption systems rely on the concept of a key. A key is the basis for a transformation, usually mathematical, of an ordinary message into an unreadable message. For centuries, most encryption systems have relied on private key encryption. Public key encryption is the only challenge to private key encryption that has appeared in the last 30 years.

"Digital signatures"
A digital signature is a number that is attached to a document. For example, in an authentication system that uses public-key encryption, digital signatures are used to sign certificates.

"Policy sets" on page 774
Policy sets and bindings define and configure your WS-Security requirements, supported by WebSphere Message Broker, for the SOAPInput, SOAPReply, SOAPRequest, SOAPAsyncRequest, and SOAPAsyncResponse nodes.

"WS-Security" on page 765
Web Services Security (WS-Security) describes enhancements to SOAP messaging to provide quality of protection through message integrity, message confidentiality, and single message authentication. WS-Security mechanisms can be used to accommodate a wide variety of security models and encryption technologies.

"WS-Security mechanisms" on page 768
The WS-Security specification provides three mechanisms for securing Web services at the message level: authentication, integrity, and confidentiality.

Digital signatures

A digital signature is a number that is attached to a document. For example, in an authentication system that uses public-key encryption, digital signatures are used to sign certificates.

This signature establishes the following information:

- The integrity of the message: Is the message intact? That is, has the message been modified between the time it was digitally signed and now?
- The identity of the signer of the message: Is the message authentic? That is, was the message signed by the user who claims to have signed it?

A digital signature is created in two steps. The first step distills the document into a large number. This number is the digest code or fingerprint. The digest code is then encrypted, which results in the digital signature. The digital signature is appended to the document from which the digest code is generated.

Several options are available for generating the digest code. This process is not encryption, but a sophisticated checksum. The message cannot regenerate from the resulting digest code. The crucial aspect of distilling the document to a number is that if the message changes, even in a trivial way, a different digest code results. When the recipient gets a message and verifies the digest code by recomputing it, any changes in the document result in a mismatch between the stated and the computed digest codes.

To stop someone from intercepting a message, changing it, recomputing the digest code, and retransmitting the modified message and code, you need a way to verify the digest code as well. To verify the digest code, reverse the use of the public and private keys. For private communication, it makes no sense to encrypt messages with your private key; these keys can be decrypted by anyone with your public key. However, this technique can be useful for proving that a message came from you. No one can create it because no-one else has your private key. If some meaningful message results from decrypting a document by using someone's public key, the decryption process verifies that the holder of the corresponding private key did encrypt the message.

The second step in creating a digital signature takes advantage of this reverse application of public and private keys. After a digest code is computed for a

document, the digest code is encrypted with the sender's private key. The result is the digital signature, which is attached to the end of the message.

When the message is received, the recipient follows these steps to verify the signature:

1. Recomputes the digest code for the message.
2. Decrypts the signature by using the sender's public key. This decryption yields the original digest code for the message.
3. Compares the original and recomputed digest codes. If these codes match, the message is both intact and authentic. If not, something has changed and the message is not to be trusted.

Related concepts:

"Public key cryptography" on page 354
All encryption systems rely on the concept of a key. A key is the basis for a transformation, usually mathematical, of an ordinary message into an unreadable message. For centuries, most encryption systems have relied on private key encryption. Public key encryption is the only challenge to private key encryption that has appeared in the last 30 years.

"Digital certificates" on page 356
Certificates provide a way of authenticating users. Instead of requiring each participant in an application to authenticate every user, third-party authentication relies on the use of digital certificates.

"Policy sets" on page 774
Policy sets and bindings define and configure your WS-Security requirements, supported by WebSphere Message Broker, for the SOAPInput, SOAPReply, SOAPRequest, SOAPAsyncRequest, and SOAPAsyncResponse nodes.

"WS-Security" on page 765
Web Services Security (WS-Security) describes enhancements to SOAP messaging to provide quality of protection through message integrity, message confidentiality, and single message authentication. WS-Security mechanisms can be used to accommodate a wide variety of security models and encryption technologies.

"WS-Security mechanisms" on page 768
The WS-Security specification provides three mechanisms for securing Web services at the message level: authentication, integrity, and confidentiality.

Broker administration security

Broker administration security controls the rights of users to complete administrative tasks for a broker and its resources.

The following topics introduce the concepts that you need to understand before you can set up broker administration security, and explain the steps involved in securing brokers and their resources:

- "Broker administration security overview" on page 362
- "Setting up broker administration security" on page 368

Related concepts:

Chapter 5, "Security," on page 351
Security is an important consideration for both developers of WebSphere Message Broker applications, and for system administrators configuring WebSphere Message Broker authorities.

"Security overview" on page 351
When you are designing a WebSphere Message Broker application it is important to consider the security measures that are needed to protect the information in the system.

"WS-Security" on page 765
Web Services Security (WS-Security) describes enhancements to SOAP messaging to provide quality of protection through message integrity, message confidentiality, and single message authentication. WS-Security mechanisms can be used to accommodate a wide variety of security models and encryption technologies.

Related tasks:

"Setting up z/OS security" on page 556
On z/OS, you must complete several security configuration tasks before WebSphere Message Broker can work correctly.

Broker administration security overview

Broker administration security controls the rights of users to complete administrative tasks for a broker and its resources.

Broker administration security is an optional feature of the broker. When you create a broker, the default setting is that broker administration security is not enabled. You can specify an additional parameter to enable security when you create the broker. You can also change the status of the broker administration security after you have created the broker, and can therefore enable or disable it when appropriate.

When you have enabled broker administration security, set up security control by registering WebSphere MQ permissions for specific user IDs. Permissions are recorded on the following authorization queues that are defined on the broker queue manager:

- SYSTEM.BROKER.AUTH. This queue represents the broker and its properties. Only one queue exists of this name for each broker. This queue is defined as a local queue.
- One SYSTEM.BROKER.AUTH.*EG* for each execution group that you define on the broker, where *EG* is the name of the execution group. These queues are defined as alias queues.

When you create a broker, the queue SYSTEM.BROKER.AUTH is created. Read, write, and execute authorities are automatically granted to the user group mqbrkrs on this queue. This queue is created even if you do not enable security at this time.

If you enable security, the broker checks the authorizations that you have set up on this queue when it receives a request that views or changes its properties or resources. If the user ID associated with the request is not authorized, the broker refuses the request.

When you create an execution group on a broker for which you have enabled security, the execution group authorization queue SYSTEM.BROKER.AUTH.*EG* is created, where *EG* is the name of the execution group. Read, write, and execute authorities are automatically granted to the user group mqbrkrs on this queue.

If you are migrating from Version 6.1 or Version 6.0, and you use Access Control Lists (ACLs) that you define to the Configuration Manager, you cannot migrate your ACLs directly to Version 7.0. Review the guidance provided in "Migrating

Configuration Manager ACLs" on page 179 to understand how to set up security by using the ACLs as a basis for security in your Version 7.0 environment.

See the following topics for more information about permissions and queues:

- "Broker permissions and equivalent WebSphere MQ permissions" on page 365
- "Authorization queues for broker administration security" on page 366

Authorization on z/OS

On z/OS, WebSphere MQ uses the System Authorization Facility (SAF) to route requests for authority checks to an external security manager (ESM) such as the z/OS Security Server Resource Access Control Facility (RACF). WebSphere MQ does no authority checks of its own. All information about broker administration security on z/OS assumes that you are using RACF as your ESM. If you are using a different ESM, you might need to interpret the information provided for RACF in a way that is relevant to your ESM.

If you are activating security on the WebSphere MQ queue manager on z/OS for the first time, you must set up the profiles or other resources that are required by your ESM to access queues. You must also check that the queue manager is configured to access the security profiles in the correct class; MQQUEUE for uppercase queue names and MXQUEUE for mixed case queue names.

For further information about queue manager security and security profiles, see the *z/OS System Administration Guide* and *z/OS System Setup Guide* sections of the WebSphere MQ Version 7 Information Center online.

Authority checking

If you have activated broker administration security, all actions performed by users of the following interfaces are subject to authority checking:

- A WebSphere Message Broker Toolkit session
- A WebSphere Message Broker Explorer session
- A Java program that uses the CMP API to perform operations on the broker
- All the following commands:
  - **mqsichangeresourcestats**
  - **mqsicreateexecutiongroup**
  - **mqsideleteexecutiongroup**
  - **mqsideploy**
  - **mqsilist**
  - **mqsimode**
  - **mqsireloadsecurity**
  - **mqsireportresourcestats**
  - **mqsistartmsgflow**
  - **mqsistopmsgflow**

  For additional authorization required for these commands, see "Commands and authorizations for broker administration security" on page 3646.

  You can run all commands that are not stated here only on the computer on which the broker is running. Either your user ID, or the ID under which the broker is running, must be a member of the security group mqbrkrs when you run the unlisted commands. Each command topic describes the authority that is required.

Users of the WebSphere Message Broker Explorer and WebSphere Message Broker Toolkit who do not have read, write, and execute authority for the broker or execution groups, have only restricted access to those resources. An icon is

displayed against each resource to indicate that user authority is restricted. The actions that the user can request against a resource are determined by the restricted authority that is in place for that user.

Authority persistence

If a user ID is granted authority to access a broker, the access is retained at least until the current connection or session is ended by one of the following events:

- The user closes the WebSphere Message Broker Explorer or WebSphere Message Broker Toolkit session
- The CMP API application disconnects from the `BrokerProxy` object

Even if you update or remove the authority for this user ID, the authorization does not change while the connection is active.

However, authorization is always checked for operations against execution groups and message flows; if you change or remove the authorization for a user ID to an execution group, subsequent requests in the current connection might fail.

Additional administration security

In your environment, a check on the user ID making a request might not provide a sufficient level of security. If you require a more secure solution, one or both of the following options are available:

- You can enable SSL on a WebSphere MQ client connection between the source of the request (for example, the WebSphere Message Broker Explorer) and the target queue manager on which the broker is running.
- You can configure your WebSphere MQ network so that certain types of users can be directed through a specific server connection (SVRCONN) channel, provided they comply with the CHLAUTH rules. For details of channel security, see the *System Administration Guide* section in the WebSphere MQ Version 7 Information Center online.

Related concepts:

"Security overview" on page 351
When you are designing a WebSphere Message Broker application it is important to consider the security measures that are needed to protect the information in the system.

Related tasks:

"Migrating Configuration Manager ACLs" on page 179
If you are migrating from WebSphere Message Broker , WebSphere Message Broker Version 6.0, or WebSphere Event Broker Version 6.0, you can use the Access Control Lists (ACLs) that you set up in the Configuration Manager as the basis for your security model in Version 7.0.

"Enabling broker administration security" on page 368
Enable broker administration security on a broker to control which users can complete specific tasks against that broker and its resources.

"Authorizing users for broker administration" on page 371
Grant authority to one or more groups or users to authorize them to complete specific tasks against a broker and its resources.

"Enabling SSL on the WebSphere MQ Java Client" on page 540
The WebSphere MQ Java Client supports SSL-encrypted connections over the server-connection (SVRCONN) channel between an application and the queue manager. Configure SSL support for connections between applications that use the CMP API (including the WebSphere Message Broker Toolkit and the WebSphere Message Broker Explorer) and a broker.

"Setting up broker administration security" on page 368
Control the actions that users can request against a broker and its resources.

Related reference:

"Commands" on page 3672
All WebSphere Message Broker Toolkit and runtime commands that are provided on distributed systems are listed, grouped by function, with references to command details.

Related information:

WebSphere MQ Version 7 Information Center online

Broker permissions and equivalent WebSphere MQ permissions

If you have enabled broker administration security, you can give different permissions to user IDs to allow them to complete various actions against a broker or its resources.

When a user requests an action against a broker or an execution group, the broker accesses the appropriate authorization queues to check that the user ID has the correct authority for that action against the target resource.

Permission to perform a broker administration task is mapped to a WebSphere MQ authority associated with the relevant authorization queue, and is created and maintained by the broker administrator. The mapping from broker permission to WebSphere MQ permission is shown in the following table.

Broker permission	WebSphere MQ permission
Read	Inquire
Write	Put
Execute	Set

For information about the authorizations that are required for specific tasks, see "Tasks and authorizations for broker administration security" on page 3645.

WebSphere MQ specific and generic profiles

WebSphere MQ supports both specific and generic profiles to manage WebSphere MQ permissions. When you enable broker administration security, you can create specific profiles to define WebSphere MQ permissions on SYSTEM.BROKER.AUTH and on one or more SYSTEM.BROKER.AUTH.*EG* queues (where *EG* is the name of a specific execution group).

You might want to grant a user, or group of users, authority to a number of execution groups, or perhaps all execution groups. You can use a WebSphere MQ generic profile to grant authority in this way. A generic profile defines authority to an existing set of execution groups, and all additional groups, that match the profile. A generic profile is one that uses special characters (wildcard characters) in the profile name, such as asterisks (*).

For example, if you want to create a generic profile to authorize access to all execution groups defined on the broker, you can specify SYSTEM.BROKER.AUTH.**. If you want a profile for a set of execution groups with names that all start with the same character string, you can specify SYSTEM.BROKER.AUTH.TEST**.

For more information about WebSphere MQ generic profile wildcard characters, see “Using wildcard characters”, and for information about WebSphere MQ generic profile priorities, see “Profile priorities”. You can find both topics in the *System Administration Guide* section of the WebSphere MQ Version 7 Information Center online.

Related concepts:

“Security overview” on page 351
When you are designing a WebSphere Message Broker application it is important to consider the security measures that are needed to protect the information in the system.

“Authorization queues for broker administration security”
If you have enabled broker administration security, the broker examines specific queues to determine if a user has the authority to complete a particular task against a broker or its resources.

Related tasks:

“Enabling broker administration security” on page 368
Enable broker administration security on a broker to control which users can complete specific tasks against that broker and its resources.

“Authorizing users for broker administration” on page 371
Grant authority to one or more groups or users to authorize them to complete specific tasks against a broker and its resources.

Related reference:

“Tasks and authorizations for broker administration security” on page 3645
If you have enabled broker administration security, users require specific authority so that they can complete administration tasks.

Related information:

WebSphere MQ Version 7 Information Center online

Authorization queues for broker administration security

If you have enabled broker administration security, the broker examines specific queues to determine if a user has the authority to complete a particular task against a broker or its resources.

When you create a broker, the queue SYSTEM.BROKER.AUTH is created. Read, write, and execute authorities are automatically granted to the user group mqbrkrs on this queue. This queue is created even if you do not enable security at this time.

The SYSTEM.BROKER.AUTH queue is created as a local queue, and is used to define which users are authorized to perform actions on the broker and the broker properties.

When you create an execution group on a broker for which you have enabled security, the execution group authorization queue SYSTEM.BROKER.AUTH.*EG* is created. where *EG* is the name of the execution group. Read, write, and execute authorities are automatically granted to the user group mqbrkrs on this queue. The dedicated execution group queues are created as aliases to the queue SYSTEM.BROKER.AUTH.

If you create a broker without administration security, you can change it later. If you have defined one or more execution groups on that broker when you change its security setting, the required execution group authorization queues are defined.

A queue can be created only by a user ID that is a member of the WebSphere MQ security group mqm. Therefore the user ID who creates a broker, changes a broker, and the ID under which the broker is running when an execution group is created, must be a member of that security group. If the user ID does not have this authority, a message is returned to the command (for the **mqsichangebroker** command only), or written to the system log, with the error and the name of the queue. You must create the queue yourself, or ask your WebSphere MQ administrator to create it for you.

WebSphere MQ restricts the length of a queue name to 48 characters. Queue name characters must be in the En_US ASCII character set, and contain only uppercase and lowercase letters, digits, and the following special characters; period (.), forward slash (/), underscore (_), and percent (%). If the name of your execution group includes a character that is not valid, that character is replaced in the WebSphere MQ queue name by an underscore character. For example, if you create an execution group with the name `test@environment`, the authorization queue is created with the name SYSTEM.BROKER.AUTH.test_environment.

If you are running a secure environment, limit the names of your execution groups to 29 characters. This limit ensures that the authorization queue names generated, which include the prefix SYSTEM.BROKER.AUTH, do not exceed the WebSphere MQ limit of 48 characters.

If your execution group names do not all conform to the length and character requirements, execution groups with similar names might result in a shared authorization queue. If this situation occurs, a warning message is returned to the user that issued the command, or is written to the system log, when the second execution group is created to state that the queue is shared.

When you delete an execution group, its associated authorization queue is retained. The queue is deleted if you specify the appropriate parameter when you delete the broker. The queue can be reused if you re-create the execution group, but you must check the authorities that you have defined on the queue to ensure that they are still valid.

If you rename an execution group, you must first create an authorization queue with the appropriate name. You must also re-create the WebSphere MQ permissions associated with the original authorization queue on this queue before you rename the execution group; the broker does not perform this task on your behalf. The broker rejects the rename request if the authorization queue does not exist, to ensure that security is not affected by the renaming. If you do not re-create these permissions, no user IDs are authorized to perform a task against the renamed execution group.

When you delete a broker, you can specify that all its authorization queues are also deleted; they are not deleted by default. If you specify that the queue manager is deleted at this time, all queues are deleted.

Related concepts:

"Security overview" on page 351
When you are designing a WebSphere Message Broker application it is important to consider the security measures that are needed to protect the information in the system.

"Broker permissions and equivalent WebSphere MQ permissions" on page 365
If you have enabled broker administration security, you can give different permissions to user IDs to allow them to complete various actions against a broker

or its resources.

Related tasks:

"Enabling broker administration security"
Enable broker administration security on a broker to control which users can complete specific tasks against that broker and its resources.

"Authorizing users for broker administration" on page 371
Grant authority to one or more groups or users to authorize them to complete specific tasks against a broker and its resources.

Related information:

WebSphere MQ Version 7 Information Center online

Setting up broker administration security

Control the actions that users can request against a broker and its resources.

About this task

You can activate administrative control for an individual broker when you create it. If appropriate, you can enable the security after creation by changing broker properties. When you have activated broker administration security, grant users or groups authority to complete their expected tasks.

- "Enabling broker administration security"
- "Authorizing users for broker administration" on page 371
- "Disabling broker administration security" on page 380

On z/OS, WebSphere MQ uses the System Authorization Facility (SAF) to route requests for authority checks to an external security manager (ESM) such as the z/OS Security Server Resource Access Control Facility (RACF). WebSphere MQ does no authority checks of its own. All information about broker administration security on z/OS assumes that you are using RACF as your ESM. If you are using a different ESM, you might need to interpret the information provided for RACF in a way that is relevant to your ESM.

Related concepts:

"Security overview" on page 351
When you are designing a WebSphere Message Broker application it is important to consider the security measures that are needed to protect the information in the system.

"Broker administration security overview" on page 362
Broker administration security controls the rights of users to complete administrative tasks for a broker and its resources.

Related reference:

"Tasks and authorizations for broker administration security" on page 3645
If you have enabled broker administration security, users require specific authority so that they can complete administration tasks.

"Commands and authorizations for broker administration security" on page 3646
If you have enabled broker administration security, users require specific authority to be able to run the administration commands.

Enabling broker administration security

Enable broker administration security on a broker to control which users can complete specific tasks against that broker and its resources.

About this task

You can optionally enable administration security for a broker when you create it. If you decide to enable administrative security after you have created the broker, you can change the appropriate broker property.

When you create or change a broker, your user ID must be a member of the WebSphere MQ control group mqm.

Procedure

1. To enable broker administration security when you create the broker, select the security option in the "Create broker" wizard in WebSphere Message Broker Explorer, or specify the parameter **-s active** on the **mqsicreatebroker** command. For example, to create a broker called MB7BROKER with security enabled on AIX, enter the following command:

   ```
   mqsicreatebroker MB7BROKER -q MB7QMGR -s active
   ```

   The broker creates the authorization queue SYSTEM.BROKER.AUTH. This queue is used to define which users are authorized to perform an action on the broker.

   The broker also assigns default permissions of inquire, put, and set authority to this queue. These permissions grant read, write, and execute authority on the broker to all members of the mqbrkrs group. Therefore, you must ensure that at least one member of your broker administration team is a member of this group. You must also manage the membership of this group with care, and ensure that this level of authorization is granted only to users who require it.

   On z/OS, these permissions are implemented as levels of authority in the external security manager (ESM) that you are using with WebSphere MQ. If you are using RACF as your ESM, the levels are hierarchical: for example, ALTER access implies READ and WRITE access. You must therefore check the documentation for your ESM to understand the authorization levels that it supports. On distributed platforms, no equivalent hierarchy exists, and the three permissions are independent.
2. To enable broker administration security on an existing broker:
   a. Stop the broker in the WebSphere Message Broker Explorer, or run the **mqsistop** command.
   b. Select the security option for this broker in the WebSphere Message Broker Explorer, or run the **mqsichangebroker** command, specifying the parameter **-s active**. For example, to enable security for the broker MB7BROKER, enter the following command:

      ```
      mqsichangebroker MB7BROKER -s active
      ```

      The broker creates a queue for each defined execution group, with a name that conforms to the format SYSTEM.BROKER.AUTH.*EG*, where *EG* is the name of the execution group. It assigns default permissions of inquire, put, and set authority to the queue, which grants read, write, and execute access to the execution group and its properties, for the mqbrkrs group. These queues, and the broker authorization queue SYSTEM.BROKER.AUTH, are now ready for use.

      The names of queues that are generated for your execution groups might not match exactly the name of the execution groups, because WebSphere MQ enforces some restrictions on the authorization queue names. For details of these restrictions, and the possible effects, see "Authorization queues for broker administration security" on page 366.

c. Start the broker in the WebSphere Message Broker Explorer, or run the **mqsistart** command.

3. Check that the user ID under which your broker is running is a member of the WebSphere MQ security group mqm. Without this authority, the broker cannot create or delete the authorization queues for execution groups at run time.

   Because mqm authority grants full access control to all WebSphere MQ resources, you might not want your broker running with this level of authority. If you do not want the broker to run with mqm authority, you must work with your WebSphere MQ administrator to ensure that the required queues are created (and deleted) at the appropriate time.

   If you want to give your broker mqm authority:

   - On Linux and UNIX systems, add to mqm the user ID that started the broker.
   - On Windows, add to mqm the user ID that you specified as the service user ID. When you add this user ID, the same level of authority is granted to all user IDs defined in the same primary group. You must therefore control carefully your group memberships to ensure that access is not granted to user IDs that do not require it.
   - On z/OS, grant equivalent permissions to the started task user ID.

4. Check also that the user ID associated with the broker, defined in the previous step, has WebSphere MQ altuser authority. This authority is required by the broker to request WebSphere MQ to check authorities.

   Display registry entries for a broker by using **mqsireportbroker** *brokerName*.

What to do next

Next: Grant authority to users to reflect what tasks you want them to be able to complete, by populating the queues with the appropriate details. This task is described in "Authorizing users for broker administration" on page 371.

Related concepts:

"Security overview" on page 351
When you are designing a WebSphere Message Broker application it is important to consider the security measures that are needed to protect the information in the system.

"Broker permissions and equivalent WebSphere MQ permissions" on page 365
If you have enabled broker administration security, you can give different permissions to user IDs to allow them to complete various actions against a broker or its resources.

Related tasks:

"Authorizing users for broker administration" on page 371
Grant authority to one or more groups or users to authorize them to complete specific tasks against a broker and its resources.

"Disabling broker administration security" on page 380
Disable broker administration security to remove control over a broker and its resources.

"Starting and stopping a broker" on page 921
Run the appropriate command to start or stop a broker.

Related reference:

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

"**mqsireportbroker** command" on page 3919
Use the **mqsireportbroker** command to display broker registry entries.

"Tasks and authorizations for broker administration security" on page 3645
If you have enabled broker administration security, users require specific authority so that they can complete administration tasks.

"Commands and authorizations for broker administration security" on page 3646
If you have enabled broker administration security, users require specific authority to be able to run the administration commands.

Related information:

WebSphere MQ Version 7 Information Center online

Authorizing users for broker administration

Grant authority to one or more groups or users to authorize them to complete specific tasks against a broker and its resources.

About this task

You must activate broker administration security before you can grant or revoke permissions for specific user IDs. When you first activate security, you must set up initial control for each user to authorize them to perform certain operations. You can then grant additional authority, revoke permissions, as required.

You can grant authorities to individual principals (user IDs), to groups of users, or both, on all platforms:

- If you grant a group or a user ID authority at the broker level (on queue SYSTEM.BROKER.AUTH), it does not inherit authority for execution groups. You must explicitly grant authority to all, or to individual, execution groups.
- On Linux and UNIX, you can authorize both principals and groups, but if you specify a principal, you must grant authority to the primary group to which that principal belongs.
- If a user ID is a member of the WebSphere MQ security group mqm, it automatically has authority to all WebSphere MQ objects.
- On Windows, if a user ID is a member of the security group Administrators, it automatically has authority to all WebSphere MQ objects.

When you change authorizations on a queue, the broker accesses the updated values the next time that a request is processed. You do not have to stop and restart the broker.

If you update user ID or group membership by using the operating system facilities on the platform on which the broker queue manager is running, you must ensure that the queue manager is aware of these changes. Select the option **Refresh Authorization Service** in the WebSphere MQ Explorer to notify the queue manager of the updated status.

The authority that is required depends on the requirements of the user:

- "Required authority for administrators" on page 372
- "Required authority for users who connect to the broker" on page 372
- "Required authority for developers" on page 373

The way in which you set up the required authorities differs by platform:

- "Granting and revoking authority on Linux, UNIX, and Windows systems" on page 374
- "Granting and revoking authority on z/OS systems" on page 376

These platform-specific topics also give examples of command usage for viewing what authorizations are in place by using the WebSphere MQ **dspmqaut** command, and for dumping this information by using the **dmpmqaut** command.

Required authority for administrators:
About this task

When you activate administration security, the WebSphere MQ permissions for inquire, put, and set are granted for the group mqbrkrs for the queue SYSTEM.BROKER.AUTH. These permissions grant read, write, and execute authority on the broker and its properties to all user IDs that are members of mqbrkrs.

If you want additional user IDs to have administrator authorization, either add those IDs to the group mqbrkrs, or add WebSphere MQ permissions for inquire, put, and set for those user IDs to this queue.

The following table summarizes the WebSphere MQ permissions that are required:

Object	Name	Permissions
Queue manager	The queue manager associated with the broker; for example, MB7QMGR	Connect Inquire
Queue	SYSTEM.BROKER.DEPLOY.QUEUE	Put
Queue	SYSTEM.BROKER.DEPLOY.REPLY	Get Put
Queue	SYSTEM.BROKER.AUTH	Inquire Put Set
Queue	SYSTEM.BROKER.AUTH.*EG*	Inquire Put Set

For more information about permissions on authorization queues, see "Broker permissions and equivalent WebSphere MQ permissions" on page 365 and "Tasks and authorizations for broker administration security" on page 3645.

Required authority for users who connect to the broker:
About this task

If a user or application wants to connect to a broker, you must grant them the appropriate permissions. All applications written to the CMP API, and users of the WebSphere Message Broker Explorer and the WebSphere Message Broker Toolkit, require permissions based on their expected actions. The following table shows the WebSphere MQ permissions that are required:

Object	Name	Permissions
Queue manager	The queue manager associated with the broker; for example, MB7QMGR	Connect Inquire
Queue	SYSTEM.BROKER.DEPLOY.QUEUE	Put
Queue	SYSTEM.BROKER.DEPLOY.REPLY	Get Put
Queue	SYSTEM.BROKER.AUTH	Inquire[1]
Queue	SYSTEM.BROKER.AUTH.*EG*	Inquire[1]

Notes:

1. Users and applications can connect to the broker without this level of authority, but are unable to request actions against the broker, including viewing properties.

Required authority for developers:
About this task

If your users are working with existing execution groups, and development resources such as BAR files, add the WebSphere MQ permissions inquire, put, and set for those users to one or more SYSTEM.BROKER.AUTH.*EG* queues (where *EG* is the name of the execution group).

When you run a broker with administration security enabled, you might need to restrict the names and the length of the names that you give to your execution groups, because WebSphere MQ enforces some restrictions on the authorization queue names. For details of these restrictions, and the possible effects, see "Authorization queues for broker administration security" on page 366.

Related concepts:

"Security overview" on page 351
When you are designing a WebSphere Message Broker application it is important to consider the security measures that are needed to protect the information in the system.

"Broker administration security overview" on page 362
Broker administration security controls the rights of users to complete administrative tasks for a broker and its resources.

Related tasks:

"Enabling broker administration security" on page 368
Enable broker administration security on a broker to control which users can complete specific tasks against that broker and its resources.

"Granting and revoking authority on Linux, UNIX, and Windows systems" on page 374
Grant or revoke authority to one or more groups or users to complete specific tasks against a broker running on Linux, UNIX, or Windows.

"Granting and revoking authority on z/OS systems" on page 376
Grant or revoke authority to one or more groups or users to complete specific tasks against a broker running on z/OS.

"Disabling broker administration security" on page 380
Disable broker administration security to remove control over a broker and its resources.

Related information:

WebSphere MQ Version 7 Information Center online

Granting and revoking authority on Linux, UNIX, and Windows systems:

Grant or revoke authority to one or more groups or users to complete specific tasks against a broker running on Linux, UNIX, or Windows.

Before you begin

Before you start:

Activate broker administration security for the broker before you grant and revoke authority for requests sent to that broker.

About this task

Use WebSphere MQ commands to set up and manage your required security levels. If you prefer, you can make authorization changes to the security queues by using the WebSphere MQ Explorer.

For security reasons, it is important that authorities are set correctly. The **setmqaut** command grants and revokes authorities cumulatively. Therefore, to avoid retaining unwanted pre-existing authorities, it is helpful to set authorities explicitly on each **setmqaut** command, rather than granting and revoking individual authorities. Granting and revoking is achieved by specifying "`-all`" (to remove all authorities) followed by the required authorities.

The following command grants execute authority and retains any pre-existing authorities:

```
setmqaut -m test -t queue -n SYSTEM.BROKER.AUTH -g group1 +set
```

The following command grants execute authority only and does not retain pre-existing authorities:

```
setmqaut -m test -t queue -n SYSTEM.BROKER.AUTH -g group1 -all +set
```

Multiple authorities can also be set in this manner. For example, the following command grants execute and write authorities only (and not retain pre-existing authorities):

```
setmqaut -m test -t queue -n SYSTEM.BROKER.AUTH -g group1 -all +set +put
```

It is also helpful to use the **dspmqaut** command after each **setmqaut** command, to check that authorities have been correctly set.

For further information about the commands shown in the following examples, and for details of the parameters, see the WebSphere MQ Version 7 Information Center online.

Examples

All the examples shown here are for a broker that is associated with the queue manager `test`.

Grant only execute authority to the broker to the user IDs that are defined in the group group1:

```
setmqaut -m test -t queue -n SYSTEM.BROKER.AUTH -g group1 -all +set
dspmqaut -m test -t queue -n SYSTEM.BROKER.AUTH -g group1
```

Grant only execute and write authority to the broker to the user IDs that are defined in the group group2:

```
setmqaut -m test -t queue -n SYSTEM.BROKER.AUTH -g group2 -all +set +put
dspmqaut -m test -t queue -n SYSTEM.BROKER.AUTH -g group2
```

Revoke execute authority from the user IDs that are defined in the group group2:

```
setmqaut -m test -t queue -n SYSTEM.BROKER.AUTH -g group2 -set
dspmqaut -m test -t queue -n SYSTEM.BROKER.AUTH -g group2
```

Using a generic WebSphere MQ profile on a UNIX system, grant only write authority for all execution groups for the user IDs that are defined in the group group3:

```
setmqaut -m test -t queue -n "SYSTEM.BROKER.AUTH.**" -g group3 -all +put
dspmqaut -m test -t queue -n "SYSTEM.BROKER.AUTH.**" -g group3
```

Note: You enclose generic profile names in quotes on UNIX and Linux systems. For more information see the WebSphere MQ Version 7 Information Center online and search for the "Using OAM generic profiles on UNIX systems and Windows" topic.

Using a generic WebSphere MQ revoke write authority on a UNIX system for all execution groups for the user IDs that are defined in the group group3:

```
setmqaut -m test -t queue -n "SYSTEM.BROKER.AUTH.**" -g group3 -all –put
dspmqaut -m test -t queue -n "SYSTEM.BROKER.AUTH.**" -g group3
```

Grant only read authority for a specific execution group called `default` for group group4:

```
setmqaut -m test -t queue -n SYSTEM.BROKER.AUTH.default -g group4 -all +inq
dspmqaut -m test -t queue -n SYSTEM.BROKER.AUTH.default -g group4
```

Revoke execute and write authority for a specific execution group called `default` for group group5:

```
setmqaut -m test -t queue -n SYSTEM.BROKER.AUTH.default -g group5 -set -put
dspmqaut -m test -t queue -n SYSTEM.BROKER.AUTH.default -g group5
```

Using a generic WebSphere MQ on a non-UNIX system, dump all WebSphere MQ authorities for all execution groups:

```
dmpmqaut -m test -t queue -n SYSTEM.BROKER.AUTH.**
```

Related concepts:

"Security overview" on page 351
When you are designing a WebSphere Message Broker application it is important to consider the security measures that are needed to protect the information in the system.

"Broker administration security overview" on page 362
Broker administration security controls the rights of users to complete administrative tasks for a broker and its resources.

Related tasks:

"Enabling broker administration security" on page 368
Enable broker administration security on a broker to control which users can complete specific tasks against that broker and its resources.

"Disabling broker administration security" on page 380
Disable broker administration security to remove control over a broker and its

resources.

"Authorizing users for broker administration" on page 371
Grant authority to one or more groups or users to authorize them to complete specific tasks against a broker and its resources.

"Granting and revoking authority on z/OS systems"
Grant or revoke authority to one or more groups or users to complete specific tasks against a broker running on z/OS.

Related information:


WebSphere MQ Version 7 Information Center online

Granting and revoking authority on z/OS systems:

Grant or revoke authority to one or more groups or users to complete specific tasks against a broker running on z/OS.

Before you begin

Before you start:

Activate broker administration security for the broker before you grant and revoke authority for requests sent to that broker.

About this task

Configure the external security manager (ESM) that you are using with WebSphere MQ to grant the required permissions on z/OS systems. For example, if you are using RACF, set up profiles to hold the information required for WebSphere MQ security checking. The examples in this topic assume that you are using RACF.

Complete the following steps:

Procedure

1. Activate security on the queue manager that is associated with the broker.
2. Activate queue security on the same queue manager:
   - If you use uppercase profiles, you must define profiles in `MQQUEUE (Member Class)` or `GMQQUEUE (Group Class)`.
   - If you use mixed case profiles, you must define profiles in `MXQUEUE (Member Class)` or `GMXQUEUE (Group Class)`.

   For example:
   - Define the broker profile for queue manager `MQ01`:
     ```
     RDEFINE MQQUEUE MQ01.SYSTEM.BROKER.AUTH UACC(NONE)
     ```
   - Define the profile for all execution groups on queue manager `MQ01`:
     ```
     RDEFINE MXQUEUE MQ01.SYSTEM.BROKER.AUTH.** UACC(NONE)
     ```
   - Define a profile for the specific execution group called `default` for queue manager `MQ01`:
     ```
     RDEFINE MXQUEUE MQ01.SYSTEM.BROKER.AUTH.default UACC(NONE)
     ```
3. Activate the WebSphere MQ class so that security checks can be made by the broker. For example, activate the class `MQQUEUE`:
   ```
   SETROPTS CLASSACT(MQQUEUE)
   ```

4. Define WebSphere MQ permissions. The mapping between broker permissions, associated WebSphere MQ permissions, and associated RACF access levels is shown in the following table.

Broker permission	WebSphere MQ permission	RACF access level
Read	Inquire	READ
Write	Put	UPDATE
Execute	Set	ALTER

Examples

All the examples shown here are for a broker that is associated with the queue manager `MQ01`.

Add execute permission for group GROUP1 to the broker:

```
PERMIT MQ01.SYSTEM.BROKER.AUTH CLASS(MQQUEUE) ID(GROUP1) ACCESS(ALTER)
```

Remove broker execute permission to the broker for group GROUP2:

```
PERMIT MQ01.SYSTEM.BROKER.AUTH CLASS(MQQUEUE) ID(GROUP2) ACCESS(ALTER) DEL
```

Add write permission to all execution groups for group GROUP3:

```
PERMIT MQ01.SYSTEM.BROKER.AUTH.** CLASS(MXQUEUE) ID(GROUP3) ACCESS(UPDATE)
```

Remove write permission to all execution groups for group GROUP4:

```
PERMIT MQ01.SYSTEM.BROKER.AUTH.** CLASS(MXQUEUE) ID(GROUP4) DEL
```

Add read permission to a specific execution group called `default` for group GROUP5:

```
PERMIT MQ01.SYSTEM.BROKER.AUTH.default CLASS(MXQUEUE) ID(GROUP5) ACCESS(READ)
```

The following JCL file shows one way in which you can check the RACF permissions that you have set for the broker `MQTEST`:

```
//RACFDUMP JOB ,MQTEST,USER=MQTEST,TIME=1,MSGCLASS=H
//STEP1 EXEC PGM=IKJEFT01,REGION=64M,DYNAMNBR=99
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
 /* LIST ALL EXISTING PROFILES IN THE MQADMIN CLASS */
 SEARCH CLASS(MQADMIN)
 /* LIST UPPERCASE PROFILES IN THE MQQUEUE MEMBER CLASS */
 SEARCH CLASS(MQQUEUE)
 /* LIST MIXED CASE PROFILES IN THE MQQUEUE MEMBER CLASS */
 SEARCH CLASS(MXQUEUE)
 /* LIST THE QMGR PROFILE */
 RLIST MQADMIN MI09.NO.SUBSYS.SECURITY ALL
/*
```

This JCL job might produce results like the following output:

```
READY
 /* LIST ALL EXISTING PROFILES IN THE MQADMIN CLASS */
READY
 SEARCH CLASS(MQADMIN)
EP00.NO.SUBSYS.SECURITY
EP01.NO.SUBSYS.SECURITY
EP02.NO.SUBSYS.SECURITY
EP03.NO.SUBSYS.SECURITY
EP04.NO.SUBSYS.SECURITY
MA00.NO.SUBSYS.SECURITY
```

```
MA01.NO.SUBSYS.SECURITY
MA02.NO.SUBSYS.SECURITY
MA03.NO.SUBSYS.SECURITY
MA04.NO.SUBSYS.SECURITY
MA05.NO.SUBSYS.SECURITY
MA06.NO.SUBSYS.SECURITY
MA07.NO.SUBSYS.SECURITY
MA08.NO.SUBSYS.SECURITY
MA09.NO.SUBSYS.SECURITY
MA10.NO.SUBSYS.SECURITY
MA11.ALTERNATE.USER.KMCMUL
MA11.ALTERNATE.USER.KMCMUL3
MA11.ALTERNATE.USER.MA15USR
MA11.CHANNEL.MA11.TO.REG1
MA11.CONTEXT
MA11.NO.CMD.CHECKS
MA11.NO.CMD.RESC.CHECKS
MA11.NO.CMDS.CHECKS
MA11.NO.CMDS.RESC.CHECKS
MA11.NO.SUBSYS.SECURITY
MA11.RESLEVEL
MA12.NO.SUBSYS.SECURITY
MA13.NO.SUBSYS.SECURITY
MA14.NO.SUBSYS.SECURITY
MA15.NO.SUBSYS.SECURITY
MA16.NO.SUBSYS.SECURITY
MA17.NO.SUBSYS.SECURITY
MA18.NO.SUBSYS.SECURITY
MA19.NO.SUBSYS.SECURITY
MA20.NO.SUBSYS.SECURITY
MI00.NO.SUBSYS.SECURITY
MI01.NO.SUBSYS.SECURITY
MI02.NO.SUBSYS.SECURITY
MI03.NO.SUBSYS.SECURITY
MI04.NO.SUBSYS.SECURITY
MI05.NO.SUBSYS.SECURITY
MI06.NO.SUBSYS.SECURITY
MI07.NO.SUBSYS.SECURITY
MI08.NO.SUBSYS.SECURITY
MI09.ALTERNATE.USER.MI09STC
MI09.ALTERNATE.USER.NHARRIS
MI09.NO.CMD.CHECKS
MI09.NO.CONNECT.CHECKS
MI09.NO.CONTEXT.CHECKS
MI09.NO.SUBSYS.SECURITY
MI10.NO.SUBSYS.SECURITY
MI11.NO.SUBSYS.SECURITY
MI12.NO.SUBSYS.SECURITY
MI13.NO.SUBSYS.SECURITY
MI14.NO.SUBSYS.SECURITY
MI15.NO.SUBSYS.SECURITY
MI16.NO.SUBSYS.SECURITY
MI17.NO.SUBSYS.SECURITY
MI18.NO.SUBSYS.SECURITY
MI19.NO.SUBSYS.SECURITY
MI20.NO.SUBSYS.SECURITY
MI09.CHANNEL.** (G)
MI09.QUEUE.** (G)
READY
 /* LIST UPPERCASE PROFILES IN THE MQQUEUE MEMBER CLASS */
READY
 SEARCH CLASS(MQQUEUE)
MA11.INPUT2.QUEUE
MA11.KMBRK
MA11.MA11.DEAD.QUEUE
MA11.MA15
MA11.REG1
```

```
MA11.SUBSCRIBER.RESULTS.QUEUE
MA11.SUBSCRIBER3.RESULTS.QUEUE
MA11.SUBSCRIBER4.RESULTS.QUEUE
MA11.SUBSCRIBER5.RESULTS.QUEUE
MA11.SUBSCRIBER6.RESULTS.QUEUE
MA11.SUBSCRIBER9.RESULTS.QUEUE
MA11.SYSTEM.CHANNEL.EVENT
MA11.SYSTEM.CHANNEL.SYNCQ
MA11.SYSTEM.CLUSTER.COMMAND.QUEUE
MA11.SYSTEM.COMMAND.INPUT
MA11.SYSTEM.COMMAND.REPLY.MODEL
MI09.SYSTEM.BROKER.AUTH.SECURITY_EXE
MA11.SYSTEM.BROKER.** (G)
MA11.SYSTEM.** (G)
MA11.** (G)
MI09.** (G)
READY
 /* LIST MIXED CASE PROFILES IN THE MQQUEUE MEMBER CLASS */
READY
 SEARCH CLASS(MXQUEUE)
NO ENTRIES MEET SEARCH CRITERIA
READY
 /* LIST THE QMGR PROFILE */
READY
 RLIST MQADMIN MI09.NO.SUBSYS.SECURITY ALL
CLASS      NAME
-----      ----
MQADMIN    MI09.NO.SUBSYS.SECURITY

GROUP CLASS NAME
----- ----- ----
GMQADMIN

RESOURCE GROUPS
-------- ------
NONE

LEVEL  OWNER      UNIVERSAL ACCESS  YOUR ACCESS  WARNING
-----  --------   ----------------  -----------  -------
 00    MQTEST          NONE               NONE    NO

INSTALLATION DATA
-----------------
NONE

APPLICATION DATA
----------------
NONE

SECLEVEL
--------
NO SECLEVEL

CATEGORIES
----------
NO CATEGORIES

SECLABEL
--------
NO SECLABEL

AUDITING
--------
FAILURES(READ)

NOTIFY
------
```

```
NO USER TO BE NOTIFIED

CREATION DATE  LAST REFERENCE DATE  LAST CHANGE DATE
 (DAY) (YEAR)       (DAY) (YEAR)      (DAY) (YEAR)
-------------  ------------------  ----------------
  237    09          237    09         237    09

ALTER COUNT   CONTROL COUNT   UPDATE COUNT   READ COUNT
-----------   -------------   ------------   ----------
  000000         000000         000000         000000

USER      ACCESS   ACCESS COUNT
----      ------   ------ -----
NO USERS IN ACCESS LIST

   ID     ACCESS  ACCESS COUNT  CLASS                    ENTITY  NAME
-------- -------  ------------ -------- ---------------------------------------
NO ENTRIES IN CONDITIONAL ACCESS LIST
READY
END
```

Related concepts:

"Security overview" on page 351
When you are designing a WebSphere Message Broker application it is important to consider the security measures that are needed to protect the information in the system.

"Broker administration security overview" on page 362
Broker administration security controls the rights of users to complete administrative tasks for a broker and its resources.

Related tasks:

"Enabling broker administration security" on page 368
Enable broker administration security on a broker to control which users can complete specific tasks against that broker and its resources.

"Disabling broker administration security"
Disable broker administration security to remove control over a broker and its resources.

"Authorizing users for broker administration" on page 371
Grant authority to one or more groups or users to authorize them to complete specific tasks against a broker and its resources.

"Granting and revoking authority on Linux, UNIX, and Windows systems" on page 374
Grant or revoke authority to one or more groups or users to complete specific tasks against a broker running on Linux, UNIX, or Windows.

Disabling broker administration security

Disable broker administration security to remove control over a broker and its resources.

About this task

You can disable administrative security for a broker by updating the relevant broker property.

Procedure

1. Stop the broker in the WebSphere Message Broker Explorer, or run the **mqsistop** command.

2. Clear the security option for this broker in the WebSphere Message Broker Explorer, or run the **mqsichangebroker** command, specifying the parameter **-s inactive**. For example, to disable security for the broker MB7BROKER, enter the following command:

   ```
   mqsichangebroker MB7BROKER -s inactive
   ```

3. Start the broker in the WebSphere Message Broker Explorer, or run the **mqsistart** command.

Results

When this command completes, all users are able to complete tasks against the broker; for example, they can run the **mqsichangeproperties** command to change broker properties. They can also complete all tasks against all execution groups that have been defined, and are defined in the future, on this broker.

When you disable broker administration security, the authorization queues that are associated with this broker are retained. If you enable security again, these queues are reused, therefore you must ensure that the authorizations defined by these queues are correct.

Related concepts:

"Security overview" on page 351
When you are designing a WebSphere Message Broker application it is important to consider the security measures that are needed to protect the information in the system.

"Broker administration security overview" on page 362
Broker administration security controls the rights of users to complete administrative tasks for a broker and its resources.

Related tasks:

"Enabling broker administration security" on page 368
Enable broker administration security on a broker to control which users can complete specific tasks against that broker and its resources.

"Authorizing users for broker administration" on page 371
Grant authority to one or more groups or users to authorize them to complete specific tasks against a broker and its resources.

Related reference:

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

Activating broker administration security for WebSphere MQ Version 7.1, or later

Describes how to enable channel authentication security and the effect of this on a broker queue manager.

About this task

When you create a broker, if the WebSphere MQ queue manager does not exist, the queue manager is automatically created. If WebSphere MQ Version 7.1, or later, has been selected for the queue manager, the channel authentication security will be automatically disabled.

If more precise control over the access granted to connecting systems is required at a channel level, channel authentication security can be enabled. For more details, see WebSphere MQ Version 7.1 Channel authentication records

To enable channel authentication security in order to start using channel authentication records you must run this MQSC command:

```
ALTER QMGR CHLAUTH(ENABLED
```

Once enabled, there will be consequences for any channel based communication with a broker. However, this should not affect any privileged or non-privileged user from accessing local brokers. See following table for a definition of privileged users who have full administrative authorities:

Table 7. . Privileged users by platform.

Platform	Privileged users
Windows systems	• SYSTEM • Members of the mqm group • Members of the Administrators group
UNIX and Linux systems	• Members of the mqm group

Privileged or non-privileged users wanting to remotely administer a broker by means of `CMP/API/MBX/Toolkit` must run the following commands in order to grant their user access:

Procedure

1. Enable remote administration on Queue Manager
2. `setmqaut -m` *`QMNAME`* `-n SYSTEM.MQEXPLORER.REPLY.MODEL -t queue -p` *`username`* `+dsp +inq +put +get`
3. `SET CHLAUTH('SYSTEM.BKR.CONFIG')TYPE(ADDRESSMAP)ADDRESS('address-of-machine-who-is-allowed')MCAUSER('NonPrivilegedUser')ACTION(ADD)`

   Where 'NonPrivilegedUser' is a user defined on the remote machine.

What to do next

The rule would need to be set up for any ip-address wanting to administer the broker remotely.

Related concepts:

"Security overview" on page 351
When you are designing a WebSphere Message Broker application it is important to consider the security measures that are needed to protect the information in the system.

Related reference:

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

Message flow security

Control access to individual messages in a message flow, using the identity of the messages.

The following topics introduce the concepts that you need to understand before you can configure message flow security, and they explain the steps involved in setting up security for your message flows:

- "Message flow security overview"
- "Setting up message flow security" on page 431

Related concepts:

Chapter 5, "Security," on page 351
Security is an important consideration for both developers of WebSphere Message Broker applications, and for system administrators configuring WebSphere Message Broker authorities.

"Security overview" on page 351
When you are designing a WebSphere Message Broker application it is important to consider the security measures that are needed to protect the information in the system.

"WS-Security" on page 765
Web Services Security (WS-Security) describes enhancements to SOAP messaging to provide quality of protection through message integrity, message confidentiality, and single message authentication. WS-Security mechanisms can be used to accommodate a wide variety of security models and encryption technologies.

Related tasks:

"Setting up broker administration security" on page 368
Control the actions that users can request against a broker and its resources.

"Setting up z/OS security" on page 556
On z/OS, you must complete several security configuration tasks before WebSphere Message Broker can work correctly.

Message flow security overview

WebSphere Message Broker provides a security manager, which enables you to control access to individual messages in a message flow, using the identity of the message.

You can configure the broker to perform end-to-end processing of an identity carried in a message through a message flow. This capability enables you to configure security for a message flow, allowing you to control access based on the identity associated with the message and providing a security mechanism that is independent of both transport type and message format.

If you do not enable message flow security, the default security facilities in WebSphere Message Broker are based on the facilities provided by the transport mechanism. In this case, the broker processes all messages that are delivered to it, using the broker service identity as a proxy identity for all message instances. Any identity that is present in the incoming message is ignored.

Instead of delegating this authority to the transport mechanism, the security manager enables the broker to:

- Extract the identity from an inbound message
- Authenticate the identity (using an external security provider)
- Map the identity to an alternative identity (using an external security provider)
- Check that either the alternative identity or the original identity is authorized to access the message flow (using an external security provider)
- Propagate either the alternative identity or the original identity with an outbound message.

The security functions that are associated with a message flow are controlled by using "Security profiles" on page 387, which are created by the broker administrator and accessed by the security manager at run time. The following external security providers (also known as Policy Decision Points or PDPs) are supported:

- WS-Trust V1.3 compliant security token servers (including TFIM V6.2) for authentication, mapping, and authorization
- Tivoli Federated Identity Manager (TFIM) V6.1 for authentication, mapping, and authorization
- Lightweight Directory Access Protocol (LDAP) for authentication and authorization

You can invoke message flow security by configuring either a security enabled input node or a SecurityPEP node. The SecurityPEP node enables you to invoke the message flow security manager at any point in the message flow between an input node and an output (or request) node.

For an overview of the sequence of events that occur when the message flow security manager is invoked using either a security enabled input node or a SecurityPEP node, see the following topics:

- "Invoking message flow security using a security enabled input node" on page 406
- "Invoking message flow security using a SecurityPEP node" on page 411

The input nodes that support message flow security are:

- MQInput
- HTTPInput
- SCAInput
- SCAAsyncResponse
- SOAPInput

However, the support for treating security exceptions as normal exceptions is provided by only the MQInput, HTTPInput, SCAInput, and SCAAsyncResponse nodes; it is not available in the SOAPInput node.

The output nodes that support identity propagation are:

- MQOutput
- HTTPRequest
- SCARequest
- SCAAsyncRequest
- SOAPRequest
- SOAPAsyncRequest

If the message flow is a Web Service that is implemented by using the "SOAP nodes" on page 1609, the identity can be taken from the "WS-Security" on page 765 header tokens that are defined through appropriate "Policy sets" on page 774 and bindings.

To improve performance, the authentication, authorization, and mapping information from the configured providers is cached for reuse. You can use the

mqsireloadsecurity command to reload the security cache, and you can use the **mqsichangeproperties** command to set the expiry and sweep intervals for the security cache.

For a SOAPRequest and SOAPAsyncRequest node, an appropriate policy set and bindings can be defined to specify how the token is placed in the WS-Security header (rather than the underlying transport headers). For more information, see "Policy sets" on page 774.

The following topics in this section provide more detailed information about message flow security:

Related concepts:

"Invoking message flow security using a security enabled input node" on page 406
You can invoke the message flow security manager by configuring a security enabled input node.

"Invoking message flow security using a SecurityPEP node" on page 411
You can invoke the message flow security manager at any point in a message flow, between an input node and an output or request node, by using a SecurityPEP node.

Related tasks:

"Setting up message flow security" on page 431
Set up security on a message flow to control access based on the identity of a message passing through the message flow.

"Configuring the extraction of an identity or security token" on page 447
You can configure the SecurityPEP node or security enabled input nodes to extract the identity or security token from a message and store it in the properties tree identity fields, enabling it to be processed throughout the message flow and propagated at output or request nodes.

"Creating a security profile" on page 433
You can create a security profile for use with Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2. Support is also provided for TFIM V6.1, for compatibility with previous versions of WebSphere Message Broker. You can create the security profile by using either the **mqsicreateconfigurableservice** command or an editor in the WebSphere Message Broker Explorer.

"Configuring identity authentication and security token validation" on page 450
You can configure a message flow to perform identity authentication or security token validation using Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token sever (STS) such as Tivoli Federated Identity Manager (TFIM) Version 6.2. Support for TFIM V6.1 is also provided, for compatibility with previous versions of WebSphere Message Broker.

"Configuring identity mapping" on page 463
Configure a security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2, to map the incoming security token and, if required, to authenticate and authorize it.

"Configuring authorization" on page 470
Configure the broker to use an external security provider to authorize an identity in a message flow. You can use either Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS) such as Tivoli Federated Identity Manager (TFIM) V6.2. Support for TFIM V6.1 is also provided for compatibility with previous versions of WebSphere Message Broker.

"Configuring for identity propagation" on page 492
To enable a message flow to perform identity propagation, the input nodes must extract the identity and the output node must propagate it.

Related reference:

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"**mqsireloadsecurity** command" on page 3911
Use the **mqsireloadsecurity** command to force the immediate expiry of some or all the entries in the security cache.

"Parameter values for the securitycache component" on page 3815
Select the objects and properties associated with the securitycache component that you want to change.

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

"HTTPInput node" on page 4474
Use the HTTPInput node to receive an HTTP message from an HTTP client for processing by a message flow.

"SOAPInput node" on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

"SOAPAsyncRequest node" on page 4750
Use the SOAPAsyncRequest node with the SOAPAsyncResponse node to construct a pair of message flows that call a Web service asynchronously.

"SCAInput node" on page 4707
Use the SCAInput node with the SCAReply node to process messages from WebSphere Process Server.

"SecurityPEP node" on page 4729
Use the SecurityPEP node in a message flow to invoke the message flow security manager at any point in the message flow.

"HTTPRequest node" on page 4488
Use the HTTPRequest node to interact with a web service.

"MQOutput node" on page 4612
Use the MQOutput node to send messages to clients that connect to the broker using the WebSphere MQ Enterprise Transport and that use the MQI and AMI application programming interfaces.

Security profiles

A security profile defines the security operations that are to be performed in a message flow at SecurityPEP nodes and security enabled input and output nodes.

Security profiles are configured by the broker administrator before deploying a message flow, and are accessed by the security manager at run time.

A security profile allows a broker administrator to specify whether identity and security token propagation, authentication, authorization, and mapping are performed on the identity or security tokens associated with messages in the message flow, and if so, which external security provider (also known as a Policy Decision Point or PDP) is used. IBM Tivoli Federated Identity Manager (TFIM) V6.1, and WS-Trust v1.3 compliant security token servers (including TFIM V6.2), are supported for authentication, authorization, and mapping. Lightweight Directory Access Protocol (LDAP) is supported for authentication and authorization.

Security profiles apply to the SecurityPEP node and to security enabled input, output, and request nodes, and are configured by the administrator at deployment time in the Broker Archive editor. These nodes have a **Security Profile** property (in the Broker Archive editor), which can be left blank, set to *No Security*, or set to a specific security profile name. Set *No Security* to explicitly turn off security for the node. If you leave the **Security Profile** property blank, the node inherits the **Security Profile** property that is set at the message flow level. If you leave the **Security Profile** property blank at both levels, security is turned off for the node. When this property is set to the name of a specific security profile, that profile determines what message flow security is configured. If the named security profile does not exist in the run time, the message flow fails to deploy. If the specified external security provider does not support the type of token configured on the node for the security operation, an error is reported and the message flow fails to deploy.

The security profile also specifies whether propagation is required. A pre-configured profile that specifies propagation is provided for use by output and request nodes. This profile is the *Default Propagation* security profile. This profile can also be used on an input node to extract tokens and put them into the message tree ready for propagation or processing in a SecurityPEP node.

Security profiles contain values for the following properties:

authentication
Defines the type of authentication that is performed on the source identity.

This property applies only to SecurityPEP nodes and input nodes. For more information, see “Authentication and validation” on page 398.

authenticationConfig
Defines the information that the broker needs to connect to the provider, and the information needed to look up the identity tokens. It is a provider-specific configuration string. This property applies only to SecurityPEP nodes and input nodes.

mapping
Defines the type of mapping that is performed on the source identity. This property applies only to SecurityPEP nodes and input nodes. For more information, see “Identity mapping” on page 403.

mappingConfig
Defines how the broker connects to the provider, and contains additional information required to look up the mapping routine. It is a provider-specific configuration string. This property applies only to SecurityPEP nodes and input nodes.

authorization
Defines the types of authorization checks that are performed on the mapped or source identity. This property applies only to SecurityPEP nodes and input nodes. For more information, see “Authorization” on page 401.

authorizationConfig
Defines how the broker connects to the provider, and contains additional information that can be used to check access (for example, a group that can be checked for membership). It is a provider-specific configuration string. This property applies only to SecurityPEP nodes and input nodes.

passwordValue
Defines how passwords are treated when they enter a message flow. If *PLAIN* is selected, the password appears in the Properties folder in plain text. If *OBFUSCATE* is selected, the password appears in the Properties folder in base64 encoding. If *MASK* is selected, the password appears in the Properties folder as four asterisks (****). This property applies only to SecurityPEP nodes and input nodes.

Propagation
Enables or disables identity propagation on output and request nodes. On the security enabled input nodes, you can choose to select only identity propagation, without specifying any other security operations, to make the extracted incoming identity or security token available for use in the other nodes in the message flow, such as output or request nodes. For more information, see “Identity and security token propagation” on page 426.

For information on configuring a security profile for LDAP, TFIM, or a WS-Trust v1.3 compliant security token server (STS), see “Creating a security profile” on page 433.

Related concepts:

“Identity” on page 390
In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

“Authentication and validation” on page 398
Authentication is the process of establishing the identity of a user or system and verifying that the identity is valid. Applying authentication to a SAML security token involves validating the assertions that it carries and confirming that it is

being processed within its validity period.

"Identity mapping" on page 403
Identity mapping is the transformation of a security token from one format to another format, or the federation of an identity from one realm to an equivalent identity in another realm.

"Authorization" on page 401
Authorization is the process of verifying that an identity token has permission to access a message flow.

"Identity and security token propagation" on page 426
Identity and security token propagation enables the identity and security tokens (associated with each message) to be propagated throughout a message flow, and on to target applications through output or request nodes.

Related tasks:

"Configuring the extraction of an identity or security token" on page 447
You can configure the SecurityPEP node or security enabled input nodes to extract the identity or security token from a message and store it in the properties tree identity fields, enabling it to be processed throughout the message flow and propagated at output or request nodes.

"Configuring identity authentication and security token validation" on page 450
You can configure a message flow to perform identity authentication or security token validation using Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token sever (STS) such as Tivoli Federated Identity Manager (TFIM) Version 6.2. Support for TFIM V6.1 is also provided, for compatibility with previous versions of WebSphere Message Broker.

"Configuring identity mapping" on page 463
Configure a security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2, to map the incoming security token and, if required, to authenticate and authorize it.

"Creating a security profile" on page 433
You can create a security profile for use with Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2. Support is also provided for TFIM V6.1, for compatibility with previous versions of WebSphere Message Broker. You can create the security profile by using either the **mqsicreateconfigurableservice** command or an editor in the WebSphere Message Broker Explorer.

"Configuring authorization" on page 470
Configure the broker to use an external security provider to authorize an identity in a message flow. You can use either Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS) such as Tivoli Federated Identity Manager (TFIM) V6.2. Support for TFIM V6.1 is also provided for compatibility with previous versions of WebSphere Message Broker.

"Configuring for identity propagation" on page 492
To enable a message flow to perform identity propagation, the input nodes must extract the identity and the output node must propagate it.

"Setting up message flow security" on page 431
Set up security on a message flow to control access based on the identity of a message passing through the message flow.

Related reference:

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

"HTTPInput node" on page 4474
Use the HTTPInput node to receive an HTTP message from an HTTP client for processing by a message flow.

"SOAPInput node" on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

"SOAPAsyncRequest node" on page 4750
Use the SOAPAsyncRequest node with the SOAPAsyncResponse node to construct a pair of message flows that call a Web service asynchronously.

"HTTPRequest node" on page 4488
Use the HTTPRequest node to interact with a web service.

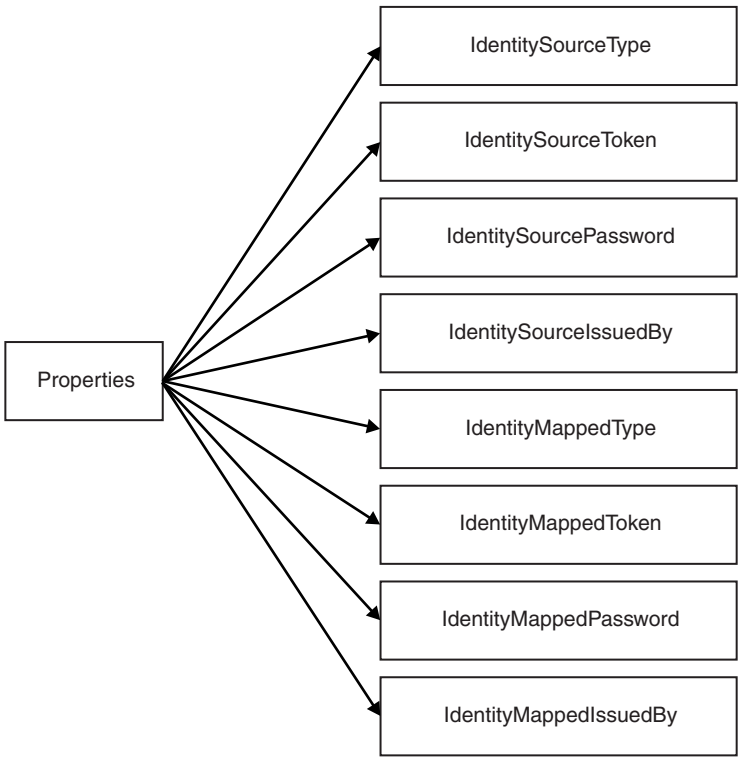
"MQOutput node" on page 4612
Use the MQOutput node to send messages to clients that connect to the broker using the WebSphere MQ Enterprise Transport and that use the MQI and AMI application programming interfaces.

"SecurityPEP node" on page 4729
Use the SecurityPEP node in a message flow to invoke the message flow security manager at any point in the message flow.

Identity

In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

When a SecurityPEP node or a supported input node is configured with a security profile, the extracted identity is held in the broker as eight properties in the Properties folder of the message tree structure. These properties define two identities in the broker: source and mapped. For both the source and mapped identities, values are held for **Type**, **Token**, **Password**, and **IssuedBy** properties:



The **Identity token type** property on the security-enabled input nodes can be set to a value of *Transport Default*, which causes the token type to be created from the default identity header or fields of the transport. For WebSphere MQ, *Transport Default* provides an identity type of *Username*. For HTTP, *Transport Default* provides an identity type of *Username and Password*. The Identity token type property on the SecurityPEP node can be set to *Current Token*, which enables it to use the identity in the Properties folder fields instead of extracting a new identity from the message.

The following table shows the support that is provided (by the message flow security manager and external security providers) for the extraction of different types of security token. For information about the token types that are supported for identity propagation, see “Identity and security token propagation” on page 426.

Table 8. Support for security token types - token extraction

Token type (format)	Broker security manager support for token extraction	External security provider support
Username	Username tokens are supported for extraction by the following nodes: • HTTPInput • MQInput • SCAInput • SCAAsyncResponse • SecurityPEP • SOAPInput The token is obtained from one of the following transport headers: • MQ – From MQMD user ID • HTTP – From HTTP BasicAuth header containing only a username part • SOAP – From a WS-Security UsernameToken header. The policy set and binding (associated with the SOAP node) must define a username profile, and the wsse:UsernameToken must contain only a `wsse:Username` element. – From the Kerberos subject in a WS-Security header. The policy set and binding (associated with the SOAP node) must define a Kerberos profile. – From the HTTP BasicAuth header containing only a username part if no policy set is defined on the SOAP node. Alternatively, the token can be taken from any part of the message tree when the token location is specified (on the node) using an XPath expression or ESQL field path. The literal string value used by the broker (and which you can use to specify the token type in an ESQL or Java program) is *username*.	LDAP: Authorization

Table 8. Support for security token types - token extraction (continued)

Token type (format)	Broker security manager support for token extraction	External security provider support
Username and password or Username and RACF PassTicket	Username and password tokens are supported for extraction by the following nodes: • HTTPInput • MQInput • SCAInput • SecurityPEP • SCAAsyncResponse • SOAPInput The token is obtained from one of the following transport headers: • HTTP – From HTTP BasicAuth header containing both a username and password part • SOAP – From a WS-Security UsernameToken header. The policy set and binding (associated with the SOAP node) must define a username profile and the wsse:UsernameToken must contain both `wsse:Username` and `wsse:Password` elements. – From the HTTP BasicAuth header containing only a username part if no policy set is defined on the SOAP node Alternatively, the token can be obtained from any part of the message tree when the token location is specified (on the node) using an XPath expression or ESQL path. The password token can carry either a clear text password or a RACF PassTicket. If you are using a WS-Trust V1.3 STS (such as TFIM V6.2), you can use it to map (issue) or validate RACF PassTickets, by specifying the token type as *Username and password*. This support is available with security enabled input nodes and SecurityPEP nodes. The literal string value used by the broker (and which you can use to specify the token type in an ESQL or	LDAP: • Authentication • Authorization TFIM V6.1: • Authentication • Mapping • Authorization WS-Trust V1.3 STS (including TFIM V6.2): • Authentication • Mapping • Authorization

Table 8. Support for security token types - token extraction (continued)

Token type (format)	Broker security manager support for token extraction	External security provider support
SAML assertion	SAML tokens are supported for extraction by the following nodes: • SecurityPEP • MQInput • HTTPInput • SCAInput • SCAAsyncResponse • SOAPInput The token is obtained from one of the following places: • SOAP – From a WS-Security header. The policy set and binding (associated with the SOAP node) must define a SAML profile. • Any part of the message tree when the token location is specified using an XPath expression or ESQL path. The literal string value used by the broker (and which you can use to specify the token type in an ESQL or Java program) is *SAML*.	WS-Trust V1.3 STS (including TFIM V6.2): • Authentication • Mapping • Authorization

Table 8. Support for security token types - token extraction (continued)

Token type (format)	Broker security manager support for token extraction	External security provider support
Kerberos GSS v5 AP_REQ	Kerberos tickets are supported for processing by the message flow security manager from the SecurityPEP node. A WS-Security Kerberos token profile is supported by the following SOAP nodes, but in this case, the Kerberos Key Distribution Center is communicated with directly, and the properties folder is populated with a Username token representing the Kerberos subject: • SOAPInput The token is obtained from any part of the message tree at a SecurityPEP node, if the token location is specified using an XPath expression or ESQL path. The literal string value used by the broker (and which you can use to specify the token type in an ESQL or Java program) is *kerberosTicket*.	WS-Trust V1.3 STS (including TFIM V6.2): • Authentication • Mapping • Authorization
LTPA v2 token	LTPA tokens are supported for extraction by the following nodes: • SecurityPEP • SOAPInput The token is obtained from any part of the message tree at a SecurityPEP node, if the token location is specified using an XPath expression or ESQL path. The literal string value used by the broker (and which you can use to specify the token type in an ESQL or Java program) is *LTPA*.	WS-Trust V1.3 STS (including TFIM V6.2): • Authentication • Mapping • Authorization

Table 8. Support for security token types - token extraction (continued)

Token type (format)	Broker security manager support for token extraction	External security provider support
X.509 Certificate	X.509 tokens are supported for extraction by the following nodes: • SecurityPEP • MQInput • HTTPInput • SCAInput • SCAAsyncResponse • SOAPInput The token is obtained from one of the following places: • SOAP – From a WS-Security header. The policy set and binding (associated with the SOAP node) must define an X.509 profile. • Any part of the message tree when the token location is specified using an XPath expression or ESQL path. The literal string value used by the broker (and which you can use to specify the token type in an ESQL or Java program) is *X.509*.	TFIM V6.1: • Authentication • Mapping • Authorization. WS-Trust V1.3 STS (including TFIM V6.2): • Authentication • Mapping • Authorization.
Universal WSSE token	Universal WSSE tokens are supported for extraction by the SecurityPEP node only. The token is obtained from any part of the message tree at a SecurityPEP node, if the token location is specified using an XPath expression or ESQL path. The literal string value used by the broker (and which you can use to specify the token type in an ESQL or Java program) is *UniversalWsse*.	WS-Trust V1.3 STS (including TFIM V6.2): • Authentication • Mapping • Authorization.

The source identity is set by the SecurityPEP or input node only if a security profile is associated with the node. The information to complete these fields is typically found in the headers of a message but can also be located in the body (provided that the node has been configured with an ESQL Path or XPath reference for the various properties). If multiple identities are available (for example, if you are using message aggregation), the first identity is used. The token extraction is transport specific and can be performed only using transports that support the

flow of identities. These transports are: Websphere MQ, HTTP(S), and SOAP. See "MQInput node" on page 4594 and "HTTPInput node" on page 4474 for more information.

You can modify the values in the properties (for example, from ESQL), but do not write to the *IdentitySource** values. For example, you can create a custom identity mapping routine in ESQL or Java by using the *IdentitySource** values to create custom *IdentityMapped** values.

SAML and Universal WSSE tokens are stored in the Properties tree IdentitySourceToken or IdentityMappedToken field as a character bit stream. To access this data as a message tree, parse it into a suitable parser, such as XMLNSC:

```
-- Parse the mapped SAML2.0 token in the properties folder and set it in the message body
   CREATE LASTCHILD OF OutputRoot DOMAIN('XMLNSC') PARSE(InputRoot.Properties.IdentityMappedToken,
   InputProperties.Encoding, InputProperties.CodedCharSetId);
```

To set either SAML or Universal WSSE tokens into the properties fields, you must obtain the bit stream of a tree; for example, by using the ESQL `ASBITSTREAM` function.

Related concepts:

"Authentication and validation" on page 398
Authentication is the process of establishing the identity of a user or system and verifying that the identity is valid. Applying authentication to a SAML security token involves validating the assertions that it carries and confirming that it is being processed within its validity period.

"Identity mapping" on page 403
Identity mapping is the transformation of a security token from one format to another format, or the federation of an identity from one realm to an equivalent identity in another realm.

"Authorization" on page 401
Authorization is the process of verifying that an identity token has permission to access a message flow.

"Identity and security token propagation" on page 426
Identity and security token propagation enables the identity and security tokens (associated with each message) to be propagated throughout a message flow, and on to target applications through output or request nodes.

"Security profiles" on page 387
A security profile defines the security operations that are to be performed in a message flow at SecurityPEP nodes and security enabled input and output nodes.

Related tasks:

"Configuring the extraction of an identity or security token" on page 447
You can configure the SecurityPEP node or security enabled input nodes to extract the identity or security token from a message and store it in the properties tree identity fields, enabling it to be processed throughout the message flow and propagated at output or request nodes.

"Configuring identity authentication and security token validation" on page 450
You can configure a message flow to perform identity authentication or security token validation using Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token sever (STS) such as Tivoli Federated Identity Manager (TFIM) Version 6.2. Support for TFIM V6.1 is also provided, for compatibility with previous versions of WebSphere Message Broker.

"Configuring identity mapping" on page 463
Configure a security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2, to map the incoming security token and, if required, to authenticate

and authorize it.

"Configuring authentication with HTTP basic authentication" on page 451
Use a security profile to configure HTTP basic authentication in the HTTPRequest or SOAPInput nodes.

"Creating a security profile" on page 433
You can create a security profile for use with Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2. Support is also provided for TFIM V6.1, for compatibility with previous versions of WebSphere Message Broker. You can create the security profile by using either the **mqsicreateconfigurableservice** command or an editor in the WebSphere Message Broker Explorer.

"Configuring authorization" on page 470
Configure the broker to use an external security provider to authorize an identity in a message flow. You can use either Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS) such as Tivoli Federated Identity Manager (TFIM) V6.2. Support for TFIM V6.1 is also provided for compatibility with previous versions of WebSphere Message Broker.

"Configuring for identity propagation" on page 492
To enable a message flow to perform identity propagation, the input nodes must extract the identity and the output node must propagate it.

Related reference:

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

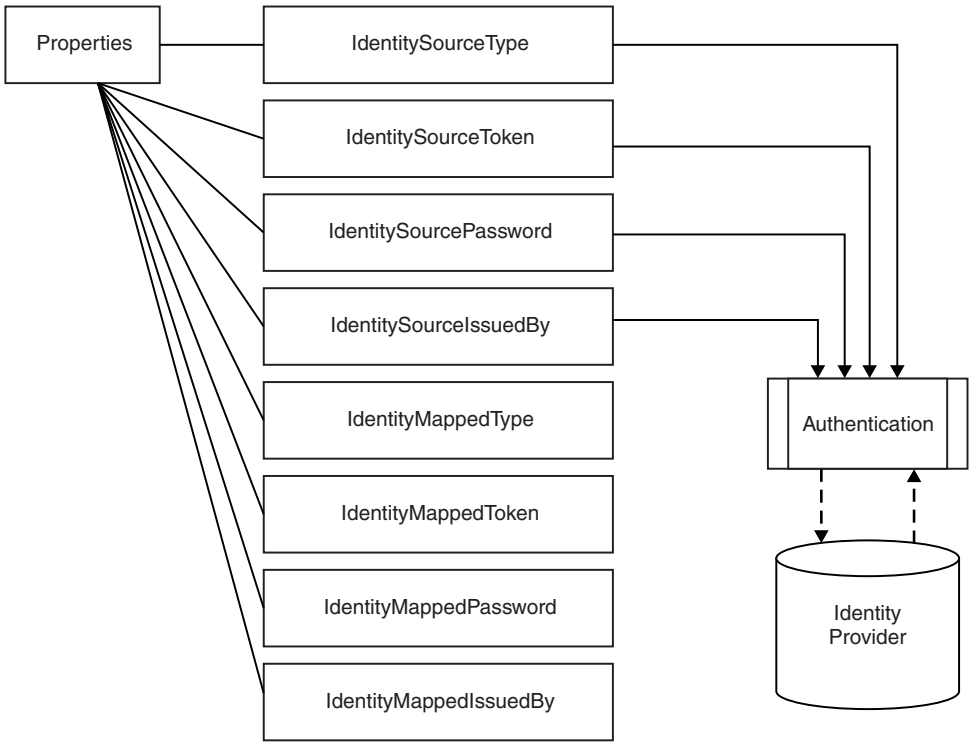
"HTTPInput node" on page 4474
Use the HTTPInput node to receive an HTTP message from an HTTP client for processing by a message flow.

"SecurityPEP node" on page 4729
Use the SecurityPEP node in a message flow to invoke the message flow security manager at any point in the message flow.

Authentication and validation

Authentication is the process of establishing the identity of a user or system and verifying that the identity is valid. Applying authentication to a SAML security token involves validating the assertions that it carries and confirming that it is being processed within its validity period.

In WebSphere Message Broker message flow security, authentication involves the security manager passing the identity type and token to an external security provider. For more information about security tokens, see "Identity" on page 390.



The following external security providers (also known as Policy Decision Points) are supported for authentication:

- Lightweight Directory Access Protocol (LDAP)
- Tivoli Federated Identity Manager (TFIM) V6.1
- WS-Trust V1.3 security token servers (STS), including TFIM V6.2

The external security provider checks the identities and returns a value to confirm whether the identity is authentic. If the identity is not authentic, a security exception is raised.

Some identity providers support only a single type of authentication token. If a token of another type is passed into the message flow, an exception is raised. For example, LDAP supports only a *Username and password* token.

You can use an LDAP provider for the authentication of an incoming identity token. The LDAP server must be LDAP Version 3 compliant.

Alternatively, you can use a WS-Trust v1.3 STS provider (for example, TFIM Version 6.2) for the authentication of an incoming identity or security token. The security manager invokes the WS-Trust v1.3 provider once, even if it is set for additional security operations (such as mapping or authorization). As a result, when you are using TFIM, you must configure a single module chain to perform all the required authentication, mapping, and authorization operations.

For more information about using TFIM V6.2 for authentication, see "Authentication, mapping, and authorization with TFIM V6.2 and TAM" on page 419.

TFIM V6.1 is also supported, for compatibility with previous versions of WebSphere Message Broker. For more information about using TFIM V6.1 for authentication, see "Authentication, mapping, and authorization with TFIM V6.1 and TAM" on page 416.

Related concepts:

"Identity" on page 390
In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

"Security profiles" on page 387
A security profile defines the security operations that are to be performed in a message flow at SecurityPEP nodes and security enabled input and output nodes.

Related tasks:

"Configuring identity authentication and security token validation" on page 450
You can configure a message flow to perform identity authentication or security token validation using Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token sever (STS) such as Tivoli Federated Identity Manager (TFIM) Version 6.2. Support for TFIM V6.1 is also provided, for compatibility with previous versions of WebSphere Message Broker.

"Configuring authentication with HTTP basic authentication" on page 451
Use a security profile to configure HTTP basic authentication in the HTTPRequest or SOAPInput nodes.

"Creating a security profile" on page 433
You can create a security profile for use with Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2. Support is also provided for TFIM V6.1, for compatibility with previous versions of WebSphere Message Broker. You can create the security profile by using either the **mqsicreateconfigurableservice** command or an editor in the WebSphere Message Broker Explorer.

"Setting up message flow security" on page 431
Set up security on a message flow to control access based on the identity of a message passing through the message flow.

Related reference:

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

"HTTPInput node" on page 4474
Use the HTTPInput node to receive an HTTP message from an HTTP client for processing by a message flow.

"SOAPInput node" on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

"SOAPAsyncRequest node" on page 4750
Use the SOAPAsyncRequest node with the SOAPAsyncResponse node to construct a pair of message flows that call a Web service asynchronously.

"HTTPRequest node" on page 4488
Use the HTTPRequest node to interact with a web service.

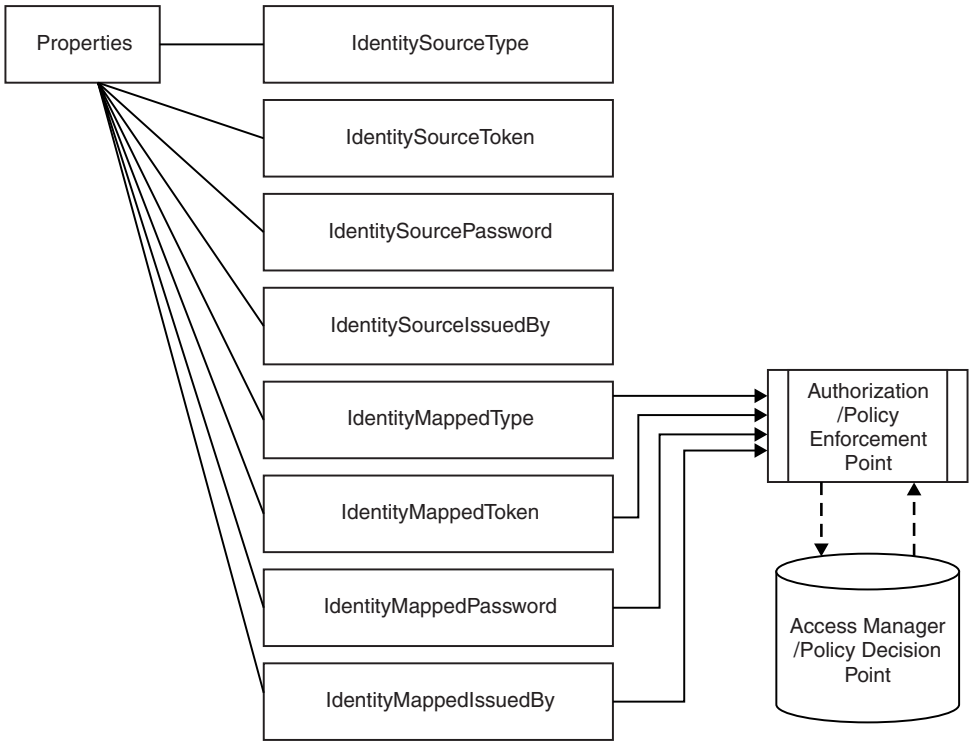
"MQOutput node" on page 4612
Use the MQOutput node to send messages to clients that connect to the broker using the WebSphere MQ Enterprise Transport and that use the MQI and AMI application programming interfaces.

"SecurityPEP node" on page 4729
Use the SecurityPEP node in a message flow to invoke the message flow security manager at any point in the message flow.

Authorization

Authorization is the process of verifying that an identity token has permission to access a message flow.

If authentication and mapping are configured, they are used to verify the identity before it is authorized.



If a mapped identity exists, authorization is applied to the mapped identity. If a mapped identity does not exist, the source identity is used.

If you specify LDAP as the provider for authorization, the security manager queries the configured LDAP server (which must be LDAP Version 3 compliant), to validate that the identity is a member of the LDAP group that is configured in the security profile.

If you specify *WS-Trust v1.3 STS* as the provider for authorization, the security manager invokes the security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2, to validate that the identity token provided has permission to access the message flow. If you are using TFIM V6.1 rather than TFIM V6.2, you can specify *TFIM* as the provider for authorization.

For more information about using TFIM V6.2 for authorization, see "Authentication, mapping, and authorization with TFIM V6.2 and TAM" on page 419.

For information about using TFIM V6.1 for authorization, see “Authentication, mapping, and authorization with TFIM V6.1 and TAM” on page 416.

Related concepts:

“Identity” on page 390
In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

“Authentication and validation” on page 398
Authentication is the process of establishing the identity of a user or system and verifying that the identity is valid. Applying authentication to a SAML security token involves validating the assertions that it carries and confirming that it is being processed within its validity period.

“Identity mapping” on page 403
Identity mapping is the transformation of a security token from one format to another format, or the federation of an identity from one realm to an equivalent identity in another realm.

“Identity and security token propagation” on page 426
Identity and security token propagation enables the identity and security tokens (associated with each message) to be propagated throughout a message flow, and on to target applications through output or request nodes.

“Security profiles” on page 387
A security profile defines the security operations that are to be performed in a message flow at SecurityPEP nodes and security enabled input and output nodes.

“Security exception processing” on page 429
A security exception is raised when a message flow security failure occurs during security processing in an input node or SecurityPEP node.

“Authentication, mapping, and authorization with TFIM V6.2 and TAM” on page 419
You can use WebSphere Message Broker, Tivoli Federated Identity Manager (TFIM) V6.2, and Tivoli Access Manager (TAM) to control authentication, mapping, and authorization.

“Authentication, mapping, and authorization with TFIM V6.1 and TAM” on page 416
Use WebSphere Message Broker, Tivoli Federated Identity Manager (TFIM) V6.1, and Tivoli Access Manager (TAM) to control authentication, mapping, and authorization.

Related tasks:

“Configuring identity mapping” on page 463
Configure a security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2, to map the incoming security token and, if required, to authenticate and authorize it.

“Creating a security profile” on page 433
You can create a security profile for use with Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2. Support is also provided for TFIM V6.1, for compatibility with previous versions of WebSphere Message Broker. You can create the security profile by using either the **mqsicreateconfigurableservice** command or an editor in the WebSphere Message Broker Explorer.

“Configuring authorization” on page 470
Configure the broker to use an external security provider to authorize an identity in a message flow. You can use either Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS) such as Tivoli Federated Identity Manager (TFIM) V6.2. Support for TFIM V6.1 is also provided for compatibility with previous versions of WebSphere Message Broker.

"Configuring for identity propagation" on page 492
To enable a message flow to perform identity propagation, the input nodes must extract the identity and the output node must propagate it.

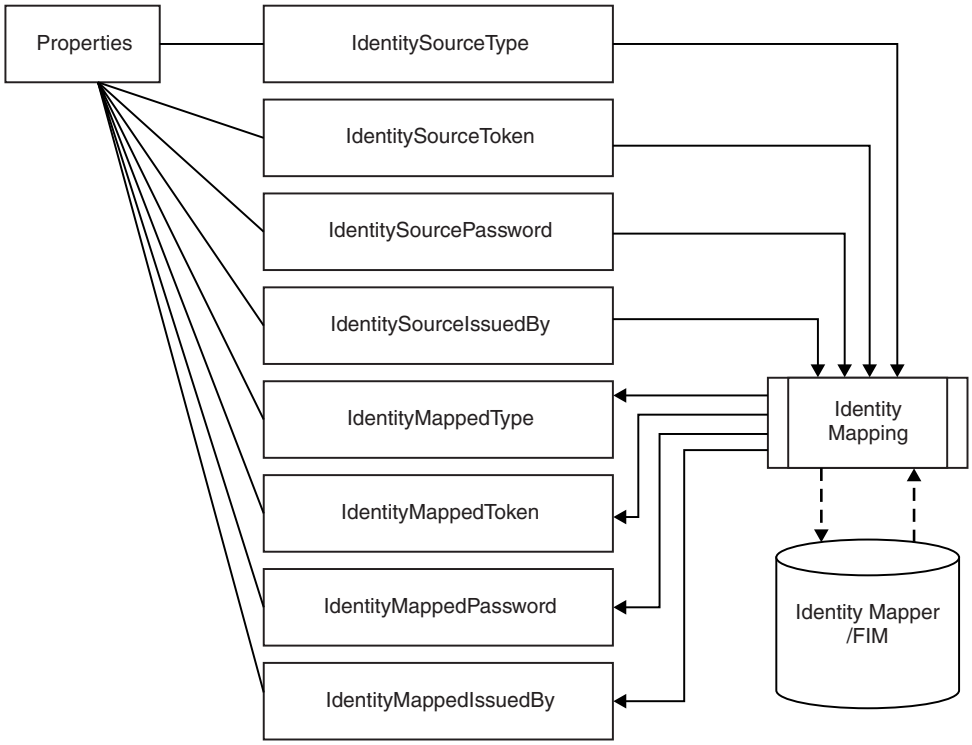
"Setting up message flow security" on page 431
Set up security on a message flow to control access based on the identity of a message passing through the message flow.

Related reference:

"SecurityPEP node" on page 4729
Use the SecurityPEP node in a message flow to invoke the message flow security manager at any point in the message flow.

Identity mapping

Identity mapping is the transformation of a security token from one format to another format, or the federation of an identity from one realm to an equivalent identity in another realm.



WebSphere Message Broker provides support for identity mapping (also known as identity federation) and token issuance and exchange. Identity mapping is the process of mapping an identity in one realm to another identity in a different realm. For example, you might map *User001* from the eSellers realm to *eSellerUser01* in the eShipping realm. Token issuance and exchange involves the mapping of a token of one type to a token of a different type. For example, an incoming Username and Password token from a client over MQ might be mapped into an equivalent SAML assertion, to be propagated to a Web Services call. Alternatively, you might exchange a SAML 1.1 assertion from a client application for an equivalent SAML 2.0 assertion for an updated backend server.

Mapping using a WS-Trust provider

The WebSphere Message Broker security manager supports mapping operations through WS-Trust V1.3 compliant security token servers (STS), such as IBM Tivoli Federated Identity Manager (TFIM) V6.2. Mapping is performed for SecurityPEP nodes and input nodes that have an associated security profile that includes a mapping operation configured with a WS-Trust V1.3 STS.

WS-Trust V1.3 (TFIM V6.2) can also be selected for authentication and authorization in the profile. However, if more than one security operation is associated with the security profile, only one WS-Trust request is issued to the STS. As a result, the STS must be configured to perform all the required operations. For example, if a TFIM V 6.2 server is specified, the TFIM module chain that is invoked must include the appropriate validate, authorize, and issue modules. If you require each operation to be performed through a separate WS-Trust call, you must use a series of SecurityPEP nodes, each associated with a different security profile that is configured for only one security operation (authentication, authorization, or mapping).

If the security profile specifies only mapping with WS-Trust v1.3 STS, the request is sent with a request type of *Issue*, whereas a mixed set of security operations sends a request type of *Validate*. When mapping is included, the STS must return a token in its response, even if it is the original token; otherwise, an error occurs.

To provide compatibility with previous versions of WebSphere Message Broker, support is also provided for TFIM V6.1.

In the broker, identity mapping is performed at the input node or SecurityPEP node, after authentication but before authorization. The source identity is passed to an identity mapper for processing. If both mapping and authorization are configured, the authorization operation uses the mapped output token rather than the source token, which means that the authorization is performed on the federated identity.

Mapping is not performed in output nodes, even if the node has been configured with a security profile.

WebSphere Message Broker supports mapping between any type of security token that is supported by the configured security provider. For more information about the support provided, see “Identity” on page 390.

When mapping from an X.509 certificate, TFIM can validate the certificate but cannot verify the identity of the original sender. However, if it is required, this verification integrity check can be performed by the SOAPInput node. For more information, see “WS-Security mechanisms” on page 768.

For more information about using TFIM V6.2 for mapping, see “Authentication, mapping, and authorization with TFIM V6.2 and TAM” on page 419.

For information about using TFIM V6.1, see “Authentication, mapping, and authorization with TFIM V6.1 and TAM” on page 416.

User-defined mapping

When you develop a message flow, you can implement a custom token mapping to be used for identity propagation. For example, you can implement a custom token

mapping using a compute node (which can be a Compute, JavaCompute, or PHPCompute node) following the input node. In the compute node, you can read the source identity values from the Properties folder, process them, then write the new identity values to the mapped identity fields. If there is no identity provided in the message, you can still use a compute node to insert some identity credentials into the mapped identity fields. The mapped identity fields are then used in place of the source identity fields by subsequent nodes. Any security operations that are configured on the input node are performed using the source identity, before you can create a new identity in the mapped identity fields (by using the compute node). However, you can include a SecurityPEP node after your compute node has created a new mapped identity.

Related concepts:

"Identity" on page 390
In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

"Authentication and validation" on page 398
Authentication is the process of establishing the identity of a user or system and verifying that the identity is valid. Applying authentication to a SAML security token involves validating the assertions that it carries and confirming that it is being processed within its validity period.

"Authorization" on page 401
Authorization is the process of verifying that an identity token has permission to access a message flow.

"Identity and security token propagation" on page 426
Identity and security token propagation enables the identity and security tokens (associated with each message) to be propagated throughout a message flow, and on to target applications through output or request nodes.

"Security profiles" on page 387
A security profile defines the security operations that are to be performed in a message flow at SecurityPEP nodes and security enabled input and output nodes.

"Authentication, mapping, and authorization with TFIM V6.1 and TAM" on page 416
Use WebSphere Message Broker, Tivoli Federated Identity Manager (TFIM) V6.1, and Tivoli Access Manager (TAM) to control authentication, mapping, and authorization.

"Authentication, mapping, and authorization with TFIM V6.2 and TAM" on page 419
You can use WebSphere Message Broker, Tivoli Federated Identity Manager (TFIM) V6.2, and Tivoli Access Manager (TAM) to control authentication, mapping, and authorization.

Related tasks:

"Configuring identity mapping" on page 463
Configure a security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2, to map the incoming security token and, if required, to authenticate and authorize it.

"Setting up message flow security" on page 431
Set up security on a message flow to control access based on the identity of a message passing through the message flow.

Related reference:

"HTTPRequest node" on page 4488
Use the HTTPRequest node to interact with a web service.

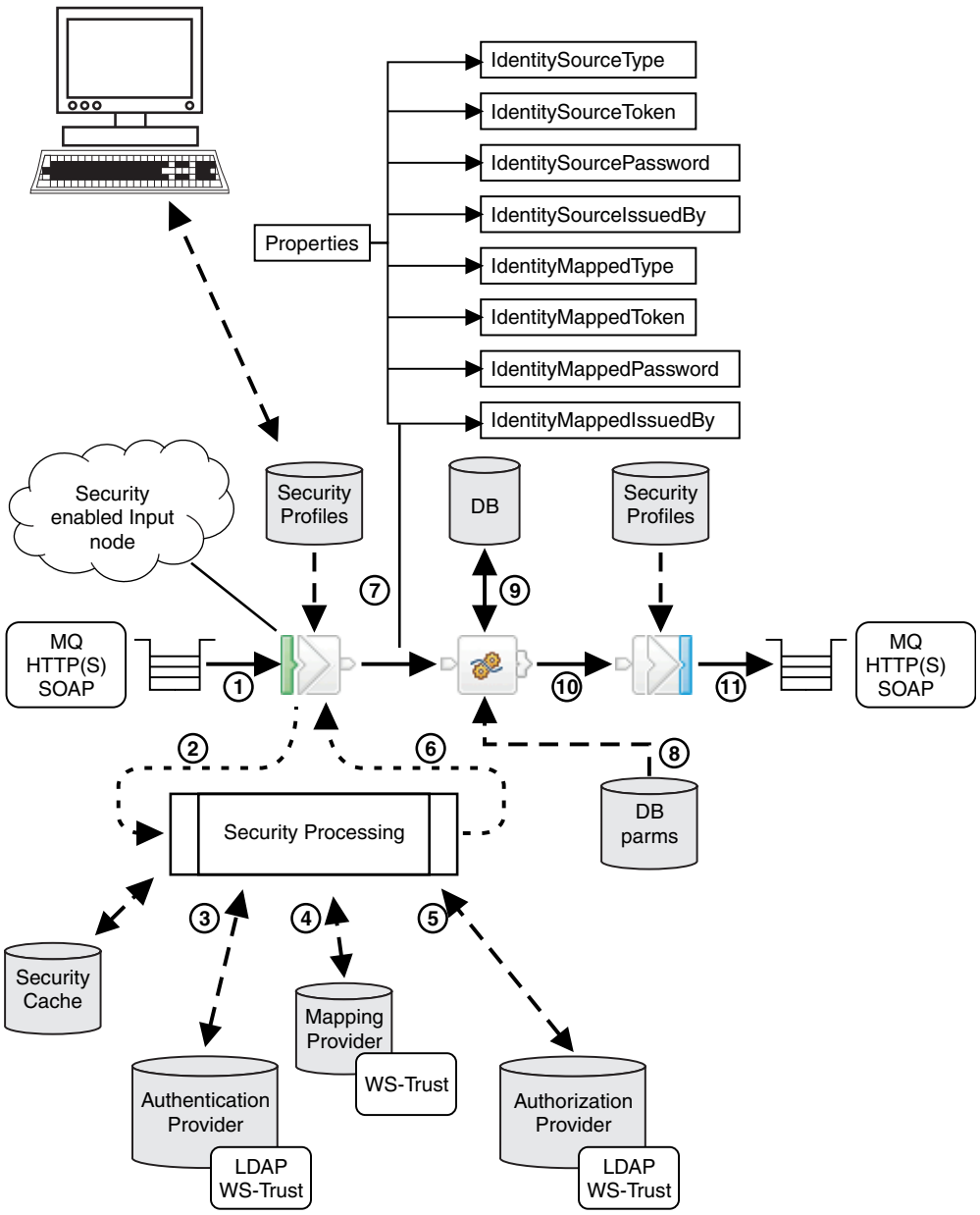
"MQOutput node" on page 4612
Use the MQOutput node to send messages to clients that connect to the broker using the WebSphere MQ Enterprise Transport and that use the MQI and AMI application programming interfaces.

"SecurityPEP node" on page 4729
Use the SecurityPEP node in a message flow to invoke the message flow security manager at any point in the message flow.

Invoking message flow security using a security enabled input node

You can invoke the message flow security manager by configuring a security enabled input node.

The following diagram shows an example message flow and gives an overview of the sequence of events that occur when an input message is received by a security enabled input node in the message flow:



The following steps explain the sequence of events that occur when a message arrives at a security enabled input node in the message flow. The numbers correspond to those in the preceding diagram:

1. When a message arrives at a security enabled input node (MQ, HTTP, SCA, or SOAP), the presence of a security profile associated with the node indicates whether message flow security is configured. SOAP nodes can implement some WS-Security capabilities without using the broker's security manager; for more information, see “Implementing WS-Security” on page 769. The broker's security manager is called to read the profile, which specifies the combination of propagation, authentication, authorization, and mapping to be performed with the identity of the message. It also specifies the external security provider (also known as the Policy Decision Point or PDP) to be used.

   You can create security profiles by using either the **mqsicreateconfigurableservice** command or an editor in the WebSphere Message Broker Explorer. You then use the Broker Archive editor to configure

the security profile on either an individual node or the whole message flow. If you associate the security profile with the message flow, the security profile applies to all security enabled input and output nodes and SecurityPEP nodes in the message flow. However, a security profile that is associated with an individual node takes precedence over a security profile that is associated with the message flow. A predefined security profile, called *Default propagation*, is provided for setting identity propagation. To explicitly set no security on a node, set the security profile to *No security*.

2. If a security profile is associated with the node or message flow, the security enabled input node extracts the identity information from the input message (based on the configuration on the node's **Security** properties page) and sets the Source Identity elements in the Properties folder. If the security tokens cannot be successfully extracted, a security exception is raised.

   If you require a SOAPInput node to use the identity in the WS-Security header (rather than an underlying transport identity), you must also define and specify an appropriate policy set and bindings for the relevant token profile. For more information, see "Policy sets" on page 774.

   For MQ, HTTP, or SCA input nodes, the **Security** properties page is used to configure the extraction of the identity. This defaults to *Transport Default*. For example, an HTTPInput node extracts a username and password from the HTTP BasicAuth header. The **Security** properties page allows the Identity token type and its location to be explicitly configured to control the extraction. This source identity information can be in a message header, the message body, or both.

   For information about the tokens that are supported by each node, see "Identity" on page 390.

3. If authentication is specified in the security profile, the security manager calls the configured security provider to authenticate the identity. A failure results in a security exception being returned to the node. The security providers that are supported by Message Broker for authentication are LDAP, WS-Trust v1.3 compliant security token servers (such as TFIM V6.2) and TFIM V6.1.

   A security cache is provided for the authentication result, which enables subsequent messages (with the same credentials) arriving at the message flow to be completed with the cached result, provided that it has not expired.

4. If identity mapping is specified in the security profile, the security manager calls the configured security provider to map the identity to an alternative identity. A failure results in a security exception being returned to the node. Otherwise, the mapped identity information is set in the Mapped Identity elements in the Properties folder.

   The security providers that are supported by Message Broker for identity mapping are WS-Trust V1.3 compliant security token servers (such as TFIM V6.2) and TFIM V6.1.

   A security cache is provided for the result of the identity mapping.

   Alternatively, you can use a SecurityPEP node at any point in the message flow to map the identity that was authenticated at the security enabled input node. For more information, see "Invoking message flow security using a SecurityPEP node" on page 411.

5. If authorization is specified in the security profile, the security manager calls the configured security provider to authorize that the identity (either mapped or source) has access to this message flow. A failure results in a security exception being returned to the node.

The security providers that are supported by Message Broker for authorization are LDAP, WS-Trust V1.3 compliant security token servers (such as TFIM V6.2) and TFIM V6.1.

A security cache is provided for the authorization result.

Alternatively, you can use a SecurityPEP node at any point in the message flow to authorize the identity that was authenticated at the security enabled input node. For more information, see "Invoking message flow security using a SecurityPEP node" on page 411.

6. When all security processing is complete, or when a security exception is raised by the message flow security manager, control returns to the input node.

   When a security exception is returned to the security enabled input node, it performs the appropriate transport handling and ends the message flow transaction. For example, an HTTPInput node returns an HTTP header with a 401 HTTP response code, without propagating to an output terminal. A SOAPInput node returns a SOAP Fault, reporting the security exception. Alternatively, if the `Treat security exceptions as normal` property is set on the security enabled input node, a security exception is propagated to the node's Failure terminal. The security enabled input node propagates to its Out terminal only if all the configured operations in the associated security profile complete successfully.
7. The message, including the Properties folder and its source and mapped identity information, is propagated down the message flow.
8. At subsequent nodes in the message flow an identity might be required to access a resource such as a database. The identity used to access such a resource is a proxy identity, either the broker's identity or an identity configured for the specific resource by using the **mqsisetdbparms** command.
9. When you are developing a message flow you can use the identity fields in the Properties folder for application processing (for example, identity-based routing or content building based on identity). Also, as an alternative to invoking mapping through a WS-Trust V1.3 enabled STS (such as TFIM V6.2) or TFIM V6.1, you can set the mapped identity fields in a compute node, such as a Compute, JavaCompute, PHPCompute, or Mapping node.
10. When the message reaches a security enabled output or request node (MQOutput, HTTPRequest, SOAPRequest, or SOAPAsyncRequest), a security profile (with propagation enabled) associated with the node indicates that the current identity token is to be propagated when the message is sent.

    If the security profile indicates that propagation is required, the mapped identity is used. If the mapped identity is not set, or if it has a token type that is not supported by the node, the source identity is used. If no identity is set, or if neither the mapped nor source identity has a token type that is supported by the node, a security exception is returned to the node.

    SOAP nodes also require the appropriate policy set and bindings for the token profile to be associated with the node.

    If you want to include a security token in the message that is issued at an output node, and if the output node is not able to propagate that type of token, you can use a compute node (before the output node) to put the token from the properties tree into the relevant message location.

    For information about the tokens that are supported by each node, see "Identity and security token propagation" on page 426.
11. The propagated identity is included in the appropriate message header when it is sent.

Related concepts:

"Message flow security overview" on page 383
WebSphere Message Broker provides a security manager, which enables you to control access to individual messages in a message flow, using the identity of the message.

"Invoking message flow security using a SecurityPEP node" on page 411
You can invoke the message flow security manager at any point in a message flow, between an input node and an output or request node, by using a SecurityPEP node.

Related tasks:

"Setting up message flow security" on page 431
Set up security on a message flow to control access based on the identity of a message passing through the message flow.

"Configuring the extraction of an identity or security token" on page 447
You can configure the SecurityPEP node or security enabled input nodes to extract the identity or security token from a message and store it in the properties tree identity fields, enabling it to be processed throughout the message flow and propagated at output or request nodes.

"Creating a security profile" on page 433
You can create a security profile for use with Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2. Support is also provided for TFIM V6.1, for compatibility with previous versions of WebSphere Message Broker. You can create the security profile by using either the **mqsicreateconfigurableservice** command or an editor in the WebSphere Message Broker Explorer.

"Configuring identity authentication and security token validation" on page 450
You can configure a message flow to perform identity authentication or security token validation using Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token sever (STS) such as Tivoli Federated Identity Manager (TFIM) Version 6.2. Support for TFIM V6.1 is also provided, for compatibility with previous versions of WebSphere Message Broker.

"Configuring identity mapping" on page 463
Configure a security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2, to map the incoming security token and, if required, to authenticate and authorize it.

"Configuring authorization" on page 470
Configure the broker to use an external security provider to authorize an identity in a message flow. You can use either Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS) such as Tivoli Federated Identity Manager (TFIM) V6.2. Support for TFIM V6.1 is also provided for compatibility with previous versions of WebSphere Message Broker.

"Configuring for identity propagation" on page 492
To enable a message flow to perform identity propagation, the input nodes must extract the identity and the output node must propagate it.

Related reference:

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"**mqsireloadsecurity** command" on page 3911
Use the **mqsireloadsecurity** command to force the immediate expiry of some or all the entries in the security cache.

"Parameter values for the securitycache component" on page 3815
Select the objects and properties associated with the securitycache component that you want to change.

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

"HTTPInput node" on page 4474
Use the HTTPInput node to receive an HTTP message from an HTTP client for processing by a message flow.

"SOAPInput node" on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

"SOAPAsyncRequest node" on page 4750
Use the SOAPAsyncRequest node with the SOAPAsyncResponse node to construct a pair of message flows that call a Web service asynchronously.

"SCAInput node" on page 4707
Use the SCAInput node with the SCAReply node to process messages from WebSphere Process Server.

"SecurityPEP node" on page 4729
Use the SecurityPEP node in a message flow to invoke the message flow security manager at any point in the message flow.

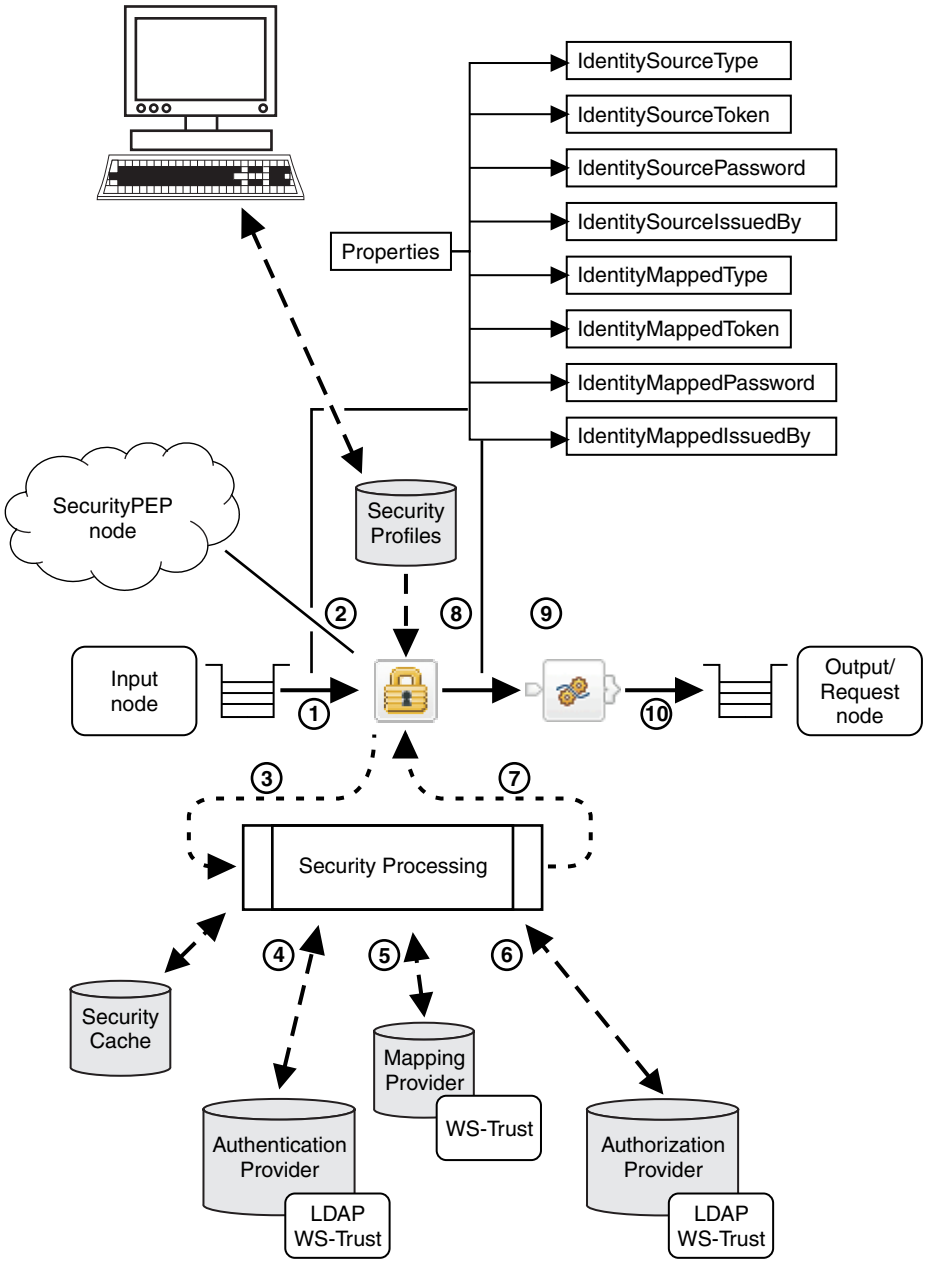
"HTTPRequest node" on page 4488
Use the HTTPRequest node to interact with a web service.

"MQOutput node" on page 4612
Use the MQOutput node to send messages to clients that connect to the broker using the WebSphere MQ Enterprise Transport and that use the MQI and AMI application programming interfaces.

Invoking message flow security using a SecurityPEP node

You can invoke the message flow security manager at any point in a message flow, between an input node and an output or request node, by using a SecurityPEP node.

The following diagram shows an example message flow and gives an overview of the sequence of events that occur when an input message is received by an input node that is not security enabled (or that has no associated security profile) and is later processed by a SecurityPEP node in the message flow:



The following steps explain the sequence of events that occur when a message arrives at an input node that is not security enabled (or that has no associated security profile). The numbers correspond to those in the preceding diagram:

1. You can use a SecurityPEP node at any point in a message flow between an input and an output or request node. The SecurityPEP node enables security to be applied in a message flow in the following situations:
   - When the message flow input node is not security enabled (for example, FileInput, TCPIPClientInput, SAPInput, and JMSInput nodes).
   - When the message flow input node is security enabled and might be configured to perform authentication operations, but the message flow is required to perform some routing or filtering before the business function being invoked is known; as a result, authorization needs to be performed later in the message flow logic.

- When the message flow includes multiple output or request nodes, which require a specific identity mapping to be performed before each node, to obtain the appropriate security tokens for propagation.

The message tree that is propagated into the SecurityPEP node includes the properties tree identity fields. These fields are empty, unless a security enabled input node (or a prior SecurityPEP node) has already extracted identity tokens, and possibly performed some security operations.

2. When a message arrives at a SecurityPEP, the presence of a security profile associated with the node indicates whether message flow security is configured. The broker's security manager is called to read the profile, which specifies the combination of propagation, authentication, authorization, and mapping to be performed with the identity of the message. It also specifies the external security provider (also known as the Policy Decision Point or PDP) to be used.

   You can create security profiles by using either the **mqsicreateconfigurableservice** command or an editor in the WebSphere Message Broker Explorer. You then use the Broker Archive editor to configure the security profile on either an individual node or the whole message flow. If you associate the security profile with the message flow, the security profile applies to all security enabled input and output and SecurityPEP nodes in the message flow. However, a security profile that is associated with an individual node takes precedence over a security profile that is associate with the message flow. Predefined security profiles are provided for setting identity propagation and for explicitly setting no security on a node.

3. If a security profile is associated with the SecurityPEP node or message flow, the node extracts the identity information from the message tree based on the node configuration and sets the Source Identity elements in the Properties folder. If the node sets a token type of *Current token*, the existing identity tokens in the Mapped Identity properties fields are used (if they exists); if there are no identity tokens in the Mapped Identity properties fields, the tokens in the Source Identity properties fields are used. If the security tokens cannot be successfully extracted, a security exception is raised and propagated to the failure terminal (if wired).

4. If authentication is specified in the security profile, the security manager calls the configured security provider to authenticate the identity. A failure results in a security exception being returned to the node. The security providers that are supported by Message Broker for authentication are LDAP, WS-Trust v1.3 compliant security token servers (such as TFIM V6.2), and TFIM V6.1.

   A security cache is provided for the authentication result, which enables subsequent messages (with the same credentials) arriving at the message flow to be completed with the cached result, provided that it has not expired.

5. If identity mapping is specified in the security profile, the security manager calls the configured security provider to map the identity to an alternative identity. A failure results in a security exception being returned to the node. Otherwise, the mapped identity information is set in the Mapped Identity elements in the Properties folder.

   The security providers that are supported by Message Broker for identity mapping are WS-Trust V1.3 compliant security token servers (such as TFIM V6.2) and TFIM V6.1.

   A security cache is provided for the result of the identity mapping.

6. If authorization is specified in the security profile, the security manager calls the configured security provider to authorize that the identity (either mapped or source) has access to this message flow. A failure results in a security exception being returned to the node.

   The security providers that are supported by Message Broker for authorization are LDAP, WS-Trust V1.3 compliant security token servers (such as TFIM V6.2) and TFIM V6.1.

   A security cache is provided for the authorization result.
7. When all security processing is complete, or when a security exception is raised by the message flow security manager, control returns to the SecurityPEP node.

   When a security exception is returned to the SecurityPEP node, the exception is either propagated to the failure terminal if it is connected, or returned to the preceding node as a recoverable exception. The SecurityPEP node propagates to its Out terminal only if all the configured operations in the associated security profile complete successfully.
8. The message, including the populated Properties folder and its source and mapped identity information, is propagated down the message flow.
9. When you are developing a message flow, you can use the identity fields in the Properties folder for application processing (for example, identity-based routing or content building based on identity). If the identity is to be propagated in an outbound message from an output or request node that does not support propagation of the token, you can use a compute node (including a Compute, JavaCompute, PHPCompute, or Mapping node), to move the identity token into the required transport header or message body location.
10. When the message reaches an output node, a security profile associated with the node can indicate whether an identity is to be taken from the Properties folder and propagated when the message is sent. Only specific transport nodes can propagate tokens that are the default for the transport; any other token type must be handled by a compute node, as described above.

Related concepts:

"Message flow security overview" on page 383
WebSphere Message Broker provides a security manager, which enables you to control access to individual messages in a message flow, using the identity of the message.

"Invoking message flow security using a security enabled input node" on page 406
You can invoke the message flow security manager by configuring a security enabled input node.

Related tasks:

"Setting up message flow security" on page 431
Set up security on a message flow to control access based on the identity of a message passing through the message flow.

"Configuring the extraction of an identity or security token" on page 447
You can configure the SecurityPEP node or security enabled input nodes to extract the identity or security token from a message and store it in the properties tree identity fields, enabling it to be processed throughout the message flow and propagated at output or request nodes.

"Creating a security profile" on page 433
You can create a security profile for use with Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2. Support is also provided for TFIM V6.1, for compatibility with previous versions of WebSphere Message Broker. You can create the security profile by using either the **mqsicreateconfigurableservice** command or an editor in the WebSphere Message Broker Explorer.

"Configuring identity authentication and security token validation" on page 450
You can configure a message flow to perform identity authentication or security token validation using Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token sever (STS) such as Tivoli Federated Identity Manager (TFIM) Version 6.2. Support for TFIM V6.1 is also provided, for compatibility with previous versions of WebSphere Message Broker.

"Configuring identity mapping" on page 463
Configure a security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2, to map the incoming security token and, if required, to authenticate and authorize it.

"Configuring authorization" on page 470
Configure the broker to use an external security provider to authorize an identity in a message flow. You can use either Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS) such as Tivoli Federated Identity Manager (TFIM) V6.2. Support for TFIM V6.1 is also provided for compatibility with previous versions of WebSphere Message Broker.

"Configuring for identity propagation" on page 492
To enable a message flow to perform identity propagation, the input nodes must extract the identity and the output node must propagate it.

Related reference:

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"**mqsireloadsecurity** command" on page 3911
Use the **mqsireloadsecurity** command to force the immediate expiry of some or all the entries in the security cache.

"Parameter values for the securitycache component" on page 3815
Select the objects and properties associated with the securitycache component that you want to change.

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

"HTTPInput node" on page 4474
Use the HTTPInput node to receive an HTTP message from an HTTP client for processing by a message flow.

"SOAPInput node" on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

"SOAPAsyncRequest node" on page 4750
Use the SOAPAsyncRequest node with the SOAPAsyncResponse node to construct a pair of message flows that call a Web service asynchronously.

"SCAInput node" on page 4707
Use the SCAInput node with the SCAReply node to process messages from WebSphere Process Server.

"SecurityPEP node" on page 4729
Use the SecurityPEP node in a message flow to invoke the message flow security manager at any point in the message flow.

"HTTPRequest node" on page 4488
Use the HTTPRequest node to interact with a web service.

"MQOutput node" on page 4612
Use the MQOutput node to send messages to clients that connect to the broker using the WebSphere MQ Enterprise Transport and that use the MQI and AMI application programming interfaces.

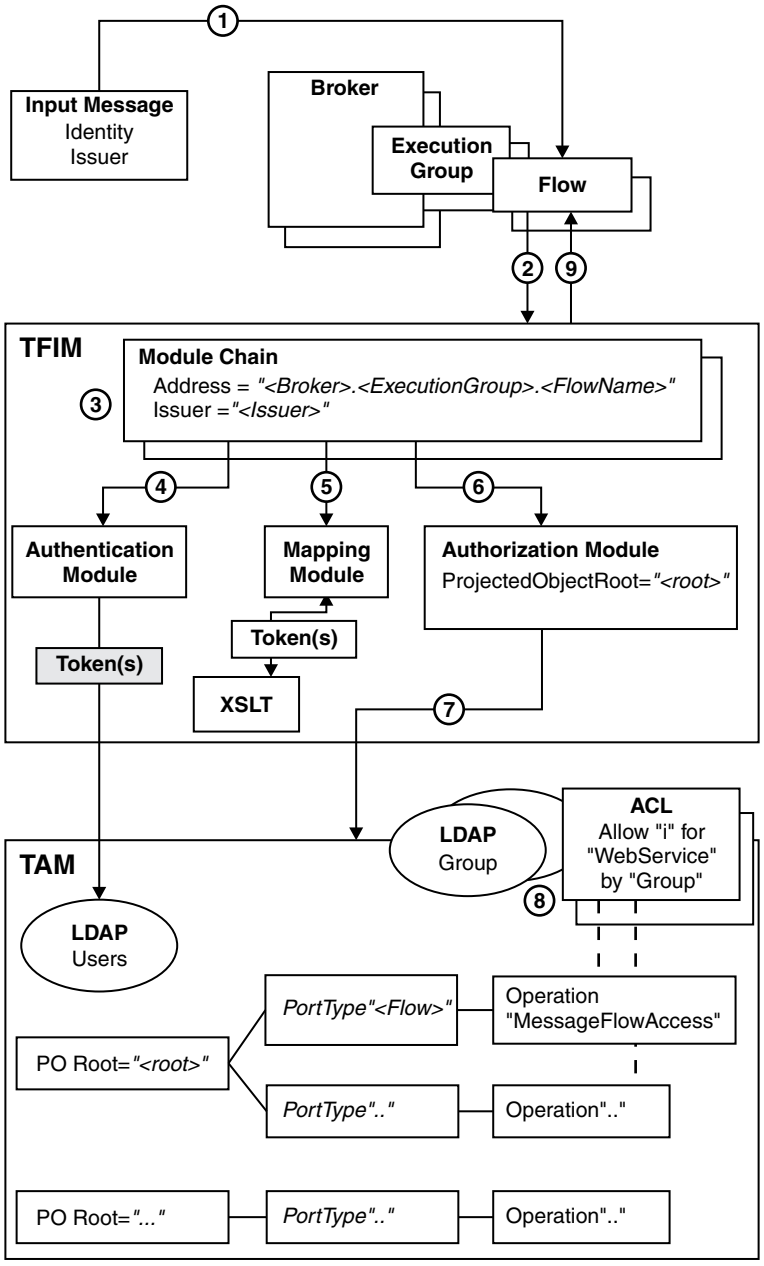
Authentication, mapping, and authorization with TFIM V6.1 and TAM

Use WebSphere Message Broker, Tivoli Federated Identity Manager (TFIM) V6.1, and Tivoli Access Manager (TAM) to control authentication, mapping, and authorization.

Note: Support for TFIM V6.1 is included for compatibility with previous versions of WebSphere Message Broker. If possible, upgrade to TFIM V6.2 and refer to the information in "Authentication, mapping, and authorization with TFIM V6.2 and TAM" on page 419.

WebSphere Message Broker makes a single TFIM WS-Trust call for an input node that is configured with a TFIM security profile, which means that a single module chain must be configured to perform all the required authentication, mapping, and authorization operations.

The following diagram shows the configuration of WebSphere Message Broker, TFIM, and TAM to enable authentication, mapping, and authorization of an identity in a message flow:



The numbers in the preceding diagram correspond to the following sequence of events:

1. A message enters a message flow.
2. A WS-Trust request is issued by the broker, with these properties:
   - RequestType = Validate
   - Identity = Token(s) from input message
   - Issuer = Issuer from input message
   - AppliesTo Address = *"Broker.ExecutionGroup.FlowName"*
   - PortType = *"FlowName"*
   - Operation = "MessageFlowAccess"

3. TFIM selects a module chain to process the WS-Trust request, based on the AppliesTo Address and Issuer properties of the request.
4. A module chain can perform authentication if it includes a module (such as a UsernameTokenSTSModule or X509STSModule) in `validate` mode.
5. A module chain can perform mapping by using an XSLTransformationModule in `mapping` mode to manipulate the identity information.
6. A module chain can perform authorization by using an AuthorizationSTSModule in `other` mode. The module chain must be configured with a *Protected Object Root* value.
7. The AuthorizationSTSModule performs the authorization check by making a request to TAM with these properties:
   - Action = "i" (invoke)
   - Action Group = "WebService"
   - Protected Object = *"ProtectedObjectRoot.FlowName.MessageFlowAccess"*

   where *"i"* and *"WebService"* are default values used by an AuthorizationSTSModule; and *FlowName* and *MessageFlowAccess* are the WS-Trust request PortType and Operation values.
8. TAM processes the authorization request by:
   a. Finding the Access Control Lists (ACLs) associated with protected object *"<ProtectedObjectRoot>.<FlowName>.MessageFlowAccess"*.
   b. Checking whether or not the ACLs grant action "i" on action group "WebService" to the user (with the user either named directly, or by membership of a named group).
9. The WS-Trust reply is returned to the broker. If this action is the result of a mapping request, the WS-Trust reply contains the mapped identity token.

For further information about how to configure TFIM, see the IBM Tivoli Federated Identity Manager Information Center.

For information about how to configure TAM, see the IBM Tivoli Access Manager Information Center.

Related concepts:

"Identity" on page 390
In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

"Authentication and validation" on page 398
Authentication is the process of establishing the identity of a user or system and verifying that the identity is valid. Applying authentication to a SAML security token involves validating the assertions that it carries and confirming that it is being processed within its validity period.

"Authorization" on page 401
Authorization is the process of verifying that an identity token has permission to access a message flow.

"Identity mapping" on page 403
Identity mapping is the transformation of a security token from one format to another format, or the federation of an identity from one realm to an equivalent identity in another realm.

"Security profiles" on page 387
A security profile defines the security operations that are to be performed in a message flow at SecurityPEP nodes and security enabled input and output nodes.

Related tasks:

"Configuring identity authentication and security token validation" on page 450
You can configure a message flow to perform identity authentication or security token validation using Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token sever (STS) such as Tivoli Federated Identity Manager (TFIM) Version 6.2. Support for TFIM V6.1 is also provided, for compatibility with previous versions of WebSphere Message Broker.

"Configuring identity mapping" on page 463
Configure a security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2, to map the incoming security token and, if required, to authenticate and authorize it.

"Creating a security profile" on page 433
You can create a security profile for use with Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2. Support is also provided for TFIM V6.1, for compatibility with previous versions of WebSphere Message Broker. You can create the security profile by using either the **mqsicreateconfigurableservice** command or an editor in the WebSphere Message Broker Explorer.

"Configuring authorization" on page 470
Configure the broker to use an external security provider to authorize an identity in a message flow. You can use either Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS) such as Tivoli Federated Identity Manager (TFIM) V6.2. Support for TFIM V6.1 is also provided for compatibility with previous versions of WebSphere Message Broker.

"Configuring authorization with TFIM V6.1" on page 483
You can configure a message flow to perform authorization on an identity by using Tivoli Federated Identity Manager (TFIM) V6.1.

"Setting up message flow security" on page 431
Set up security on a message flow to control access based on the identity of a message passing through the message flow.

Related reference:

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

"HTTPInput node" on page 4474
Use the HTTPInput node to receive an HTTP message from an HTTP client for processing by a message flow.

"SOAPInput node" on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

Authentication, mapping, and authorization with TFIM V6.2 and TAM

You can use WebSphere Message Broker, Tivoli Federated Identity Manager (TFIM) V6.2, and Tivoli Access Manager (TAM) to control authentication, mapping, and authorization.

WebSphere Message Broker makes a single TFIM WS-Trust call for an input node or SecurityPEP node that is configured with a WS-Trust V1.3 STS security profile. As a result, a single module chain must be configured to perform all the required authentication, mapping, and authorization operations.

When you use a WS-Trust v1.3 STS for authentication, authorization, or mapping, a request is made to the trust service with the following parameters, which control the STS processing. If you are using TFIM V6.2, these parameters are used in the selection of the TFIM module chain:

Parameter	Value
RequestType	The type of request issued to the trust service. Valid values are: **`Issue`** This value can be specified when mapping is the only operation that is set to `WS-Trust V1.3 STS` in the security profile. It is not valid if `WS-Trust V1.3 STS` is specified for authentication or authorization. The namespace qualified value is `http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue`, which shows in TFIM V6.2 as *Issue Oasis URI*. **`Validate`** This value must be set when the security profile also includes authentication or authorization (in addition to mapping) for the same `WS-Trust V1.3 STS` provider. The namespace qualified value is `http://docs.oasis-open.org/ws-sx/ws-trust/200512/Validate`, which shows in TFIM V6.2 as *Validate Oasis URI*.
Issuer	This value is determined by the effective setting of the `IssuedBy` property on the **Basic** tab of the SecurityPEP node or the **Security** tab of the input node.

Parameter	Value

AppliesTo	This value is determined by the type of node:
	MQInput or SCAInput node with MQ binding:
	The WebSphere MQ IRI of the node's input queue; for example:
	`wmq://msg/queue/queue_name@queue_manager_name`
	HTTPInput, SOAPInput, or SOAPAsyncResponse node with HTTP binding:
	The endpoint URL; for example:
	`http://myflow/myInputNodePath`
	SecurityPEP node with a default (blank) WS-Trust AppliesTo address:
	The URN for the message flow that contains the node; for example:
	`urn:/broker_name.execution_group_name.flow_name`
	SecurityPEP node with WS-Trust AppliesTo address set on the Advanced tab of the node:
	The URI value configured in the property. This value is typically the URL of the target service that is used when you invoke a mapping operation to obtain the required token for the following request node; for example:
	`http://remotehost.ibm.com:9080/targetservice`
	You can also set the `AppliesTo service name` and `AppliesTo port type` properties on the **Advanced** tab of the node. The WS-Trust request includes these optional elements only when they are configured. These values are typically valid QNames; for example:
	`http://myservice.mycom.com:myservicename`
	When you set these properties in the SecurityPEP node, you must configure them in the TFIM module chain:
	• In the service name and port type TFIM properties, the information to the left of the colon must match the namespace URI of the WS-Addressing namespace that is used for the PortType and ServiceName elements in the WS-Trust request set by the broker, which is:
	`http://www.w3.org/2005/08/addressing`
	• The information to the right of the colon in the service name and port type TFIM properties must match the value configured on the SecurityPEP node. You can also configure a regular expression in TFIM to specify a match

This section describes an authorization configuration that you can use to perform the authorization operation with TFIM V6.2 and TAM.

In the security profile, set the TFIM V6.2 endpoint for the authorization operation. When you create a module chain to be used by a security enabled input node or SecurityPEP node, and resolved by *AppliesTo* information, you must include the TFIM TAMAuthorizationSTModule to invoke TAM authorization.

The TAMAuthorizationSTSModule requires the following TFIM STS universal user context attributes:

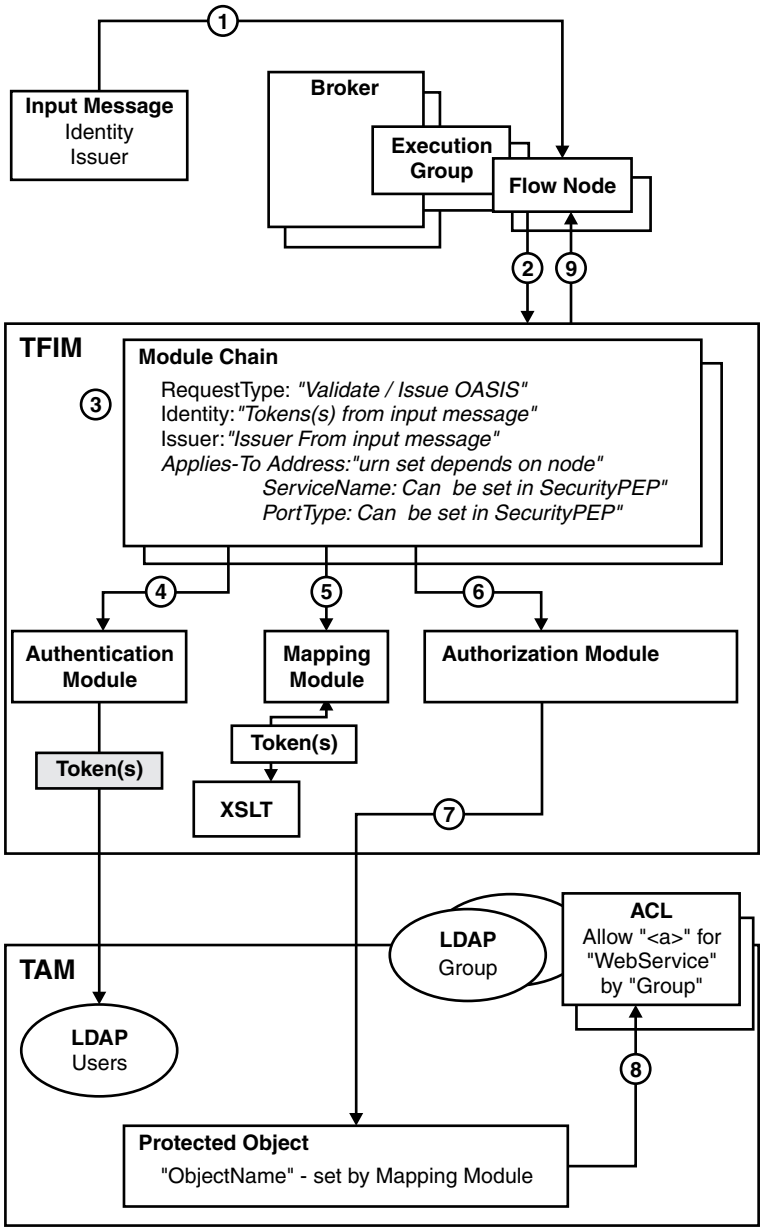
PrincipalName
The username to be authorized. This username must exist in your TAM user repository.

ObjectName
The TAM object name of the resource on which an authorization check is to be made. Typically this is derived from the *AppliesTo* information that is passed by the message flow security manager from the security enabled input node or SecurityPEP node.

Action
The TAM action to be authorized; for example, *x* (eXecute).

The TAM Access Control List (ACL), which determines the authorization decision, is located in the TAM protected object space using the path that is set on the *ObjectName* attribute of the TFIM STS universal user context input to the TAMAuthorizationSTSModule module.

The following diagram shows the configuration of WebSphere Message Broker, TFIM V6.2, and TAM to enable authentication, mapping, and authorization of an identity in a message flow:



The numbers in the preceding diagram correspond to the following sequence of events:

1. A message enters a message flow.
2. A WS-Trust request is issued by the broker, with the *RequestType*, *Issuer*, and *AppliesTo* properties set.
3. TFIM selects a module chain to process the WS-Trust request, based on the *RequestType*, *Issuer*, and *AppliesTo* properties of the request.
4. A module chain can perform authentication if it includes a module in *Validate* mode that is appropriate to the token type that is being passed in the request from the message flow input message. For example, a Username and Password token can be authenticated using a UsernameTokenSTSModule .

5. The module chain must perform some mapping by using an XSLTransformationModule in mapping mode to manipulate the identity information and to provide the required context attributes in the TFIM stsuser object for use by subsequent modules.
6. A module chain can perform authorization in TAM by using the TAMAuthorizationSTSModule.
7. The TAMAuthorizationSTSModule performs the authorization check by making a request to TAM with these properties:
   - Action = *a* (where *a* is the stsuser context action attribute). For example, *x* for eXecute could be set using the following code:

     ```
     <stsuuser:ContextAttributes>
         <!-- Action -->
         <stsuuser:Attribute name="Action" type="urn:ibm:names:ITFIM:stsmodule:tamazn">
           <stsuuser:Value>x</stsuuser:Value>
         </stsuuser:Attribute>
       </stsuuser:ContextAttributes>
     ```

   - Action Group = *WebService*
   - Protected Object = *ProtectedObjectName* (where *ProtectedObjectName* is the stsuser context action attribute). For example, *x* for eXecute could be set using the following code:

     ```
     <stsuuser:ContextAttributes>
         <!-- ObjectName -->
         <stsuuser:Attribute name="ObjectName" type="urn:ibm:names:ITFIM:stsmodule:tamazn">
           <stsuuser:Value>ProtectedObjectName</stsuuser:Value>
         </stsuuser:Attribute>
       </stsuuser:ContextAttributes>
     ```

     Typically, *ProtectedObjectName* is set conditionally from the *AppliesTo* information in the request.
8. TAM processes the authorization request by:
   a. Finding the Access Control Lists (ACLs) associated with protected object *ProtectedObjectName*
   b. Checking whether the ACLs grant action *a* on action group *WebService* to the user (the user is named either directly or indirectly, through membership of a named group).
9. The WS-Trust reply is returned to the broker. If this action is the result of a mapping request, the WS-Trust reply contains the mapped identity token.

For further information about how to configure TFIM, see the IBM Tivoli Federated Identity Manager Information Center.

For information about how to configure TAM, see the IBM Tivoli Access Manager Information Center.

Related concepts:

"Identity" on page 390
In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

"Authentication and validation" on page 398
Authentication is the process of establishing the identity of a user or system and verifying that the identity is valid. Applying authentication to a SAML security token involves validating the assertions that it carries and confirming that it is being processed within its validity period.

"Authorization" on page 401
Authorization is the process of verifying that an identity token has permission to

access a message flow.

"Identity mapping" on page 403
Identity mapping is the transformation of a security token from one format to another format, or the federation of an identity from one realm to an equivalent identity in another realm.

"Security profiles" on page 387
A security profile defines the security operations that are to be performed in a message flow at SecurityPEP nodes and security enabled input and output nodes.

Related tasks:

"Configuring identity authentication and security token validation" on page 450
You can configure a message flow to perform identity authentication or security token validation using Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token sever (STS) such as Tivoli Federated Identity Manager (TFIM) Version 6.2. Support for TFIM V6.1 is also provided, for compatibility with previous versions of WebSphere Message Broker.

"Configuring identity mapping" on page 463
Configure a security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2, to map the incoming security token and, if required, to authenticate and authorize it.

"Creating a security profile" on page 433
You can create a security profile for use with Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2. Support is also provided for TFIM V6.1, for compatibility with previous versions of WebSphere Message Broker. You can create the security profile by using either the **mqsicreateconfigurableservice** command or an editor in the WebSphere Message Broker Explorer.

"Configuring authorization" on page 470
Configure the broker to use an external security provider to authorize an identity in a message flow. You can use either Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS) such as Tivoli Federated Identity Manager (TFIM) V6.2. Support for TFIM V6.1 is also provided for compatibility with previous versions of WebSphere Message Broker.

"Setting up message flow security" on page 431
Set up security on a message flow to control access based on the identity of a message passing through the message flow.

Related reference:

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

"SecurityPEP node" on page 4729
Use the SecurityPEP node in a message flow to invoke the message flow security manager at any point in the message flow.

"HTTPInput node" on page 4474
Use the HTTPInput node to receive an HTTP message from an HTTP client for processing by a message flow.

"SOAPInput node" on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

Identity and security token propagation

Identity and security token propagation enables the identity and security tokens (associated with each message) to be propagated throughout a message flow, and on to target applications through output or request nodes.

In an enterprise system, you can use different physical identities or security tokens (such as user names, certificates, and SAML assertions) to represent a single logical identity through different parts of the enterprise. The propagation of an identity or security token ensures that the logical identity is kept throughout the system by mapping between the various physical forms as necessary. For example, a message might enter the system using a certificate, but a user name token might be required for server processing of the message. Identity mapping is used to convert from the certificate to the username token, and identity propagation ensures that the mapped identity is placed in the correct place for the outbound transport.

When an output or request node propagates an identity, the mapped identity is used. If the mapped identity is not set, or if it has a token type that is not supported by the node, the source identity is used. If no identity is set, or if neither the mapped nor source identity has a token type that is supported by the node, a security exception is thrown by the node.

The output nodes that support identity propagation are:

- CICSRequest
- HTTPRequest
- IMSRequest
- MQOutput
- SAPRequest
- SCAAsyncRequest
- SCARequest
- SOAPAsyncRequest
- SOAPRequest

The following table shows the support that is provided by the message flow security manager for the propagation of the different types of security token. For more information about these security tokens, see “Identity” on page 390.

Table 9. Support for security token types - token propagation

Token type (format)	Broker security manager support	Token propagated in	
Username	Username tokens are supported for propagation by the following nodes: • CICSRequest • HTTPRequest • IMSRequest • MQOutput • SCAAsyncRequest • SCARequest	**CICS**	The request security credentials
		HTTP	BasicAuth header
		IMS	The request security credentials
		MQ	MQMD.UserIdentifier transport header

Table 9. Support for security token types - token propagation (continued)

Token type (format)	Broker security manager support	Token propagated in
Username and password	Username and password tokens are supported for propagation by the following nodes: • CICSRequest • HTTPRequest • IMSRequest • MQOutput • SAPRequest • SCAAsyncRequest • SCARequest • SOAPAsyncRequest • SOAPRequest	**CICS** The request security credentials **HTTP** BasicAuth header **IMS** The request security credentials **MQ** MQMD.UserIdentifier transport header **SAP** The request security credentials **SOAP** • BasicAuth header, if there is no policy set and binding • SOAP header, if a policy set and binding sets the Username token profile • Kerberos client credentials, if a policy set and binding sets the Kerberos token profile
SAML assertion	SAML tokens are supported for propagation by the following nodes: • SOAPRequest • SOAPAsyncRequest	SOAP header when a policy set and binding sets the SAML token profile
X.509 certificate	X.509 tokens are supported for propagation by the following nodes: • SOAPRequest • SOAPAsyncRequest	SOAP header when a policy set and binding sets the X.509 binary token profile
LTPA v2 token	LTPA v2 tokens are supported for propagation by the following nodes: • SOAPRequest • SOAPAsyncRequest	SOAP header when a policy set and binding sets the LTPA token profile
Universal WSSE token	Universal WSSE tokens are not supported for propagation by any node.	

For information about how to configure a message flow to propagate a message identity, see “Configuring for identity propagation” on page 492. For more information about how one physical identity is converted to another, see “Identity mapping” on page 403.

Related concepts:

"Identity" on page 390
In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

"Authentication and validation" on page 398
Authentication is the process of establishing the identity of a user or system and verifying that the identity is valid. Applying authentication to a SAML security token involves validating the assertions that it carries and confirming that it is being processed within its validity period.

"Identity mapping" on page 403
Identity mapping is the transformation of a security token from one format to another format, or the federation of an identity from one realm to an equivalent identity in another realm.

"Authorization" on page 401
Authorization is the process of verifying that an identity token has permission to access a message flow.

"Security profiles" on page 387
A security profile defines the security operations that are to be performed in a message flow at SecurityPEP nodes and security enabled input and output nodes.

Related tasks:

"Configuring identity mapping" on page 463
Configure a security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2, to map the incoming security token and, if required, to authenticate and authorize it.

"Creating a security profile" on page 433
You can create a security profile for use with Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2. Support is also provided for TFIM V6.1, for compatibility with previous versions of WebSphere Message Broker. You can create the security profile by using either the **mqsicreateconfigurableservice** command or an editor in the WebSphere Message Broker Explorer.

"Configuring for identity propagation" on page 492
To enable a message flow to perform identity propagation, the input nodes must extract the identity and the output node must propagate it.

"Configuring authentication with HTTP basic authentication" on page 451
Use a security profile to configure HTTP basic authentication in the HTTPRequest or SOAPInput nodes.

"Setting up message flow security" on page 431
Set up security on a message flow to control access based on the identity of a message passing through the message flow.

"Propagating security credentials to an SAP request" on page 2065
The SAPRequest node can use an identity that is present in the Properties folder of the message tree structure for the security credentials in a request, by using the `Propagate` property on the security profile that is defined on the node.

"Propagating security credentials to IMS" on page 2144
The IMSRequest node can use an identity that is present in the Properties folder of the message tree structure for the security credentials in a request, by using the `Propagate` property on the security profile that is defined for the node.

"Propagating security credentials to CICS Transaction Server for z/OS" on page 2208
The CICSRequest node can use an identity that is present in the Properties folder of the message tree structure for the security credentials in a request, by using the `Propagate` property on the security profile that is defined on the node.

Related reference:

"CICSRequest node" on page 4321
Use the CICSRequest node to call CICS Transaction Server for z/OS programs over TCP/IP-based IP InterCommunications (IPIC) protocol.

"HTTPInput node" on page 4474
Use the HTTPInput node to receive an HTTP message from an HTTP client for processing by a message flow.

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

"SOAPInput node" on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

"SecurityPEP node" on page 4729
Use the SecurityPEP node in a message flow to invoke the message flow security manager at any point in the message flow.

"SCAInput node" on page 4707
Use the SCAInput node with the SCAReply node to process messages from WebSphere Process Server.

"SCARequest node" on page 4719
Use the SCARequest node to send a request to WebSphere Process Server. The node is a synchronous request and response node, and blocks after sending the request until the response is received. The node can also send one-way requests.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

"IMSRequest node" on page 4504
Use the IMSRequest node to send a request to run a transaction on a local or remote IMS system, and wait for a response. IMS Connect must be configured and running on the IMS system.

Security exception processing

A security exception is raised when a message flow security failure occurs during security processing in an input node or SecurityPEP node.

Security exceptions are processed in a different way from other errors on the input node. An error is typically caught on the input node and routed down the Failure terminal for error processing, but security exceptions are not processed in the same way. By default, the broker does not allow security exceptions to be caught within the message flow, but backs the message out or returns an error (as in the case of HTTP). Security exceptions in input nodes are managed in this way to prevent a security denial of service attack filling the logs and destabilizing the system.

However, security exceptions in SecurityPEP nodes are managed in a different way. If a security operation fails in a SecurityPEP node, a security exception is raised, wrapped in a normal recoverable exception, which invokes the error handling that is provided by the message flow.

If you have designed the message flow to be in a secure area and you want to explicitly perform processing of security exceptions, you can select the **Treat Security Exceptions as normal exceptions** property on the MQInput or HTTPInput nodes. This property causes security exceptions to be processed in the same way as other exceptions in the message flow.

If you associate the Default Propagation security profile with an output or request node, the token type of the mapped or source security token must be the same as the transport default for that node; otherwise, a security exception occurs. For example, for an MQOutput node, the token type must be *Username*, for an HTTPRequest node, the token type must be *Username + Password*, and for a SOAPRequest node, the token type must be the type that is defined in either the policy set and binding or the transport binding.

For information on how to diagnose the causes of security exceptions, see "Diagnosing security problems" on page 496.

Related concepts:

"Identity" on page 390
In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

"Authentication and validation" on page 398
Authentication is the process of establishing the identity of a user or system and verifying that the identity is valid. Applying authentication to a SAML security token involves validating the assertions that it carries and confirming that it is being processed within its validity period.

"Identity mapping" on page 403
Identity mapping is the transformation of a security token from one format to another format, or the federation of an identity from one realm to an equivalent identity in another realm.

"Authorization" on page 401
Authorization is the process of verifying that an identity token has permission to access a message flow.

"Identity and security token propagation" on page 426
Identity and security token propagation enables the identity and security tokens (associated with each message) to be propagated throughout a message flow, and on to target applications through output or request nodes.

"Security profiles" on page 387
A security profile defines the security operations that are to be performed in a message flow at SecurityPEP nodes and security enabled input and output nodes.

Related tasks:

"Configuring the extraction of an identity or security token" on page 447
You can configure the SecurityPEP node or security enabled input nodes to extract the identity or security token from a message and store it in the properties tree identity fields, enabling it to be processed throughout the message flow and propagated at output or request nodes.

"Setting up message flow security" on page 431
Set up security on a message flow to control access based on the identity of a message passing through the message flow.

"Diagnosing security problems" on page 496
This topic explains how to find out why access to a secured flow has been denied.

Related reference:

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

"HTTPInput node" on page 4474
Use the HTTPInput node to receive an HTTP message from an HTTP client for processing by a message flow.

Setting up message flow security

Set up security on a message flow to control access based on the identity of a message passing through the message flow.

About this task

You can configure the broker to perform end-to-end processing of an identity carried in a message through a message flow. Administrators can configure security at message flow level, controlling access based on the identity flowed in a message. This security mechanism is independent of both the transport and the message format.

To set up security for a message flow, perform the tasks described in the following topics:

Procedure

1. "Creating a security profile" on page 433
2. "Configuring the extraction of an identity or security token" on page 447
3. "Configuring identity authentication and security token validation" on page 450
4. "Configuring identity mapping" on page 463
5. "Configuring authorization" on page 470
6. "Configuring for identity propagation" on page 492
7. "Database security" on page 495
8. "Diagnosing security problems" on page 496

What to do next

If the message flow is a Web service implemented by using the "SOAP nodes" on page 1609, and the identity is to be taken from the "WS-Security" on page 765 header tokens, you must also create appropriate "Policy sets" on page 774 and bindings, then configure them on the relevant SOAP nodes (in addition to the security profile). See "Associating policy sets and bindings with message flows and nodes" on page 785.

To work with an identity, you must configure the policy sets and bindings for the relevant capabilities:

- To work with a Username and Password identity, configure the policy sets and bindings for "Username token capabilities" on page 791.
- To work with an X.509 Certificate identity, configure the policy sets and bindings for "X.509 certificate token capabilities" on page 795.

  In the policy set binding, the `Certificates` mode of the X.509 certificate authentication token must be set as `Trust Any` (rather than `Trust Store`), so that

the certificate is passed to the security provider defined by the security profile. Setting *Trust Store* causes the certificate to be validated in the local broker trust store.

- To work with a SAML assertion, configure the policy sets and bindings for "SAML token capabilities" on page 804.
- To work with an LTPA token, configure the policy sets and bindings for "LTPA token capabilities" on page 813.
- To work with a Kerberos ticket, configure the policy sets and bindings for "Kerberos token capabilities" on page 809.

For more information, see "Policy Sets and Policy Set Bindings editor: Authentication and Protection Tokens panel" on page 6854.

Related concepts:

"Identity" on page 390
In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

"Authentication and validation" on page 398
Authentication is the process of establishing the identity of a user or system and verifying that the identity is valid. Applying authentication to a SAML security token involves validating the assertions that it carries and confirming that it is being processed within its validity period.

"Identity mapping" on page 403
Identity mapping is the transformation of a security token from one format to another format, or the federation of an identity from one realm to an equivalent identity in another realm.

"Authorization" on page 401
Authorization is the process of verifying that an identity token has permission to access a message flow.

"Identity and security token propagation" on page 426
Identity and security token propagation enables the identity and security tokens (associated with each message) to be propagated throughout a message flow, and on to target applications through output or request nodes.

"Security profiles" on page 387
A security profile defines the security operations that are to be performed in a message flow at SecurityPEP nodes and security enabled input and output nodes.

"Message flow security overview" on page 383
WebSphere Message Broker provides a security manager, which enables you to control access to individual messages in a message flow, using the identity of the message.

Related tasks:

"Configuring the extraction of an identity or security token" on page 447
You can configure the SecurityPEP node or security enabled input nodes to extract the identity or security token from a message and store it in the properties tree identity fields, enabling it to be processed throughout the message flow and propagated at output or request nodes.

"Creating a security profile" on page 433
You can create a security profile for use with Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2. Support is also provided for TFIM V6.1, for compatibility with previous versions of WebSphere Message Broker. You can create the security profile by using either the **mqsicreateconfigurableservice** command or an editor in the WebSphere Message Broker Explorer.

"Creating a security profile for LDAP" on page 435
Create a security profile for use with Lightweight Directory Access Protocol (LDAP) or Secure LDAP (LDAPS), by using either the **mqsicreateconfigurableservice** command or an editor in the WebSphere Message Broker Explorer.

"Configuring identity authentication and security token validation" on page 450
You can configure a message flow to perform identity authentication or security token validation using Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token sever (STS) such as Tivoli Federated Identity Manager (TFIM) Version 6.2. Support for TFIM V6.1 is also provided, for compatibility with previous versions of WebSphere Message Broker.

"Configuring identity mapping" on page 463
Configure a security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2, to map the incoming security token and, if required, to authenticate and authorize it.

"Configuring authorization" on page 470
Configure the broker to use an external security provider to authorize an identity in a message flow. You can use either Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS) such as Tivoli Federated Identity Manager (TFIM) V6.2. Support for TFIM V6.1 is also provided for compatibility with previous versions of WebSphere Message Broker.

"Configuring for identity propagation" on page 492
To enable a message flow to perform identity propagation, the input nodes must extract the identity and the output node must propagate it.

Creating a security profile

You can create a security profile for use with Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2. Support is also provided for TFIM V6.1, for compatibility with previous versions of WebSphere Message Broker. You can create the security profile by using either the **mqsicreateconfigurableservice** command or an editor in the WebSphere Message Broker Explorer.

About this task

Before you can enable security on a node or a message flow, you need to create a security profile that defines the security operations that you want to perform.

You can create a security profile for use with external security providers to provide the required security enforcement and mapping. You can configure the security profile to use different security providers for different security functions; for example, you might use LDAP for authentication and WS-Trust V1.3 STS for mapping and authorization.

If you want to extract and propagate an identity without security enforcement or mapping, you can use the supplied security profile called Default Propagation. The Default Propagation profile is a predefined profile that requests only identity propagation.

To create a security profile, see:

- "Creating a security profile for LDAP" on page 435
- "Creating a security profile for WS-Trust V1.3 (TFIM V6.2)" on page 440
- "Creating a security profile for TFIM V6.1" on page 444

Related concepts:

"Identity" on page 390
In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

"Authentication and validation" on page 398
Authentication is the process of establishing the identity of a user or system and verifying that the identity is valid. Applying authentication to a SAML security token involves validating the assertions that it carries and confirming that it is being processed within its validity period.

"Identity mapping" on page 403
Identity mapping is the transformation of a security token from one format to another format, or the federation of an identity from one realm to an equivalent identity in another realm.

"Authorization" on page 401
Authorization is the process of verifying that an identity token has permission to access a message flow.

"Identity and security token propagation" on page 426
Identity and security token propagation enables the identity and security tokens (associated with each message) to be propagated throughout a message flow, and on to target applications through output or request nodes.

"Security profiles" on page 387
A security profile defines the security operations that are to be performed in a message flow at SecurityPEP nodes and security enabled input and output nodes.

"Security exception processing" on page 429
A security exception is raised when a message flow security failure occurs during security processing in an input node or SecurityPEP node.

Related tasks:

"Configuring the extraction of an identity or security token" on page 447
You can configure the SecurityPEP node or security enabled input nodes to extract the identity or security token from a message and store it in the properties tree identity fields, enabling it to be processed throughout the message flow and propagated at output or request nodes.

"Configuring identity authentication and security token validation" on page 450
You can configure a message flow to perform identity authentication or security token validation using Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token sever (STS) such as Tivoli Federated Identity Manager (TFIM) Version 6.2. Support for TFIM V6.1 is also provided, for compatibility with previous versions of WebSphere Message Broker.

"Configuring identity mapping" on page 463
Configure a security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2, to map the incoming security token and, if required, to authenticate and authorize it.

"Configuring authorization" on page 470
Configure the broker to use an external security provider to authorize an identity in a message flow. You can use either Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS) such as Tivoli Federated Identity Manager (TFIM) V6.2. Support for TFIM V6.1 is also provided for compatibility with previous versions of WebSphere Message Broker.

"Configuring for identity propagation" on page 492
To enable a message flow to perform identity propagation, the input nodes must extract the identity and the output node must propagate it.

"Setting up message flow security" on page 431
Set up security on a message flow to control access based on the identity of a message passing through the message flow.

Related reference:

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

Creating a security profile for LDAP:

Create a security profile for use with Lightweight Directory Access Protocol (LDAP) or Secure LDAP (LDAPS), by using either the **mqsicreateconfigurableservice** command or an editor in the WebSphere Message Broker Explorer.

Before you begin

Before you start:

Ensure that you have an LDAP server that is LDAP Version 3 compliant, for example:

- IBM Tivoli Directory Server
- Microsoft Active Directory
- OpenLDAP.

About this task

If your LDAP directory does not permit login by unrecognized user IDs, and does not grant search access rights on the subtree, you must also set up a separate authorized login ID that the broker can use for the search. For information on how to do this, see "Configuring authorization with LDAP" on page 472 or "Configuring authentication with LDAP" on page 453.

Creating a security profile using mqsicreateconfigurableservice:
About this task

You can use the **mqsicreateconfigurableservice** command to create a security profile that uses LDAP for authentication, authorization, or both. The security profile ensures that each message has an authenticated ID and is authorized for the message flow.

Procedure

1. Open a command window that is configured for your environment.
2. Enter the **mqsicreateconfigurableservice** command on the command line. For example:

   ```
   mqsicreateconfigurableservice WBRK_BROKER -c SecurityProfiles -o LDAP
   -n authentication,authenticationConfig,authorization,authorizationConfig,propagation,rejectBla
   -v "LDAP,\"ldap://ldap.acme.com:389/ou=sales,o=acme.com\",LDAP,
   \"ldap://ldap.acme.com:389/cn=All Sales,ou=acmegroups,o=acme.com\",TRUE,TRUE
   ```

   You must enclose the LDAP URL (which contains commas) with escaped double quotation marks (\" and \") so that the URL commas are not confused with the comma separator of the value parameter of **mqsicreateconfigurableservice**.

   If the LDAP URL includes an element name with a space, in this case `cn=All Sales`, the set of values after the -v flag must be enclosed by double quotation marks, (")

For more information about the structure of the command, refer to the "**mqsicreateconfigurableservice** command" on page 3849.

You can define the security-specific parts of the command in the following way:

a. Set the **authentication** to *LDAP*. This ensures that the incoming identity is validated.

b. Set the **authenticationConfig** using the following syntax:

ldap[s]**://***server***[:***port***]/***baseDN***[?[***uid_attr***][?[base|sub]]]**

For example:

```
ldap://ldap.acme.com:389/ou=sales,o=acme.com
ldaps://localhost:636/ou=sales,o=acme?cn?base
```

ldap: (Required) Fixed protocol string.

s: (Optional) Specifies whether SSL should be used. Default is not to use SSL.

server: (Required) The name or IP address of the LDAP server to contact.

port: (Optional) The port to connect to. Default is 389 (non-SSL). For LDAP servers with SSL enabled, the port is typically 636.

baseDN
(Required) String defining the base distinguished name (DN) of all users in the directory. If users exist in different subtrees, specify a common subtree under which a search on the username uniquely resolves to the required user entry, and set the *sub* attribute.

uid_attr:
(Optional) String defining the attribute to which the incoming username maps, typically uid, CN, or email address. Default is *uid*.

base|sub:
(Optional) Defines whether to perform a base or subtree search. If `base` is defined, the authentication is faster because the DN of the user can be constructed from the *uid_att*, *username*, and *baseDN* values. If *sub* is selected, a search must be performed before the DN can be resolved. Default is *sub*.

c. Set the **authorization** to *LDAP*. This ensures that the incoming identity is checked for group membership in LDAP.

d. Set the **authorizationConfig** using the following syntax:

```
ldap[s]://server[:port]/groupDN[?[member_attr]
[?[base|sub][?[x-userBaseDN=baseDN,
x-uid_attr=uid_attr]]]]
```

For example:

```
ldap://ldap.acme.com:389/cn=All Sales,ou=acmegroups,
o=acme.com?uniquemember?sub?x-userBaseDN=ou=sales%2co=ibm.com,
x-uid_attr=emailaddress
```

ldap: (Required) Fixed protocol string

s: (Optional) Specifies whether SSL is used. Default is not to use SSL.

server: (Required) The name or IP address of the LDAP server to contact.

port: (Optional) The port to connect to. Default is 389 (non-SSL). For LDAP servers with SSL enabled, the port is typically 636.

groupDN
(Required) Fully defined distinguished name of the group in which users must be members to be granted access.

member_attr:
(Optional) The attribute of the group used to filter the search. Default is to look for both **member** and **uniquemember** attributes.

The following options are required only if authentication has not preceded the authorization, and if the authentication configuration string has not been specified. If the authentication configuration string has been specified, the following parameters are ignored and those provided by the **baseDN**, **uid_attr**, and **[base|sub]** for authentication are used instead:

base|sub:
(Optional) Defines whether to perform a base or subtree search. If `base` is defined, the authentication is faster because the DN of the user can be constructed from uid_att + username + baseDN. If `sub` is selected, a search must be performed before the DN can be resolved. Default is `sub`.

baseDN
(Optional) String defining the base distinguished name of all users in the directory. Must be preceded by the string `x-userBaseDN`. Any commas in the BaseDN must be rendered as `%2c`.

x-uid_attr:
(Optional) String defining the attribute to which the incoming username should map, typically uid, CN, or email address. Default is uid. Must be preceded by the string `x-uid_attr`.

When you submit the command from a batch (.bat) file or command (.cmd) file, if the LDAP URL includes an extension with LDAP URL "percent hex hex" escaped characters (for example, a comma replaced by `%2c`, or a space replaced by `%20`), the percent signs must be escaped from the batch interpreter with an extra percent sign (`%%`). For example:

```
mqsicreateconfigurableservice WBRK_BROKER -c SecurityProfiles -o LDAP_URI_FUN
-n authentication,authenticationConfig,authorization,authorizationConfig -v
"LDAP,\"ldap://ldap.acme.com:389/ou=sales,o=acme.com?emailaddress?sub\",
LDAP,\"ldap://ldap.acme.com:389/cn=All Sales,ou=acmegroups,
o=acme.com?report?base?x-BaseDN=ou=sales%%2co=acme.com,
x-uid_attr=emailaddress\""
```

The selected group must be defined on the LDAP server, and all of the required users must be members of the group.

3. If you need to reconfigure the security profile after it has been created, use the **mqsichangeproperties** command.

Creating a security profile using the WebSphere Message Broker Explorer:
About this task

You can use the WebSphere Message Broker Explorer to create a security profile for LDAP.

Procedure

1. In the WebSphere Message Broker Explorer, right-click on the broker with which you want to work, and click **Properties**.
2. In the Properties window, select the Security tab, and click **Security Profiles**. The Security Profiles window is displayed, containing a list of existing security profiles for the broker on the left, and a pane in which you can configure the profile on the right.
3. Click **Add** to create a new profile and add it to the list. You can edit the name of the security profile by highlighting it in the list and pressing **F2**.

4. Configure the security profile using the entry fields on the right side of the pane:
   a. Select the type of **Authentication** required. This can be `LDAP`, `TFIM`, or `NONE`.
   b. If you have selected LDAP for authentication, edit the following fields in the **LDAP Parameters** section:
      - LDAP Host
      - LDAP baseDN
      - LDAP uid attr
      - LDAP search Scope

      The values that you enter in the **LDAP Parameters** fields create a configuration string, which is displayed in the **Authentication Config** field. For information about the valid values for the parameters, see "Creating a security profile using mqsicreateconfigurableservice" on page 435.
   c. Select the type of **Mapping** required. This can be either `TFIM` or `NONE`.
   d. If you have selected TFIM for mapping, type the URL of the TFIM server in the **TFIM Configuration** field of the **TFIM Parameters** section.

      The value that you specify in the **TFIM Configuration** field creates a configuration string, which is displayed in the **Mapping Config** field.
   e. Select the type of **Authorization** required. This can be `LDAP`, `TFIM`, or `NONE`.
   f. If you have selected LDAP for authorization, edit the following fields in the **LDAP Parameters** section:
      - LDAP Host
      - LDAP baseDN
      - LDAP uid attr
      - LDAP search Scope
      - LDAP group baseDN
      - LDAP group member.

      The values that you enter in the **LDAP Parameters** fields create a configuration string, which is displayed in the **Authorization Config** field. For information about the valid values for the parameters, see "Creating a security profile using mqsicreateconfigurableservice" on page 435.
   g. In the **Propagation** field, specify whether you require the identity to be propagated. The default is `False`.
   h. In the **Reject Empty Password** field, specify whether you want the security manager to reject a username that has an empty password token, without passing it to LDAP. The default is `False`, which means that a username is passed to LDAP even if it has an empty password token.
   i. In the **Password Value** field, select the way in which the password is displayed in the properties folder. The options are:

      PLAIN The password appears in the Properties folder as plain text.

      OBFUSCATE
      The password appears in the Properties folder as base64 encoding.

      MASK The password appears in the Properties folder as four asterisks (****).
5. Click **Finish** to deploy the security profile to the broker.

Results

To delete an existing security profile, select the profile in the list and then click **Delete**.

Related concepts:

"Identity" on page 390
In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

"Authentication and validation" on page 398
Authentication is the process of establishing the identity of a user or system and verifying that the identity is valid. Applying authentication to a SAML security token involves validating the assertions that it carries and confirming that it is being processed within its validity period.

"Identity mapping" on page 403
Identity mapping is the transformation of a security token from one format to another format, or the federation of an identity from one realm to an equivalent identity in another realm.

"Authorization" on page 401
Authorization is the process of verifying that an identity token has permission to access a message flow.

"Identity and security token propagation" on page 426
Identity and security token propagation enables the identity and security tokens (associated with each message) to be propagated throughout a message flow, and on to target applications through output or request nodes.

"Security profiles" on page 387
A security profile defines the security operations that are to be performed in a message flow at SecurityPEP nodes and security enabled input and output nodes.

"Security exception processing" on page 429
A security exception is raised when a message flow security failure occurs during security processing in an input node or SecurityPEP node.

Related tasks:

"Configuring the extraction of an identity or security token" on page 447
You can configure the SecurityPEP node or security enabled input nodes to extract the identity or security token from a message and store it in the properties tree identity fields, enabling it to be processed throughout the message flow and propagated at output or request nodes.

"Configuring identity authentication and security token validation" on page 450
You can configure a message flow to perform identity authentication or security token validation using Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token sever (STS) such as Tivoli Federated Identity Manager (TFIM) Version 6.2. Support for TFIM V6.1 is also provided, for compatibility with previous versions of WebSphere Message Broker.

"Configuring identity mapping" on page 463
Configure a security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2, to map the incoming security token and, if required, to authenticate and authorize it.

"Creating a security profile" on page 433
You can create a security profile for use with Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2. Support is also provided for TFIM V6.1, for compatibility with previous versions of WebSphere Message Broker. You can create the security profile by using either the **mqsicreateconfigurableservice** command or an editor in the WebSphere Message Broker Explorer.

"Configuring authorization" on page 470
Configure the broker to use an external security provider to authorize an identity in a message flow. You can use either Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS) such as Tivoli Federated Identity Manager (TFIM) V6.2. Support for TFIM V6.1 is also provided for compatibility with previous versions of WebSphere Message Broker.

"Configuring for identity propagation" on page 492
To enable a message flow to perform identity propagation, the input nodes must extract the identity and the output node must propagate it.

"Setting up message flow security" on page 431
Set up security on a message flow to control access based on the identity of a message passing through the message flow.

Related reference:

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

"HTTPInput node" on page 4474
Use the HTTPInput node to receive an HTTP message from an HTTP client for processing by a message flow.

"SOAPInput node" on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

"SOAPAsyncRequest node" on page 4750
Use the SOAPAsyncRequest node with the SOAPAsyncResponse node to construct a pair of message flows that call a Web service asynchronously.

"HTTPRequest node" on page 4488
Use the HTTPRequest node to interact with a web service.

"MQOutput node" on page 4612
Use the MQOutput node to send messages to clients that connect to the broker using the WebSphere MQ Enterprise Transport and that use the MQI and AMI application programming interfaces.

Creating a security profile for WS-Trust V1.3 (TFIM V6.2):

You can create a security profile for a WS-Trust V1.3 compliant Security Token Server (STS), for example, Tivoli Federated Identity Manager (TFIM) V6.2, for any combination of the following security operations: authentication, authorization, and mapping.

About this task

You can use either the **mqsicreateconfigurableservice** command or an editor in the WebSphere Message Broker Explorer to create the security profile:

- "Creating a profile using mqsicreateconfigurableservice"
- "Creating a profile using the WebSphere Message Broker Explorer" on page 442

Creating a profile using mqsicreateconfigurableservice:
About this task

To create a security profile that uses a WS-Trust V1.3 compliant Security Token Server (STS), you can use the **mqsicreateconfigurableservice** command by setting the configuration parameter to the full URL of the STS. The URL must consist of the transport scheme, host name, port, and path. For TFIM V6.2 WS-Trust V1.3 endpoint, the path is `/TrustServerWST13/services/RequestSecurityToken`. For example:

```
http://stsserver.mycompany.com:9080/TrustServerWST13/services/RequestSecurityToken
```

Procedure

To create a security profile that uses WS-Trust v1.3 for mapping, enter the following command:

```
mqsicreateconfigurableservice brokername -c SecurityProfiles
-o profilename -n mapping,mappingConfig
-v "WS-Trust v1.3 STS",http://stsserver.mycompany.com:9080/TrustServerWST13/services/RequestSecuri
```

If the URL specifies an address beginning with `https://`, an SSL secured connection is used for requests to the WS-Trust v1.3 server. For example, to create a security profile that uses an HTTPS connection to WS-Trust v1.3 for mapping, enter the following command:

```
mqsicreateconfigurableservice brokername -c SecurityProfiles
-o profilename -n mapping,mappingConfig
-v "WS-Trust v1.3 STS",https://stsserver.mycompany.com:9080/TrustServerWST13/services/RequestSecur
```

In addition to specifying the security profile URL as an address beginning with `https://`, you can configure the following advanced parameters, by setting broker environment variables:

MQSI_STS_SSL_PROTOCOL
The version of the SSL protocol to be used. Valid values are:
- SSL
- SSLv3
- TLS

The initial value is `SSLv3`.

MQSI_STS_SSL_ALLOWED_CIPHERS
A space-separated list of the encryption ciphers that can be used. For a list of all the cipher suites that are supported by WebSphere Message Broker, see the Java product information for your operating system. For operating systems that use IBM Java, see Appendix A of the IBM JSSE2 Guide: http://www.ibm.com/developerworks/java/jdk/security/60/secguides/jsse2Docs/JSSE2RefGuide.html

MQSI_STS_REQUEST_TIMEOUT
The STS request timeout, specified in seconds. The initial value is `100`. For information about providing environment variables to the broker, see "Setting up a command environment" on page 213.

If WS-Trust v1.3 STS is selected for more than one operation (for example, for authentication and mapping), the WS-Trust v1.3 server URL must be identical for all the operations, and is therefore specified only once.
The following example creates a security profile that uses TFIM V6.2 for authentication, mapping, and authorization:

```
mqsicreateconfigurableservice MYBROKER -c SecurityProfiles -o MyWSTrustProfile
-n authentication,mapping,authorization,propagation,mappingConfig
-v "WS-Trust v1.3 STS","WS-Trust v1.3 STS","WS-Trust v1.3 STS",TRUE,http://stsserver.mycompany.com:90
```

Creating a profile using the WebSphere Message Broker Explorer:
About this task

You can use the WebSphere Message Broker Explorer to create a security profile for using WS-Trust v1.3.

Procedure

1. In the WebSphere Message Broker Explorer, right-click on the broker with which you want to work, and click **Properties**.
2. In the Properties window, select the Security tab, and click **Security Profiles**. The Security Profiles window is displayed, containing a list of existing security profiles for the broker on the left and, on the right, a pane in which you can configure the profile.
3. Click **Add** to create a new profile and add it to the list. You can edit the name of the security profile by highlighting it in the list and pressing **F2**. The security profile name must not include spaces.
4. Configure the security profile using the entry fields on the right side of the pane:
   a. Select the security provider that you require for `Authentication`, `Mapping`, and `Authorization`. When you select `WS-Trust v1.3 STS` for any of these options, the **STS URI** field in the **Security Token Service (STS) Parameters** group is enabled.
   b. Type the URL of the WS-Trust v1.3 STS into the **STS URL** field. The STS URL must contain the following URL parts:
      - Transport scheme (http or https)
      - Host name (a fully qualified domain name)
      - Port
      - Path (for example, for TFIM V6.2: `/TrustServerWST13/services/RequestSecurityToken`)

      For example:

      ```
      http://stsserver.mycompany.com:9080/TrustServerWST13/services/RequestSecurityToken
      ```

      The URL that you enter forms a configuration string, which is displayed in one or more of the configuration fields (**Authentication Config**, **Mapping Config**, and **Authorization Config**), depending on the security operations that are configured to use `WS-Trust v1.3 STS`.

      For more information about the valid values for the configuration parameter, see “Creating a profile using mqsicreateconfigurableservice” on page 441.
   c. In the **Propagation** field, specify whether you require the identity to be propagated. The default is `False`.

d. In the **Password Value** field, select the way in which the password is displayed in the properties folder. The password is optional, and is required only when the token type is `Username + Password`. The options are:

   PLAIN The password is shown in the Properties folder as plain text.

   OBFUSCATE
   The password is shown in the Properties folder as base64 encoding.

   MASK The password is shown in the Properties folder as four asterisks (****).

5. Click **Finish** to deploy the security profile to the broker.

Related concepts:

"Identity" on page 390
In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

"Authentication and validation" on page 398
Authentication is the process of establishing the identity of a user or system and verifying that the identity is valid. Applying authentication to a SAML security token involves validating the assertions that it carries and confirming that it is being processed within its validity period.

"Identity mapping" on page 403
Identity mapping is the transformation of a security token from one format to another format, or the federation of an identity from one realm to an equivalent identity in another realm.

"Authorization" on page 401
Authorization is the process of verifying that an identity token has permission to access a message flow.

"Identity and security token propagation" on page 426
Identity and security token propagation enables the identity and security tokens (associated with each message) to be propagated throughout a message flow, and on to target applications through output or request nodes.

"Security profiles" on page 387
A security profile defines the security operations that are to be performed in a message flow at SecurityPEP nodes and security enabled input and output nodes.

Related tasks:

"Configuring the extraction of an identity or security token" on page 447
You can configure the SecurityPEP node or security enabled input nodes to extract the identity or security token from a message and store it in the properties tree identity fields, enabling it to be processed throughout the message flow and propagated at output or request nodes.

"Configuring identity authentication and security token validation" on page 450
You can configure a message flow to perform identity authentication or security token validation using Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token sever (STS) such as Tivoli Federated Identity Manager (TFIM) Version 6.2. Support for TFIM V6.1 is also provided, for compatibility with previous versions of WebSphere Message Broker.

"Configuring identity mapping" on page 463
Configure a security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2, to map the incoming security token and, if required, to authenticate and authorize it.

"Creating a security profile" on page 433
You can create a security profile for use with Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2. Support is also provided for TFIM V6.1, for compatibility with previous versions of WebSphere Message Broker. You can create the security profile by using either the **mqsicreateconfigurableservice** command or an editor in the WebSphere Message Broker Explorer.

"Configuring authorization" on page 470
Configure the broker to use an external security provider to authorize an identity in a message flow. You can use either Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS) such as Tivoli Federated Identity Manager (TFIM) V6.2. Support for TFIM V6.1 is also provided for compatibility with previous versions of WebSphere Message Broker.

"Configuring for identity propagation" on page 492
To enable a message flow to perform identity propagation, the input nodes must extract the identity and the output node must propagate it.

"Setting up message flow security" on page 431
Set up security on a message flow to control access based on the identity of a message passing through the message flow.

Related reference:

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"SecurityPEP node" on page 4729
Use the SecurityPEP node in a message flow to invoke the message flow security manager at any point in the message flow.

Creating a security profile for TFIM V6.1:

You can create a security profile for Tivoli Federated Identity Manager (TFIM) V6.1 for any combination of the following functions: authentication, authorization, and mapping. You can use either the **mqsicreateconfigurableservice** command or an editor in the WebSphere Message Broker Explorer to create the security profile.

About this task

Note: Support for TFIM V6.1 is included for compatibility with previous versions of WebSphere Message Broker. If possible, upgrade to TFIM V6.2 and follow the instructions in "Creating a security profile for WS-Trust V1.3 (TFIM V6.2)" on page 440.

Creating a profile using mqsicreateconfigurableservice:
About this task

To create a security profile that uses TFIM V6.1, you can use the **mqsicreateconfigurableservice** command by setting the configuration parameter to the URL of the TFIM server. For example: `http://tfimserver.mycompany.com:9080`

Procedure

To create a security profile that uses TFIM V6.1 for mapping, enter the following command:

```
mqsicreateconfigurableservice brokername -c SecurityProfiles -o profilename
-n mapping,mappingConfig -v TFIM,http://tfimserver.mycompany.com:9080
```

If the URL specifies an address beginning with `https://`, an SSL secured connection is used for requests to the TFIM server. For example, to create a security profile that uses an HTTPS connection to TFIM for mapping, enter the following command:

```
mqsicreateconfigurableservice brokername -c SecurityProfiles -o profilename
-n mapping,mappingConfig -v TFIM,https://tfimserver.mycompany.com:9443
```

where `https://tfimserver.mycompany.com:9443` is the address of the TFIM server. If TFIM is selected for more than one operation (for example, for authentication and mapping), the TFIM server URL must be identical for all the operations, and is therefore specified only once.
The following example creates a security profile that uses TFIM for authentication, mapping, and authorization:

```
mqsicreateconfigurableservice MYBROKER -c SecurityProfiles -o TFIM
-n authentication,mapping,authorization,propagation,mappingConfig
-v TFIM,TFIM,TFIM,TRUE,http://tfimhost1.ibm.com:9080
```

Creating a profile for TFIM V6.1 using the WebSphere Message Broker Explorer:
About this task

You can use the WebSphere Message Broker Explorer to create a security profile for using TFIM V6.1.

Procedure

1. In the WebSphere Message Broker Explorer, right-click on the broker with which you want to work, and click **Properties**.
2. In the Properties window, select the Security tab, and click **Security Profiles**. The Security Profiles window is displayed, containing a list of existing security profiles for the broker on the left and, on the right, a pane in which you can configure the profile.
3. Click **Add** to create a new profile and add it to the list. You can edit the name of the security profile by highlighting it in the list and pressing **F2**.
4. Configure the security profile using the entry fields on the right side of the pane:
   a. Select the type of **Authentication**, **Mapping**, and **Authorization** required. If you select `TFIM V6.1` for any of these options, the **TFIM Configuration** field at the bottom of the pane is enabled.
   b. If you have selected TFIM V6.1 for authentication, mapping or authorization, type the URL of the TFIM server into the **TFIM Configuration** field. The URL that you enter forms a configuration string, which is displayed in one or more of the configuration fields (**Authentication Config**, **Mapping Config**, and **Authorization Config**) depending on the entry fields that have TFIM selected.

      For more information about the valid values for the configuration parameter, see "Creating a profile using mqsicreateconfigurableservice" on page 444.
   c. In the **Propagation** field, specify whether you require the identity to be propagated. The default is `False`.
   d. In the **Password Value** field, select the way in which the password is displayed in the properties folder. The options are:

      PLAIN The password is shown in the Properties folder as plain text.

      OBFUSCATE
      The password is shown in the Properties folder as base64 encoding.

MASK The password is shown in the Properties folder as four asterisks (****).

5. Click **Finish** to deploy the security profile to the broker.

Related concepts:

"Identity" on page 390
In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

"Authentication and validation" on page 398
Authentication is the process of establishing the identity of a user or system and verifying that the identity is valid. Applying authentication to a SAML security token involves validating the assertions that it carries and confirming that it is being processed within its validity period.

"Identity mapping" on page 403
Identity mapping is the transformation of a security token from one format to another format, or the federation of an identity from one realm to an equivalent identity in another realm.

"Authorization" on page 401
Authorization is the process of verifying that an identity token has permission to access a message flow.

"Identity and security token propagation" on page 426
Identity and security token propagation enables the identity and security tokens (associated with each message) to be propagated throughout a message flow, and on to target applications through output or request nodes.

"Security profiles" on page 387
A security profile defines the security operations that are to be performed in a message flow at SecurityPEP nodes and security enabled input and output nodes.

Related tasks:

"Configuring the extraction of an identity or security token" on page 447
You can configure the SecurityPEP node or security enabled input nodes to extract the identity or security token from a message and store it in the properties tree identity fields, enabling it to be processed throughout the message flow and propagated at output or request nodes.

"Configuring identity authentication and security token validation" on page 450
You can configure a message flow to perform identity authentication or security token validation using Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token sever (STS) such as Tivoli Federated Identity Manager (TFIM) Version 6.2. Support for TFIM V6.1 is also provided, for compatibility with previous versions of WebSphere Message Broker.

"Configuring identity mapping" on page 463
Configure a security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2, to map the incoming security token and, if required, to authenticate and authorize it.

"Creating a security profile" on page 433
You can create a security profile for use with Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2. Support is also provided for TFIM V6.1, for compatibility with previous versions of WebSphere Message Broker. You can create the security profile by using either the **mqsicreateconfigurableservice** command or an editor in the WebSphere Message Broker Explorer.

"Configuring authorization" on page 470
Configure the broker to use an external security provider to authorize an identity in a message flow. You can use either Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS) such as Tivoli Federated Identity Manager (TFIM) V6.2. Support for TFIM V6.1 is also provided for compatibility with previous versions of WebSphere Message Broker.

"Configuring for identity propagation" on page 492
To enable a message flow to perform identity propagation, the input nodes must extract the identity and the output node must propagate it.

"Setting up message flow security" on page 431
Set up security on a message flow to control access based on the identity of a message passing through the message flow.

Related reference:

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

Configuring the extraction of an identity or security token

You can configure the SecurityPEP node or security enabled input nodes to extract the identity or security token from a message and store it in the properties tree identity fields, enabling it to be processed throughout the message flow and propagated at output or request nodes.

Before you begin

Before you start:

Check that an appropriate security profile exists or create a new security profile. See "Creating a security profile" on page 433.

About this task

If an input node or SecurityPEP node is associated with a profile that specifies a security operation (authentication, mapping, or authorization), or specifies propagation as enabled, the node can retrieve an identity or security token from the message bit stream.

- An MQInput node, with the `Identity token type` security property set to `Transport default`, retrieves the UserIdentifier element from the message descriptor (MQMD) and puts it into the Identity Source Token element of the Properties folder. At the same time, it sets the Identity Source Type element to `username` and the Identity Source Issued By element to `MQMD.PutApplName` (the put application name).
- An HTTPInput node, with the `Identity token type` security property set to `Transport default`, retrieves the BasicAuth header from the HTTP request, decodes it, and puts it into the Identity Source Token and Password elements in the Properties folder. At the same time, it sets the Identity Source Type element to `username + Password` and the Identity Source Issued By element to the HTTP header `UserAgent` property.
- A SOAPInput node retrieves the appropriate tokens as defined by the configured WS-Security policy sets and bindings, or (if they are not set), the transport binding determines the token type; for example, HTTP transport is BasicAuth. The SOAPInput node then populates the identity source fields in the Properties

folder with the retrieved tokens. With a Kerberos policy set and bindings, the token type is a Username containing the Service Principal Name (SPN) from the Kerberos ticket.

- A SecurityPEP node, with the `Identity token type` property set to `Current token`, can use the token that has been extracted by an upstream input or SecurityPEP node and stored in the Properties folder.

In some cases, the information extracted from the transport headers is not set or is insufficient to perform authentication or authorization. For example, for authentication to occur, a Username + Password type token is required; however, with WebSphere MQ, only a username is available, which means that the incoming identity has to be trusted. However, you can increase security by applying transport-level security using WebSphere MQ Extended Security Edition.

If the transport header cannot provide the required identity credentials, the information must be provided as part of the body of the incoming message. To enable the identity information to be taken from the body of the message, you must specify the location of the information by using either the **Security** tab on the HTTP, MQ, and SCA input nodes or the **Basic** tab on the SecurityPEP node, or by configuring the required policy set and bindings WS-Security profile on the SOAP node. A SOAP node with a Kerberos policy set and bindings extracts a Username token containing the Service Principal Name (SPN) of the Kerberos ticket.

Procedure

1. In **Identity Token Type**, specify the type of identity token that is in the message. The type can have one of the following values:
   - `Transport Default` (on the security enabled input nodes)
   - `Current token` (on the SecurityPEP node)
   - `Username`
   - `Username and password`
   - `X.509 Certificate`
   - `SAML assertion`
   - `Kerberos GSS v5 AP_REQ` (on the SecurityPEP node)
   - `LTPA v2 token` (on the SecurityPEP node)
   - `Universal WSSE token` (on the SecurityPEP node)

   On the security enabled input nodes, the default value is *Transport Default*. On the SecurityPEP node, the default value is *Current token*, which means that the token type that exists in the identity mapped or source field in the Properties folder is used.
2. In **Identity Token Location**, specify the location in the message where the identity is specified. This string is in the form of an ESQL field reference, XPath expression, or string literal, and must resolve to a token with the type `Username`, `Username and password`, `SAML assertion`, `Kerberos GSS v5 AP_REQ`, `LTPA v2 token`, or `X.509 Certificate`. If you use a string literal, it must be enclosed in single quotes and must not contain a period (.).
3. In **Identity Password Location**, enter the location in the message where the password is specified. This string is in the form of an ESQL field reference, XPath expression, or string literal, and must resolve to a string containing a password. If you use a string literal, it must be enclosed in single quotes and must not contain a period (.). This option can be set only if the **Identity Token Type** is set to `Username and password`.

4. In **Identity IssuedBy Location**, specify a string or path expression to show where (in the message) information about the issuer of the identity is held. This string is in the form of an ESQL field reference, XPath expression, or string literal, defining where the identity was defined. If you use a string literal, it must be enclosed in single quotes and must not contain a period (.).

   If you leave this property blank on the security enabled input nodes, the transport header value is used (if there is one). For example, for MQ the MQMD.PutApplName value is used. If you leave this property blank on the SecurityPEP node, the WS-Trust request is sent to the STS without the optional Issuer element in the WS-Trust message.
5. (Optional) Ensure that all input nodes share the same information by promoting the properties to the message flow.

What to do next

To enable the extraction of an identity in a security enabled input node or SecurityPEP node, select a security profile that has at least one security operation configured (authentication, mapping, or authorization) or propagation enabled:

1. In the Message Broker Toolkit, right-click the BAR file, then click **Open with** > **Broker Archive Editor**.
2. Click the **Manage and Configure** tab.
3. Click the flow or node on which you want to set the security profile. The properties that you can configure for the message flow or for the node are displayed in the **Properties** view.
4. In the **Security Profile Name** field, select a security profile.
5. Save the BAR file.

Related concepts:

"Identity" on page 390
In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

"Authentication and validation" on page 398
Authentication is the process of establishing the identity of a user or system and verifying that the identity is valid. Applying authentication to a SAML security token involves validating the assertions that it carries and confirming that it is being processed within its validity period.

"Identity mapping" on page 403
Identity mapping is the transformation of a security token from one format to another format, or the federation of an identity from one realm to an equivalent identity in another realm.

"Authorization" on page 401
Authorization is the process of verifying that an identity token has permission to access a message flow.

"Identity and security token propagation" on page 426
Identity and security token propagation enables the identity and security tokens (associated with each message) to be propagated throughout a message flow, and on to target applications through output or request nodes.

"Security profiles" on page 387
A security profile defines the security operations that are to be performed in a message flow at SecurityPEP nodes and security enabled input and output nodes.

"Policy sets" on page 774
Policy sets and bindings define and configure your WS-Security requirements, supported by WebSphere Message Broker, for the SOAPInput, SOAPReply,

SOAPRequest, SOAPAsyncRequest, and SOAPAsyncResponse nodes.

Related tasks:

"Configuring identity authentication and security token validation"
You can configure a message flow to perform identity authentication or security token validation using Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token sever (STS) such as Tivoli Federated Identity Manager (TFIM) Version 6.2. Support for TFIM V6.1 is also provided, for compatibility with previous versions of WebSphere Message Broker.

"Creating a security profile" on page 433
You can create a security profile for use with Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2. Support is also provided for TFIM V6.1, for compatibility with previous versions of WebSphere Message Broker. You can create the security profile by using either the **mqsicreateconfigurableservice** command or an editor in the WebSphere Message Broker Explorer.

"Configuring authorization" on page 470
Configure the broker to use an external security provider to authorize an identity in a message flow. You can use either Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS) such as Tivoli Federated Identity Manager (TFIM) V6.2. Support for TFIM V6.1 is also provided for compatibility with previous versions of WebSphere Message Broker.

"Configuring for identity propagation" on page 492
To enable a message flow to perform identity propagation, the input nodes must extract the identity and the output node must propagate it.

"Setting up message flow security" on page 431
Set up security on a message flow to control access based on the identity of a message passing through the message flow.

Related reference:

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

"HTTPInput node" on page 4474
Use the HTTPInput node to receive an HTTP message from an HTTP client for processing by a message flow.

"SOAPInput node" on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

Configuring identity authentication and security token validation

You can configure a message flow to perform identity authentication or security token validation using Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token sever (STS) such as Tivoli Federated Identity Manager (TFIM) Version 6.2. Support for TFIM V6.1 is also provided, for compatibility with previous versions of WebSphere Message Broker.

Before you begin

Before you start:

Check that an appropriate security profile exists, or create a new security profile. See "Creating a security profile" on page 433.

About this task

For information about configuring authentication with HTTP basic authentication, LDAP, a WS-Trust V1.3 STS (TFIM V6.2), or TFIM V6.1, see:

- "Configuring authentication with HTTP basic authentication"
- "Configuring authentication with LDAP" on page 453
- "Configuring authentication or security token validation with a WS-Trust v1.3 STS (TFIM V6.2)" on page 457
- "Configuring authentication with TFIM V6.1" on page 459

Related concepts:

"Identity" on page 390
In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

"Authentication and validation" on page 398
Authentication is the process of establishing the identity of a user or system and verifying that the identity is valid. Applying authentication to a SAML security token involves validating the assertions that it carries and confirming that it is being processed within its validity period.

"Security profiles" on page 387
A security profile defines the security operations that are to be performed in a message flow at SecurityPEP nodes and security enabled input and output nodes.

"Security exception processing" on page 429
A security exception is raised when a message flow security failure occurs during security processing in an input node or SecurityPEP node.

Related tasks:

"Configuring authentication with LDAP" on page 453
This topic describes how to configure a message flow to perform identity authentication using Lightweight Directory Access Protocol (LDAP).

"Configuring authentication with TFIM V6.1" on page 459
You can configure a message flow to perform identity authentication by using Tivoli Federated Identity Manager (TFIM) V6.1.

"Configuring authentication or security token validation with a WS-Trust v1.3 STS (TFIM V6.2)" on page 457
You can configure supported message flow input nodes or SecurityPEP nodes to perform identity authentication or security token validation using a WS-Trust v1.3 compliant security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2.

"Setting up message flow security" on page 431
Set up security on a message flow to control access based on the identity of a message passing through the message flow.

Configuring authentication with HTTP basic authentication:

Use a security profile to configure HTTP basic authentication in the HTTPRequest or SOAPInput nodes.

About this task

Basic authentication is a common extension in the HTTP protocol that allows a client to provide identity information to a remote web server in the form of a username and password sent in the HTTP header data. Security profiles in WebSphere Message Broker provide a way for message flow designers to provide these credentials without building the HTTP headers in a Compute node.

If identity propagation is enabled for the selected security profile, the HTTPRequest and SOAPRequest nodes automatically pick up username and password credentials, if present, from the Properties tree. See “Configuring for identity propagation” on page 492. The predefined security profile Default Propagation has this setting enabled.

To enable basic authentication, select an appropriate security profile for the output node or the message flow in the Broker Archive editor. The credentials are picked up from the following Properties tree locations if set:

```
Properties.IdentityMappedType
Properties.IdentityMappedToken
Properties.IdentityMappedPassword
```

If the mapped identity fields are not set, the credentials are picked up from the following Properties tree locations:

```
Properties.IdentitySourceType
Properties.IdentitySourceToken
Properties.IdentitySourcePassword
```

For basic authentication both a username and password are required, therefore the appropriate Type field must be set to the string `usernameAndPassword`. For example:

```
SET OutputRoot.Properties.IdentitySourceType='usernameAndPassword';
SET OutputRoot.Properties.IdentitySourceToken = 'myUser';
SET OutputRoot.Properties.IdentitySourcePassword = 'myPassw0rd';
```

These fields are interpreted by a subsequent HTTPRequest or SOAPRequest node and converted into a basic authentication HTTP header.

You can also propagate credentials from an input message by setting a security profile which includes propagation on an input node, and then using the input node properties `Identity token type`, `Identity Token location` and `Identity password location`. These three properties take an XPath expression that specifies the location in the input message to retrieve the appropriate token or password from. When configured correctly, these properties place the identity information in the `Properties.IdentitySourceType`, `Properties.IdentitySourceToken` and `Properties.IdentitySourcePassword` fields. HTTPRequest or SOAPRequest nodes then use these values directly, with an appropriate security policy.

You can override the configuration of the security profile by selecting the build option **Override configurable property values** in the Broker Archive editor.

Related concepts:

“Identity” on page 390
In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

“Authentication and validation” on page 398
Authentication is the process of establishing the identity of a user or system and verifying that the identity is valid. Applying authentication to a SAML security token involves validating the assertions that it carries and confirming that it is being processed within its validity period.

“Identity mapping” on page 403
Identity mapping is the transformation of a security token from one format to another format, or the federation of an identity from one realm to an equivalent identity in another realm.

"Identity and security token propagation" on page 426
Identity and security token propagation enables the identity and security tokens (associated with each message) to be propagated throughout a message flow, and on to target applications through output or request nodes.

"Security profiles" on page 387
A security profile defines the security operations that are to be performed in a message flow at SecurityPEP nodes and security enabled input and output nodes.

Related tasks:

"Configuring for identity propagation" on page 492
To enable a message flow to perform identity propagation, the input nodes must extract the identity and the output node must propagate it.

"Creating a security profile" on page 433
You can create a security profile for use with Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2. Support is also provided for TFIM V6.1, for compatibility with previous versions of WebSphere Message Broker. You can create the security profile by using either the **mqsicreateconfigurableservice** command or an editor in the WebSphere Message Broker Explorer.

Related reference:

"HTTPRequest node" on page 4488
Use the HTTPRequest node to interact with a web service.

"SOAPRequest node" on page 4828
Use the SOAPRequest node to send a SOAP request to the remote Web service.

"SOAPAsyncRequest node" on page 4750
Use the SOAPAsyncRequest node with the SOAPAsyncResponse node to construct a pair of message flows that call a Web service asynchronously.

"Broker Archive editor" on page 6794
Use the Broker Archive editor to create and manage broker archive (BAR) files.

Configuring authentication with LDAP:

This topic describes how to configure a message flow to perform identity authentication using Lightweight Directory Access Protocol (LDAP).

Before you begin

Before you start:

Before you can configure a message flow to perform identity authentication using LDAP, you need to check that an appropriate security profile exists, or create a new security profile. See "Creating a security profile for LDAP" on page 435.

About this task

To authenticate the identity of a user or system, the broker attempts to connect to the LDAP server using the username and password associated with the identity. To do this, the broker needs the following information:

- To resolve the username to an LDAP entry, the broker needs to know the base distinguished name (base DN) of the accepted login IDs. This is required to enable the broker to differentiate between different entries with the same name.
- If the identities do not all have a common base DN, but can be uniquely resolved from a subtree, the DN can be specified in the broker configuration. When a subtree search has been specified, the broker must first connect to the

LDAP server and search for the given username in order to obtain the full username distinguished name (DN) to be used for authentication. If your LDAP directory does not permit login of unrecognized IDs, and does not grant search access rights on the subtree, you must set up a separate authorized login ID that the broker can use for the search. Use the **mqsisetdbparms** command to specify a username and password. For example:

```
mqsisetdbparms -n ldap::LDAP -u username -p password
```

or

```
mqsisetdbparms -n ldap::<servername> -u username -p password
```

where <*servername*> is your base LDAP server name, for example, `ldap.mydomain.com`.

If you specify `ldap::LDAP`, it creates a default setting for the broker, which the broker attempts to use if you have not explicitly used the **mqsisetdbparms** command to create a login ID for a specific <*servername*>. All servers that do not have an explicit `ldap::servername` entry then start using the credentials in the `ldap::LDAP` entry. This means that any servers that were previously using anonymous bind by default will start using the details in `ldap::LDAP`.

The username that you specify in the **-u** parameter must be recognized by the LDAP server as a complete user name. In most cases this means that you need to specify the full DN of the user. Alternatively, by specifying a username to be anonymous, you can force the broker to bind anonymously to this LDAP server. This might be useful if you have specified a non-anonymous bind as your default (ldap::LDAP). For example:

```
mqsisetdbparms -n ldap::<servername> -u anonymous -p password
```

In this case, the value specified for *password* is ignored.

Steps for enabling LDAP authentication:

Procedure

To enable an existing message flow to perform identity authentication, use the Broker Archive editor to select a security profile that uses LDAP for authentication. You can set a security profile on a message flow or on individual input nodes. If no security profile is set for the input nodes, the setting is inherited from the setting on the message flow.

1. Switch to the Broker Application Development perspective.
2. In the Broker Development view, right-click the BAR file and then click **Open with** > **Broker Archive Editor**.
3. Click the **Manage and Configure** tab.
4. Click the flow or node on which you want to set the security profile. The properties that you can configure for the message flow or for the node are displayed in the **Properties** view.
5. In the **Security Profile Name** field, select a security profile that uses LDAP for authentication.
6. Save the BAR file.

What to do next

For a SOAPInput node to use the identity in the WS-Security header (rather than an underlying transport identity) an appropriate policy set and bindings must also be defined and specified. For more information, see “Policy sets” on page 774.

If the message identity does not contain enough information for authentication, the information must be taken from the message body. For example, if a password is required for authentication but the message came from WebSphere MQ with only a username, the password information must be taken from the message body. For more information, see "Configuring the extraction of an identity or security token" on page 447.

Related concepts:

"Identity" on page 390
In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

"Authentication and validation" on page 398
Authentication is the process of establishing the identity of a user or system and verifying that the identity is valid. Applying authentication to a SAML security token involves validating the assertions that it carries and confirming that it is being processed within its validity period.

"Identity mapping" on page 403
Identity mapping is the transformation of a security token from one format to another format, or the federation of an identity from one realm to an equivalent identity in another realm.

"Authorization" on page 401
Authorization is the process of verifying that an identity token has permission to access a message flow.

"Identity and security token propagation" on page 426
Identity and security token propagation enables the identity and security tokens (associated with each message) to be propagated throughout a message flow, and on to target applications through output or request nodes.

"Security profiles" on page 387
A security profile defines the security operations that are to be performed in a message flow at SecurityPEP nodes and security enabled input and output nodes.

"Security exception processing" on page 429
A security exception is raised when a message flow security failure occurs during security processing in an input node or SecurityPEP node.

Related tasks:

"Configuring the extraction of an identity or security token" on page 447
You can configure the SecurityPEP node or security enabled input nodes to extract the identity or security token from a message and store it in the properties tree identity fields, enabling it to be processed throughout the message flow and propagated at output or request nodes.

"Configuring identity mapping" on page 463
Configure a security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2, to map the incoming security token and, if required, to authenticate and authorize it.

"Creating a security profile" on page 433
You can create a security profile for use with Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2. Support is also provided for TFIM V6.1, for compatibility with previous versions of WebSphere Message Broker. You can create the security profile by using either the **mqsicreateconfigurableservice** command or an editor in the WebSphere Message Broker Explorer.

"Configuring authorization" on page 470
Configure the broker to use an external security provider to authorize an identity in a message flow. You can use either Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS) such as Tivoli Federated Identity Manager (TFIM) V6.2. Support for TFIM V6.1 is also provided for compatibility with previous versions of WebSphere Message Broker.

"Configuring identity authentication and security token validation" on page 450
You can configure a message flow to perform identity authentication or security token validation using Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token sever (STS) such as Tivoli Federated Identity Manager (TFIM) Version 6.2. Support for TFIM V6.1 is also provided, for compatibility with previous versions of WebSphere Message Broker.

"Configuring for identity propagation" on page 492
To enable a message flow to perform identity propagation, the input nodes must extract the identity and the output node must propagate it.

"Setting up message flow security" on page 431
Set up security on a message flow to control access based on the identity of a message passing through the message flow.

Related reference:

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

"HTTPInput node" on page 4474
Use the HTTPInput node to receive an HTTP message from an HTTP client for processing by a message flow.

"SOAPInput node" on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

"SOAPAsyncRequest node" on page 4750
Use the SOAPAsyncRequest node with the SOAPAsyncResponse node to construct a pair of message flows that call a Web service asynchronously.

"HTTPRequest node" on page 4488
Use the HTTPRequest node to interact with a web service.

"MQOutput node" on page 4612
Use the MQOutput node to send messages to clients that connect to the broker using the WebSphere MQ Enterprise Transport and that use the MQI and AMI application programming interfaces.

Configuring authentication or security token validation with a WS-Trust v1.3 STS (TFIM V6.2):

You can configure supported message flow input nodes or SecurityPEP nodes to perform identity authentication or security token validation using a WS-Trust v1.3 compliant security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2.

Before you begin

Before you start:

Before you can configure identity authentication or token validation, you need to check that an appropriate security profile exists, or create a new security profile. See "Creating a security profile for WS-Trust V1.3 (TFIM V6.2)" on page 440.

About this task

When the security profile is configured to use WS-Trust V1.3 STS for authentication, the broker security manager issues trust requests and processes trust responses according to the WS-Trust V1.3 standard.

When you use a WS-Trust v1.3 STS for authentication, a request is made to the trust service with the following parameters, which control the STS processing. If you are using TFIM V6.2, the following parameters are used in the selection of the TFIM module chain:

- RequestType
- Issuer
- AppliesTo

For more information about these parameters, see:"Authentication, mapping, and authorization with TFIM V6.2 and TAM" on page 419.

The WS-Trust v1.3 specification, published by OASIS, is available at:
`http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.html`

.

Steps for enabling WS-Trust v1.3 authentication:

Procedure

To enable an existing message flow to perform authentication or token validation, use the Broker Archive editor to select a security profile that uses a WS-Trust v1.3 STS for authentication and to associate it with the node or message flow. If a security profile is specified on either a message flow or a node, the profile must be available when the message flow is deployed; otherwise, a deployment error occurs.

1. In the Message Broker Toolkit, right-click the BAR file, then click **Open with** > **Broker Archive Editor**.
2. Click the **Manage and Configure** tab.
3. Click the flow or node on which you want to set the security profile. The properties that you can configure for the message flow or for the node are displayed in the **Properties** view.

4. In the **Security Profile Name** field, select a security profile that configures WS-Trust v1.3 STS for authentication.
5. Save the BAR file.

What to do next

For a SOAPInput node to use the identity in the WS-Security header (rather than an underlying transport identity) an appropriate policy set and bindings must also be defined and specified. For more information, see "Policy sets" on page 774.

If the message identity (or security token) does not contain enough information for authentication, the information must be taken from the message body. For example, if a password is required for authentication but the message came from WebSphere MQ with only a username, the password information must be taken from the message body. For more information, see "Configuring the extraction of an identity or security token" on page 447.

Related concepts:

"Identity" on page 390
In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

"Authentication and validation" on page 398
Authentication is the process of establishing the identity of a user or system and verifying that the identity is valid. Applying authentication to a SAML security token involves validating the assertions that it carries and confirming that it is being processed within its validity period.

"Security profiles" on page 387
A security profile defines the security operations that are to be performed in a message flow at SecurityPEP nodes and security enabled input and output nodes.

"Security exception processing" on page 429
A security exception is raised when a message flow security failure occurs during security processing in an input node or SecurityPEP node.

Related tasks:

"Configuring the extraction of an identity or security token" on page 447
You can configure the SecurityPEP node or security enabled input nodes to extract the identity or security token from a message and store it in the properties tree identity fields, enabling it to be processed throughout the message flow and propagated at output or request nodes.

"Creating a security profile" on page 433
You can create a security profile for use with Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2. Support is also provided for TFIM V6.1, for compatibility with previous versions of WebSphere Message Broker. You can create the security profile by using either the **mqsicreateconfigurableservice** command or an editor in the WebSphere Message Broker Explorer.

"Configuring authorization" on page 470
Configure the broker to use an external security provider to authorize an identity in a message flow. You can use either Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS) such as Tivoli Federated Identity Manager (TFIM) V6.2. Support for TFIM V6.1 is also provided for compatibility with previous versions of WebSphere Message Broker.

"Configuring identity authentication and security token validation" on page 450
You can configure a message flow to perform identity authentication or security token validation using Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token sever (STS) such as Tivoli Federated Identity Manager (TFIM) Version 6.2. Support for TFIM V6.1 is also provided, for compatibility with previous versions of WebSphere Message Broker.

"Setting up message flow security" on page 431
Set up security on a message flow to control access based on the identity of a message passing through the message flow.

Related reference:

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

"HTTPInput node" on page 4474
Use the HTTPInput node to receive an HTTP message from an HTTP client for processing by a message flow.

"SOAPInput node" on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

"SOAPAsyncRequest node" on page 4750
Use the SOAPAsyncRequest node with the SOAPAsyncResponse node to construct a pair of message flows that call a Web service asynchronously.

"HTTPRequest node" on page 4488
Use the HTTPRequest node to interact with a web service.

"MQOutput node" on page 4612
Use the MQOutput node to send messages to clients that connect to the broker using the WebSphere MQ Enterprise Transport and that use the MQI and AMI application programming interfaces.

"SecurityPEP node" on page 4729
Use the SecurityPEP node in a message flow to invoke the message flow security manager at any point in the message flow.

Configuring authentication with TFIM V6.1:

You can configure a message flow to perform identity authentication by using Tivoli Federated Identity Manager (TFIM) V6.1.

Before you begin

Before you start:

Before you can configure a message flow to perform identity authentication, you need to check that an appropriate security profile exists, or create a new security profile. See "Creating a security profile for TFIM V6.1" on page 444.

About this task

Note: Support for TFIM V6.1 is included for compatibility with previous versions of WebSphere Message Broker. If possible, upgrade to TFIM V6.2 and follow the instructions in "Configuring authentication or security token validation with a WS-Trust v1.3 STS (TFIM V6.2)" on page 457.

When you use TFIM V6.1 for authentication, a request is made to the TFIM trust service with the following three parameters, which select the module chain:

- Issuer = Properties.IdentitySourceIssuedBy
- Applies To = The Fully Qualified Name of the Flow: <*Brokername*>.<*Execution Group Name*>.<*Message Flow Name*>
- Token = Properties.IdentitySourceToken

For more information about these parameters, see "Authentication, mapping, and authorization with TFIM V6.1 and TAM" on page 416.

For further information about how to configure TFIM, see the IBM Tivoli Federated Identity Manager Information Center.

Steps for enabling TFIM authentication:

Procedure

To enable an existing message flow to perform identity authentication, use the Broker Archive editor to select a security profile that uses TFIM for authentication. You can set a security profile on a message flow or on individual input nodes. If no security profile is set for the input nodes, the setting is inherited from the setting on the message flow.

1. Switch to the Broker Application Development perspective.
2. In the Broker Development view, right-click the BAR file, then click **Open with** > **Broker Archive Editor**.
3. Click the **Manage and Configure** tab.
4. Click the flow or node on which you want to set the security profile. The properties that you can configure for the message flow or for the node are displayed in the **Properties** view.
5. In the **Security Profile Name** field, select a security profile that uses TFIM for authentication.
6. Save the BAR file.

What to do next

For a SOAPInput node to use the identity in the WS-Security header (rather than an underlying transport identity) an appropriate policy set and bindings must also be defined and specified. For more information, see "Policy sets" on page 774.

If the message identity does not contain enough information for authentication, the information must be taken from the message body. For example, if a password is required for authentication but the message came from WebSphere MQ with only a username, the password information must be taken from the message body. For more information, see "Configuring the extraction of an identity or security token" on page 447.

Related concepts:

"Identity" on page 390
In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

"Authentication and validation" on page 398
Authentication is the process of establishing the identity of a user or system and verifying that the identity is valid. Applying authentication to a SAML security token involves validating the assertions that it carries and confirming that it is being processed within its validity period.

"Identity mapping" on page 403
Identity mapping is the transformation of a security token from one format to another format, or the federation of an identity from one realm to an equivalent identity in another realm.

"Authorization" on page 401
Authorization is the process of verifying that an identity token has permission to access a message flow.

"Identity and security token propagation" on page 426
Identity and security token propagation enables the identity and security tokens (associated with each message) to be propagated throughout a message flow, and on to target applications through output or request nodes.

"Security profiles" on page 387
A security profile defines the security operations that are to be performed in a message flow at SecurityPEP nodes and security enabled input and output nodes.

"Security exception processing" on page 429
A security exception is raised when a message flow security failure occurs during security processing in an input node or SecurityPEP node.

Related tasks:

"Configuring the extraction of an identity or security token" on page 447
You can configure the SecurityPEP node or security enabled input nodes to extract the identity or security token from a message and store it in the properties tree identity fields, enabling it to be processed throughout the message flow and propagated at output or request nodes.

"Configuring identity mapping" on page 463
Configure a security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2, to map the incoming security token and, if required, to authenticate and authorize it.

"Creating a security profile" on page 433
You can create a security profile for use with Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2. Support is also provided for TFIM V6.1, for compatibility with previous versions of WebSphere Message Broker. You can create the security profile by using either the **mqsicreateconfigurableservice** command or an editor in the WebSphere Message Broker Explorer.

"Configuring authorization" on page 470
Configure the broker to use an external security provider to authorize an identity in a message flow. You can use either Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS) such as Tivoli Federated Identity Manager (TFIM) V6.2. Support for TFIM V6.1 is also provided for compatibility with previous versions of WebSphere Message Broker.

"Configuring identity authentication and security token validation" on page 450
You can configure a message flow to perform identity authentication or security token validation using Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token sever (STS) such as Tivoli Federated Identity Manager (TFIM) Version 6.2. Support for TFIM V6.1 is also provided, for compatibility with previous versions of WebSphere Message Broker.

"Configuring for identity propagation" on page 492
To enable a message flow to perform identity propagation, the input nodes must extract the identity and the output node must propagate it.

"Setting up message flow security" on page 431
Set up security on a message flow to control access based on the identity of a message passing through the message flow.

Related reference:

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

"HTTPInput node" on page 4474
Use the HTTPInput node to receive an HTTP message from an HTTP client for processing by a message flow.

"SOAPInput node" on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

"SOAPAsyncRequest node" on page 4750
Use the SOAPAsyncRequest node with the SOAPAsyncResponse node to construct a pair of message flows that call a Web service asynchronously.

"HTTPRequest node" on page 4488
Use the HTTPRequest node to interact with a web service.

"MQOutput node" on page 4612
Use the MQOutput node to send messages to clients that connect to the broker using the WebSphere MQ Enterprise Transport and that use the MQI and AMI application programming interfaces.

Configuring identity mapping

Configure a security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2, to map the incoming security token and, if required, to authenticate and authorize it.

Before you begin

Before you start:

Before you can configure a message flow to perform identity mapping, you need to check that an appropriate security profile exists, or create a new security profile. For information about security profiles, see "Creating a security profile" on page 433.

About this task

WebSphere Message Broker provides support for identity mapping (also known as identity federation) and token issuance and exchange. Identity mapping is the process of mapping an identity in one realm to another identity in a different realm. For example, you might map *User001* from the eSellers realm to *eSellerUser01* in the eShipping realm. Token issuance and exchange involves the mapping of a token of one type to a token of a different type. For example, an incoming Username and Password token from a client over MQ might be mapped into an equivalent SAML assertion, to be propagated to a Web Services call. Alternatively, you might exchange a SAML 1.1 assertion from a client application for an equivalent SAML 2.0 assertion for an updated backend server.

For information about configuring identity mapping with either a WS-Trust V1.3 STS (for example, TFIM V6.2) or with TFIM V6.1, see:

- "Configuring identity mapping with a WS-Trust V1.3 STS (TFIM V6.2)" on page 465
- "Configuring identity mapping with TFIM V6.1" on page 467

Related concepts:

"Identity" on page 390
In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

"Authentication and validation" on page 398
Authentication is the process of establishing the identity of a user or system and verifying that the identity is valid. Applying authentication to a SAML security token involves validating the assertions that it carries and confirming that it is being processed within its validity period.

"Identity mapping" on page 403
Identity mapping is the transformation of a security token from one format to another format, or the federation of an identity from one realm to an equivalent identity in another realm.

"Authorization" on page 401
Authorization is the process of verifying that an identity token has permission to access a message flow.

"Identity and security token propagation" on page 426
Identity and security token propagation enables the identity and security tokens (associated with each message) to be propagated throughout a message flow, and on to target applications through output or request nodes.

"Security profiles" on page 387
A security profile defines the security operations that are to be performed in a

message flow at SecurityPEP nodes and security enabled input and output nodes.

"Security exception processing" on page 429
A security exception is raised when a message flow security failure occurs during security processing in an input node or SecurityPEP node.

Related tasks:

"Configuring the extraction of an identity or security token" on page 447
You can configure the SecurityPEP node or security enabled input nodes to extract the identity or security token from a message and store it in the properties tree identity fields, enabling it to be processed throughout the message flow and propagated at output or request nodes.

"Configuring identity authentication and security token validation" on page 450
You can configure a message flow to perform identity authentication or security token validation using Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token sever (STS) such as Tivoli Federated Identity Manager (TFIM) Version 6.2. Support for TFIM V6.1 is also provided, for compatibility with previous versions of WebSphere Message Broker.

"Creating a security profile" on page 433
You can create a security profile for use with Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2. Support is also provided for TFIM V6.1, for compatibility with previous versions of WebSphere Message Broker. You can create the security profile by using either the **mqsicreateconfigurableservice** command or an editor in the WebSphere Message Broker Explorer.

"Configuring authorization" on page 470
Configure the broker to use an external security provider to authorize an identity in a message flow. You can use either Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS) such as Tivoli Federated Identity Manager (TFIM) V6.2. Support for TFIM V6.1 is also provided for compatibility with previous versions of WebSphere Message Broker.

"Configuring for identity propagation" on page 492
To enable a message flow to perform identity propagation, the input nodes must extract the identity and the output node must propagate it.

"Setting up message flow security" on page 431
Set up security on a message flow to control access based on the identity of a message passing through the message flow.

Related reference:

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

"HTTPInput node" on page 4474
Use the HTTPInput node to receive an HTTP message from an HTTP client for processing by a message flow.

"SOAPInput node" on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

"SOAPAsyncRequest node" on page 4750
Use the SOAPAsyncRequest node with the SOAPAsyncResponse node to construct a pair of message flows that call a Web service asynchronously.

"HTTPRequest node" on page 4488
Use the HTTPRequest node to interact with a web service.

"MQOutput node" on page 4612
Use the MQOutput node to send messages to clients that connect to the broker using the WebSphere MQ Enterprise Transport and that use the MQI and AMI application programming interfaces.

Configuring identity mapping with a WS-Trust V1.3 STS (TFIM V6.2):

Configure a WS-Trust V1.3 compliant security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2, to map the incoming security token and, if required, to authenticate and authorize it.

Before you begin

Before you start:

Before you can configure a message flow to perform identity mapping, you need to check that an appropriate security profile exists, or create a new security profile. For information about security profiles, see "Creating a security profile" on page 433.

About this task

To configure TFIM V6.2 to map an incoming security token, you must create a custom module chain in TFIM, which performs the security operations. The TFIM configuration controls the token type that is returned from the mapping.

When you use a WS-Trust V1.3 STS for identity mapping, a request is made to the security token server with the following parameters, which control the STS processing:

- RequestType
- Issuer
- AppliesTo

If you are using TFIM V6.2, these parameters are used in the selection of the module chain.

The security manager invokes the WS-Trust v1.3 provider only once, even if it is set for additional security operations (such as authentication or authorization). As a result, when you are using TFIM V6.2, you must configure a single module chain to perform all the required authentication, mapping, and authorization operations.

When the security profile includes a mapping operation, the STS (for example, TFIM V6.2) must return a security token in its response. The STS can return the original unmodified token if no token exchange is required.

For more information about these parameters, see:"Authentication, mapping, and authorization with TFIM V6.2 and TAM" on page 419.

The WS-Trust v1.3 specification, published by OASIS, is available at:
`http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.html`

For information on how to configure TFIM, see the IBM Tivoli Federated Identity Manager Information Center.

Follow these steps to enable an existing message flow to perform identity mapping.

Procedure

Using the Broker Archive editor, select a security profile that has mapping enabled. You can set a security profile on a message flow or on individual input nodes. If no security profile is set for the input nodes, the setting is inherited from the setting on the message flow.

1. In the Message Broker Toolkit, right-click the BAR file, then click **Open with** > **Broker Archive Editor**.
2. Click the **Manage and Configure** tab.
3. Click the message flow or node on which you want to set the security profile. The properties that you can configure for the message flow or for the node are displayed in the **Properties** view.
4. In the **Security Profile Name** field, enter the name of a security profile that has mapping enabled.
5. Save the BAR file.

Related concepts:

"Identity" on page 390
In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

"Authentication and validation" on page 398
Authentication is the process of establishing the identity of a user or system and verifying that the identity is valid. Applying authentication to a SAML security token involves validating the assertions that it carries and confirming that it is being processed within its validity period.

"Identity mapping" on page 403
Identity mapping is the transformation of a security token from one format to another format, or the federation of an identity from one realm to an equivalent identity in another realm.

"Authorization" on page 401
Authorization is the process of verifying that an identity token has permission to access a message flow.

"Identity and security token propagation" on page 426
Identity and security token propagation enables the identity and security tokens (associated with each message) to be propagated throughout a message flow, and on to target applications through output or request nodes.

"Security profiles" on page 387
A security profile defines the security operations that are to be performed in a message flow at SecurityPEP nodes and security enabled input and output nodes.

"Security exception processing" on page 429
A security exception is raised when a message flow security failure occurs during security processing in an input node or SecurityPEP node.

Related tasks:

"Configuring the extraction of an identity or security token" on page 447
You can configure the SecurityPEP node or security enabled input nodes to extract the identity or security token from a message and store it in the properties tree identity fields, enabling it to be processed throughout the message flow and propagated at output or request nodes.

"Configuring identity authentication and security token validation" on page 450
You can configure a message flow to perform identity authentication or security token validation using Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token sever (STS) such as Tivoli Federated Identity Manager (TFIM) Version 6.2. Support for TFIM V6.1 is also provided, for

compatibility with previous versions of WebSphere Message Broker.

"Creating a security profile" on page 433
You can create a security profile for use with Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2. Support is also provided for TFIM V6.1, for compatibility with previous versions of WebSphere Message Broker. You can create the security profile by using either the **mqsicreateconfigurableservice** command or an editor in the WebSphere Message Broker Explorer.

"Configuring authorization" on page 470
Configure the broker to use an external security provider to authorize an identity in a message flow. You can use either Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS) such as Tivoli Federated Identity Manager (TFIM) V6.2. Support for TFIM V6.1 is also provided for compatibility with previous versions of WebSphere Message Broker.

"Configuring for identity propagation" on page 492
To enable a message flow to perform identity propagation, the input nodes must extract the identity and the output node must propagate it.

"Setting up message flow security" on page 431
Set up security on a message flow to control access based on the identity of a message passing through the message flow.

Related reference:

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

"HTTPInput node" on page 4474
Use the HTTPInput node to receive an HTTP message from an HTTP client for processing by a message flow.

"SOAPInput node" on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

"SOAPAsyncRequest node" on page 4750
Use the SOAPAsyncRequest node with the SOAPAsyncResponse node to construct a pair of message flows that call a Web service asynchronously.

"HTTPRequest node" on page 4488
Use the HTTPRequest node to interact with a web service.

"MQOutput node" on page 4612
Use the MQOutput node to send messages to clients that connect to the broker using the WebSphere MQ Enterprise Transport and that use the MQI and AMI application programming interfaces.

Configuring identity mapping with TFIM V6.1:

Configure Tivoli Federated Identity Manager (TFIM) V6.1 to map the incoming security token and, if required, to authenticate and authorize it.

Before you begin

Before you start:

Before you can configure a message flow to perform identity mapping, you need to check that an appropriate security profile exists, or create a new security profile.

For information about security profiles, see “Creating a security profile” on page 433.

About this task

Note: Support for TFIM V6.1 is included for compatibility with previous versions of WebSphere Message Broker. If possible, upgrade to TFIM V6.2 and follow the instructions in “Configuring identity mapping with a WS-Trust V1.3 STS (TFIM V6.2)” on page 465.

To configure TFIM V6.1 to map the incoming security token, you need to create a custom module chain in TFIM, which performs the security operations. The TFIM configuration controls the token type that is returned from the mapping.

When you use TFIM for mapping, a request is made to the TFIM trust service with the following three parameters, which select the module chain:

- Issuer = Properties.IdentitySourceIssuedBy
- AppliesTo = The fully qualified name of the flow: *Brokername.Execution Group Name.Message Flow Name*
- Token = Properties.IdentitySourceToken

The security manager invokes the security provider only once, even if it is set for additional security operations (such as authentication or authorization). As a result, when you are using TFIM V6.1, you must configure a single module chain to perform all the required authentication, mapping, and authorization operations.

For information on how to configure TFIM, see the IBM Tivoli Federated Identity Manager Information Center.

Follow these steps to enable an existing message flow to perform identity mapping.

Procedure

Using the Broker Archive editor, select a security profile that has mapping enabled. You can set a security profile on a message flow or on individual input nodes. If no security profile is set for the input nodes, the setting is inherited from the setting on the message flow.

1. In the Message Broker Toolkit, right-click the BAR file, then click **Open with** > **Broker Archive Editor**.
2. Click the **Manage and Configure** tab.
3. Click the flow or node on which you want to set the security profile. The properties that you can configure for the message flow or for the node are displayed in the **Properties** view.
4. In the **Security Profile Name** field, enter the name of a security profile that has mapping enabled.
5. Save the BAR file.

Related concepts:

“Identity” on page 390
In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

"Authentication and validation" on page 398
Authentication is the process of establishing the identity of a user or system and verifying that the identity is valid. Applying authentication to a SAML security token involves validating the assertions that it carries and confirming that it is being processed within its validity period.

"Identity mapping" on page 403
Identity mapping is the transformation of a security token from one format to another format, or the federation of an identity from one realm to an equivalent identity in another realm.

"Authorization" on page 401
Authorization is the process of verifying that an identity token has permission to access a message flow.

"Identity and security token propagation" on page 426
Identity and security token propagation enables the identity and security tokens (associated with each message) to be propagated throughout a message flow, and on to target applications through output or request nodes.

"Security profiles" on page 387
A security profile defines the security operations that are to be performed in a message flow at SecurityPEP nodes and security enabled input and output nodes.

"Security exception processing" on page 429
A security exception is raised when a message flow security failure occurs during security processing in an input node or SecurityPEP node.

Related tasks:

"Configuring the extraction of an identity or security token" on page 447
You can configure the SecurityPEP node or security enabled input nodes to extract the identity or security token from a message and store it in the properties tree identity fields, enabling it to be processed throughout the message flow and propagated at output or request nodes.

"Configuring identity authentication and security token validation" on page 450
You can configure a message flow to perform identity authentication or security token validation using Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token sever (STS) such as Tivoli Federated Identity Manager (TFIM) Version 6.2. Support for TFIM V6.1 is also provided, for compatibility with previous versions of WebSphere Message Broker.

"Creating a security profile" on page 433
You can create a security profile for use with Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2. Support is also provided for TFIM V6.1, for compatibility with previous versions of WebSphere Message Broker. You can create the security profile by using either the **mqsicreateconfigurableservice** command or an editor in the WebSphere Message Broker Explorer.

"Creating a security profile for LDAP" on page 435
Create a security profile for use with Lightweight Directory Access Protocol (LDAP) or Secure LDAP (LDAPS), by using either the **mqsicreateconfigurableservice** command or an editor in the WebSphere Message Broker Explorer.

"Creating a security profile for TFIM V6.1" on page 444
You can create a security profile for Tivoli Federated Identity Manager (TFIM) V6.1 for any combination of the following functions: authentication, authorization, and mapping. You can use either the **mqsicreateconfigurableservice** command or an editor in the WebSphere Message Broker Explorer to create the security profile.

"Configuring authorization"
Configure the broker to use an external security provider to authorize an identity in a message flow. You can use either Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS) such as Tivoli Federated Identity Manager (TFIM) V6.2. Support for TFIM V6.1 is also provided for compatibility with previous versions of WebSphere Message Broker.

"Configuring for identity propagation" on page 492
To enable a message flow to perform identity propagation, the input nodes must extract the identity and the output node must propagate it.

"Setting up message flow security" on page 431
Set up security on a message flow to control access based on the identity of a message passing through the message flow.

Related reference:

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

"HTTPInput node" on page 4474
Use the HTTPInput node to receive an HTTP message from an HTTP client for processing by a message flow.

"SOAPInput node" on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

"SOAPAsyncRequest node" on page 4750
Use the SOAPAsyncRequest node with the SOAPAsyncResponse node to construct a pair of message flows that call a Web service asynchronously.

"HTTPRequest node" on page 4488
Use the HTTPRequest node to interact with a web service.

"MQOutput node" on page 4612
Use the MQOutput node to send messages to clients that connect to the broker using the WebSphere MQ Enterprise Transport and that use the MQI and AMI application programming interfaces.

Configuring authorization

Configure the broker to use an external security provider to authorize an identity in a message flow. You can use either Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS) such as Tivoli Federated Identity Manager (TFIM) V6.2. Support for TFIM V6.1 is also provided for compatibility with previous versions of WebSphere Message Broker.

Before you begin

Before you start:

Check that an appropriate security profile exists, or create a new security profile. See "Creating a security profile" on page 433.

About this task

For information about configuring authorization for LDAP, WS-Trust V1.3 (TFIM V6.2), or TFIM V6.1, see:

- "Configuring authorization with LDAP" on page 472
- "Configuring authorization with a WS-Trust v1.3 STS (TFIM V6.2)" on page 475
- "Configuring authorization with TFIM V6.1" on page 483

What to do next

Related concepts:

"Identity" on page 390
In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

"Authorization" on page 401
Authorization is the process of verifying that an identity token has permission to access a message flow.

"Security profiles" on page 387
A security profile defines the security operations that are to be performed in a message flow at SecurityPEP nodes and security enabled input and output nodes.

Related tasks:

"Configuring identity authentication and security token validation" on page 450
You can configure a message flow to perform identity authentication or security token validation using Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token sever (STS) such as Tivoli Federated Identity Manager (TFIM) Version 6.2. Support for TFIM V6.1 is also provided, for compatibility with previous versions of WebSphere Message Broker.

"Configuring authorization with LDAP" on page 472
This topic describes how to configure a message flow to perform authorization on an identity using Lightweight Directory Access Protocol (LDAP).

"Configuring authorization with a WS-Trust v1.3 STS (TFIM V6.2)" on page 475
You can configure supported message flow input nodes or SecurityPEP nodes to perform authorization of an identity or security token by using a WS-Trust v1.3 compliant security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2.

"Configuring authorization with TFIM V6.1" on page 483
You can configure a message flow to perform authorization on an identity by using Tivoli Federated Identity Manager (TFIM) V6.1.

"Configuring identity mapping" on page 463
Configure a security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2, to map the incoming security token and, if required, to authenticate and authorize it.

"Creating a security profile" on page 433
You can create a security profile for use with Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2. Support is also provided for TFIM V6.1, for compatibility with previous versions of WebSphere Message Broker. You can create the security profile by using either the **mqsicreateconfigurableservice** command or an editor in the WebSphere Message Broker Explorer.

"Configuring for identity propagation" on page 492
To enable a message flow to perform identity propagation, the input nodes must extract the identity and the output node must propagate it.

"Setting up message flow security" on page 431
Set up security on a message flow to control access based on the identity of a message passing through the message flow.

Configuring authorization with LDAP:

This topic describes how to configure a message flow to perform authorization on an identity using Lightweight Directory Access Protocol (LDAP).

Before you begin

Before you start:

Before you can configure a message flow to perform authorization, you need to check that an appropriate security profile exists, or create a new security profile. See "Creating a security profile for LDAP" on page 435.

About this task

When LDAP is used for authorization, the broker needs to determine whether the incoming username is a member of the given group. To do this, the broker requires the following information:

- To resolve the username to an LDAP entry, the broker needs to know the base distinguished name (Base DN) of the accepted login IDs. This is required to enable the broker to differentiate between different entries with the same name.
- To get an entry list from a group name, the group name must be the distinguished name of the group, not just a common name. An LDAP search is made for the group, and the username is checked by finding an entry matching the distinguished name of the user.
- If your LDAP directory does not permit login by unrecognized IDs, and does not grant search access rights on the subtree, you must set up a separate authorized login ID that the broker can use for the search. Use the **mqsisetdbparms** command to specify a username and password:

  ```
  mqsisetdbparms -n ldap::LDAP -u username -p password
  ```

  or

  ```
  mqsisetdbparms -n ldap::<servername> -u username -p password
  ```

  where <*servername*> is your base LDAP server name. For example: `ldap.mydomain.com`.

  If you specify `ldap::LDAP`, it creates a default setting for the broker, which the broker attempts to use if you have not explicitly used the **mqsisetdbparms** command to create a login ID for a specific <*servername*>. All servers that do not have an explicit `ldap::servername` entry then start using the credentials in the

`ldap::LDAP` entry. This means that any servers that were previously using anonymous bind by default will start using the details in `ldap::LDAP`.

The username that you specify in the **-u** parameter must be recognized by the LDAP server as a complete user name. In most cases this means that you need to specify the full DN of the user. Alternatively, by specifying a username to be anonymous, you can force the broker to bind anonymously to this LDAP server. This might be useful if you have specified a non-anonymous bind as your default (ldap::LDAP). For example:

```
mqsisetdbparms -n ldap::<servername> -u anonymous -p password
```

In this case, the value specified for *password* is ignored.

Steps for enabling LDAP authorization:

Procedure

To enable an existing message flow to perform authorization using LDAP, use the Broker Archive editor to select a security profile that has authorization enabled. You can set a security profile on a message flow or on individual input nodes. If no security profile is set for the input nodes, the setting is inherited from the setting on the message flow.

1. Switch to the Broker Application Development perspective.
2. In the Broker Development view, right-click the BAR file and then click **Open with** > **Broker Archive Editor**.
3. Click the **Manage and Configure** tab.
4. Click the flow or node on which you want to set the security profile. The properties that you can configure for the message flow or for the node are displayed in the **Properties** view.
5. In the **Security Profile Name** field, select a security profile that uses LDAP for authorization.
6. Save the BAR file.

What to do next

For a SOAPInput node to use the identity in the WS-Security header (rather than an underlying transport identity) an appropriate policy set and bindings must also be defined and specified. For more information, see “Policy sets” on page 774.

Related concepts:

“Identity” on page 390
In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

“Authentication and validation” on page 398
Authentication is the process of establishing the identity of a user or system and verifying that the identity is valid. Applying authentication to a SAML security token involves validating the assertions that it carries and confirming that it is being processed within its validity period.

“Identity mapping” on page 403
Identity mapping is the transformation of a security token from one format to another format, or the federation of an identity from one realm to an equivalent identity in another realm.

“Authorization” on page 401
Authorization is the process of verifying that an identity token has permission to access a message flow.

"Identity and security token propagation" on page 426
Identity and security token propagation enables the identity and security tokens (associated with each message) to be propagated throughout a message flow, and on to target applications through output or request nodes.

"Security profiles" on page 387
A security profile defines the security operations that are to be performed in a message flow at SecurityPEP nodes and security enabled input and output nodes.

"Security exception processing" on page 429
A security exception is raised when a message flow security failure occurs during security processing in an input node or SecurityPEP node.

Related tasks:

"Configuring the extraction of an identity or security token" on page 447
You can configure the SecurityPEP node or security enabled input nodes to extract the identity or security token from a message and store it in the properties tree identity fields, enabling it to be processed throughout the message flow and propagated at output or request nodes.

"Configuring identity authentication and security token validation" on page 450
You can configure a message flow to perform identity authentication or security token validation using Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token sever (STS) such as Tivoli Federated Identity Manager (TFIM) Version 6.2. Support for TFIM V6.1 is also provided, for compatibility with previous versions of WebSphere Message Broker.

"Configuring authorization" on page 470
Configure the broker to use an external security provider to authorize an identity in a message flow. You can use either Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS) such as Tivoli Federated Identity Manager (TFIM) V6.2. Support for TFIM V6.1 is also provided for compatibility with previous versions of WebSphere Message Broker.

"Configuring identity mapping" on page 463
Configure a security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2, to map the incoming security token and, if required, to authenticate and authorize it.

"Creating a security profile" on page 433
You can create a security profile for use with Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2. Support is also provided for TFIM V6.1, for compatibility with previous versions of WebSphere Message Broker. You can create the security profile by using either the **mqsicreateconfigurableservice** command or an editor in the WebSphere Message Broker Explorer.

"Creating a security profile for LDAP" on page 435
Create a security profile for use with Lightweight Directory Access Protocol (LDAP) or Secure LDAP (LDAPS), by using either the **mqsicreateconfigurableservice** command or an editor in the WebSphere Message Broker Explorer.

"Configuring for identity propagation" on page 492
To enable a message flow to perform identity propagation, the input nodes must extract the identity and the output node must propagate it.

"Setting up message flow security" on page 431
Set up security on a message flow to control access based on the identity of a message passing through the message flow.

Related reference:

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a

broker external resource.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

"HTTPInput node" on page 4474
Use the HTTPInput node to receive an HTTP message from an HTTP client for processing by a message flow.

"SOAPInput node" on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

"SOAPAsyncRequest node" on page 4750
Use the SOAPAsyncRequest node with the SOAPAsyncResponse node to construct a pair of message flows that call a Web service asynchronously.

"HTTPRequest node" on page 4488
Use the HTTPRequest node to interact with a web service.

"MQOutput node" on page 4612
Use the MQOutput node to send messages to clients that connect to the broker using the WebSphere MQ Enterprise Transport and that use the MQI and AMI application programming interfaces.

Configuring authorization with a WS-Trust v1.3 STS (TFIM V6.2):

You can configure supported message flow input nodes or SecurityPEP nodes to perform authorization of an identity or security token by using a WS-Trust v1.3 compliant security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2.

Before you begin

Before you start:

Before you configure a message flow to perform authorization with a WS-Trust v1.3 STS:

- Check that an appropriate security profile exists, or create a new security profile. See "Creating a security profile for WS-Trust V1.3 (TFIM V6.2)" on page 440.

About this task

The message flow security manager issues an authorization request to the WS-Trust service with the following parameters, which select the TFIM module chain to be used:

- RequestType

- Issuer
- AppliesTo

For more information about these parameters, see:"Authentication, mapping, and authorization with TFIM V6.2 and TAM" on page 419.

In addition to configuring Message Broker to perform authorization with a WS-Trust compliant STS, you must configure TAM. For information about how to do this, see the following topics:

- "Creating a module chain in TFIM V6.2" on page 478
- "Configuring TAM for authorization using TFIM V6.2" on page 480

Steps for enabling authorization using a WS-Trust v1.3 STS provider:

Procedure

To enable an existing message flow to enforce authorization using a WS-Trust v1.3 STS provider, use the Broker Archive editor to select a security profile that has authorization set for that provider. You can set a security profile on a message flow or on individual input nodes or SecurityPEP nodes. If you leave the **Security Profile** property blank, the node inherits the **Security Profile** property that is set at the message flow level. If you leave the property blank at both levels, security is turned off for the node.

1. In the Message Broker Toolkit, right-click the BAR file, then click **Open with** > **Broker Archive Editor**.
2. Click the **Manage and Configure** tab.
3. Click the flow or node on which you want to set the security profile. The properties that you can configure for the message flow or for the node are displayed in the **Properties** view.
4. In the **Security Profile Name** field, select a security profile that has authorization set for *WS-Trust V1.3 STS*.
5. Save the BAR file.

What to do next

For a SOAPInput node to use the token in the WS-Security header (rather than an underlying transport identity) an appropriate policy set and bindings must also be defined and specified. For more information, see "Policy sets" on page 774.

The WS-Trust v1.3 specification is available at: `http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.html`.

Related concepts:

"Identity" on page 390
In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

"Authentication and validation" on page 398
Authentication is the process of establishing the identity of a user or system and verifying that the identity is valid. Applying authentication to a SAML security token involves validating the assertions that it carries and confirming that it is being processed within its validity period.

"Identity mapping" on page 403
Identity mapping is the transformation of a security token from one format to another format, or the federation of an identity from one realm to an equivalent

identity in another realm.

"Authorization" on page 401
Authorization is the process of verifying that an identity token has permission to access a message flow.

"Identity and security token propagation" on page 426
Identity and security token propagation enables the identity and security tokens (associated with each message) to be propagated throughout a message flow, and on to target applications through output or request nodes.

"Security profiles" on page 387
A security profile defines the security operations that are to be performed in a message flow at SecurityPEP nodes and security enabled input and output nodes.

"Security exception processing" on page 429
A security exception is raised when a message flow security failure occurs during security processing in an input node or SecurityPEP node.

Related tasks:

"Configuring the extraction of an identity or security token" on page 447
You can configure the SecurityPEP node or security enabled input nodes to extract the identity or security token from a message and store it in the properties tree identity fields, enabling it to be processed throughout the message flow and propagated at output or request nodes.

"Configuring identity authentication and security token validation" on page 450
You can configure a message flow to perform identity authentication or security token validation using Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token sever (STS) such as Tivoli Federated Identity Manager (TFIM) Version 6.2. Support for TFIM V6.1 is also provided, for compatibility with previous versions of WebSphere Message Broker.

"Configuring identity mapping" on page 463
Configure a security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2, to map the incoming security token and, if required, to authenticate and authorize it.

"Creating a security profile" on page 433
You can create a security profile for use with Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2. Support is also provided for TFIM V6.1, for compatibility with previous versions of WebSphere Message Broker. You can create the security profile by using either the **mqsicreateconfigurableservice** command or an editor in the WebSphere Message Broker Explorer.

"Configuring for identity propagation" on page 492
To enable a message flow to perform identity propagation, the input nodes must extract the identity and the output node must propagate it.

"Setting up message flow security" on page 431
Set up security on a message flow to control access based on the identity of a message passing through the message flow.

Related reference:

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

"HTTPInput node" on page 4474
Use the HTTPInput node to receive an HTTP message from an HTTP client for processing by a message flow.

"SOAPInput node" on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

"SOAPAsyncRequest node" on page 4750
Use the SOAPAsyncRequest node with the SOAPAsyncResponse node to construct a pair of message flows that call a Web service asynchronously.

"HTTPRequest node" on page 4488
Use the HTTPRequest node to interact with a web service.

"MQOutput node" on page 4612
Use the MQOutput node to send messages to clients that connect to the broker using the WebSphere MQ Enterprise Transport and that use the MQI and AMI application programming interfaces.

Creating a module chain in TFIM V6.2:

This topic describes how to create a module chain in Tivoli Federated Identity Manager (TFIM) V6.2.

About this task

When you use a WS-Trust v1.3 security token server (STS) for authentication, authorization, or mapping (or any combination of those operations), a single WS-Trust request is made to the trust service with the required parameters, which control the STS processing.

To enable WebSphere Message Broker to use TFIM V6.2 for authorization, you need to configure TFIM to process the single WS-Trust request from the broker security manager. To configure TFIM, you must create a module chain to handle the request:

Procedure

1. Create a *Custom* module chain, and ensure that the chain performs all the actions that are specified in the broker security profile (Authenticate, Map, Authorize).
2. Set the *RequestType*, *Issuer* and *AppliesTo* properties of the module chain, so that it is invoked for the requests from the security enabled input node or SecurityPEP node. The parameters that are passed by the broker to TFIM are shown in the table in "Authentication, mapping, and authorization with TFIM V6.2 and TAM" on page 419.

What to do next

If your module chain includes an authorization module, and if the module specifies TAM, you must configure TAM to process the authorization requests from TFIM. For more information about how to do this, see "Configuring TAM for authorization using TFIM V6.2" on page 480.

Related concepts:

"Identity" on page 390
In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

"Authentication and validation" on page 398
Authentication is the process of establishing the identity of a user or system and verifying that the identity is valid. Applying authentication to a SAML security token involves validating the assertions that it carries and confirming that it is being processed within its validity period.

"Authorization" on page 401
Authorization is the process of verifying that an identity token has permission to access a message flow.

"Authentication, mapping, and authorization with TFIM V6.2 and TAM" on page 419
You can use WebSphere Message Broker, Tivoli Federated Identity Manager (TFIM) V6.2, and Tivoli Access Manager (TAM) to control authentication, mapping, and authorization.

"Security profiles" on page 387
A security profile defines the security operations that are to be performed in a message flow at SecurityPEP nodes and security enabled input and output nodes.

"Security exception processing" on page 429
A security exception is raised when a message flow security failure occurs during security processing in an input node or SecurityPEP node.

Related tasks:

"Configuring TAM for authorization using TFIM V6.2" on page 480
Configure Tivoli Access Manager (TAM) to enable authorization using Tivoli Federated Identity Manager (TFIM) V6.2.

"Configuring identity authentication and security token validation" on page 450
You can configure a message flow to perform identity authentication or security token validation using Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token sever (STS) such as Tivoli Federated Identity Manager (TFIM) Version 6.2. Support for TFIM V6.1 is also provided, for compatibility with previous versions of WebSphere Message Broker.

"Configuring authorization with a WS-Trust v1.3 STS (TFIM V6.2)" on page 475
You can configure supported message flow input nodes or SecurityPEP nodes to perform authorization of an identity or security token by using a WS-Trust v1.3 compliant security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2.

"Creating a security profile" on page 433
You can create a security profile for use with Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2. Support is also provided for TFIM V6.1, for compatibility with previous versions of WebSphere Message Broker. You can create the security profile by using either the **mqsicreateconfigurableservice** command or an editor in the WebSphere Message Broker Explorer.

"Setting up message flow security" on page 431
Set up security on a message flow to control access based on the identity of a

message passing through the message flow.

Related reference:

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

"HTTPInput node" on page 4474
Use the HTTPInput node to receive an HTTP message from an HTTP client for processing by a message flow.

"SOAPInput node" on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

"SOAPAsyncRequest node" on page 4750
Use the SOAPAsyncRequest node with the SOAPAsyncResponse node to construct a pair of message flows that call a Web service asynchronously.

"HTTPRequest node" on page 4488
Use the HTTPRequest node to interact with a web service.

"MQOutput node" on page 4612
Use the MQOutput node to send messages to clients that connect to the broker using the WebSphere MQ Enterprise Transport and that use the MQI and AMI application programming interfaces.

Configuring TAM for authorization using TFIM V6.2:

Configure Tivoli Access Manager (TAM) to enable authorization using Tivoli Federated Identity Manager (TFIM) V6.2.

About this task

To configure TAM for a TFIM V6.2 TAMAuthorizationSTSModule, complete the following steps using the pdadmin utility.

Procedure

1. Check that the **action group** used by the TFIM authorization module is available. The action group used is *WebService*:

   ```
   action group list
   ```

   If *WebService* is not listed, create it:

   ```
   action group create WebService
   ```

2. Display the **action** in the action group used by the TFIM authorization module. The action used is *"i"*:

   ```
   action list WebService
   ```

   If action *"i" <label> 0* is not listed, create it. The value of <*label*> can vary:

   ```
   action create i <label> 0 WebService
   ```

3. Create the Access Control List (ACL) that will be used to grant access to one or more message flows. First, create the ACL and give the administrators access to it. In this example, `iv-admin` is the administration group and `sec_master` is the main administrator:

   ```
   acl create <AclName>
   acl modify <AclName> set Group iv-admin TcmdbsvaBRxl[WebService]i
   acl modify <AclName> set User sec_master TcmdbsvaBRxl[WebService]i
   ```

4. Grant access to all authenticated users, or specific groups, by adding them to the ACL. Grant any authenticated identity access:

   ```
   acl modify <AclName> set Any-other Trx[WebService]i
   ```

   To add a specific group:

   ```
   acl modify <AclName> set group <GroupName> Trx[WebService]i
   ```

   In these strings, each occurrence of `Trx[ ]` is an action, and corresponds to the value of the stsuser Action context attribute that is passed into the TAMAuthorizationSTSModule. For more information, see "Authentication, mapping, and authorization with TFIM V6.2 and TAM" on page 419.

5. Create a protected object space path in TAM to correspond to the value of the stsuser ObjectName context attribute that is passed into the TAMAuthorizationSTSModule using the following command syntax:

   ```
   objectspace create /<ObjectName>
   ```

   For more information, see "Authentication, mapping, and authorization with TFIM V6.2 and TAM" on page 419.

6. Attach the ACL to the protected object space path that you have created. Each node in the object space inherits ACLs from its parent, and a lower level ACL can override a higher level one. Use the following command syntax to attach an ACL to a node in the object space path:

   ```
   acl attach /<ObjectSpacePath> <AclName>
   ```

What to do next

For further information about configuring TAM, see the IBM Tivoli Access Manager Information Center.

Related concepts:

"Identity" on page 390
In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

"Authentication and validation" on page 398
Authentication is the process of establishing the identity of a user or system and verifying that the identity is valid. Applying authentication to a SAML security token involves validating the assertions that it carries and confirming that it is being processed within its validity period.

"Identity mapping" on page 403
Identity mapping is the transformation of a security token from one format to another format, or the federation of an identity from one realm to an equivalent identity in another realm.

"Authorization" on page 401
Authorization is the process of verifying that an identity token has permission to access a message flow.

"Identity and security token propagation" on page 426
Identity and security token propagation enables the identity and security tokens (associated with each message) to be propagated throughout a message flow, and on to target applications through output or request nodes.

"Security profiles" on page 387
A security profile defines the security operations that are to be performed in a message flow at SecurityPEP nodes and security enabled input and output nodes.

"Security exception processing" on page 429
A security exception is raised when a message flow security failure occurs during security processing in an input node or SecurityPEP node.

"Authentication, mapping, and authorization with TFIM V6.2 and TAM" on page 419
You can use WebSphere Message Broker, Tivoli Federated Identity Manager (TFIM) V6.2, and Tivoli Access Manager (TAM) to control authentication, mapping, and authorization.

Related tasks:

"Creating a module chain in TFIM V6.2" on page 478
This topic describes how to create a module chain in Tivoli Federated Identity Manager (TFIM) V6.2.

"Configuring the extraction of an identity or security token" on page 447
You can configure the SecurityPEP node or security enabled input nodes to extract the identity or security token from a message and store it in the properties tree identity fields, enabling it to be processed throughout the message flow and propagated at output or request nodes.

"Configuring identity authentication and security token validation" on page 450
You can configure a message flow to perform identity authentication or security token validation using Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token sever (STS) such as Tivoli Federated Identity Manager (TFIM) Version 6.2. Support for TFIM V6.1 is also provided, for compatibility with previous versions of WebSphere Message Broker.

"Configuring identity mapping" on page 463
Configure a security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2, to map the incoming security token and, if required, to authenticate and authorize it.

"Creating a security profile" on page 433
You can create a security profile for use with Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2. Support is also provided for TFIM V6.1, for compatibility with previous versions of WebSphere Message Broker. You can create the security profile by using either the **mqsicreateconfigurableservice** command or an editor in the WebSphere Message Broker Explorer.

"Configuring for identity propagation" on page 492
To enable a message flow to perform identity propagation, the input nodes must extract the identity and the output node must propagate it.

"Setting up message flow security" on page 431
Set up security on a message flow to control access based on the identity of a message passing through the message flow.

Related reference:

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

"HTTPInput node" on page 4474
Use the HTTPInput node to receive an HTTP message from an HTTP client for processing by a message flow.

"SOAPInput node" on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

"SOAPAsyncRequest node" on page 4750
Use the SOAPAsyncRequest node with the SOAPAsyncResponse node to construct a pair of message flows that call a Web service asynchronously.

"HTTPRequest node" on page 4488
Use the HTTPRequest node to interact with a web service.

"MQOutput node" on page 4612
Use the MQOutput node to send messages to clients that connect to the broker using the WebSphere MQ Enterprise Transport and that use the MQI and AMI application programming interfaces.

Configuring authorization with TFIM V6.1:

You can configure a message flow to perform authorization on an identity by using Tivoli Federated Identity Manager (TFIM) V6.1.

Before you begin

Before you start:

Before you configure a message flow to perform authorization with TFIM V6.1:

- Check that an appropriate security profile exists, or create a new security profile. See "Creating a security profile for TFIM V6.1" on page 444.
- Define the required users and groups in TFIM.

About this task

Note: Support for TFIM V6.1 is included for compatibility with previous versions of WebSphere Message Broker. If possible, upgrade to TFIM V6.2 and follow the instructions in "Configuring authorization with a WS-Trust v1.3 STS (TFIM V6.2)" on page 475.

The broker security manager issues an authorization request to the TFIM trust service with the following three parameters, which select the TFIM module chain to be used:

- Issuer = Properties.IdentitySourceIssuedBy
- Applies To = The Fully Qualified Name of the Flow: <*Brokername*>.<*Execution Group Name*>.<*Message Flow Name*>
- Token = Properties.IdentitySourceToken

Authorization is performed with TFIM using an instance of the TFIM AuthorizationSTSModule in the selected module chain. The TFIM AuthorizationSTSModule must be set with `Mode = Other`. This AuthorizationSTSModule authorizes a user by checking an Access Control List (ACL) from Tivoli Access Manager (TAM). TFIM performs the authorization check by verifying that the action "i" (invoke) has been granted in an ACL for the `WebService` action group.

The ACL is found starting from the root of the TAM object space using a path formed from the Authorization module **`Web service protected object name`** parameter, followed by the **`Port Type`** and **`Operation Name`** from the authorization request. When the broker makes an authorization request to TFIM, the **`Port Type`** and **`Operation Name`** parameters have the following values:

- PortType:<*Message flow name*>
- Operation "MessageFlowAccess"

Therefore, the ACL is found at this location in the TAM object space:

```
/<WSProtectedObjectName>.<MessageFlowName>."MessageFlowAccess"
```

For more information about this process and the parameters, see “Authentication, mapping, and authorization with TFIM V6.1 and TAM” on page 416.

Steps for enabling TFIM authorization:

Procedure

To enable an existing message flow to perform authorization with TFIM, use the Broker Archive editor to select a security profile that has authorization enabled. You can set a security profile on a message flow or on individual input nodes. If no security profile is set for the input nodes, the setting is inherited from the setting on the message flow.

1. Switch to the Broker Application Development perspective.
2. In the Broker Development view, right-click the BAR file, then click **Open with** > **Broker Archive Editor**.
3. Click the **Manage and Configure** tab.
4. Click the flow or node on which you want to set the security profile. The properties that you can configure for the message flow or for the node are displayed in the **Properties** view.
5. In the **Security Profile Name** field, select a security profile that has authorization enabled.
6. Save the BAR file.

What to do next

For a SOAPInput node to use the identity in the WS-Security header (rather than an underlying transport identity) an appropriate policy set and bindings must also be defined and specified. For more information, see "Policy sets" on page 774.

In addition to configuring Message Broker to perform authorization with TFIM, you must configure TFIM and TAM. For information about how to do this, see the following topics:

- "Creating a module chain in TFIM V6.1" on page 487
- "Configuring TAM for authorization using TFIM V6.1" on page 489.

For further information on how to configure TFIM, see the IBM Tivoli Federated Identity Manager Information Center.

Related concepts:

"Identity" on page 390
In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

"Authentication and validation" on page 398
Authentication is the process of establishing the identity of a user or system and verifying that the identity is valid. Applying authentication to a SAML security token involves validating the assertions that it carries and confirming that it is being processed within its validity period.

"Identity mapping" on page 403
Identity mapping is the transformation of a security token from one format to another format, or the federation of an identity from one realm to an equivalent identity in another realm.

"Authorization" on page 401
Authorization is the process of verifying that an identity token has permission to access a message flow.

"Identity and security token propagation" on page 426
Identity and security token propagation enables the identity and security tokens (associated with each message) to be propagated throughout a message flow, and on to target applications through output or request nodes.

"Security profiles" on page 387
A security profile defines the security operations that are to be performed in a message flow at SecurityPEP nodes and security enabled input and output nodes.

"Security exception processing" on page 429
A security exception is raised when a message flow security failure occurs during security processing in an input node or SecurityPEP node.

Related tasks:

"Configuring the extraction of an identity or security token" on page 447
You can configure the SecurityPEP node or security enabled input nodes to extract the identity or security token from a message and store it in the properties tree identity fields, enabling it to be processed throughout the message flow and propagated at output or request nodes.

"Configuring identity authentication and security token validation" on page 450
You can configure a message flow to perform identity authentication or security token validation using Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token sever (STS) such as Tivoli Federated Identity Manager (TFIM) Version 6.2. Support for TFIM V6.1 is also provided, for compatibility with previous versions of WebSphere Message Broker.

"Configuring identity mapping" on page 463
Configure a security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2, to map the incoming security token and, if required, to authenticate and authorize it.

"Creating a security profile" on page 433
You can create a security profile for use with Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2. Support is also provided for TFIM V6.1, for compatibility with previous versions of WebSphere Message Broker. You can create the security profile by using either the **mqsicreateconfigurableservice** command or an editor in the WebSphere Message Broker Explorer.

"Configuring for identity propagation" on page 492
To enable a message flow to perform identity propagation, the input nodes must extract the identity and the output node must propagate it.

"Setting up message flow security" on page 431
Set up security on a message flow to control access based on the identity of a message passing through the message flow.

Related reference:

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

"HTTPInput node" on page 4474
Use the HTTPInput node to receive an HTTP message from an HTTP client for processing by a message flow.

"SOAPInput node" on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

"SOAPAsyncRequest node" on page 4750
Use the SOAPAsyncRequest node with the SOAPAsyncResponse node to construct a pair of message flows that call a Web service asynchronously.

"HTTPRequest node" on page 4488
Use the HTTPRequest node to interact with a web service.

"MQOutput node" on page 4612
Use the MQOutput node to send messages to clients that connect to the broker using the WebSphere MQ Enterprise Transport and that use the MQI and AMI application programming interfaces.

Creating a module chain in TFIM V6.1:

This topic describes how to create a module chain in Tivoli Federated Identity Manager (TFIM) V6.1.

About this task

To enable Message Broker to use TFIM V6.1 for authorization, you need to configure TFIM to process the security request from the message flow. To do this you need to create a module chain in TFIM to handle the request:

Procedure

1. Create a *Custom* module chain, and ensure that the chain performs all the actions required (Authenticate, Map, Authorize).
2. Set the *Issuer* and *AppliesTo* properties of the module chain, so that it is invoked for the requests from the message flow. When the broker makes a request to TFIM, the **Port Type** and **Operation Name** parameters have the following values:
   - PortType:<*Message flow name*>
   - Operation "MessageFlowAccess"

   The *RequestType* is always set to *Validate*.
3. To perform authorization in a module chain, add an instance of the Authorization module in *other* mode, which allows the module parameter **Web Service protected object name** to be set for the Tivoli Access Manager (TAM) configuration.

What to do next

When you have created the module chain in TFIM, see "Configuring TAM for authorization using TFIM V6.1" on page 489 for information on how to configure TAM to process authorization requests from TFIM.

Related concepts:

"Identity" on page 390
In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

"Authentication and validation" on page 398
Authentication is the process of establishing the identity of a user or system and verifying that the identity is valid. Applying authentication to a SAML security token involves validating the assertions that it carries and confirming that it is being processed within its validity period.

"Authorization" on page 401
Authorization is the process of verifying that an identity token has permission to access a message flow.

"Authentication, mapping, and authorization with TFIM V6.1 and TAM" on page 416
Use WebSphere Message Broker, Tivoli Federated Identity Manager (TFIM) V6.1, and Tivoli Access Manager (TAM) to control authentication, mapping, and authorization.

"Security profiles" on page 387
A security profile defines the security operations that are to be performed in a message flow at SecurityPEP nodes and security enabled input and output nodes.

"Security exception processing" on page 429
A security exception is raised when a message flow security failure occurs during

security processing in an input node or SecurityPEP node.

Related tasks:

"Configuring identity authentication and security token validation" on page 450
You can configure a message flow to perform identity authentication or security token validation using Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token sever (STS) such as Tivoli Federated Identity Manager (TFIM) Version 6.2. Support for TFIM V6.1 is also provided, for compatibility with previous versions of WebSphere Message Broker.

"Configuring TAM for authorization using TFIM V6.1" on page 489
This topic describes how to configure Tivoli Access Manager (TAM) to enable authorization using Tivoli Federated Identity Manager (TFIM) V6.1.

"Creating a security profile" on page 433
You can create a security profile for use with Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2. Support is also provided for TFIM V6.1, for compatibility with previous versions of WebSphere Message Broker. You can create the security profile by using either the **mqsicreateconfigurableservice** command or an editor in the WebSphere Message Broker Explorer.

"Setting up message flow security" on page 431
Set up security on a message flow to control access based on the identity of a message passing through the message flow.

Related reference:

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

"HTTPInput node" on page 4474
Use the HTTPInput node to receive an HTTP message from an HTTP client for processing by a message flow.

"SOAPInput node" on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

"SOAPAsyncRequest node" on page 4750
Use the SOAPAsyncRequest node with the SOAPAsyncResponse node to construct a pair of message flows that call a Web service asynchronously.

"HTTPRequest node" on page 4488
Use the HTTPRequest node to interact with a web service.

"MQOutput node" on page 4612
Use the MQOutput node to send messages to clients that connect to the broker using the WebSphere MQ Enterprise Transport and that use the MQI and AMI application programming interfaces.

Configuring TAM for authorization using TFIM V6.1:

This topic describes how to configure Tivoli Access Manager (TAM) to enable authorization using Tivoli Federated Identity Manager (TFIM) V6.1.

About this task

To configure TAM to process an authorization request from TFIM, complete the following steps. The examples relate to the TAM Version 6.01 pdadmin utility:

Procedure

1. Check that the **action group** used by the TFIM authorization module is available. The action group used is *WebService*:

   ```
   action group list
   ```

   If *WebService* is not listed, create it:

   ```
   action group create WebService
   ```

2. Display the **action** in the action group used by the TFIM authorization module. The action used is *"i"*:

   ```
   action list WebService
   ```

   If action *"i" <label> 0* is not listed, create it. The value of <*label*> can vary:

   ```
   action create i <label> 0 WebService
   ```

3. Create the Access Control List (ACL) that will be used to grant access to one or more message flows. First, create the ACL and give the administrators access to it. In this example, `iv-admin` is the administration group and `sec_master` is the main administrator:

   ```
   acl create <AclName>
   acl modify <AclName> set Group iv-admin TcmdbsvaBRxl[WebService]i
   acl modify <AclName> set User sec_master TcmdbsvaBRxl[WebService]i
   ```

4. Grant access to all authenticated users, or specific groups, by adding them to the ACL. Grant any authenticated identity access:

   ```
   acl modify <AclName> set Any-other Trx[WebService]i
   ```

   To add a specific group:

   ```
   acl modify <AclName> set group <GroupName> Trx[WebService]i
   ```

5. Define protected object spaces in TAM for authorization of message flows:
   a. Create the *application container object* as the root of the protected object space. This is the name that is used to link an instance of a TFIM AuthorizationSTSModule (within a module chain) into the TAM object space. The container object name is specified to match the **Web Service protected object name** parameter on a TFIM Authorization module.

      ```
      objectspace create /<ContainerObjectName> <Description> 14
      ```

   b. Create the container objects in the tree for each broker message flow that is being authorized. The message flow name is used by TFIM to locate a point in the TAM Object Space tree for Authorization, through the attached ACL. The message flow name is passed as the **PortType** in the WS-Trust request to TFIM. Use the following command to create the object tree node representing each flow to be authorized:

```
object create /<ContainerObjectName>/<FlowName> <Description> 11 ispolicyattachable yes
```

The **ispolicyattachable** parameter applies to all levels, so you can attach an ACL at any level.

c. Create the leaf object that represents the authorized object to grant access to the message flow. This is the fixed string *MessageFlowAccess*, which the broker sends to TFIM through the TFIM **OperationName** extension to the WS-Trust request. A fixed name (MessageFlowAccess) is used instead of a true operation name, because the broker does not necessarily know at the input node which operation a flow is going to perform. The command syntax is:

```
object create /<ContainerObjectName>/<FlowName>/MessageFlowAccess <Description> 12 ispolicyattachable yes
```

where <*FlowName*> has been created in a previous step.

6. Attach the ACL to the relevant node in the protected object space tree. Each node in the object space inherits ACLs from its parent, and a lower level ACL can override a higher level one. Use the following command syntax to attach an ACL to a node in the object space:

```
acl attach /<ObjectSpacePath> <AclName>
```

To attach an ACL to the leaf node:

```
acl attach /<ContainerObjectName>/<FlowName>/MessageFlowAccess <AclName>
```

What to do next

For further information about configuring TAM, see the IBM Tivoli Access Manager Information Center.

Related concepts:

"Identity" on page 390
In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

"Authentication and validation" on page 398
Authentication is the process of establishing the identity of a user or system and verifying that the identity is valid. Applying authentication to a SAML security token involves validating the assertions that it carries and confirming that it is being processed within its validity period.

"Identity mapping" on page 403
Identity mapping is the transformation of a security token from one format to another format, or the federation of an identity from one realm to an equivalent identity in another realm.

"Authorization" on page 401
Authorization is the process of verifying that an identity token has permission to access a message flow.

"Identity and security token propagation" on page 426
Identity and security token propagation enables the identity and security tokens (associated with each message) to be propagated throughout a message flow, and on to target applications through output or request nodes.

"Security profiles" on page 387
A security profile defines the security operations that are to be performed in a message flow at SecurityPEP nodes and security enabled input and output nodes.

"Security exception processing" on page 429
A security exception is raised when a message flow security failure occurs during security processing in an input node or SecurityPEP node.

Related tasks:

"Configuring the extraction of an identity or security token" on page 447
You can configure the SecurityPEP node or security enabled input nodes to extract the identity or security token from a message and store it in the properties tree identity fields, enabling it to be processed throughout the message flow and propagated at output or request nodes.

"Configuring identity authentication and security token validation" on page 450
You can configure a message flow to perform identity authentication or security token validation using Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token sever (STS) such as Tivoli Federated Identity Manager (TFIM) Version 6.2. Support for TFIM V6.1 is also provided, for compatibility with previous versions of WebSphere Message Broker.

"Configuring identity mapping" on page 463
Configure a security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2, to map the incoming security token and, if required, to authenticate and authorize it.

"Creating a security profile" on page 433
You can create a security profile for use with Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2. Support is also provided for TFIM V6.1, for compatibility with previous versions of WebSphere Message Broker. You can create the security profile by using either the **mqsicreateconfigurableservice** command or an editor in the WebSphere Message Broker Explorer.

"Configuring for identity propagation" on page 492
To enable a message flow to perform identity propagation, the input nodes must extract the identity and the output node must propagate it.

"Setting up message flow security" on page 431
Set up security on a message flow to control access based on the identity of a message passing through the message flow.

Related reference:

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

"HTTPInput node" on page 4474
Use the HTTPInput node to receive an HTTP message from an HTTP client for processing by a message flow.

"SOAPInput node" on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

"SOAPAsyncRequest node" on page 4750
Use the SOAPAsyncRequest node with the SOAPAsyncResponse node to construct a pair of message flows that call a Web service asynchronously.

"HTTPRequest node" on page 4488
Use the HTTPRequest node to interact with a web service.

"MQOutput node" on page 4612
Use the MQOutput node to send messages to clients that connect to the broker using the WebSphere MQ Enterprise Transport and that use the MQI and AMI application programming interfaces.

Configuring for identity propagation

To enable a message flow to perform identity propagation, the input nodes must extract the identity and the output node must propagate it.

Before you begin

Before you start:

Before you can configure a message flow to perform identity propagation, you must check that an appropriate security profile exists, or create a new security profile. See "Creating a security profile" on page 433.

About this task

An input node extracts security tokens if it is configured with a security profile at deployment time. An output node propagates an identity if it is configured with a security profile that enables propagation at deployment time.

To enable a message flow to perform identity propagation, complete the following steps.

Procedure

By using the Broker Archive editor, select a security profile that has identity propagation enabled. You can use the Default Propagation profile, which is a predefined profile that requests only identity propagation. You can set a security profile on a message flow or on individual input and output nodes. If no security profile is set for the input and output nodes, the setting is inherited from the setting on the message flow.

1. In the Broker Development view, right-click the BAR file, then click **Open with** > **Broker Archive Editor**.
2. Click the **Manage and Configure** tab.
3. Click the flow or node on which you want to set the security profile. The properties that you can configure for the message flow or for the node are displayed in the **Properties** view.
4. In the **Security Profile Name** field, select a security profile that has identity propagation enabled.
5. Save the BAR file.

What to do next

For a SOAPRequest or SOAPAsyncRequest node, you can define an appropriate policy set and bindings to specify how the propagated identity is placed in the

WS-Security header (rather than the underlying transport headers). For more information, see “Policy sets” on page 774.

On SOAPRequest and SOAPAsyncRequest nodes, only Username and SAML tokens can be propagated. However, on the SOAPRequest and SOAPAsyncRequest nodes with a Kerberos policy set and bindings, a Username and password token can be propagated into the node to provide the Kerberos client credentials.

For the SAPRequest node, you can propagate only the user name and password. For the CICSRequest and IMSRequest nodes, you can propagate the user name, or the user name and password.

If the message identity does not contain enough information for identity propagation, you can use any of the following methods to acquire the necessary information:

- Take the information from the message body. For example, if the message comes from WebSphere MQ with only a username token, and the output is an HTTP request node requiring a Username + Password token, the password might be present in the body of the incoming message. For more information, see “Configuring the extraction of an identity or security token” on page 447.
- Configure an identity mapper using TFIM. For more information, see the IBM Tivoli Federated Identity Manager Information Center.
- Use ESQL or Java to set the Mapped Identity fields in the Properties tree.

Related concepts:

“Identity” on page 390
In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

“Authentication and validation” on page 398
Authentication is the process of establishing the identity of a user or system and verifying that the identity is valid. Applying authentication to a SAML security token involves validating the assertions that it carries and confirming that it is being processed within its validity period.

“Identity mapping” on page 403
Identity mapping is the transformation of a security token from one format to another format, or the federation of an identity from one realm to an equivalent identity in another realm.

“Authorization” on page 401
Authorization is the process of verifying that an identity token has permission to access a message flow.

“Identity and security token propagation” on page 426
Identity and security token propagation enables the identity and security tokens (associated with each message) to be propagated throughout a message flow, and on to target applications through output or request nodes.

“Security profiles” on page 387
A security profile defines the security operations that are to be performed in a message flow at SecurityPEP nodes and security enabled input and output nodes.

“Security exception processing” on page 429
A security exception is raised when a message flow security failure occurs during security processing in an input node or SecurityPEP node.

Related tasks:

"Configuring the extraction of an identity or security token" on page 447
You can configure the SecurityPEP node or security enabled input nodes to extract the identity or security token from a message and store it in the properties tree identity fields, enabling it to be processed throughout the message flow and propagated at output or request nodes.

"Configuring identity authentication and security token validation" on page 450
You can configure a message flow to perform identity authentication or security token validation using Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token sever (STS) such as Tivoli Federated Identity Manager (TFIM) Version 6.2. Support for TFIM V6.1 is also provided, for compatibility with previous versions of WebSphere Message Broker.

"Configuring identity mapping" on page 463
Configure a security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2, to map the incoming security token and, if required, to authenticate and authorize it.

"Creating a security profile" on page 433
You can create a security profile for use with Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2. Support is also provided for TFIM V6.1, for compatibility with previous versions of WebSphere Message Broker. You can create the security profile by using either the **mqsicreateconfigurableservice** command or an editor in the WebSphere Message Broker Explorer.

"Configuring authorization" on page 470
Configure the broker to use an external security provider to authorize an identity in a message flow. You can use either Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS) such as Tivoli Federated Identity Manager (TFIM) V6.2. Support for TFIM V6.1 is also provided for compatibility with previous versions of WebSphere Message Broker.

"Setting up message flow security" on page 431
Set up security on a message flow to control access based on the identity of a message passing through the message flow.

Related reference:

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

"HTTPInput node" on page 4474
Use the HTTPInput node to receive an HTTP message from an HTTP client for processing by a message flow.

"SOAPInput node" on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

"SOAPAsyncRequest node" on page 4750
Use the SOAPAsyncRequest node with the SOAPAsyncResponse node to construct a pair of message flows that call a Web service asynchronously.

"HTTPRequest node" on page 4488
Use the HTTPRequest node to interact with a web service.

"MQOutput node" on page 4612
Use the MQOutput node to send messages to clients that connect to the broker using the WebSphere MQ Enterprise Transport and that use the MQI and AMI application programming interfaces.

Database security

You can access databases from the message flows that you deploy to your brokers, and must therefore consider the steps you might want to take to secure that access.

About this task

After you have defined user IDs that you want the broker to use for database access by using the **mqsisetdbparms** command, you must authorize those IDs so that the broker can access your databases from deployed message flows.

If you have migrated your broker from a previous release, the broker accessed a database for its own use, and you might have defined the user ID and password used to access that database by specifying a database connection user ID and password with the **-u** and **-p** parameters on the **mqsicreatebroker** command. Alternatively, you might have used the broker service user ID and its password (specified with the **-i** and **-a** parameters on the same command). When you migrate the broker, these parameters are migrated and stored, and are used by the migrated broker for access to databases that do not have specific ID access defined. On WebSphere Message Broker Version 7.0, you can use only the **mqsisetdbparms** command to set or change values for database access by the broker, because the parameters have been deprecated. (The service user ID and password are required on Windows, but are no longer used for database access.)

WebSphere Message Broker does not provide special commands to administer databases. Discuss your database security requirements with the database administrator for the database manager that you are using, or refer to the documentation provided by your database supplier.

For further information about possible security requirements, and examples of setting up security for DB2 and Oracle databases, see "Authorizing access to user databases" on page 662.

Related concepts:

"Security overview" on page 351
When you are designing a WebSphere Message Broker application it is important to consider the security measures that are needed to protect the information in the system.

Related reference:

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

Diagnosing security problems

This topic explains how to find out why access to a secured flow has been denied.

About this task

By default, security exceptions occurring in an input node are not processed in the same way as other errors (see "Security exception processing" on page 429). Security exceptions are not logged to the system event log, to prevent a security denial of service attack filling the logs and destabilizing the system.

This means that, by default, you cannot diagnose input node security exceptions in the same way as other errors. However, in a SecurityPEP node, a failing security operation causes a security exception to be raised, wrapped in a normal recoverable exception, which invokes the error handling that is provided by the message flow.

To see what might be causing the security exceptions, you can do either of the following things:

- Select the **Treat Security Exceptions as normal exceptions** property on the input nodes.
- Use the user trace.

The following steps show you how to use the user trace to find out why access to a secured message flow has been denied:

Procedure

1. Use the **mqsireloadsecurity** command to clear the security cache, so that the traced request goes to the security provider rather than using a result held in the cache. This ensures that the reason codes returned from the security provider are displayed in the traced exception.
2. Enable user trace for the message flow, using either the workbench or the **mqsichangetrace** command (see "Starting user trace" on page 3197 for more information).
3. Resend the request that has been rejected by the security provider.
4. Stop the user trace, using either the workbench or the **mqsichangetrace** command.
5. Use the **mqsireadlog** command to examine the trace information that was recorded by the user trace. This trace information contains the error codes provided by the broker and the security provider.

Related concepts:

"Security exception processing" on page 429
A security exception is raised when a message flow security failure occurs during security processing in an input node or SecurityPEP node.

"Message flow security overview" on page 383
WebSphere Message Broker provides a security manager, which enables you to control access to individual messages in a message flow, using the identity of the

message.

Related tasks:

"Setting up message flow security" on page 431
Set up security on a message flow to control access based on the identity of a message passing through the message flow.

"Starting user trace" on page 3197
Use user trace for debugging your applications; you can trace brokers, execution groups, and deployed message flows. Start user trace facilities using the **mqsichangetrace** command or the WebSphere Message Broker Explorer.

"Stopping user trace" on page 3202
Use user trace for debugging your applications; you can trace brokers, execution groups, and deployed message flows. Stop user trace facilities by using the **mqsichangetrace** command or the WebSphere Message Broker Explorer.

"Retrieving user trace" on page 3204
Use the **mqsireadlog** command to access the trace information that is recorded by the user trace facilities.

"Creating a security profile" on page 433
You can create a security profile for use with Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2. Support is also provided for TFIM V6.1, for compatibility with previous versions of WebSphere Message Broker. You can create the security profile by using either the **mqsicreateconfigurableservice** command or an editor in the WebSphere Message Broker Explorer.

Related reference:

"**mqsireloadsecurity** command" on page 3911
Use the **mqsireloadsecurity** command to force the immediate expiry of some or all the entries in the security cache.

"**mqsichangetrace** command" on page 3822
Use the **mqsichangetrace** command to set the tracing characteristics for a broker.

"**mqsireadlog** command" on page 3905
Use the **mqsireadlog** command to retrieve trace records for the specified component.

"Parameter values for the securitycache component" on page 3815
Select the objects and properties associated with the securitycache component that you want to change.

"SecurityPEP node" on page 4729
Use the SecurityPEP node in a message flow to invoke the message flow security manager at any point in the message flow.

Broker component security

You must consider several security aspects when you are setting up brokers running on Windows, Linux, or UNIX platforms.

About this task

For an introduction to various aspects of security, see "Security overview" on page 351.

This section does not apply to z/OS. Refer to "Setting up z/OS security" on page 556 and "Summary of required access (z/OS)" on page 3985 for information about setting up broker security on z/OS.

Before you start setting up security for your brokers, refer to “Planning for security when you install WebSphere Message Broker” on page 353, which contains links to security information that you need before, during, and after installation of WebSphere Message Broker.

Use the following list of tasks as a security checklist. Each item comprises a list of reminders or questions about the security tasks to consider for your brokers. The answers to the questions provide the security information that you require to configure your brokers, and also give you information about other security controls that you might want to employ.

- “Creating user IDs”
- “Considering security for the WebSphere Message Broker Toolkit and WebSphere Message Broker Explorer” on page 500
- “Considering security for a broker” on page 501
- “Implementing SSL authentication” on page 504
- “Using security exits” on page 555

Related tasks:

“Planning for security when you install WebSphere Message Broker” on page 353
The Installation Guide describes the security tasks that you must complete before, during, and after installation.

“Considering security for a broker” on page 501
Consider several factors when you are deciding which users can execute broker commands, and which users can control security for other broker resources.

“Considering security for the WebSphere Message Broker Toolkit and WebSphere Message Broker Explorer” on page 500
Set up appropriate levels of security for the WebSphere Message Broker Toolkit and WebSphere Message Broker Explorer.

“Setting up z/OS security” on page 556
On z/OS, you must complete several security configuration tasks before WebSphere Message Broker can work correctly.

Related reference:

“Security requirements for Windows systems” on page 3651
Security requirements depend on the administrative task that you want to perform.

“Security requirements for Linux and UNIX platforms” on page 3648
View a summary of the authorizations in a Linux or UNIX environment.

“Summary of required access (z/OS)” on page 3985
The professionals in your organization require access to components and resources on z/OS.

Creating user IDs

When you plan the administration of your broker configuration, you might have to define one or more user IDs for the tasks associated with particular roles.

About this task

Some operating systems, and other products, impose restrictions on user IDs:

- On Windows systems, user IDs can be up to 12 characters long, but on Linux, UNIX, and z/OS systems, they are restricted to eight characters.
- Database products might also restrict user IDs to eight characters; for example DB2 has this restriction. If you have a mixed environment, ensure that the user IDs that you use are limited to a maximum of eight characters.

- Ensure that the case (upper, lower, or mixed) of user IDs is consistent. In some environments, uppercase and lowercase user IDs are considered the same, but in other environments, user IDs of different case are considered unique.

  For example, on Windows systems, the user IDs 'tester' and 'TESTER' are identical; on Linux and UNIX systems, they are recognized as different user IDs.
- Check the validity of spaces and special characters in user IDs to ensure that, if used, these characters are accepted by all relevant systems and products that you install.

Consider the following roles:

Procedure

- Administrator user IDs that can issue **mqsi*** commands. Ensure that all these user IDs are members of the mqbrkrs group.

  If you enable broker administration security, you must set up additional authorization for user IDs for the commands listed in "Commands and authorizations for broker administration security" on page 3646. For more information about broker administration security, see "Broker administration security overview" on page 362.
- On Windows only, service user IDs under which brokers run. For further information, see "Deciding which user account to use for the broker service ID" on page 502.
- Workbench users. See "Considering security for the WebSphere Message Broker Toolkit and WebSphere Message Broker Explorer" on page 500 for information about checking and securing connections from the WebSphere Message Broker Toolkit.

Related concepts:

"Security overview" on page 351
When you are designing a WebSphere Message Broker application it is important to consider the security measures that are needed to protect the information in the system.

"Broker administration security overview" on page 362
Broker administration security controls the rights of users to complete administrative tasks for a broker and its resources.

Related tasks:

"Considering security for a broker" on page 501
Consider several factors when you are deciding which users can execute broker commands, and which users can control security for other broker resources.

"Authorizing access to user databases" on page 662
When you have created a user database, you must authorize the broker and its execution groups to access it.

Related reference:

"Security requirements for Windows systems" on page 3651
Security requirements depend on the administrative task that you want to perform.

"Security requirements for Linux and UNIX platforms" on page 3648
View a summary of the authorizations in a Linux or UNIX environment.

"Installation Guide" on page 233
Installation information for WebSphere Message Broker is provided in the *Installation Guide* that is supplied as a PDF file with your product package.

Considering security for the WebSphere Message Broker Toolkit and WebSphere Message Broker Explorer

Set up appropriate levels of security for the WebSphere Message Broker Toolkit and WebSphere Message Broker Explorer.

Before you begin

When you create a broker, a default SVRCONN channel, `SYSTEM.BKR.CONFIG`, is created. This channel supports connections from one or more remote clients to the broker. Clients that are running on the same computer as the broker connect directly to the queue manager; they do not require a connection through a channel.

If you want to secure the connection between your WebSphere Message Broker Toolkit session, or WebSphere Message Broker Explorer session, and the broker, you can configure the SVRCONN channel to specify security options.

You can use a single channel for all client connections, create a channel for each client connection, or share connections between two or more clients that have the same security requirements. You can use the default channel, and create additional channels if required. If you do not use the default channel, you must set the alternative name in the connection properties.

You can secure the connection between the WebSphere Message Broker Explorer, the WebSphere Message Broker Toolkit, a command that uses the CMP interface (**mqsichangeresourcestats**, **mqsicreateexecutiongroup**, **mqsideleteexecutiongroup**, **mqsideploy**, **mqsilist**, **mqsimode**, **mqsireloadsecurity**, **mqsireportresourcestats**, **mqsistartmsgflow**, **mqsistopmsgflow**), or a CMP application, by using one or both of the following options:

- WebSphere MQ channel security exits
- Secure Socket Layer (SSL)

See "Securing the channel between the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer and the broker" for details about how to set up the security you want.

Ensure that user IDs are not more than eight characters long.

Securing the channel between the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer and the broker

About this task

Implement one or both of the following options for the WebSphere Message Broker Toolkit and WebSphere Message Broker Explorer clients:

Procedure

- Create and enable a pair of WebSphere MQ security exits to run at the client and broker ends of the SVRCONN channel that connects the two components. Program these exits to verify client users with the security manager on the computer on which the broker is running.

  For more information about creating and enabling security exits, see "Security exits" on page 354.
- Implement SSL security on the channel. You must have the appropriate software to manage SSL certificate stores; for example, you can install either the

WebSphere MQ Client or the Server, and use the IBM Key Management tools for the client. You can use either JKS or PKCS12 stores.

1. Use WebSphere MQ facilities to update the SVRCONN definition to specify the required value in the SSLCIPH attribute.
2. In the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer, define the connection to the broker. You can set the SSL fields only when you define the connection; you cannot change them later. If you have already defined your connection, delete it, and define it again.
3. Select the cipher suite that matches the value you set for the SSLCIPH property of the SVRCONN channel.
4. Enter the full path and name for the keystore and truststore, or click **Browse** to search for them.
5. Add the queue manager certificate to the client truststore.
6. If you want server-only (one way) certification, set the SVRCONN channel attribute SSLCAUTH to OPTIONAL.
7. If you want mutual (two way) certification:
   - Set the SVRCONN channel attribute SSLCAUTH to REQUIRED.
   - Add the client certificate to the queue manager truststore.

For further details about setting up SSL configuration, see "Enabling SSL on the WebSphere MQ Java Client" on page 540.

For more information about configuring connections to be secured with SSL, see the WebSphere MQ Java Client developerWorks article.

Related concepts:

"Security exits" on page 354
Use security exit programs to verify that the partner at the other end of a connection is genuine.

Related tasks:

"Using security exits" on page 555
Define a security exit on the WebSphere MQ channel when you create a broker connection.

Related reference:

"Security requirements for Windows systems" on page 3651
Security requirements depend on the administrative task that you want to perform.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

Considering security for a broker

Consider several factors when you are deciding which users can execute broker commands, and which users can control security for other broker resources.

About this task

Although most security for the broker and broker resources is optional, you might find it appropriate to restrict the tasks that some user IDs can perform. You can then apply greater control to monitor changes.

You can control all broker administration tasks by enabling broker administration security when you create a broker. You can also change existing brokers to enable administration security. This option is described in "Setting up broker administration security" on page 368, and is independent of the options described in this section.

When you are deciding which users are to perform the different tasks, consider the following steps:

Procedure

1. "Deciding which user account to use for the broker service ID"
2. "Setting security on the broker queues" on page 503
3. "Securing the broker registry" on page 503

Deciding which user account to use for the broker service ID

About this task

On a Linux or UNIX operating system, when you run the **mqsistart** command with a user ID that is a member of the mqm and mqbrkrs groups, the user ID under which you run the **mqsistart** command becomes the user ID under which the broker component process runs.

On the Windows platform the broker runs under a service user account. To decide which user ID to use for the broker service ID answer the following questions:

Procedure

1. Do you want your broker to run under a Windows local account?
   a. No: Go to the next question.
   b. Yes: Ensure that your user ID has the following characteristics:
      - It is defined in your local domain.
      - It is a member of the mqbrkrs group.

      Go to "Setting security on the broker queues" on page 503.
2. Do you want your broker to run under a Windows domain account?
   a. No: Go to the next question.
   b. Yes: Assume that your computer named, for example, WKSTN1, is a member of a domain named DOMAIN1. When you run a broker using, for example, DOMAIN1\user1, ensure that:
      - Your user ID has been granted the `Logon as a service` privilege (from the Local Security Policy).
      - DOMAIN1\user1 is a member of DOMAIN1\MyDomainGroup group, where MyDomainGroup is a domain group which you have defined on your domain controller.
      - DOMAIN1\MyDomainGroup is a member of WKSTN1\mqbrkrs.

      Go to "Setting security on the broker queues" on page 503.
3. Do you want your broker to run under theWindows built in `LocalSystem` account?

a. Yes: Specify `LocalSystem` for the *–i* parameter on the **mqsicreatebroker** or **mqsichangebroker** command.

   In either case you must enter the *–a* (password) parameter on the command line, but the value entered is ignored.

   Go to "Setting security on the broker queues."

Results

Note that for cases one and two above, the user ID chosen must be granted the `Logon as a service` privilege.

This is normally done automatically by the **mqsichangebroker** command or the **mqsichangeproperties** command when a service user ID is specified that does not have this privilege.

However, if you want to do this manually before running these commands, you can do this by using the Local Security Policy tool in Windows, which you can access by selecting **Control Panel** > **Performance and maintenance** > **Administrative Tools** > **Local Security Policy**.

Setting security on the broker queues

About this task

When you run the **mqsicreatebroker** command, the local mqbrkrs group is granted access to internal queues whose names begin with the characters SYSTEM.BROKER.

Securing the broker registry

About this task

Broker operation depends on the information in the broker registry, which you must secure to guard against accidental corruption. The broker registry is stored in the Windows registry or the Linux or UNIX file system. Set your operating system security options so that only user IDs that are members of the group mqbrkrs can read from or write to *brokername*`/CurrentVersion` and all subkeys.

Related concepts:

"Security overview" on page 351
When you are designing a WebSphere Message Broker application it is important to consider the security measures that are needed to protect the information in the system.

"Authorization for configuration tasks" on page 353
Authorization is the process of granting or denying access to a system resource.

Related reference:

"Security requirements for Windows systems" on page 3651
Security requirements depend on the administrative task that you want to perform.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

Implementing SSL authentication

Use SSL authentication to enhance security in your broker environment.

About this task

The following topics contain instructions for implementing SSL authentication:

- "Setting up a public key infrastructure"
- "Listing SSL cipher suites" on page 511
- "Implementing SSL authentication on z/OS" on page 512
- "Configuring the broker to use SSL with JMS nodes" on page 530
- "Configuring SOAPInput and SOAPReply nodes to use SSL (HTTPS)" on page 532
- "Configuring SOAPRequest and SOAPAsyncRequest nodes to use SSL (HTTPS)" on page 533
- "Configuring HTTPInput and HTTPReply nodes to use SSL (HTTPS)" on page 535
- "Configuring an HTTPRequest node to use SSL (HTTPS)" on page 538
- "Enabling SSL on the WebSphere MQ Java Client" on page 540
- "Securing the connection to CICS Transaction Server for z/OS by using SSL" on page 547
- "Securing the connection to IMS by using SSL" on page 549
- "Configuring TCP/IP client nodes to use SSL" on page 551
- "Configuring TCP/IP server nodes to use SSL" on page 553

Related concepts:

"Security overview" on page 351
When you are designing a WebSphere Message Broker application it is important to consider the security measures that are needed to protect the information in the system.

Related reference:

"Security requirements for administrative tasks" on page 3644
You can configure access permissions to govern which users and groups can manipulate objects in the broker network. Security requirements for administrative tasks depend on the platform that you use.

Setting up a public key infrastructure

Configure keystores, truststores, passwords, and certificates to enable SSL communication, and Web Services Security.

Before you begin

Decide how you will use the public key infrastructure (PKI). You can configure keystores and truststores at either broker level (one keystore, one truststore, and one personal certificate for each broker) or at execution group level (one keystore, one truststore, and one personal certificate for each execution group). Execution groups that do not have PKI configured use the broker-level PKI configuration.

Encryption strength

The WebSphere Message Broker Java runtime environment (JRE) is provided with strong but limited strength encryption. If you cannot import keys into keystores,

limited strength encryption might be the cause. Either start ikeyman by using the **strmqikm** command, or download unrestricted jurisdiction policy files from IBM developer kits: Security information.

Important: Your country of origin might have restrictions on the import, possession, use, or re-export to another country, of encryption software. Before downloading or using the unrestricted policy files, you must check the laws of your country. Check its regulations and its policies concerning the import, possession, use, and re-export of encryption software, to determine whether it is permitted. Note that when applying a fix pack to an existing Message Broker installation, the JVM is overwritten, including any updated policy set files. These must be restored before you restart the broker.

WebSphere Message Broker currently supports up to 4096 bit keys. Larger keys require more CPU resources for encryption and decryption.

About this task

This topic uses the command line tool, **gsk7cmd**, to create and populate keystores and truststores. The **gsk7cmd** tool is a part of the Global Secure Toolkit, supplied with WebSphere MQ. Other options, supplied with the WebSphere Message Broker JVM, include:

- A command-line tool, **keytool**.
- A graphical tool, **iKeyman**.

To create the infrastructure, complete the following tasks:

1. "Creating a keystore file or a truststore"
2. "Creating a self-signed certificate for test use" on page 506
3. "Importing a certificate for production use" on page 506
4. "Viewing details of a certificate" on page 506
5. "Extracting a certificate" on page 507
6. "Adding a signer certificate to the truststore" on page 508
7. "Listing all certificates in a keystore" on page 509
8. "Configuring PKI at broker level" on page 509
9. "Configuring PKI at execution group level" on page 510

Creating a keystore file or a truststore:

The keystore file contains the personal certificate for the broker or for the execution group. You can have only one personal certificate in the keystore. You can store signer certificates in the same file, or create a separate file, known as a truststore.

Procedure

1. Set the JAVA_HOME environment variable, for example:

   ```
   set JAVA_HOME=C:\WMB70\7.0\jre15
   ```

2. Issue the following command:

   ```
   gsk7cmd -keydb -create
     -db keystore_name
     [-pw password]
     -type jks
   ```

   The *password* argument is optional. If you omit it, you are prompted to enter a password. For example:

```
gsk7cmd -keydb -create
  -db myBrokerKeystore.jks
  -type jks
A password is required to access this key database.
Please enter a password:
```

Creating a self-signed certificate for test use:

Use self-signed certificates only for testing SSL, not in production.

Procedure

Enter the following command:

```
gsk7cmd -cert -create
  -db keystore_name
  [-pw password]
  -label cert_label
  -dn "distinguished_name"
```

For example:

```
gsk7cmd -cert -create
  -db myBrokerKeystore.jks
  -label MyCert
  -dn "CN=MyBroker.Server,O=IBM,OU=ISSW,L=Hursley,C=GB"
A password is required to access this key database.
Please enter a password:
```

Importing a certificate for production use:

Import a personal certificate from a certificate authority for production use.

Procedure

Issue the following command:

```
gsk7cmd -cert -import
  -db pkcs12_file_name
  [-pw pkcs12_password]
  -label label
  -type pkcs12
  -target keystore_name
  [-target_pw keystore_password]
```

For example:

```
gsk7cmd -cert -import
  -db SOAPListenerCertificate.p12
  -label soaplistener
  -type pkcs12
  -target myBrokerKeystore.jks
  -target_pw  myBrokerKpass
A password is required to access this key database.
Please enter a password:
```

Viewing details of a certificate:
Procedure

Issue the following command:

```
gsk7cmd -cert -details
  -db keystore_name
  [-pw password]
  -label label
```

For example:

```
gsk7cmd -cert -details
  -db myKeyStore.jks
  -label MyCert
A password is required to access this key database.
Please enter a password:

Label: MyCert
Key Size: 1024
Version: X509 V3
Serial Number: 4A D7 39 1F
Issued By: MyBroker.Server
ISSW
IBM
Hursley, GB
Subject: MyBroker.Server
ISSW
IBM
Hursley, GB
Valid From: 15 October 2009 16:00:47 o'clock BST To: 15 October 2010 16:00:47 o'
clock BST
Fingerprint: 98:5D:C4:70:A0:28:84:72:FB:F6:3A:D2:D2:F5:EE:8D:30:33:87:82
Signature Algorithm: 1.2.840.113549.1.1.4
Trust Status: enabled
```

Extracting a certificate:

Generate a copy of a self-signed certificate that you can import as a trusted (or signer certificate) into a truststore file. Use this procedure only for testing, not production.

About this task

Certificates can be extracted in two formats:

- Base64-encoded ASCII data (`.arm`). This format is convenient for inclusion in XML messages, and transmission over the Internet.
- Binary DER data (`.der`).

Procedure

Issue the following command:

```
gsk7cmd -cert -extract
  -db keystore_name
  -pw keystore_passwd
  -label label
  -target file_name
  [-format ascii | binary]
```

For example:

```
gsk7cmd -cert -extract
  -db myBrokerKeystore.jks
  -pw myKeyPass
  -label MyCert
  -target MyCert.arm
  -format ascii
```

You can then view the certificate in a text editor, such as Notepad:

```
notepad MyCert.arm
-----BEGIN CERTIFICATE-----
MIICIzCCAYygAwIBAgIEStc5HzANBgkqhkiG9w0BAQQFADBWMQswCQYDVQQGEwJHQjEQMA4GA1UE
BxMHSHVyc2xleTEMMAoGA1UEChMDSUJNMQ0wCwYDVQQLEwRJU1NXMRgwFgYDVQQDEw9NeUJyb2tl
```

```
ci5TZXJ2ZXIwHhcNMDkxMDE1MTUwMDQ3WhcNMTAxMDE1MTUwMDQ3WjBWMQswCQYDVQQGEwJHQjEQ
MA4GA1UEBxMHSHVyc2xleTEMMAoGA1UEChMDSUJNMQ0wCwYDVQQLEwRJU1NXMRgwFgYDVQQDEw9N
eUJyb2tlci5TZXJ2ZXIwgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAMwkK5kFLwC29YsHLXlf
hd0CgqFeytHlI0sZesdi8hEPXKsOzs3OQta2b0GZyUbBkh4tNeUHNWE9o7Hx2/SfziPQRKUw908R
F/6FPaHGezRkkaLJGX3uEhjt/2+n5tOJGytnKWaWJTpzdmZ79c0XjFvO83q3yXPYjKzq8rS1iVBf
AgMBAAEwDQYJKoZIhvcNAQEEBQADgYEAQEjpvZkjRcg3AHqY4RWbSMtXVWFFyoHSbjymR8IdURoQ
DCGZ2jsv3kxQLADaCXOBYgohGJAHS7PzkQoHUCiHR0kusyuAt1MNYbhEcs+BYAzvsSz1ay4oiqCw
Qs3aeNLVOb9c1RyzbuKYZl0uX59GAfGVLvyk6vQ/g7wPVL4TVgc=
-----END CERTIFICATE-----
```

Adding a signer certificate to the truststore:

Add a signer certificate to the truststore of a broker or execution group.

About this task

The following steps show how to add an extracted certificate as signer certificate to the truststore file. Adding the broker self-signed certificate to a broker or execution group truststore enables request nodes (HTTP or SOAP) to send test messages to input nodes (HTTP or SOAP) when the flows are running on the broker or execution group.

Procedure

Issue the following command:

```
gsk7cmd -cert -add
  -db truststore_name
  [-pw password]
  -label label
  -file file_name
  -format [ascii | binary]
```

For example:

```
gsk7cmd -cert -add
  -db myBrokerTruststore.jks
  -label CACert
  -file TRUSTEDPublicCerticate.arm
  -format ascii
```

You can view details of the certificate:

```
gsk7cmd -cert -details -db myBrokerTruststore.jks -label CACert
A password is required to access this key database.
Please enter a password:

Label: CACert
Key Size: 1024
Version: X509 V3
Serial Number: 49 49 23 1B
Issued By: VSR1BK
ISSW
IBM
GB
Subject: VSR1BK
ISSW
IBM
GB
Valid From: 17 December 2008 16:04:43 o'clock GMT To: 17 December 2009 16:04:43
o'clock GMT
Fingerprint: CB:39:E7:D8:1D:C0:00:A1:3D:B1:97:69:7A:A7:77:19:6D:09:C2:A7
Signature Algorithm: 1.2.840.113549.1.1.4
Trust Status: enabled
```

Listing all certificates in a keystore:
Procedure

Issue the following command:

```
gsk7cmd -cert -list
  -db keystore_name
```

For example:

```
gsk7cmd -cert -list
  -db myBrokerKeystore.jks
A password is required to access this key database.
Please enter a password:

Certificates in database: myBrokerKeystore.jks
    verisign class 1 public primary certification authority - g3
    verisign class 4 public primary certification authority - g3
    verisign class 1 public primary certification authority - g2
    verisign class 4 public primary certification authority - g2
    verisign class 2 public primary certification authority
    entrust.net global client certification authority
    rsa secure server certification authority
    verisign class 2 public primary certification authority - g3
    verisign class 2 public primary certification authority - g2
    verisign class 3 secure server ca
    verisign class 3 public primary certification authority
    verisign class 3 public primary certification authority - g3
    verisign class 3 public primary certification authority - g2
    thawte premium server ca
    verisign class 1 public primary certification authority
    entrust.net global secure server certification authority
    thawte personal basic ca
    thawte personal premium ca
    thawte personal freemail ca
    verisign international server ca - class 3
    thawte server ca
    entrust.net certification authority (2048)
    cacert
    entrust.net client certification authority
    entrust.net secure server certification authority
    soaplistener
    mycert
```

Configuring PKI at broker level:

Define the broker registry properties that identify the location, name, and password of the keystore and truststore files.

About this task

1. Start the broker:

   ```
   mqsistart broker_name
   ```

2. Display the current settings of the broker registry properties:

   ```
   mqsireportproperties broker_name
     -o BrokerRegistry
     –r
   ```

3. Set the keystore property:

   ```
   mqsichangeproperties broker_name
     -o BrokerRegistry
     -n brokerKeystoreFile
     -v C:\WMB\MQSI\7.0\MyBrokerKeystore.jks
   ```

4. Set the truststore property:

```
mqsichangeproperties broker_name
  -o BrokerRegistry
  -n brokerTruststoreFile
  -v C:\WMB\MQSI\7.0\MyBrokerTruststore.jks
```

5. Stop the broker:

```
mqsistop broker_name
```

6. Set the password for the keystore:

```
mqsisetdbparms broker_name
  -n brokerKeystore::password
  -u ignore
  -p keystore_pass
```

7. Set the password for the truststore:

```
mqsisetdbparms broker_name
  -n brokerTruststore::password
  -u ignore
  -p truststore_pass
```

8. Start the broker:

```
mqsistart broker_name
```

9. Display and verify the broker registry properties:

```
mqsireportproperties broker_name
  -o BrokerRegistry –r
```

Configuring PKI at execution group level:

Define the ComIbmJVMManager properties for the required execution group to identify the location, name, and password of the keystore and truststore files.

About this task

1. Start the broker.

```
mqsistart broker_name
```

2. Display the current settings of the ComIbmJVMManager properties.

```
mqsireportproperties broker_name
  -e exec_grp_name
  -o ComIbmJVMManager -r
```

3. Set the keystore property.

```
mqsichangeproperties broker_name
  -e exec_grp_name
  -o ComIbmJVMManager
  -n keystoreFile
  -v C:\WMB\MQSI\7.0\MyBrokerexec_grp_name Keystore.jks
```

4. Set the keystore password key property. The value for this property is in the format *any_prefix_name*::password. This value is used to correlate the password defined in the **mqsisetdbparms** command.

```
mqsichangeproperties broker_name
  -e exec_grp_name
  -o ComIbmJVMManager
  -n keystorePass
  -v exec_grp_nameKeystore::password
```

5. Set the truststore property.

```
mqsichangeproperties broker_name
  -e exec_grp_name
  -o ComIbmJVMManager
  -n truststoreFile
  -v C:\WMB\MQSI\7.0\MyBrokerexec_grp_name Truststore.jks
```

6. Set the truststore password key property. The value for this property is in the format *any_prefix_name*::password. This value is used to correlate the password defined in the **mqsisetdbparms** command.

```
mqsichangeproperties broker_name
  -e exec_grp_name
  -o ComIbmJVMManager
  -n truststorePass
  -v exec_grp_nameTruststore::password
```

7. Stop the broker.

```
mqsistop broker_name
```

8. Set the password for the keystore.

```
mqsisetdbparms broker_name
  -n exec_grp_nameKeystore::password
  -u ignore
  -p keystore_pass
```

9. Set the password for the truststore.

```
mqsisetdbparms broker_name
  -n exec_grp_nameTruststore::password
  -u ignore
  -p truststore_pass
```

10. Start the broker.

```
mqsistart broker_name
```

11. Display and verify the ComIbmJVMManager properties.

```
mqsireportproperties broker_name
  -e exec_grp_name
  -o ComIbmJVMManager -r
```

For information about cipher-suite requirements (such as the cryptographic algorithm and corresponding key lengths), see the developerWorks article on the Java Secure Socket Extension (JSSE) IBMJSSE2 Provider.

Related tasks:

"Implementing SSL authentication" on page 504
Use SSL authentication to enhance security in your broker environment.

"Configuring the broker to use SSL with JMS nodes" on page 530
Configure your broker to work with a JMS provider that supports JMS clients that can connect by using the Secure Sockets Layer (SSL) protocol.

"Configuring HTTPInput and HTTPReply nodes to use SSL (HTTPS)" on page 535
Configure the HTTPInput and HTTPReply nodes to communicate with other applications that use HTTPS by creating a keystore file, configuring the broker or execution group to use SSL, and creating a message flow to process HTTPS requests.

"Configuring an HTTPRequest node to use SSL (HTTPS)" on page 538
Configure the HTTPRequest node to communicate with other applications that use HTTP over SSL.

Listing SSL cipher suites

You can view the available cipher suites in the WebSphere Message Broker Toolkit when you connect to a remote broker. You can also view a list of the cipher suites that are supported by WebSphere Message Broker.

About this task

A cipher suite is a collection of algorithms that are used to encrypt data. During SSL authentication, the client and server compare cipher suites and select the first

one that they have in common. If no suitable cipher suites exist, the server returns a handshake failure alert and closes the connection.

To use SSL encryption, you need a Java Secure Socket Extension (JSSE) provider. WebSphere Message Broker supports the ciphers that are provided by the JSSE provider. For more information about Java security, see IBM Java security web page.

You can view the available cipher suites in the WebSphere Message Broker Toolkit when you connect to a remote broker:

1. In the Brokers view, right-click **Brokers**, then click **Connect to a Remote Broker**.
2. Provide a queue manager name, host name, and port number, then click **Next**.
3. Click the arrow to the right of **Cipher suite**.

A list of the cipher suites that are available in the Toolkit is displayed.

For a list of all the cipher suites that are supported by WebSphere Message Broker, see Appendix A of the IBM JSSE2 Guide.

Related tasks:

"Implementing SSL authentication" on page 504
Use SSL authentication to enhance security in your broker environment.

"Connecting to a remote broker" on page 902
To administer your brokers by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer, you must connect to the broker.

"Connecting to a remote broker on z/OS in the WebSphere Message Broker Explorer" on page 904
To administer your brokers by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer, you must connect to the broker.

"Configuring TCP/IP client nodes to use SSL" on page 551
Configure a TCP/IP configuration to use SSL to secure connectivity to and from the TCPIP client nodes.

"Configuring TCP/IP server nodes to use SSL" on page 553
Configure a TCP/IP configuration to use SSL to secure connectivity to and from the TCPIP server nodes.

Related reference:

"SOAPRequest node" on page 4828
Use the SOAPRequest node to send a SOAP request to the remote Web service.

"SOAPAsyncRequest node" on page 4750
Use the SOAPAsyncRequest node with the SOAPAsyncResponse node to construct a pair of message flows that call a Web service asynchronously.

"HTTPRequest node" on page 4488
Use the HTTPRequest node to interact with a web service.

IBM Java security web page

IBM JSSE2 Guide

Implementing SSL authentication on z/OS

Use SSL authentication to enhance security in your broker environment.

About this task

Complete the following tasks to implement SSL authentication for brokers running on z/OS.

1. “Generate a broker certificate using RACF as the Certification Authority (z/OS)”
2. “Create and initialize the broker keystore and truststore (z/OS)” on page 514
3. “Configure WebSphere Message Broker on z/OS for SSL” on page 516

Alternatively, you can enable SSL for WebSphere Message Broker on z/OS by following the instructions in “Enabling SSL on z/OS by using AT-TLS” on page 517.

Generate a broker certificate using RACF as the Certification Authority (z/OS):

You can use RACF as the Certification Authority (CA) for internal certificates in your enterprise.

About this task

To generate broker certificates, take the following steps:

1. Create the RACF CA signer certificate. This self-signed certificate is used to sign any other personal certificates created or requested in RACF. This step is required once.
2. Export the RACF CA signer certificate in CERTDER format. This certificate must be extracted without private keys; CERTDER is a binary format that guarantees that no private keys are exported.
3. Create the broker personal certificate. A copy of the certificate and of the private keys is maintained in RACF for future reissue or validation. This certificate must be associated with the broker user ID. Create a personal certificate for each broker or execution group for which you want to enable SSL.
4. Export the broker personal certificate in PKCS12DER format. PKCS12DER is a password-protected, binary format that contains the broker certificate and its private keys. You will later import it into the broker keystore; see “Create and initialize the broker keystore and truststore (z/OS)” on page 514.

Example commands for each step are as follows:

Procedure

1. Create the RACF CA signer certificate. For example:

```
RACDCERT CERTAUTH GENCERT +
   SUBJECTSDN(CN('RACF Cert Authority') T('PROD') +
   OU('RACF Group') +
   O('IBM') +
   L('HURSLEY') SP('WINCHESTER') C('GB')) +
   KEYUSAGE(CERTSIGN) +
   WITHLABEL('RACFCA') +
   NOTAFTER(DATE(2020/01/30)) +
   SIZE(1024)
```

2. Export the RACFCA certificate in CERTDER format. For example:

```
RACDCERT CERTAUTH EXPORT(LABEL('RACFCA')) +
   DSN('CSQP.CSQPBRK.CACERT.DER') FORMAT(CERTDER)

OPUT 'CSQP.CSQPBRK.CACERT.DER' +
    '/u/CSQPBRK/ssl/csqpbrk.ca.der' +
   BINARY CONVERT(NO)
```

   The OPUT command is optional. It is used to copy the certificate into a HFS file before FTP to another server.

3. Create the broker personal certificate. For example:

```
RACDCERT ID(CSQPBRK) +
   GENCERT SUBJECTSDN(CN('BROKER.HTTP.CSQPBRK') T('PROD') +
   OU('ISSW') O('IBM') +
   L('HURSLEY') SP('WINCHESTER') C('GB')) +
   WITHLABEL('CSQPBRKCERT') SIZE(1024) +
   SIGNWITH(CERTAUTH LABEL('RACFCA')) +
   KEYUSAGE(HANDSHAKE DATAENCRYPT DOCSIGN) +
   NOTAFTER(DATE(2020/01/30))
```

4. Export the broker certificate in PKCS12 format. For example:

```
RACDCERT ID(CSQPBRK) EXPORT(LABEL('CSQPBRKCERT')) +
     DSN('CSQP.CSQPBRK.PERSCERT.P12') +
     FORMAT(PKCS12DER) PASSWORD('changeit')

OPUT 'CSQP.CSQPBRK.PERSCERT.P12' +
     '/u/CSQPBRK/ssl/csqpbrk.pers.p12' +
     BINARY CONVERT(NO)
```

What to do next

What to do next: Create the broker keystore and import the personal certificate and RACF CA signer certificates.

Related tasks:

"Implementing SSL authentication on z/OS" on page 512
Use SSL authentication to enhance security in your broker environment.

"Create and initialize the broker keystore and truststore (z/OS)"
Create a keystore and import your personal certificate and signer certificates.

"Configure WebSphere Message Broker on z/OS for SSL" on page 516
Define the location of the keystore and truststore, set passwords, and enable SSL.

Create and initialize the broker keystore and truststore (z/OS):

Create a keystore and import your personal certificate and signer certificates.

Before you begin

Before you start:

- Create the necessary certificates.

Note: Due to export restrictions, the IBM JDKs ship with a set of restricted policy files that limit the size of the cryptographic keys that are supported. To overcome these restrictions, use the unrestricted policy files in the `$JAVA_HOME/lib/security` directory:

- local_policy.jar
- US_export_policy.jar

The unrestricted policy files are the same for the IBM JDK 1.4.2, IBM JDK 5, and IBM JDK 6. These files are in the `JAVA_HOME/demo/jce/policy-files/unrestricted` directory.

About this task

This topic describes how to use the same file as keystore and truststore. To specify different files, complete the process twice:

- Do not import signer certificates into the keystore.

- Do not import personal certificates into the truststore.

The tasks use **keytool** to create the keystore. An alternative is the **ikeyman** graphical tool, which requires an X Window System.

The following are the steps required to create and initialize the broker keystore:

1. Create the keystore. **keytool** requires a dummy key to be created to force the creation of the keystore file. The dummy key is deleted after the keystore is created.
2. Import the CA signer certificate or certificates. These are certificates that have signed certificates of client applications that connect to the WebSphere Message Broker and that are accepted as trusted applications.

Example commands for each step are as follows:

Procedure

1. Create the JKS keystore. For example:

```
/usr/lpp/java/J6.0/bin/keytool -genkey
     -alias DUMMY
     -keystore  /u/CSQPBRK/ssl/csqpbrkKeystore.jks
     -storetype jks
     -dname  "CN=DUMMY,OU=BROKER,O=IBM,L=Hursley,C=GB"
     -storepass changeit
     -keypass changeit
```

2. Delete the dummy key. For example:

```
/usr/lpp/java/J6.0/bin/keytool -delete
     -alias DUMMY
     -keystore  /u/CSQPBRK/ssl/csqpbrkKeystore.jks
     -storepass changeit
```

3. Optional: Import the CA signer certificates. Omit this step if you require separate files for a keystore and truststore, and are creating a keystore. For example:

```
/usr/lpp/java/J6.0/bin/keytool -import
     -keystore /u/CSQPBRK/ssl/csqpbrkKeystore.jks
     -storepass changeit
     -alias RACFCA
     -file /u/CSQPBRK/ssl/csqpbrk.ca.der -v
```

4. Optional: Import the broker personal certificate. Omit this step if you require separate files for a keystore and truststore, and are creating a truststore. For example:

```
/usr/lpp/java/J6.0/bin/keytool -import
  -keystore /u/CSQPBRK/ssl/csqbrkKeystore.jks
  -storepass changeit
  -alias CSQPBRK
  -file /u/CSQPBRK/ssl/csqpbrk.pers.p12
  -v
  -pkcs12
  -keypass changeit
  -noprompt
```

5. List the contents of the broker keystore. For example:

```
/usr/lpp/java/J6.0/bin/keytool -list
   -keystore /u/CSQPBRK/ssl/csqbrkKeystore.jks
   -storepass changeit
```

What to do next

What to do next: "Configure WebSphere Message Broker on z/OS for SSL."

Related tasks:

"Implementing SSL authentication on z/OS" on page 512
Use SSL authentication to enhance security in your broker environment.

"Generate a broker certificate using RACF as the Certification Authority (z/OS)" on page 513
You can use RACF as the Certification Authority (CA) for internal certificates in your enterprise.

"Configure WebSphere Message Broker on z/OS for SSL"
Define the location of the keystore and truststore, set passwords, and enable SSL.

Configure WebSphere Message Broker on z/OS for SSL:

Define the location of the keystore and truststore, set passwords, and enable SSL.

Before you begin

Before you start: complete the following tasks:

- "Generate a broker certificate using RACF as the Certification Authority (z/OS)" on page 513
- "Create and initialize the broker keystore and truststore (z/OS)" on page 514

About this task

The process is essentially the same as on Windows and UNIX. This topic describes how to enable SSL at broker level; it can also be done at execution group level for the SOAP nodes. See "Configuring SOAPInput and SOAPReply nodes to use SSL (HTTPS)" on page 532 and "Configuring SOAPRequest and SOAPAsyncRequest nodes to use SSL (HTTPS)" on page 533 for a description of the process on distributed platforms.

To execute the following commands, you can run the BIPCHPR job in the broker component library.

Procedure

1. Define the location of the keystore. This example shows how to define a keystore at broker level. For example:

```
BPXBATSL PGM -
  /usr/lpp/mqsi/V7R0M0/bin/-
mqsichangeproperties -
  CSQPBRK -
  -o BrokerRegistry -
  -n brokerKeystoreFile -
  -v /u/CSQPBRK/ssl/csqbrkKeystore.jks
```

2. Define the location of the truststore. For example:

```
BPXBATSL PGM -
  /usr/lpp/mqsi/V7R0M0/bin/-
mqsichangeproperties -
  CSQPBRK -
  -o BrokerRegistry -
  -n brokerTruststoreFile -
  -v /u/CSQPBRK/ssl/csqbrkKeystore.jks
```

3. Enable the HTTPS Connector. For example:

```
BPXBATSL PGM -
  /usr/lpp/mqsi/V7R0M0/bin/-
mqsichangeproperties -
  CSQPBRK –
  -b httplistener -
  -o HTTPListener -
  -n enableSSLConnector -
  -v true
```

4. Optional: Enable client authentication. For example:

```
BPXBATSL PGM -
  /usr/lpp/mqsi/V7R0M0/bin/-
mqsichangeproperties -
  CSQPBRK –
  -b httplistener -
  -o HTTPSConnector -
  -n clientAuth -
  -v true
```

5. Stop the broker. You must stop the broker before you can define passwords.
6. Define the keystore password. For example:

```
BPXBATSL PGM -
  /usr/lpp/mqsi/V7R0M0/bin/-
mqsisetdbparms -
  CSQPBRK –
  -n brokerKeystore::password –
  -u ignore -
  -p changeit
```

7. Define the truststore password. For example:

```
BPXBATSL PGM -
  /usr/lpp/mqsi/V7R0M0/bin/-
mqsisetdbparms -
  CSQPBRK –
  -n brokerTruststore::password –
  -u ignore -
  -p changeit
```

8. Start the broker.
9. Verify and test your configuration.

Related tasks:

"Implementing SSL authentication on z/OS" on page 512
Use SSL authentication to enhance security in your broker environment.

"Generate a broker certificate using RACF as the Certification Authority (z/OS)" on page 513
You can use RACF as the Certification Authority (CA) for internal certificates in your enterprise.

"Create and initialize the broker keystore and truststore (z/OS)" on page 514
Create a keystore and import your personal certificate and signer certificates.

"Starting and stopping a broker on z/OS" on page 924
Run the appropriate command from SDSF to start or stop a broker.

Enabling SSL on z/OS by using AT-TLS:

You can use Application Transparent Transport Layer Security (AT-TLS) to provide Secure Sockets Layer (SSL) services on behalf of WebSphere Message Broker on z/OS. AT-TLS is part of z/OS Communication Server.

About this task

You can enable SSL on z/OS by following the instructions in "Implementing SSL authentication on z/OS" on page 512. This topic describes an alternative method that uses AT-TLS to enable SSL without the need to complete configuration steps in WebSphere Message Broker. AT-TLS provides the following benefits when using SSL/TLS protocols with WebSphere Message Broker on z/OS:

- AT-TLS uses RACF key rings and certificates.
- The Policy Agent (PAGENT) manages the rules and policies that define how SSL is used to connect to WebSphere Message Broker.
- PAGENT can distribute the rules and policies in a z/OS SYSPLEX environment.
- The WebSphere Message Broker administrator does not have to set any WebSphere Message Broker properties for SSL.
- HTTP or SOAP nodes in message flows can have standard HTTP settings (no SSL/HTTPS).

To configure AT-TLS in your z/OS environment for WebSphere Message Broker, complete the following steps.

Procedure

1. Create a RACF key ring by following the instructions in "Creating a RACF key ring" on page 520.
2. Configure and activate PAGENT by following the instructions in "Configuring and activating the policy agent (PAGENT)" on page 522.
3. Define and install AT-TLS policies for WebSphere Message Broker by following the instructions in "Defining and installing AT-TLS policies" on page 524.
4. Test and verify AT-TLS by using WebSphere Message Broker, as described in "Testing and verifying AT-TLS" on page 527.

Related concepts:

"Application Transparent Transport Layer Security" on page 519
AT-TLS is a service provided by the z/OS Communication Server Policy Agent (PAGENT) and the TCP/IP stack. The AT-TLS service manages SSL connections on behalf of applications that are running on z/OS. The z/OS applications are unaware that SSL is used in the connection with partner applications.

Related tasks:

"Creating a RACF key ring" on page 520
To create a RACF key ring, you must first generate a RACF CA certificate and a personal certificate for WebSphere Message Broker, then connect the certificates to the key ring.

"Configuring and activating the policy agent (PAGENT)" on page 522
Configure PAGENT by updating the TCP/IP profile, granting RACF permission to TCP/IP resources, preparing the PAGENT startup JCL, and activating syslogd.

"Defining and installing AT-TLS policies" on page 524
Define and install AT-TLS policies by using the IBM Configuration Assistant for z/OS Communications Server.

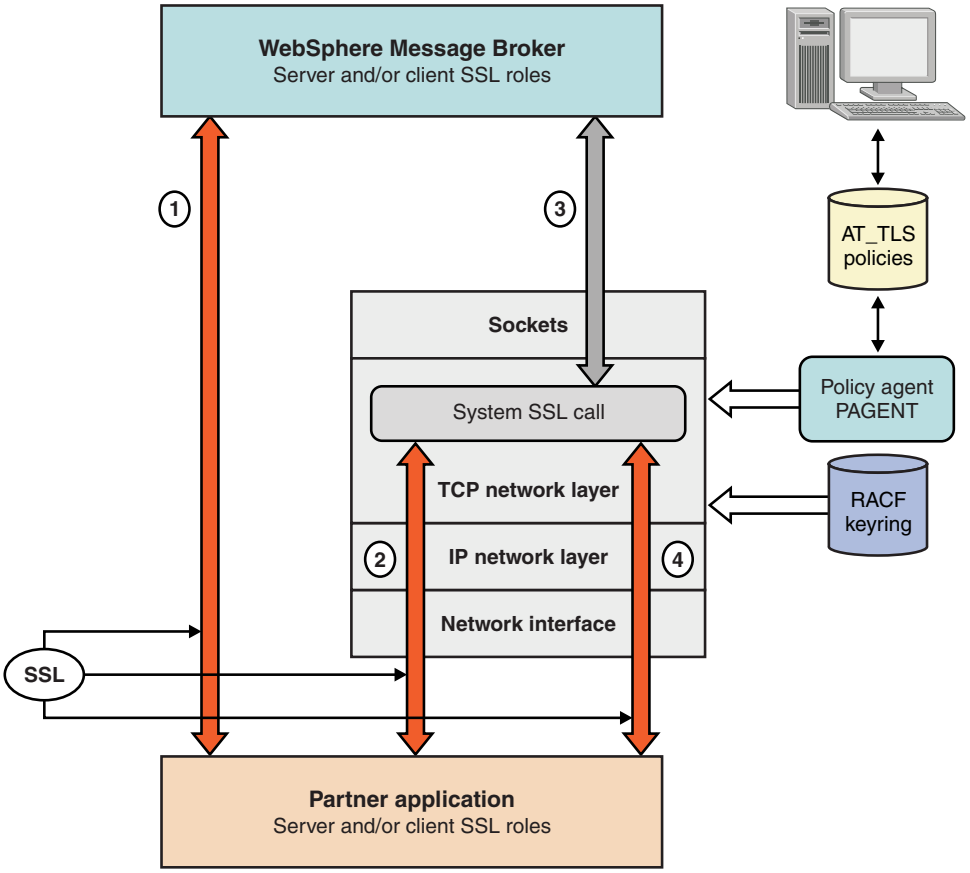
"Testing and verifying AT-TLS" on page 527
Test and verify AT-TLS by using the SOAP Nodes sample.

"Diagnosing problems with PAGENT and AT-TLS" on page 529
To help with problem determination with PAGENT and AT-TLS, activate tracing and logging.

Application Transparent Transport Layer Security:

AT-TLS is a service provided by the z/OS Communication Server Policy Agent (PAGENT) and the TCP/IP stack. The AT-TLS service manages SSL connections on behalf of applications that are running on z/OS. The z/OS applications are unaware that SSL is used in the connection with partner applications.

The following diagram shows how AT-TLS works. The numbers in the diagram represent the steps that follow the diagram.



1. Step 1 represents an SSL connection when AT-TLS is not used, which requires that WebSphere Message Broker and the partner application are both enabled for SSL.
2. The SSL handshake is managed by AT-TLS in the TCP layer.
3. Inbound or outbound application data is received or sent in the clear by WebSphere Message Broker. The TCP layer validates and decrypts inbound data from partner applications, or encrypts outbound data to partner applications.
4. Inbound or outbound application data is protected by SSL.

AT-TLS components

RACF key ring
The RACF key ring contains the WebSphere Message Broker personal certificate and the partner application signer certificate.

AT-TLS policies
This file contains the rules and policies that control the SSL connections

that are managed by AT-TLS. These policies are created by the network administrator, and are checked and enforced by the TCP network layer of the TCP/IP stack.

Policy Agent
This component manages and distributes network policies, including AT-TLS policies, to the TCP/IP stack or stacks. The policy agent is also called PAGENT. For AT-TLS to function successfully, PAGENT must be configured correctly and operational.

TCP/IP stack
The TCP/IP stack is the component that implements the AT-TLS services. The TCP network layer is where SSL connections are intercepted, the SSL handshake is performed, and data is decrypted and encrypted. The TCP/IP stack uses RACF services to validate and accept certificates that are presented by the partner application during the handshake. RACF retrieves the WebSphere Message Broker personal certificate from the key ring.

Related tasks:

"Enabling SSL on z/OS by using AT-TLS" on page 517
You can use Application Transparent Transport Layer Security (AT-TLS) to provide Secure Sockets Layer (SSL) services on behalf of WebSphere Message Broker on z/OS. AT-TLS is part of z/OS Communication Server.

"Creating a RACF key ring"
To create a RACF key ring, you must first generate a RACF CA certificate and a personal certificate for WebSphere Message Broker, then connect the certificates to the key ring.

"Configuring and activating the policy agent (PAGENT)" on page 522
Configure PAGENT by updating the TCP/IP profile, granting RACF permission to TCP/IP resources, preparing the PAGENT startup JCL, and activating syslogd.

"Defining and installing AT-TLS policies" on page 524
Define and install AT-TLS policies by using the IBM Configuration Assistant for z/OS Communications Server.

"Testing and verifying AT-TLS" on page 527
Test and verify AT-TLS by using the SOAP Nodes sample.

"Diagnosing problems with PAGENT and AT-TLS" on page 529
To help with problem determination with PAGENT and AT-TLS, activate tracing and logging.

Creating a RACF key ring:

To create a RACF key ring, you must first generate a RACF CA certificate and a personal certificate for WebSphere Message Broker, then connect the certificates to the key ring.

About this task

Each RACF key ring has its own name up to 237 characters long and is associated with a user ID. A RACF key ring is connected to a set of personal certificates and trusted certificates that are stored in the RACF database. The RACF command **RACDCERT** is used to create and delete key rings and to connect or disconnect certificates to the key rings. RACF key rings are also called System Authorization Facility (SAF) key rings. SAF is an open standard to access security services.

To create a RACF key ring to be used by AT-TLS on behalf of WebSphere Message Broker, complete the following steps.

Procedure

1. Generate a RACF certificate authority (CA) certificate.

   You can use RACF as a CA to generate and sign personal certificates for their internal systems or applications. This certificate must be created once, and it is used to sign every personal certificate that is generated by RACF. The following example shows how to use a RACF command to generate a RACF CA certificate.

   ```
   RACDCERT CERTAUTH GENCERT +
     SUBJECTSDN(CN('MQRootCA') +
     OU('ISSW') +
     O('IBM') +
     L('HURSLEY') SP('WINCHESTER') C('GB')) +
     KEYUSAGE(HANDSHAKE DATAENCRYPT DOCSIGN CERTSIGN) +
     WITHLABEL('MQRootCA') +
     NOTAFTER(DATE(2020/01/30)) +
     SIZE(1024)
   ```

2. Generate a personal certificate for WebSphere Message Broker.

   This certificate identifies a specific instance of WebSphere Message Broker. This certificate is presented to the partner application during the SSL handshake. This certificate must be associated with the user ID under which WebSphere Message Broker is running. The following example shows how to use a RACF command to generate the personal certificate for a broker called WI02BRK that is running under user ID WI02USR.

   ```
   RACDCERT ID(WI02USR) +
      GENCERT SUBJECTSDN(CN('WI02BRK') +
      OU('ISSW') O('IBM') +
      L('HURSLEY') SP('WINCHESTER') C('GB')) +
      WITHLABEL('WI02BRK') SIZE(1024) +
      SIGNWITH(CERTAUTH LABEL('MQRootCA')) +
      KEYUSAGE(HANDSHAKE DATAENCRYPT DOCSIGN) +
      NOTAFTER(DATE(2012/01/30))
   ```

3. Create a RACF key ring and connect the certificates to the key ring.

   The RACF key ring must be associated with a user ID (in this case, the WebSphere Message Broker user ID). The key ring must have a name (in this case, the same name as the broker), and the WebSphere Message Broker personal certificate must be connected to the key ring. The following example shows how to use a RACF command to create a key ring and connect the WebSphere Message Broker personal certificate.

   ```
   RACDCERT ID(WI02USR) ADDRING(WI02BRK)
   RACDCERT ID(WI02USR) +
    CONNECT(ID(WI02USR) LABEL('WI02BRK') RING(WI02BRK))
   RACDCERT ID(WI02USR) LISTRING(WI02BRK)
   ```

   For RACF to validate a partner application certificate, you must import the signer certificate of the CA that generated and signed the personal certificate of the partner application. Typically, this certificate is extracted from the partner application keystore, transferred to z/OS as a data set (WI02USR.VSR1BK.DER), imported to RACF, and connected to the RACF key ring as signer (trusted) certificate. The following example shows how to use a RACF command to add a signer certificate to RACF and connect it to the RACF key ring.

```
RACDCERT CERTAUTH ADD('WI02USR.VSR1BK.DER') +
WITHLABEL('VSR1BK') TRUST
RACDCERT CERTAUTH LIST(LABEL('VSR1BK')
RACDCERT ID(WI02USR) +
 CONNECT(CERTAUTH LABEL('VSR1BK') RING(WI02BRK))
RACDCERT ID(WI02USR) LISTRING(WI02BRK)
```

What to do next

Next: Configure and activate the policy agent by following the instructions in "Configuring and activating the policy agent (PAGENT)."

Related concepts:

"Application Transparent Transport Layer Security" on page 519
AT-TLS is a service provided by the z/OS Communication Server Policy Agent (PAGENT) and the TCP/IP stack. The AT-TLS service manages SSL connections on behalf of applications that are running on z/OS. The z/OS applications are unaware that SSL is used in the connection with partner applications.

Related tasks:

"Enabling SSL on z/OS by using AT-TLS" on page 517
You can use Application Transparent Transport Layer Security (AT-TLS) to provide Secure Sockets Layer (SSL) services on behalf of WebSphere Message Broker on z/OS. AT-TLS is part of z/OS Communication Server.

"Configuring and activating the policy agent (PAGENT)"
Configure PAGENT by updating the TCP/IP profile, granting RACF permission to TCP/IP resources, preparing the PAGENT startup JCL, and activating syslogd.

"Defining and installing AT-TLS policies" on page 524
Define and install AT-TLS policies by using the IBM Configuration Assistant for z/OS Communications Server.

"Testing and verifying AT-TLS" on page 527
Test and verify AT-TLS by using the SOAP Nodes sample.

"Diagnosing problems with PAGENT and AT-TLS" on page 529
To help with problem determination with PAGENT and AT-TLS, activate tracing and logging.

Configuring and activating the policy agent (PAGENT):

Configure PAGENT by updating the TCP/IP profile, granting RACF permission to TCP/IP resources, preparing the PAGENT startup JCL, and activating syslogd.

Before you begin

Before you start:

- Create a RACF key ring by following the instructions in "Creating a RACF key ring" on page 520.

About this task

To enable PAGENT for AT-TLS, complete the following steps. For a more detailed description of how to install and configure PAGENT, see the *Policy-based networking* chapter of the z/OS Communications Server IP Configuration Guide on the z/OS library web page.

Procedure

1. Update the TCP/IP profile.

   You must make two changes to the TCP/IP profile to enable AT-TLS:

   - Add the statement `TCPCONFIG TTLS` to activate the functionality of AT-TLS inside the TCP/IP stack.
   - Add PAGENT to the AUTOLOG list.

2. Grant RACF permissions to TCP/IP resources.

   Users require permissions to the following resources as part of activating PAGENT:

   a. Define PAGENT as a started task with its own user ID.
   b. The `EZB.INITSTACK.sysname.tcpprocname` resource profile controls which users can have access to the TCP/IP stack before PAGENT is active. Give READ access to all users who do not require PAGENT policies to access the TCP/IP stack; for example, PAGENT, NETVIEW, DB2, and so on.
   c. The `EZB.PAGENT.sysname.tcpprocname.*` resource controls which users can start, stop, and refresh PAGENT. Give READ access to the users who are allowed to run the TSO/Unix commands **`Pagent`** or **`pasearch`**.
   d. The user ID of PAGENT must have READ access to the BPX.DAEMON facility.

   For more detailed information about the RACF permissions, check the sample `EZARACF` in the `TCPIP.SEZAINST` library.

3. Prepare the PAGENT startup JCL.

   a. Copy the sample JCL PAGENT in the `TCPIP.SEZAINST` library to the system procedure library (for example, `SYS1.PROCLIB`).
   b. Edit the JCL according to your installation standards. Specify the location of the PAGENT configuration file (for example, `/etc/pagent/pagent.config`). You can specify the location and name of the configuration file by setting the environment variable `PAGENT_CONFIG_FILE=/etc/pagent/pagent.config`. The environment variables for the TCP/IP stack are usually specified in a member (for example, ENVVARS) of the TCP/IP parameters library (for example, `TCPIP.PARMS`). The PAGENT JCL has ddname STDENV that points to the member with the environment variables definitions.

      The PAGENT configuration file (`/etc/pagent/pagent.config`) specifies the location and name of the PAGENT stack-specific configuration file by using the statement `TcpImage: TcpImage TCPIP /etc/pagent/TCPIP.image FLUSH NOPURGE 1800`.

      The stack-specific configuration file (`/etc/pagent/TCPIP.image`) specifies the location and name of the AT-TLS policies file by using the statement `TTLSConfig: TTLSConfig /etc/pagent/TCPIP_TTLS.policy`.

4. Activate the system log daemon (syslogd).

   Syslogd acts as the central message logging facility for PAGENT and AT-TLS. Syslogd is not specific to the policy infrastructure, but the policy infrastructure depends on syslogd to provide a central logging facility to maintain an audit trail. If you do not start syslogd, messages are lost. Start one syslog daemon per LPAR.

What to do next

Define and install AT-TLS policies for WebSphere Message Broker by following the instructions in “Defining and installing AT-TLS policies” on page 524.

Related concepts:

"Application Transparent Transport Layer Security" on page 519
AT-TLS is a service provided by the z/OS Communication Server Policy Agent (PAGENT) and the TCP/IP stack. The AT-TLS service manages SSL connections on behalf of applications that are running on z/OS. The z/OS applications are unaware that SSL is used in the connection with partner applications.

Related tasks:

"Enabling SSL on z/OS by using AT-TLS" on page 517
You can use Application Transparent Transport Layer Security (AT-TLS) to provide Secure Sockets Layer (SSL) services on behalf of WebSphere Message Broker on z/OS. AT-TLS is part of z/OS Communication Server.

"Creating a RACF key ring" on page 520
To create a RACF key ring, you must first generate a RACF CA certificate and a personal certificate for WebSphere Message Broker, then connect the certificates to the key ring.

"Defining and installing AT-TLS policies"
Define and install AT-TLS policies by using the IBM Configuration Assistant for z/OS Communications Server.

"Testing and verifying AT-TLS" on page 527
Test and verify AT-TLS by using the SOAP Nodes sample.

"Diagnosing problems with PAGENT and AT-TLS" on page 529
To help with problem determination with PAGENT and AT-TLS, activate tracing and logging.

Defining and installing AT-TLS policies:

Define and install AT-TLS policies by using the IBM Configuration Assistant for z/OS Communications Server.

Before you begin

Before you start:

- Configure and activate PAGENT by following the instructions in "Configuring and activating the policy agent (PAGENT)" on page 522.

About this task

You can create the AT-TLS policies by using the IBM Configuration Assistant for z/OS Communications Server, a Java application that you can download from IBM. Complete the following steps to define the policies required to enable SSL support on behalf of WebSphere Message Broker for z/OS running SOAPInput and SOAPRequest nodes:

Procedure

1. Start the configuration assistant by clicking **Start** > **All Programs** > **IBM Programs** > **IBM Configuration Assistant for z/OS** > **Configuration Assistant V1R10**.
2. Click **Add a New z/OS Image**, enter the name of your z/OS image (LPAR) and a description, then click **OK**.
3. In the Configuration Assistant Navigation pane, select the image that you added in step 2, click **Add New TCP/IP Stack**, enter the stack name and description, then click **OK**.
4. In the Configuration Assistant Navigation pane, select the stack that you added in step 3, select **AT-TLS** from the list of technologies, then click **Enable**.

5. Click **Configure**.
6. Click **Add**. The Connectivity Rule wizard opens. Click **Next**
7. Identify the data endpoints by completing the following fields. A generic rule facilitates testing, but can be made more specific later.
   a. In the Local data endpoint field, select **ALL_IP_Addresses**.
   b. In the Remote data endpoint field, select **ALL_IP_Addresses**.
   c. In the Connectivity Rule Name field, enter a suffix for the name of the rules, then click **Next**.
8. Select a requirement map by clicking **Add**. The map is used to match the type of IP traffic with the security level to be implemented by AT-TLS.
9. Enter a name and description for the requirement map, then click **Work with Traffic Descriptors**. Two traffic descriptors are required: one for the inbound SOAP requests (WebSphere Message Broker is the server), and another for the outbound SOAP requests (WebSphere Message Broker is the client).
10. Create an inbound traffic descriptor by clicking **Add** , enter a name and description, then click **OK**.
11. Enter details about the inbound traffic descriptor:
    a. For the local port, select **Single port** and set the port number to **7800** (the port on which the SOAPInput node normally listens).
    b. For the remote port, select **All ports**.
    c. Set the Indicate the TCP connect direction field to **Inbound only**.
    d. In the Jobname field, enter an asterisk (*).
    e. In the User ID field, enter an asterisk (*).
    f. Select **Use the following key ring database**.
    g. Select **Key ring is in SAF produce (such as RACF)**, then enter the name of the key ring.
    h. Set the AT-TLS handshake role to **Server**, then click **AT-TLS Advanced**.
    i. Enter the label of the WebSphere Message Broker personal certificate, then click **OK**.
12. Click **OK** to save the traffic details for inbound SOAP traffic, then click **OK** to create the traffic descriptor for inbound SOAP.
13. Create an outbound traffic descriptor by clicking **Add**, add a name and description, then click **OK** .
14. Enter details about the outbound traffic descriptor:
    a. For the local port, select **All ports**.
    b. For the remote port, select **Single port** and set the port number to **7843**.
    c. Set the Indicate the TCP connect direction to **Outbound only**.
    d. In the Jobname field, enter an asterisk (*).
    e. In the User ID field, enter an asterisk (*).
    f. Select **Use the following key ring database**.
    g. Select **Key ring is in SAF produce (such as RACF)**, then enter the name of the key ring.
    h. Set the AT-TLS handshake role to **Client**, then click **AT-TLS Advanced**.
    i. Enter the label of the WebSphere Message Broker personal certificate, then click **OK**.
15. Click **OK** to save the traffic details for outbound SOAP traffic, then click **OK** to create the traffic descriptor for outbound SOAP.
16. Click **Close**.

17. To create a security level for WebSphere Message Broker, click **Work with Security Levels**, then click **Add**.
    a. On the Name and Type tab, enter a name and description.
    b. On the Ciphers tab, select **Use TLS V1**, **Use SSL V3**, and **Use System SSL defaults**, then click **OK**.
18. To add traffic descriptors to the requirement map, select **SOAP_Server** and **SOAP_Client** from the Objects list, then click **Add**.
19. For each traffic descriptor, select the AT-TLS security level that you created in step 17, then click **OK**.
20. Click **Next** and set the appropriate Optional Connectivity Rule Settings, which are used to set tracing levels, tuning parameters, and timings when the rule is in effect..
21. Click **Finish**.
22. To save changes to the AT-TLS rules, click **Apply changes**, then click **Main perspective**.
23. To install the AT-TLS policy, select **AT-TLS technology**, click **Install**, then click **FTP** to send the policy rules to the LPAR.
24. Specify the FTP parameters:
    a. Enter the LPAR host name and set the port number to 21.
    b. Enter your user ID and password.
    c. Enter the AT-TLS policy file location and name (for example, `/etc/pagent/TCPIP_TTLS.policy`.
    d. Select **Default transfer mode**.
    e. Click **Send**, wait for file transfer to complete, then check that the transfer was successful.
    f. Click **Close**.
    g. After the file transfer, refresh or restart PAGENT.

    The AT-TLS policies have been created and deployed.

What to do next

Next: Test and verify AT-TLS for WebSphere Message Broker by following the instructions in "Testing and verifying AT-TLS" on page 527.

Related concepts:

"Application Transparent Transport Layer Security" on page 519
AT-TLS is a service provided by the z/OS Communication Server Policy Agent (PAGENT) and the TCP/IP stack. The AT-TLS service manages SSL connections on behalf of applications that are running on z/OS. The z/OS applications are unaware that SSL is used in the connection with partner applications.

Related tasks:

"Enabling SSL on z/OS by using AT-TLS" on page 517
You can use Application Transparent Transport Layer Security (AT-TLS) to provide Secure Sockets Layer (SSL) services on behalf of WebSphere Message Broker on z/OS. AT-TLS is part of z/OS Communication Server.

"Creating a RACF key ring" on page 520
To create a RACF key ring, you must first generate a RACF CA certificate and a personal certificate for WebSphere Message Broker, then connect the certificates to the key ring.

"Configuring and activating the policy agent (PAGENT)" on page 522
Configure PAGENT by updating the TCP/IP profile, granting RACF permission to

TCP/IP resources, preparing the PAGENT startup JCL, and activating syslogd.

"Testing and verifying AT-TLS"
Test and verify AT-TLS by using the SOAP Nodes sample.

"Diagnosing problems with PAGENT and AT-TLS" on page 529
To help with problem determination with PAGENT and AT-TLS, activate tracing and logging.

Testing and verifying AT-TLS:

Test and verify AT-TLS by using the SOAP Nodes sample.

Before you begin

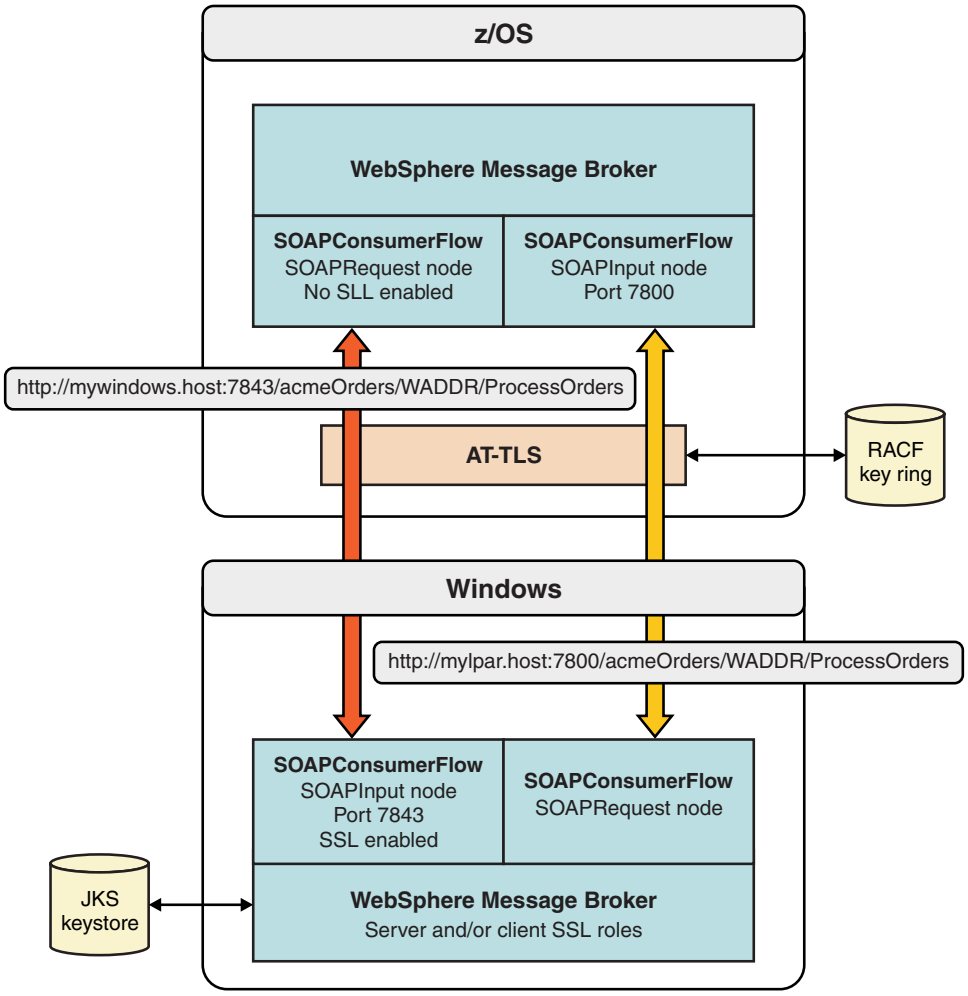
Before you start:

Complete the following tasks:

- "Creating a RACF key ring" on page 520
- "Configuring and activating the policy agent (PAGENT)" on page 522
- "Defining and installing AT-TLS policies" on page 524

About this task

Use the following scenario to test and verify AT-TLS with WebSphere Message Broker:



This scenario has the following components:

- One instance of WebSphere Message Broker is running on z/OS.
- Another instance of WebSphere Message Broker is running on Windows.
- SOAP node sample flows from the WebSphere Message Broker Toolkit samples gallery are deployed to the two WebSphere Message Broker instances.
- The broker instance running on z/OS is not enabled to use SSL. The SSL services are provided by AT-TLS.
- AT-TLS policies are active and PAGENT is running.
- The broker instance running on Windows is enabled to use SSL, and it has keystores defined with personal and signer certificates loaded.

The flows receive or send SOAP inbound and outbound requests between the two instances of WebSphere Message Broker. The SOAP consumer flow in one broker sends requests to the SOAP provider flow in the other broker. The SOAP nodes on the Windows broker are SSL enabled. The scenario shows the URLs that are configured on the SOAPRequest node on each broker.

For more information about how to configure, deploy, and run the SOAP Nodes sample, see “Samples” on page 98.

Related concepts:

"Application Transparent Transport Layer Security" on page 519
AT-TLS is a service provided by the z/OS Communication Server Policy Agent (PAGENT) and the TCP/IP stack. The AT-TLS service manages SSL connections on behalf of applications that are running on z/OS. The z/OS applications are unaware that SSL is used in the connection with partner applications.

Related tasks:

"Enabling SSL on z/OS by using AT-TLS" on page 517
You can use Application Transparent Transport Layer Security (AT-TLS) to provide Secure Sockets Layer (SSL) services on behalf of WebSphere Message Broker on z/OS. AT-TLS is part of z/OS Communication Server.

"Creating a RACF key ring" on page 520
To create a RACF key ring, you must first generate a RACF CA certificate and a personal certificate for WebSphere Message Broker, then connect the certificates to the key ring.

"Configuring and activating the policy agent (PAGENT)" on page 522
Configure PAGENT by updating the TCP/IP profile, granting RACF permission to TCP/IP resources, preparing the PAGENT startup JCL, and activating syslogd.

"Defining and installing AT-TLS policies" on page 524
Define and install AT-TLS policies by using the IBM Configuration Assistant for z/OS Communications Server.

"Diagnosing problems with PAGENT and AT-TLS"
To help with problem determination with PAGENT and AT-TLS, activate tracing and logging.

Diagnosing problems with PAGENT and AT-TLS:

To help with problem determination with PAGENT and AT-TLS, activate tracing and logging.

About this task

PAGENT has its own log file with the default name `/tmp/pagent.log`, which contains messages about loading AT-TLS rules. Invalid rules are rejected, and errors are written to the PAGENT log file. You can specify different levels of logging in the PAGENT configuration file by using the statement `LogLevel`. `LogLevel 511` gives the most verbose logging.

The TCP/IP stack and the AT-TLS service log messages use SYSLOGD. The AT-TLS level of tracing is specified by using the advanced options in the connectivity rules. The highest (more verbose) level of tracing is 255.

The name and location of the log files are specified in the configuration file of the SYSLOGD (`/etc/syslog.conf`). The following example shows a configuration that can be used during testing:

```
*.* /var/log/%Y/%m/%d/errors
```

This syslogd configuration creates log files with names like `/var/log/2010/03/20/errors`. Every time syslogd is restarted, it creates a directory based on the current date. A good procedure is to restart syslogd once a day at midnight.

For more information, see the *Diagnosing Policy Agent problems* chapter of the z/OS Communications Server IP Diagnosis Guide on the z/OS library web page.

Related concepts:

"Application Transparent Transport Layer Security" on page 519
AT-TLS is a service provided by the z/OS Communication Server Policy Agent (PAGENT) and the TCP/IP stack. The AT-TLS service manages SSL connections on behalf of applications that are running on z/OS. The z/OS applications are unaware that SSL is used in the connection with partner applications.

Related tasks:

"Enabling SSL on z/OS by using AT-TLS" on page 517
You can use Application Transparent Transport Layer Security (AT-TLS) to provide Secure Sockets Layer (SSL) services on behalf of WebSphere Message Broker on z/OS. AT-TLS is part of z/OS Communication Server.

"Creating a RACF key ring" on page 520
To create a RACF key ring, you must first generate a RACF CA certificate and a personal certificate for WebSphere Message Broker, then connect the certificates to the key ring.

"Configuring and activating the policy agent (PAGENT)" on page 522
Configure PAGENT by updating the TCP/IP profile, granting RACF permission to TCP/IP resources, preparing the PAGENT startup JCL, and activating syslogd.

"Defining and installing AT-TLS policies" on page 524
Define and install AT-TLS policies by using the IBM Configuration Assistant for z/OS Communications Server.

"Testing and verifying AT-TLS" on page 527
Test and verify AT-TLS by using the SOAP Nodes sample.

Configuring the broker to use SSL with JMS nodes

Configure your broker to work with a JMS provider that supports JMS clients that can connect by using the Secure Sockets Layer (SSL) protocol.

Before you begin

Before you start: Create a keystore file to store the broker's certificates: "Setting up a public key infrastructure" on page 504.

About this task

The JMS 1.1 Specification states that JMS does not provide features for controlling or configuring message integrity or message privacy. JMS providers typically support these additional features, and provide their own administration tools to configure these services. Clients can get the appropriate security configuration as part of the administered objects that they use.

If you want to apply SSL security to the JMS connections created by the three built-in nodes JMSInput, JMSOutput, and JMSReply, check the documentation supplied by your chosen JMS provider. The configuration of the JNDI administered objects that are used by the JMS nodes is specific to each JMS provider.

The three built-in nodes JMSInput, JMSOutput, and JMSReply are referred to in this topic by the generic term JMS nodes; apply the information and instructions here to the specific type of node that you are using.

One example of a JMS provider that provides SSL support for connecting JMS clients is TIBCO Enterprise Message Service (EMS). The following sections describe the authentication model used for JMS nodes, with specific reference to TIBCO EMS, and provide information about how to connect JMS nodes to a TIBCO EMS JMS Server securely by using SSL:

1. “SSL authentication model for the JMS nodes”
2. “Configuring your JMS nodes to use SSL-enabled JNDI administered objects”

SSL authentication model for the JMS nodes:
About this task

The JMS provider TIBCO EMS supports Java clients that can use either the Java Secure Sockets Extension (JSSE) Java package, or an SSL implementation supplied by Entrust. For details about the services provided, see the documentation provided with your chosen package.

TIBCO EMS supports a number of different authentication scenarios, but JMS nodes can use only client authentication to the server. In this scenario, the TIBCO EMS server requests the client's digital certificate during an SSL handshake, and checks its issuer against the server's list of trusted Certificate Authorities. If the authority is not in the server's list, further communications are prevented with the JMS node.

Therefore, you must configure the EMS server to explicitly enable client authentication of the SSL certificates in its configuration file; configure the JNDI administered SSL JMS connection factories for the same level of support.

Configuring your JMS nodes to use SSL-enabled JNDI administered objects:
About this task

The JMS nodes use JNDI to look up a connection factory object that is used to create JMS connections to a TIBCO EMS server.

Procedure

1. Configure the JMS node property Connection factory name to specify a pre-configured connection factory that is enabled for SSL connectivity.

   Make sure that you have set the appropriate parameters in the corresponding SSL JMS connection factory definition:

   - Enable client authentication
   - Specify the SSL protocol in the server URL
   - Set other parameters to define the support your require.

   See the provider's documentation for information about how to generate this JNDI administered object:
2. Configure the JMS node property Location JNDI Bindings with the URL that points to the JNDI bindings containing the JNDI administered objects for SSL connectivity.

   For TIBCO EMS , this URL takes the following format:

   `tibjmsnaming://server_name:ssl_port`

   - *server_name* is the host name of the computer where the server is installed.
   - *ssl_port* is the server port for SSL connectivity; typically, this is port 7243 for a TIBCO EMS server.
3. Make the TIBCO EMS client JAR files available to the broker to which you deploy the message flow that includes your JMS nodes. Use the **mqsicreateconfigurableservice** or the **mqsichangeproperties** command to set the JMSProviders configurable service property jarsURL to point to the directory that contains the JMS provider's client JAR files and the SSL vendor's JAR files.

If you are using JSSE for the SSL support, the following JAR files are typically located in the `jarsURL` directory:

- `jsse.jar`
- `net.jar`
- `jcert.jar`
- `tibcrypt.jar`

You can find standard non-SSL client JAR files in the same location.

Configuring SOAPInput and SOAPReply nodes to use SSL (HTTPS)

Configure the SOAP nodes to communicate with other applications that use HTTPS by creating a keystore file, and configuring the broker to use SSL.

Before you begin

Before you start: Set up a public key infrastructure (PKI) at broker or execution group level: "Setting up a public key infrastructure" on page 504.

About this task

Follow these steps to configure the SOAPInput and SOAPReply nodes to communicate with other applications using HTTP over SSL:

1. Configure HTTP listener properties for an execution group
2. Test your configuration

Configuring HTTP listener properties for an execution group:

You do not need to set properties to enable the HTTP listener for an execution group, but can optionally change values.

Procedure

1. Optional: Enable Client Authentication (mutual authentication):

```
mqsichangeproperties broker_name
  -e execution_group_name
  -o HTTPSConnector
  -n clientAuth -v true
```

2. Optional: Change the SSL protocol. The default protocol for the SOAPInput node is TLS. Issue the following command to change it to SSL:

```
mqsichangeproperties broker_name
  -e execution_group_name
  -o HTTPSConnector
  -n sslProtocol -v SSL
```

What to do next

If you have changed any of the HTTP listener properties for the execution group, take the following steps:

1. Restart the broker.
2. Use the following command to display HTTP listener properties:

```
mqsireportproperties broker_name
  -e execution_group_name
  -o HTTPSConnector  -r
```

Testing your configuration:

About this task

Use the SOAP Nodes sample to test your configuration. You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Procedure

1. Import the sample.
2. Enable SSL in the SOAPNodesSampleConsumerFlow message flow:
   a. Open the Invoke_submitOP subflow.
   b. Change the HTTPTransport properties for the SOAPRequest node. Set an https **Web service URL**, and change other SSL properties as necessary.
3. Enable SSL in the SOAPNodesSampleProvider message flow. Select **Use HTTPS** in the HTTP Transport properties panel.
4. Refresh the BAR file and deploy.
5. Test the sample.

Related concepts:

"Security overview" on page 351
When you are designing a WebSphere Message Broker application it is important to consider the security measures that are needed to protect the information in the system.

Related tasks:

"Implementing SSL authentication" on page 504
Use SSL authentication to enhance security in your broker environment.

"Configuring an HTTPRequest node to use SSL (HTTPS)" on page 538
Configure the HTTPRequest node to communicate with other applications that use HTTP over SSL.

Related reference:

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"HTTPInput node" on page 4474
Use the HTTPInput node to receive an HTTP message from an HTTP client for processing by a message flow.

"HTTPReply node" on page 4484
Use the HTTPReply node to return a response from the message flow to an HTTP client. This node generates the response to an HTTP client from which the input message was received by the HTTPInput node, and waits for confirmation that it has been sent.

"HTTPRequest node" on page 4488
Use the HTTPRequest node to interact with a web service.

Configuring SOAPRequest and SOAPAsyncRequest nodes to use SSL (HTTPS)

Configure the SOAPRequest and SOAPAsyncRequest nodes to communicate with other applications that use HTTP over SSL.

Before you begin

Before you start: Set up a public key infrastructure (PKI) at broker or execution group level: "Setting up a public key infrastructure" on page 504.

Configuring the nodes:
Procedure

1. On the HTTP Transport page of the properties for the node, set the Web Service URL to point to the HTTP server to call.
2. Set other SSL properties as appropriate.

What to do next

Test your configuration.

Testing your configuration:
About this task

Use the SOAP Nodes sample to test your configuration. You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Procedure

1. Import the sample.
2. Enable SSL in the SOAPNodesSampleConsumerFlow message flow:
   a. Open the Invoke_submitOP subflow.
   b. Change the HTTPTransport properties for the SOAPRequest node. Set an https **Web service URL**, and change other SSL properties as necessary.
3. Enable SSL in the SOAPNodesSampleProvider message flow. Select **Use HTTPS** in the HTTP Transport properties panel.
4. Refresh the BAR file and deploy.
5. Test the sample.

Related concepts:

"Security overview" on page 351
When you are designing a WebSphere Message Broker application it is important to consider the security measures that are needed to protect the information in the system.

Related tasks:

"Implementing SSL authentication" on page 504
Use SSL authentication to enhance security in your broker environment.

"Configuring an HTTPRequest node to use SSL (HTTPS)" on page 538
Configure the HTTPRequest node to communicate with other applications that use HTTP over SSL.

Related reference:

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"HTTPInput node" on page 4474
Use the HTTPInput node to receive an HTTP message from an HTTP client for processing by a message flow.

"HTTPReply node" on page 4484
Use the HTTPReply node to return a response from the message flow to an HTTP client. This node generates the response to an HTTP client from which the input message was received by the HTTPInput node, and waits for confirmation that it has been sent.

"HTTPRequest node" on page 4488
Use the HTTPRequest node to interact with a web service.

Configuring HTTPInput and HTTPReply nodes to use SSL (HTTPS)

Configure the HTTPInput and HTTPReply nodes to communicate with other applications that use HTTPS by creating a keystore file, configuring the broker or execution group to use SSL, and creating a message flow to process HTTPS requests.

Before you begin

Before you start: Set up a public key infrastructure (PKI) at broker level by following the instructions in "Setting up a public key infrastructure" on page 504.

About this task

Follow these steps to configure the HTTPInput and HTTPReply nodes to communicate with other applications using HTTP over SSL:

1. If you are using the broker listener:Configure the broker to use SSL
2. If you are using the execution group listener: Configure the execution group to use SSL
3. Create a message flow
4. Test your configuration

If you have configured your broker and execution groups such that the broker listener is used for some execution groups, and the execution group listener for other execution groups, you must complete step 1 for the first set of execution groups and step 2 for each execution group in the second set.

For information about which listener to use for HTTPS messages, see "HTTP listeners" on page 1589.

Configuring the broker to use SSL:
About this task

Complete the following steps:

Procedure

1. Turn on SSL support in the broker, by setting a value for **enableSSLConnector**

   ```
   mqsichangeproperties broker name
     -b httplistener -o HTTPListener
     -n enableSSLConnector -v true
   ```

2. Optional: If you do not want to use the default port 7083 for HTTPS messages, specify the **port** on which the broker listens:

```
mqsichangeproperties broker name
  -b httplistener -o HTTPSConnector
  -n port -v Port to listen on for https
```

On UNIX systems, only processes that run under a privileged user account (in most cases, root) can bind to ports lower than 1024.For the broker to listen on these ports, the user ID under which the broker is started must be root.

3. Optional: Enable Client Authentication (mutual authentication):

```
mqsichangeproperties broker_name -b httplistener -o HTTPSConnector
  -n clientAuth -v true
```

4. Restart the broker after changing one or more of the HTTP listener properties.
5. Optional: Use the following commands to display HTTP listener properties:

```
mqsireportproperties broker_name -b httplistener -o AllReportableEntityNames -a
mqsireportproperties broker_name -b httplistener -o HTTPListener -a
mqsireportproperties broker_name -b httplistener -o HTTPSConnector  -a
```

Configuring an execution group to use SSL:
About this task

Complete the following steps:

Procedure

1. Optional: Specify a specific port on which the execution group listens for HTTPS requests, or leave the value unset to use the next available port number.

```
mqsichangeproperties broker name
  -e execution_group_name -o HTTPSConnector
  -n explicitlySetPortNumber -v port_number
```

On UNIX systems, only processes that run under a privileged user account (in most cases, root) can bind to ports lower than 1024. For the execution group to listen on these ports, the user ID under which the broker is started must be root.

If you do not complete this step, the first available port in the default range (7843 - 7884) is used.

2. Optional: Enable Client Authentication (mutual authentication):

```
mqsichangeproperties broker_name
  -e execution_group_name -o HTTPSConnector
  -n clientAuth -v true
```

3. Optional: Change the SSL protocol. The default protocol for the HTTPInput node is TLS. Run the following command to change it to SSL:

```
mqsichangeproperties broker_name
  -e execution_group_name -o HTTPSConnector
  -n sslProtocol -v SSL
```

4. Restart the broker after changing one or more of the listener properties.
5. Optional: Use the following command to display HTTPS properties:

```
mqsireportproperties broker_name
  -e execution_group_name -o HTTPSConnector  -r
```

Creating a message flow to process HTTPS requests:
About this task

You can create a simple message flow to use HTTPS by connecting an HTTPInput node to an HTTPReply node. The two most important properties to set on the HTTPInput node are:

- Path suffix for URL; for example, /* or /testHTTPS.

- Use HTTPS.

/* means that the HTTPInput node matches against any request that is sent to the HTTP listener on a designated port. This option is useful for testing purposes, but is not suitable for production systems.

You can now deploy the message flow to the broker. If you have completed all the documented steps, message BIP3132 is written to the local system log (on Windows, the event log), stating that the HTTPS listener has been started.

You can now test the system.

Testing your configuration:
About this task

The simplest method of testing whether HTTPS is configured correctly is to use a Web browser to make a request to the broker over HTTPS.

Start a Web browser and enter the following URL:

```
https://localhost:7083/testHTTPS
```

Change values in the URL to reflect the changes that you have made in your broker configuration; for example, the port number. When a window is displayed asking you to accept the certificate, select **Yes**. The browser refreshes the window and displays an empty HTML page:

- In Mozilla browsers, the empty HTML page looks like the following example:

```
<html>
  <body/>
</html>
```

- In Internet Explorer, the following information is displayed:

```
XML document must have a top level element. Error processing resource
'https://localhost:7083/testHTTPS'
```

These responses mean that a blank page was returned, indicating that the setup worked correctly. To add content to the empty page, you can add a Compute node to the flow.

You can use another HTTPS client to process HTTPS requests. Read the documentation for the client to find out how to configure it to make client connections over SSL.

You can also use another HTTPS client, such as a Java or .net client, instead of the Web browser. Depending on the type of client, you might need to export the certificate (which was created with keytool) from the keystore file associated with the HTTP listener, then import it into the keystore for the client. Read the client documentation to find out how to configure the client to make client connections over SSL.

Related concepts:

"Security overview" on page 351
When you are designing a WebSphere Message Broker application it is important to consider the security measures that are needed to protect the information in the system.

"HTTP listeners" on page 1589
You can choose between broker-wide listeners and execution group (embedded) listeners to manage HTTP messages in your HTTP flows. Learn about the two types of listener, how ports are assigned to them, and how you can switch from one to the other for individual execution groups.

Related tasks:

"Implementing SSL authentication" on page 504
Use SSL authentication to enhance security in your broker environment.

"Configuring an HTTPRequest node to use SSL (HTTPS)"
Configure the HTTPRequest node to communicate with other applications that use HTTP over SSL.

Related reference:

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"HTTPInput node" on page 4474
Use the HTTPInput node to receive an HTTP message from an HTTP client for processing by a message flow.

"HTTPReply node" on page 4484
Use the HTTPReply node to return a response from the message flow to an HTTP client. This node generates the response to an HTTP client from which the input message was received by the HTTPInput node, and waits for confirmation that it has been sent.

"HTTPRequest node" on page 4488
Use the HTTPRequest node to interact with a web service.

Configuring an HTTPRequest node to use SSL (HTTPS)

Configure the HTTPRequest node to communicate with other applications that use HTTP over SSL.

Before you begin

Before you start: Set up a public key infrastructure (PKI) at broker level: "Setting up a public key infrastructure" on page 504

About this task

This topic describes the steps that you need to follow when configuring an HTTPRequest node on a Windows system. The steps that you must follow on other operating systems are almost identical.

To enable an HTTPRequest node to communicate using HTTP over SSL, an HTTPS server application is required. The information provided in this topic shows how to use the HTTPInput node for SSL as the server application, but the same details also apply when you are using any other server application.

Complete the following sub-tasks:

1. "Creating a message flow to make HTTPS requests" on page 539
2. "Testing your example" on page 539.

Creating a message flow to make HTTPS requests:
About this task

The following message flow creates a generic message flow for converting a WebSphere MQ message into an HTTP Request:

Procedure

1. Create a message flow with the nodes MQInput->HTTPRequest->Compute->MQOutput.
2. On the MQInput node, set the queue name to HTTPS.IN1 and create the WebSphere MQ queue.
3. On the MQOutput node, set the queue name to HTTPS.OUT1 and create the WebSphere MQ queue.
4. On the HTTPRequest node, set the Web Service URL to point to the HTTP server to call. To call the HTTPInput node, use `https://localhost:7083/testHTTPS`.
5. On the Advanced properties tab of the HTTPRequest node, set the **Response message location in tree** property to *OutputRoot.BLOB*.
6. On the Compute node, add the following ESQL code:

```
CREATE COMPUTE MODULE test_https_Compute
    CREATE FUNCTION Main() RETURNS BOOLEAN
    BEGIN
        -- CALL CopyMessageHeaders();
        CALL CopyEntireMessage();
        set OutputRoot.HTTPResponseHeader = null;
        RETURN TRUE;
    END;

    CREATE PROCEDURE CopyMessageHeaders() BEGIN
        DECLARE I INTEGER;
        DECLARE J INTEGER;
        SET I = 1;
        SET J = CARDINALITY(InputRoot.*[]);
        WHILE I < J DO
            SET OutputRoot.*[I] = InputRoot.*[I];
            SET I = I + 1;
        END WHILE;
    END;

    CREATE PROCEDURE CopyEntireMessage() BEGIN
        SET OutputRoot = InputRoot;
    END;
END MODULE;
```

What to do next

The message flow is now ready to be deployed to the broker and tested.

Testing your example:
About this task

To test that the example works, complete the following steps:

Procedure

1. Follow the instructions in "Configuring HTTPInput and HTTPReply nodes to use SSL (HTTPS)" on page 535, including testing the example.
2. Deploy the HTTPRequest message flow.

3. Put a message to the WebSphere MQ queue HTTPS.IN1. If successful, a message appears on the output queue. If the process fails, an error appears in the local error log (which is the event log on Windows).

Related concepts:

"Security overview" on page 351
When you are designing a WebSphere Message Broker application it is important to consider the security measures that are needed to protect the information in the system.

Related tasks:

"Implementing SSL authentication" on page 504
Use SSL authentication to enhance security in your broker environment.

"Configuring HTTPInput and HTTPReply nodes to use SSL (HTTPS)" on page 535
Configure the HTTPInput and HTTPReply nodes to communicate with other applications that use HTTPS by creating a keystore file, configuring the broker or execution group to use SSL, and creating a message flow to process HTTPS requests.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Creating ESQL for a node" on page 2394
Create ESQL code to customize the behavior of a Compute, Database, DatabaseInput, or Filter node in an ESQL file.

"Deploying resources" on page 3234
Deploy message flow applications to execution groups by sending a broker archive (BAR) file to a broker, which unpacks and stores the contents ready for when your message flows are started.

Related reference:

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"HTTPInput node" on page 4474
Use the HTTPInput node to receive an HTTP message from an HTTP client for processing by a message flow.

"HTTPReply node" on page 4484
Use the HTTPReply node to return a response from the message flow to an HTTP client. This node generates the response to an HTTP client from which the input message was received by the HTTPInput node, and waits for confirmation that it has been sent.

"HTTPRequest node" on page 4488
Use the HTTPRequest node to interact with a web service.

Enabling SSL on the WebSphere MQ Java Client

The WebSphere MQ Java Client supports SSL-encrypted connections over the server-connection (SVRCONN) channel between an application and the queue

manager. Configure SSL support for connections between applications that use the CMP API (including the WebSphere Message Broker Toolkit and the WebSphere Message Broker Explorer) and a broker.

About this task

For one-way authentication, when the client CMP application authenticates the broker, perform the following steps:

Procedure

1. Generate or obtain all the appropriate keys and certificates. You must include a signed pkcs12 certificate for the server and the appropriate public key for the certificate authority that signed the pkcs12 certificate. See "Creating SSL certificates for the WebSphere MQ Java Client" on page 542, for some example steps for creating keys and certificates.
2. Add the pkcs12 certificate to the queue manager certificate store and assign it to the queue manager. Use the standard WebSphere MQ facilities; for example, WebSphere MQ Explorer.
3. Add the certificate of the certificate authority to the JSSE truststore of the Java Virtual Machine (JVM) at the CMP application end using a tool such as Keytool.
4. Decide which cipher suite to use and change the properties on the server-connection channel by using WebSphere MQ Explorer, to specify the cipher suite to be used. This channel has a default name of `SYSTEM.BKR.CONFIG`; this name is used unless you have specified a different name on the Connect to Remote Broker wizard; see "Connecting to a remote broker" on page 902 and "Connecting to a remote broker on z/OS in the WebSphere Message Broker Explorer" on page 904.
5. Add the required parameters (cipher suite, for example) to the CMP application. If a truststore other than the default is used, its full path must be passed in by the truststore parameter.

Results

When you have performed these steps, the CMP application connects to the broker if it has a valid signed key that has been signed by a trusted certificate authority.

What to do next

For two-way authentication, when the broker also authenticates the CMP application, perform the following additional steps:

1. Generate or obtain all the appropriate keys and certificates. You must include a signed pkcs12 certificate for the client and the appropriate public key for the certificate authority that signed the pkcs12 certificate. See "Creating SSL certificates for the WebSphere MQ Java Client" on page 542, for some example steps for creating keys and certificates.
2. Add the certificate of the certificate authority to the queue manager certificate store by using the standard WebSphere MQ facilities.
3. Set the server-connection channel to always authenticate. Specify SSLCAUTH(REQUIRED) in **runmqsc**, or in WebSphere MQ Explorer.
4. Add the pkcs12 certificate to the JSSE keystore of the JVM at the CMP application end by using a tool such as Keytool.

5. If you are not using the default keystore, its full path must be passed into the CMP through the keystore parameter

When you have performed these steps, the broker allows the CMP application to connect only if that application has a certificate signed by one of the certificate authorities in its keystore.

You can make further restrictions by using the sslPeerName field; for example, you can allow connections only from certificate holders with a specific company or department name in their certificates. In addition, you can invoke a security exit for communications between the CMP applications and the broker; see "Using security exits" on page 555.

Related concepts:

"Security overview" on page 351
When you are designing a WebSphere Message Broker application it is important to consider the security measures that are needed to protect the information in the system.

Related tasks:

"Using security exits" on page 555
Define a security exit on the WebSphere MQ channel when you create a broker connection.

"Implementing SSL authentication" on page 504
Use SSL authentication to enhance security in your broker environment.

"Considering security for the WebSphere Message Broker Toolkit and WebSphere Message Broker Explorer" on page 500
Set up appropriate levels of security for the WebSphere Message Broker Toolkit and WebSphere Message Broker Explorer.

Related reference:

"Security requirements for administrative tasks" on page 3644
You can configure access permissions to govern which users and groups can manipulate objects in the broker network. Security requirements for administrative tasks depend on the platform that you use.

Related information:

WebSphere MQ Version 7 Information Center online

Creating SSL certificates for the WebSphere MQ Java Client:

The WebSphere MQ Java Client supports SSL-encrypted connections over the server-connection (SVRCONN) channel between an application and the queue manager. To configure SSL-encrypted connections you must first create key stores and certificates.

Before you begin

Before you start:

- Create a broker
- Start the broker
- Set the environment variable JAVA_HOME to the location of the IBM Key Management tools in the WebSphere MQ install, for example `C:\Program Files\IBM\WebSphere MQ\gskit\jre\` or `/opt/mqm/ssl/jre`.

About this task

Each WebSphere MQ queue manager has a key repository for certificates. When an application attempts to connect to a secure queue manager, the application's certificate must be validated against the contents of the queue manager's key repository. One option for configuring SSL for the queue manager is to use a self-signed certificate.

Two certificates must be signed and created. One must be created for the server queue manager, and a second created for the client, for example the WebSphere Message Broker Explorer.

The instructions in this topic use the **gsk7cmd** command to create and sign the certificates. Run the **gsk7cmd** for a full list of the parameters you can use on the command. To run the **gsk7cmd** command:

- On Windows, enter the following command on a command line:
  `C:\Program Files\IBM\gsk7\bin\gsk7cmd`
- On Linux, enter the following command on a command line:
  `/opt/mqm/ssl/jre/bin/gsk7cmd`

Refer to the WebSphere MQ security documentation for more information about SSL, and creating certificates.

Creating a server certificate for the queue manager:
About this task

Use the IBM Key Management tools on the command line to create a certificate for the queue manager. In the following example you must replace the following parameters with your own values:

password
A password for the certificate repository.

qmname
The name of the queue manager for which you want to create a certificate in lower case.

QMNAME
The name of the queue manager for which you want to create a certificate in upper case.

Procedure

1. Run the following command to create a key repository of type `cms`:

   ```
   gsk7cmd
     -keydb-create
     -dbkey.kdb
     -pwPASSWORD
     -typecms -stash
   ```

   The `key.crl`, `key.kdb`, `key.rdb`, and `key.sth` key files are created.
2. Run the following command to create a self-signed certificate, where the `-dn` flag contains details of your organization:

   ```
   gsk7cmd
     -cert-create
     -dbkey.kdb
   ```

```
  -pwPASSWORD
  -label"qmname"
  -dn"CN=My Queue Manager,O=My Company,C=UK"
  -expire1000
```

3. Run the following command to create a request for a personal certificate:

```
gsk7cmd
  -certreq-create
  -dbkey.kdb
  -pwPASSWORD
  -label"ibmwebspheremqqmname"
  -dn"CN=My Queue Manager,O=My Company,C=UK"
  -fileQMNAME_request.arm
```

4. Sign the certificate using a certificate authority.
   - To obtain a certificate from a certificate authority, you must send the file containing a certificate signing request to your chosen certificate authority.
   - Alternatively you can use the IBM Key Management tools on the command line to sign the certificate.

```
gsk7cmd
  -cert-sign
  -dbkey.kdb
  -pwPASSWORD
  -label"qmname"
  -fileQMNAME_request.arm
  -targetQMNAME_signed.arm
  -expire364
```

5. Run the following command to add the signed certificate to the repository:

```
gsk7cmd
  -cert-receive
  -dbkey.kdb
  -pwPASSWORD
  -fileQMNAME_signed.arm
```

6. Run the following command to export the signed client userid certificate in a transferable format (in this case PKCS12), with the associated private key and public CA certificate:

```
gsk7cmd
  -cert-export
  -dbkey.kdb
  -pwPASSWORD
  -label"ibmwebspheremqqmname"
  -targetQMNAME_personal.p12
  -target_pwPASSWORD
  -target_typepkcs12
```

7. Delete the certificate from the repository:

```
gsk7cmd
  -cert-delete
  -dbkey.kdb
  -pwPASSWORD
  -label"ibmwebspheremqqmname"
```

8. Create a subdirectory called *QMNAME*_CMS, and navigate to this directory on the command line.
9. Run the following command to create a certificate repository in the *QMNAME*_CMS directory:

```
gsk7cmd
  -keydb-create
  -dbkey.kdb
  -pwPASSWORD
  -typecms –stash
```

10. Run the following command to import the PKCS12 file into the repository:

```
gsk7cmd
  -cert-import
  -file"QMNAME_personal.p12"
  -pwPASSWORD
  -typepkcs12
  -targetkey.kdb
  -target_pwPASSWORD
  -target_typecms
```

11. Return to the original directory in which you created the key repository in step 1.

What to do next

Follow the steps in the next section to create a client certificate for the WebSphere Message Broker Explorer.

Creating a client certificate for the WebSphere Message Broker Explorer:
About this task

Use the IBM Key Management tools on the command line to create a certificate for the WebSphere Message Broker Explorer. In the following example you must replace the following parameters with your own values:

password
A password for the certificate repository.

qmname
The name of the queue manager for which you want to create a certificate in lower case. This is the same value used in the steps to create a client certificate for the queue manager.

USERID
The user id for which you want to create a certificate.

Procedure

1. In the directory where you created a key repository for the server queue manager in step 1 above, run the following command to create a request (private key plus certificate details) for a certificate to be signed for the server queue manager:

```
gsk7cmd
  -certreq-create
  -dbkey.kdb
  -pwPASSWORD
  -label"ibmwebspheremqqmname"
  -dn"CN=userid@mycompany.com,O=My Company,C=UK"
  -fileUSERID_request.arm
```

2. Run the following command to sign the certificate:

```
gsk7cmd
  -cert-sign
  -dbkey.kdb
  -pwPASSWORD
  -label"qmname"
  -fileUSERID_request.arm
  -targetUSERID_signed.arm
  -expire364
```

3. Run the following command to add the signed certificate to the repository:

```
gsk7cmd
  -cert-receive
  -dbkey.kdb
  -pwPASSWORD
  -fileUSERID_signed.arm
```

4. Run the following command to export the signed client userid certificate in a transferable format (in this case pkcs12), with the associated private key and public CA certificate:

```
gsk7cmd
  -cert-export
  -dbkey.kdb
  -pwPASSWORD
  -label"ibmwebspheremqqmname"
  -targetUSERID_personal.p12
  -target_pwPASSWORD
  -target_typepkcs12
```

5. Delete the certificate from the repository:

```
gsk7cmd
  -cert-delete
  -dbkey.kdb
  -pwPASSWORD
  -label"ibmwebspheremqqmname"
```

6. Create a subdirectory called *USERID*_JKS, and navigate to this directory on the command line.
7. Run the following command to create a certificate repository in the *USERID*_JKS directory:

```
gsk7cmd
  -keydb-create
  -dbkeyStore.jks
  -pwPASSWORD
  -typejks
```

8. Run the following command to import the pkcs12 file into the repository:

```
gsk7cmd
  -cert-import
  -file"USERID_personal.p12"
  -pwPASSWORD
  -typepkcs12
  -targetkeyStore.jks
  -target_pwPASSWORD
  -target_typejks"
```

9. Return to the original directory in which you created the key repository in step 1.

What to do next

You must now copy the files from the Label_CMS directory to your queue manager's SSL directory. For example, /var/mqm/qmgrs/*QM1*/ssl or C:\Program Files\IBM\WebSphere MQ\Qmgrs*QM1*\ssl. The keystore.jksfile in the LABEL_JKS directory must be on the same machine as the WebSphere Message Broker Explorer. You might also require the AMQCLCHL.TAB file to be copied to the same system as the WebSphere Message Broker Explorer. This file can be found in the queue manager's @ipcc directory, for example, /var/mqm/qmgrs/*QM1*/@ipcc or C:\Program Files\IBM\WebSphere MQ\qmgrs*QM1*\@ipcc.

When you configure the SSL settings in the WebSphere Message Broker Explorer you must specify the full path to the keystore.jksfile.

Related concepts:

"Security overview" on page 351
When you are designing a WebSphere Message Broker application it is important to consider the security measures that are needed to protect the information in the system.

Related tasks:

"Using security exits" on page 555
Define a security exit on the WebSphere MQ channel when you create a broker connection.

"Implementing SSL authentication" on page 504
Use SSL authentication to enhance security in your broker environment.

"Enabling SSL on the WebSphere MQ Java Client" on page 540
The WebSphere MQ Java Client supports SSL-encrypted connections over the server-connection (SVRCONN) channel between an application and the queue manager. Configure SSL support for connections between applications that use the CMP API (including the WebSphere Message Broker Toolkit and the WebSphere Message Broker Explorer) and a broker.

Related reference:

"Security requirements for administrative tasks" on page 3644
You can configure access permissions to govern which users and groups can manipulate objects in the broker network. Security requirements for administrative tasks depend on the platform that you use.

Related information:

WebSphere MQ Version 7 Information Center online

Securing the connection to CICS Transaction Server for z/OS by using SSL

Configure the CICSRequest node to communicate with CICS Transaction Server for z/OS over the Secure Sockets Layer (SSL) protocol by updating a CICSConnection configurable service or the CICSRequest node to use SSL.

Before you begin

Before you start:

Ensure that you have completed the following tasks:

1. The CICSRequest node does not support a separate truststore, so the keystore file must provide both personal and signer certificates. If client-authentication (CLIENTAUTH) is enabled in the TCPIPSERVICE in CICS, the broker keystore file must also contain a personal certificate that is trusted by CICS. To set up a public key infrastructure (PKI) at broker or execution group level, follow the instructions in "Setting up a public key infrastructure" on page 504.
2. Create a message flow project and message set project, as described in "Creating a message flow project" on page 1425 and "Working with a message set project" on page 2838.
3. Define the COMMAREA data structure as a message set, as described in "Defining a CICS Transaction Server for z/OS data structure" on page 2193.
4. Configure IP InterCommunications (IPIC) protocol on CICS, as described in "Preparing the environment for the CICSRequest node" on page 736.

About this task

To configure the CICSRequest node to use SSL, complete the following steps:

Procedure

1. For client-authenticated (CLIENTAUTH) SSL connections, CICS expects the SSL client certificate to be mapped to a RACF user ID. Therefore the SSL client certificate must be mapped to a RACF user ID before attempting to establish the SSL connection to CICS. If the client certificate is not mapped to a RACF user ID, the broker might display a `ECI_ERR_NO_CICS` response. You can map a client certificate to a RACF user ID by using the RACF command **RACDCERT**, which stores the client certificate in the RACF database and associates a user ID with it, or by using RACF certificate name filtering. Client certificates can be mapped one-to-one with a user ID, or a mapping from one to the other can be provided to allow a many-to-one mapping. You can achieve this mapping by using one of the following methods:
   - **Associating a client certificate with a RACF user ID**
     a. Copy the certificate that you want to process into an MVS™ sequential file. The file must have variable length, blocked records (RECFM=VB), and be accessible from TSO.
     b. Run the **RACDCERT** command in TSO by using the following syntax:

        ```
        RACDCERT ADD('datasetname') TRUST [ ID(userid) ]
        ```

        Where:
        - *datasetname* is the name of the data set containing the client certificate.
        - *userid* is the user ID to be associated with the certificate. This parameter is optional. If omitted, the certificate is associated with the user issuing the **RACDCERT** command.

        When you issue the **RACDCERT** command, RACF creates a profile in the DIGTCERT class. This profile associates the certificate with the user ID. You can then use the profile to translate a certificate to a user ID without giving a password. For full details of RACF commands, see z/OS Security Server RACF Command Language Reference.
   - **RACF certificate name filtering**

     With certificate name filtering, client certificates are not stored in the RACF database. The association between one or more certificates and a RACF user ID is achieved by defining a filter rule that matches the distinguished name of the certificate owner or issuer (CA). A sample filter rule might look like the following example:

     ```
     RACDCERT ID(DEPT3USR) MAP SDNFILTER
     (OU=DEPT1.OU=DEPT2.O=IBM.L=LOC.SP=NY.C=US)
     ```

     This sample filter rule would associate user ID DEPT3USR with all certificates when the distinguished name of the certificate owner contains the organizational unit DEPT1 and DEPT2, the organization IBM, the locality LOC, the state/province NY, and the country US.
2. Turn on SSL support in the broker by setting the `cicsServer` property on the CICSConnection configurable service, as shown in the following example. This example changes the CICSRequest node that is configured to use the *myCICSConnection* configurable service for the CICS instance that is running at `mycicsregion.com` port 56789. After you run this command, the CICSRequest node connects to CICS over SSL.

   ```
   mqsichangeproperties MB7BROKER -c CICSConnection -o myCICSConnection -n
   cicsServer -v ssl://mycicsregion.com:56789
   ```

   Alternatively you can configure the `CICS server` property directly on the CICSRequest node.

What to do next

Next: When you have configured the broker or the CICSRequest node to use SSL, develop a message flow that contains a CICSRequest node by following the steps in "Developing a message flow with a CICSRequest node" on page 2199.

Related concepts:

"Security overview" on page 351
When you are designing a WebSphere Message Broker application it is important to consider the security measures that are needed to protect the information in the system.

"CICS Transaction Server for z/OS connectivity" on page 2174
Use the CICSRequest node to connect WebSphere Message Broker with CICS Transaction Server for z/OS applications.

"Using the WebSphere Message Broker Explorer to work with configurable services" on page 644
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view, add, modify and delete configurable services.

Related tasks:

"Implementing SSL authentication" on page 504
Use SSL authentication to enhance security in your broker environment.

"Changing connection information for the CICSRequest node" on page 738
You can create a configurable service that the CICSRequest node or message flow refers to at run time for connection information, instead of defining the connection properties on the node or the message flow. The advantage being that you can change the host name and performance values without needing to redeploy your message flow.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

Related reference:

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

"CICSRequest node" on page 4321
Use the CICSRequest node to call CICS Transaction Server for z/OS programs over TCP/IP-based IP InterCommunications (IPIC) protocol.

Securing the connection to IMS by using SSL

Configure the IMSRequest node to communicate with IMS over the Secure Sockets Layer (SSL) protocol by creating a keystore file, and configuring the broker to use SSL.

Before you begin

Before you start:

Set up a public key infrastructure (PKI) at broker level by following the instructions in "Setting up a public key infrastructure" on page 504.

About this task

To configure the IMSRequest node to use SSL, complete the following steps.

Procedure

1. Turn on SSL support in the broker by setting the `UseSSL` and `SSLEncryptionType` properties on the IMSConnect configurable service, as shown in the following example.

   This example changes the IMSRequest node that is configured to use the *myIMSConnectService* configurable service. After you run this command, the IMSRequest node connects to IMS over SSL.

   ```
   mqsichangeproperties MB7BROKER -c IMSConnect -o myIMSConnectService -n
   UseSSL,SSLEncryptionType -v True,Weak
   ```

2. Optional: Develop a message flow that contains a IMSRequest node.
3. Test your configuration.

Related concepts:

"Security overview" on page 351
When you are designing a WebSphere Message Broker application it is important to consider the security measures that are needed to protect the information in the system.

"IMS connections" on page 2137
Open Transaction Manager Access (OTMA) is used to provide access to IMS from WebSphere Message Broker.

"Using the WebSphere Message Broker Explorer to work with configurable services" on page 644
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view, add, modify and delete configurable services.

Related tasks:

"Implementing SSL authentication" on page 504
Use SSL authentication to enhance security in your broker environment.

"Changing connection information for the IMSRequest node" on page 732
You can create a configurable service that the IMSRequest node or message flow refers to at run time for connection information, instead of defining the connection properties on the node or the message flow. The advantage being that you can change the host name, performance, and security values without needing to redeploy your message flow.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

Related reference:

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"IMSRequest node" on page 4504
Use the IMSRequest node to send a request to run a transaction on a local or remote IMS system, and wait for a response. IMS Connect must be configured and running on the IMS system.

SSL and the TCP/IP nodes

Configure a TCP/IP configuration to use SSL to secure connectivity to and from the TCPIP nodes.

Related tasks:

"Configuring TCP/IP client nodes to use SSL"
Configure a TCP/IP configuration to use SSL to secure connectivity to and from the TCPIP client nodes.

"Configuring TCP/IP server nodes to use SSL" on page 553
Configure a TCP/IP configuration to use SSL to secure connectivity to and from the TCPIP server nodes.

"Implementing SSL authentication" on page 504
Use SSL authentication to enhance security in your broker environment.

Configuring TCP/IP client nodes to use SSL:

Configure a TCP/IP configuration to use SSL to secure connectivity to and from the TCPIP client nodes.

You can create or modify TCP/IP client connections that use SSL, by creating or modifying a configurable service. You can specify the type of protocol, and the allowed cipher suites. By default, SSL is not enabled for any configurable services. The nodes use the standard broker keystore and truststore configuration.

Before you begin

Before you start: Set up a public key infrastructure (PKI) at broker or execution group level by following the instructions in "Setting up a public key infrastructure" on page 504.

About this task

Follow these steps to configure the TCPIP nodes to use SSL:

1. "Changing a TCP/IP client configuration to use SSL"
2. "Creating a TCP/IP client configuration that uses SSL" on page 552

Changing a TCP/IP client configuration to use SSL:
About this task

Use the **mqsichangeproperties** command to change an existing TCPIPClient configurable service.

Procedure

1. The following command specifies that the myTCPIPClientService configurable service must use SSLv3 as the protocol, with any available cipher suite.

```
mqsichangeproperties MYBROKER
  -c TCPIPClient
  -o myTCPIPClientService
  -n SSLProtocol
  -v SSLv3
```

2. Restart the execution group that contains the message flows.

Creating a TCP/IP client configuration that uses SSL:
About this task

Use the **mqsicreateconfigurableservice** command to create a TCPIPClient configurable service.

Procedure

1. The following command creates a TCPIPClient configurable service for making connections on port 1455 on the local machine. It uses the SSL protocol SSLv3 with a specific list of allowed cipher suites.

```
mqsicreateconfigurableservice MYBROKER
  -c TCPIPClient
  -o myTCPIPClientService
  -n Port,Hostname,SSLProtocol,SSLCiphers
  -v 1455,localhost,SSLv3,SSL_RSA_WITH_RC4_128_MD5;
     SSL_RSA_WITH_3DES_EDE_CBC_SHA
```

2. Restart the execution group that contains the message flows.

Testing your configuration:
About this task

Use either a TCPIPClientInput node, or a TCPIPClientOutput node to open a connection to a remote SSL server application that is listening on a TCP/IP port.

Related concepts:

"SSL and the TCP/IP nodes" on page 551
Configure a TCP/IP configuration to use SSL to secure connectivity to and from the TCPIP nodes.

"Security overview" on page 351
When you are designing a WebSphere Message Broker application it is important to consider the security measures that are needed to protect the information in the system.

Related tasks:

"Implementing SSL authentication" on page 504
Use SSL authentication to enhance security in your broker environment.

"Configuring TCP/IP server nodes to use SSL" on page 553
Configure a TCP/IP configuration to use SSL to secure connectivity to and from the TCPIP server nodes.

Related reference:

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

Configuring TCP/IP server nodes to use SSL:

Configure a TCP/IP configuration to use SSL to secure connectivity to and from the TCPIP server nodes.

You can create or modify TCP/IP server connections that use SSL, by creating or modifying a configurable service. You can specify:

The type of protocol.

The allowed cipher suites.

A key alias.

Whether a connecting client should provide authentication information.

By default, SSL is not enabled for any configurable services. The nodes use the standard broker keystore and truststore configuration.

Before you begin

Before you start: Set up a public key infrastructure (PKI) at broker level by following the instructions in "Setting up a public key infrastructure" on page 504.

About this task

Follow these steps to configure the TCPIP nodes to use SSL:

1. "Changing a TCP/IP server configuration to use SSL"
2. "Creating a TCP/IP server configuration that uses SSL"

Changing a TCP/IP server configuration to use SSL:
About this task

Use the **mqsichangeproperties** command to change an existing TCPIPServer configurable service.

Procedure

1. The following command changes a TCPIPServer configurable service to use SSLv3 with any available cipher suite. Connecting clients are not asked to authenticate.

```
mqsichangeproperties MYBROKER
  -c TCPIPClient
  -o myTCPIPServerService
  -n SSLProtocol
  -v SSLv3
```

2. Restart the execution group that contains the message flows.

Creating a TCP/IP server configuration that uses SSL:
About this task

Use the **mqsicreateconfigurableservice** command to create a TCPIPServer configurable service.

Procedure

1. The following command creates a TCPIPServer configurable service for making connections on port 1455. It uses the SSL protocol SSLv3 with a specific list of allowed cipher suites. Connecting clients are required to authenticate.

```
mqsicreateconfigurableservice MYBROKER
     -c TCPIPServer
     -o myTCPIPServerService
     -n Port,SSLProtocol,SSLCiphers,SSLClientAuth
     -v 1455,SSLv3,SSL_RSA_WITH_RC4_128_MD5;
        SSL_RSA_WITH_3DES_EDE_CBC_SHA,require
```

2. Restart the execution group that contains the message flows.

Using an SSL key alias:
About this task

A key alias identifies the key that is to be used for the SSL connection, if the keystore for your broker or execution group contains more than one key. Use the **mqsichangeproperties** or **mqsicreateconfigurableservice** as appropriate, with the `SSLKeyAlias` property. The default value `""` or `none`, means that an SSL key alias is not used. Any other string identifies the alias.

Note: If the keystore contains more than one key, and no key alias is defined, the Java virtual machine arbitrarily chooses a key at run time.

The following command creates a TCPIPServer configurable service for making connections on port 1455. It uses the SSL protocol SSLv3 with the cipher suites SSL_RSA_WITH_RC4_128_MD5 and SSL_RSA_WITH_3DES_EDE_CBC_SHA. It requires the client to authenticate itself, and uses the key alias `MyKey` to identify the key to be used.

```
mqsicreateconfigurableservice MYBROKER
  -c TCPIPServer
  -o myTCPIPServerService
  -n Port,SSLProtocol,SSLCiphers,SSLClientAuth,SSLKeyAlias
  -v 1455,SSLv3,SSL_RSA_WITH_RC4_128_MD5;SSL_RSA_WITH_3DES_EDE_CBC_SHA
     ,require,MyKey
```

The following command changes a TCPIPServer configurable service to use the first key retrieved from the keystore, with SSL protocol SSLv3. SSLClientAuth is disabled.

```
mqsichangeproperties MYBROKER
  -c TCPIPClient
  -o myTCPIPServerService
  -n SSLProtocol
  -v SSLv3
```

Testing your configuration:
About this task

To test your configuration, connect an SSL-enabled client, such as another program, or a web browser, to the server port. Connection error messages, such as handshake failures, or untrusted keys, indicate that you must change the configuration.

Client identity:
About this task

If SSL client authentication is requested or required, and the client successfully authenticates, the distinguished name is present as an identity source token in the

properties parser, in the tree propagated from the Open terminal at connection time. This applies only to the TCPIPServerInput node.

The `IdentitySourceToken` field is set to the distinguished name from the client certificate.

The `IdentitySourceType` field is set to the string `username`.

The `IdentitySourceIssuedBy` field is set to the issuer of the certificate presented by the client.

If SSL client authentication is requested, and the client does not provide the required credentials, the fields are set to blank.

Related concepts:

"Security overview" on page 351
When you are designing a WebSphere Message Broker application it is important to consider the security measures that are needed to protect the information in the system.

Related tasks:

"Implementing SSL authentication" on page 504
Use SSL authentication to enhance security in your broker environment.

Related reference:

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

Using security exits

Define a security exit on the WebSphere MQ channel when you create a broker connection.

About this task

To create a security exit on the WebSphere MQ channel that you define for communications between the WebSphere Message Broker Toolkit, or WebSphere Message Broker Explorer and the broker, you must define a security exit when you create the connection.

Procedure

1. Select from the following options:
   - In the WebSphere Message Broker Toolkit, right-click the Brokers folder and click **Add Remote Broker**.
   - In the WebSphere Message Broker Explorer, right-click the Brokers folder and click **Connect** > **Remote Broker**.

   The Create a Broker Connection wizard opens.
2. Enter the values for **Queue Manager Name**, **Host**, and **Port** that you want to use. In the WebSphere Message Broker Explorer, click **Next**.
3. Enter the Security Exit **Class** name. The name must be a valid Java class name.

4. Set the **JAR File location** for the Security Exit that is required on this connection. Click **Browse** to find the file location.

Results

The security exit is started every time a message passes across the connection.

What to do next

Alternatively, use SSL to communicate between the Administration API for WebSphere Message Broker (also known as the CMP API) and the broker; see "Implementing SSL authentication" on page 504.

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

"Security overview" on page 351
When you are designing a WebSphere Message Broker application it is important to consider the security measures that are needed to protect the information in the system.

"Security exits" on page 354
Use security exit programs to verify that the partner at the other end of a connection is genuine.

Related tasks:

"Broker component security" on page 497
You must consider several security aspects when you are setting up brokers running on Windows, Linux, or UNIX platforms.

"Implementing SSL authentication" on page 504
Use SSL authentication to enhance security in your broker environment.

Related reference:

"Security requirements for administrative tasks" on page 3644
You can configure access permissions to govern which users and groups can manipulate objects in the broker network. Security requirements for administrative tasks depend on the platform that you use.

Setting up z/OS security

On z/OS, you must complete several security configuration tasks before WebSphere Message Broker can work correctly.

About this task

The steps you need to follow are described in this topic and also in the following topics:

- "Setting up WebSphere MQ" on page 558
- "Setting up WebSphere Message Broker Toolkit access on z/OS" on page 559

Decide on the started task name of the broker. This name is used to set up started task authorizations, and to manage your system performance.

Decide on a data set naming convention for your WebSphere Message Broker PDSE. A typical name might be `WMQI.MQP1BRK.CNTL`, where `MQP1` is the queue manager name. You must give the WebSphere Message Broker, WebSphere MQ,

and z/OS administrators access to these data sets. You can give these professionals controlled access in several ways, for example:

- Give each user individual access to the specific data set.
- Define a generic data set profile, defining a group that contains the user IDs of the administrators. Grant the group control access to the generic data set profile.

Define an OMVS group segment for this group so that information can be extracted from the External Security Manager (ESM) database to enable you to use Publish/Subscribe security.

Define an OMVS segment for the started task user ID and give its home directory sufficient space for any WebSphere Message Broker dumps. Consider using the started task procedure name as the started task user ID. Check that your OMVS segment is defined by using the following TSO command:

```
LU userid OMVS
```

The command output includes the OMVS segment, for example:

```
USER=MQP1BRK NAME=SMITH, JANE OWNER=TSOUSER
CREATED=99.342 DEFAULT-GROUP=TSOUSER PASSDATE=01.198
PASS-INTERVAL=30
......
OMVS INFORMATION
----------------
UID=0000070594
HOME=/u/MQP1BRK
PROGRAM=/bin/sh
CPUTIMEMAX=NONE
ASSIZEMAX=NONE
FILEPROCMAX=NONE
PROCUSERMAX=NONE
THREADSMAX=NONE
MMAPAREAMAX=NONE
```

The command:

```
df -P /u/MQP1BRK
```

displays the amount of space used and available, where `/u/MQP1BRK` is the value from `HOME` (on a previous line). This command shows you how much space is currently available in the file system. Check with your data administrators that this space is sufficient. You require a minimum of 400 000 blocks available if a dump is taken.

Associate the started task procedure with the user ID to be used. For example, you can use the STARTED class in RACF. The WebSphere Message Broker and z/OS administrators must agree on the name of the started task.

WebSphere Message Broker administrators need an OMVS segment and a home directory. Check the setup previously described.

The started task user IDs and the WebSphere Message Broker administrators need access to the install processing files, the component-specific files, and the home directory of the started task. During customization, the file ownership can be changed to alter group access. This change might require super user authority.

When the service user ID is *root*, all libraries loaded by the broker, including all user-written plug-in libraries, and all shared libraries that they might access, also

have root access to all system resources (for example, file sets). Review and assess the risk involved in granting this level of authorization.

For more information about various aspects of security, see "Security overview" on page 351.

Related concepts:

"Security overview" on page 351
When you are designing a WebSphere Message Broker application it is important to consider the security measures that are needed to protect the information in the system.

Related tasks:

"Setting up WebSphere MQ"
This is part of the larger task of setting up security on z/OS, and gives details of the authorities that your user ID needs to perform the required operations.

"Setting up WebSphere Message Broker Toolkit access on z/OS" on page 559
Access to the WebSphere Message Broker Toolkit is controlled from Windows or Linux on x86.

Related reference:

"Customization tasks and roles on z/OS" on page 3984
System programmers do most of the customization of WebSphere Message Broker.

"Summary of required access (z/OS)" on page 3985
The professionals in your organization require access to components and resources on z/OS.

Setting up WebSphere MQ

This is part of the larger task of setting up security on z/OS, and gives details of the authorities that your user ID needs to perform the required operations.

About this task

The user ID of the person running the create broker (`BIPCRBK`) job needs *UPDATE* access to the component PDSE, *READ/EXECUTE* access to the installation directory, and *READ/WRITE/EXECUTE* access to the broker directory.

If you do not use queue manager security, you do not need to read the rest of this topic. Topic "Creating the broker component" on page 629 provides detailed statements on how to protect your queues.

By default, the broker's internal queues, which all have names of the form:

```
SYSTEM.BROKER.*
```

should be protected. These names cannot be changed. Restrict access to the broker started task user ID, and to WebSphere Message Broker administrators.

If you are using Publish/Subscribe, subscribers must have authority to PUT to SYSTEM.BROKER.CONTOL.QUEUE.

You can control which applications can use queues used by message flows. Applications must be able to PUT and GET to queues defined in all nodes.

Related concepts:

"Security overview" on page 351
When you are designing a WebSphere Message Broker application it is important to consider the security measures that are needed to protect the information in the

system.

Related tasks:

"Setting up z/OS security" on page 556
On z/OS, you must complete several security configuration tasks before WebSphere Message Broker can work correctly.

"Setting up WebSphere Message Broker Toolkit access on z/OS"
Access to the WebSphere Message Broker Toolkit is controlled from Windows or Linux on x86.

"Security considerations on z/OS" on page 597

"Creating the broker component" on page 629
When you are creating a broker on z/OS, one of the tasks is to create the broker component.

Related reference:

"Customization tasks and roles on z/OS" on page 3984
System programmers do most of the customization of WebSphere Message Broker.

"Summary of required access (z/OS)" on page 3985
The professionals in your organization require access to components and resources on z/OS.

Setting up WebSphere Message Broker Toolkit access on z/OS

Access to the WebSphere Message Broker Toolkit is controlled from Windows or Linux on x86.

About this task

The WebSphere Message Broker Toolkit must run on Windows or Linux on x86.

Related concepts:

"Security overview" on page 351
When you are designing a WebSphere Message Broker application it is important to consider the security measures that are needed to protect the information in the system.

Related tasks:

"Setting up z/OS security" on page 556
On z/OS, you must complete several security configuration tasks before WebSphere Message Broker can work correctly.

"Setting up WebSphere MQ" on page 558
This is part of the larger task of setting up security on z/OS, and gives details of the authorities that your user ID needs to perform the required operations.

"Security considerations on z/OS" on page 597

Related reference:

"Customization tasks and roles on z/OS" on page 3984
System programmers do most of the customization of WebSphere Message Broker.

"Summary of required access (z/OS)" on page 3985
The professionals in your organization require access to components and resources on z/OS.

"Security requirements for Windows systems" on page 3651
Security requirements depend on the administrative task that you want to perform.

"Security requirements for Linux and UNIX platforms" on page 3648
View a summary of the authorizations in a Linux or UNIX environment.

Execution group user IDs on z/OS

On z/OS, you can specify an alternative user ID to run an execution group so that it accesses resources according to the permissions assigned to it, rather than the permissions assigned to the main broker user ID.

You can specify an alternative user ID to run an execution group, which means that you can run one or more message flows under a different user ID from the main broker ID. When external resources are accessed by a message flow, access is granted according to the permissions assigned to the user ID that is running the execution group. By having different user IDs for different execution groups, you can control the access to resources at the level of the execution group rather than at the level of the main broker user ID. The user IDs for the execution groups must be in the same group as the main broker user ID, so that shared resources can be read and updated.

On z/OS, the user ID assigned to the broker is the started task (STC) user ID that is assigned to the started task JCL. By default, each broker on z/OS has a single started task JCL, which is used to start the main broker address space and all associated execution group address spaces. However, you can specify a different started task JCL, and therefore a different user ID, for one or more execution groups. As a result, execution groups can be started using a different started task JCL and run under different user IDs with different permissions to access resources. For example, an execution group can access messages from WebSphere MQ through the execution group's task ID (rather than the main broker ID) by default. Execution groups can also access files according to the permissions that are assigned to the execution group's user ID.

You can specify the required environment variable in the main broker profile, BIPBPROF. You can use the `MQSI_STARTEDTASK_FIXED_`*`EG`*, `MQSI_STARTEDTASK_MULTI_`*`EG`*, or `MQSI_STARTEDTASK_DEFAULT` environment variables to specify a different started task and user ID, for one or more execution groups. These environment variables override the started task and user ID that are associated with the broker, and replace them with the started task and user ID associated with a specific execution group:

- Use the `MQSI_STARTEDTASK_FIXED_`*`EG=STC`* environment variable to specify the name of a single execution group (where *EG* is the 8-character name of the execution group, and *STC* is the name of the execution group started task JCL).
- Use the `MQSI_STARTEDTASK_MULTI_`*`EG=STC`* environment variable to override the user ID for multiple execution groups (where *EG* functions as a wildcard and *STC* is the name of the started task JCL that is used to start each of the execution groups). For example, specify `MQ05` in place of `EG` to override the user ID for any execution groups in which the last 8 characters start with `MQ05`.
- Use the `MQSI_STARTEDTASK_DEFAULT=`*`STC`* environment variable to override the started task JCL (STC) for all execution groups, unless it is overridden by the `MQSI_STARTEDTASK_FIXED_`*`EG`* or `MQSI_STARTEDTASK_MULTI_`*`EG`* environment variable.

For information about how to define a user ID on an execution group, see "Specifying an alternative user ID to run an execution group on z/OS" on page 561.

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related tasks:

"Specifying an alternative user ID to run an execution group on z/OS"
You can change the user ID under which an execution group runs so that it can access resources according to the permissions assigned to it, rather than the permissions assigned to the main broker user ID.

"Setting up z/OS security" on page 556
On z/OS, you must complete several security configuration tasks before WebSphere Message Broker can work correctly.

"Creating a broker on z/OS" on page 620
Create the broker component and the other resources on which it depends.

"Customizing the broker JCL" on page 625
This subtask is part of the larger task of creating a broker on z/OS.

"Creating the environment file" on page 626
This is part of the larger task of creating a broker on z/OS.

"Copying the broker started task to the procedures library" on page 629
This is part of the larger task of creating a broker on z/OS.

Specifying an alternative user ID to run an execution group on z/OS

You can change the user ID under which an execution group runs so that it can access resources according to the permissions assigned to it, rather than the permissions assigned to the main broker user ID.

Before you begin

Before you start

Before starting this task, read the following topics:

- "Creating a broker on z/OS" on page 620
- "Execution group user IDs on z/OS" on page 560

About this task

Complete the following steps to specify an alternative user ID for the execution group, to be used instead of the broker's user ID:

Procedure

1. Create the new RACF started task profile with a new user ID, which will be used to run the execution group. Consider the following points when you are creating the new started task:
   - The new started task must be created with an OMVS segment including a unique UID, home directory, and the ability to create data sets under the broker's HLQ and alias.
   - The started task procedure name to be used for the execution group address space must start with the same four characters as the main broker started task. For example, if the main broker started task is `MQ01BRK`, the started task name for the execution group could be `MQ01EG1` but not `MQ02EG2`. As a result, consistency is maintained between the main broker started task, the execution group, and the queue manager, which helps to identify the relationship between them. If the first four characters are not the same, the execution group is started using the main broker started task JCL.

2. Ensure that the new user ID associated with the new started task JCL has the same RACF primary group as the existing broker user ID, so that they can access shared resources. Also ensure that the new user ID has the required privileges to the existing broker filesystem and dataset (which it should have through the primary group access).
3. Ensure that the MQ and SMF authorizations are updated for the new user ID; for more information, see "Summary of required access (z/OS)" on page 3985.
4. Copy the existing broker started task JCL to the new started task JCL in the PROCLIB.
5. Ensure that the main broker user ID has been granted permission to the SUPERUSER.PROCESS.KILL RACF profile. This permission is required so that the main control address space can recover any existing execution group address spaces in the event of a failure.
6. Refresh the started RACF classes to implement the updates.
7. Change the user ID by adding the appropriate environment variable to the broker's profile.
   - The execution group name specified in the environment variable is the last 8 characters of the execution group, after any overrides have been applied. This is the same 8-character name that is displayed as the STEPNAME against the execution group address space in SDSF.
   - Ensure that the execution group name contains only characters that are valid in the environment variable. If invalid characters are used, the user ID cannot be overridden.
   - If you specify more than one environment variable, they are read in the following order (with MQSI_STARTEDTASK_FIXED_*EG* taking precedence):
     a. MQSI_STARTEDTASK_FIXED_*EG*
     b. MQSI_STARTEDTASK_MULTI_*EG*
     c. MQSI_STARTEDTASK_DEFAULT
8. Submit BIPGEN to the broker's ENVFILE.
9. Restart the broker.

Related concepts:

"Execution group user IDs on z/OS" on page 560
On z/OS, you can specify an alternative user ID to run an execution group so that it accesses resources according to the permissions assigned to it, rather than the permissions assigned to the main broker user ID.

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related tasks:

"Setting up z/OS security" on page 556
On z/OS, you must complete several security configuration tasks before WebSphere Message Broker can work correctly.

"Creating a broker on z/OS" on page 620
Create the broker component and the other resources on which it depends.

"Customizing the broker JCL" on page 625
This subtask is part of the larger task of creating a broker on z/OS.

"Creating the environment file" on page 626
This is part of the larger task of creating a broker on z/OS.

"Copying the broker started task to the procedures library" on page 629
This is part of the larger task of creating a broker on z/OS.

Chapter 6. Configuring brokers for development environments

Set up application development environments on Linux on x86 or Windows to create, test, and deploy message flows and associated resources.

About this task

If you are new to WebSphere Message Broker, you can set up a basic development environment on either Windows or Linux on x86 by installing the broker component and the WebSphere Message Broker Toolkit, and creating a default configuration. Run the Default Configuration wizard to create a broker, and try the sample programs that are supplied in the WebSphere Message Broker Toolkit.

Some of the samples depend on the default configuration, which creates a broker that has the fixed name, MB7BROKER, and a queue manager for the broker that has the fixed name, MB7QMGR.

Details of the default configuration are provided in “Creating a default configuration” on page 564.

When you have tried out your first few samples, you can create and configure more brokers, and set up an environment that can support your application developers. Many samples are supplied, and provide education and guidance about how to use many of the facilities of WebSphere Message Broker.

See the full list of samples in “Samples” on page 98. The samples are grouped in sections according to the features that they use, and the operations that they perform. For example, the section "Web services samples" lists the samples that use Web services nodes to communicate with WebSphere Service Registry and Repository (WSRR) and other Web applications.

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

To create and configure brokers in addition to, or instead of, the default configuration, follow the instructions in “Creating a development environment” on page 567.

When you start the WebSphere Message Broker Toolkit, a workbench session opens, which you can use to create, configure, and manage your application development resources. You can configure your workbench session in various ways to suit your working environment and preferences. These options are described in “Configuring the workbench” on page 570.

Related tasks:

Chapter 7, “Configuring brokers for test and production environments,” on page 579
Create one or more brokers on one or more computers, and configure them on your test and production systems to process messages that contain your business data.

Chapter 8, “Administering brokers and broker resources,” on page 899
Administering brokers and associated broker resources includes the tasks that you perform frequently to activate and manage those resources. Choose the method you prefer to administer your brokers and associated resources.

Chapter 9, “Developing message flow applications,” on page 1019
Develop message flows to process your business messages and data.

Creating a default configuration

Use the Default Configuration wizard to create and test a basic broker configuration.

Before you begin

Before you run the Default Configuration wizard, complete the following steps:

- Install the WebSphere Message Broker Toolkit, the Broker component, and the Broker component prerequisite products on this computer.
- Access the Default Configuration wizard through the WebSphere Message Broker Toolkit, which is available only on Linux on x86 and Windows.

About this task

By using the Default Configuration wizard, you create a default configuration on your local computer so that you can explore the product and run the samples. You can also remove the default configuration, if it exists, that has been created on your logon account.

- “Creating the default configuration”
- “Removing the default configuration” on page 565

The default configuration is described in more detail in “Verifying your WebSphere Message Broker installation” on page 290. This topic describes how to verify your installation on Linux on x86, Linux on x86-64, or Windows by using either the WebSphere Message Broker Toolkit or the WebSphere Message Broker Explorer.

Creating the default configuration

About this task

The wizard creates the following resources:

- A sample broker named MB7BROKER.
- A WebSphere MQ queue manager named MB7QMGR.

Procedure

1. Start the Default Configuration wizard from the WebSphere Message Broker Toolkit Welcome page, which is displayed the first time you start the WebSphere Message Broker Toolkit. If the Welcome page is not displayed, open it in the WebSphere Message Broker Toolkit by clicking **Help** > **Welcome**.
2. Click **Get Started** on the Welcome page, then click **Create the Default Configuration**.
3. Click **Start the Default Configuration wizard**. The Default Configuration wizard is displayed.
4. The Welcome page of the wizard describes what is about to happen, click **Next** to continue. You can click **Cancel** at any time to cancel the creation of the default configuration.

The wizard checks that the default configuration is not already created.

5. The Default Configuration Summary page lists the resources that will be created. Click **Next** to continue.
6. The Default Configuration Progress page lists the background configuration actions as they occur, and indicates successful completion. You can cancel the creation of the default configuration by clicking **Cancel**. The wizard backs out all the configuration tasks and displays the progress and success of the process. The configuration process is written to a log file in the Eclipse workspace directory:
   - Linux `/home/user_name/IBM/wmbt70/workspace/.metadata/DefaultConfigurationWizard.log`
   - Windows `C:\Documents and settings\user_name\IBM\wmbt70\workspace\.metadata\DefaultConfigurationWizard.log`

   If the default configuration is created successfully, you see an appropriate message. If errors occur, you see an appropriate message and the wizard backs out all the configuration tasks. If an error occurs during the back out process, the wizard displays a list of resources that you must remove manually.
7. You can use the samples to verify the default configuration. **Launch Samples Wizard when finished** is selected by default. Click **Finish** to open the Prepare the Samples wizard.

   If you do not want to open the Prepare the Samples wizard, clear **Launch Samples Wizard when finished** before clicking **Finish**.

   If you are viewing this information from within the WebSphere Message Broker Toolkit, you can open the samples manually by clicking the following sample:

   - Pager

   Alternatively, click **Product Overview** > **Samples** in the WebSphere Message Broker information center in the WebSphere Message Broker Toolkit, to display a list of the available samples.

   You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Removing the default configuration

Procedure

1. Start the Default Configuration wizard from the WebSphere Message Broker Toolkit Welcome page, which is displayed after you start the WebSphere Message Broker Toolkit. If the Welcome page is not displayed, open it in the WebSphere Message Broker Toolkit by clicking **Help** > **Welcome**.
2. Click **Get Started** on the Welcome page, then click **Create the Default Configuration**.
3. Click **Remove the Default Configuration wizard**. The Default Configuration wizard is displayed.
4. The Welcome page of the wizard describes what is about to happen. You can click **Cancel** at any time to cancel the removal of the default configuration.

   The wizard checks that the default configuration is already created.
5. The Remove Default Configuration Summary page lists the resources that will be removed. Click **Next** to continue.

6. The Default Configuration Progress page lists the removal actions as they occur, and indicates successful completion. The removal process is written to a log file in the Eclipse workspace directory:
   - Linux `/home/user_name/IBM/wmbt70/workspace/.metadata/DefaultConfigurationWizard.log`
   - Windows `C:\Documents and settings\user_name\IBM\wmbt70\workspace\.metadata\DefaultConfigurationWizard.log`
7. A message confirms that the default configuration has been removed successfully. Click **Finish** to close the wizard.

   If errors occur during the removal of the default configuration, the wizard displays the errors, and also writes them to the log file. Follow the advice in the log, and try each step again.

Results

If you experience problems when you are using the wizard to remove the default configuration, you might have to remove the default configuration manually. For more information, see "You experience problems with the default configuration" on page 3382.

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

"Samples" on page 98
The WebSphere Message Broker Toolkit provides samples that show the features that are available in WebSphere Message Broker, and how to use them. This topic provides links to the information about the individual samples.

Related tasks:

"Creating a development environment" on page 567
Create a development environment on Linux on x86 or Windows to develop your message flow applications.

Chapter 7, "Configuring brokers for test and production environments," on page 579
Create one or more brokers on one or more computers, and configure them on your test and production systems to process messages that contain your business data.

"Resolving problems that occur when you start resources" on page 3371
Use the advice given here to help you to resolve common problems that can occur when you start resources.

"Resolving problems when running samples" on page 3366
Use the advice given here to help you to resolve common problems that can arise when you run or remove samples.

Related reference:

"Installation Guide" on page 233
Installation information for WebSphere Message Broker is provided in the *Installation Guide* that is supplied as a PDF file with your product package.

"WebSphere MQ resources for the broker" on page 585
Each broker depends on a number of WebSphere MQ resources: some are required, others are optional, and depend on your environment and requirements. Some of these resources are created for you, but others you must define for yourself.

Creating a development environment

Create a development environment on Linux on x86 or Windows to develop your message flow applications.

Before you begin

Before you start:

- "Installing the Broker component" on page 267
- "Installing the WebSphere Message Broker Toolkit" on page 276

On Windows, you can use the Launchpad to install both these components. For details, see "Installing by using the Windows Launchpad" on page 262.

If you are new to WebSphere Message Broker, and have not created a default configuration, you might want to use that option to set up your first configuration. You can use the default environment to run some of the supplied sample programs, and explore the features that are supported by WebSphere Message Broker. Follow the instructions in "Creating a default configuration" on page 564.

About this task

To create your own development environment on a computer that is running Linux on x86 or Windows, complete the following steps:

Procedure

1. Start the WebSphere Message Broker Toolkit in one of the following ways:
   - Linux From the main menu:
     - On Red Hat, click **Programming** > **IBM WebSphere Message Broker Toolkit**.
     - On SUSE Linux, click **All Applications** > **WebSphere Message Broker Toolkit**.
   - Windows Click **Start** > **All Programs** > **IBM WebSphere Message Broker Toolkit** > **IBM WebSphere Message Broker Toolkit 7.0** > **WebSphere Message Broker Toolkit 7.0**, or double-click the shortcut on your desktop.
   - Use the following commands in a command prompt from their location in the root directory for the package group:
     - Linux
       ```
       ./eclipse -product com.ibm.etools.msgbroker.tooling.ide
       ```
     - Windows
       ```
       eclipse.exe -product com.ibm.etools.msgbroker.tooling.ide
       ```

   A new session of the WebSphere Message Broker Toolkit opens, and prompts you for the name of a workspace in which to store the resources that you create in this session.

   You can accept the default value, `workspace`, type your preferred name into the text field, or click Browse to select the location of your choice. Select **Use this as the default and do not ask again** to inhibit the display of the workspace dialog next time you start the WebSphere Message Broker Toolkit.
2. Click **OK** to confirm the location of your workspace. Your WebSphere Message Broker Toolkit session opens, and displays the Welcome page. You can tailor various settings in the WebSphere Message Broker Toolkit to suit your

requirements and your working environment; for example, the colors and fonts. These options are described in "Configuring the workbench" on page 570.

3. Close the Welcome page. You can return to the Welcome page later by clicking **Help** > **Welcome**. If more than one option is listed, select WebSphere Message Broker.
4. In the Broker Application Development perspective, select the Brokers view. Alternatively, click **Window** > **Show view** > **Other** > **Broker Runtime** > **Brokers** to show the Brokers view.
5. Right-click **WebSphere Message Brokers**, and click **New** > **Local Broker** to create a broker, and complete the wizard for this task.

   If you want more detailed instructions, see "Creating a broker for a development environment" on page 569.

   Your configuration might be affected by the operation mode of your broker. The default operation mode is enterprise mode, in which your broker runs with no restrictions. Check with your broker administrator which operation mode applies to your organization; you might have to change the mode of your broker after you have created it. For more information about operation modes, see "Operation modes" on page 48.
6. You must create at least one execution group on your broker. To add additional execution groups to your broker, right-click the broker, and click **New** > **Execution Group**. Enter a name for your execution group, and click **OK**.

   The execution group is the runtime environment in which your message flows run. You can create many execution groups on a single broker, and you can deploy your message flows to one or more execution groups on one or more brokers.

   For more detailed instructions about this task, see "Creating an execution group using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 937.

Results

Your configuration is now ready to use.

What to do next

Next: You can start to develop resources to deploy to your broker. You can create your own message flows, or you can use the patterns and samples that are provided to get you started. For details of these options, see Chapter 9, "Developing message flow applications," on page 1019.

This task has covered the minimum set of steps that you must complete to create a broker and configure your development environment. Typically, as an application developer, you are working in a single platform environment to create, deploy, and test your message flows before they are ready for use in a test of production environment.

More options are available for enhanced configurations, including configurable services. When the requirements of your message flow applications extend beyond this basic configuration, and you need additional configuration to support those requirements, you can find details of these more advanced options in Chapter 7, "Configuring brokers for test and production environments," on page 579.

Related tasks:

Chapter 6, “Configuring brokers for development environments,” on page 563
Set up application development environments on Linux on x86 or Windows to create, test, and deploy message flows and associated resources.

“Creating a default configuration” on page 564
Use the Default Configuration wizard to create and test a basic broker configuration.

Chapter 7, “Configuring brokers for test and production environments,” on page 579
Create one or more brokers on one or more computers, and configure them on your test and production systems to process messages that contain your business data.

Chapter 8, “Administering brokers and broker resources,” on page 899
Administering brokers and associated broker resources includes the tasks that you perform frequently to activate and manage those resources. Choose the method you prefer to administer your brokers and associated resources.

Creating a broker for a development environment

Create a broker by using the WebSphere Message Broker Toolkit on Linux on x86 or Windows.

Before you begin

Before you start:

If you are using the WebSphere Message Broker Explorer, see “Creating a broker using the WebSphere Message Broker Explorer” on page 618.

- On Windows, you must have administrator access rights to create brokers by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer. When creating a broker, you might be prompted to agree to the use of administrator rights or be prompted to enter an administrator user ID and password.
- If you want to configure the broker as a WebSphere MQ trusted application, see “Using WebSphere MQ trusted applications” on page 613.
- Read “Considering security for a broker” on page 501.
- Check which operation mode you are licensed to use. If you do not set a mode, the automatic default is `enterprise` mode; see “Operation modes” on page 48.

About this task

When you create a broker, if the WebSphere MQ queue manager does not exist, the queue manager is automatically created. If WebSphere MQ Version 7.1, or later, has been selected for the queue manager, the channel auth security is automatically disabled.

If the specified queue manager already exists when creating the broker, then it is assumed that the user has applied the appropriate security configuration to meet their requirements, and therefore channel auth security is not disabled.

To create a broker by using the WebSphere Message Broker Toolkit:

Procedure

1. In the Broker Application Development perspective, select the Brokers view. Alternatively, click **Window** > **Show view** > **Other** > **Broker Runtime** > **Brokers** to show the Brokers view.
2. Right-click the Brokers folder, and click **New** > **Local Broker**. The New Local Broker wizard is displayed.
3. Enter a name for the broker and queue manager. You cannot create a broker on a queue manager that is already associated with a broker.
4. On Windows, enter your user name and password. These parameters are not required on Linux on x86.
5. Optional: On Windows, select whether to start the broker automatically when Windows starts.
6. Enter a name for the default execution group, or accept the value "default".
7. Optional: Enter a value for the flow debug port for the default execution group.
8. Click **Finish**.

Results

You have created and started a broker.

What to do next

Return to the instructions in "Creating a development environment" on page 567.

Related concepts:

"WebSphere Message Broker Toolkit" on page 31
The WebSphere Message Broker Toolkit is an integrated development environment and graphical user interface based on the Eclipse platform.

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

Related tasks:

Chapter 6, "Configuring brokers for development environments," on page 563
Set up application development environments on Linux on x86 or Windows to create, test, and deploy message flows and associated resources.

"Creating a broker" on page 611
You can create brokers on every platform that is supported by WebSphere Message Broker. The broker runs as a 64-bit application on all platforms except Linux on x86 and Windows on x86.

"Creating a default configuration" on page 564
Use the Default Configuration wizard to create and test a basic broker configuration.

Configuring the workbench

You can configure various settings in the workbench to suit your requirements and your working environment.

About this task

The following topics show you how to configure aspects of the workbench:

Procedure

- "Changing WebSphere Message Broker Toolkit preferences"
- "Changing workbench capabilities" on page 572
- "Changing WebSphere Message Broker Explorer preferences" on page 654
- "Configuring CVS to run with the WebSphere Message Broker Toolkit" on page 573
- "Configuring the WebSphere Message Broker Toolkit to run Rational ClearCase" on page 574
- "Creating a working set" on page 575
- "Integrating the Rational Team Concert client with the WebSphere Message Broker Toolkit" on page 576

Results

A minimum display resolution of at least 1024 x 768 is required for some dialog boxes, such as the Preferences dialog box.

Related concepts:

"WebSphere Message Broker Toolkit" on page 31
The WebSphere Message Broker Toolkit is an integrated development environment and graphical user interface based on the Eclipse platform.

Related reference:

"WebSphere Message Broker Toolkit" on page 6783
The WebSphere Message Broker Toolkit provides your application development environment on Windows and Linux on x86.

Changing WebSphere Message Broker Toolkit preferences

The WebSphere Message Broker Toolkit has a large number of preferences that you can change to suit your requirements. Some of these are specific to the product plug-ins that you have installed within the workbench, including those for WebSphere Message Broker. Others control more general options, such as the colors and fonts in which information is displayed.

About this task

To access the WebSphere Message Broker Toolkit preferences:

Procedure

1. Select **Window** > **Preferences**.
2. Click the plus sign associated with **Workbench**, typically the first entry in the left pane. An expanded list of options appears, Select the aspect of the WebSphere Message Broker Toolkit that you want to modify. These options might be of interest:

   Startup and shutdown

   Switch on, or off, the prompt at toolkit startup that asks you to confirm the workspace location. Typically you switch this prompt off, so that it does not appear, but you can force it to appear next time you start the WebSphere Message Broker Toolkit if you want to specify a different location.

   You can also specify whether to display the dialog box that asks you to confirm shutdown of the WebSphere Message Broker Toolkit.

Colors and fonts
Change the default fonts and colors that appear in the WebSphere Message Broker Toolkit.

Perspectives
On this dialog, your choices include the option to open a new perspective in a new window.

3. When you have made your changes, click **OK** to close the Preferences dialog.

What to do next

Below the Workbench category in the Preference dialog are items that refer specifically to WebSphere Message Broker resources, such as message flows. Review the following topics for information about setting preferences and other values that are specific to your use of these resources:

- "Message flow preferences" on page 4016
- "Changing ESQL preferences" on page 2408
- "Configuring message set preferences" on page 2840
- Chapter 10, "Testing and debugging message flow applications," on page 3143
- Changing trace settings

Related concepts:

"WebSphere Message Broker Toolkit" on page 31
The WebSphere Message Broker Toolkit is an integrated development environment and graphical user interface based on the Eclipse platform.

Related reference:

"WebSphere Message Broker Toolkit" on page 6783
The WebSphere Message Broker Toolkit provides your application development environment on Windows and Linux on x86.

Changing workbench capabilities

You can configure the workbench to disable access to some of the functional capabilities of WebSphere Message Broker.

About this task

Capabilities is an Eclipse concept that allows you to enable or disable the components of a product. By default, all components of the workbench are enabled.

To access the workbench capabilities:

Procedure

1. Select **Window** > **Preferences**.
2. Click the plus sign associated with **General**. An expanded list of options appears.
3. Click **Capabilities**. You can use the capabilities that are listed to enable or disable various product components; the capabilities are grouped according to a set of predefined categories.
4. Select **Message Broker Toolkit** from the list of capabilities that is displayed, and select the **Advanced** button. A window opens that has a check box for each of the predefined categories.

5. Select the check boxes for the categories that you want to either enable or disable; click either the **Enable All** or **Disable All** button and click **OK**. A pane describes the functionality that is enabled following this action.

What to do next

The predefined categories for the workbench are listed together with a reference to more information about the relevant functional area of WebSphere Message Broker:

- **Message Broker Toolkit - Administration**. See Chapter 8, "Administering brokers and broker resources," on page 899.
- **Message Broker Toolkit - Core**. See "WebSphere Message Broker Toolkit" on page 31.
- **Message Broker Toolkit - Development**. See Chapter 9, "Developing message flow applications," on page 1019.

Related concepts:

"WebSphere Message Broker Toolkit" on page 31
The WebSphere Message Broker Toolkit is an integrated development environment and graphical user interface based on the Eclipse platform.

Related reference:

"WebSphere Message Broker Toolkit" on page 6783
The WebSphere Message Broker Toolkit provides your application development environment on Windows and Linux on x86.

Configuring CVS to run with the WebSphere Message Broker Toolkit

Install CVS as a normal program by following the usual prompts. Not all versions of *CVSNT* are supported by Eclipse.

Procedure

1. Configure CVS by carrying out the following tasks:
   a. Create a directory on your computer, for example, on Windows - `c:\CVSRepository.`
   b. Start the *CVSNT* control panel. Select **Start** > **Programs** > **CVSNT** to see the icon on the desktop.
   c. Stop both the `CVS Service` and the `CVS Lock Service`.
   d. Select the `Repositories` tag, click **Add** and create a new repository. Note that no entry appears on the screen the first time that you do this.
   e. Use the `...` button on the next window to select the directory that you created in step 1a and click **OK**. Note that when CVS has finished formatting its repository the backslash in the directory name is changed to a forward slash.
   f. Select the `Service Status` tab and restart both the `CVS Service` and the `CVS Lock Service`.
2. Enable the CVS Revision tag to be populated in the Eclipse Version fields in the WebSphere Message Broker Toolkit. To do this on Windows:
   a. Select `Window->Preferences`
   b. Expand the `Team` section and click **CVS**
   c. Use the drop down in the **Default keyword substitution:** field and set the value to `ASCII with keyword expansion(-kkv)`

3. Add the WebSphere Message Broker file types to the Eclipse CVS configuration. To do this:
   a. Select **File Content** in the `Team` section of the window you opened in step 2 on page 573
   b. Click **Add** and add `msgflow` as an allowable file extension. Ensure that the value is set to ASCII.
   c. Repeat the above procedure for the following file extensions that the broker uses:
      - `esql`
      - `mset`
      - `mxsd`

      If you use CVS to store other file types, for example, COBOL copybooks add the appropriate file types as well.
   d. Click **OK** when you have finished.

Related concepts:

"Version and keyword information for deployable objects" on page 1443
Use the Broker Archive editor to view the version and keyword information of deployable objects.

Related reference:

"Description properties for a message flow" on page 4016
The description properties for a message flow include the `Version`, `Short Description` and `Long Description`. To view and edit the properties of a message flow click **Flow** > **Properties**.

"Message flow preferences" on page 4016
You can change preferences that determine properties of message flows when you create them.

"Message set preferences" on page 5366
Preferences for message sets.

Configuring the WebSphere Message Broker Toolkit to run Rational ClearCase

To use Rational ClearCase with the WebSphere Message Broker Toolkit, enable the capability in the Preferences page.

About this task

To enable Rational ClearCase in the WebSphere Message Broker Toolkit:

Procedure

1. Click **Window** > **Preferences** to open the Preferences window.
2. Expand **General** in the left pane, and click **Capabilities**.
3. In the Capabilities pane, click **Advanced**. The Advanced window opens.
4. Expand **Team**, and ensure that **ClearCase SCM Adapter** is selected.
5. Click **OK** to close the Advanced window.
6. Click **OK** or **Apply** to apply your changes.

Results

After you enable the ClearCase capability, the ClearCase menu is displayed in the Broker Application Development perspective.

What to do next

To work with ClearCase:

1. Click **ClearCase** > **Connect to Rational ClearCase**.
2. Right-click your project and click **Team** > **Add to Version Control** to add your projects to the ClearCase source control.
3. After you have added your projects to the ClearCase source control, you can perform ClearCase operations.

Related concepts:

"Development repository" on page 45
Use a development repository to benefit from features such as version control and access control of files, which make it easier for teams to work on shared resources.

"Version and keyword information for deployable objects" on page 1443
Use the Broker Archive editor to view the version and keyword information of deployable objects.

Related tasks:

"Configuring CVS to run with the WebSphere Message Broker Toolkit" on page 573
Install CVS as a normal program by following the usual prompts. Not all versions of *CVSNT* are supported by Eclipse.

Related reference:

"Message flow preferences" on page 4016
You can change preferences that determine properties of message flows when you create them.

"Message set preferences" on page 5366
Preferences for message sets.

Creating a working set

Create a working set to limit the number of resources that are displayed in the Broker Development view.

Before you begin

Before you start:

To read about working sets, see "Working sets" on page 42.

About this task

By creating and using a working set, you can reduce the visual complexity of what is displayed in the Broker Development view, making it easier to manage and work with your projects.

To create a new working set, complete the following steps:

Procedure

1. Click the down arrow of the working set section, <all resources>, in the Broker Development view. A list is displayed containing existing working sets and options for editing and deleting existing working sets, and for creating a new working set.
2. Click **New Working Set**. The New Working Set wizard opens.

3. Enter a name for the working set.
4. Select the resources that you want to include in this working set. You can also include all the projects that are dependent on your selected resources by selecting **Automatically include dependent projects in this working set**.
5. Click **Finish**.

Results

The new working set and its associated resources are displayed in the Broker Development view.

What to do next

Next:

In addition to creating new working sets, you can also select, edit, and delete existing working sets by using the options in the Broker Development view menu.

Related concepts:

"Resources" on page 36
The projects, folders, and files that you work with in the WebSphere Message Broker Toolkit workspace are called *resources*. By default, these resources are stored with their metadata in the workspace directory in your local file system. The workspace directory is created the first time that you start the WebSphere Message Broker Toolkit.

"Quick Start wizards overview" on page 1409
You can use a Quick Start wizard to set up the basic resources that are required to develop a message flow application. The wizard sets up and gives names to containers for the resources that you need when you develop a message flow.

Related tasks:

"Creating an application from scratch" on page 1411
Use the "Start from scratch" wizard to create the basic resources that are required to develop a broker application.

"Creating an application based on WSDL or XSD files" on page 1413
You can use existing WSDL or XSD files as the basis for your solution.

"Creating an application based on an existing message set" on page 1415
Create a new application that is based on an existing message set.

Related reference:

"Broker Application Development perspective" on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

Integrating the Rational Team Concert client with the WebSphere Message Broker Toolkit

How you integrate the Rational Team Concert™ client with the WebSphere Message Broker Toolkit.

About this task

To integrate the Rational Team Concert client with the WebSphere Message Broker Toolkit, you must have the Rational Team Concert client in the same package group as the WebSphere Message Broker Toolkit.

The supported version of Rational Team Concert is V2.0.0.2.

Procedure

Use the Installation Manager to install the Rational Team Concert client into the same package group as the WebSphere Message Broker Toolkit.
Using the Installation Manager adds all the Rational Team Concert client capability into the WebSphere Message Broker Toolkit workspace.
You can also install the WebSphere Message Broker Toolkit into the Rational Team Concert client package group.
If the products are in the same package group, they can be installed in any order.

What to do next

To work with Rational Team Concert source control:

1. Start the WebSphere Message Broker Toolkit.
2. Open the Work Items perspective and click **Window** > **Open Perspective** > **Other** > **Work Items**.
3. Click the **Create Repository Connection** link in the Team Artifacts view.
4. Follow the dialog and enter the information given to you by your Jazz™ administrator.
5. To add your project to the repository, right-click the project and select **Team** > **Share Project**.
6. In the Share Project window, select Jazz Source Control as the repository type.

See Getting started in your Rational Team Concert source control workspace for more details about the Rational Team Concert source control operations.

You can also use the Jazz Team Build as your build engine for the WebSphere Message Broker Toolkit; see Building with Jazz Team Build for more information.

Related reference:

"Message flow preferences" on page 4016
You can change preferences that determine properties of message flows when you create them.

"Message set preferences" on page 5366
Preferences for message sets.

Chapter 7. Configuring brokers for test and production environments

Create one or more brokers on one or more computers, and configure them on your test and production systems to process messages that contain your business data.

About this task

On Linux on x86 and Windows, you can install the broker and the WebSphere Message Broker Toolkit, and create an initial configuration to learn about WebSphere Message Broker, and start to develop your applications. You can also configure an environment for more advanced application development and unit test. These tasks are described in Chapter 6, "Configuring brokers for development environments," on page 563.

To configure your test and production environments, use the information in this section to plan and configure the resources that you want:

Procedure

1. Plan the system.
2. If you are configuring a broker on z/OS, Customize the z/OS environment.
3. Ensure that you have the correct authorization and permissions to create and access components. For more information, see "Authorization for configuration tasks" on page 353 and "Broker component security" on page 497.
4. Create the components.
5. If you are using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer, Configure brokers in the developer or administrator workbench.
6. If you want to ensure that you are using the correct operation mode for your license, Check the operation mode of your broker.
7. Create and configure the databases.
8. If you want to ensure the data integrity of transactions, Configure global coordination of transactions.
9. If you want to connect to external resources such as Enterprise Information Systems, IMS, or JMS, Configure properties to connect to external resources.
10. If you want to configure the storage of events for aggregation, Collector, or timeout nodes, or configure monitoring event sources, Configure internal resources that are required by message flows.
11. If you want to view objects in a different language or code page, Change the locale.

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

"Execution groups" on page 53
An execution group is a named grouping of message flows that have been assigned to a broker. The broker enforces a degree of isolation between message flows in distinct execution groups by ensuring that they run in separate address spaces, or as unique processes.

"Authorization for configuration tasks" on page 353
Authorization is the process of granting or denying access to a system resource.

Related tasks:

Chapter 8, "Administering brokers and broker resources," on page 899
Administering brokers and associated broker resources includes the tasks that you perform frequently to activate and manage those resources. Choose the method you prefer to administer your brokers and associated resources.

"Broker component security" on page 497
You must consider several security aspects when you are setting up brokers running on Windows, Linux, or UNIX platforms.

"Developing applications that use the Administration API" on page 956
Develop Java applications that use the Administration API (also known as the CMP API) to communicate with, deploy to, and manage brokers and their associated resources.

Planning a broker environment

When you start to plan your broker environment, you must first consider your resource naming conventions and the design of the WebSphere MQ infrastructure.

About this task

The following topics describe these factors:

- "Considering resource naming conventions" on page 581
- "Designing the WebSphere MQ infrastructure" on page 584

You might also want to consider how your configuration might affect performance of your brokers and message flows. See "Considering performance in the broker environment" on page 586 for information about these aspects.

Procedure

You must also consider how to configure your brokers:

1. Performance: What is the required message throughput? See "Optimizing message flow throughput" on page 587.

   What is the size of the messages that are being processed? Larger messages take longer to process. A few brokers handling many messages might degrade overall performance. See "Considering performance in the broker environment" on page 586.
2. Operation mode: The mode in which your broker is working can affect the number of execution groups and message flows that you can deploy, and the types of node that you can use. See "Restrictions that apply in each operation mode" on page 3657. (This option does not apply to brokers that you create on z/OS systems.)
3. Application independence: Do you need to isolate applications from each other? You might want to separate applications that serve different functions; for example, personnel and finance.

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

"Routing using publish/subscribe applications" on page 2215
You can route your messages to applications using the publish/subscribe method of messaging.

Related tasks:

"Configuring brokers" on page 610
Create and configure the brokers that you want on the operating system of your choice.

"Considering performance in the broker environment" on page 586
When you design your broker environment, and the resources associated with the brokers, decisions that you make can affect the performance of your brokers and applications.

"Configuring user databases" on page 659
Configure databases to hold application or business data that you can access from your message flows.

Considering resource naming conventions

When you plan a new WebSphere Message Broker network, one of your first tasks must be to establish a convention for naming the resources that you create within this network.

Information is provided about the following resources:

- Product component naming conventions
- WebSphere MQ naming conventions
- Database naming conventions

Related reference:

"Designing the WebSphere MQ infrastructure" on page 584
You must create and manage the WebSphere MQ resources that are required to support your brokers, and the applications that connect to them to supply or receive messages.

Naming conventions for brokers and associated resources

Establish a naming convention for all the WebSphere Message Broker resources in your broker environment to ensure that names are unique, and that users creating new resources can be confident of not introducing duplication or confusion.

Consider the names that you use for your brokers and resources:

Brokers

When you create a broker, give it a name that is unique within your broker environment. You must use the same name for that broker when you create it on the system in which it is installed, using the command **mqsicreatebroker**. Broker names are case sensitive except on Windows platforms.

Execution groups

Each execution group name must be unique within a broker.

Message flows and message processing nodes

Each message processing node must be unique within the message flow it is assigned to. For example, if you include two MQOutput nodes in a single message flow, provide a unique name for each one.

Message flow names must be unique within the broker. All references to that name are always to the same message flow. If you assign the same message flow to more than one broker, you must ensure that you maintain unique message flow names across those brokers.

Message sets and messages

Each message name must be unique within the message set to which it belongs.

Message set names must be unique within the broker. All references to that name are always to the same message set. If you assign the same message set to more than one broker, you must ensure that you maintain unique message set names across those brokers.

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related tasks:

Chapter 7, "Configuring brokers for test and production environments," on page 579
Create one or more brokers on one or more computers, and configure them on your test and production systems to process messages that contain your business data.

Related reference:

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"Naming conventions for WebSphere Message Broker for z/OS" on page 3984
Decide upon a naming convention for your WebSphere Message Broker for z/OS broker to make customizing, operating, and administering easier.

WebSphere MQ naming conventions

All WebSphere Message Broker resources have dependencies on WebSphere MQ services and objects. You must therefore also consider what conventions to adopt for WebSphere MQ object names. If you already have a WebSphere MQ naming convention, use a compatible extension of this convention for WebSphere Message Broker resources.

When you create a broker, you must specify a queue manager name. This queue manager is created for you if it does not already exist. Every broker must have a dedicated queue manager.

Ensure that every queue manager name is unique within your network of interconnected queue managers, whether or not every queue manager is in your WebSphere Message Broker network. This ensures that each queue manager can unambiguously identify the target queue manager to which any given message must be sent, and that WebSphere Message Broker applications can also interact with basic WebSphere MQ applications.

WebSphere MQ supports a number of objects defined to queue managers. These objects (queues, channels, and processes) also have naming conventions and restrictions.

In summary, the restrictions are:

- All names must be a maximum of 48 characters in length (channels have a maximum of 20 characters).
- The name of each object must be unique within its type (for example, queue or channel).
- Names for all objects starting with the characters SYSTEM. are reserved for use by IBM.

Additionally, there is a restriction to the length of the user identifier on each platform:

- On AIX, UNIX systems, and z/OS, the maximum length is eight characters.
- On Windows, the maximum length is 20 characters.

There are a few restrictions for naming resources: see "Naming conventions for WebSphere Message Broker for z/OS" on page 3984.

Related tasks:

"Creating a broker" on page 611
You can create brokers on every platform that is supported by WebSphere Message Broker. The broker runs as a 64-bit application on all platforms except Linux on x86 and Windows on x86.

Related reference:

"Naming conventions for WebSphere Message Broker for z/OS" on page 3984
Decide upon a naming convention for your WebSphere Message Broker for z/OS broker to make customizing, operating, and administering easier.

Database naming conventions

Consider the naming conventions you use for databases that you create for application use.

Ensure that the databases that you use for application data, which are accessed through your deployed message flows, are uniquely named throughout your network, so that confusion and errors are avoided by all your users.

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related tasks:

"Creating the user databases" on page 661
If your message flows create, update, retrieve, or delete application and business data in one or more user databases, create the databases before you deploy the message flows to a broker.

"Configuring brokers" on page 610
Create and configure the brokers that you want on the operating system of your choice.

Designing the WebSphere MQ infrastructure

You must create and manage the WebSphere MQ resources that are required to support your brokers, and the applications that connect to them to supply or receive messages.

Brokers

A broker uses WebSphere MQ messages to provide information, status, and instructions about its internal operations. Connections are also required by each WebSphere Message Broker Toolkit with each broker with which it communicates.

Some of these resources that are required are created for you when you create brokers and execution groups. The requirements associated with the broker are described in "WebSphere MQ resources for the broker" on page 585.

Applications and message flows

Your applications exchange messages and other data by communicating with message flows that are running in the broker. You can connect your applications to the broker by using one of the supported communications methods. If your applications are written to use WebSphere MQ, the requirement for the channels or client connections are determined by the types of nodes that you include in your message flows. These resources are application-specific, and you must create these resources yourself.

The following nodes might require WebSphere MQ resources:

- MQInput and MQOutput
- MQReply and MQGet

For more information about creating resources, see the *Intercommunication* section of the WebSphere MQ Version 7 Information Center online.

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related tasks:

"Configuring brokers" on page 610
Create and configure the brokers that you want on the operating system of your choice.

"Customizing the z/OS environment" on page 591
If you are planning to use a z/OS environment, consider whether to create your brokers on z/OS. You must also complete a number of tasks to configure your environment.

"Connecting client applications" on page 1537
Connect your client applications to the broker by using one or more of the supported protocols from other resources and software servers in your network.

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"Message flows" on page 4015
Use the reference information in this section to develop your message flows and related resources.

Related information:

WebSphere MQ Version 7 Information Center online

WebSphere MQ resources for the broker

Each broker depends on a number of WebSphere MQ resources: some are required, others are optional, and depend on your environment and requirements. Some of these resources are created for you, but others you must define for yourself.

WebSphere MQ resources created for you

When you create a broker, the following WebSphere MQ resources are created for you:

- On distributed systems, the queue manager for the broker. Each broker must be associated with a dedicated queue manager. Specify a queue manager name when you create the broker. If this queue manager does not exist, it is created for you.
- Fixed-name queues on the queue manager that hosts this broker. The broker uses these queues to send messages to provide information, status, and instructions about its internal operations

WebSphere MQ resources that you must create yourself

Depending on the setup of your broker, you might need to create some WebSphere MQ resources yourself. You might need some or all the following resources:

- If you create a broker on z/OS, you must create the queue manager. See "Creating a broker on z/OS" on page 620 for more details.
- Define listener connections on the broker queue manager. You must define one listener connection for every protocol that your applications use; for example, TCP/IP.

Create WebSphere MQ resources by using one of the following commands and utilities:

- **runmqsc**
- The PCF interface
- WebSphere MQ Explorer
- WebSphere Message Broker Explorer

For more information about creating resources, see the *Intercommunication* section of the WebSphere MQ Version 7 Information Center online.

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related tasks:

"Creating a broker on z/OS" on page 620
Create the broker component and the other resources on which it depends.

Related reference:

"Designing the WebSphere MQ infrastructure" on page 584
You must create and manage the WebSphere MQ resources that are required to support your brokers, and the applications that connect to them to supply or receive messages.

Related information:

Considering performance in the broker environment

When you design your broker environment, and the resources associated with the brokers, decisions that you make can affect the performance of your brokers and applications.

About this task

Message flows

A message flow includes an input node that receives a message from an application over a particular protocol; for example, WebSphere MQ. The message must be parsed by the input node, although some parsers support partial parsing which might reduce processing, because only the parts of the message that are referenced are parsed. Other processing in a message flow that might affect performance are the amount, efficiency, and complexity of ESQL, access to databases, and how many message tree copies are made.

You must consider how you split your business logic; how much work should the application do, and how much should the message flow do? Every interaction between an application and a message flow involves I/O and message parsing, and therefore adds to processing time. Design your message flows, and design or restructure you applications, to minimize these interactions.

For more information about these factors, see “Optimizing message flow response times” on page 3264.

Messages and message models

The type, format, and size of the messages that are processed can have a significant effect on the performance of a message flow. For example, if you process persistent messages, they have to be stored for safekeeping.

You might need to process messages with a well-defined structure; if so, you can create MRM models for your messages. If you do not plan to interrogate the structure, you can work with undefined messages, such as BLOB messages.

If you are working in XML, be aware that it can be verbose, and therefore produce large messages, but XML message content is easier to understand than other formats, such as CWF. Field size and order might be important; these factors can be included in your MRM model.

For more information about these factors, see “Optimizing message flow response times” on page 3264 and Performance considerations for regular expressions in TDS messages.

Broker configuration

You can create and configure one or more brokers, on one or more computers, and for each broker you can create multiple execution groups, and multiple message flows. Your configuration decisions can influence message flow performance, and how efficiently messages can be processed.

For more information about these factors, see “Tuning the broker” on page 3254, “Optimizing message flow throughput” on page 587.

All these factors are examined in more detail in the Designing for Performance SupportPac (IP04).

For a description of common performance scenarios, review “Resolving problems with performance” on page 3504.

For further articles about WebSphere Message Broker and performance, review these sources:

- The Business Integration Zone on developerWorks. This site has a search facility; enter "performance" and review the links that are returned.
- The developerWorks article on message flow performance.
- The developerWorks article on monitoring resource use.

Related tasks:

“Optimizing message flow throughput”
Each message flow that you design must provide a complete set of processing for messages received from a certain source. This design might result in very complex message flows that include large numbers of nodes that can cause a performance overhead, and might create potential bottlenecks. You can increase the number of message flows that process your messages to provide the opportunity for parallel processing and therefore improved throughput.

“Optimizing message flow response times” on page 3264
You can use different solutions to improve message flow response times.

“Setting configuration timeout values” on page 3258
Change timeout values that affect configuration tasks in the broker.

“Tuning the broker” on page 3254
You can complete several tasks that enable you to tune different aspects of the broker performance.

“Resolving problems with performance” on page 3504
Use the advice given here to help you to resolve common problems with performance.

Related reference:

“Performance considerations when using regular expressions” on page 6309
Take care when specifying regular expressions: some forms of regular expression can involve a large amount of work to find the best match, which might degrade performance.

Optimizing message flow throughput

Each message flow that you design must provide a complete set of processing for messages received from a certain source. This design might result in very complex message flows that include large numbers of nodes that can cause a performance overhead, and might create potential bottlenecks. You can increase the number of message flows that process your messages to provide the opportunity for parallel processing and therefore improved throughput.

About this task

The mode that your broker is working in can affect the number of message flows that you can use; see “Restrictions that apply in each operation mode” on page 3657.

You can also consider the way in which the actions taken by the message flow are committed, and the order in which messages are processed.

Consider the following options for optimizing message flow throughput:

Multiple threads processing messages in a single message flow
When you deploy a message flow, the broker automatically starts an

instance of the message flow for each input node that it contains. This behavior is the default. However, if you have a message flow that handles a very large number of messages, the input source (for example, a WebSphere MQ queue) might become a bottleneck.

You can update the `Additional Instances` property of the deployed message flow in the BAR file: the broker starts additional copies of the message flow on separate threads, providing parallel processing. This option is the most efficient way of handling this situation, if you are not concerned about the order in which messages are processed.

If the message flow receives messages from a WebSphere MQ queue, you can influence the order in which messages are processed by setting the `Order Mode` property of the MQInput node:

- If you set `Order Mode` to By User ID, the node ensures that messages from a specific user (identified by the UserIdentifier field in the MQMD) are processed in guaranteed order. A second message from one user is not processed by an instance of the message flow if a previous message from this user is currently being processed by another instance of the message flow.
- If you set `Order Mode` to By Queue Order, the node processes one message at a time to preserve the order in which the messages are read from the queue. Therefore, this node behaves as though you have set the `Additional Instances` property of the message flow to zero.
- If you set `Order Mode` to `User Defined`, you can order messages by any message element, by setting an XPath or ESQL expression in the Order field location property. The node ensures that messages with the same value in the order field message element are processed in guaranteed order. A second message with the same value in the order field message element is not processed by an instance of the message flow if a previous message with the same value is currently being processed by another instance of the message flow.

  If the field is missing, an exception is raised, and the message is rolled back. NULL and empty values are processed separately, in parallel.

  If you set `Order Mode` to By User ID or `User Defined`, and the message flow uses transformation nodes, it is advisable to set the `Parse Timing` to Immediate.

For publish/subscribe applications that communicate with the broker over any supported protocol, messages for a particular topic are published by brokers in the same order as they are received from publishers (subject to reordering based on message priority, if applicable). Therefore each subscriber receives messages from a particular broker, on a particular topic, from a particular publisher, in the order that they are published by that publisher.

However, it is possible for messages, occasionally, to be delivered out of order. This situation can happen, for example, if a link in the network fails and subsequent messages are routed by another link.

If you need to ensure the order in which messages are received, you can use either the **SeqNum** (sequence number) or **PubTime** (publish time stamp) parameter on the **Publish** command for each published message, to calculate the order of publishing.

For more information about the techniques recommended for all MQI and AMI users, see the *Application Programming Reference* and *Application Programming Guide* sections in the WebSphere MQ Version 7 Information

Center online for programs written to the MQI, and the *WebSphere MQ Application Messaging Interface* book for programs written to the AMI.

The *WebSphere MQ Application Messaging Interface* book is available from the WebSphere MQ Library web page (listed under Version 5.3), or from SupportPac MA0F on the WebSphere MQ SupportPacs web page.

See also the *Publish/Subscribe User's Guide* section in the WebSphere MQ Version 7 Information Center online.

The broker does not provide message ordering for messages received across WebSphere MQ Web Services Transport.

Multiple copies of the message flow in a broker

You can also deploy several copies of the same message flow to different execution groups in the same broker. This option has similar effects to increasing the number of processing threads in a single message flow, although typically provides less noticeable gains.

This option also removes the ability to determine the order in which the messages are processed, because, if there is more than one copy of the message flow active in the broker, each copy can be processing a message at the same time, from the same queue. The time taken to process a message might vary, and multiple message flows accessing the same queue might therefore read messages from the input source in a random order. The order of messages produced by the message flows might not correspond to the order of the original messages.

Ensure that the applications that receive message from these message flows can tolerate out-of-order messages. Additionally, ensure that the input nodes in these message flows are suitable for deployment to different processes.

Copies of the message flow in multiple brokers

You can deploy several copies of the same message flow to different brokers. This option requires changes to your configuration, because you must ensure that applications that supply messages to the message flow can put their messages to the right input queue or port. You can often make these changes when you deploy the message flow by setting the message flow's configurable properties.

The scope of the message flow

You might find that, in some circumstances, you can split a single message flow into several different flows to reduce the scope of work that each message flow performs. If you do split your message flow, be aware that it is not possible to run the separate message flows in the same unit of work, and if transactional aspects to your message flow exist (for example, the updating of multiple databases), this option does not provide a suitable solution.

The following two examples show when you might want to split a message flow:

1. In a message flow that uses a RouteToLabel node, the input queue has become a bottleneck. You can use another copy of the message flow in a second execution group, but this option is not appropriate if you want all of the messages to be handled in the order in which they are shown on the queue. You can consider splitting out each branch of the message flow that starts with a Label node by providing an input queue and input node for each branch. This option might be

appropriate, because when the message is routed by the RouteToLabel node to the relevant Label node, it has some level of independence from all other messages.

You might also need to provide another input queue and input node to complete any common processing that the Label node branches connect to when unique processing has been done.

2. If you have a message flow that processes very large messages that take a considerable time to process, you might be able to:
   a. Create other copies of the message flow that use a different input queue (you can set this option up in the message flow itself, or you can update this property when you deploy the message flow).
   b. Set up WebSphere MQ queue aliases to redirect messages from some applications to the alternative queue and message flow.

   You can also create a new message flow that replicates the function of the original message flow, but only processes large messages that are immediately passed on to it by the original message flow, that you modified to check the input message size and redirect the large messages.

The frequency of commits

If a message flow receives input messages on a WebSphere MQ queue, you can improve its throughput for some message flow scenarios by modifying its default properties after you have added it to a BAR file. (These options are not available if the input messages are received by other input nodes; commits in those message flows are performed for each message.)

The following properties control the frequency with which the message flow commits transactions:

- `Commit Count`. This property represents the number of messages processed from the input queue before an MQCMIT is issued.
- `Commit Interval`. This property represents the time interval that elapses before an MQCMIT is started.

Related tasks:

"Optimizing message flow response times" on page 3264
You can use different solutions to improve message flow response times.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"Configurable message flow properties" on page 4020
When you add a message flow to a broker archive (BAR) file in preparation for deploying it to a broker, you can set additional properties that influence its run time operation. These properties are available for review and update when you

select the **Manage and Configure** tab for the broker archive file.

"Publish message" on page 6405

Related information:

WebSphere MQ Version 7 Information Center online

Customizing the z/OS environment

If you are planning to use a z/OS environment, consider whether to create your brokers on z/OS. You must also complete a number of tasks to configure your environment.

About this task

Although you might be installing only one broker initially, you might want to consider how the product will be used in your organization in a few years time. Planning ahead makes developing your WebSphere Message Broker configuration easier.

The following rules apply to the configuration:

- A broker requires access to a queue manager.
- A broker cannot share its queue manager with another broker.
- You cannot use WebSphere MQ shared queues to hold data related to WebSphere Message Broker as SYSTEM.BROKER queues, but you can use shared queues for your message flow queues.

You can find details of the WebSphere MQ queues that are created and used by WebSphere Message Broker on z/OS in "**mqsicreatebroker** command" on page 3831.

When planning to work in a z/OS environment, you must complete the following tasks:

- Create started task procedures for each broker that you plan to use. These procedures must be defined, in the started task table, with an appropriate user ID.
- Decide on your recovery strategy. As part of your systems architecture, you must have a strategy for restarting systems if they end abnormally. Common solutions are to use automation products like Tivoli NetView® for z/OS or the Automatic Restart Manager (ARM) facility. You can configure WebSphere Message Broker to use ARM.
- Plan for corequisite products, including UNIX System Services, Resource Recovery Services (RRS), WebSphere MQ, and Java.

  You might also want to include DB2 in your configuration, if your message flows access databases.
- Ensure that the runtime library system (RTLS) for the broker is turned off in the default options of the Language Environment® for the system. This setting is required because the broker code is compiled using `XPLINK`, and `XPLINK` applications cannot be started while RTLS is active.
- Collect broker statistics on z/OS.

See the following topics for more information:

- "z/OS customization overview" on page 592
- "Customizing UNIX System Services on z/OS" on page 598

- "WebSphere MQ planning for z/OS" on page 601
- "Resource Recovery Service planning on z/OS" on page 602
- "Defining the started tasks to z/OS Workload Manager (WLM)" on page 602
- "Automatic Restart Manager planning" on page 603
- "Mounting file systems" on page 604
- "Checking the permission of the installation directory" on page 606
- "Customizing the version of Java on z/OS" on page 607
- "Checking APF attributes of `bipimain` on z/OS" on page 607
- "Collecting broker statistics on z/OS" on page 608
- "Configuring an execution group address space as non-swappable on z/OS" on page 608

For an overview of how to create WebSphere Message Broker brokers, see "Creating a broker on WebSphere Message Broker for z/OS" on page 609.

Related tasks:

"Setting up z/OS security" on page 556
On z/OS, you must complete several security configuration tasks before WebSphere Message Broker can work correctly.

"Creating a broker on WebSphere Message Broker for z/OS" on page 609
An overview of how you create a broker on WebSphere Message Broker for z/OS.

"Using WebSphere MQ shared queues for input and output (z/OS)" on page 1546
On z/OS systems, you can define WebSphere MQ shared queues as input and output queues for message flows. You might need to serialize access to those messages.

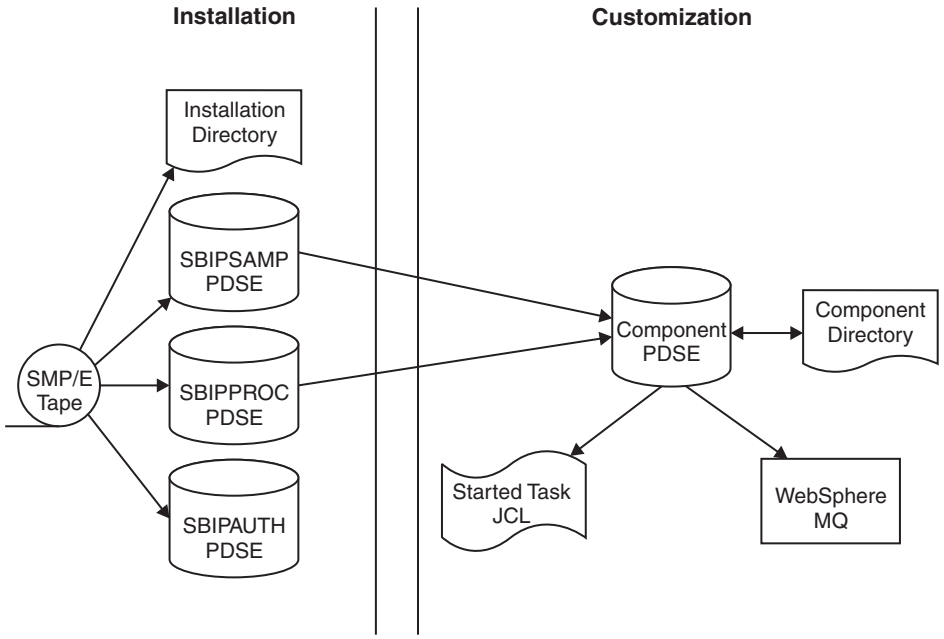
Related reference:

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

z/OS customization overview

After you have used SMP/E to install WebSphere Message Broker for z/OS, the installed executable code is located inside the file system. JCL samples are located in the PDS `<hlq>.SBIPSAMP`, the JCL procedures are located in the PDS `<hlq>.SBIPPROC`, and load module for synchronizing statistics with SMF are located in the PDS `<hlq>.SBIPAUTH`.

The following diagram gives an overview of the post-installation process.



To perform the customization, update and submit the required JCL. All necessary JCL is supplied to create the runtime environments of your broker. You start the broker using one of the supplied JCL files, which is run as a started task.

For more information, see:

- "Installation directory on z/OS"
- "Components on z/OS" on page 594
- "Component directory on z/OS" on page 594
- "Component PDSE on z/OS" on page 595
- "XPLink on z/OS" on page 595
- "Using the file system on z/OS" on page 596
- "Administration log messages on z/OS" on page 597
- "Security considerations on z/OS" on page 597
- Overview of message serialization on z/OS

Related tasks:

"Creating a broker on z/OS" on page 620
Create the broker component and the other resources on which it depends.

Installation directory on z/OS

On z/OS, the SMP/E installation places all the product executable files into a directory of a file system under UNIX System Services (USS).

For further information on mounting file systems and allocating space, including performance considerations, see "Mounting file systems" on page 604

Related concepts:

"z/OS customization overview" on page 592
After you have used SMP/E to install WebSphere Message Broker for z/OS, the installed executable code is located inside the file system. JCL samples are located in the PDS `<hlq>.SBIPSAMP`, the JCL procedures are located in the PDS `<hlq>.SBIPPROC`, and load module for synchronizing statistics with SMF are located

in the PDS <hlq>.SBIPAUTH.

"Component directory on z/OS"
The *component directory* is the root directory of the component's runtime environment.

Related tasks:

"Mounting file systems" on page 604
WebSphere Message Broker requires several directories to be defined on the file system at run time.

Components on z/OS

A *component* is a set of runtime processes that perform a specific set of functions, and comprises a broker.

A broker that is running has a control address space and one additional address space for each deployed execution group. When the control address space is started, the broker component is started automatically. This behavior can be changed by an optional start parameter in the started task.

The *component name* is the external name of the component and is used, for example, in the WebSphere Message Broker WebSphere Message Broker Toolkit.

Each component requires a name, which is usually the name of the started task that runs the component. This is typically the queue manager name with a suffix of the facility; for example, MQP1BRK for the broker.

Each component has its own runtime environment in UNIX System Services and needs its own WebSphere MQ queue manager.

Related concepts:

"z/OS customization overview" on page 592
After you have used SMP/E to install WebSphere Message Broker for z/OS, the installed executable code is located inside the file system. JCL samples are located in the PDS <hlq>.SBIPSAMP, the JCL procedures are located in the PDS <hlq>.SBIPPROC, and load module for synchronizing statistics with SMF are located in the PDS <hlq>.SBIPAUTH.

"Component directory on z/OS"
The *component directory* is the root directory of the component's runtime environment.

Component directory on z/OS

The *component directory* is the root directory of the component's runtime environment.

The *component directory* is also referred to as ComponentDirectory in some instances within the code. Both the WebSphere Message Broker administrator and the component require read and write access to the component directory.

An example directory for the broker component is /mqsi/brokers/MQP1BRK

For further information on mounting file systems and allocating space, including performance considerations, see "Mounting file systems" on page 604

Related concepts:

"z/OS customization overview" on page 592
After you have used SMP/E to install WebSphere Message Broker for z/OS, the installed executable code is located inside the file system. JCL samples are located in the PDS <hlq>.SBIPSAMP, the JCL procedures are located in the PDS <hlq>.SBIPPROC, and load module for synchronizing statistics with SMF are located in the PDS <hlq>.SBIPAUTH.

"Components on z/OS" on page 594

"Installation directory on z/OS" on page 593
On z/OS, the SMP/E installation places all the product executable files into a directory of a file system under UNIX System Services (USS).

Related tasks:

"Mounting file systems" on page 604
WebSphere Message Broker requires several directories to be defined on the file system at run time.

Component PDSE on z/OS

On z/OS, the *component PDSE* contains jobs customized for a single component. These jobs are used to create and administer the component.

The members specific to a component type are copied from<hlq>.SBIPSAMP and <hlq>.SBIPPROC to the component PDSE. These are then customized for the component.

The broker started-task user ID requires read access to its component PDSE at run time.

You must not share a PDSE across more than one *SYSplex* or *GRSplex*.

Related concepts:

"z/OS customization overview" on page 592
After you have used SMP/E to install WebSphere Message Broker for z/OS, the installed executable code is located inside the file system. JCL samples are located in the PDS <hlq>.SBIPSAMP, the JCL procedures are located in the PDS <hlq>.SBIPPROC, and load module for synchronizing statistics with SMF are located in the PDS <hlq>.SBIPAUTH.

"Components on z/OS" on page 594

"Component directory on z/OS" on page 594
The *component directory* is the root directory of the component's runtime environment.

"Installation directory on z/OS" on page 593
On z/OS, the SMP/E installation places all the product executable files into a directory of a file system under UNIX System Services (USS).

XPLink on z/OS

XPLink is a z/OS technology used by the C and C++ compilers to reduce the cost of function calling for programs written in these languages.

Many products, including WebSphere Message Broker for z/OS, use XPLink technology to improve their performance. To ensure the highest possible performance gains, WebSphere Message Broker requires as many as possible of the software components it uses to be XPLink-compliant. These include the broker, Java runtime, ODBC, and z/OS Language Environment.

The WebSphere Message Broker broker has been compiled by IBM to use XPLink technology and has been link-edited within the SMP/E environment to call the

appropriate XPLink routines of the software components it uses. Normally, these XPLink-enabled components are configured during their customization, and the broker needs only to locate the appropriate libraries to become XPLink-enabled.

Related concepts:

"z/OS customization overview" on page 592
After you have used SMP/E to install WebSphere Message Broker for z/OS, the installed executable code is located inside the file system. JCL samples are located in the PDS `<hlq>.SBIPSAMP`, the JCL procedures are located in the PDS `<hlq>.SBIPPROC`, and load module for synchronizing statistics with SMF are located in the PDS `<hlq>.SBIPAUTH`.

Related tasks:

"Customizing the version of Java on z/OS" on page 607
Check the version of Java in your enterprise, and change it, if necessary.

Using the file system on z/OS

If you have more than one MVS image, consider how you will use the file system. You can share files in a file system with different members of a sysplex. The file system is mounted on one MVS image and requests to the file are routed to the owning system using XCF from systems which do not have it mounted.

About this task

This is part of the larger task of customizing your z/OS environment.

Moving a broker from one image to another is straightforward and the files can be shared.

However, there is a performance overhead when using files shared between images in a file system because the data flows through the Coupling Facility (this is true for trace and other diagnostic data).

For further information on mounting file systems and allocating space, including performance considerations, see "Mounting file systems" on page 604

Space requirements:
About this task

For details of the disk space required, see "Disk space requirements on z/OS" on page 3586.

Related tasks:

"Customizing the z/OS environment" on page 591
If you are planning to use a z/OS environment, consider whether to create your brokers on z/OS. You must also complete a number of tasks to configure your environment.

"Mounting file systems" on page 604
WebSphere Message Broker requires several directories to be defined on the file system at run time.

Related reference:

"Disk space requirements on z/OS" on page 3586
The installation of WebSphere Message Broker for z/OS uses approximately 400 MB of disk space; plan on using 500 MB to allow for the component directories, and for new service fixes to be applied.

"Customization tasks and roles on z/OS" on page 3984
System programmers do most of the customization of WebSphere Message Broker.

Administration log messages on z/OS
About this task

This is part of the larger task of customizing your z/OS environment.

On z/OS, all address spaces have a job log where BIP messages issued by WebSphere Message Broker appear. Additionally, all messages appear on the `syslog` and important operator messages are filtered to the console using MPF (Message Processing Facility).

To prevent the operator's console receiving unnecessary BIP messages, you must configure MPF to suppress all BIP messages, with the exception of important messages. Note that you do not need to have the `USS SYSLOG` configured.

Related tasks:

"Customizing the z/OS environment" on page 591
If you are planning to use a z/OS environment, consider whether to create your brokers on z/OS. You must also complete a number of tasks to configure your environment.

Related reference:

"Customization tasks and roles on z/OS" on page 3984
System programmers do most of the customization of WebSphere Message Broker.

Security considerations on z/OS
About this task

This is part of the larger task of customizing your z/OS environment.

The role of the WebSphere Message Broker administrator includes customizing and configuring, running utilities, performing problem determination, and collecting diagnostic materials. People involved in these activities need WebSphere Message Broker authorities. You must set up some security for WebSphere Message Broker to work properly. The information that you need to do this is in "Setting up z/OS security" on page 556.

Related tasks:

"Customizing the z/OS environment" on page 591
If you are planning to use a z/OS environment, consider whether to create your brokers on z/OS. You must also complete a number of tasks to configure your environment.

"Setting up z/OS security" on page 556
On z/OS, you must complete several security configuration tasks before WebSphere Message Broker can work correctly.

"Setting up WebSphere MQ" on page 558
This is part of the larger task of setting up security on z/OS, and gives details of the authorities that your user ID needs to perform the required operations.

"Setting up WebSphere Message Broker Toolkit access on z/OS" on page 559
Access to the WebSphere Message Broker Toolkit is controlled from Windows or Linux on x86.

Related reference:

"Customization tasks and roles on z/OS" on page 3984
System programmers do most of the customization of WebSphere Message Broker.

Customizing UNIX System Services on z/OS

WebSphere Message Broker requires the setup of some UNIX System Services system parameters.

About this task

This task is part of the larger task of customizing your z/OS environment.

You can use the **SETOMVS** operator command for dynamic changes or the BPXPRMxx PARMLIB member for permanent changes. See the *z/OS UNIX System Services* section in the z/OS V1R9.0 LibraryCenter.

Use the **DISPLAY OMVS,OPTIONS** command to display your current OMVS options.

Do not include the broker addresses if you use the IEFUSI exit to limit the region size of OMVS address spaces.

Set the UNIX System Services parameters shown in the following table.

Description	Parameter	Value
The maximum core dump file size (in bytes) that a process can create. Allow an unlimited size.	MAXCORESIZE	2 147 483 647
The CPU time (in seconds) that a process is allowed to use. Allow an unlimited CPU time. If you do not set MAXCPUTIME to the maximum value, shown here, you can set it on the TIME parameter of the broker started task JCL. For more information, see *Statements and parameters for BIPXPRMxx* in the MVS Initialization Tuning and Reference of the z/OS V1R9.0 LibraryCenter.	MAXCPUTIME	2 147 483 647
The address space region size. Set to the size of the biggest address space.	MAXASSIZE	> 1 073 741 824 A minimum value of 393 216 000 bytes is required.
The maximum number of processes for each user.	MAXPROCUSER	Set to a number greater than all brokers and associated execution groups, as well as any queue managers and channel initiators that are running with the same user ID.
The maximum number of file descriptors that a single process can have open concurrently.	MAXFILEPROC	The number of descriptors includes all open files, directories, sockets, and pipes. A minimum value of 65536 is required.

Description	Parameter	Value
Specifies the maximum number of threads that a single process can have active. Depends on the definitions of message flows.	MAXTHREADS MAXTHREADTASKS	The value of MAXTHREADS and MAXTHREADTASKS depends on your application. To calculate the value needed for each message flow: 1. Multiply the number of input nodes by the number of instances (additional threads +1). 2. Sum the values of all message flows, then add 10 to the sum. 3. Add to the sum the number of threads used for each HTTP listener.

Deploying a message flow that starts an execution group in a new address space uses z/OS UNIX System Services Semaphore and SharedMemorySegment resources. Each new address space uses a semaphore and SharedMemorySegment. The SharedMemorySegment is deleted immediately after the new address space has started, but the semaphore remains for the life of the new address space.

Certain z/OS UNIX System Services system parameters can affect the start of a new execution group address space, if you set them incorrectly. These parameters include:

- IPCSEMNIDS
- IPCSHMNIDS
- IPCSHMNSEGS

You must have a minimum of three semaphores for each execution group address space that is started.

You must set IPCSEMNIDS to a value four times the number of potential execution group address spaces on a system.

You must have one SharedMemorySegment for each execution group address space that is started. You must set IPCSEMNIDS to a value that exceeds the number of potential execution group address spaces on a system.

A control address space (BIPSERVICE and BIPBROKER processes) can be attached to many SharedMemorySegments - potentially, one for each execution group address space started for that broker. You must set IPCSHMNSEGS to a value that exceeds the potential number of execution groups for each broker.

Ensuring sufficient space for temporary files

The environment variable TMPDIR is the path name of the directory being used for temporary files. If it is not set, the z/OS shell uses `/tmp`.

When starting WebSphere Message Broker components, sufficient space is required in the directory referenced by TMPDIR. In particular, Java requires sufficient space to hold all JAR files required by WebSphere Message Broker.

If you do not allocate sufficient space, the execution group address spaces abend with a 2C1 code.

Allow at least 50 MB of space in this directory for WebSphere Message Broker components. More space might be needed if you deploy large user-defined nodes or other JAR files to the broker component.

Defining WebSphere Message Broker files as shared-library programs

The shared library region is a z/OS feature that enables address spaces to share dynamic link library (DLL) files. This feature enables brokers to share DLLs rather than each broker having to load the DLLs individually. The shared DLLs can be broker DLLs, or DLLs from other products such as Java. The amount of real storage used by MVS, and the time it takes for each broker to start, can thus be greatly reduced.

If you plan to deploy to more than one execution group on z/OS, you can reduce the amount of storage required by the execution group address spaces. Set the shared-library extended attribute on the following files:

```
/usr/lpp/mqsi/bin/*
/usr/lpp/mqsi/lil/*
/usr/lpp/mqsi/lib/*
/usr/lpp/mqsi/lib/wbimb/*
```

To set the shared-library attribute, use the OMVS **extattr** command with the +1 option. For example:

```
extattr +l /usr/lpp/mqsi/bin/*
```

To find out if the shared-library extended attribute has been set, use the **ls -E** command. For example, use the command **ls -E bipimain** to generate the following response:

```
-rwxr-x---  a-l-  1 USER  GROUP   139264 Mar 15 10:05 bipimain
```

where l (lowercase L, as in a-l-) shows that the program is enabled to run in a shared address space.

For more information, on using the shared library extended attribute, see the z/OS V1R9.0 LibraryCenter .

The storage that is reserved for the shared library region is allocated in each broker address space when the first address space is started. The amount of storage that is allocated is controlled by the SHRLIBRGNSIZE parameter in z/OS, which is in the BPXPRMxx member of SYS1.PARMLIB. You can tune the amount of storage that is allocated for the shared library region. Investigate how much space you need, bearing in mind that other applications besides WebSphere Message Broker might be using the shared library region. Then, adjust the SHRLIBRGNSIZE parameter accordingly.

Use the following MVS command to check that you have enough SHRLIBRGNSIZE to contain all the shared-library programs that are to be used on the system:

```
DISPLAY OMVS,LIMITS
```

If there is unused space in the shared library region, modify SHRLIBRGNSIZE to reduce the amount of storage allocated for the shared library region.

Storage is carved out of the high end of private storage of each address space that loads a system shared library object. The amount of storage allocated from each of these address spaces is based on the value of the SHRLIBRGNSIZE parameter in

the BPXPRMxx PARMLIB member. If this value is too high, the storage set aside for mapping the shared library region might interfere with the private storage requirements of individual address spaces. Therefore, specify the minimum size that is required to contain all of the shared library programs that are to be used on the system.

Note: z/OS UNIX attempts to map the entire SHRLIBRGNSIZE into the private region, not just the portion that contains programs. If the private region is too small to map the entire SHRLIBRGNSIZE, this shared library region is not used. No message is issued to indicate what has happened.

Use the **SETOMVS** operator command to change SHRLIBRGNSIZE dynamically, or the BPXPRMxx PARMLIB member for permanent changes. See the *z/OS UNIX System Services* section in the z/OS V1R9.0 LibraryCenter.

Related tasks:

"Customizing the z/OS environment" on page 591
If you are planning to use a z/OS environment, consider whether to create your brokers on z/OS. You must also complete a number of tasks to configure your environment.

Related reference:

"Customization tasks and roles on z/OS" on page 3984
System programmers do most of the customization of WebSphere Message Broker.

Related information:

z/OS V1R9.0 LibraryCenter

WebSphere MQ planning for z/OS

This is part of the larger task of customizing your z/OS environment.

About this task

You are required to have a separate WebSphere MQ queue manager for each broker.

All WebSphere Message Broker for z/OS system queues are defined during customization.

Your queue manager must have a dead-letter queue. Check that this queue exists by using the WebSphere MQ command:

```
+cpf  DIS QMGR DEADQ
```

Check that the queue exists by using the command:

```
 +cpf DIS QL(name) STGCLASS
```

Then use the:

```
+cpf DIS STGCLASS(...)
```

to check the `STGCLASS` value is valid. If the queue manager does not have a valid dead-letter queue, you must define one.

Creating and deleting components on z/OS requires the command server on the WebSphere MQ queue manager to be started. This component is normally started automatically (refer to the see the *z/OS System Administration Guide* section of the WebSphere MQ Version 7 Information Center online for more details).

The command server requires a reply-to queue based on SYSTEM.COMMAND.REPLY.MODEL; by default, this model queue is defined as permanent dynamic. However, if you leave the queue defined in this way, each time you run a create or delete component command these reply-to queues remain defined to the queue manager. To avoid these queues persisting, you can set the SYSTEM.COMMAND.REPLY.MODEL queue as temporary dynamic.

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related tasks:

"Customizing the z/OS environment" on page 591
If you are planning to use a z/OS environment, consider whether to create your brokers on z/OS. You must also complete a number of tasks to configure your environment.

Related information:

WebSphere MQ Version 7 Information Center online

Resource Recovery Service planning on z/OS

About this task

This is part of the larger task of customizing your z/OS environment.

WebSphere Message Broker for z/OS uses Resource Recovery Service (RRS) to coordinate changes to WebSphere MQ and DB2 resources. Ensure it is configured and active on your system, because your broker cannot connect to DB2 unless RRS is active.

Refer to the following manuals for detailed information about RRS: *z/OS V1R5.0 MVS Setting Up a Sysplex* and *z/OS V1R5.0 MVS Programming: Resource Recovery SA22-7616*.

Related tasks:

"Customizing the z/OS environment" on page 591
If you are planning to use a z/OS environment, consider whether to create your brokers on z/OS. You must also complete a number of tasks to configure your environment.

Related reference:

"Customization tasks and roles on z/OS" on page 3984
System programmers do most of the customization of WebSphere Message Broker.

Defining the started tasks to z/OS Workload Manager (WLM)

This is part of the larger task of customizing your z/OS environment.

About this task

With z/OS workload management (WLM), you define performance goals and assign a business importance to each goal. You define the goals for work in business terms, and the system decides how much resource, such as CPU and storage, should be given to it to meet the goal. Workload Manager constantly monitors the system, and adapts processing to meet the goals.

WebSphere Message Broker automatically defines the JOBACCT token for each execution group started, so that the broker systems administrator can use WLM classification to map the JOBACCT token to specific Service and Report classes.

Each execution group can be assigned to a different Service and Report class. Whenever an execution group address space is started, the STEPNAME is assigned to the JOBACCT token.

When an execution group address space is started on z/OS, the STEPNAME is derived from the execution group label; the STEPNAME is defined as the last eight characters. Any characters that are not valid for a STEPNAME are replaced with the `@` character.

For example, an execution group address space with the label `MyExecutionGroup`, has a STEPNAME of `IONGROUP` and assigns this same value to the JOBACCT.

The STEPNAME/JOBACCT is not guaranteed to be unique across multiple execution group address spaces on the same LPAR. If you require a unique STEPNAME/JOBACCT across multiple execution groups, you must use a suitable execution group naming standard.

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related tasks:

"Customizing the z/OS environment" on page 591
If you are planning to use a z/OS environment, consider whether to create your brokers on z/OS. You must also complete a number of tasks to configure your environment.

Related reference:

"Customization tasks and roles on z/OS" on page 3984
System programmers do most of the customization of WebSphere Message Broker.

Automatic Restart Manager planning

This is part of the larger task of customizing your z/OS environment.

WebSphere Message Broker for z/OS allows you to register a component to the Automatic Restart Manager (ARM).

When customizing a component, you register it with ARM by editing the following environment variables in the component profile:

Description	Name
Switch that determines whether ARM is used (`YES` or `NO`).	`MQSI_USE_ARM`
ARM element name	`MQSI_ARM_ELEMENTNAME`
ARM element type	`MQSI_ARM_ELEMENTTYPE`

By default components do not register to ARM; the initial setting is `MQSI_USE_ARM=NO`. You can override this default value by setting `MQSI_USE_ARM=YES` and providing an ARM element name and type.

MQSI_ARM_ELEMENTNAME must be a maximum of eight characters in length, because WebSphere Message Broker adds a prefix of SYSWMQI. For example, if you supply the value MQP1BRK to ARM_ELEMENTNAME, the element you define in your ARM policy is SYSWMQI_MQP1BRK.

To enable automatic restart you must also:

- Set up an ARM couple data set.
- Define the automatic restart actions that you want z/OS to perform in an ARM policy.
- Start the ARM policy.

The following manuals provide detailed information about ARM couple data sets, including samples:

- *z/OS MVS Programming: Sysplex Services Guide*
- *z/OS MVS Programming: Sysplex Services Reference*
- *z/OS MVS Setting up a Sysplex*

You can access these manuals from the z/OS V1R8.0 LibraryCenter

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related tasks:

"Customizing the z/OS environment" on page 591
If you are planning to use a z/OS environment, consider whether to create your brokers on z/OS. You must also complete a number of tasks to configure your environment.

Mounting file systems

WebSphere Message Broker requires several directories to be defined on the file system at run time.

About this task

++HOME++ is the location of the environment file (ENVFILE) used to create the runtime environment in which WebSphere Message Broker runs.

++INSTALL++ refers to the WebSphere Message Broker installation directories.

++COMPONENTDIRECTORY++ is the location where all deployed configuration is written to and read from by the WebSphere Message Broker runtime libraries.

++JAVA++ is the location of the Java installation.

++MQPATH++ is the location of theWebSphere MQ installation.

Because WebSphere Message Broker can run in a shared file system sysplex environment, it is important for performance reasons that these directories are mounted locally to the LPAR in which WebSphere Message Broker is started. This is particularly important for startup performance; if the WebSphere Message Broker installation directories are not mounted locally, startup times can increase significantly.

You can check file system ownership from USS using the command **df -v**. For example, the following output shows that the broker installation file system is owned by MVS1:

```
/usr/lpp/mqsi/V7R0M0:>df -v .
Mounted on     Filesystem               Avail/Total    Files      Status
/usr/lpp/mqsi/V7R0M0 (OMVS.PLEXS.MQSI.V700.WBIMB) 7984/806400    4294966503
   Available
HFS, Read/Write, Device:89, ACLS=Y
File System Owner : MVS1        Automove=Y      Client=N
Filetag : T=off   codeset=0
```

To create a directory in an already mounted file system use the `mkdir` command. For example:

```
mkdir -p /mqsi/brokers/MQP1BRK
```

To mount a new file system, follow the instructions given in the *z/OS UNIX System Services Planning* manual.

From USS, use the following instructions:

```
mkdir -p /mqsi/brokers/MQP1BRK
mount -f MQSI.BROKER.MQP1BRK /mqsi/brokers/MQP1BRK
```

From TSO, use the following instructions:

```
ALLOCATE DATASET('MQSI.BROKER.MQP1BRK') DSNTYPE(HFS) SPACE(5,5) DIR(1) CYL
FREE DATASET('MQSI.BROKER.MQP1BRK')
MOUNT FILESYSTEM('MQSI.BROKER.MQP1BRK') TYPE(HFS)
      MOUNTPOINT('/mqsi/brokers/MQP1BRK')
```

Note that the preceding `ALLOCATE` command is an example; the dataset should be allocated the correct amount of storage as described in "Disk space requirements on z/OS" on page 3586

Related concepts:

"Component directory on z/OS" on page 594
The *component directory* is the root directory of the component's runtime environment.

"Installation directory on z/OS" on page 593
On z/OS, the SMP/E installation places all the product executable files into a directory of a file system under UNIX System Services (USS).

Related tasks:

"Customizing the z/OS environment" on page 591
If you are planning to use a z/OS environment, consider whether to create your brokers on z/OS. You must also complete a number of tasks to configure your environment.

"Using the file system on z/OS" on page 596
If you have more than one MVS image, consider how you will use the file system. You can share files in a file system with different members of a sysplex. The file system is mounted on one MVS image and requests to the file are routed to the owning system using XCF from systems which do not have it mounted.

Related reference:

"Disk space requirements on z/OS" on page 3586
The installation of WebSphere Message Broker for z/OS uses approximately 400 MB of disk space; plan on using 500 MB to allow for the component directories, and for new service fixes to be applied.

Checking the permission of the installation directory

This is part of the larger task of customizing your z/OS environment.

About this task

You must ensure that the appropriate user IDs, for example, the WebSphere Message Broker Administrator and all component Started Task user IDs, have READ and EXECUTE permission to the WebSphere Message Broker installation directory.

Using group access control to set these permissions.

Procedure

1. Display the permissions on the installation directory using the `ls` command.

   ```
   ls -l /usr/lpp/mqsi
   ```

   This command displays lines like the following response:

   ```
   drwxr-xr-x   2 TSOUSER  MQM     8192 Jun 17 09:54 bin
   ```

   In this example, MQM is the group associated with the directory. Those user IDs requiring permission to the directory must be a member of this group.

   This example also shows the permissions defined for the directory. User `TSOUSER` has `rwx` (READ, WRITE, and EXECUTE), group MQM has `rx`, and any other user ID has `rx`.
2. Ensure that the user IDs requiring permission have a group that matches that of the installation directory. Use the following command, where `userid` is the ID you want to check:

   ```
   id <userid>
   ```
3. If the installation directory does not have a valid group, use the command `chgrp` to set the group of the directory:

   ```
   chgrp -R <group> <pathname>
   ```

   For example:

   ```
   chgrp -R MQSI /usr/lpp/mqsi
   ```

   You must be the owner of the group or have superuser authority to use this command.
4. If the installation directory does not have the correct permissions for the group (READ / EXECUTE), use the command `chmod` to change the permissions:

   ```
   chmod -R g=rx <pathname>
   ```

   For example:

   ```
   chmod -R g=rx /usr/lpp/mqsi
   ```

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related tasks:

"Customizing the z/OS environment" on page 591
If you are planning to use a z/OS environment, consider whether to create your brokers on z/OS. You must also complete a number of tasks to configure your environment.

Related reference:

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

Customizing the version of Java on z/OS

Check the version of Java in your enterprise, and change it, if necessary.

About this task

This task is part of the larger task of customizing your z/OS environment.

For information about the levels of Java that WebSphere Message Broker supports, see WebSphere Message Broker Requirements. The 64 bit version of Java is required.

To check the current version of Java in a broker component, complete the following steps.

Procedure

1. Change to the `bin` directory in which you installed your Java product.

   For example, if Java is installed in `/usr/lpp/java`, change to the `/usr/lpp/java/bin` directory, and type `./java -fullversion`. You receive the following response, or a response with similar content:

```
java version "1.6.0"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.6.0)
Classic VM (build 1.6.0 J2RE 1.6.0 IBM z/OS Persistent Reusable VM build cm142-20040917
 (JIT enabled: jitc))
```

   The response in this example confirms that Java 1.6.0 is the current version, which is the minimum level required for this platform.
2. If you want to set, or change, the current Java version, edit `JAVAHOME` in the broker profile (`BIPBPROF`).

   For further details about the broker profile, see "Creating a broker on WebSphere Message Broker for z/OS" on page 609.
3. Run `BIPGEN` to re-create the broker `ENVFILE` and any execution group specific `ENVFILEs`
4. Restart the broker to pick up changes to the `ENVFILE`.

Related tasks:

"Customizing the z/OS environment" on page 591
If you are planning to use a z/OS environment, consider whether to create your brokers on z/OS. You must also complete a number of tasks to configure your environment.

"Creating a broker on WebSphere Message Broker for z/OS" on page 609
An overview of how you create a broker on WebSphere Message Broker for z/OS.

Checking APF attributes of `bipimain` on z/OS

This task is part of the larger task of setting up your z/OS environment.

About this task

Use the `extattr` command to display the attributes of the object `bipimain`. For example:

```
extattr /usr/lpp/mqsi/bin/bipimain
```

The attribute APF authorized must be set to YES. If it is not, use extattr +a bipimain to set this attribute. For example:

```
extattr +a /usr/lpp/mqsi/bin/bipimain
```

You must have the appropriate authorization to issue this command.

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related tasks:

"Customizing the z/OS environment" on page 591
If you are planning to use a z/OS environment, consider whether to create your brokers on z/OS. You must also complete a number of tasks to configure your environment.

Collecting broker statistics on z/OS

About this task

If you need to write broker statistics to SMF, you need to have the library <HLQ>.SBIPAUTH in your STEPLIB.

This library and all the libraries in the STEPLIB concatenation need to be APF authorized.

You can use the "**mqsichangeflowstats** command" on page 3744 with o=SMF for this purpose.

You must not share a PDSE across more than one *SYSplex* or *GRSplex*.

Related tasks:

"Customizing the z/OS environment" on page 591
If you are planning to use a z/OS environment, consider whether to create your brokers on z/OS. You must also complete a number of tasks to configure your environment.

Related reference:

"**mqsichangeflowstats** command" on page 3744
Use the **mqsichangeflowstats** command to control the accumulation of statistics about message flow operation.

Configuring an execution group address space as non-swappable on z/OS

About this task

Because broker execution groups run as processes in UNIX System Services they cannot be set as NOSWAP in the PPT.

Instead, you can set the following environment variables in the broker environment file so that some, or all, of the address spaces of the broker execution groups request that they become non-swappable by the system; see "Creating the environment file" on page 626 for further information on adding an environment variable to a broker.

```
MQSI_NOSWAP=yes
```

sets the address spaces of all the execution groups to be non-swappable.

```
MQSI_NOSWAP_egname=yes
```

issues a request to the system, for each execution group labelled `egname`, that the address space be set as non-swappable.

```
MQSI_NOSWAP_uuid=yes
```

issues a request to the system, for each execution group with the UUID labelled `uuid`, that the address space be set as non-swappable.

In order for the above requests to succeed, the broker's started task ID needs `READ` access to the `BPX.STOR.SWAP` facility class through their external security manager, for example, RACF.

When an application makes an address space non-swappable, it can cause additional real storage in the system to be converted to preferred storage. Because preferred storage cannot be configured offline, using this service can reduce the installation's ability to re-configure storage in the future.

Related tasks:

"Customizing the z/OS environment" on page 591
If you are planning to use a z/OS environment, consider whether to create your brokers on z/OS. You must also complete a number of tasks to configure your environment.

"Creating a broker on z/OS" on page 620
Create the broker component and the other resources on which it depends.

"Creating the environment file" on page 626
This is part of the larger task of creating a broker on z/OS.

Creating a broker on WebSphere Message Broker for z/OS

An overview of how you create a broker on WebSphere Message Broker for z/OS.

About this task

Before you start

Before starting this task, you must have installed:

- WebSphere MQ for z/OS, with the optional JMS feature applied; for example, mounted at `/usr/lpp/mqm`.
- WebSphere Message Broker for z/OS, with a broker file system mounted; for example, `/usr/lpp/mqsi`.

If you want to connect a WebSphere Message Broker Toolkit directly to your z/OS system, you *must* install the optional WebSphere MQ Client Attach feature.

If you do not have the WebSphere MQ Client Attach feature installed, you can connect the WebSphere Message Broker Toolkit through an intermediate queue manager.

Read through all the sub topics in the "Customizing the z/OS environment" on page 591 section, and follow the guidance within those topics.

Procedure

1. Determine the customization information for your environment. The following list of topics contains tables of information that needs to be gathered before you can proceed with creating a broker on z/OS. Complete the information that your enterprise requires.

   If necessary, discuss the requirements with your system and WebSphere MQ administrators.

   - Broker:
     - "Installation information - broker" on page 621
     - "Component information - broker" on page 622
2. Set up security for the started task user IDs. Start with "Setting up z/OS security" on page 556.
3. Create the broker by carrying out the tasks listed in "Creating a broker on z/OS" on page 620. Start with "Creating the broker PDSE" on page 622.

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related information:

WebSphere MQ Version 7 Information Center online

Configuring brokers

Create and configure the brokers that you want on the operating system of your choice.

Before you begin

Before you start:

Ensure that the following requirements are met:

- Your user ID has the correct authorizations to perform the task. The authorizations are defined in "Security requirements for administrative tasks" on page 3644.
- Windows On Windows: you have created a user ID to be used as the service user ID. You specify this ID when you create the broker; it is used to run the broker.

  For more information about user ID authorization and creation, refer to "Planning for security when you install WebSphere Message Broker" on page 353.
- You have initialized the command environment on distributed systems; see "Setting up a command environment" on page 213.

About this task

To set up a default configuration, or an environment for application development on Linux on x86 or Windows, see Chapter 6, "Configuring brokers for development environments," on page 563. If you are creating brokers for test and production environments, use the tasks included in this section.

When you have created your physical components, you can administer your test and production environments either by using the WebSphere Message Broker Explorer, or programmatically by using the Administration API (the CMP API).

This collection of tasks uses specific resource names and user IDs. These names are examples only; you can use your own names. Follow existing naming conventions for WebSphere MQ and other resources.

Procedure

- "Creating a broker"
- "Verifying brokers" on page 630

What to do next

Next: Continue your broker configuration tasks in the WebSphere Message Broker Explorer. For day to day administration of your brokers and associated resources, see Chapter 8, "Administering brokers and broker resources," on page 899.

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related tasks:

"Configuring brokers in the WebSphere Message Broker Explorer" on page 635
Configure your local and remote brokers by using the WebSphere Message Broker Explorer.

"Developing applications that use the Administration API" on page 956
Develop Java applications that use the Administration API (also known as the CMP API) to communicate with, deploy to, and manage brokers and their associated resources.

"Planning for security when you install WebSphere Message Broker" on page 353
The Installation Guide describes the security tasks that you must complete before, during, and after installation.

Related reference:

"Security requirements for administrative tasks" on page 3644
You can configure access permissions to govern which users and groups can manipulate objects in the broker network. Security requirements for administrative tasks depend on the platform that you use.

Creating a broker

You can create brokers on every platform that is supported by WebSphere Message Broker. The broker runs as a 64-bit application on all platforms except Linux on x86 and Windows on x86.

Before you begin

Before you start:

Complete the following tasks:

- Ensure that your user ID has the correct authorizations to perform the task. Refer to "Security requirements for administrative tasks" on page 3644.

- On distributed systems, you must set up your command-line environment before creating a broker, by running the product profile or console; refer to "Setting up a command environment" on page 213.
- On z/OS, you must create and start the queue manager for this broker before you create the component.

About this task

Create a broker by using the command line on the computer on which you have installed the broker component. On Windows and Linux on x86, you can alternatively use the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer to complete this task.

- You must give the broker a name that is unique on the local computer.
- Broker names are case sensitive on all supported platforms, except Windows.
- You must associate each broker with its own dedicated WebSphere MQ queue manager.
- Brokers can access only local queue managers, so you cannot create a broker on a queue manager that is on a remote system.

The mode in which your broker is working can affect the number of execution groups and message flows that you can deploy, and the types of node that you can use. See "Restrictions that apply in each operation mode" on page 3657.

To create a broker, follow the link for the appropriate platform. Alternatively, follow the link to create a broker using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer.

- Linux and UNIX
- Windows
- z/OS
- "Creating a broker using the WebSphere Message Broker Explorer" on page 618

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

"Operation modes" on page 48
The operation mode that you use for your broker is determined by the license that you purchase.

Related tasks:

"Configuring a WebSphere Message Broker as a WebSphere MQ service" on page 894
If you want to operate your WebSphere Message Broker as a WebSphere MQ service.

"Using WebSphere MQ trusted applications" on page 613
Configure a broker to run as a WebSphere MQ trusted application.

"Modifying a broker" on page 631
Modify a broker by using the command line on the system where the broker component is installed.

"Viewing broker properties" on page 927
You can view broker properties by using the **mqsireportbroker** command. You can also use the WebSphere Message Broker Explorer to view broker properties.

“Deleting a broker” on page 930
Delete a broker using the command line on the system where the broker component is installed.

“Considering security for a broker” on page 501
Consider several factors when you are deciding which users can execute broker commands, and which users can control security for other broker resources.

Related reference:

“**mqsicreatebroker** command” on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

“Security requirements for administrative tasks” on page 3644
You can configure access permissions to govern which users and groups can manipulate objects in the broker network. Security requirements for administrative tasks depend on the platform that you use.

“Support for 32-bit and 64-bit platforms” on page 3589
WebSphere Message Broker operates in 32-bit mode or 64-bit mode, on supported operating systems.

Using WebSphere MQ trusted applications

Configure a broker to run as a WebSphere MQ trusted application.

Before you begin

Before you start:

You must complete the following tasks:

- Ensure that your user ID is a member of the mqm group. On UNIX and Linux, specify the user ID mqm as the service user ID when you create the broker. On Windows, use any service user ID that is a member of mqm. Refer to “Security requirements for administrative tasks” on page 3644.
- Review the restrictions that WebSphere MQ places on trusted applications that apply to your environment. See the section "Connection to a queue manager using the MQCONNX call" in the *Application Programming Guide* section of the WebSphere MQ Version 7 Information Center online.

About this task

You can configure a broker to run as a trusted (fastpath) application on all supported platforms except z/OS, where the option is not applicable. If the broker is configured as a trusted application, it runs in the same process as the WebSphere MQ queue manager agent, and all broker processes benefit from an improvement in the overall system performance.

A broker does not run as a trusted application by default; you either create a trusted application by using the “**mqsicreatebroker** command” on page 3831, or modify an existing broker by using the “**mqsichangebroker** command” on page 3723.

Configuring a broker as a trusted application does not affect the operation of WebSphere MQ channel agents or listeners. For more information about running these as trusted applications, see the section "Running channels and listeners as trusted applications" in the *Intercommunication* section of the WebSphere MQ Version 7 Information Center online.

Take care when deploying user-defined nodes or parsers. Because a trusted application (the broker) runs in the same operating system process as the queue manager, a user-defined node or parser might compromise the integrity of the queue manager. Consider fully the restrictions that apply to your environment and test user-defined nodes and parsers in a non-trusted environment before deploying them in a trusted broker.

You can either configure a broker to run as a trusted application when you create it, or modify an existing broker.

Procedure

- To create a broker on a command line, run the **mqsicreatebroker** command with the `-t` flag, which specifies that the broker is created as a trusted application.

  For example, enter the following command to create a broker called MB7BROKER as a trusted application:

  ```
  mqsicreatebroker MB7BROKER -q MB7QMGR -i mqm -t
  ```

  See "Creating a broker" on page 611 for more detailed information about how to create a broker for your platform.
- To modify an existing broker:
  1. Run the **mqsistop** command on the command line to stop the broker.
  2. Run the **mqsichangebroker** command with the `-t` flag. For example, enter the following command to modify a broker called MB7BROKER to run as a trusted application:

     ```
     mqsichangebroker MB7BROKER -t
     ```

     See "Modifying a broker" on page 631 for more detailed information about how to modify a broker for your platform.
  3. Restart the broker by using the **mqsistart** command. The broker restarts with fastpath set.

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related tasks:

"Starting and stopping a broker" on page 921
Run the appropriate command to start or stop a broker.

"Creating a broker" on page 611
You can create brokers on every platform that is supported by WebSphere Message Broker. The broker runs as a 64-bit application on all platforms except Linux on x86 and Windows on x86.

"Modifying a broker" on page 631
Modify a broker by using the command line on the system where the broker component is installed.

Related reference:

"Security requirements for administrative tasks" on page 3644
You can configure access permissions to govern which users and groups can manipulate objects in the broker network. Security requirements for administrative tasks depend on the platform that you use.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

Related information:

WebSphere MQ Version 7 Information Center online

Creating a broker on Linux and UNIX systems

On Linux and UNIX systems, create brokers on the command line; on Linux on x86, you can also create brokers in the WebSphere Message Broker Toolkit.

Before you begin

Before you start:

- If you want to configure the broker as a WebSphere MQ trusted application, see "Using WebSphere MQ trusted applications" on page 613.
- Read "Considering security for a broker" on page 501.
- Check which operation mode you are licensed to use. If you do not set a mode, the automatic default is `enterprise` mode; see "Operation modes" on page 48.

About this task

When you create a broker, if the WebSphere MQ queue manager does not exist, the queue manager is automatically created. If WebSphere MQ Version 7.1, or later, has been selected for the queue manager, the channel auth security is automatically disabled.

If the specified queue manager already exists when creating the broker, then it is assumed that the user has applied the appropriate security configuration to meet their requirements, and therefore channel auth security is not disabled.

To create a broker:

Procedure

1. Ensure that you are logged in using a user ID that has authority to run the **mqsicreatebroker** command.
2. Run the **mqsiprofile** script to set up the command environment for the broker:

   ```
   . install_dir/bin/mqsiprofile
   ```

   You must run this script before you can run the WebSphere Message Broker commands.

   For more information, see "Setting up a command environment" on page 213.
3. The command **mqsicreatebroker** defaults to using the primary installation of WebSphere MQ, if present, for the creation of queue managers.

   If you are creating a broker on a queue manager that already exists, there is no need to configure your WebSphere MQ environment.
4. Use the **mqsicreatebroker** command to create the broker.

   For example, if you want to create a broker called MB7BROKER on a queue manager called MB7QMGR, enter the following command:

   ```
   mqsicreatebroker MB7BROKER -q MB7QMGR
   ```
5. To enable function that becomes available in WebSphere Message Broker fix packs, use the **-f** parameter on the **mqsichangebroker** command. For more information, see "**mqsichangebroker** command" on page 3723.

Results

You have created a broker.

What to do next

Next: Complete the following tasks:

1. Start the broker by using the **mqsistart** command.
2. Connect to the broker from the WebSphere Message Broker Toolkit, the WebSphere Message Broker Explorer, or a CMP API application.

When you have completed these tasks, you can create the resources that you want to associate with the broker; for example message flows. You can create and work with resources by using either the WebSphere Message Broker Toolkit or the CMP API.

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related tasks:

"Setting up a command environment" on page 213
After you have installed the product on one of the distributed systems, you must initialize the environment before you can use a runtime component or command.

"Using WebSphere MQ trusted applications" on page 613
Configure a broker to run as a WebSphere MQ trusted application.

"Developing applications that use the Administration API" on page 956
Develop Java applications that use the Administration API (also known as the CMP API) to communicate with, deploy to, and manage brokers and their associated resources.

"Controlling the functional level of WebSphere Message Broker" on page 51
You can control the functional level of your broker to enable new functionality added in WebSphere Message Broker fix packs.

Related reference:

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsistart** command" on page 3965
Use the **mqsistart** command to start the specified broker if all initial verification tests complete successfully.

Creating a broker on Windows

On Windows, you can create brokers on the command line.

Before you begin

Before you start:

- If you want to configure the broker as a WebSphere MQ trusted application, see "Using WebSphere MQ trusted applications" on page 613.
- Read "Considering security for a broker" on page 501.
- Check which operation mode you are licensed to use. If you do not set a mode, the automatic default is `enterprise` mode; see "Operation modes" on page 48.

About this task

When you create a broker, if the WebSphere MQ queue manager does not exist, the queue manager is automatically created. If WebSphere MQ Version 7.1, or later, has been selected for the queue manager, the channel auth security is automatically disabled.

If the specified queue manager already exists when creating the broker, then it is assumed that the user has applied the appropriate security configuration to meet their requirements, and therefore channel auth security is not disabled.

To create a broker by using the command line, complete the following steps:

Procedure

1. Open a WebSphere Message Broker command prompt for the runtime installation in which you want to create the broker. For more information about initializing the runtime environment, see "Command environment: Windows systems" on page 306.

   On Windows 7 and Windows Server 2008 systems, you must open a command console with elevated privileges. To open a command console with elevated privileges, use the **mqsicommandconsole** command. For more information, see "**mqsicommandconsole** command" on page 3830.
2. The command **mqsicreatebroker** defaults to using the primary installation of WebSphere MQ, if present, for the creation of queue managers.

   If you are creating a broker on a queue manager that already exists, there is no need to configure your WebSphere MQ environment.
3. Use the **mqsicreatebroker** command to create the broker.

   For example, if you want to create a broker called MB7BROKER on a queue manager called MB7QMGR, enter the following command:

   ```
   mqsicreatebroker MB7BROKER -i wbrkuid -a wbrkpw -q MB7QMGR
   ```

   where *wbrkuid* and *wbrkpw* are the user name and password under which the broker runs.

   For more information about the command options, see "**mqsicreatebroker** command" on page 3831.

Results

You have created a broker.

What to do next

Next: Complete the following tasks:

1. Start the broker by using the **mqsistart** command.
2. Connect to the broker from the WebSphere Message Broker Toolkit or a CMP API application.

When you have completed these tasks, you can create the resources that you want to associate with the broker; for example message flows. You can create and work with resources by using either the WebSphere Message Broker Toolkit or the CMP API.

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related tasks:

"Setting up a command environment" on page 213
After you have installed the product on one of the distributed systems, you must initialize the environment before you can use a runtime component or command.

"Using WebSphere MQ trusted applications" on page 613
Configure a broker to run as a WebSphere MQ trusted application.

"Developing applications that use the Administration API" on page 956
Develop Java applications that use the Administration API (also known as the CMP API) to communicate with, deploy to, and manage brokers and their associated resources.

Related reference:

"**mqsicommandconsole** command" on page 3830
Use the **mqsicommandconsole** command to launch an elevated command console from which commands that require elevation on Windows can be run.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsistart** command" on page 3965
Use the **mqsistart** command to start the specified broker if all initial verification tests complete successfully.

Creating a broker using the WebSphere Message Broker Explorer

On Linux on x86 or Windows, you can create brokers by using the WebSphere Message Broker Explorer.

Before you begin

Before you start:

If you are using the WebSphere Message Broker Toolkit, see "Creating a broker for a development environment" on page 569.

- On Windows, you must have administrator access rights to create brokers by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer. When creating a broker, you might be prompted to agree to the use of administrator rights or be prompted to enter an administrator user ID and password.
- If you want to configure the broker as a WebSphere MQ trusted application, see "Using WebSphere MQ trusted applications" on page 613.
- Read "Considering security for a broker" on page 501.
- Check which operation mode you are licensed to use. If you do not set a mode, the automatic default is `enterprise` mode; see "Operation modes" on page 48.

About this task

When you create a broker, if the WebSphere MQ queue manager does not exist, the queue manager is automatically created. If WebSphere MQ Version 7.1, or later, has been selected for the queue manager, the channel auth security is automatically disabled.

If the specified queue manager already exists when creating the broker, then it is assumed that the user has applied the appropriate security configuration to meet their requirements, and therefore channel auth security is not disabled.

To create a broker by using the WebSphere Message Broker Explorer:

Procedure

1. Right-click the Brokers folder, and click **New** > **Local Broker**. The Create Broker wizard is displayed.
2. Enter a name for the broker, and click **Next**.
3. Enter a name for the queue manager, or select an existing queue manager from the list of available local queue managers. You cannot create a broker on a queue manager that is already associated with a broker, or on a queue manager that is on a remote system.
4. Enter values for your user name and password. On Windows only, the default user ID is LocalSystem (for the Windows Local System Account). Local System is a special account and has no password, therefore the password entry field is unavailable.
5. You must create an execution group for the broker. By default, an execution group called default is created. If you want to create an execution group with a different name, enter the new name in **Create execution group**. You can create additional execution groups using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer after the broker is created.
6. Optional: On Windows, select whether to start the broker automatically when Windows starts.
7. Optional: Select **Enable administration security** to activate broker administration security to control which users can complete specific tasks against that broker and its resources.
8. The wizard starts processing your request. If the action succeeds, the wizard displays messages in the summary panel. If an action fails, the wizard reports the error in a message dialog.

   If you know what is causing the error, and can fix it, correct the error and click **Yes**. The wizard reissues the command. If you do not know what is causing the error, or you cannot fix it, click **No**. The wizard backs out any actions that have already completed and returns your system to its initial state.
9. Click **Finish** to close the wizard.

Results

You have created and started a broker.

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

Related tasks:

"Creating a broker" on page 611
You can create brokers on every platform that is supported by WebSphere Message Broker. The broker runs as a 64-bit application on all platforms except Linux on x86 and Windows on x86.

"Creating a default configuration" on page 564
Use the Default Configuration wizard to create and test a basic broker configuration.

"Setting up broker administration security" on page 368
Control the actions that users can request against a broker and its resources.

"Configuring brokers in the WebSphere Message Broker Explorer" on page 635
Configure your local and remote brokers by using the WebSphere Message Broker Explorer.

Chapter 8, "Administering brokers and broker resources," on page 899
Administering brokers and associated broker resources includes the tasks that you perform frequently to activate and manage those resources. Choose the method you prefer to administer your brokers and associated resources.

Creating a broker on z/OS

Create the broker component and the other resources on which it depends.

About this task

To create your broker, perform the following tasks in order:

1. "Collecting the information required to create a broker"
2. "Creating the broker PDSE" on page 622
3. "Creating the broker directory on z/OS" on page 623
4. "Customizing the broker component data set" on page 624
5. "Customizing the broker JCL" on page 625
6. "Creating the environment file" on page 626
7. "Creating an execution group specific environment file" on page 627
8. "Creating the broker component" on page 629
9. "Copying the broker started task to the procedures library" on page 629

What to do next

To enable function that becomes available in WebSphere Message Broker fix packs, use the **-f** parameter on the **mqsichangebroker** command. For more information, see "**mqsichangebroker** command" on page 3723.

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related tasks:

"Modifying a broker on z/OS" on page 634
Use the **mqsichangebroker** command on z/OS to modify your broker.

"Deleting a broker on z/OS" on page 933
Delete the physical broker component.

"Starting and stopping a broker on z/OS" on page 924
Run the appropriate command from SDSF to start or stop a broker.

"Controlling the functional level of WebSphere Message Broker" on page 51
You can control the functional level of your broker to enable new functionality added in WebSphere Message Broker fix packs.

Collecting the information required to create a broker:

This is part of the larger task of creating a broker on z/OS.

About this task

You must complete the information in each of the tables, at the following links, before continuing:

- "Installation information - broker"
- "Component information - broker" on page 622

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related tasks:

"Creating a broker on z/OS" on page 620
Create the broker component and the other resources on which it depends.

Related reference:

"Customization planning checklist for z/OS" on page 3991
Use the information contained in the following tables to make a note of the values to use when you customize your system variables on z/OS.

Installation information - broker:

Decide on the values for the list of the JCL variables for your system; an example installation value is provided for each one.

Collect the information shown in the Description column, and complete the values that you require for your particular system. You can see a complete list of the variables that you can customize in "z/OS JCL variables" on page 3994.

Description	JCL variable	Example installation value	Your installation value
Fully qualified name of the product's SBIPPROC data set	N/A	<hlq>.SBIPPROC	
Fully qualified name of the product's SBIPSAMP data set	N/A	<hlq>.SBIPSAMP	
File system directory where the product has been installed	++INSTALL++	/usr/lpp/mqsi	
The letter for the language in which you want messages shown.	++LANGLETTER++[2]	E (English)	E
Locale of environment where commands are run by submitting JCL	++LOCALE++	C	
Time zone of environment where commands are run by submitting JCL	++TIMEZONE++	GMT0BST	
Location of Java installation	++JAVA++	/usr/lpp/java/IBM/J1.6	
WebSphere MQ high-level qualifier	++WMQHLQ++	MQM.V700	

Notes:

1. See the WebSphere MQ documentation for a list of supported national languages.

Related tasks:

"Creating a broker on z/OS" on page 620
Create the broker component and the other resources on which it depends.

"Collecting the information required to create a broker" on page 620
This is part of the larger task of creating a broker on z/OS.

Related information:

WebSphere MQ Version 7 Information Center online

Component information - broker:

Decide on the details of your broker component.

Collect the information shown in the Description column and complete the values you require for your particular system. You can see a complete list of variables you can customize in "z/OS JCL variables" on page 3994.

Description	JCL variable	Example component value	Your component value
Home directory of the file system for the broker user ID	++HOME++	/u/mqp1usr/mqp1brk	
Queue Manager associated with the broker	++QUEUEMANAGER++	MQP1	
File system directory where the broker is to exist	++COMPONENTDIRECTORY++	/mqsi/brokers/MQP1BRK	
Broker name	++COMPONENTNAME++	MQP1BRK	
Data set where all JCL relevant to the broker is saved	++COMPONENTDATASET++	TESTDEV.BROKER.MQP1BRK	
Name of the Started Task JCL; can be a maximum of eight characters	++STARTEDTASKNAME++	MQP1BRK	
mqsicreatebroker options	++OPTIONS++	Any additional optional parameters for the **mqsicreatebroker** command	

Related tasks:

"Creating a broker on z/OS" on page 620
Create the broker component and the other resources on which it depends.

"Collecting the information required to create a broker" on page 620
This is part of the larger task of creating a broker on z/OS.

Creating the broker PDSE:

This is part of the larger task of creating a broker on z/OS.

About this task

Each broker requires a PDSE or a PDS. A PDSE is preferable to a PDS because free space is available without the need to compress the data set.

Create the broker PDSE, for example using option 3.2 on ISPF. The name of the PDSE must be the same as the JCL variable ++COMPONENTDATASET++. Allocate a data set that has the following characteristics:

- Eight directory blocks
- 15 tracks (or 1 cylinder) of 3390 DASD with a record format of fixed blocked 80
- A suitable block size (for example 27920)
- A data set type of `library`

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related tasks:

"Creating a broker on z/OS" on page 620
Create the broker component and the other resources on which it depends.

"Creating the broker directory on z/OS"
This is part of the larger task of creating a broker on z/OS.

Creating the broker directory on z/OS:

This is part of the larger task of creating a broker on z/OS.

About this task

Before you start

Before starting this task, you must have completed "Collecting the information required to create a broker" on page 620 and "Creating the broker PDSE" on page 622.

Procedure

1. Use the TSO command OMVS to get into OMVS.
2. Create the broker root directory using the command:

   ```
   mkdir -p <ComponentDirectory>
   ```

   The name of the directory must be the same as the JCL variable ++COMPONENTDIRECTORY++.
3. Display the contents of the directory, which is currently empty, using the command:

   ```
   ls -dl /var/wmqi/MQP1BRK
   ```
4. Display the permissions on the directory using the command:

   ```
   ls -al /var/wmqi/MQP1BRK
   ```
5. Ensure that the user ID of the person doing the customization has a group that matches the group of the directory. Use the following command, where `userid` is the ID you want to check:

   ```
   id <userid>
   ```
6. Check that the directory has a valid group, and that the group has rwx permissions. If they do not, use the following command to set the group of the directory:

   ```
   chgrp <group> <pathname>
   ```

   For example:

```
chgrp WMQI /var/wmqi/MQP1BRK
```

You must be the owner of the group, or have superuser authority, to use this command.

7. To give the group READ, WRITE, and EXECUTE access, use the following command:

```
chmod g=rwx <pathname>
```

For example:

```
chmod g=rwx /usr/wmqi/MQP1BRK
```

8. To display the amount of space used and available, use the following command:

```
df -P /var/wmqi/MQP1BRK
```

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related tasks:

"Creating a broker on z/OS" on page 620
Create the broker component and the other resources on which it depends.

"Creating the broker PDSE" on page 622
This is part of the larger task of creating a broker on z/OS.

"Checking the permission of the installation directory" on page 606
This is part of the larger task of customizing your z/OS environment.

Customizing the broker component data set:

This is part of the larger task of creating a broker on z/OS.

About this task

Before you start

Before starting this task, you must have completed "Collecting the information required to create a broker" on page 620.

Create the broker data set in TSO, identified by `++COMPONENTDATASET++`, as instructed in the following steps:

Procedure

1. Copy all the listed files from `<hlq>.SBIPSAMP` to `++COMPONENTDATASET++`.
2. Copy all the listed files from `<hlq>.SBIPPROC` to `++COMPONENTDATASET++`.

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related tasks:

"Creating a broker on z/OS" on page 620
Create the broker component and the other resources on which it depends.

"Collecting the information required to create a broker" on page 620
This is part of the larger task of creating a broker on z/OS.

Related reference:

"Contents of the broker PDSE" on page 3991
After you have successfully customized the broker, the broker PDSE members have been set up.

Customizing the broker JCL:

This subtask is part of the larger task of creating a broker on z/OS.

About this task

Before you start

Before starting this task, you must have completed "Customizing the broker component data set" on page 624.

All JCL has a standard header, which includes the following items:

- A brief description of its function.
- A description where further information can be found, relating to the function of the JCL.
- If appropriate, a topic number.
- The section listing the JCL variables themselves.

Each JCL file defines its own `STEPLIB`.

You can customize the files using an ISPF edit macro that you have to tailor, or you can change each of the PDSE members manually.

`BIPEDIT` is a REXX program that you can use to help you customize your JCL. After you have customized `BIPEDIT`, you can run this REXX program against the other JCL files to change their JCL variables.

When you update `BIPBPROF` (the broker profile), the changes are not accessible until you run `BIPGEN` to copy the profile to the file system and create the `ENVFILE`. You must run `BIPGEN` each time you update `BIPBPROF` for the changes to take effect. If you have execution group specific profiles, and you change either `BIPBPROF` or `BIPEPROF` (renamed to the execution group label), you must also run `BIPGEN`.

Procedure

1. Customize the renamed `BIPEDIT` file. Use the information you collected in:
   - "Installation information - broker" on page 621
   - "Component information - broker" on page 622
2. Activate the renamed `BIPEDIT` file before you customize other JCL files, by running the following TSO command:

   ```
   ALTLIB ACTIVATE APPLICATION(EXEC) DA('COMPONENTDATASET')
   ```

   where `'COMPONENTDATASET'` is identical to `++COMPONENTDATASET++`.

   This command is active for the local ISPF session for which it was issued. If you have split screen sessions, the other sessions are not able to use this command. If you use ISPF option 6 to issue the command, use ISPF option 3.4 to edit the data set, which enables you to use the edit command.
3. Edit each JCL file. Run the renamed `BIPEDIT` file by typing its name on the command line (for example `MQ01EDBK`). Instead of editing a member, you might want to `View` it until you have resolved all problems in your REXX program. Alternatively, you can `Cancel` the `Edit` session instead of saving it.

You must set a value for all the variables listed in the JCL; if you do not do so, the JCL does not work correctly.

What to do next

Some JCL files include ++OPTIONS++ for a command; you must replace them with additional optional parameters specific to the command on z/OS, or remove them. Typically, you must replace or remove these options in addition to running BIPEDIT. If you do not require additional options, remove ++OPTIONS++ by using the following command:

```
"c ++OPTIONS++ '' all"
```

where ' ' represents two single quotation marks.

Save the edit macro and run this macro against all the members except the edit macro itself.

If the user ID submitting the BIPCRBK command has the appropriate WebSphere MQ authorities, you can ignore the optional **mqsicreatebroker** parameters -1 and -2.

If you expect different administrators to create the WebSphere MQ resources, you can consider using one of these optional parameters; see "**mqsicreatebroker** command" on page 3831 for further information.

You must be aware that another process might be using the current ENVFILE, therefore you must consider whether updating the current ENVFILE in the file system might have other effects.

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related tasks:

"Creating a broker on z/OS" on page 620
Create the broker component and the other resources on which it depends.

Related reference:

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

Creating the environment file:

This is part of the larger task of creating a broker on z/OS.

About this task

Before you start

Before starting this task, you must have completed "Customizing the broker JCL" on page 625

Procedure

1. Review the BIPBPROF member. If you define parameters for all users, you can configure BIPBPROF to use these parameters.

Note that the BIPBPROF environment file is the main broker file, and it is used by default for any execution group address spaces, unless a specific execution group environment file exists.

For example, if the time zone option TZ is set as a system-wide parameter for all users, you can remove it from BIPBPROF.

2. Submit member BIPGEN. Review the job output and make sure that the environment file in the output contains the parameters that you expect.

   If you change BIPBPROF, or system-wide parameters, you must submit BIPGEN again to pick up the changes.

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related tasks:

"Creating a broker on z/OS" on page 620
Create the broker component and the other resources on which it depends.

"Customizing the broker JCL" on page 625
This subtask is part of the larger task of creating a broker on z/OS.

Creating an execution group specific environment file:

You can create an execution group specific environment file, which is used instead of the default broker environment file when you restart the execution group.

Before you begin

Before you start

Before starting this task, you must have completed "Customizing the broker JCL" on page 625

About this task

This task forms an optional step of the larger task of creating a broker on z/OS.

By default a broker has a single environment file (ENVFILE), which is created from BIPBPROF by submitting BIPGEN. The main broker address space and any execution group address spaces use this environment file.

You might find it necessary to have an execution group specific environment file and you can achieve this by carrying out the following procedure. In this situation, on restart, the execution group address space uses the specific environment file and not the default broker environment file.

This functionality is enabled only when the function level of the WebSphere Message Broker fix packs is set, for example, 7.0.0.7. To enable function that becomes available in WebSphere Message Broker fix packs, use the **-f** parameter on the **mqsichangebroker** command. For more information, see "**mqsichangebroker** command" on page 3723.

Procedure

1. Review the BIPBPROF member. If you define parameters required by the broker and all execution group address spaces, you can configure BIPBPROF to use these parameters.

2. Take a copy of BIPEPROF and rename the copy of the file to be the same as the execution group label in the component dataset, using the following rule.

   The name should equal the last eight characters of the execution group label. Note that any characters that are not valid for a STEPNAME are replaced with the @ character.

   Furthermore, if the first character of a STEPNAME is not an alphabetic character, that character is replaced with an A.

   For more information about characters that you can use, see "Administration in z/OS" on page 3979.

   For example, an execution group address space with the label MyExecutionGroup, has an execution group specific profile called IONGROUP.
3. Customize the renamed BIPEPROF file to include any environment variable settings specific to the execution group.
4. Edit BIPGEN, adding an additional step at the end for the new execution group specific profile, for example:

   ```
   //BIPEG01 EXEC PROC=BIPEGEN,EG=IONGROUP
   ```
5. Submit member BIPGEN to re-create the default broker environment file, and to create the new execution group specific environment file. Review the job output and make sure that the environment file in the output contains the parameters that you expect.

   If you change BIPBPROF, or the renamed BIPEPROF file, or system-wide parameters, you must submit BIPGEN again to pick up the changes.

What to do next

Multiple Execution Group specific environment files

If you have a requirement for multiple execution group specific files, carry out the following tasks:

1. Create an execution group specific profile for each, following the naming rules specified in Step 2 above.
2. Add additional steps to the end of BIPGEN giving each step a different name, and specifying the correct eight character execution group label.
3. Resubmit BIPGEN to create all the environment files.

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related tasks:

"Creating a broker on z/OS" on page 620
Create the broker component and the other resources on which it depends.

"Customizing the broker JCL" on page 625
This subtask is part of the larger task of creating a broker on z/OS.

"Creating the environment file" on page 626
This is part of the larger task of creating a broker on z/OS.

"Controlling the functional level of WebSphere Message Broker" on page 51
You can control the functional level of your broker to enable new functionality added in WebSphere Message Broker fix packs.

Creating the broker component:

When you are creating a broker on z/OS, one of the tasks is to create the broker component.

About this task

If the user ID submitting the BIPCRBK command has the appropriate WebSphere MQ authority, you can ignore the optional **mqsicreatebroker** parameters -1 and -2. If it is your intention to have different administrators create the WebSphere MQ resources, you can consider using one of these optional parameters; see "**mqsicreatebroker** command" on page 3831 for further information.

Procedure

1. Submit job BIPCRBK with option -1. This job creates the files and directories that are placed in the default storage group. You must run this job first; to run this job you need authority to access the broker root directory.
2. Edit BIPCRBK and submit the job with option -2. This job creates the WebSphere MQ queues. If you do not have the requisite authority, ask your WebSphere MQ system administrator to run the job.
3. Ensure that the jobs have run successfully by:

   Checking the STDOUT stream in the JOBLOG.

   Viewing STDOUT for any errors and checking for `BIP8071I: Successful command completion`.

What to do next

If you encounter any problems, delete the broker and re-create it using the following procedure. You must have the appropriate authority to run the jobs.

1. Edit and configure job BIPDLBK.
2. Run job BIPDLBK with the same option, or options, that caused the problems when you ran the BIPCRBK job.
3. Correct the problems and run the BIPCRBK job again.

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related tasks:

"Creating a broker on z/OS" on page 620
Create the broker component and the other resources on which it depends.

Related reference:

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

Copying the broker started task to the procedures library:

This is part of the larger task of creating a broker on z/OS.

About this task

Before you start

Before starting this task, you must have completed “Creating the broker component” on page 629.

Procedure

1. Ensure that the user ID for the broker started task is defined and that the broker procedure is associated with the user ID. If you are using a security manager, for example RACF, update the started class for your broker. See “Setting up z/OS security” on page 556 and “Summary of required access (z/OS)” on page 3985 for more information.
2. Copy the Started Task JCL (`BIPBRKP`) to the procedures library, for example `USER.PROCLIB`.

Related concepts:

“The broker environment” on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related tasks:

“Setting up z/OS security” on page 556
On z/OS, you must complete several security configuration tasks before WebSphere Message Broker can work correctly.

“Creating a broker on z/OS” on page 620
Create the broker component and the other resources on which it depends.

“Creating the broker component” on page 629
When you are creating a broker on z/OS, one of the tasks is to create the broker component.

“Starting and stopping a broker on z/OS” on page 924
Run the appropriate command from SDSF to start or stop a broker.

Related reference:

“Summary of required access (z/OS)” on page 3985
The professionals in your organization require access to components and resources on z/OS.

Verifying brokers

Use the **mqsilist** command to display the brokers that you have created on your computer.

About this task

If you run the **mqsilist** command with no parameters, a one line summary for every broker that you have created for the current installation is displayed. For example:

```
BIP1281I: Broker 'BrokerA' on queue manager 'QMA' is running.

BIP1281I: Broker 'MB7BROKER' on queue manager 'MB7QM' is running.

BIP1281I: Broker 'MBTEST1' on queue manager 'TESTQM' is stopped.

BIP8071I: Successful command completion.
```

You can request further details about all resources by specifying parameter **-d 1** or **-d 2** on the command, You can also specify the **-r** parameter so that the command recursively returns information about execution groups, and the message flows and files that you have deployed to those execution groups.

You can also use this command to display a list of brokers that you have created for all concurrent installations on this computer. For example, you might have installed both Version 6.0 and Version 7.0 for assessment and migration.

```
mqsilist -a
```

In earlier versions, only brokers that you have created are displayed. If you want to view information about other components, you must run the corresponding command for that version

Related reference:

"**mqsilist** command" on page 3882
Use the **mqsilist** command to list installed brokers and their associated resources.

Modifying a broker

Modify a broker by using the command line on the system where the broker component is installed.

Before you begin

Before you start:

You must have completed the following tasks:

- Ensure that your user ID has the correct authorizations to perform the task; see "Security requirements for administrative tasks" on page 3644.
- Create a broker.
- On Windows, Linux, and UNIX systems, you must set up your command-line environment before performing this task by running the product profile or console; see "Setting up a command environment" on page 213.

About this task

The parameters you can change on the broker affect the physical broker that was created by using the command line.

You can also modify the broker in the WebSphere Message Broker Explorer, where you can change broker properties, such as security settings.

Choose the appropriate task for your platform from the following links:

- "Modifying a broker on Windows, Linux, and UNIX systems" on page 632
- "Modifying a broker on z/OS" on page 634
- "Configuring brokers in the WebSphere Message Broker Explorer" on page 635

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related tasks:

"Creating a broker" on page 611
You can create brokers on every platform that is supported by WebSphere Message Broker. The broker runs as a 64-bit application on all platforms except Linux on x86 and Windows on x86.

"Deleting a broker" on page 930
Delete a broker using the command line on the system where the broker

component is installed.

"Starting and stopping a broker" on page 921
Run the appropriate command to start or stop a broker.

"Viewing broker properties" on page 927
You can view broker properties by using the **mqsireportbroker** command. You can also use the WebSphere Message Broker Explorer to view broker properties.

Related reference:

"Security requirements for administrative tasks" on page 3644
You can configure access permissions to govern which users and groups can manipulate objects in the broker network. Security requirements for administrative tasks depend on the platform that you use.

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

"**mqsireportbroker** command" on page 3919
Use the **mqsireportbroker** command to display broker registry entries.

Modifying a broker on Windows, Linux, and UNIX systems

Use the **mqsichangebroker** command on Windows, Linux, and UNIX to modify your broker.

Before you begin

Before you start:

You must have completed the following task for the appropriate platform:

- "Creating a broker on Linux and UNIX systems" on page 615
- "Creating a broker on Windows" on page 616

About this task

To modify a broker on Windows, Linux, and UNIX:

Procedure

1. Stop the broker by using the **mqsistop** command.
2. Enter the **mqsichangebroker** command, specifying the broker name and one or more parameters that you want to change.

   ```
   mqsichangebroker brokername
   <<-i ServiceUserID> -a ServicePassword>
   <-t | -n> <-l UserLilPath>
   <-g ConfigurationTimeout> <-k ConfigurationDelayTimeout>
   <-v StatisticsMajorInterval> <-P httpListenerPort> <-y ldapPrincipal>
   <-z ldapCredentials> <-c ICUconverterpath> <-x userExitPath>
   <-e activeUserExits> <-o operationMode>  <-f functionlevel>
   ```

   where:

   brokername
   Is the broker name.

   -i Is the service user ID that is used to run the broker (applicable on Windows only).

   -a Is the password for the service user ID (applicable on Windows only).

   -t Indicates that the broker runs as a WebSphere MQ trusted application.

-n Indicates that the broker must cease to run as a WebSphere MQ trusted application.

-l Indicates from where LIL (loadable implementation libraries) files are loaded.

-g Is the maximum time (in seconds) to allow a broker to process a deployed message.

-k Is the maximum time (in seconds) to allow a broker to process a minimum size deployed message.

-v Is the time (in minutes) for the duration of the interval for collecting statistics archive records.

-P Is the port that the broker HTTP listener will use.

The broker starts this listener when a message flow that includes HTTP nodes or Web Services support is started

-y Is the user principal for access to an LDAP directory.

-z Is the user password for access to LDAP.

-c Is a delimited set of directories to search for additional code page converters.

-x Is the path that contains the location of all user exits to be loaded for execution groups in this broker.

-e Is the list of active user exits.

-o Is the operation mode that the broker will use.

-f Indicates the maximum function level of your broker that you want to enable.

-s Indicates whether broker administration security is enabled.

For example:

- To change the user ID that is used to run the broker on Windows, enter the following command:

  ```
  mqsichangebroker MB7BROKER -i wbrkuid -a wbrkpw
  ```

- To change the configuration timeout, enter the following command:

  ```
  mqsichangebroker MB7BROKER -g 500
  ```

- To activate broker administration security, enter the following command:

  ```
  mqsichangebroker MB7BROKER -s active
  ```

For further information about these parameters, and more examples, see "**mqsichangebroker** command - Linux and UNIX systems" on page 3724 and "**mqsichangebroker** command - Windows systems" on page 3729.

3. Restart the broker by using the **mqsistart** command. The broker restarts with the new properties.

Results

You cannot change all the parameters with which you created a broker. If you cannot change a property by using the **mqsichangebroker** command, delete the broker and create another broker with the new properties.

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to

route, transform, and enrich in flight messages.

"Operation modes" on page 48
The operation mode that you use for your broker is determined by the license that you purchase.

Related tasks:

"Starting and stopping a broker" on page 921
Run the appropriate command to start or stop a broker.

"Creating a broker" on page 611
You can create brokers on every platform that is supported by WebSphere Message Broker. The broker runs as a 64-bit application on all platforms except Linux on x86 and Windows on x86.

"Deleting a broker" on page 930
Delete a broker using the command line on the system where the broker component is installed.

Related reference:

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

"**mqsistart** command" on page 3965
Use the **mqsistart** command to start the specified broker if all initial verification tests complete successfully.

"**mqsistop** command" on page 3972
Use the **mqsistop** command to stop the specified component.

Modifying a broker on z/OS

Use the **mqsichangebroker** command on z/OS to modify your broker.

Before you begin

Before you start:

You must have completed the following task:

- "Creating a broker on z/OS" on page 620

About this task

To modify a broker:

Procedure

1. Ensure that the broker is running.
2. Stop the broker components by issuing the following command:
   ```
   /F BROKERNAME, PC
   ```
3. When the broker has stopped, use the **MVS MODIFY** command with the **changebroker** parameters that you want to change. For example:
   ```
   /F <BROKERNAME>,cb g=100,k=200
   ```
4. Restart the broker components by issuing the following command:
   ```
   /F BROKERNAME, SC
   ```
   The broker now uses the changed parameters.

What to do next

You cannot change all the parameters with which you created a broker. If you cannot modify a parameter that you want to change by using the **changebroker** command, delete the broker and create a new one. By creating another broker you can redefine all the parameters.

You can change the following parameters:

g ConfigurationChangeTimeout

k InternalConfigurationTimeout

l UserLilPath

v StatisticsMajorInterval

P HTTPListenerPort

y LdapPrincipal

z LdapCredentials

c ICUConverterPath

x UserExitPath

e ActiveUserExits

f functionlevel

For further information about these parameters, see "**mqsichangebroker** command - z/OS" on page 3733.

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related tasks:

"Creating a broker on z/OS" on page 620
Create the broker component and the other resources on which it depends.

"Deleting a broker on z/OS" on page 933
Delete the physical broker component.

Related reference:

"START and STOP commands on z/OS" on page 3981

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

"**mqsistart** command" on page 3965
Use the **mqsistart** command to start the specified broker if all initial verification tests complete successfully.

"**mqsistop** command" on page 3972
Use the **mqsistop** command to stop the specified component.

Configuring brokers in the WebSphere Message Broker Explorer

Configure your local and remote brokers by using the WebSphere Message Broker Explorer.

Before you begin

If you are using systems other than Linux on x86 or Windows, you must first create your brokers on those systems; see “Configuring brokers” on page 610.

If you are using Windows, you must have administrator access rights to create brokers by using the WebSphere Message Broker Explorer. When creating a broker, you might be prompted to agree to the use of administrator rights or be prompted to enter an administrator user ID and password.

About this task

On Linux on x86 or Windows, use the WebSphere Message Broker Explorer to create local brokers, and to configure, modify, and administer your local and remote brokers.

To start the WebSphere Message Broker Explorer:

- Linux Open a command shell in which the **mqsiprofile** command has not been run and enter the `strmqcfg` command, or run `/usr/bin/strmqcfg`.
- Windows Click **Start** > **All Programs** > **IBM WebSphere MQ** > **WebSphere MQ Explorer**, or double-click the shortcut on your desktop labeled 'WebSphere MQ Explorer'. On Windows Vista and Windows Server 2008 systems, you must right-click **WebSphere MQ Explorer**, and select **Run as Administrator** to start the application with the appropriate privileges.

In the WebSphere Message Broker Explorer you can configure broker, execution group and message flow properties including security and configurable services. See the following tasks for more information:

- “Configuring broker properties in the WebSphere Message Broker Explorer” on page 637
- “Using the WebSphere Message Broker Explorer to work with configurable services” on page 644

Related concepts:

“WebSphere Message Broker Explorer” on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

Related tasks:

“Planning a broker environment” on page 580
When you start to plan your broker environment, you must first consider your resource naming conventions and the design of the WebSphere MQ infrastructure.

“Configuring brokers” on page 610
Create and configure the brokers that you want on the operating system of your choice.

Chapter 8, “Administering brokers and broker resources,” on page 899
Administering brokers and associated broker resources includes the tasks that you perform frequently to activate and manage those resources. Choose the method you prefer to administer your brokers and associated resources.

“Viewing message flow accounting and statistics data” on page 3300
You can use the Broker Statistics and Broker Statistics Graph views in the WebSphere Message Broker Explorer to view snapshot accounting and statistics data as it is produced by the broker.

“Changing WebSphere Message Broker Explorer preferences” on page 654
Change preferences in the WebSphere Message Broker Explorer.

Configuring broker properties in the WebSphere Message Broker Explorer

Brokers and associated resource such as execution groups and message flows have properties that can modified. Use the WebSphere Message Broker Explorer to configure the properties for your brokers, execution groups and message flows. Alternatively you can use the command line or the Administration API.

About this task

See the following tasks for properties on message flows:

- "Configuring description properties in the WebSphere Message Broker Explorer"
- "Configuring DataPower security settings" on page 639

See the following tasks for properties on execution groups:

- "Configuring description properties in the WebSphere Message Broker Explorer"
- "Configuring the flow debug port in the WebSphere Message Broker Explorer" on page 641
- "Configuring DataPower security settings" on page 639

See the following tasks for properties on brokers:

- "Configuring description properties in the WebSphere Message Broker Explorer"
- "Changing the operation mode of your broker using the WebSphere Message Broker Explorer" on page 642
- "Changing the broker security settings in the WebSphere Message Broker Explorer" on page 643
- "Configuring DataPower security settings" on page 639

Related tasks:

"Configuring brokers in the WebSphere Message Broker Explorer" on page 635
Configure your local and remote brokers by using the WebSphere Message Broker Explorer.

"Configuring brokers" on page 610
Create and configure the brokers that you want on the operating system of your choice.

Chapter 8, "Administering brokers and broker resources," on page 899
Administering brokers and associated broker resources includes the tasks that you perform frequently to activate and manage those resources. Choose the method you prefer to administer your brokers and associated resources.

Configuring description properties in the WebSphere Message Broker Explorer

Use the WebSphere Message Broker Explorer to configure the `Long Description` and `Short Description` on your brokers, execution groups and deployed message flows. Alternatively, you can use the Administration API to configure the `Long Description` properties.

About this task

The `Short Description` is stored and used by the WebSphere Message Broker Explorer. The `Long Description` is stored by the broker, and can be viewed within

the WebSphere Message Broker Explorer or Administration API. You can also use the Long Description to add keywords to your message flows.

To configure the Long Description and Short Description on your brokers, execution groups, or message flows:

Procedure

1. In the Navigator view, expand the Brokers folder.
2. Right-click the object with which you want to work, and click **Properties**. This object can be a broker, execution group, or message flow. The Properties window is displayed. The Long Description and Short Description are fields on the **General** tab.
3. Enter the values for the Long Description, Short Description, or both. If you are working with a message flow, you can add keywords into the Long Description field. When you define a keyword, you must follow certain rules to ensure that the information can be parsed. The following example shows some values that you might want to define in the Long Description property:

   ```
   $MQSI Author=John Smith MQSI$
   $MQSI Flow 1 Version=v1.3.2 MQSI$
   ```

   Do not use the following characters within keywords, because they cause unpredictable behavior:

   ```
   ^ $ . | \ < > ? + * = & [ ] ( )
   ```

   You can use these characters in the values that are associated with keywords; for example:

   - `$MQSI RCSVER=$id$ MQSI$` is acceptable
   - `$MQSI $name=Fred MQSI$` is not acceptable
4. Click **Apply** to update the description properties.
5. Click **OK** to exit the Properties window.

Results

The description properties for the selected object are now updated, and you can view these properties in the **Properties QuickView** for the object in the WebSphere Message Broker Explorer.

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

"Message flow version and keywords" on page 1445
When you are developing a message flow, you can define the version of the message flow as well as other key information that you want to be associated with it.

Related tasks:

"Configuring broker properties in the WebSphere Message Broker Explorer" on page 637
Brokers and associated resource such as execution groups and message flows have properties that can modified. Use the WebSphere Message Broker Explorer to configure the properties for your brokers, execution groups and message flows. Alternatively you can use the command line or the Administration API.

"Configuring brokers in the WebSphere Message Broker Explorer" on page 635
Configure your local and remote brokers by using the WebSphere Message Broker

Explorer.

Related reference:

"WebSphere Message Broker Explorer views" on page 6838
The WebSphere Message Broker Explorer can be used to manage and administer your brokers and deployed resources.

Configuring DataPower security settings

Use the DataPower Security wizard in the WebSphere Message Broker Explorer to configure an external DataPower appliance to handle the WS-Security Policy for your HTTP, HTTPS input, and SOAP nodes within your message flow. The DataPower box is configured to decrypt incoming messages to your flow and encrypt outgoing messages from your flow without requiring any changes to the message flows or broker configuration.

Before you begin

Before you start:

To use the DataPower Security wizard you must have access to the SYSTEM.DEF.SVRCONN channel on the broker's queue manager. Your clients must send their messages direct to the DataPower appliance on a Client port you specify.

About this task

The following steps are required to configure a DataPower appliance for WS-Security for your message flows:

- Select which HTTP(S)Input and SOAP nodes you want to configure your security for.
- Create a DataPower connection profile or edit an existing profile.
- Use or alter the default Policy Sets to specify your encryption and decryption WS- Security parameters.
- Specify which specific Crypto Keys to use from the DataPower box

On the DataPower appliance the following configuration is created after you run the DataPower Security wizard:

- An XML Firewall with optionally Back (for HTTPSInput Nodes) and Front (Client) SSL connection.
- An XML Firewall Policy consisting of a list of inbound/ request rules and an outbound/ response rule per HTTP Input or SOAP Node.
- Each inbound/ request rule consists of a decryption action with parameters specified from the Policy Set.
- Each outbound/ response rule consists of an encryption action with parameters specified from the Policy Set.

To configure DataPower security for your message flows:

Procedure

1. Right-click on the message flow or execution group with which you want to work, and click **Properties**. You can enable security handling on a single message flow containing HTTP, HTTPS input and SOAP nodes, or you can select an execution group to enable security handling for these nodes in all the message flows in the execution group.

2. In the Properties window, click **DataPower** on the left to open the DataPower tab.
3. Click **Configure Security** to open the Security on DataPower Appliance window. The HTTP, HTTPS input, and SOAP nodes from your message flows are displayed in the Flow Details table.
4. Select a Policy Set Binding from the list of options. If you select the No Policy Set Bindings option, no encryption or decryption nodes are specified in your policy rules. You can use this option as a test for the communication channels before applying a policy set binding. To create a policy set binding, click **Edit Policy Sets**. See "Policy Sets and Policy Set Bindings editor" on page 6841, for more information about the Policy Sets and Policy Set Bindings editor.
5. In the DataPower details section, select a User profile from the list of options. Click **Edit Profiles** to create or edit connection profiles. To create a profile:
   a. In the DataPower Connection Profiles window, click **Add**.
   b. Click in the relevant cell in the table to edit the values. You must provide a valid user name, domain, and the host name of your DataPower appliance.
   c. Click **Finish**. The new or edited profile is now available to select in the Security on DataPower Appliance window.
   d. Add a password for the profile in the Password field.

   You can also use the DataPower Connection Profiles window to import and export profiles in the WebSphere Message Broker Explorer on different machines.
6. You must now decide whether to create a new Policy or merge with an existing Policy. If you attempt to merge with a policy that does not exist, a new one is created. A merge adds request and response rules to your policy, but it does not overwrite any preexisting rules. A merge also does not alter your existing firewall settings.
7. Enter the name or names of your XML Firewalls, and the Client Ports on which your HTTP clients connect to your DataPower box.
8. Optional: Select the nodes to configure in the Flow Details section, and click **Next** to select XML Firewall SSL settings, Decryption, and Encryption rules for your DataPower device.
9. Click **Finish**. An attempt is made to connect to your domain on your DataPower box to retrieve your Crypto Profiles.
10. Click **Yes** to confirm that you want to alter the configuration of your DataPower appliance.

Results

You have configured DataPower security settings for your message flow or execution group.

Related concepts:

"WS-Security" on page 765
Web Services Security (WS-Security) describes enhancements to SOAP messaging to provide quality of protection through message integrity, message confidentiality, and single message authentication. WS-Security mechanisms can be used to accommodate a wide variety of security models and encryption technologies.

"Policy sets" on page 774
Policy sets and bindings define and configure your WS-Security requirements, supported by WebSphere Message Broker, for the SOAPInput, SOAPReply, SOAPRequest, SOAPAsyncRequest, and SOAPAsyncResponse nodes.

Related tasks:

"Configuring broker properties in the WebSphere Message Broker Explorer" on page 637
Brokers and associated resource such as execution groups and message flows have properties that can modified. Use the WebSphere Message Broker Explorer to configure the properties for your brokers, execution groups and message flows. Alternatively you can use the command line or the Administration API.

"Configuring brokers in the WebSphere Message Broker Explorer" on page 635
Configure your local and remote brokers by using the WebSphere Message Broker Explorer.

Configuring the flow debug port in the WebSphere Message Broker Explorer

Before you can debug a message flow, you must configure and enable the flow debug port. The flow debug port is also known as the Java debug port, and the JVM debug port.

About this task

Use the WebSphere Message Broker Explorer, or WebSphere Message Broker Toolkit to set the flow debug port value to enable the flow debugger to be connected to an execution group. Alternatively, you can use the command console or the Administration API to configure this property.

Using the WebSphere Message Broker Toolkit:
Procedure

1. Select the broker with which you want to work in the Brokers view.
2. Right-click the execution group which you want to work, and click **Launch Debugger**. If the flow debug port is already configured, the port number is displayed in the Enable window.
3. To change the flow debug port, click **Configure**.
4. Set a port number for the debug port. The port number must be a number that is not already in use.
5. Click **OK**. The execution group is stopped and restarted, and the flow debug port is configured to use the specified port.
6. If you want to continue to attach the debugger to the execution group for debugging, click **OK**. To exit the Enable window without attaching the debugger, click **Cancel**.

What to do next

To start debugging your message flows, switch to the Debug perspective. Right-click the execution group, and click **Terminate Debugger** to stop debugging on the selected execution group.

Using the WebSphere Message Broker Explorer:
Procedure

1. Expand the Brokers folder in the Navigator view.
2. Right-click the execution group which you want to work, and click **Properties**. The Properties window is displayed.
3. Click **Flow Debug Port** on the left to display the Flow Debug Port tab.
4. Set a port number for the debug port. The port number must be a number that is not already in use.

5. Click **Apply**.
6. Click **OK** to exit the Properties view.
7. Right-click the execution group, and click **Flow Debug Port** > **Enable** to enable debugging on the selected execution group.

What to do next

You can connect the flow debugger to your execution group by using the WebSphere Message Broker Toolkit, see the instructions in "Using the WebSphere Message Broker Toolkit" on page 641. Click **Flow Debug Port** > **Disable** to disable debugging on the selected execution group.

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

"Flow debugger overview" on page 3158
Use the flow debugger in the WebSphere Message Broker Toolkit to track messages through your message flows.

Related tasks:

"Configuring broker properties in the WebSphere Message Broker Explorer" on page 637
Brokers and associated resource such as execution groups and message flows have properties that can modified. Use the WebSphere Message Broker Explorer to configure the properties for your brokers, execution groups and message flows. Alternatively you can use the command line or the Administration API.

"Attaching the flow debugger to an execution group for debugging" on page 3160
Before you can debug your message flow, you must attach the flow debugger to the execution group where your flow is deployed, then start a debugging session.

Related reference:

"JVM parameter values" on page 3813
Select the objects and properties associated with the Java Virtual Machine (JVM) that you want to change.

Changing the operation mode of your broker using the WebSphere Message Broker Explorer

You must ensure that your brokers are running in the operation mode for which you have purchased a license. You can change the operation mode of your broker WebSphere Message Broker Explorer. Alternatively, you can use the **mqsimode** command to change the operation mode.

About this task

To change the operation mode of your broker by using the WebSphere Message Broker Explorer:

Procedure

1. Switch to the WebSphere Message Broker Explorer, and expand the Brokers folder in the Navigator view.
2. Right-click the broker which you want to work, and click **Properties**. The Properties window is displayed.
3. Enter the value for the mode of your broker in the Operation Mode field, in the **General** tab. The options you can set in this field are:
   - `enterprise`

- `starter`
- `adapter`
- `entry`

4. Click **Apply**.
5. Click **OK** to exit the Properties view.

Results

The operation mode of your broker is updated. You can see the operation mode of your broker in the Properties QuickView.

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

"Operation modes" on page 48
The operation mode that you use for your broker is determined by the license that you purchase.

Related tasks:

"Configuring broker properties in the WebSphere Message Broker Explorer" on page 637
Brokers and associated resource such as execution groups and message flows have properties that can modified. Use the WebSphere Message Broker Explorer to configure the properties for your brokers, execution groups and message flows. Alternatively you can use the command line or the Administration API.

Related reference:

"**mqsimode** command" on page 3899
Use the **mqsimode** command to configure and retrieve operation mode information.

Changing the broker security settings in the WebSphere Message Broker Explorer

Use the WebSphere Message Broker Explorer to configure security settings for your broker. Alternatively, you can use the command line to change these values.

About this task

You can use the WebSphere Message Broker Explorer to configure the following security settings on your broker:

- Cache Sweep Interval
- Cache Timeout
- Security Profiles
- Policy Sets

Procedure

1. In the WebSphere Message Broker Explorer, expand the Brokers folder in the Navigator view.
2. Right-click the execution group which you want to work with, and click **Properties**. The Properties window is displayed.
3. Click **Security** on the left to display the Security tab.
4. Enter a value in seconds for the **Cache Sweep Interval** field. The default value for this field is 300.

5. Enter a value in seconds for the **Cache Timeout** field. The default value for this field is 60.
6. Click **Apply** to update the Cache Sweep Interval and Cache Timeout properties.
7. Click **Security Profiles** to configure security profiles on your broker. See "Creating a security profile" on page 433, for more information about the creating Security Profiles.
8. Click **Policy Sets** See "Policy Sets and Policy Set Bindings editor" on page 6841, for more information about the Policy Sets and Policy Set Bindings editor.
9. Click **OK** to exit the Properties view.

Results

You have configured security settings for your broker.

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

"Security profiles" on page 387
A security profile defines the security operations that are to be performed in a message flow at SecurityPEP nodes and security enabled input and output nodes.

"Policy sets" on page 774
Policy sets and bindings define and configure your WS-Security requirements, supported by WebSphere Message Broker, for the SOAPInput, SOAPReply, SOAPRequest, SOAPAsyncRequest, and SOAPAsyncResponse nodes.

Related tasks:

"Configuring broker properties in the WebSphere Message Broker Explorer" on page 637
Brokers and associated resource such as execution groups and message flows have properties that can modified. Use the WebSphere Message Broker Explorer to configure the properties for your brokers, execution groups and message flows. Alternatively you can use the command line or the Administration API.

Related reference:

"WebSphere Message Broker Explorer views" on page 6838
The WebSphere Message Broker Explorer can be used to manage and administer your brokers and deployed resources.

"Parameter values for the securitycache component" on page 3815
Select the objects and properties associated with the securitycache component that you want to change.

Using the WebSphere Message Broker Explorer to work with configurable services

Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view, add, modify and delete configurable services.

Instead of defining properties on the node or message flow, you can create configurable services so that nodes and message flows can refer to them to find properties at run time. If you use this method, you can change the values of attributes for a configurable service on the broker, which then affects the behavior of a node or message flow without the need for redeployment.

Unless it is explicitly stated by the function that is using the configurable service, you need to stop and start the execution group, or the applicable message flows, for the change of property value to take effect.

You can create and name new configurable services, based on IBM defined templates. Alternatively, you can modify the existing IBM defined configurable services. If you modify an IBM defined configurable services, their default values are overwritten in the system registry.

Use the WebSphere Message Broker Explorer to complete the following tasks for configurable services:

- "Creating a new configurable service"
- "Viewing configurable services" on page 647
- "Modifying an IBM defined configurable service" on page 648
- "Modifying a configurable service" on page 649
- "Importing a configurable service" on page 650
- "Exporting a configurable service" on page 651
- "Deleting a configurable service" on page 652

Alternatively, you can use the runtime commands to work with configurable services, see "Configurable services" on page 1296.

For a full list of configurable services and their properties, see "Configurable services properties" on page 3766.

Related concepts:

"Configurable services" on page 1296
Configurable services are typically runtime properties. You can use them to define properties that are related to external services on which the broker relies; for example, an SMTP server or a JMS provider.

Related reference:

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

Creating a new configurable service

Use the WebSphere Message Broker Explorer to create a new configurable service to define properties for an external service on which the broker relies.

About this task

You can add a new configurable service to your broker based on an IBM defined configurable service template using the WebSphere Message Broker Explorer. You must provide a name for your configurable service, which must not duplicate an existing configurable service name.

As an alternative to creating a new configurable service, you can modify an IBM defined configurable service, see "Modifying an IBM defined configurable service" on page 648.

Not all configurable service types are available to select within WebSphere Message Broker Explorer. If you want to create a configurable service and the type is not listed, you must first create an instance of that type via the command line. For information about creating configurable services by using the command line, see "**mqsicreateconfigurableservice** command" on page 3849.

To add a new configurable service using the WebSphere Message Broker Explorer:

Procedure

1. In the Navigator view, expand the broker on which you want to add a new configurable service.
2. Right-click the Configurable Services folder and click **New** > **Configurable service**. The Configurable services window is displayed.
3. Enter a name for your configurable service.
4. Select the type of configurable service to create.
5. For some configurable services, you can select an IBM defined template to provide default values you can use or update. If appropriate, select the IBM defined template to use for your configurable service.
6. Enter values for the configurable service properties. The properties are populated with default values.
7. Click **OK** to create the new configurable service.

What to do next

The properties for the configurable service are not used by the broker until you restart the message flows that use the configurable service. You can stop and start the broker to ensure that the configurable service is available to all resources running on the broker.

Related concepts:

"Using the WebSphere Message Broker Explorer to work with configurable services" on page 644
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view, add, modify and delete configurable services.

Related tasks:

"Viewing configurable services" on page 647
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view the properties of configurable services defined on your broker.

"Modifying an IBM defined configurable service" on page 648
You can create a new configurable service for an external service on which the broker relies, by modifying predefined configurable services provided by IBM. Use the WebSphere Message Broker Explorer to view and modify existing configurable

services.

"Modifying a configurable service" on page 649
Use the WebSphere Message Broker Explorer to view and modify existing configurable services.

"Deleting a configurable service" on page 652
Use the WebSphere Message Broker Explorer to delete custom configurable services.

Related reference:

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

Viewing configurable services

Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view the properties of configurable services defined on your broker.

About this task

You can view the properties of configurable services that you have defined on the broker, or you can view the properties of IBM defined configurable services.

To view a configurable service:

Procedure

1. In the Navigator view, expand the broker on which you want to view the configurable services.
2. Expand the Configurable Services folder. If you want to view IBM defined configurable services, right-click the Configurable Services folder, and click **Show IBM Templates**.
3. Right-click the configurable service that you want to view, and click **Properties**. The Configurable services window is displayed, and shows the properties for the selected configurable service.
4. Click **OK** to close the Configurable services.

Related concepts:

"Using the WebSphere Message Broker Explorer to work with configurable services" on page 644
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view, add, modify and delete configurable services.

Related tasks:

"Creating a new configurable service" on page 645
Use the WebSphere Message Broker Explorer to create a new configurable service to define properties for an external service on which the broker relies.

"Modifying a configurable service" on page 649
Use the WebSphere Message Broker Explorer to view and modify existing configurable services.

"Modifying an IBM defined configurable service" on page 648
You can create a new configurable service for an external service on which the broker relies, by modifying predefined configurable services provided by IBM. Use the WebSphere Message Broker Explorer to view and modify existing configurable services.

"Deleting a configurable service" on page 652
Use the WebSphere Message Broker Explorer to delete custom configurable services.

Related reference:

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

Modifying an IBM defined configurable service

You can create a new configurable service for an external service on which the broker relies, by modifying predefined configurable services provided by IBM. Use the WebSphere Message Broker Explorer to view and modify existing configurable services.

About this task

You can modify IBM defined configurable services to add a new configurable service to your broker. You cannot change the name of an IBM defined configurable service, and you cannot delete it. If you want to use a custom name for a configurable service, or if you want to create different configurable services of the same type, see "Creating a new configurable service" on page 645.

To modify an IBM defined configurable service:

Procedure

1. In the Navigator view, expand the broker on which you want to modify an IBM defined configurable service.
2. Expand the Configurable Services folder.
3. Right-click the configurable service that you want to modify, and click **Properties**. The Configurable services window is displayed.
4. Enter values for the configurable service properties. The **Basic** tab shows the key properties for the configurable service. Additional properties to configure can be found on the **Advanced** tab. The properties are populated with default values.
5. Click **OK** to update the configurable service.

What to do next

The properties for the configurable service are not used by the broker until you restart the message flows that use the configurable service. You can stop and start the broker to ensure that the configurable service is available to all resources running on the broker.

Related concepts:

"Using the WebSphere Message Broker Explorer to work with configurable services" on page 644
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view, add, modify and delete configurable services.

Related tasks:

"Viewing configurable services" on page 647
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view the properties of configurable services defined on your broker.

"Creating a new configurable service" on page 645
Use the WebSphere Message Broker Explorer to create a new configurable service to define properties for an external service on which the broker relies.

"Modifying a configurable service"
Use the WebSphere Message Broker Explorer to view and modify existing configurable services.

"Deleting a configurable service" on page 652
Use the WebSphere Message Broker Explorer to delete custom configurable services.

Related reference:

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

Modifying a configurable service

Use the WebSphere Message Broker Explorer to view and modify existing configurable services.

About this task

You can modify all the properties of an existing configurable service, except for the name.

To modify a configurable service:

Procedure

1. In the Navigator view, expand the broker on which you want to modify a configurable service.
2. Expand the Configurable Services folder. If you want to modify an IBM defined configurable service that is not displayed, right-click the Configurable Services folder, and click **Show IBM Templates**. The predefined services are displayed in the Configurable Services folder.
3. Right-click the configurable service that you want to modify, and click **Properties**. The Configurable services window is displayed.
4. Enter values for the configurable service properties. The properties are populated with default values.
5. Click **OK** to update the configurable service.

What to do next

The properties for the configurable service are not used by the broker until you restart the message flows that use the configurable service. You can stop and start the broker to ensure that the configurable service is available to all resources running on the broker.

Related concepts:

"Using the WebSphere Message Broker Explorer to work with configurable services" on page 644
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view, add, modify and delete configurable services.

Related tasks:

"Viewing configurable services" on page 647
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view the properties of configurable services defined on your broker.

"Creating a new configurable service" on page 645
Use the WebSphere Message Broker Explorer to create a new configurable service to define properties for an external service on which the broker relies.

"Modifying an IBM defined configurable service" on page 648
You can create a new configurable service for an external service on which the broker relies, by modifying predefined configurable services provided by IBM. Use the WebSphere Message Broker Explorer to view and modify existing configurable services.

"Deleting a configurable service" on page 652
Use the WebSphere Message Broker Explorer to delete custom configurable services.

Related reference:

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

Importing a configurable service

Use the WebSphere Message Broker Explorer to import a configurable service from another broker.

About this task

The properties of a configurable service can be exported from a broker, and imported by a new broker. The properties of the configurable service are stored in a `.configurableservice` file.

If the configurable service is defined on a broker in the WebSphere Message Broker Explorer, you can drag the configurable service from the broker to a new broker.

Alternatively, to a import a configurable service from a `.configurableservice` file:

Procedure

1. In the Navigator view, expand the broker to which you want to import a configurable service.
2. Right-click the Configurable Services folder, and click **Import *.configurableservice**.
3. Click the configurable service files that you want to import, and click **OK**.

What to do next

You must restart the message flows that used the configurable service for the change to be recognized by the message flows. You can stop and start the broker to ensure that the configurable service is updated for all the resources running on the broker.

Related concepts:

"Using the WebSphere Message Broker Explorer to work with configurable services" on page 644
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view, add, modify and delete configurable services.

Related tasks:

"Viewing configurable services" on page 647
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view the properties of configurable services defined on your broker.

"Creating a new configurable service" on page 645
Use the WebSphere Message Broker Explorer to create a new configurable service to define properties for an external service on which the broker relies.

"Modifying an IBM defined configurable service" on page 648
You can create a new configurable service for an external service on which the broker relies, by modifying predefined configurable services provided by IBM. Use the WebSphere Message Broker Explorer to view and modify existing configurable services.

"Modifying a configurable service" on page 649
Use the WebSphere Message Broker Explorer to view and modify existing configurable services.

"Exporting a configurable service"
Use the WebSphere Message Broker Explorer to export a configurable service to use on another broker.

Related reference:

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

Exporting a configurable service

Use the WebSphere Message Broker Explorer to export a configurable service to use on another broker.

About this task

The properties of a configurable service can be exported from a broker, and imported by a new broker. The properties of the configurable service are stored in a `.configurableservice` file.

If the configurable service is defined on a broker in the WebSphere Message Broker Explorer, you can drag the configurable service from the broker to a new broker.

Alternatively, to export a configurable service to a `.configurableservice` file:

Procedure

1. In the Navigator view, expand the broker to which you want to import a configurable service.
2. Expand the Configurable Services folder.
3. Right-click the configurable service that you want to export, and then click **Export *.configurableservice**.
4. Enter a name for the `.configurableservice` file, and click **OK**.

Related concepts:

"Using the WebSphere Message Broker Explorer to work with configurable services" on page 644
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view, add, modify and delete configurable services.

Related tasks:

"Viewing configurable services" on page 647
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view the properties of configurable services defined on your broker.

"Creating a new configurable service" on page 645
Use the WebSphere Message Broker Explorer to create a new configurable service to define properties for an external service on which the broker relies.

"Modifying an IBM defined configurable service" on page 648
You can create a new configurable service for an external service on which the broker relies, by modifying predefined configurable services provided by IBM. Use the WebSphere Message Broker Explorer to view and modify existing configurable services.

"Modifying a configurable service" on page 649
Use the WebSphere Message Broker Explorer to view and modify existing configurable services.

"Importing a configurable service" on page 650
Use the WebSphere Message Broker Explorer to import a configurable service from another broker.

Related reference:

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

Deleting a configurable service

Use the WebSphere Message Broker Explorer to delete custom configurable services.

About this task

You can delete custom configurable services that you have defined on a broker, but you cannot delete the IBM defined configurable services on a broker, even if you have previously modified the IBM defined configurable service.

To delete a configurable service:

Procedure

1. In the Navigator view, expand the broker on which you want to delete a configurable service.
2. Expand the Configurable Services folder.
3. Right-click the configurable service that you want to delete, and click **Delete**.
4. Click **OK** to confirm you want to delete the configurable service.

What to do next

You must restart the message flows that used the configurable service for the change to be recognized by the message flows. You can stop and start the broker to ensure that the configurable service is updated for all the resources running on the broker.

Related concepts:

"Using the WebSphere Message Broker Explorer to work with configurable services" on page 644
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to

view, add, modify and delete configurable services.

Related tasks:

"Viewing configurable services" on page 647
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view the properties of configurable services defined on your broker.

"Creating a new configurable service" on page 645
Use the WebSphere Message Broker Explorer to create a new configurable service to define properties for an external service on which the broker relies.

"Modifying an IBM defined configurable service" on page 648
You can create a new configurable service for an external service on which the broker relies, by modifying predefined configurable services provided by IBM. Use the WebSphere Message Broker Explorer to view and modify existing configurable services.

"Modifying a configurable service" on page 649
Use the WebSphere Message Broker Explorer to view and modify existing configurable services.

Related reference:

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

Working with UserDefined configurable services

Use the WebSphere Message Broker Explorer to create, view, and modify a UserDefined configurable service.

About this task

For further information on using the WebSphere Message Broker Explorer with configurable services, see:

- "Creating a new configurable service" on page 645
- "Viewing configurable services" on page 647
- "Modifying a configurable service" on page 649

To add a new configurable service:

Procedure

1. Switch to the WebSphere MQ Explorer perspective.
2. In the Domains view, expand the broker on which you want to add a new configurable service.
3. Right-click the Configurable Services folder and click **New** > **Configurable service**. The Configurable Service window is displayed.
4. Enter a name for your configurable service.
5. Select `UserDefined` as the type of configurable service to create.
6. Enter the values you require for the configurable service properties, using the add and delete buttons that become enabled for UserDefined configurable services. There are no predefined values for a UserDefined configurable service.
7. Click **Finish** to create the new configurable service.

Related concepts:

"Using the WebSphere Message Broker Explorer to work with configurable services" on page 644
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view, add, modify and delete configurable services.

Related tasks:

"Viewing configurable services" on page 647
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view the properties of configurable services defined on your broker.

"Creating a new configurable service" on page 645
Use the WebSphere Message Broker Explorer to create a new configurable service to define properties for an external service on which the broker relies.

"Modifying a configurable service" on page 649
Use the WebSphere Message Broker Explorer to view and modify existing configurable services.

"Deleting a configurable service" on page 652
Use the WebSphere Message Broker Explorer to delete custom configurable services.

"Working with properties of a configurable service of type UserDefined at run time in a JavaCompute node" on page 987
Use the CMP API in a JavaCompute node to query, set, create, and delete properties dynamically at run time in configurable services that you have defined with type UserDefined.

Related reference:

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

Changing WebSphere Message Broker Explorer preferences

Change preferences in the WebSphere Message Broker Explorer.

About this task

You can change the following preferences to tailor how you work in the WebSphere Message Broker Explorer:

- Display resources in the Broker Explorer view.
- Warn before deleting Administration log events.
- Broker connection wait times.
- DataPower configuration properties.
- Service trace properties
- SSL parameters.

To change WebSphere Message Broker Explorer preferences:

Procedure

1. Start the WebSphere Message Broker Explorer.
2. Click **Window** > **Preferences**.
3. Expand the **Broker Explorer** category in the left pane.
4. Make your selections.
5. Click **OK**.

Related concepts:

"WebSphere Message Broker Toolkit" on page 31
The WebSphere Message Broker Toolkit is an integrated development environment and graphical user interface based on the Eclipse platform.

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

Related tasks:

"Configuring the workbench" on page 570
You can configure various settings in the workbench to suit your requirements and your working environment.

"Changing trace settings from the WebSphere Message Broker Explorer" on page 3549
Collect trace information, in addition to user and service trace, by selecting options in the WebSphere Message Broker Explorer.

Changing the operation mode of your broker

Change the operation mode in which your broker is working by using the **mqsimode** command.

About this task

You must change your broker configuration to ensure that your brokers are running in the operation mode for which you have purchased a license. You can change the mode to `starter`, `adapter`, `entry`, or `enterprise`.

When you view the broker in the WebSphere Message Broker Explorer, the current mode of each broker is displayed. If the resources deployed to a broker exceed the permitted amounts, the display indicates these violations. You can change the broker mode in the WebSphere Message Broker Explorer, see "Changing the operation mode of your broker using the WebSphere Message Broker Explorer" on page 642. Alternatively, you can use the **mqsimode** command to make any required changes.

Procedure

1. Open a command prompt.
   - Linux On Linux, run the **mqsiprofile** command to initialize the command environment.
   - Windows On Windows, click **Start** > **Programs** > **IBM WebSphere Message Broker 7.0** > **Command Console** to open a command console.
2. Run the **mqsimode** command with the **-o** parameter to change the mode of the broker, or without the **-o** parameter to view the current setting; see "**mqsimode** command" on page 3899.
3. Check for error messages. If you attempt to reconfigure the broker to a mode which is not sufficient for the deployed resources, the **mqsimode** command issues a warning indicating that changing the mode is not allowed. Resolve any violations, if required; see "Resolving problems that occur during deployment of message flows" on page 3440.
4. (Optional) Run the **mqsimode** command again to confirm that there are no violations.

Example

See further examples of changing the mode of your broker:

- "Example: Changing the operation mode of your broker" on page 3904
- "Example: Changing the Trial Edition to the full edition" on page 3903

What to do next

Related concepts:

"Operation modes" on page 48
The operation mode that you use for your broker is determined by the license that you purchase.

Related tasks:

"Checking the operation mode of your broker" on page 657
Use the **mqsimode** command to find out the operation mode of your broker.

"Moving from Trial Edition"
You want to convert from Trial Edition mode to an alternative edition.

"Setting up a command environment" on page 213
After you have installed the product on one of the distributed systems, you must initialize the environment before you can use a runtime component or command.

Related reference:

"Restrictions that apply in each operation mode" on page 3657
The operation mode in which your broker is working defines how many execution groups you can use, and which nodes are available.

"**mqsimode** command" on page 3899
Use the **mqsimode** command to configure and retrieve operation mode information.

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

Moving from Trial Edition

You want to convert from Trial Edition mode to an alternative edition.

Before you begin

Before you start: Contact your IBM representative to upgrade your license.

Procedure

1. Open a command prompt:
   - Linux On Linux and UNIX, run the **mqsiprofile** command to initialize the command environment.
   - Windows On Windows, click **Start** > **Programs** > **IBM WebSphere Message Broker 7.0** > **Command Console** to open a command console.
2. Use the **mqsimode** command to change the mode of each broker. Specify the mode that you require for each broker by setting the **-o** parameter (*Mode_Type*) to `enterprise`, `starter`, `entry`, or `adapter`. For example, to change a local broker to run in `enterprise` mode, run the following command:

   ```
   mqsimode  MB7BROKER –o enterprise
   ```

   where MB7BROKER is the name of your broker.

To change a remote broker, specify a configuration file that defines the broker (*brokername*.broker), or one or more connection parameters that identify the broker **-i**, **-p**, **-q**).

3. Restart all your brokers; see "Starting and stopping a broker" on page 921.

What to do next

Next: Ensure that you upgrade other required products (for example, WebSphere MQ) if you have trial versions of those products.

You must change your broker configuration to ensure that your brokers are running in the operation mode for which you have purchased a license; see "Changing the operation mode of your broker" on page 655.

Note: If you installed WebSphere Message Broker using the Trial Edition package, all brokers are created in trial mode by default and you have to follow these steps for every newly created broker. In order to create brokers with a default mode of operation other than trial, you have to uninstall the WebSphere Message Broker Trial Edition product and install the full WebSphere Message Broker product; see "Uninstalling" on page 331 and "Installing the Broker component" on page 267.

Related concepts:

"Operation modes" on page 48
The operation mode that you use for your broker is determined by the license that you purchase.

Related tasks:

"Checking the operation mode of your broker"
Use the **mqsimode** command to find out the operation mode of your broker.

"Changing the operation mode of your broker" on page 655
Change the operation mode in which your broker is working by using the **mqsimode** command.

"Setting up a command environment" on page 213
After you have installed the product on one of the distributed systems, you must initialize the environment before you can use a runtime component or command.

Related reference:

"Restrictions that apply in each operation mode" on page 3657
The operation mode in which your broker is working defines how many execution groups you can use, and which nodes are available.

"**mqsimode** command" on page 3899
Use the **mqsimode** command to configure and retrieve operation mode information.

Checking the operation mode of your broker

Use the **mqsimode** command to find out the operation mode of your broker.

About this task

Run the **mqsimode** command to report the mode that is being used by your broker. Use one form of the connection parameters to identify the broker that you want to check.

Running this command also reports all mode violations.

Example

For example:

```
mqsimode –i localhost –p 1414 –q MB7QMGR
```

where **-i** (IP address), **-p** (port), and **-q** (queue manager) parameters represent the connection details for the queue manager associated with your broker. If you have created the broker on the computer on which you run this command, you can specify the broker name instead.

If your broker is running in `starter` mode, and the name of your broker is *Broker_Name*, this command displays the following messages:

```
BIP1044: Connecting to the broker's queue manager...
BIP1807: Discovering mode information from broker 'Broker_Name'...
BIP1802: Broker 'Broker_Name' is in starter mode.
BIP8071: Successful command completion.
```

Related concepts:

"Operation modes" on page 48
The operation mode that you use for your broker is determined by the license that you purchase.

Related tasks:

"Changing the operation mode of your broker" on page 655
Change the operation mode in which your broker is working by using the **mqsimode** command.

Related reference:

"Restrictions that apply in each operation mode" on page 3657
The operation mode in which your broker is working defines how many execution groups you can use, and which nodes are available.

"**mqsimode** command" on page 3899
Use the **mqsimode** command to configure and retrieve operation mode information.

Advanced configuration

Change your broker configuration to use one or more of the advanced options available.

About this task

- If you are developing message flows that include resources such as the WebSphere Adapters nodes, IMS nodes, or the CICSRequest node, you must set up your broker environment to support these applications. Details of the setup that you require is in "Configuring for applications" on page 659.
- You can use WS-Security with your web services message flows to provide quality of protection through message integrity, message confidentiality, and single message authentication. See "WS-Security" on page 765.
- If you have been working with WebSphere Message Broker on z/OS, and want to move your configuration to a distributed platform, see "Moving from WebSphere Message Broker on a distributed system to z/OS" on page 818.
- WebSphere Message Broker supports a number of locales on each platform; details of what is supported, and how to use code converters, are provided in "Changing locales" on page 819.

Related tasks:

"Customizing the z/OS environment" on page 591
If you are planning to use a z/OS environment, consider whether to create your brokers on z/OS. You must also complete a number of tasks to configure your environment.

"Configuring brokers" on page 610
Create and configure the brokers that you want on the operating system of your choice.

"Changing the operation mode of your broker" on page 655
Change the operation mode in which your broker is working by using the **mqsimode** command.

Configuring for applications

If you are developing message flows that access external resources such as databases, Enterprise Information Systems, IMS, CICS Transaction Server for z/OS, or email servers, you must set up your broker environment to support these applications.

About this task

- "Configuring user databases"
- "Configuring global coordination of transactions (two-phase commit)" on page 697
- "Configuring properties to connect to external resources" on page 716
- "Configuring internal resources required by flows" on page 753

Related tasks:

"Configuring brokers" on page 610
Create and configure the brokers that you want on the operating system of your choice.

"Connecting to Enterprise Information Systems" on page 1912
Use WebSphere Adapters to communicate with Enterprise Information Systems (EIS) such as SAP, Siebel, PeopleSoft, and JD Edwards.

"Working with IMS" on page 2128
You can use the IMSRequest node to connect to IMS, a message-based transaction manager and hierarchical-database manager for z/OS.

"Working with CICS Transaction Server for z/OS" on page 2172
Use the CICSRequest node to connect to CICS Transaction Server for z/OS applications.

"Processing email messages" on page 1786
You can configure the EmailOutput node to deliver an email from a message flow to an email server that supports Simple Mail Transfer Protocol (SMTP). You can also configure the EmailInput node to retrieve an email from an email server that supports Post Office Protocol 3 (POP3) or Internet Message Access Protocol (IMAP).

Configuring user databases

Configure databases to hold application or business data that you can access from your message flows.

About this task

Databases that hold application or business data are known as *user databases*, and are read from and written to by nodes within the message flows that you deploy to one or more brokers in your domain.

In some situations, and for some applications, you might need to ensure the integrity of the data that you hold in user databases across multiple systems and resource managers by coordinating table updates and the writing to one database with the deletion of data in another. To achieve these goals, you must configure your databases, your brokers, and your message flows to be globally coordinated.

For more information about the requirement for, and set up of, user databases, and the restrictions that apply, see "Databases overview" on page 2109.

The process of making databases available has the following phases:

1. Optional: Create and configure user databases. If your message flows interact with databases, you must create and configure those databases ready for connection by the broker on behalf of the message flows. For user databases, you can configure ODBC and JDBC connections.
2. Optional: If your user databases contain critical information, coordinate their updates through a transaction manager.

   On distributed systems, the WebSphere MQ queue manager is the transaction manager that interacts with the resource managers (the database providers). On z/OS, RRS provides equivalent coordination.

To complete these phases:

Procedure

1. If you want to access user databases from your deployed message flows, create and configure the databases and the connections to them:
   a. Create the databases.
   b. Authorize access to the databases.
   c. Optional: If you want your databases to participate in globally coordinated transactions, configure the databases for global coordination.
2. On distributed platforms, create and configure connections to the databases that you have created:
   a. If your message flows use an ODBC connection to a database, enable an ODBC connection for that database. Repeat this step for each database that you want to access in this way.

      Note that you can use the **mqsicvp** command as an ODBC test tool; see "Enabling ODBC connections to the databases" on page 668 for further information.
   b. If your message flows use a JDBC connection to a database, enable a JDBC connection for that database. Repeat this step for each database that you want to access in this way.
3. On z/OS, connect to the database.
4. Optional: If you want your databases to participate in globally coordinated transactions, configure the environment for global coordination.

Related tasks:

"Configuring brokers" on page 610
Create and configure the brokers that you want on the operating system of your choice.

"Customizing the z/OS environment" on page 591
If you are planning to use a z/OS environment, consider whether to create your brokers on z/OS. You must also complete a number of tasks to configure your environment.

Related reference:

"Supported databases" on page 3591
You can optionally configure databases to contain data that is accessed by your message flows. Databases from IBM and other suppliers are supported at specific versions on supported operating systems.

Creating the user databases:

If your message flows create, update, retrieve, or delete application and business data in one or more user databases, create the databases before you deploy the message flows to a broker.

About this task

For information about which databases you can use, see "Supported databases" on page 3591.

To create the databases on distributed systems:

Procedure

1. Create the databases that you require. Discuss your requirements with your database administrator, or see the documentation supplied with the database product that you are using. Make sure that you run a database profile, if required for your platform and database manager.

   The following samples use databases, and provide outline instructions for creating DB2 and Oracle databases:

   - Airline Reservations
   - Data Warehouse
   - Error Handler
   - Message Routing
   - Simplified Database Routing
   - TLOG Processor (DB2 only)

   You might find it helpful to use the instructions in these samples as a starting point for database creation.

   You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.
2. If you are creating Sybase databases on AIX, run the Sybase profile before you run **mqsiprofile**. For more information about using **mqsiprofile** to initialize command environments on Linux and UNIX platforms, see "Command environment: Linux and UNIX systems" on page 310.

Results

You have now created a database for business data.

What to do next

Next: If you have been following the instructions in "Working with databases" on page 2109, the next task is "Authorizing access to user databases" on page 662.

Related reference:

"Supported databases" on page 3591
You can optionally configure databases to contain data that is accessed by your message flows. Databases from IBM and other suppliers are supported at specific versions on supported operating systems.

"Database locations" on page 3595
The broker can access databases set up on the local computer or on a remote server, subject to restrictions.

Authorizing access to user databases:

When you have created a user database, you must authorize the broker and its execution groups to access it.

Before you begin

Before you start: create the databases.

Use the **mqsisetdbparms** command to specify a user ID and password that the broker can use to access each database. If you want to define a default user ID and password that the broker can use if you have not defined specific values for a particular database, run the **mqsisetdbparms** command and set the **ResourceName** to the value `dsn::DSN` with your chosen default user ID and password.

If you have migrated your broker from a previous release, the broker accessed a database for its own use, and you might have defined the user ID and password used to access that database by specifying a database connection user ID and password with the **-u** and **-p** parameters on the **mqsicreatebroker** command. Alternatively, you might have used the broker service user ID and its password (specified with the **-i** and **-a** parameters on the same command). When you migrate the broker, these parameters are migrated and stored, and are used by the migrated broker for access to databases that do not have specific ID access defined.

The following values and order of preference are used by the broker:

1. First, on all platforms: The user ID and password that you have set for the specific database, by using the **mqsisetdbparms** and specifying the database in the **-n** parameter.
2. Second, on all platforms: The user ID and password that you have set for all other databases, by using the **mqsisetdbparms** and specifying `dsn::DSN` in the **-n** parameter.
3. Third, the values are platform-specific:
   a. Windows On Windows: The broker service ID and password that you specified on the **mqsicreatebroker** command.
   b. Linux UNIX On Linux and UNIX: The user ID `mqsiUser` and password `********` (these values are fixed).
   c. z/OS On z/OS: The user ID `""` and password `""`, which cause the connection to be made with the broker started task user ID.

On z/OS only, if you do not specify a password when you specify `dsn::DSN` in the **-n** parameter, the broker connects to the DB2 database with its started task user ID. The broker uses the user ID that you have specified on the command when it creates a fully-qualified SQL statement, such as in ESQL, for stored procedures. For non-stored procedure SQL statements, where the schema is not specified, DB2 uses the CURRENTSQLID value. If you have not specified a CURRENTSQLID value,

the broker uses the user ID that you specified on the **mqsisetdbparms** command. If you have not provided a user ID by using that command, DB2 uses the broker started task ID.

The user ID that the broker uses to access databases must have the following authorizations:

- The user ID must be authorized to connect to the database.
- The user ID must have appropriate privileges on the user database objects that are accessed by the message flow; for example, tables, procedures, and indexes.

If you expect to deploy message flows that participate in globally coordinated transactions to a broker, you must provide additional authorization. For more information, see "Configuring databases for global coordination of transactions" on page 665.

About this task

The way that you authorize access depends on the database manager that you are using, and the platform on which you have created the database. The instructions might also vary from release to release of a single database. Consult your database administrator, or see the documentation for the appropriate database when you perform this task.

The following sections provide examples of the steps that you can take to provide the required authorization for specific databases:

- "DB2 authorization"
- "Oracle authorization" on page 664

DB2 authorization:
About this task

To authorize access to a DB2 database, you can use either the DB2 Control Center or the DB2 command line:

Procedure

- To use the DB2 Control Center:
  1. Start the DB2 Control Center.
  2. Expand the object tree until you find the database that you created.
  3. Expand the tree under the database then click the **User and Group Objects** folder. The **DB Users** and **DB Groups** folders are displayed in the right pane.
  4. In the right pane, right-click the **DB Users** folder then click **Add**. The Add User notebook opens.
  5. From the list, click the user ID that you want to authorize to access the database (for example, `mqsiuid`). The user ID that you select must be the user ID that you specify to be used for access to this database when you run the **mqsisetdbparms** command. The user ID must exist on the operating system before you can select it; if it does not exist, define the user ID on the operating system.
  6. Select the appropriate options from the choices in the dialog box that is labeled **Choose the appropriate authorities to grant to the selected user** for the database. The following options are available:
     - Connect database
     - Create tables

- Create packages
- Register functions to run in database manager process

7. Click **OK**. The authorities are granted. The dialog box closes.
8. Close the DB2 Control Center.

- To use the DB2 command line:
  1. Open a DB2 command window.
  2. Connect to the database with a user ID that has DB2 system administration (SYSADM or DBADM) authority. Substitute the correct database and ID in the following example command:

     ```
     db2 connect to broker_db user SysAd_id
     ```

  3. Run the following command to grant the required privileges to the user ID that the broker will use to connect to the database. Substitute the correct ID for your broker in the following example command, if you are not using `mqsiuid`:

     ```
     db2 grant connect, createtab, bindadd, create_external_routine on
        database to user mqsiuid
     ```

What to do next

Next: If you have been following the instructions in "Working with databases" on page 2109, the next task is "Configuring databases for global coordination of transactions" on page 665.

Oracle authorization:
About this task

You must have database administrator (DBA) privileges to authorize access to an Oracle database.

To authorize access to an Oracle database:

Procedure

1. Log on as the Oracle database administrator (DBA) to the database using SQL*Plus.
2. Modify the privileges of the user ID that you have specified for database connection to ensure that the broker can successfully access the database.
3. If appropriate, increase the quota (disk space) available for table spaces associated with this database.

What to do next

Next: If you have been following the instructions in "Working with databases" on page 2109, the next task is "Configuring databases for global coordination of transactions" on page 665.

Related concepts:

"Message flow transactions" on page 1281
A transaction describes a set of updates that are made by an application program, which must be managed together. The updates might be made to one or more systems. The updates made by the program are controlled by the environment in which the program executes, and either all are completed, or none. This property of a transaction is known as *consistency*: transactions might have other properties of *atomicity*, *isolation*, and *durability*.

Related tasks:

"Enabling ODBC connections to the databases" on page 668
Set up the resources and environment that the broker requires for Open Database Connectivity (ODBC) connections to user databases on distributed systems.

"Enabling JDBC connections to the databases" on page 683
Configure connections to a user database through a JDBCProvider service.

"Configuring databases for global coordination of transactions"
If your message flow interacts with a user database, and you want to globally coordinate the updates made to the database with other actions within the message flow, configure your databases for global coordination.

Related reference:

"Supported databases" on page 3591
You can optionally configure databases to contain data that is accessed by your message flows. Databases from IBM and other suppliers are supported at specific versions on supported operating systems.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

Related information:

DB2 V9.1 Information Center (distributed systems)

Configuring databases for global coordination of transactions:

If your message flow interacts with a user database, and you want to globally coordinate the updates made to the database with other actions within the message flow, configure your databases for global coordination.

Before you begin

Before you start: Create your database and authorize access to it.

About this task

If you restart a user database while the broker is still running, you must also restart the broker. The broker cannot detect that the database has stopped, and WebSphere MQ therefore retains its old connections to the database. When the database starts again, the broker tries, and fails, to use these connections.

To configure databases for coordinated message flows, follow the instructions relevant to your database manager:

- DB2
- Oracle
- Sybase

Configuring DB2 for global coordination of transactions:
Before you begin

You must complete these steps for databases that you connect to with an ODBC or a JDBC connection.

You must have database administrator (DBA) privileges to perform the following tasks.

About this task

To configure DB2 database instances for global coordination of transactions:

Procedure

1. Windows Linux Windows and Linux on x86 systems only: for each 32-bit instance that is involved in the global coordination, run the following commands to set the Transaction Process Monitor name (TP_MON_NAME) to MQ:

   ```
   db2 update dbm cfg using TP_MON_NAME MQ
   db2stop
   db2start
   ```

   UNIX Linux On Linux and UNIX systems (except for Linux on x86), do not set this variable for 32-bit or 64-bit instances.
2. Ensure that you have adequate connection resources and find out from the broker administrator whether the broker uses TCP/IP or shared memory to connect to databases.

   To use TCP/IP connections, see the example in the section about message SQL1224N in "Resolving problems when using databases" on page 3491.

   To enable extended shared memory:

   a. On the DB2 server, run the following commands:

      ```
      export EXTSHM=ON
      db2set DB2ENVLIST=EXTSHM
      db2stop
      db2start
      ```

   b. Ensure that shared memory support is enabled in the broker environment. For more information, see "Configuring global coordination with DB2" on page 700.
3. If you are connecting a broker on a distributed platform to a DB2 instance on z/OS, you must configure DB2 Connect™ to enable support for global coordination. Ensure that you have already configured a DB2 alias to represent the database by using DB2 Connect.

   Perform the following tasks on the system that hosts the broker:

   a. Turn on the Connection Concentrator by configuring the DB2 database manager configuration parameters so that the value of the **MAX_CONNECTIONS** parameter is greater than the value of the **MAX_COORDAGENTS** parameter:

      ```
      db2 update dbm cfg using MAX_CONNECTIONS max_connections_value
      ```

      where *max_connections_value* is greater than the existing value of the **MAX_COORDAGENTS** parameter.

   b. Define the SPM name as the name of the system that hosts the broker:

      ```
      db2 update dbm cfg using SPM_NAME host_name
      ```

      where *host_name* is the TCP/IP name of the system that hosts the broker.

   c. Stop, then restart DB2 on the system that hosts the broker to apply the changes:

      ```
      db2stop
      db2start
      ```

DB2 Connect is now configured to enable global coordination of message flows that are deployed to the broker (on a distributed platform) and that access DB2 on z/OS.

Results

The DB2 database instances are now configured for global coordination.

What to do next

Next: See "Configuring ODBC connections for globally coordinated transactions" on page 699.

Configuring Oracle for global coordination of transactions:
Before you begin

You must complete these steps for databases that you connect to with an ODBC connection only.

You must have database administrator (DBA) privileges to perform the following tasks.

About this task

To configure Oracle databases for global coordination of transactions:

Procedure

1. Ensure that the JAVA_XA package is present on the Oracle database by using, for example, the following Oracle SQLPLUS command:

   ```
   describe JAVA_XA;
   ```

   For more information, see the Oracle product documentation.
2. Ensure that the user ID that the broker uses to access the database has the necessary Oracle privileges to access the DBA_PENDING_TRANSACTIONS view. You can grant the required access by using, for example, the following Oracle SQLPLUS command:

   ```
   grant select on DBA_PENDING_TRANSACTIONS to userid;
   ```

Results

The Oracle databases are now configured for global coordination.

What to do next

Next: See "Configuring ODBC connections for globally coordinated transactions" on page 699.

Configuring Sybase for global coordination of transactions:
Before you begin

You must complete these steps for databases that you connect to with an ODBC connection only.

You must have database administrator (DBA) privileges to perform the following tasks.

Procedure

To configure Sybase databases for global coordination of transactions, ensure that the user ID that the broker uses to access the database has been granted the Sybase role of `dtm_tm_role`.

Results

The Sybase databases are now configured for global coordination.

What to do next

Next: See "Configuring ODBC connections for globally coordinated transactions" on page 699.

Related concepts:

"The transactional model" on page 1285
The transactional model describes the way in which you can use transactions in message flows to accomplish certain tasks and results.

Related tasks:

"Configuring transactionality for message flows" on page 1290
A message flow runs in a single transaction, which is started when data is received by an input node, and can be committed or rolled back when all processing has completed.

"Configuring ODBC connections for globally coordinated transactions" on page 699
Configure the definition of your ODBC databases to the transaction manager (the queue manager).

"Enabling ODBC connections to the databases"
Set up the resources and environment that the broker requires for Open Database Connectivity (ODBC) connections to user databases on distributed systems.

"Enabling JDBC connections to the databases" on page 683
Configure connections to a user database through a JDBCProvider service.

Related reference:

"Supported databases" on page 3591
You can optionally configure databases to contain data that is accessed by your message flows. Databases from IBM and other suppliers are supported at specific versions on supported operating systems.

Related information:

WebSphere MQ Version 7 Information Center online

Enabling ODBC connections to the databases:

Set up the resources and environment that the broker requires for Open Database Connectivity (ODBC) connections to user databases on distributed systems.

About this task

You can configure both ODBC and Java Database Connectivity (JDBC) connections for access to user databases:

- To set up ODBC connections to user databases, follow the instructions in this section.

Optionally, after configuring the ODBC connection parameters, run the mqsicvp command to verify that the broker can connect to the data source, and to provide useful information about the data source and its interface.On Linux and UNIX systems, this command also checks that the ODBC environment is set up correctly.

- To set up JDBC connections to user databases, see "Enabling JDBC connections to the databases" on page 683.
- On z/OS systems, see "Data sources on z/OS" on page 4014 for information about enabling connections to databases. You do not have to follow the tasks described in this section.

Delete the ODBCINI64 environment variable if it exists; it is not required by WebSphere Message Broker Version 7.0.

For more information, see "User database connections" on page 2110.

The sample `odbc.ini` file that is supplied, and the information contained in these configuration topics, include all the connection parameters that are supported for connections to your databases. Any additional parameters that are provided by the DataDirect drivers are not tested or supported in a broker environment; consider your requirements carefully before specifying other parameters in your tailored ODBC `.ini` files.

To enable connections on distributed systems:

Procedure

Define the ODBC DSNs according to your platform:

Windows **On Windows:**
Follow the instructions in "Connecting to a database from Windows systems" on page 670.

Linux UNIX **On Linux and UNIX systems using the DataDirect Drivers:**
These drivers are used to access DB2, Informix, Oracle, Sybase, and SQLServer databases.

Follow the instructions in "Connecting to a database from Linux and UNIX systems using the DataDirect drivers" on page 674.

Linux UNIX **On Linux and UNIX systems using the WebSphere Message Broker Database Extender:**
Follow the instructions in "Connecting to a database from Linux and UNIX systems using the WebSphere Message Broker ODBC Database Extender (IE02)" on page 682.

You have now configured the ODBC DSNs for your user databases.

Results

You have now enabled the broker to make connections to your user databases.

What to do next

Next: If you have been using the instructions in "Working with databases" on page 2109, the next task is "Configuring ODBC connections for globally coordinated transactions" on page 699 (optional).

Related concepts:

"User database connections" on page 2110
User databases contain business data that is written and accessed by deployed message flows. You must create connections from the broker to the user database by using ODBC or JDBC.

Related tasks:

"Working with databases" on page 2109
Create and configure databases to use with your message flow applications.

"Enabling JDBC connections to the databases" on page 683
Configure connections to a user database through a JDBCProvider service.

"Connecting to a database from Linux and UNIX systems using the DataDirect drivers" on page 674
To enable a broker to connect to a database, define the ODBC data source name (DSN) for the database.

"Connecting to a database from Linux and UNIX systems using the WebSphere Message Broker ODBC Database Extender (IE02)" on page 682
WebSphere Message Broker ODBC Database Extender encapsulates the unixODBC driver manager and this topic describes how you set up and configure the broker to use it.

Related reference:

"Support for 32-bit and 64-bit platforms" on page 3589
WebSphere Message Broker operates in 32-bit mode or 64-bit mode, on supported operating systems.

"Supported databases" on page 3591
You can optionally configure databases to contain data that is accessed by your message flows. Databases from IBM and other suppliers are supported at specific versions on supported operating systems.

"Sample DataDirect odbc.ini file" on page 3660
A copy of the sample DataDirect ODBC definition file that is supplied with WebSphere Message Broker.

"Sample WebSphere Message Broker ODBC Database Extender (IE02) configuration files" on page 3596
How you use WebSphere Message Broker ODBC Database Extender (IE02) to load the correct data source driver.

"**mqsicvp** command" on page 3857
Use the **mqsicvp** command to perform verification tests on a broker, or to verify ODBC connections.

Connecting to a database from Windows systems:

To enable a broker to connect to a database, define the ODBC data source name (DSN) for the database.

Before you begin

Before you start: check that you have set up your environment so that the broker can connect to the database. Most database managers set up the required environment when you install, but others supply a database profile that you must run. For information about environments and running database profiles, see Setting up a command environment: Windows platforms.

For 32-bit Data Flow Engines (DFEs), you must use the 32-bit ODBC driver manager to make the 32-bit ODBC definitions. Similarly, for 64 bit DFEs, you must

use the 64-bit ODBC driver manager to make the 64-bit ODBC definitions.

About this task

Configure an ODBC data source by using the ODBC Data Source Administrator:

1. Click **Start** > **Control Panel** > **Administrative Tools** > **Data Sources (ODBC)**.

   If you are using 32-bit DFEs on Windows 64-bit systems, complete the following steps:

   a. Make a copy of **Data Sources (ODBC)**.
   b. Right-click the **Data Sources (ODBC)** shortcut and select properties.
   c. In the 64 bit ODBC dialogue, by default the target points to `%SystemRoot%\system32\odbcad32.exe`. Copy the shortcut, modify the target to point to `%SystemRoot%\SysWoW64\Odbcad32.exe`, and use the revised copy of the shortcut instead.

2. Click the **System DSN** tab and click **Add**.
3. Complete the steps in the following sections for the databases that you are working with.

   If you need more information about a particular database product, see the product-specific documentation.

DB2 UDB

Define a data source for DB2 UDB:

1. Select the driver `IBM DB2 ODBC DRIVER`.
2. Enter the data source name (DSN) and description.
3. Select the correct database alias from the list.
4. Click **Finish** to save your definition.
5. Click **OK** to close the ODBC Data Source Administrator.

You must register the data source as a system data source.

If you prefer, you can use the Configuration Assistant instead of the ODBC Data Source Administrator:

1. Open the DB2 Configuration Assistant.
2. Right-click the database and select **Change Database**.
3. Select **Data Source**.
4. Select **Register this database for ODBC**. Select the system data source option.
5. Click **Finish**.
6. The Test Connection dialog opens automatically and you can test the various connections.

Informix Dynamic Server

Define a data source for Informix Dynamic Server:

1. Select the driver `IBM INFORMIX ODBC DRIVER`.
2. On the **Connection** tab, specify:
   - The Informix server name.
   - The server host name.
   - The Informix network service name (as defined in the services file).
   - The network protocol (for example, `olsoctcp`).
   - The Informix data source name.
   - The user identifier to access the data source within.

- The password for that user identifier.

3. Click **Apply**.
4. Click **Test Connection** to check your supplied values.
5. Click **OK** to close the ODBC Data Source Administrator.

Microsoft SQL Server

Define a data source for Microsoft SQL Server:

1. Select the driver for the version of SQL Server that you are using:
   - `SQL Native Client` for SQL Server 2005.
   - `SQL Native Client` 10.0 for SQL Server 2008.
2. Specify a name and description.
3. Select the correct server from the list.
4. Click **Finish** to save your definition.
5. Click **OK** to close the ODBC Data Source Administrator.

Oracle

Define a data source for Oracle:

1.
   - If you are using WebSphere Message Broker for Windows 32-bit , select the driver `WebSphere Message Broker DataDirect Technologies 6.0 32-BIT Oracle Wire Protocol`.
   - If you are using WebSphere Message Broker for Windows 64-bit , select the driver `WebSphere Message Broker DataDirect Technologies 6.0 64-BIT Oracle Wire Protocol`.

   The ODBC Oracle Driver Setup dialog box opens.
2. On the **General** tab:
   a. Enter the DSN name, description, and host name of the machine where Oracle is running, the port number on which Oracle is listening, and the Oracle SID that you want to connect to.
3. On the **Advanced** tab:
   a. Select **Enable SQLDescribeParam**.
   b. Select **Procedure Returns Results**. The resultant ODBC definition in the Windows registry has a string value called **`ProcedureRetResults`** with the value `1`.
   c. Select **Login Timeout** and set the value to `0`.
4. Click **OK** to close the ODBC Data Source Administrator.
5. Click **Start** > **Run**.
6. Type `REGEDIT` in the **Open** field and click **OK**.
7. In the Registry Editor, navigate to the correct location.
   - If you are using WebSphere Message Broker for Windows 32-bit on Windows 32-bit editions: `HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBC.INI`
   - If you are using WebSphere Message Broker for Windows 32-bit on Windows 64-bit editions: `HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\ODBC\ODBC.INI`
   - If you are using WebSphere Message Broker for Windows 64-bit on Windows 64-bit editions: `HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBC.INI`
8. Expand that location, and right-click your DSN entry. Select **New** > **String Value**.

9. Specify **WorkArounds** for the string name.
   a. Right-click **WorkArounds**.
   b. Select **Modify**.
   c. Type the data value `536870912`.
10. Close the Registry Editor.

Sybase Adaptive Server Enterprise

Define a data source for Sybase Adaptive Server Enterprise:

1.
   - If you are using WebSphere Message Broker for Windows 32-bit , select the driver `WebSphere Message Broker DataDirect Technologies 6.0 32-BIT Sybase Wire Protocol`.
   - If you are using WebSphere Message Broker for Windows 64-bit , select the driver `WebSphere Message Broker DataDirect Technologies 6.0 64-BIT Sybase Wire Protocol`.
2. Enter the DSN name, description, and network address of the server, where the network address is made up of **`MyHostMachineName,MyHostMachinePortNumber`**.
3. On the **Advanced** tab:
   - Select **Enable Describe Parameter**.
   - Select **Login Timeout** and set the value to `0`.
4. On the **Performance** tab:
   - Ensure that the **Prepare Method** setting is `1 - Partial`.
5. Click **Start** > **Run**.
6. Type `REGEDIT` in the **Open** field and click **OK**.
7. In the Registry Editor, navigate to the correct location:
   - If you are using WebSphere Message Broker for Windows 32-bit on Windows 32-bit editions: `HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBC.INI`
   - If you are using WebSphere Message Broker for Windows 32-bit on Windows 64-bit editions: `HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\ODBC\ODBC.INI`
   - If you are using WebSphere Message Broker for Windows 64-bit on Windows 64-bit editions: `HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBC.INI`
8. Expand that location, and right-click your DSN entry. Select **New** > **String Value**. Specify **SelectUserName** for the string, and set the value to `1`.
9. Right-click your DSN again, and select **New** > **String Value**. Specify **EnableSPColumnTypes** for the string, and set the value to `2`.
10. Right-click your DSN again, and select **New** > **String Value**. Specify **TimestampTruncationBehavior** for the string, and set the value to `1`.
11. Right-click your DSN again, and select **New** > **String Value**. Specify **XAConnOptBehaviour** for the string, and set the value to `3`.
12. Close the Registry Editor.

solidDB

Define a data source for solidDB:

1. Select the driver IBM solidDB - (Unicode) DRIVER.
2. Enter the description.

3. Enter the communication port in the network location field, for example, `tcp 2315`.
4. Click **Finish** to save your definition.
5. Click **OK** to close the ODBC Data Source Administrator.

The solidDB is only supported on Windows 32-bit operating systems.

Results

You have now configured your ODBC data source names on Windows.

What to do next

Next: Configure the environment for issuing console commands, and for running the broker, so that the broker can access the required database libraries. For more information, see "Setting your environment to support 32-bit access to databases" on page 681.

Related tasks:

"Setting up a command environment" on page 213
After you have installed the product on one of the distributed systems, you must initialize the environment before you can use a runtime component or command.

Related reference:

"Supported databases" on page 3591
You can optionally configure databases to contain data that is accessed by your message flows. Databases from IBM and other suppliers are supported at specific versions on supported operating systems.

Connecting to a database from Linux and UNIX systems using the DataDirect drivers:

To enable a broker to connect to a database, define the ODBC data source name (DSN) for the database.

Before you begin

The following information is for DB2, Informix, Oracle, Sybase, and SQLServer databases only. For all other supported databases, see "Connecting to a database from Linux and UNIX systems using the WebSphere Message Broker ODBC Database Extender (IE02)" on page 682.

Before you start:
- Ensure that the database has been created, see "Creating the user databases" on page 661.
- Ensure that the broker is authorized to access the database, see "Authorizing access to user databases" on page 662.
- Check that you have set up your environment so that the broker can access the database; you might have to run a database profile supplied by the database vendor. For further information, see Setting up a command environment: Linux and UNIX systems.

Procedure

1. Copy the `odbc.ini` sample file that is supplied in the `install_dir/ODBC/V6.0` directory to a location of your choice. For example, copy the file to `/var/mqsi` which is the user directory for WebSphere Message Broker, and the preferred location for this copied file. Each broker service user ID on the system can therefore use its own DSN definitions.

See the sample file contents in "Sample DataDirect odbc.ini file" on page 3660.

2. Ensure that the `odbc.ini` file has file ownership of `mqm:mqbrkrs` and has the same permissions as the supplied sample file.
3. Set the ODBCINI environment variable to point to your `odbc.ini` file, specifying a full path and file name. Make sure that you point to the copy, do not point to the `odbc.ini` file in the installation directories.
4. If you are connecting to DB2 or Informix databases, set the library search path environment variable to show the location of the libraries for the database manager that you are using.

   For more information about the library search path, ask your database administrator (DBA), or see the documentation for your database manager.

   The library search path environment variable depends on your platform:

   - Linux Solaris On Linux and Solaris, set LD_LIBRARY_PATH.
   - HP-UX On HP-UX, set SHLIB_PATH.
   - AIX On AIX, set LIBPATH.

   Updates to the library search path are not required for other supported databases.
5. If you are using a DB2 database instance that is installed on AIX, a single process can make a maximum of 10 connections that use shared memory to a DB2 database. Use TCP/IP mode to connect to the database instance; see "DB2 error message SQL1224N is issued when you connect to DB2" on page 3494.
6. Edit the final stanza in the `odbc.ini` file, the [ODBC] stanza, to specify the location of the ODBC Driver Manager and to control tracing. The exact details in the stanza depend on the platform.

   To ensure that you edit the correct `odbc.ini` file, you can open the file in the vi text editor by using the following command:

   ```
   vi $ODBCINI
   ```

   a. In **InstallDir**, add the WebSphere Message Broker installation location to complete the fully qualified path to the ODBC directory. If you do not specify this value correctly, the ODBC definition does not work.

   b. In **TraceFile**, type the fully qualified path and file name to which the ODBC trace is written. Trace files can become large; specify a directory with plenty of free disk space.

   c. In **TraceDll**, add the WebSphere Message Broker installation location to complete the fully qualified path to the ODBC trace DLL.

   d. Accept the default values shown in the sample `odbc.ini` file for all the other entries in the stanza.

      For example on AIX:

      ```
      ;########################################
      ;###### Mandatory information stanza ######
      ;########################################

      [ODBC]
      ;# To turn on ODBC trace set Trace=1
      Trace=0
      TraceOptions=3
      TraceFile=<A Directory with plenty of free space to hold trace
      output>/odbctrace.out
      TraceDll=<Your Broker install directory>/ODBC/V6.0/lib/odbctrac.so
      InstallDir=<Your Broker install directory>/ODBC/V6.0
      UseCursorLib=0
      IANAAppCodePage=4
      UNICODE=UTF-8
      ```

7. Edit the first stanza in the odbc.ini file, the [ODBC Data Sources] stanza, to list the DSN of each database.

   For example on AIX:

   ```
   ;########################################
   ;###### List of data sources stanza #######
   ;########################################
   [ODBC Data Sources]
   DB2DB=IBM DB2 ODBC Driver
   ORACLEDB=DataDirect 6.0 Oracle Wire Protocol
   ORACLERACDB=DataDirect 6.0 Oracle Wire Protocol (Real Application Clusters)
   SYBASEDB=DataDirect 6.0 Sybase Wire Protocol
   SYBASEDBUTF8=DataDirect 6.0 Sybase UTF8 Wire Protocol
   SQLSERVERDB=DataDirect 6.0 SQL Server Wire Protocol
   INFORMIXDB=IBM Informix ODBC Driver
   ```

   List all your DSNs in your odbc.ini file, regardless of the database manager. On Oracle and Sybase, you can define multiple DSNs to resolve to the same database; however, if you are using global coordination of transactions, do not use this option because it might cause data integrity problems.

8. For each database that you listed in the [ODBC Data Sources] stanza, create a stanza in the odbc.ini file after the [ODBC Data Sources] stanza. The entries in the stanza depend on the database manager. Slight differences also occur between operating systems, for example the file paths to the drivers.

   For a DB2 database instance:

   For Linux on x86:

   a. In **Driver**, add the full path of your DB2 installation.
   b. In **Description**, type a meaningful description of the database. This field is for information only and does not affect the connection.
   c. In **Database**, type the DB2 alias. The data source name must be the same as the database alias name. If you are using a remote DB2 database, you must set up your client-server connection to resolve this alias to the correct database.

      If the requirement is to have multiple stanzas that refer to the same DB2 database, aliases must be created in DB2 by using the DB2 CATALOG command. These aliases can then have their own stanza in the ODBCINI file.

      The ODBCINI file cannot be used to set up aliases for DB2.

      For more information, see the DB2 documentation.

      For example:

      ```
      ;# DB2 stanza
      [DB2DB]
      Driver=<Your DB2 install directory>/lib32/libdb2.so
      Description=DB2 ODBC Database
      Database=DB2DB
      ```

   For all other platforms:

   a. In **Driver**, accept the value as shown in the sample odbc.ini file.
   b. In **Description**, type a meaningful description of the database. This field is for information only and does not affect the connection.
   c. In **Database**, type the DB2 alias. The data source name must be the same as the database alias name. If you are using a remote DB2 database, you must set up your client-server connection to resolve this alias to the correct database. For more information, see the DB2 documentation.

If the requirement is to have multiple stanzas that refer to the same DB2 database, aliases must be created in DB2 by using the DB2 CATALOG command. These aliases can then have their own stanza in the ODBCINI file.

The ODBCINI file cannot be used to set up aliases for DB2.

For example, on AIX:

```
;# DB2 stanza
[DB2DB]
DRIVER=libdb2Wrapper.so
Description=DB2DB DB2 ODBC Database
Database=DB2DB
```

For an Oracle database:

For all platforms:

a. In **Driver**, add the WebSphere Message Broker installation location to complete the fully qualified path to the driver shown in the sample `odbc.ini` file.

b. In **Description**, type a meaningful description of the database. This field is for information only and does not affect the connection.

c. In **HostName**, type the name or IP address of the machine that is hosting your Oracle system.

d. In **PortNumber**, type the number of the port on which your Oracle server is listening on the machine you specified in **HostName**.

e. In **SID**, type the Oracle service name that you want to connect to on the system you specified in **HostName**.

f. Accept the default values shown in the sample `odbc.ini` file for all the other entries in the stanza.

For example on AIX:

```
;# Oracle stanza
[ORACLEDB]
Driver=<Your Broker install directory>/ODBC/V6.0/lib/UKora24.so
Description=DataDirect 6.0 Oracle Wire Protocol
HostName=<Your Oracle Server Machine Name>
PortNumber=<Port on which Oracle is listening on HostName>
SID=<Your Oracle SID>
CatalogOptions=0
EnableStaticCursorsForLongData=0
ApplicationUsingThreads=1
EnableDescribeParam=1
OptimizePrepare=1
WorkArounds=536870912
ProcedureRetResults=1
ColumnSizeAsCharacter=1
LoginTimeout=0
```

For an Oracle database that uses Real Application Clusters:

For all platforms:

a. In **Driver**, add the WebSphere Message Broker installation location to complete the fully qualified path to the driver shown in the sample `odbc.ini` file.

b. In **Description**, type a meaningful description of the database. This field is for information only and does not affect the connection.

c. In **HostName**, type the name or IP address of the machine that is hosting your primary (preferred) Oracle instance.

d. In **PortNumber**, type the number of the port on which your Oracle server is listening on the machine you specified in **HostName**.

e. In **ServiceName**, type the Oracle Real Application Cluster service name that you want to connect to on the instance you specified in **HostName**.

f. In **AlternateServers**, provide a list of alternative locations for this service for situations when the primary location, defined in **HostName**, is unavailable. Each location specification consists of three parts, separated by colons. Enter these values as one continuous string; the text in this example has been split to improve readability.

```
HostName=<Alternative host name>
:PortNumber=<Oracle listner port on alternative server>
:ServiceName=<Service name on the alternative server>
```

If you want to specify more than one AlternateServer, separate each additional location specification with a comma. Whenever a new database connection is required, for example after an Oracle instance failover, the primary location will be tried first. However, if the primary location is unavailable, the driver will try the list of alternative locations in turn.

g. Accept the default values shown in the sample `odbc.ini` file for all the other entries in the stanza.

For example on AIX:

```
;# Oracle Real Application Clusters stanza
[ORACLERACDB]
Driver=<Your Broker install directory>/ODBC/V6.0/lib/UKora24.so
Description=DataDirect 6.0 Oracle Wire Protocol
HostName=<Your Oracle Server Machine Name>
PortNumber=<Port on which Oracle is listening on HostName>
ServiceName=<Your Oracle Real Application Cluster Service Name>
AlternateServers=(HostName=<Your alternative host name>:PortNumber=<Port on
which Oracle is listening on the alternative host>:ServiceName=<Your
Oracle Real Application Cluster Service Name>)
CatalogOptions=0
EnableStaticCursorsForLongData=0
ApplicationUsingThreads=1
EnableDescribeParam=1
OptimizePrepare=1
WorkArounds=536870912
ProcedureRetResults=1
ColumnSizeAsCharacter=1
LoginTimeout=0
```

For a Sybase database:

For all platforms except Linux on IBM z Systems:

a. In **Driver**, add the WebSphere Message Broker installation location to complete the fully qualified path to the driver shown in the sample `odbc.ini` file.

b. In **Description**, type a meaningful description of the database. This field is for information only and does not affect the connection.

c. In **Database**, type the name of the database to which you want to connect by default. If you do not specify a value, the default value is the database defined by your system administrator for each user.

d. In **NetworkAddress**, type the network address of your Sybase ASE server (this address is required for local and remote databases). Specify an IP address or server name as follows:

```
<Your Sybase server name or IP address>,<Your Sybase port number>
```

For example: `Sybaseserver,5000`. You can also specify the IP address directly, for example `199.226.224.34,5000`. You can find the port number in the Sybase interfaces file that is named `interfaces`.

e. Accept the default values shown in the sample `odbc.ini` file for all the other entries in the stanza.

For example on AIX:

```
;# Sybase Stanza
[SYBASEDB]
Driver=<Your Broker install directory>/ODBC/V6.0/lib/UKase24.so
Description=DataDirect 6.0 Sybase Wire Protocol
Database=<Your Database Name>
ApplicationUsingThreads=1
EnableDescribeParam=1
OptimizePrepare=1
SelectMethod=0
NetworkAddress=<Your Sybase Server Name>,<Your Sybase Port Number>
SelectUserName=1
ColumnSizeAsCharacter=1
EnableSPColumnTypes=2
LoginTimeout=0
TimestampTruncationBehavior=1
XAConnOptBehaviour=3
```

If you want to use a UNICODE UTF8 Sybase data source, add the following line to the end of your Sybase stanza:

```
Charset=UTF8
```

For remote access to an SQL Server database

For all platforms except Linux on POWER, Linux on IBM z Systems, and Solaris on x86-64:

a. In **Driver**, add the WebSphere Message Broker installation location to complete the fully qualified path to the driver shown in the sample `odbc.ini`.

b. In **Description**, type a meaningful description of the database. This field is for information only and does not affect the connection.

c. In **Address**, either:

1) Specify an IP address or server name and a port number to locate the SQLServer database you want to connect to, as follows:

```
<Your SQLServer machine name or IP address>,<Your SQLServer port number>
```

or

2) Specify an IP address or server name and an instance name to locate the SQLServer database you want to connect to, as follows:

```
<Your SQLServer machine name or IP address>\<Your SQLServer instance name>
```

If your instance name is blank, specify `<Your SQLServer machine name or IP address>`

d. In **Database**, type the name of the database to which you want to connect by default. If you do not specify a value, the default value is the database defined by your system administrator for each user.

e. Accept the default values shown in the sample `odbc.ini` file for all the other entries in the stanza.

For example, on AIX:

```
;# UNIX to SQLServer stanza
[SQLSERVERDB]
Driver=<Your Broker install directory>/ODBC/V6.0/lib/UKmsss24.so
Description=DataDirect 6.0 SQL Server Wire Protocol
Address=<Your SQLServer Machine Name>,<Your SQLServer Port Number>
;# Alternative way to locate server using a named instance
;# Address=<Your SQLServer Machine Name>\<Your SQLServer Instance Name>
AnsiNPW=Yes
Database=db
QuotedId=No
ColumnSizeAsCharacter=1
```

For an Informix database

For Linux on x86

a. In **Driver**, add the full path of your Informix Client library.
b. In **Description**, type a meaningful description of the database. This field is for information only and does not affect the connection.
c. In **ServerName**, type the name of the Informix IDS server.
d. In **Database**, type the name of the database to which you want to connect by default. If you do not specify a value, the default value is the database that is defined by your system administrator for each user.

   For example

```
;# Informix Stanza
[INFORMIXDB]
Driver=<Your Informix Client Directory>/lib/cli/iclit09b.so
Description=IBM Informix ODBC Driver
ServerName=<Your Informix Server Name>
Database=<Your Database Name>
```

For all other platforms:

a. In **Driver**, accept the value as shown in the sample `odbc.ini` file.
b. In **Description**, type a meaningful description of the database. This field is for information only and does not affect the connection.
c. In **ServerName**, type the name of the Informix IDS server.
d. In **Database**, type the name of the database to which you want to connect by default. If you do not specify a value, the default value is the database that is defined by your system administrator for each user.

   For example, on AIX:

```
;# Informix Stanza
[INFORMIXDB]
Driver=libinfWrapper.so
Description=IBM Informix ODBC Driver
ServerName=<Your Informix Server Name>
Database=<Your Database Name>
```

9. Ensure that you have edited all three parts of the `odbc.ini` file:
   - The [ODBC Data Source] stanza at the top of the `odbc.ini` file.
   - A stanza for each data source.
   - The [ODBC] stanza at the end of the `odbc.ini` file.

   If you do not configure all three parts correctly, the ODBC DSNs do not work and the broker is unable to connect to the database.

Results

You have now configured database connections on Linux and UNIX. You can check that the ODBC environment is configured correctly by running the **mqsicvp** command. For more information, see "**mqsicvp** command" on page 3857.

Related tasks:

"Creating the user databases" on page 661
If your message flows create, update, retrieve, or delete application and business data in one or more user databases, create the databases before you deploy the message flows to a broker.

Related reference:

"Supported databases" on page 3591
You can optionally configure databases to contain data that is accessed by your message flows. Databases from IBM and other suppliers are supported at specific versions on supported operating systems.

"Sample DataDirect odbc.ini file" on page 3660
A copy of the sample DataDirect ODBC definition file that is supplied with WebSphere Message Broker.

Related information:

WebSphere MQ Version 7 Information Center online

Setting your environment to support 32-bit access to databases:

When you have configured your ODBC data source names (DSNs), you must also configure the environment so that you can issue console commands, and the brokers that you start can access the required database libraries. For example, if you have a DB2 database, you must add the DB2 client libraries to your library search path.

About this task

Windows On Windows platforms, the environment is typically set up for you when you install the database product, and no further action is required. However, some database managers provide a database profile that you must run to enable the connection from the broker; for further information, see Setting up a command environment: Windows platforms.

Linux UNIX On Linux and UNIX systems, run a profile for each database you want to access. For example, on DB2 you must run `db2profile`; other database vendors have similar profiles. For further information, see Setting up a command environment: Linux and UNIX systems.

What to do next

Next: If you have been following the instructions in "Working with databases" on page 2109, the next task is "Configuring ODBC connections for globally coordinated transactions" on page 699 (optional).

Related tasks:

"Connecting to a database from Linux and UNIX systems using the DataDirect drivers" on page 674
To enable a broker to connect to a database, define the ODBC data source name (DSN) for the database.

Related reference:

"Support for 32-bit and 64-bit platforms" on page 3589
WebSphere Message Broker operates in 32-bit mode or 64-bit mode, on supported operating systems.

"Supported databases" on page 3591
You can optionally configure databases to contain data that is accessed by your message flows. Databases from IBM and other suppliers are supported at specific versions on supported operating systems.

"Sample DataDirect odbc.ini file" on page 3660
A copy of the sample DataDirect ODBC definition file that is supplied with WebSphere Message Broker.

Connecting to a database from Linux and UNIX systems using the WebSphere Message Broker ODBC Database Extender (IE02):

WebSphere Message Broker ODBC Database Extender encapsulates the unixODBC driver manager and this topic describes how you set up and configure the broker to use it.

Before you begin

Before you start:

Read the information about the unixODBC Project and the IBM solidDB product family.

About this task

WebSphere Message Broker ODBC Database Extender is required when using WebSphere Message Broker to interface with an ODBC data source that is not supported through the DataDirect ODBC drivers.

Procedure

To configure the broker:

1. Make a copy of the sample `odbc.ini` file supplied in `install_dir/ODBC/unixodbc/odbc.ini`, where `install.dir` is the directory in which you installed WebSphere Message Broker. For example, copy the file to `/var/mqsi` which is the user directory for WebSphere Message Broker, and the preferred location for this copied file.
2. Set up the environment variable `ODBCUOINI` to point to the full path and name of this file. Make sure that you point to the copy, do not point to the `odbc.ini` file in the installation directories.
3. Make a copy of the sample `odbcinst.ini` file supplied in `install_dir/ODBC/unixodbc/odbcinst.ini`, where `install.dir` is the directory in which you installed WebSphere Message Broker. For example, copy the file to `/var/mqsi` which is the user directory for WebSphere Message Broker, and the preferred location for this copied file.
4. Set up the environment variable `ODBCSYSINI` to point to the directory containing your `odbcinst.ini` file. Make sure that you point to the copy, do not point to the `odbcinst.ini` file in the installation directories.

For information about how to populate the configuration files see "Sample WebSphere Message Broker ODBC Database Extender (IE02) configuration files" on page 3596

Related tasks:

"Connecting to a database from Windows systems" on page 670
To enable a broker to connect to a database, define the ODBC data source name (DSN) for the database.

Related reference:

"Sample WebSphere Message Broker ODBC Database Extender (IE02) configuration files" on page 3596
How you use WebSphere Message Broker ODBC Database Extender (IE02) to load the correct data source driver.

Enabling JDBC connections to the databases:

Configure connections to a user database through a JDBCProvider service.

About this task

Use a JDBC connection from Java programs that are associated with a JavaCompute node or a user-defined node that is written in Java.

You must also set up JDBC connections if your message flows include DatabaseRetrieve or DatabaseRoute nodes.

If you configure a JDBC type 4 connection from an application running on a Linux, UNIX, or Windows system, you can configure your broker and queue manager to include interactions with the databases in globally-coordinated transactions. On z/OS, JDBC connections can be broker-coordinated only.

The information provided in this section is independent of whether your operating systems, brokers, execution groups, queue managers, and databases operate in 32-bit or 64-bit mode, except where stated.

When you write Java classes for a JavaCompute node or a user-defined node, your code must comply with the following restrictions:

- Do not include any code that performs a COMMIT or a ROLLBACK function.
- Do not close the connection to the database. The broker manages all connections, and closes a connection if it is idle for approximately one minute, or if the message flow completes.

To configure JDBC type 4 connections:

Procedure

1. Set up your JDBC provider definition.
2. Optional: Set up security.
3. Optional: Configure for global-coordination of transactions.
4. Optional: If your broker is running on a Windows system, authorize access to JDBCProvider resources.

What to do next

Next: If you have been following the instructions in "Working with databases" on page 2109, the next task is "Setting up a JDBC provider for type 4 connections" on page 684.

When you have completed configuration of the databases, add or modify Java code in your JavaCompute or user-defined nodes to access the database that is identified in the JDBCProvider service.

Related concepts:

"Databases overview" on page 2109
Brokers can access *user databases* to manipulate your business data. If you have user databases, you must configure them before you can access them from your message flow.

Related tasks:

"Interacting with databases by using the JavaCompute node" on page 2661
Access databases from Java code included in the JavaCompute node.

"Extending the capability of a Java message processing or output node" on page 3069
Within a message processing or output node, you can add extended functions to your Java node.

Related reference:

"Supported databases" on page 3591
You can optionally configure databases to contain data that is accessed by your message flows. Databases from IBM and other suppliers are supported at specific versions on supported operating systems.

Setting up a JDBC provider for type 4 connections:

Use the **mqsicreateconfigurableservice** or the **mqsichangeproperties** command to configure a JDBC provider service.

Before you begin

Before you start:

- Create a broker
- Create your database following the database documentation.

About this task

When you include a DatabaseRetrieve, DatabaseRoute, JavaCompute, Mapping, or Java user-defined node in a message flow, and interact with a database in that node, the broker must establish a connection with the database to fulfill the operations that are performed by the node. You must define a JDBCProvider configurable service to provide the broker with the information that it needs to complete the connection.

A JDBCProvider configurable service supports connections to one database only; you must create a service for each database that your nodes or Java applications connect to.

To set up a JDBC provider for type 4 connections by using the WebSphere Message Broker Explorer, see "Creating a new configurable service" on page 645.

To set up a JDBC provider for type 4 connections by using the **mqsicreateconfigurableservice** or **mqsichangeproperties** commands, complete the following steps:

Procedure

1. Identify the type of database for which you require a JDBCProvider service.

Supported JDBC drivers and databases are shown in "Supported databases" on page 3591; support for globally coordinated (XA) transactions is restricted on some platforms and for some databases.

2. Run the **mqsireportproperties** command to view the list of available JDBCProvider services. Substitute the name of your broker in place of *broker_name*.

```
mqsireportproperties broker_name -c JDBCProviders -a -o AllReportableEntityNames
```

The command response lists all the JDBCProvider configurable services that are defined. If you have not created your own definitions, the following list of default supplied services is shown:

- DB2
- Informix
- Informix_With_Date_Format
- Microsoft_SQL_Server
- Oracle
- Sybase_JConnect6_05

If you are connecting to an Informix database:

- Use Informix_With_Date_Format for compatibility with client applications that are dependent on the date format connection attribute that was used by earlier versions of Informix servers.
- Use Informix for client applications that are not dependent on the date format attribute.

3. View the contents of the relevant JDBCProvider service definition. For example, run the following command to display the supplied Oracle definition:

```
mqsireportproperties broker_name -c JDBCProviders -o Oracle -r
```

The command response lists all the properties for the Oracle definition. If you have not changed this definition, the properties are set to initial values, some of which you must change to create a viable definition. For example, the property **databaseName** is set to `default_Database_Name`, and you must change it to identify the specific database that you want to connect to.

A JDBCProvider service has the following properties:

- **connectionUrlFormat**. A pattern that represents the connection URL definition, which is specific to a particular database type. For example, the pattern for DB2 is defined with the following content:

  ```
  jdbc:db2://[serverName]:[portNumber]/[databaseName]:user=[user];password=[password];
  ```

  The pattern is used and completed by the broker at run time when it connects to the database. The values in brackets, for example `[serverName]`, are substituted by the broker into the pattern by using the values that you have specified on the **mqsicreateconfigurableservice**, **mqsichangeproperties**, or **mqsisetdbparms** commands.

  The following values and order of preference are used by the broker to substitute the user ID and password in the pattern:

  a. First, on all platforms: The user ID and password that you have set for the specific database, by using the **mqsisetdbparms** and specifying the database in the **-n** parameter.

  b. Second, on all platforms: The user ID and password that you have set for all other databases, by using the **mqsisetdbparms** and specifying `jdbc::JDBC` in the **-n** parameter.

  c. Third, the values are platform-specific:

     1) Windows On Windows: The broker service ID and password that you specified on the **mqsicreatebroker** command.

2) Linux UNIX On Linux and UNIX: The user ID `mqsiUser` and password `********` (these values are fixed).

3) z/OS On z/OS: The user ID `""` and password `""`.

If you are using one of the supplied JDBCProvider services, do not use the **mqsichangeproperties** command to change the pattern itself; changes made to the pattern might cause unpredictable results.

If you use the **mqsicreateconfigurableservice** command to define your own JDBCProvider service, use the **mqsireportproperties** command to check that the content of the **connectionUrlFormat** string exactly matches the default supplied provider for the database type that you are using.

In addition, if you are working on z/OS, and are using the JCL files BIPCRCS and BIPCHPR to define your JDBCProvider service, ensure that your 3270 emulator is configured to use the same code page that the broker is running in. If the code pages do not match, the **connectionUrlFormat** string pattern that you define might not be recognized correctly by the broker.

- **connectionUrlFormat Attr1-5**. If the defined URL pattern for a database contains non-standard JDBC data source properties, such as a server identifier, specify these properties in addition to the standard attributes by using one of five general-purpose connection URL attributes. For example:
  - If **connectionURLFormat** = `jdbc:oracle:thin:[user]/[password]@[serverName]:[portNumber]:[connectionUrlFormatAttr1]`, **connectionUrlFormatAttr1** must contain an Oracle server identifier, which you must supply by defining the value for the property **connectionUrlFormatAttr1** on the **mqsicreateconfigurableservice** or **mqsichangeproperties** command. The broker can then substitute all the required values into the required pattern.
  - If **connectionURLFormat** = `jdbc:informix-sqli://[serverName]:[portNumber]/[databaseName]:informixserver=[connectionUrlFormatAttr1];user=[user];password=[password]`, **connectionUrlFormatAttr1** must contain the name of the Informix instance on the server (typically specified by the INFORMIXSERVER environment variable). This value is case-sensitive.
- **databaseName**. The name of the database to which the data source entry enables connections; for example, `employees`.
- **databaseType**. The database type; for example, `DB2`.
- **databaseVersion**. The database version; for example, `9.1`.
- **description**. An optional property to describe the data source definition.
- **environmentParms**. ForDB2 only. An optional property specifying a list of data source properties of the form `name=value` each separated by a semicolon.
- **jarsURL**. The local directory path, on the system on which the broker is running, where the JAR file that contains the type 4 driver class is located.

  In addition, a storage area network disk can be used for the directory path, but a mapped network drive to a remote computer cannot be used.
- **maxConnectionPoolSize**. Optionally set this property to create a JDBC connection pool. For more information, see “Using a JDBC connection pool to manage database resources used by an execution group” on page 1003.
- **portNumber**. The port number on which the database server is listening; for example, `50000`.

- **securityIdentity**. A unique security key to perform a second broker registry lookup to find an entry under the broker security identities, which store the encrypted password for the user on the associated host system; for example, `mysecurityIdentity`.

  Create a security identity by using the **mqsisetdbparms** command, as described in "Securing a JDBC type 4 connection" on page 689. The value of **securityIdentity** (for example, `mysecurityIdentity`) must match the value that you specify following the prefix `jdbc::` for the parameter **-n** on that command.

  The security identity provides a user ID and password value pair, which are used to access the specified data source defined for a particular JDBCProvider service entry. This property is ignored if the connection URL does not contain both a user ID and password pair, which require property values to be substituted for such inserts.

  The default values, which you can set by specifying a **ResourceName** of `jdbc::JDBC` on the **mqsisetdbparms** command, are used under the following conditions:

  – If the **securityIdentity** is blank, or if you have not changed it from the default value `default_User@default_Server`, but the identity is required for the connection URL pattern.

  – If you have entered a valid unique security identity key, but it cannot be found under the DSN key.

- **serverName**. The name of the server; for example, `host1`.
- **type4DatasourceClassName**. The name of the JDBC data source class name that is used to establish a type 4 connection to a remote database, and to coordinate transaction support. For example, specify `com.ibm.db2.jcc.DB2XADataSource` for DB2, or specify `oracle.jdbc.xa.client.OracleXADataSource` for Oracle. You must specify the XA class name when using the **getJDBCType4Connection()** API call for coordinated transactions. If the database server does not support XA transactions, or you do not want to use the XA protocol, this property is optional and you must set the **jdbcProviderXASupport** property to false.
- **type4DriverClassName**. The name of the JDBC type 4 driver class name that is used to establish a connection. For example, specify `com.ibm.db2.jcc.DB2Driver` for DB2, or specify `oracle.jdbc.OracleDriver` for Oracle.
- **jdbcProviderXASupport**. An optional property that controls whether the broker connects to a database server using the XA Protocol. By default this property is set to true. If the database server is not enabled for XA Support, or coordinated transactions are not required, set the value to false. In which case the type 4 driver specified using the **type4DriverClassName** property is used, instead of the type 4 datasource specified in the **type4DatasourceClassName** property.

4. If you want to use the provided definition, run the **mqsichangeproperties** command to replace default values with the values specific to your database and environment. If you are in any doubt about the required values, consult your database administrator, or check the documentation that is provided with your chosen database. Some values depend on how and where you have installed the database product; for example, the property **jarsURL** identifies the location of the JAR files supplied and installed by the database provider.
5. If you want to create a new configurable service, perhaps because you want to retain the supplied service as a template for future definitions, run the **mqsicreateconfigurableservice** command to create the definition.

```
mqsicreateconfigurableservice broker_name -c JDBCProviders -o provider_name
-n list of properties -v list of values
```

Enter the command on a single line; the example is split to enhance readability.

Specify all the properties that are required by the database provider that you have chosen. To specify a list of properties and values, separate the items after each flag with a comma. For example, `-n databaseName,databaseType -v EmployeeDB,DB2`. If you do not specify all the properties on the **mqsicreateconfigurableservice** command, you can update them later with the **mqsichangeproperties** command.

6. When you have set up or modified your JDBCProvider service, you must reload any execution groups which currently use, or intend to use, the JDBCProvider service.

What to do next

Next: If required, set up security for the JDBC connection, set up the environment to include the JDBCProvider service in globally coordinated transactions, or both.

Related concepts:

"Databases overview" on page 2109
Brokers can access *user databases* to manipulate your business data. If you have user databases, you must configure them before you can access them from your message flow.

Related tasks:

"Securing a JDBC type 4 connection" on page 689
Set up security for the JDBC connection if required by the database provider.

"Configuring a JDBC type 4 connection for globally coordinated transactions" on page 691
If you want the database that you access through a JDBC type 4 connection to participate in globally coordinated transactions, set up the appropriate environment.

"Authorizing access to JDBC type 4 JDBCProvider resources on Windows" on page 694
Authorize the broker and queue manager to access shared resources that are associated with the JDBCProvider. This task is required only if you want the database updates to be included in globally coordinated transactions on Windows systems.

"Interacting with databases by using the JavaCompute node" on page 2661
Access databases from Java code included in the JavaCompute node.

"Extending the capability of a Java message processing or output node" on page 3069
Within a message processing or output node, you can add extended functions to your Java node.

Related reference:

"Supported databases" on page 3591
You can optionally configure databases to contain data that is accessed by your message flows. Databases from IBM and other suppliers are supported at specific versions on supported operating systems.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a

broker external resource.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

Securing a JDBC type 4 connection:

Set up security for the JDBC connection if required by the database provider.

Before you begin

Before you start: Set up your JDBC provider definition.

About this task

Some databases require all access to be associated with a known user ID; for others this association is optional. For example, DB2 requires a data source login name and password on all connections. If the database requires secure access to be defined, or if you choose to implement security in an optional situation, complete the task described here.

Procedure

1. Identify the user ID that you want to associate with the JDBC connection, or create a user ID with a password, following the appropriate instructions for your operating system and database.
2. Run the **mqsisetdbparms** command to associate the user ID and password with the security identity that is associated with the database that you will access using the JDBCProvider configurable service.

   The following values and order of preference are used by the broker for the JDBC connection:

   a. First, on all platforms: The user ID and password that you have set for the specific database, by using the **mqsisetdbparms** and specifying the database in the **-n** parameter.
   b. Second, on all platforms: The user ID and password that you have set for all other databases, by using the **mqsisetdbparms** and specifying `jdbc:JDBC` in the **-n** parameter.
   c. Third, the values are platform-specific:
      1) Windows On Windows: The broker service ID and password that you specified on the **mqsicreatebroker** command.
      2) Linux UNIX On Linux and UNIX: The user ID `mqsiUser` and password `********` (these values are fixed).
      3) z/OS On z/OS: The user ID `""` and password `""`.

   Use the following command format:

```
mqsisetdbparms broker_name -n security_identity -u userID -p password
```

   For example, if you want user ID `myuserid` with a password of `secretpw` to access a database on broker BROKER1, run the following command:

```
mqsisetdbparms BROKER1 -n jdbc::mySecurityIdentity -u myuserid -p secretpw
```

In the example, the *mySecurityIdentity* is prefixed with `jdbc::` to indicate the type of the connection for which the user ID and password are defined.

If you want to use the same user ID and password for more than one database, you can specify a resource name of `jdbc::JDBC` on this command, as shown in the following example:

```
mqsisetdbparms BROKER1 -n jdbc::JDBC -u defaultuid -p defaultpw
```

3. Update the corresponding **securityIdentity** property for the JDBCProvider configurable service to associate the connection with the security identity that you have defined. Use the following command format:

```
mqsichangeproperties broker_name -c JDBCProviders -o service_name -n securityIdentity -v security_identity
```

For example, if you are using the supplied JDBCProvider definition for Oracle:

```
mqsichangeproperties BROKER1 -c JDBCProviders -o Oracle -n securityIdentity -v mySecurityIdentity
```

You can use the same user ID and password definition for more than one JDBCProvider service if appropriate; specify the same security identity that you specified on the **mqsisetdbparms** command as the value for the **securityIdentity** property in each JDBCProvider service definition that uses the same access security. If you are using a default user ID and password that you have defined by specifying a **ResourceName** of `jdbc::JDBC` on the **mqsisetdbparms** command, do not change this property; retain the default value `default_User@default_Server` to cause the broker to use the default values that you have set.

What to do next

Next: if you are setting up a connection on Windows, see "Authorizing access to JDBC type 4 JDBCProvider resources on Windows" on page 694. Otherwise, see "Configuring a JDBC type 4 connection for globally coordinated transactions" on page 691.

Related concepts:

"Databases overview" on page 2109
Brokers can access *user databases* to manipulate your business data. If you have user databases, you must configure them before you can access them from your message flow.

Related tasks:

"Setting up a JDBC provider for type 4 connections" on page 684
Use the **mqsicreateconfigurableservice** or the **mqsichangeproperties** command to configure a JDBC provider service.

"Configuring a JDBC type 4 connection for globally coordinated transactions" on page 691
If you want the database that you access through a JDBC type 4 connection to participate in globally coordinated transactions, set up the appropriate environment.

"Authorizing access to JDBC type 4 JDBCProvider resources on Windows" on page 694
Authorize the broker and queue manager to access shared resources that are associated with the JDBCProvider. This task is required only if you want the database updates to be included in globally coordinated transactions on Windows systems.

"Interacting with databases by using the JavaCompute node" on page 2661
Access databases from Java code included in the JavaCompute node.

"Extending the capability of a Java message processing or output node" on page 3069
Within a message processing or output node, you can add extended functions to your Java node.

Related reference:

"Supported databases" on page 3591
You can optionally configure databases to contain data that is accessed by your message flows. Databases from IBM and other suppliers are supported at specific versions on supported operating systems.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

Configuring a JDBC type 4 connection for globally coordinated transactions:

If you want the database that you access through a JDBC type 4 connection to participate in globally coordinated transactions, set up the appropriate environment.

Before you begin

Before you start: Set up your JDBC provider definition.

About this task

Updates that you make to a database across a JDBC type 4 connection can be coordinated with other actions taken within the message flow, if you set up the resources to support coordination.

Complete the following steps:

Procedure

1. Check that the definition of your JDBCProvider service is appropriate for coordinated transactions.

   For example, to set up the required JDBC classes:

   - For DB2, set **type4DatasourceClassName** to `com.ibm.db2.jcc.DB2XADataSource` and **type4DriverClassName** to `com.ibm.db2.jcc.DB2Driver`
   - For Oracle, set **type4DatasourceClassName** to `oracle.jdbc.xa.client.OracleXADataSource` and **type4DriverClassName** to `oracle.jdbc.OracleDriver`

   Consult your database administrator or the documentation provided by your database supplier, to confirm that all the JDBCProvider service properties are set appropriately. For example, a database supplier might require secure access if it is participating in coordinated transactions.

2. Define the switch file and the database properties:

   a. Linux UNIX On Linux and UNIX systems, open the `qm.ini` file for the broker queue manager with a text editor. Add the following stanza for each database:

   ```
   XAResourceManager:
          Name=Database_Name
          SwitchFile=JDBCSwitch
          XAOpenString=JDBC_DataSource
          ThreadOfControl=THREAD
   ```

   Database_Name is the database name (DSN) of the database defined to the JDBCProvider configurable service (for example, specified by `-n databaseName -v` *Database_Name* on the **mqsichangeproperties** command).

   `JDBCSwitch` is a fixed generic name that represents the switch file for XA coordination. Use this value, or another single fixed value, in each stanza; the specific switch file that the queue manager uses is defined by the symbolic links you create in the next step.

   JDBC_DataSource is the identifier of the JDBCProvider configurable service (the value that you specified for the **-o** parameter on the **mqsichangeproperties** command).

   Define a stanza for each database (DSN) that you connect to from this broker. You must create separate definitions even if the DSNs resolve to the same physical database. Therefore, you must have a stanza for each JDBCProvider configurable service that you have defined, because each service can define the properties for a single database.

   b. Windows On Windows on x86 systems, open WebSphere MQ Explorer and select the queue manager for your broker, for example BROKERQM.

   Open the **XA resource manager** page, and modify the attributes to create the definition of the database. The attributes are the same as those shown for Linux and UNIX; **Name**, **SwitchFile**, **XAOpenString**, and **ThreadofControl**. Leave the additional attribute, **XACloseString**, blank.

   Enter the fully qualified file name in **SwitchFile**; *install_dir*`\bin\JDBCSwitch.dll`.

   c. Windows On Windows on x86-64 systems, open WebSphere MQ Explorer and select the queue manager for your broker, for example BROKERQM.

   Open the **XA resource manager** page, and modify the attributes to create the definition of the database. The attributes are the same as those shown for Linux and UNIX; **Name**, **SwitchFile**, **XAOpenString**, and **ThreadofControl**. Leave the additional attribute, **XACloseString**, blank.

   Enter *JDBCSwitch* in **SwitchFile**.

3. Set up queue manager access to the switch file:

   a. Linux UNIX On Linux and UNIX systems, create a symbolic link to the switch files that are supplied in your *install_dir*`/lib` directory.

   install_dir is the directory to which you installed the Broker component. The default location for this directory is `/opt/ibm/mqsi/`*v.r* on Linux, or `/opt/IBM/mqsi/`*v.r* on UNIX systems. The default directory includes the version and release of the product, in the format v.r (version.release).

   Set up links in the `/var/mqm/exits` directory, or the `/var/mqm/exits64` directory, or both. The file names for each platform are shown in the following table.

Platform	32-bit file	64-bit file
AIX		libJDBCSwitch.so
HP-Itanium		libJDBCSwitch.so
Linux on POWER		libJDBCSwitch.so
Linux on IBM z Systems		libJDBCSwitch.so
Linux on x86	libJDBCSwitch.so	
Linux on x86-64		libJDBCSwitch.so
Solaris on SPARC		libJDBCSwitch.so
Solaris on x86-64		libJDBCSwitch.so

Specify the same name of the switch file, JDBCSwitch or your own value, in both the /exits and /exits64 directories. For example, on AIX:

```
ln -s install_dir/lib/libJDBCSwitch.so /var/mqm/exits/JDBCSwitch

and

ln -s install_dir/lib/libJDBCSwitch.so /var/mqm/exits64/JDBCSwitch
```

b. Windows On Windows, for the 32-bit version of WebSphere Message Broker, copy the JDBCSwitch.dll file from the *install_dir*\bin directory to the \exits subdirectory in the WebSphere MQ installation directory.

c. Windows On Windows, for the 64-bit version of WebSphere Message Broker, copy the JDBCSwitch32.dll file from the *install_dir*\bin directory to the \exits subdirectory in the WebSphere MQ installation directory, and rename the file to JDBCSwitch.dll. Then, copy the JDBCSwitch.dll file from the *install_dir*\bin directory to the \exits64 subdirectory in the WebSphere MQ installation directory.

4. Configure the message flow that includes one or more nodes that access databases that are to participate in a globally coordinated transaction.
   a. Open a WebSphere Message Broker Toolkit session.
   b. Switch to the Broker Application Development perspective.
   c. Add the message flow that includes the node or nodes that connect to the database that is to participate in a globally coordinated transaction to a new or existing BAR file.
   d. Build the BAR file.
   e. Click the **Configure** tab, select the message flow that you have added, and select the **Coordinated Transaction** check box.

What to do next

Next: If your broker is running on Windows, authorize the broker and its queue manager to access resources associated with the JDBCProvider configurable service.

If you have been following the instructions in "Working with databases" on page 2109, the next task is "Configuring ODBC connections for globally coordinated transactions" on page 699 (optional).

Related concepts:

"Databases overview" on page 2109
Brokers can access *user databases* to manipulate your business data. If you have user databases, you must configure them before you can access them from your

message flow.

Related tasks:

"Setting up a JDBC provider for type 4 connections" on page 684
Use the **mqsicreateconfigurableservice** or the **mqsichangeproperties** command to configure a JDBC provider service.

"Authorizing access to JDBC type 4 JDBCProvider resources on Windows"
Authorize the broker and queue manager to access shared resources that are associated with the JDBCProvider. This task is required only if you want the database updates to be included in globally coordinated transactions on Windows systems.

"Configuring transactionality for message flows" on page 1290
A message flow runs in a single transaction, which is started when data is received by an input node, and can be committed or rolled back when all processing has completed.

"Interacting with databases by using the JavaCompute node" on page 2661
Access databases from Java code included in the JavaCompute node.

"Extending the capability of a Java message processing or output node" on page 3069
Within a message processing or output node, you can add extended functions to your Java node.

Related reference:

"Supported databases" on page 3591
You can optionally configure databases to contain data that is accessed by your message flows. Databases from IBM and other suppliers are supported at specific versions on supported operating systems.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

Authorizing access to JDBC type 4 JDBCProvider resources on Windows:

Authorize the broker and queue manager to access shared resources that are associated with the JDBCProvider. This task is required only if you want the database updates to be included in globally coordinated transactions on Windows systems.

Before you begin

Before you start: Set up your JDBC provider definition.

About this task

When the queue manager coordinates transactions, both queue manager and broker access shared memory to control a connection to the databases with which the message flow interacts. Therefore, they require the same access control of the shared memory. One method to achieve this control is to use the same ID for the broker service ID and the queue manager administrative ID.

Complete the following steps on the Windows system on which the broker is running:

Procedure

- If you defined the broker queue manager when you created the broker by running the **mqsicreatebroker** command, the two components share the same administrative ID, defined as the broker service ID, and you do not have to take further action.
- If you specified an existing queue manager when you created the broker, check that its administrative ID is the same ID as the one used for the service ID of the broker. If the ID is not the same, change the queue manager ID to be the same as the broker service ID:
  1. Click **Start** > **Run** and enter dcomcnfg. The Component Services window opens.
  2. In the left pane, expand **Component Services** > **Computers** > **My Computer** and click **DCOM Config**.
  3. In the right pane, right-click the WebSphere MQ service labeled **IBM MQSeries Services**, and click **Properties**.
  4. Click the **Identity** tab.
  5. Select **This user** and enter the user ID and password for the broker service ID to associate that ID with the queue manager.
  6. Click **OK** to confirm the change.

What to do next

Next: If you have been following the instructions in "Working with databases" on page 2109, the next task is "Configuring ODBC connections for globally coordinated transactions" on page 699 (optional).

Related concepts:

"Databases overview" on page 2109
Brokers can access *user databases* to manipulate your business data. If you have user databases, you must configure them before you can access them from your message flow.

Related tasks:

"Setting up a JDBC provider for type 4 connections" on page 684
Use the **mqsicreateconfigurableservice** or the **mqsichangeproperties** command to configure a JDBC provider service.

"Securing a JDBC type 4 connection" on page 689
Set up security for the JDBC connection if required by the database provider.

"Configuring a JDBC type 4 connection for globally coordinated transactions" on page 691
If you want the database that you access through a JDBC type 4 connection to participate in globally coordinated transactions, set up the appropriate environment.

"Interacting with databases by using the JavaCompute node" on page 2661
Access databases from Java code included in the JavaCompute node.

"Extending the capability of a Java message processing or output node" on page 3069
Within a message processing or output node, you can add extended functions to your Java node.

Related reference:

"Supported databases" on page 3591
You can optionally configure databases to contain data that is accessed by your message flows. Databases from IBM and other suppliers are supported at specific

versions on supported operating systems.

Connecting to a user database on z/OS:

Complete these tasks to connect to your user databases on z/OS.

Before you begin

Before you start: Create the broker and the database.

Procedure

1. Obtain the following DB2 values for the DB2 subsystem that you want to connect your broker to. Use these values to customize the `BIPEDIT`, `BIPDSNAO`, and `BIPBRKP` files.

JCL variable	Description	Example value
++DB2CONVERSION++	The DB2 converter	SINGLE
++DB2SUBSYSTEM++	The DB2 subsystem ID which the component connects	DFK4
++DB2LOCATION++	The DB2 location value of the DB2 subsystem to which the component connects	DSN910PK
++DB2CURRENTSQLID++	The default DB2 user ID for connecting	MQP1BRK
++DB2DSNACLIPLAN++	The DB2 plan name.	DSNACLI
++DB2HLQ++	The DB2 high-level-qualifier	SYS2.DB2.V910
++DB2RUNLIB++	The DB2 run library value	DSN910PK.RUNLIB.LOAD

2. Copy the supplied sample `BIPDSNAO` file into the component data set, and customize its content based on the values obtained in step 1. The stanza for your database will look like the following content:

```
[COMMON]
APPLTRACE=0
APPLTRACEfilename="DD:APPLTRC"
TRACETIMESTAMP=3
CONNECTTYPE=2
DIAGTRACE=0
DIAGTRACE_NO_WRAP=0
MAXCONN=0
MULTICONTEXT=0
MVSDEFAULTSSID=DFK4

; SUBSYSTEM
[DFK4]
MVSATTACHTYPE=RRSAF
PLANNAME=DSNACLI

; DATASOURCES
[DSN910PK]
CURRENTSQLID=MQP1BRK
```

3. Update the broker profile `BIPBPROF` in the component data set.
   a. Check that the environment variable `DSNAOINI` points to your copy of the `BIPDSNAO` file in the component data set. By default, it is set to the correct value; update this variable if you use a different name.
   b. Check that the environment variable `MQSI_DB2_CONVERSION` is set correctly.
4. Submit `BIPGEN` to re-create the broker `ENVFILE` and any execution group specific profiles.

5. Update the EGENV and EGNOENV steps in the broker started task JCL to include the following DB2 data sets:

```
//STEPLIB DD DISP=SHR,DSN=&DB2HLQ..SDSNEXIT
//        DD DISP=SHR,DSN=&DB2HLQ..SDSNLOAD
//        DD DISP=SHR,DSN=&DB2HLQ..SDSNLOD2
//        DD DISP=SHR,DSN=++DB2RUNLIB++
```

   Check that the PROC variable DB2HLQ='++DB2HLQ++' is not commented out.

6. Start the broker by using the **start (/S)** command.

What to do next

Next: You can set up your databases to participate in coordinated transactions by using the Resource Recovery Service (RRS). For more information, see "Resource Recovery Service planning on z/OS" on page 602.

Related reference:

"Supported databases" on page 3591
You can optionally configure databases to contain data that is accessed by your message flows. Databases from IBM and other suppliers are supported at specific versions on supported operating systems.

"Database locations" on page 3595
The broker can access databases set up on the local computer or on a remote server, subject to restrictions.

"Sample BIPDSNAO file" on page 4008
The sample BIPDSNAO file that is shipped with WebSphere Message Broker is included here for your reference.

"Sample BIPBPROF file" on page 3995
The sample BIPBPROF file that is shipped with WebSphere Message Broker is included here for your reference.

"**mqsistart** command" on page 3965
Use the **mqsistart** command to start the specified broker if all initial verification tests complete successfully.

Configuring global coordination of transactions (two-phase commit)

Globally coordinate message flow transactions with a transaction manager to ensure the data integrity of transactions. On distributed systems, the WebSphere MQ queue manager that is associated with the broker performs the transaction manager role.

Before you begin

Before you start:

Read "Message flow transactions" on page 1281 to understand how the broker handles transactions. Depending on the external resource managers that the broker accesses during the processing of its deployed message flows, you must complete the appropriate resource-dependent set of tasks to ensure that all resources are configured correctly. For example, you might have to create and configure databases.

You, or your message flow developer, must also ensure that the message flows deployed to the broker are set up to support coordination. The tasks involved in configuring the message flows correctly are described in "Configuring transactionality for message flows" on page 1290.

About this task

You can access the following external resources in a message flow transaction:

- WebSphere MQ queues and messages
- Databases
- JMS providers

On distributed platforms, the default behavior of the broker is to manage all message flow transactions by using a one-phase commit approach. In many contexts this approach is sufficient, but if your business requires assured data integrity and consistency (for example, for audit reasons, or for financial transactions), you can configure the broker and its WebSphere MQ queue manager to manage the message flow transactions in a two-stage commit approach, by using the XA protocol standard.

You configure the WebSphere MQ queue manager by updating its `qm.ini` file, to add definitions of the additional resource managers with which you want WebSphere MQ to coordinate updates. Follow the instructions provided for the resource managers that are relevant in your environment:

- ODBC connections to databases
- JDBC connections to databases
- JMS providers

z/OS On z/OS, all transactions are globally coordinated by Resource Recovery Service (RRS), therefore the instructions in this topic do not apply. However, you must ensure that RRS is available; see "Resource Recovery Service planning on z/OS" on page 602.

Results

When you have completed these steps, your message flows are processed by using global coordination, which is managed by the queue manager.

You must complete all the steps correctly; if you do not, global coordination will not work.

Example

For an example of how you can use WebSphere MQ to globally coordinate transactions, look at the following sample:

- Error Handler

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Related concepts:

"The transactional model" on page 1285
The transactional model describes the way in which you can use transactions in message flows to accomplish certain tasks and results.

Related tasks:

"Configuring transactionality for message flows" on page 1290
A message flow runs in a single transaction, which is started when data is received by an input node, and can be committed or rolled back when all processing has completed.

"Configuring databases for global coordination of transactions" on page 665
If your message flow interacts with a user database, and you want to globally coordinate the updates made to the database with other actions within the message flow, configure your databases for global coordination.

"Resource Recovery Service planning on z/OS" on page 602

"Enabling ODBC connections to the databases" on page 668
Set up the resources and environment that the broker requires for Open Database Connectivity (ODBC) connections to user databases on distributed systems.

"Enabling JDBC connections to the databases" on page 683
Configure connections to a user database through a JDBCProvider service.

"Creating a broker" on page 611
You can create brokers on every platform that is supported by WebSphere Message Broker. The broker runs as a 64-bit application on all platforms except Linux on x86 and Windows on x86.

Configuring ODBC connections for globally coordinated transactions:

Configure the definition of your ODBC databases to the transaction manager (the queue manager).

Before you begin

Before you start:

You must create and configure the databases by following the instructions in "Working with databases" on page 2109.

You, or your message flow developer, must also ensure that the message flows deployed to the broker are set up to support coordination. The tasks involved in configuring the message flows correctly are described in "Configuring transactionality for message flows" on page 1290.

Procedure

1. Ensure that your databases are configured for global coordination. For the databases that your message flows connect to, complete the tasks described in "Configuring databases for global coordination of transactions" on page 665.
2. Configure the broker environment so that the broker queue manager coordinates transactions with database resource managers. The steps to configure the broker environment depend on the database manager that you are using.
   - "Configuring global coordination with DB2" on page 700
   - "Configuring global coordination with Oracle" on page 705
   - "Configuring global coordination with Sybase" on page 710

Results

When you have completed these steps, your message flows are processed by using global coordination, which is managed by the queue manager.

You must complete all the steps correctly; if you do not, global coordination will not work.

Related concepts:

"The transactional model" on page 1285
The transactional model describes the way in which you can use transactions in message flows to accomplish certain tasks and results.

Related tasks:

"Configuring transactionality for message flows" on page 1290
A message flow runs in a single transaction, which is started when data is received by an input node, and can be committed or rolled back when all processing has completed.

"Configuring databases for global coordination of transactions" on page 665
If your message flow interacts with a user database, and you want to globally coordinate the updates made to the database with other actions within the message flow, configure your databases for global coordination.

"Resource Recovery Service planning on z/OS" on page 602

"Enabling ODBC connections to the databases" on page 668
Set up the resources and environment that the broker requires for Open Database Connectivity (ODBC) connections to user databases on distributed systems.

"Enabling JDBC connections to the databases" on page 683
Configure connections to a user database through a JDBCProvider service.

"Creating a broker" on page 611
You can create brokers on every platform that is supported by WebSphere Message Broker. The broker runs as a 64-bit application on all platforms except Linux on x86 and Windows on x86.

Configuring global coordination with DB2:

Configure your broker environment to globally coordinate message flow transactions with updates in DB2 databases under the control of your broker queue manager.

Before you begin

Ensure that the databases are configured for global coordination of transactions, see "Configuring databases for global coordination of transactions" on page 665.

Procedure

Follow the instructions appropriate to your platform:

- Linux UNIX "Linux and UNIX"
- Windows "Windows 32-bit" on page 702
- Windows "Windows 64-bit" on page 703

Linux and UNIX: Linux UNIX
Procedure

1. Linux On Linux on x86: decide whether the broker needs to connect to databases by using TCP/IP or shared memory.

   For more information about TCP/IP connections, see the example in the section about message SQL1224N in "Resolving problems when using databases" on page 3491.

To enable shared memory:

a. Stop the broker by running the following command, where *broker* is the name of your broker:

```
mqsistop broker
```

b. Run the following command to ensure that the broker is run in an environment with the extended memory variable exported:

```
export EXTSHM=ON
```

c. Restart the broker by running the following command, where *broker* is the name of your broker:

```
mqsistart broker
```

d. On the DB2 server, ensure that shared memory support is turned on. For more information, see "Configuring databases for global coordination of transactions" on page 665.

2. Linux UNIX Run the **mqsimanagexalinks** command. For more information, see "**mqsimanagexalinks** command" on page 3891.

3. Linux UNIX Configure the broker queue manager with XA resource manager information for each database that is involved in the transaction that the queue manager will globally coordinate.

a. Open the queue manager's `qm.ini` file in a text editor. The `qm.ini` file is located at `/var/mqm/qmgrs/`*queue_manager_name*`/qm.ini`, where *queue_manager_name* is the name of the broker that is associated with the queue manager.

b. At the end of the `qm.ini` file, paste the following stanza:

```
XAResourceManager:
Name=DB2
SwitchFile=db2swit
XAOpenString=db=MyDataSource,uid=MyUserId,pwd=MyPassword,toc=t
XACloseString=
ThreadOfControl=THREAD
```

The switch file is supplied by WebSphere Message Broker.

c. On the **XAOpenString** line, replace the following values with values that are appropriate for your configuration:

- *MyDataSource* is the name of the data source to which you want to connect.
- *MyUserId* must be the user name that the broker uses to connect to the database.

  You can define the user name that the broker uses in a number of ways; make sure that you specify the correct name in this file. The broker determines the user name by checking the following conditions in the order listed:

  1) A specific user name and password for this data source name (DSN), that you have defined by running the **mqsisetdbparms** command.
  2) A default user name and password for all DSNs, that you have defined by running the **mqsisetdbparms** command.
  3) A default user name and password for all DSNs, that you have defined by specifying the **-u** parameter on the **mqsicreatebroker** command.

     This parameter is valid only for brokers that you have migrated from previous releases.
  4) The broker service user name, which you define with the **-i** parameter on the **mqsicreatebroker** command

- *MyPassword* is the password that is associated with the user name.

d. Accept the default values for all the other lines in the stanza. For example:

```
XAResourceManager:
Name=DB2
SwitchFile=db2swit
XAOpenString=db=MYDB,uid=wbrkuid,pwd=wbrkpw,toc=t
XACloseString=
ThreadOfControl=THREAD
```

4. Linux UNIX Stop then restart the queue manager to apply the changes, because `qm.ini` is read only while the queue manager is running.

   To stop and restart the queue manager, enter the following commands, where *queue_manager_name* is the name of the queue manager:

   ```
   endmqm queue_manager_name
   strmqm queue_manager_name
   ```

   When the queue manager restarts, check the queue manager log for all warnings that are associated with the restart. The log files are located in `/var/mqm/qmgrs/`*queue_manager_name*`/errors`, where *queue_manager_name* is the name of the queue manager that you restarted. When the queue manager restarts successfully, the changes that you made to `qm.ini` are applied.

Results

DB2 is now configured for global coordination with the broker queue manager coordinating transactions.

What to do next

Next: you can deploy globally coordinated message flows to the broker.

Windows 32-bit: Windows
Procedure

1. Windows Configure the broker queue manager with XA resource manager information for each database that is involved in the transaction that the queue manager will globally coordinate.
   a. From the **Start** menu, open WebSphere MQ Explorer.
   b. Open the queue manager Properties dialog box, then open **XA resource managers**.
   c. Click the **Add** button to create a resource manager.
      1) In the **Name** field, enter a name to refer to a resource manager.
      2) In the **SwitchFile** field, enter the full path to the switch file, as shown in the following example where *install_dir* is the location in which the broker is installed:

         ```
         install_dir\sample\xatm\db2swit.dll
         ```

      3) In the **XAOpenString** field, paste the following string:

         ```
         db=MyDataSource,uid=MyUserId,pwd=MyPassword,toc=t
         ```

      4) In the **XAOpenString** field, replace the values with values that are appropriate for your configuration:
         a) *MyDataSource* is the name of the data source to which you want to connect.
         b) *MyUserId* must be the user name that the broker uses to connect to the database.

You can define the user name that the broker uses in a number of ways; make sure that you specify the correct name in this file. The broker determines the user name by checking the following conditions in the order listed:

i. A specific user name and password for this data source name (DSN), that you have defined by running the **mqsisetdbparms** command.

ii. A default user name and password for all DSNs, that you have defined by running the **mqsisetdbparms** command.

iii. A default user name and password for all DSNs, that you have defined by specifying the **-u** parameter on the **mqsicreatebroker** command.

This parameter is valid only for brokers that you have migrated from previous releases.

iv. The broker service user name, which you define with the **-i** parameter on the **mqsicreatebroker** command

c) *MyPassword* is the password that is associated with the user name.

For example:

```
db=MYDB,uid=wbrkuid,pwd=wbrkpw,toc=t
```

5) Accept the default values for all the other fields on the page.

2. Windows Stop then restart the queue manager to apply the changes.

To stop and restart the queue manager, enter the following commands, where *queue_manager_name* is the name of the queue manager:

```
endmqm queue_manager_name
strmqm -si queue_manager_name
```

When the queue manager restarts, check the queue manager log for all warnings that are associated with the restart. The log files are located in *install_dir*\WebSphere MQ\Qmgrs\MB7QMGR\errors, where *install_dir* is the location in which the broker is installed. When the queue manager restarts successfully, the changes that you made are applied.

Results

DB2 is now configured for global coordination with the broker queue manager coordinating transactions.

What to do next

Next: you can deploy globally coordinated message flows to the broker.

Windows 64-bit: Windows

Procedure

1. Windows Configure the broker queue manager with XA resource manager information for each database that is involved in the transaction that the queue manager will globally coordinate.

a. From the **Start** menu, open WebSphere MQ Explorer.

b. Open the queue manager Properties dialog box, then open **XA resource managers**.

c. Click the **Add** button to create a resource manager.

1) In the **Name** field, enter a name to refer to a resource manager.

2) In the **SwitchFile** field, enter *db2swit*

3) Copy the provided DB2 switch files from the broker install location to the queue managers exits and *exits64* directories (by default under `C:\Program Files (x86)\IBM\WebSphere MQ`).

   Copy the file:

   ```
   install_dir\sample\xatm\db2swit32.dll
   ```

   into the queue managers exits directory, and rename it to *db2swit.dll*. Copy the file

   ```
   install_dir\sample\xatm\db2swit.dll
   ```

   into the queue managers *exits64* directory.

4) In the **XAOpenString** field, paste the following string:

   ```
   db=MyDataSource,uid=MyUserId,pwd=MyPassword,toc=t
   ```

5) In the **XAOpenString** field, replace the values with values that are appropriate for your configuration:

   a) *MyDataSource* is the name of the data source to which you want to connect.

   b) *MyUserId* must be the user name that the broker uses to connect to the database.

      You can define the user name that the broker uses in a number of ways; make sure that you specify the correct name in this file. The broker determines the user name by checking the following conditions in the order listed:

      i. A specific user name and password for this data source name (DSN), that you have defined by running the **mqsisetdbparms** command.

      ii. A default user name and password for all DSNs, that you have defined by running the **mqsisetdbparms** command.

      iii. A default user name and password for all DSNs, that you have defined by specifying the **-u** parameter on the **mqsicreatebroker** command.

      This parameter is valid only for brokers that you have migrated from previous releases.

      iv. The broker service user name, which you define with the **-i** parameter on the **mqsicreatebroker** command

   c) *MyPassword* is the password that is associated with the user name.

   For example:

   ```
   db=MYDB,uid=wbrkuid,pwd=wbrkpw,toc=t
   ```

6) Accept the default values for all the other fields on the page.

2. Windows Stop then restart the queue manager to apply the changes.

   To stop and restart the queue manager, enter the following commands, where *queue_manager_name* is the name of the queue manager:

   ```
   endmqm queue_manager_name
   strmqm -si queue_manager_name
   ```

   When the queue manager restarts, check the queue manager log for all warnings that are associated with the restart. The log files are located in *install_dir*`\WebSphere MQ\Qmgrs\MB7QMGR\errors`, where *install_dir* is the location in which the broker is installed. When the queue manager restarts successfully, the changes that you made are applied.

Results

DB2 is now configured for global coordination with the broker queue manager coordinating transactions.

What to do next

Next: you can deploy globally coordinated message flows to the broker.

Related tasks:

"Configuring transactionality for message flows" on page 1290
A message flow runs in a single transaction, which is started when data is received by an input node, and can be committed or rolled back when all processing has completed.

"Configuring ODBC connections for globally coordinated transactions" on page 699
Configure the definition of your ODBC databases to the transaction manager (the queue manager).

Related reference:

"**mqsimanagexalinks** command" on page 3891
Use the **mqsimanagexalinks** command to set up links for a supported database on and UNIX systems for XA coordination under the control of WebSphere MQ.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"Supported databases" on page 3591
You can optionally configure databases to contain data that is accessed by your message flows. Databases from IBM and other suppliers are supported at specific versions on supported operating systems.

Related information:

WebSphere MQ Version 7 Information Center online

Configuring global coordination with Oracle:

Configure your broker environment to globally coordinate message flow transactions with updates in Oracle databases under the control of a WebSphere MQ queue manager.

Before you begin

Before you start:

- Ensure that you have configured the databases for global coordination of transactions.

About this task

Configure your broker environment for global coordination by using a 64-bit queue manager as the transaction manager with the DataDirect drivers:

Procedure

1. Linux UNIX On Linux and UNIX, run the **mqsimanagexalinks** command.

2. Configure the broker queue manager with XA resource manager information for each database that is involved in the transaction that the queue manager will globally coordinate.

   Linux UNIX **On Linux (except Linux on x86) and UNIX:**

   a. Open the queue managers `qm.ini` file in a text editor. The `qm.ini` file is located at `/var/mqm/qmgrs/`*queue_manager_name*`/qm.ini`. Where *queue_manager_name* is the name of the broker that is associated with the queue manager.

   b. Add one of the following stanzas to the end of the `qm.ini` file:

      - If you are not using Oracle Real Application Clusters:

```
XAResourceManager:
Name=OracleXA
SwitchFile=UKoradtc24.so
XAOpenString=ORACLE_XA
   +HostName=MyHostName
   +PortNumber=MyPortNumber
   +Sid=MySID
   +ACC=P/MyUserId/MyPassword
   +sestm=100+threads=TRUE
   +DataSource=MyDataSourceName
   +K=2+
XACloseString=
ThreadOfControl=THREAD
```

      - If you are using Oracle Real Application Clusters:

```
XAResourceManager:
Name=OracleXA
SwitchFile=UKoradtc24.so
XAOpenString=ORACLE_XA
   +HostName=MyHostName
   +PortNumber=MyPortNumber
   +ServiceName=MyServiceName
   +ACC=P/MyUserId/MyPassword
   +sestm=100+threads=TRUE
   +DataSource=MyDataSourceName
   +K=2+
XACloseString=
ThreadOfControl=THREAD
```

   c. On the **XAOpenString** line, replace the following values with values that are appropriate for your configuration:

      - *MyHostName* is the name of the TCP/IP host that hosts the Oracle database listener. When you are using Oracle Real Application Clusters with multiple listeners for the given Service Name, if the Oracle listener identified by the values for *MyHostName* and *MyPortNumber* in the **XAOpenString** is unavailable, the alternative Oracle listeners that you might have defined in the **AlternateServers** list in your `odbc.ini` file are also tried.
      - *MyPortNumber* is the TCP/IP port on which the Oracle database listener is listening.
      - *MySID* is the Oracle System Identifier (SID) of the database.
      - *MyUserId* must be the user name that the broker uses to connect to the database.

        You can define the user name that the broker uses in a number of ways; make sure that you specify the correct name in this file. The broker determines the user name by checking the following conditions in the order listed:

1) A specific user name and password for this data source name (DSN), that you have defined by running the **mqsisetdbparms** command.
2) A default user name and password for all DSNs, that you have defined by running the **mqsisetdbparms** command.
3) A default user name and password for all DSNs, that you have defined by specifying the **-u** parameter on the **mqsicreatebroker** command.

   This parameter is valid only for brokers that you have migrated from previous releases.
4) The broker service user name, which you define with the **-i** parameter on the **mqsicreatebroker** command

- *MyPassword* is the password that is associated with the user name.
- *MyDataSourceName* is the ODBC data source name for the database, as defined in your `odbc.ini` file.
- *MyServiceName* is the value set for the Service Name in the stanza for *MyDataSourceName* in your `odbc.ini` file.

Note: The **XAOpenString** might also contain the lines `+SQLNET=MyNetServiceName` and `+DB=MyDataSourceName`, but these lines are ignored.

d. Accept the default values for all the other lines in the stanza. For example:
   - On AIX:
     - If you are not using Oracle Real Application Clusters:

```
XAResourceManager:
Name=OracleXA
SwitchFile=UKoradtc24.so
XAOpenString=ORACLE_XA
   +HostName=diaz.hursley.ibm.com
   +PortNumber=1521+Sid=diaz
   +ACC=P/wbrkuid/wbrkpw
   +sestm=100+threads=TRUE
   +DataSource=MYDB+K=2+
XACloseString=
ThreadOfControl=THREAD
```

     - If you are using Oracle Real Application Clusters:

```
XAResourceManager:
Name=OracleXA
SwitchFile=UKoradtc24.so
XAOpenString=ORACLE_XA
   +HostName=diaz.hursley.ibm.com
   +PortNumber=1521
   +ServiceName=racxa.test.com
   +ACC=P/wbrkuid/wbrkpw
   +sestm=100+threads=TRUE
   +DataSource=MYDB+K=2+
XACloseString=
ThreadOfControl=THREAD
```

e. If you are using Oracle Real Application Clusters:
   - Set the ODBCINI environment variable to be visible in the environment from which you start your queue manager. The ODBCINI variable must reference the same file as the one being used by your broker.
   - An optional additional property, `CTO=`*`Value`*, is available for the **XAOpenString**. `CTO` is the value set for a Connection TimeOut, indicating the number of seconds that the Oracle XA switch file will wait for a response from the Oracle database to an XA request. For example, the timeout can be used to prevent long delays in the broker failing over to

an alternative Oracle Real Application Clusters node, when the active Oracle instance fails abruptly leaving socket connections hanging. The *Value* should be set to a value larger than the Oracle session timeout `sestm` value set in the **XAOpenString**. If this property is not used, or is set to zero, there will be no timeout (this is the default behaviour).

Windows **On Windows 32-bit**

a. From the **Start** menu, open WebSphere MQ Explorer.

b. Open the queue manager Properties dialog box, then open **XA resource managers**.

c. In the **SwitchFile** field, enter the full path to the switch file, as shown in the following example. Where *install_dir* is the location in which the broker is installed:

```
install_dir\bin\UKora24.dll
```

d. In the **XAOpenString** field, paste the following string:

- If you are not using Oracle Real Application Clusters:

```
ORACLE_XA
   +HostName=MyHostName
   +PortNumber=MyPortNumber
   +Sid=MySID
   +ACC=P/MyUserId/MyPassword
   +sestm=100+threads=TRUE
   +DataSource=MyDataSourceName
   +K=2+
```

- If you are using Oracle Real Application Clusters:

```
ORACLE_XA
   +HostName=MyHostName
   +PortNumber=MyPortNumber
   +ServiceName=MyServiceName
   +ACC=P/MyUserId/MyPassword
   +sestm=100+threads=TRUE
   +DataSource=MyDataSourceName
   +K=2+
```

e. In the **XAOpenString** field, replace the values with values that are appropriate for your configuration:

- *MyHostName* is the name of the TCP/IP host that hosts the Oracle database listener. When you are using Oracle Real Application Clusters with multiple listeners for the given Service Name, if the Oracle listener identified by the values for *MyHostName* and *MyPortNumber* in the **XAOpenString** is unavailable, the alternative Oracle listeners that you might have defined in the **AlternateServers** list in your `odbc.ini` file arealso tried.
- *MyPortNumber* is the TCP/IP port on which the Oracle database listener is listening.
- *MySID* is the Oracle System Identifier (SID) of the database.
- *MyUserId* must be the user name that the broker uses to connect to the database.

  You can define the user name that the broker uses in a number of ways; make sure that you specify the correct name in this file. The broker determines the user name by checking the following conditions in the order listed:

  1) A specific user name and password for this data source name (DSN), that you have defined by running the **mqsisetdbparms** command.
  2) A default user name and password for all DSNs, that you have defined by running the **mqsisetdbparms** command.

3) A default user name and password for all DSNs, that you have defined by specifying the **-u** parameter on the **mqsicreatebroker** command.

   This parameter is valid only for brokers that you have migrated from previous releases.

4) The broker service user name, which you define with the **-i** parameter on the **mqsicreatebroker** command

- *MyPassword* is the password that is associated with the user name.
- *MyDataSourceName* is the ODBC data source name for the database, as defined in your `odbc.ini` file.
- *MyServiceName* is the value set for the Service Name in the stanza for *MyDataSourceName* in your `odbc.ini` file.

Note: The **XAOpenString** might also contain the lines `+SQLNET=MyNetServiceName` and `+DB=MyDataSourceName`, but these lines are ignored.

For example:

- If you are not using Oracle Real Application Clusters:

```
ORACLE_XA+SQLNET=diaz
   +HostName=diaz.hursley.ibm.com
   +PortNumber=1521+Sid=diaz
   +ACC=P/wbrkuid/wbrkpw
   +sestm=100+threads=TRUE
   +DataSource=MYDB+DB=MYDB+K=2+
```

- If you are using Oracle Real Application Clusters:

```
ORACLE_XA+SQLNET=WMBSERVICE
   +HostName=diaz.hursley.ibm.com
   +PortNumber=1521
   +ServiceName=racxa.test.com
   +ACC=P/wbrkuid/wbrkpw
   +sestm=100+threads=TRUE
   +DataSource=MYDB+DB=MYDB+K=2+
```

f. Accept the default values for all the other fields on the page.

3. AIX On AIX, if you want to enable Oracle data sources for use in global coordination from a queue manager and broker to perform dynamic XA registration, set the following environment variable:
   `DDTEK_XA_DYNAMIC_REGISTRATION=1`

4. Linux UNIX Stop then restart the queue manager to apply the changes, because `qm.ini` is read only while the queue manager is running.

   To stop and restart the queue manager, enter the following commands, where *queue_manager_name* is the name of the queue manager:

   ```
   endmqm queue_manager_name
   strmqm queue_manager_name
   ```

   When the queue manager restarts, check the queue manager log for all warnings that are associated with the restart. The log files are located in `/var/mqm/qmgrs/`*queue_manager_name*`/errors`, where *queue_manager_name* is the name of the queue manager that you restarted. When the queue manager restarts successfully, the changes that you made to `qm.ini` are applied.

5. Windows Stop then restart the queue manager to apply the changes.

   To stop and restart the queue manager, enter the following commands, where *queue_manager_name* is the name of the queue manager:

   ```
   endmqm queue_manager_name
   strmqm -si queue_manager_name
   ```

When the queue manager restarts, check the queue manager log for all warnings that are associated with the restart. The log files are located in `/var/mqm/qmgrs/queue_manager_name/errors`, where *queue_manager_name* is the name of the queue manager that you restarted. When the queue manager restarts successfully, the changes that you made are applied.

Results

Oracle is now configured for global coordination with the broker queue manager coordinating transactions.

What to do next

Next: You can deploy globally coordinated message flows to the broker.

Related tasks:

"Configuring transactionality for message flows" on page 1290
A message flow runs in a single transaction, which is started when data is received by an input node, and can be committed or rolled back when all processing has completed.

"Configuring ODBC connections for globally coordinated transactions" on page 699
Configure the definition of your ODBC databases to the transaction manager (the queue manager).

Related reference:

"Supported databases" on page 3591
You can optionally configure databases to contain data that is accessed by your message flows. Databases from IBM and other suppliers are supported at specific versions on supported operating systems.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsimanagexalinks** command" on page 3891
Use the **mqsimanagexalinks** command to set up links for a supported database on and UNIX systems for XA coordination under the control of WebSphere MQ.

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

Related information:

WebSphere MQ Version 7 Information Center online

Configuring global coordination with Sybase:

Configure your broker environment to globally coordinate message flow transactions with updates in Sybase databases under the control of a queue manager.

Before you begin

Before you start:
- Ensure that the databases are configured for global coordination of transactions.

About this task

To configure your broker environment for global coordination using a WebSphere MQ queue manager as the transaction manager with the DataDirect drivers:

Procedure

1. Linux UNIX On Linux and UNIX, run the **mqsimanagexalinks** command.
2. Configure the broker queue manager with XA resource manager information for each database that is involved in the transaction that the queue manager will globally coordinate.

   Linux UNIX **On Linux and UNIX:**

   a. Open the queue manager's `qm.ini` file in a text editor. The `qm.ini` file is located at `/var/mqm/qmgrs/queue_manager_name/qm.ini`, where *queue_manager_name* is the name of the broker that is associated with the queue manager.

   b. At the end of the `qm.ini` file, paste the following stanza:

   ```
   XAResourceManager:
       Name=SYBASEXA
       SwitchFile=UKasedtc24.so
       XAOpenString=-NSYBASEDB -AMyServerName,MyPortNumber -Uuid -Ppwd -K2
       XACloseString=
       ThreadOfControl=THREAD
   ```

   c. On the **XAOpenString** line, replace the following values with values that are appropriate for your configuration:
      - *MyServerName* is the name of the TCP/IP host that hosts the Sybase ASE server.
      - *MyPortNumber* is the TCP/IP port on which the Sybase ASE server is listening.
      - *uid* must be the user name that the broker uses to connect to the database.

        You can define the user name that the broker uses in a number of ways; make sure that you specify the correct name in this file. The broker determines the user name by checking the following conditions in the order listed:

        1) A specific user name and password for this data source name (DSN), that you have defined by running the **mqsisetdbparms** command.
        2) A default user name and password for all DSNs, that you have defined by running the **mqsisetdbparms** command.
        3) A default user name and password for all DSNs, that you have defined by specifying the **-u** parameter on the **mqsicreatebroker** command.

           This parameter is valid only for brokers that you have migrated from previous releases.
        4) The broker service user name, which you define with the **-i** parameter on the **mqsicreatebroker** command
      - *pwd* is the password that is associated with the user name.

   d. Accept the default values for all the other lines in the stanza. For example:
      - On AIX:

```
XAResourceManager:
    Name=SYBASEXA
    SwitchFile=UKasedtc24.so
    XAOpenString=-NSYBASEDB -Adiaz,1521 -Uwbrkuid -Pwbrkpw -K2
    XACloseString=
    ThreadOfControl=THREAD
```

Windows **On Windows 32-bit:**

a. From the **Start** menu, open WebSphere MQ Explorer.

b. Open the Properties dialog box for the queue manager, then open **XA resource managers**.

c. In the **SwitchFile** field, enter the full path to the switch file, as shown in the following example where *install_dir* is the location in which the broker is installed:

install_dir\bin\ukase24.dll

d. In the **XAOpenString** field, paste the following string:

-NSYBASEDB -A*MyServerName*,*MyPortNumber* -WWinsock -U*uid* -P*pwd* -K2

e. In the **XAOpenString** field, replace the values with values that are appropriate for your configuration:

- *install_dir* is the location in which the broker is installed.
- *MyServerName* is the name of the TCP/IP host that hosts the Sybase ASE server.
- *MyPortNumber* is the TCP/IP port on which the Sybase ASE server is listening.
- *uid* must be the user name that the broker uses to connect to the database.

  You can define the user name that the broker uses in a number of ways; make sure that you specify the correct name in this file. The broker determines the user name by checking the following conditions in the order listed:

  1) A specific user name and password for this data source name (DSN), that you have defined by running the **mqsisetdbparms** command.
  2) A default user name and password for all DSNs, that you have defined by running the **mqsisetdbparms** command.
  3) A default user name and password for all DSNs, that you have defined by specifying the **-u** parameter on the **mqsicreatebroker** command.

     This parameter is valid only for brokers that you have migrated from previous releases.
  4) The broker service user name, which you define with the **-i** parameter on the **mqsicreatebroker** command
- *pwd* is the password that is associated with the user name.

  For example:

  -NSYBASEDB -Adiaz,1521 -WWinsock -Uwbrkuid -Pwbrkpw -K2

f. Accept the default values for all the other fields on the page.

3. Stop then restart the queue manager to apply the changes, because qm.ini is read only while the queue manager is running.

   To stop and restart the queue manager, enter the following commands, where *queue_manager_name* is the name of the queue manager:

```
endmqm queue_manager_name
strmqm queue_manager_name
```

When the queue manager restarts, check the queue manager log for all warnings that are associated with the restart. The log files are located in `/var/mqm/qmgrs/queue_manager_name/errors`, where *queue_manager_name* is the name of the queue manager that you restarted. When the queue manager restarts successfully, the changes that you made to `qm.ini` are applied.

Results

Sybase is now configured for global coordination with your queue manager coordinating transactions.

What to do next

Next: you can deploy globally coordinated message flows to the broker.

Related tasks:

"Configuring transactionality for message flows" on page 1290
A message flow runs in a single transaction, which is started when data is received by an input node, and can be committed or rolled back when all processing has completed.

"Configuring ODBC connections for globally coordinated transactions" on page 699
Configure the definition of your ODBC databases to the transaction manager (the queue manager).

Related reference:

"Supported databases" on page 3591
You can optionally configure databases to contain data that is accessed by your message flows. Databases from IBM and other suppliers are supported at specific versions on supported operating systems.

"**mqsicreatebroker** command - Linux and UNIX systems" on page 3835
Use the **mqsicreatebroker** command to create a broker on a Linux or UNIX systems.

"**mqsicreatebroker** command - Windows systems" on page 3840
Use the **mqsicreatebroker** command to create a broker on a Windows system.

"**mqsimanagexalinks** command" on page 3891
Use the **mqsimanagexalinks** command to set up links for a supported database on and UNIX systems for XA coordination under the control of WebSphere MQ.

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

Related information:

WebSphere MQ Version 7 Information Center online

Configuring a JDBC type 4 connection for globally coordinated transactions:

If you want the database that you access through a JDBC type 4 connection to participate in globally coordinated transactions, set up the appropriate environment.

Before you begin

Before you start: Set up your JDBC provider definition.

About this task

Updates that you make to a database across a JDBC type 4 connection can be coordinated with other actions taken within the message flow, if you set up the resources to support coordination.

Complete the following steps:

Procedure

1. Check that the definition of your JDBCProvider service is appropriate for coordinated transactions.

   For example, to set up the required JDBC classes:

   - For DB2, set **type4DatasourceClassName** to `com.ibm.db2.jcc.DB2XADataSource` and **type4DriverClassName** to `com.ibm.db2.jcc.DB2Driver`
   - For Oracle, set **type4DatasourceClassName** to `oracle.jdbc.xa.client.OracleXADataSource` and **type4DriverClassName** to `oracle.jdbc.OracleDriver`

   Consult your database administrator or the documentation provided by your database supplier, to confirm that all the JDBCProvider service properties are set appropriately. For example, a database supplier might require secure access if it is participating in coordinated transactions.

2. Define the switch file and the database properties:

   a. Linux UNIX On Linux and UNIX systems, open the `qm.ini` file for the broker queue manager with a text editor. Add the following stanza for each database:

   ```
   XAResourceManager:
          Name=Database_Name
          SwitchFile=JDBCSwitch
          XAOpenString=JDBC_DataSource
          ThreadOfControl=THREAD
   ```

   Database_Name is the database name (DSN) of the database defined to the JDBCProvider configurable service (for example, specified by `-n databaseName -v` *Database_Name* on the **mqsichangeproperties** command).

   `JDBCSwitch` is a fixed generic name that represents the switch file for XA coordination. Use this value, or another single fixed value, in each stanza; the specific switch file that the queue manager uses is defined by the symbolic links you create in the next step.

   JDBC_DataSource is the identifier of the JDBCProvider configurable service (the value that you specified for the **-o** parameter on the **mqsichangeproperties** command).

   Define a stanza for each database (DSN) that you connect to from this broker. You must create separate definitions even if the DSNs resolve to the same physical database. Therefore, you must have a stanza for each JDBCProvider configurable service that you have defined, because each service can define the properties for a single database.

   b. Windows On Windows on x86 systems, open WebSphere MQ Explorer and select the queue manager for your broker, for example BROKERQM.

   Open the **XA resource manager** page, and modify the attributes to create the definition of the database. The attributes are the same as those shown for Linux and UNIX; **Name**, **SwitchFile**, **XAOpenString**, and **ThreadofControl**. Leave the additional attribute, **XACloseString**, blank.

Enter the fully qualified file name in **SwitchFile**; *install_dir*\bin\JDBCSwitch.dll.

c. Windows On Windows on x86-64 systems, open WebSphere MQ Explorer and select the queue manager for your broker, for example BROKERQM.

Open the **XA resource manager** page, and modify the attributes to create the definition of the database. The attributes are the same as those shown for Linux and UNIX; **Name**, **SwitchFile**, **XAOpenString**, and **ThreadofControl**. Leave the additional attribute, **XACloseString**, blank.

Enter *JDBCSwitch* in **SwitchFile**.

3. Set up queue manager access to the switch file:

a. Linux UNIX On Linux and UNIX systems, create a symbolic link to the switch files that are supplied in your *install_dir*/lib directory.

install_dir is the directory to which you installed the Broker component. The default location for this directory is /opt/ibm/mqsi/*v.r* on Linux, or /opt/IBM/mqsi/*v.r* on UNIX systems. The default directory includes the version and release of the product, in the format v.r (version.release).

Set up links in the /var/mqm/exits directory, or the /var/mqm/exits64 directory, or both. The file names for each platform are shown in the following table.

Platform	32-bit file	64-bit file
AIX		libJDBCSwitch.so
HP-Itanium		libJDBCSwitch.so
Linux on POWER		libJDBCSwitch.so
Linux on IBM z Systems		libJDBCSwitch.so
Linux on x86	libJDBCSwitch.so	
Linux on x86-64		libJDBCSwitch.so
Solaris on SPARC		libJDBCSwitch.so
Solaris on x86-64		libJDBCSwitch.so

Specify the same name of the switch file, JDBCSwitch or your own value, in both the /exits and /exits64 directories. For example, on AIX:

```
ln -s install_dir/lib/libJDBCSwitch.so /var/mqm/exits/JDBCSwitch

and

ln -s install_dir/lib/libJDBCSwitch.so /var/mqm/exits64/JDBCSwitch
```

b. Windows On Windows, for the 32-bit version of WebSphere Message Broker, copy the JDBCSwitch.dll file from the *install_dir*\bin directory to the \exits subdirectory in the WebSphere MQ installation directory.

c. Windows On Windows, for the 64-bit version of WebSphere Message Broker, copy the JDBCSwitch32.dll file from the *install_dir*\bin directory to the \exits subdirectory in the WebSphere MQ installation directory, and rename the file to JDBCSwitch.dll. Then, copy the JDBCSwitch.dll file from the *install_dir*\bin directory to the \exits64 subdirectory in the WebSphere MQ installation directory.

4. Configure the message flow that includes one or more nodes that access databases that are to participate in a globally coordinated transaction.

a. Open a WebSphere Message Broker Toolkit session.

b. Switch to the Broker Application Development perspective.
c. Add the message flow that includes the node or nodes that connect to the database that is to participate in a globally coordinated transaction to a new or existing BAR file.
d. Build the BAR file.
e. Click the **Configure** tab, select the message flow that you have added, and select the **Coordinated Transaction** check box.

What to do next

Next: If your broker is running on Windows, authorize the broker and its queue manager to access resources associated with the JDBCProvider configurable service.

If you have been following the instructions in "Working with databases" on page 2109, the next task is "Configuring ODBC connections for globally coordinated transactions" on page 699 (optional).

Related concepts:

"Databases overview" on page 2109
Brokers can access *user databases* to manipulate your business data. If you have user databases, you must configure them before you can access them from your message flow.

Related tasks:

"Setting up a JDBC provider for type 4 connections" on page 684
Use the **mqsicreateconfigurableservice** or the **mqsichangeproperties** command to configure a JDBC provider service.

"Authorizing access to JDBC type 4 JDBCProvider resources on Windows" on page 694
Authorize the broker and queue manager to access shared resources that are associated with the JDBCProvider. This task is required only if you want the database updates to be included in globally coordinated transactions on Windows systems.

"Configuring transactionality for message flows" on page 1290
A message flow runs in a single transaction, which is started when data is received by an input node, and can be committed or rolled back when all processing has completed.

"Interacting with databases by using the JavaCompute node" on page 2661
Access databases from Java code included in the JavaCompute node.

"Extending the capability of a Java message processing or output node" on page 3069
Within a message processing or output node, you can add extended functions to your Java node.

Related reference:

"Supported databases" on page 3591
You can optionally configure databases to contain data that is accessed by your message flows. Databases from IBM and other suppliers are supported at specific versions on supported operating systems.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

Configuring properties to connect to external resources

You can use configurable services to prepare the environment for external resources, or change connection details for external resources.

About this task

The following topics describe how to prepare your environment to connect to external resources.

- "Configuring for Enterprise Information Systems"
- "Configuring for IMS" on page 731
- Configuring for CORBA
- "Configuring for CICS Transaction Server for z/OS" on page 736
- "Configuring for JMS" on page 747
- Configuring for WS-Security
- Configuring for email

Configuring for Enterprise Information Systems:

You can use configurable services to enable the WebSphere Adapters nodes in the broker runtime environment to connect with Enterprise Information Systems such as SAP, Siebel, and PeopleSoft.

About this task

The following topics describe how to prepare the environment to connect to Enterprise Information Systems, and how to change the connection details for adapters without the need to redeploy them.

- "Preparing the environment for WebSphere Adapters nodes"
- "Changing connection details for SAP adapters" on page 719
- "Changing connection details for Siebel adapters" on page 720
- "Changing connection details for PeopleSoft adapters" on page 722
- "Changing connection details for JD Edwards adapters" on page 724
- "Configuring EIS connections to expire after a specified time" on page 726

Preparing the environment for WebSphere Adapters nodes:

Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

Before you begin

Before you start:

Read "WebSphere Adapters nodes" on page 1914.

About this task

To enable the WebSphere Adapters nodes in the broker runtime environment, configure the broker with the location of the EIS provider JAR files and native libraries. (On Windows, the location of the JAR files cannot be a mapped network drive on a remote Windows computer; the directory must be local or on a Storage Area Network (SAN) disk.)

Procedure

1. The WebSphere Adapters nodes require libraries from the EIS vendors. For more information on how to obtain and use these libraries, see "Developing message flows that use WebSphere Adapters" on page 2033.
2. Use the following commands to make the JAR files and shared libraries available to the WebSphere Adapters nodes.
   - To set up the dependencies, use the following command.

     ```
     mqsichangeproperties broker name -c EISProviders -o EIS type -n jarsURL -v jar directory
     mqsichangeproperties broker name -c EISProviders -o EIS type -n nativeLibs -v bin directory
     ```

     For example:

     ```
     mqsichangeproperties brk6 -c EISProviders -o SAP -n jarsURL -v c:\sapjco\jars
     mqsichangeproperties brk6 -c EISProviders -o SAP -n nativeLibs -v c:\sapjco\bin
     ```

     After you have run the **mqsichangeproperties** command, restart the broker so that the changes take effect.
   - To report the dependencies, use the following command.

     ```
     mqsireportproperties broker name -c EISProviders -o EIS type -r
     ```

     For example:

     ```
     mqsireportproperties brk6 -c EISProviders -o SAP -r
     ```

   - On z/OS, run this command by customizing and submitting BIPJADPR.

   You can also connect to different versions of Siebel by creating a custom EISProviders configurable service and setting the location of the appropriate library files. For more information, see "Connecting to different versions of Siebel" on page 2077.

What to do next

When you have set up the environment for the WebSphere Adapters nodes, you can perform the preparatory tasks that are listed in "Developing message flows that use WebSphere Adapters" on page 2033.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

Related tasks:

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

Changing connection details for SAP adapters:

SAP nodes can get SAP connection details from either the adapter component or a configurable service. By using configurable services, you can change the connection details for adapters without the need to redeploy the adapters. To pick up new values when a configurable service is created or modified, you must reload the broker or execution group to which the adapter was deployed, by using the **mqsistop** and **mqsistart** commands, or the **mqsireload** command.

Before you begin

Before you start:

- Read "WebSphere Adapters nodes" on page 1914 and "Configurable services" on page 1296 for background information.

About this task

Use the SAPConnection configurable service to change connection details for an SAP adapter. The SAP node reads all connection properties from the adapter component that it is configured to use. If a configurable service exists that has the same name as the node's adapter component, the node uses the values that are defined in that configurable service to override the corresponding properties from the adapter. If any properties on the configurable service are set to an empty string, the values that are configured in the `.inadapter` or `.outadapter` file are used. The properties of the SAP configurable services are described in "Configurable services properties" on page 3766.

Creating, changing, reporting, and deleting configurable services

Procedure

- To create a configurable service, use the WebSphere Message Broker Explorer; see "Using the WebSphere Message Broker Explorer to work with configurable services" on page 644 for more information, or the **mqsicreateconfigurableservice** command, as shown in the following example. This example creates an SAPConnection configurable service for the SAP adapter *mySAPAdapter.outadapter* that connects to the SAP host `test.sap.ibm.com`, and uses client *001* for the connections into that server:

  ```
  mqsicreateconfigurableservice MB7BROKER -c SAPConnection
  -o mySAPAdapter.outadapter -n applicationServerHost,client
  -v test.sap.ibm.com,001
  ```

  To pick up the new values in the configurable service, restart the execution group and message flow.
- To change a configurable service, use the WebSphere Message Broker Explorer, or the **mqsichangeproperties** command, as shown in the following example. This example changes the connections that are used by the adapter *mySAPAdapter.outadapter*. After you run this command, all adapters connect to the production system (`production.sap.ibm.com`) instead of the test system (`test.sap.ibm.com`):

  ```
  mqsichangeproperties MB7BROKER -c SAPConnection -o mySAPAdapter.outadapter
  -n applicationServerHost -v production.sap.ibm.com
  ```

  To pick up the updated values in the configurable service, restart the execution group and message flow.

- To display all SAPConnection configurable services, use the WebSphere Message Broker Explorer, or the **mqsireportproperties** command, as shown in the following example:

  ```
  mqsireportproperties MB7BROKER -c SAPConnection -o AllReportableEntityNames -r
  ```

- You can delete a configurable service that you have created by using the WebSphere Message Broker Explorer, or the **mqsideleteconfigurableservice** command, as shown in the following example:

  ```
  mqsideleteconfigurableservice MB7BROKER -c SAPConnection
  -o mySAPAdapter.outadapter
  ```

  You can delete only custom-named configurable services; you cannot delete IBM-defined configurable services.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Configurable services" on page 1296
Configurable services are typically runtime properties. You can use them to define properties that are related to external services on which the broker relies; for example, an SMTP server or a JMS provider.

"Using the WebSphere Message Broker Explorer to work with configurable services" on page 644
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view, add, modify and delete configurable services.

Related reference:

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

Changing connection details for Siebel adapters:

Siebel nodes can get Siebel connection details from either the adapter component or a configurable service. By using configurable services, you can change the connection details for adapters without the need to redeploy the adapters. To pick

up new values when a configurable service is created or modified, you must reload the broker or execution group to which the adapter was deployed, by using the **mqsistop** and **mqsistart** commands, or the **mqsireload** command.

Before you begin

Before you start:

- Read "WebSphere Adapters nodes" on page 1914 and "Configurable services" on page 1296 for background information.

About this task

Use the SiebelConnection configurable service to change connection details for a Siebel adapter. The Siebel node reads all connection properties from the adapter component that it is configured to use. If a configurable service exists that has the same name as the adapter component of the node, the node uses the values that are defined in that configurable service to override the corresponding properties from the adapter. If a configurable service is being used, all properties that are exposed by the configurable service are taken from the configurable service. The only properties that are taken from the adapter are those that you cannot set on the configurable service. The properties of the Siebel configurable service are described in "Configurable services properties" on page 3766.

You can also connect to different versions of Siebel by creating a custom EISProviders configurable service and setting the location of the appropriate library files. For more information, see "Connecting to different versions of Siebel" on page 2077.

Creating, changing, reporting, and deleting configurable services

Procedure

- To create a configurable service, use the WebSphere Message Broker Explorer; see "Using the WebSphere Message Broker Explorer to work with configurable services" on page 644 for more information, or the **mqsicreateconfigurableservice** command, as shown in the following example. This example creates a SiebelConnection configurable service for the Siebel instance that is running on *my.siebel.qa.com*:

```
mqsicreateconfigurableservice MB7BROKER -c SiebelConnection -o mySiebelAdapter.outadapter
-n connectString -v "siebel://my.siebel.qa.com/SBA_80/SSEObjMgr_enu"
```

  To pick up the new values in the configurable service, restart the execution group and message flow.

- To change a configurable service, use the WebSphere Message Broker Explorer, or the **mqsichangeproperties** command, as shown in the following example. This example changes the connections that are used by the adapter *mySiebelAdapter.outadapter*. After you run this command, all adapters connect to the production system (`my.siebel.production.com`) instead of the test system (`my.siebel.qa.com`):

```
mqsichangeproperties MB7BROKER -c SiebelConnection -o mySiebelAdapter.outadapter -n connectString
-v "siebel://my.siebel.production.com/SBA_80/SSEObjMgr_enu"
```

  To pick up the updated values in the configurable service, restart the execution group and message flow.

- To display all SiebelConnection configurable services, use the WebSphere Message Broker Explorer, or the **mqsireportproperties** command, as shown in the following example:

```
mqsireportproperties MB7BROKER -c SiebelConnection -o AllReportableEntityNames -r
```

- You can delete a configurable service that you have created by using the WebSphere Message Broker Explorer, or the **mqsideleteconfigurableservice** command, as shown in the following example:

```
mqsideleteconfigurableservice MB7BROKER -c SiebelConnection -o mySiebelAdapter.outadapter
```

You can delete only custom-named configurable services; you cannot delete IBM-defined configurable services.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Configurable services" on page 1296
Configurable services are typically runtime properties. You can use them to define properties that are related to external services on which the broker relies; for example, an SMTP server or a JMS provider.

"Using the WebSphere Message Broker Explorer to work with configurable services" on page 644
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view, add, modify and delete configurable services.

Related reference:

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"SiebelInput node" on page 4740
Use the SiebelInput node to interact with a Siebel application.

"SiebelRequest node" on page 4745
Use the SiebelRequest node to interact with a Siebel application.

Changing connection details for PeopleSoft adapters:

PeopleSoft nodes can get PeopleSoft connection details from either the adapter component or a configurable service. By using configurable services, you can change the connection details for adapters without the need to redeploy the

adapters. To pick up new values when a configurable service is created or modified, you must reload the broker or execution group to which the adapter was deployed, by using the **mqsistop** and **mqsistart** commands, or the **mqsireload** command.

Before you begin

Before you start:

- Read "WebSphere Adapters nodes" on page 1914 and "Configurable services" on page 1296 for background information.

About this task

Use the PeopleSoftConnection configurable service to change connection details for a PeopleSoft adapter. The PeopleSoft node reads all connection properties from the adapter component that it is configured to use. If a configurable service exists that has the same name as the node's adapter component, the node uses the values that are defined in that configurable service to override the corresponding properties from the adapter. If a configurable service is being used, all properties that are exposed by the configurable service are taken from the configurable service. The only properties that are taken from the adapter are those that you cannot set on the configurable service. The properties of the PeopleSoft configurable service are described in "Configurable services properties" on page 3766.

Creating, changing, reporting, and deleting configurable services

Procedure

- To create a configurable service, use the WebSphere Message Broker Explorer; see "Using the WebSphere Message Broker Explorer to work with configurable services" on page 644 for more information, or the **mqsicreateconfigurableservice** command, as shown in the following example. This example creates a PeopleSoftConnection configurable service for the PeopleSoft instance that is running on *my.peoplesoft.qa.com*:

```
mqsicreateconfigurableservice MB7BROKER -c PeopleSoftConnection -o myPeopleSoftAdapter.outadapter
-n hostName,port -v "my.peoplesoft.qa.com",9000
```

  To pick up the new values in the configurable service, restart the execution group and message flow.

- To change a configurable service, use the WebSphere Message Broker Explorer, or the **mqsichangeproperties** command, as shown in the following example. This example changes the connections that are used by the adapter *myPeopleSoftAdapter.outadapter*. After you run this command, all adapters connect to the production system (`my.peoplesoft.production.com`) instead of the test system (`my.peoplesoft.qa.com`):

```
mqsichangeproperties MB7BROKER -c PeopleSoftConnection -o myPeopleSoftAdapter.outadapter -n hostName
-v "my.peoplesoft.production.com"
```

  To pick up the updated values in the configurable service, restart the execution group and message flow.

- To display all PeopleSoftConnection configurable services, use the WebSphere Message Broker Explorer, or the **mqsireportproperties** command, as shown in the following example:

```
mqsireportproperties MB7BROKER -c PeopleSoftConnection -o AllReportableEntityNames -r
```

- You can delete a configurable service that you have created by using the WebSphere Message Broker Explorer, or the **mqsideleteconfigurableservice** command, as shown in the following example:

```
mqsideleteconfigurableservice MB7BROKER -c PeopleSoftConnection -o myPeopleSoftAdapter.outadapter
```

  You can delete only custom-named configurable services; you cannot delete IBM-defined configurable services.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Configurable services" on page 1296
Configurable services are typically runtime properties. You can use them to define properties that are related to external services on which the broker relies; for example, an SMTP server or a JMS provider.

"Using the WebSphere Message Broker Explorer to work with configurable services" on page 644
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view, add, modify and delete configurable services.

Related reference:

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"PeopleSoftInput node" on page 4630
Use the PeopleSoftInput node to interact with a PeopleSoft application.

"PeopleSoftRequest node" on page 4635
Use the PeopleSoftRequest node to interact with a PeopleSoft application.

Changing connection details for JD Edwards adapters:

JD Edwards nodes can get JD Edwards EnterpriseOne connection details from either the adapter component or a configurable service. By using configurable services, you can change the connection details for adapters without the need to redeploy the adapters. To pick up new values when a configurable service is created or modified, you must reload the broker or execution group to which the adapter was deployed, by using the **mqsistop** and **mqsistart** commands, or the **mqsireload** command.

Before you begin

Before you start:

- Read "WebSphere Adapters nodes" on page 1914 and "Configurable services" on page 1296 for background information.

About this task

Use the JDEdwardsConnection configurable service to change connection details for a JD Edwards adapter. The JD Edwards node reads all connection properties from the adapter component that it is configured to use. If a configurable service exists that has the same name as the nodes adapter component, the node uses the values that are defined in that configurable service to override the corresponding properties from the adapter. If a configurable service is being used, all properties that are exposed by the configurable service are taken from the configurable service. The only properties that are taken from the adapter are the ones that you cannot set on the configurable service. The properties of the JD Edwards configurable service are described in "Configurable services properties" on page 3766.

Creating, changing, reporting, and deleting configurable services

Procedure

- To create a configurable service, use the WebSphere Message Broker Explorer; see "Using the WebSphere Message Broker Explorer to work with configurable services" on page 644 for more information, or the **mqsicreateconfigurableservice** command, as shown in the following example. This example creates a JDEdwardsConnection configurable service to connect to the DV7333 development Environment for a user with the Role of administrator.

```
mqsicreateconfigurableservice MYBROKER -c JDEdwardsConnection -o myJdedwardsAdapter.outadapter
-n Environment,Role -v "dv7333,administrator"
```

  To pick up the new values in the configurable service, restart the execution group and message flow.

- To change a configurable service, use the WebSphere Message Broker Explorer, or the **mqsichangeproperties** command, as shown in the following example. This example changes the connection that is used by the adapter *myJdedwardsAdapter.outadapter* to connect to the PD7333 production Environment. After you run this command, all adapters connect to the production Environment (PD7333) instead of the development system (DV7333).

```
mqsichangeproperties MYBROKER -c JDEdwardsConnection -o myJdedwardsAdapter.outadapter -n Environment,Role
-v PD7333,administrator
```

  To pick up the updated values in the configurable service, restart the execution group and message flow.

- To display all JDEdwardsConnection configurable services, use the WebSphere Message Broker Explorer, or the **mqsireportproperties** command, as shown in the following example:

```
mqsireportproperties MYBROKER -c JDEdwardsConnection -o AllReportableEntityNames -r
```

- You can delete a configurable service that has been created by using the WebSphere Message Broker Explorer, or the **mqsideleteconfigurableservice** command, as shown in the following example:

```
mqsideleteconfigurableservice MYBROKER -c JDEdwardsConnection -o myJdedwardsAdapter.outadapter
```

You can delete only custom-named configurable services; you cannot delete IBM-defined configurable services.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Configurable services" on page 1296
Configurable services are typically runtime properties. You can use them to define properties that are related to external services on which the broker relies; for example, an SMTP server or a JMS provider.

"Using the WebSphere Message Broker Explorer to work with configurable services" on page 644
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view, add, modify and delete configurable services.

Related reference:

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"JDEdwardsRequest node" on page 4524
Use the JDEdwardsRequest node to interact with a JD Edwards EnterpriseOne server.

Configuring EIS connections to expire after a specified time:

You can configure connections to SAP, Siebel, and PeopleSoft to expire after a specified time by using a configurable service.

About this task

You can use the connectionIdleTimeout property on a configurable service to control the number of connections to SAP, Siebel and PeopleSoft by closing connections that have not been used for a specified time.

Procedure

Use the **mqsicreateconfigurableservice** command to set up connections that expire when they have not been used for a specified length of time.

In the following example, the SAPConnection configurable service is configured to close connections when they have not been used for 120 seconds.

```
mqsicreateconfigurableservice MB7BROKER -c SAPConnection -o mySAPAdapter.outadapter
-n connectionIdleTimeout -v 120
```

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

Related tasks:

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

Related reference:

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

Advanced configuration properties when using IBM Sterling Connect:Direct nodes:

CDInput and CDOutput nodes can get connection details and staging directories in conjunction with a configurable service. To pick up new values when a configurable service is created or modified, you must reload the broker or execution group, by using the **mqsistop** and **mqsistart** commands, or the **mqsireload** command.

Before you begin

Before you start:

- Read "IBM Sterling Connect:Direct overview and concepts" on page 1810 and "Configurable services" on page 1296 for background information.

About this task

Use the *CDServer* configurable service to change connection details for a IBM Sterling Connect:Direct node. The properties of the *CDServer* configurable services are described in "*CDServer* configurable service properties" on page 3798.

Use the following scenarios as examples of how you use the staging directories.

Procedure

- **What should I do if I have WebSphere Message Broker and IBM Sterling Connect:Direct on the same machine, and want the files to be staged for output within the WebSphere Message Broker work path?**

  You need not set any of the properties; the default settings allow this to happen.
- **Output**
  1. **What should I do if I have WebSphere Message Broker and IBM Sterling Connect:Direct on the same machine but want to use a specified location to create files for output?** Set the `brokerPathToStagingDir` to the location you want. You do not need to set the `cdPathToStagingDir`, as WebSphere Message Broker assumes this to be the same as the `brokerPathToStagingDir`.
  2. **Can I have WebSphere Message Broker and IBM Sterling Connect:Direct on separate machines?** You can, but you must ensure that both machines have:
     - Access to the same shared filesystem.
     - The same timezone setting.

     As IBM Sterling Connect:Direct does not support NFS file systems on z/OS, this configuration does not work on z/OS
  3. **Can I have this shared filesystem mounted at different places on my WebSphere Message Broker and IBM Sterling Connect:Direct machines for platforms other than Windows?** Set the `brokerPathToStagingDir` to be the location where it is mounted on the WebSphere Message Broker machine, and the `cdPathToStagingDir` to where it is mounted on the IBM Sterling Connect:Direct machine.
  4. **Can I have this shared filesystem mounted at different places on my WebSphere Message Broker and IBM Sterling Connect:Direct machines for Windows platforms?** You must use the `\\hostname\directory` path syntax to the shared drive, rather than a mapped drive letter. Moreover, the user ID accessing the `\\hostname\directory` path, that is the:

     WebSphere Message Broker user ID, or

     IBM Sterling Connect:Direct user ID

     - depending on which one is running on Windows, must have full (write) access to the file system, using the same password.

     Set the `brokerPathToStagingDir` to be the location where it is mounted on the WebSphere Message Broker machine, and the `cdPathToStagingDir` to where it is mounted on the IBM Sterling Connect:Direct machine.
- **Input**
  1. **Can I process files in the CDInput node if the WebSphere Message Broker and receiving Connect:Direct server are on different machines?** You can, but you must ensure that both machines have:
     - Access to the same shared filesystem.
     - The same timezone setting.

As IBM Sterling Connect:Direct does not support NFS file systems on z/OS, this configuration does not work on z/OS

2. **Can I have this shared filesystem mounted at different places on my WebSphere Message Broker and IBM Sterling Connect:Direct machines for platforms other than Windows?** Set the `brokerPathToInputDir` to be the location where it is mounted on the WebSphere Message Broker machine, and the `cdPathToInputDir` to where it is mounted on the IBM Sterling Connect:Direct machine.
3. **Can I have this shared filesystem mounted at different places on my WebSphere Message Broker and IBM Sterling Connect:Direct machines for Windows platforms?** You must use the UNC path syntax to the shared drive, rather than a mapped drive letter. Moreover, the user ID accessing the UNC path, that is the:

   WebSphere Message Broker user ID, or

   IBM Sterling Connect:Direct user ID

   - depending on which one is running on Windows, must have full (write) access to the file system, using the same password.

   Set the `brokerPathToInputDir` to be the location where it is mounted on the WebSphere Message Broker machine, and the `cdPathToInputDir` to where it is mounted on the IBM Sterling Connect:Direct machine.

Results

To create a configurable service, use the WebSphere Message Broker Explorer; see "Using the WebSphere Message Broker Explorer to work with configurable services" on page 644 for more information, or the **mqsicreateconfigurableservice** command.

```
mqsicreateconfigurableservice MYBROKER -c CDServer -o myCDServer
```

where

MYBROKER
is the name of the broker

CDServer
is the name of the configurable service

myCDServer
is the object name.

Note the -o *ObjectName* parameter can take any value.

Output

This example transfers a file using the CDOutput node called *CDOutput*, in a message flow called *messageflow1*, in execution group *default*, on broker *MYBROKER*.

The broker stages the file in the local file path `/tmp/cdtransfer/MYBROKER/default/messageflow1/CDOutput`.

The Connect:Direct server then tries to transfer the file using the file path `/cdserver/transfers/MYBROKER/default/messageflow1/CDOutput`.

These properties are used when the files systems are mounted differently on the broker and Connect:Direct server machines.

```
mqsichangeproperties MYBROKER -c CDServer
-o myCDServer -n brokerPathToStagingDir,cdPathToStagingDir
-v /tmp/cdtransfer,/cdserver/transfers
```

To pick up the updated values in the configurable service, restart the execution group and message flow.

Input

This example receives a file using the CDInput node called *CDInput*, in a message flow called *messageflow1*, in execution group *default*, on broker *MYBROKER*.

A file is transferred to the Connect:Direct server, to the directory `/cdserver/transfers/example1`.

WebSphere Message Broker attempts to process the same file, but using the path `/tmp/cdtransfer/example1`.

These properties are used when the file systems are mounted differently on the WebSphere Message Broker and Connect:Direct server machines.

```
mqsichangeproperties MYBROKER -c CDServer -o myCDServer
-n brokerPathToInputDir,cdPathToInputDir
-v /tmp/cdtransfer/example1,/cdserver/transfers/example1
```

To pick up the updated values in the configurable service, restart the execution group and message flow.

To display Connect:Direct server configurable services, use the **mqsireportproperties** command, as shown in the following example:

```
mqsireportproperties MYBROKER -c CDServer -o AllReportableEntityNames -r
```

Related concepts:

"Configurable services" on page 1296
Configurable services are typically runtime properties. You can use them to define properties that are related to external services on which the broker relies; for example, an SMTP server or a JMS provider.

"Using the WebSphere Message Broker Explorer to work with configurable services" on page 644
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view, add, modify and delete configurable services.

Related tasks:

"Initiating a managed file transfer using IBM Sterling Connect:Direct" on page 1873
Use a CDOutput node to send a file from a specified directory on your primary Connect:Direct server (PNODE) to a filename and directory on a secondary Connect:Direct server (SNODE).

"Receiving a file using IBM Sterling Connect:Direct" on page 1847
Use the CDInput node to receive files from an IBM Sterling Connect:Direct network.

Related reference:

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"*CDServer* configurable service properties" on page 3798
Select the objects and properties that you want to change for the *CDServer* configurable service.

Configuring for IMS:

You can use configurable services to enable the IMS nodes in the broker runtime environment to connect with the IMS system.

About this task

The following topics describe how to prepare the environment to connect to the IMS system, and how to change the connection details without the need to redeploy your message flow.

- "Preparing the environment for IMS nodes"
- "Changing connection information for the IMSRequest node" on page 732

Preparing the environment for IMS nodes:

Before you can use the IMS nodes, you must set up the broker runtime environment so that you can access the IMS system.

Before you begin

Before you start:

Read "IBM Information Management System (IMS)" on page 2129.

About this task

Complete the following steps to ensure that WebSphere Message Broker can connect to the IMS system.

Procedure

1. Ensure that IMS Connect is installed and started on the IMS system.
2. If you do not want to configure IMS connection properties directly on the IMSRequest node, define a configurable service for each IMS system to which you want to connect.

   For example, to create an IMSConnect configurable service for the IMS instance *IMSA* that is running on *test.ims.ibm.com*, port *9999*, run the **mqsicreateconfigurableservice** command as shown:

   ```
   mqsicreateconfigurableservice MB7BROKER -c IMSConnect -o myIMSConnectService
   -n Hostname,PortNumber,DataStoreName -v test.ims.ibm.com,9999,IMSA
   ```

For details about how to create, change, and report configurable services, see "Changing connection information for the IMSRequest node." You can use the IMSConnect configurable service to configure the IMSRequest node to use Secure Sockets Layer (SSL) protocol. For more information, see "Securing the connection to IMS by using SSL" on page 549.

3. Use the **mqsisetdbparms** command to set security details in the broker store.

   For example, to associate a user ID and password pair with an IMS Connect connection, run the **mqsisetdbparms** command as shown:

   ```
   mqsisetdbparms MB7BROKER -n ims::mySecurityIdentity -u myuserid -p mypassword
   ```

Related concepts:

"IBM Information Management System (IMS)" on page 2129
IMS is a message-based transaction manager and hierarchical-database manager for z/OS. External applications can use transactions to interact with applications that run inside IMS.

Related tasks:

"Changing connection information for the IMSRequest node"
You can create a configurable service that the IMSRequest node or message flow refers to at run time for connection information, instead of defining the connection properties on the node or the message flow. The advantage being that you can change the host name, performance, and security values without needing to redeploy your message flow.

"Securing the connection to IMS by using SSL" on page 549
Configure the IMSRequest node to communicate with IMS over the Secure Sockets Layer (SSL) protocol by creating a keystore file, and configuring the broker to use SSL.

Related reference:

"IMSRequest node" on page 4504
Use the IMSRequest node to send a request to run a transaction on a local or remote IMS system, and wait for a response. IMS Connect must be configured and running on the IMS system.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

Changing connection information for the IMSRequest node:

You can create a configurable service that the IMSRequest node or message flow refers to at run time for connection information, instead of defining the connection

properties on the node or the message flow. The advantage being that you can change the host name, performance, and security values without needing to redeploy your message flow.

Before you begin

Before you start:

- Read "Configurable services" on page 1296 to find out more about configurable services.
- Read "IBM Information Management System (IMS)" on page 2129 for background information.

About this task

Use the IMSConnect configurable service to change the connection information for the IMSRequest node. Two configurable services can connect to the same instance of IMS Connect. The properties of the IMSConnect configurable service are described in "Configurable services properties" on page 3766.

You can use the IMSConnect configurable service to configure the IMSRequest node to use Secure Sockets Layer (SSL) protocol. For more information, see "Securing the connection to IMS by using SSL" on page 549.

Creating, changing, reporting, and deleting configurable services

Procedure

- To create a configurable service, use the WebSphere Message Broker Explorer; see "Using the WebSphere Message Broker Explorer to work with configurable services" on page 644 for more information, or the **mqsicreateconfigurableservice** command, as shown in the following example. This example creates an IMSConnect configurable service for the IMS instance IMSA that is running on `test.ims.ibm.com` port 9999:

  ```
  mqsicreateconfigurableservice MB7BROKER -c IMSConnect -o myIMSConnectService
  -n Hostname,PortNumber,DataStoreName -v test.ims.ibm.com,9999,IMSA
  ```

- To change a configurable service, use the WebSphere Message Broker Explorer, or the **mqsichangeproperties** command, as shown in the following example. This example changes all the nodes that are configured to use the *myIMSConnectService* configurable service. After you run this command, the IMSRequest node connects to the production system (`production.ims.ibm.com`) instead of the test system (`test.ims.ibm.com`). The command also changes to coded character set identifier (CCSID) to 37.

  ```
  mqsichangeproperties MB7BROKER -c IMSConnect -o myIMSConnectService -n Hostname,CodedCharSetID
  -v production.ims.ibm.com,37
  ```

  See "Securing the connection to IMS by using SSL" on page 549 for information about how to turn on SSL support in the broker by setting the `UseSSL` and `SSLEncryptionType` IMSConnect configurable service properties.

- To display all IMSConnect configurable services, use the WebSphere Message Broker Explorer, or the **mqsireportproperties** command, as shown in the following example:

  ```
  mqsireportproperties MB7BROKER -c IMSConnect -o AllReportableEntityNames -r
  ```

- You can delete a configurable service that you have created by using the WebSphere Message Broker Explorer, or the **mqsideleteconfigurableservice** command, as shown in the following example:

```
mqsideleteconfigurableservice MB7BROKER -c IMSConnect -o myIMSconnectService
```

You can delete only custom-named configurable services; you cannot delete IBM-defined configurable services.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Configurable services" on page 1296
Configurable services are typically runtime properties. You can use them to define properties that are related to external services on which the broker relies; for example, an SMTP server or a JMS provider.

"Using the WebSphere Message Broker Explorer to work with configurable services" on page 644
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view, add, modify and delete configurable services.

Related tasks:

"Securing the connection to IMS by using SSL" on page 549
Configure the IMSRequest node to communicate with IMS over the Secure Sockets Layer (SSL) protocol by creating a keystore file, and configuring the broker to use SSL.

"Resolving problems when you use IMS nodes" on page 3419
Advice for dealing with common problems that can arise when you develop message flows that contain IMS nodes.

Related reference:

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"IMSRequest node" on page 4504
Use the IMSRequest node to send a request to run a transaction on a local or remote IMS system, and wait for a response. IMS Connect must be configured and running on the IMS system.

Defining where the CORBARequest node gets the object reference:

You can specify an object reference name either on the CORBARequest node or by using a configurable service.

Before you begin

Before you start:

- Read “Common Object Request Broker Architecture (CORBA)” on page 2145 and “Configurable services” on page 1296 for background information.

About this task

By using configurable services, you can specify the location from which the CORBARequest node gets the object reference without the need to redeploy the message flow. You can also use the configurable service to specify this location for multiple CORBARequest nodes. To pick up new values when a configurable service is created or modified, you must reload the broker or execution group to which the message flow was deployed, by using the **mqsistop** and **mqsistart** commands, or the **mqsireload** command.

The properties of the CORBA configurable services are described in “Configurable services properties” on page 3766.

Creating, changing, reporting, and deleting configurable services

Procedure

- To create a configurable service, use the WebSphere Message Broker Explorer (as described in “Using the WebSphere Message Broker Explorer to work with configurable services” on page 644) or the **mqsicreateconfigurableservice** command shown in the following example.

  This example creates a CORBA configurable service called "myCORBAService" that connects to the CORBA host on the local host on port 2809, and looks for the object reference called "Europe.region/Market.object". (For more information about how to specify the object reference, see “CORBA naming service” on page 2154.)

  ```
  mqsicreateconfigurableservice MB7BROKER -c CORBA -o myCORBAService
  -n namingService,objectReferenceName -v localhost:2809,Europe.region/Market.object
  ```

  To pick up the new values in the configurable service, restart the execution group and message flow.
- To change a configurable service, use the WebSphere Message Broker Explorer or the **mqsichangeproperties** command shown in the following example.

  This example changes the location of the object reference.

  ```
  mqsichangeproperties MB7BROKER -c CORBA -o myCORBAService -n namingService,objectReferenceName
  ```

  To pick up the updated values in the configurable service, restart the execution group and message flow.
- To display all CORBA configurable services, use the WebSphere Message Broker Explorer or the **mqsireportproperties** command shown in the following example.

  ```
  mqsireportproperties MB7BROKER -c CORBA -o AllReportableEntityNames -r
  ```
- You can delete a configurable service that you have created by using the WebSphere Message Broker Explorer or the **mqsideleteconfigurableservice** command shown in the following example.

  ```
  mqsideleteconfigurableservice MB7BROKER -c CORBA -o myCORBAService
  ```

  You can delete only custom-named configurable services; you cannot delete IBM-defined configurable services.

Related concepts:

"Configurable services" on page 1296
Configurable services are typically runtime properties. You can use them to define properties that are related to external services on which the broker relies; for example, an SMTP server or a JMS provider.

"Using the WebSphere Message Broker Explorer to work with configurable services" on page 644
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view, add, modify and delete configurable services.

Related tasks:

"Working with CORBA" on page 2144
Use CORBA nodes to connect to CORBA Internet Inter-Orb Protocol (IIOP) applications.

Related reference:

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"CORBARequest node" on page 4349
Use the CORBARequest node to call an external CORBA application over Internet Inter-Orb Protocol (IIOP).

Configuring for CICS Transaction Server for z/OS:

You can use configurable services to enable the CICSRequest node in the broker runtime environment to connect with CICS Transaction Server for z/OS.

About this task

The following topics describe how to prepare the environment to connect to CICS, and how to change the connection details without the need to redeploy your message flow.

- "Preparing the environment for the CICSRequest node"
- "Changing connection information for the CICSRequest node" on page 738

Preparing the environment for the CICSRequest node:

Before you can use the CICSRequest node, you must configure IP InterCommunications (IPIC) protocol on the target CICS Transaction Server for z/OS.

Before you begin

Before you start:

Read "CICS Transaction Server for z/OS connectivity" on page 2174 for background information.

About this task

The CICSRequest node can send IPIC requests over TCP/IP to CICS Transaction Server for z/OS Version 3.2 and later. Complete the following steps on CICS to perform this configuration:

Procedure

1. Set the System Initialization (SIT) parameter **TCPIP=YES**.
2. Define the TCP/IP address and host name for CICS. By default, they are defined in the PROFILE.TCPIP and TCPIP.DATA data sets.
3. Add a TCP/IP listener to CICS by using the following **CEDA** command to define a TCPIPSERVICE resource in a group:

   ```
   CEDA DEF TCPIPSERVICE(service-name) GROUP(group-name)
   ```

   Ensure that the group in which you define the service is in the GRPLIST system initialization parameter, so that the listener starts when CICS is started. Key fields are explained as follows:

   POrtnumber:
   The port on which the TCP/IP service listens.

   PROtocol:
   The protocol of the service is IPIC.

   TRansaction:
   The transaction that CICS runs to handle incoming IPIC requests. Set the field to CISS, which is the default.

   Backlog:
   The Backlog field is the number of TCP/IP connection requests are sent to CICS, which are placed in a TCP/IP queue to be assigned an IPCONN connection to CICS. The default value is 1. Do not use a value of 0; a value of 0 indicates that no TCP/IP connection requests are to be assigned an IPCONN connection to CICS, which disables incoming connection requests.

   Ipaddress:
   The IP address, in dotted decimal form, on which the TCPIPSERVICE resource listens. For configurations with more than one IP stack, specify ANY to make the TCPIPSERVICE resource listen on all addresses.

   SOcketclose:
   Whether CICS waits before closing the socket after issuing a receive for incoming data on that socket. To ensure that the connection from the CICSRequest node always remains open, set SOcketclose to NO for IPIC connections.

   SSl: Whether the CICS TCP/IP service is to use Secure Sockets Layer (SSL) protocol for encryption and authentication. Valid values are NO, YES, or CLIENTAUTH. Where:

   - NO indicates that SSL is not to be used.

- YES indicates that the personal certificate of the CICS region must be trusted by WebSphere Message Broker.
- CLIENTAUTH indicates that the personal certificate of the CICS region must be trusted by WebSphere Message Broker, and the WebSphere Message Broker personal certificate must be trusted by CICS.

  The CICSRequest node does not support a separate truststore, so the keystore file must provide both personal and signer certificates. For more information, see "Securing the connection to CICS Transaction Server for z/OS by using SSL" on page 547.

4. Use the following command to install the TCPIPSERVICE definition:

   ```
   CEDA INS TCPIPSERVICE(service-name) GROUP(group-name)
   ```

Related concepts:

"CICS Transaction Server for z/OS overview" on page 2173
CICS Transaction Server for z/OS provides general-purpose transaction processing software for z/OS. CICS is a powerful application server that meets the transaction-processing needs of both large and small enterprises.

Related tasks:

"Changing connection information for the CICSRequest node"
You can create a configurable service that the CICSRequest node or message flow refers to at run time for connection information, instead of defining the connection properties on the node or the message flow. The advantage being that you can change the host name and performance values without needing to redeploy your message flow.

Related reference:

"CICSRequest node" on page 4321
Use the CICSRequest node to call CICS Transaction Server for z/OS programs over TCP/IP-based IP InterCommunications (IPIC) protocol.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

Changing connection information for the CICSRequest node:

You can create a configurable service that the CICSRequest node or message flow refers to at run time for connection information, instead of defining the connection properties on the node or the message flow. The advantage being that you can change the host name and performance values without needing to redeploy your message flow.

Before you begin

Before you start:

- Read "Configurable services" on page 1296 to find out more about configurable services.
- Read "CICS Transaction Server for z/OS overview" on page 2173 for background information.

About this task

Use the CICSConnection configurable service to change the CICS Transaction Server for z/OS connection information for the CICSRequest node. The advantage being that you can change the host name, performance, and security identity values without needing to redeploy your message flow. The properties of the CICSConnection configurable service are described in "Configurable services properties" on page 3766.

You can use the CICSConnection configurable service to configure the CICSRequest node to use Secure Sockets Layer (SSL) protocol. For more information, see "Securing the connection to CICS Transaction Server for z/OS by using SSL" on page 547.

Creating, changing, reporting, and deleting configurable services

Procedure

- To create a configurable service, use the WebSphere Message Broker Explorer; see "Using the WebSphere Message Broker Explorer to work with configurable services" on page 644 for more information, or the **mqsicreateconfigurableservice** command, as shown in the following example. This example creates a CICSConnection configurable service for the CICS instance that is running at `test.cics.ibm.com` port 12345. The broker is identified by APPLID *BRKApp* and qualifier *BRKQual*. The connection timeout is 10 seconds, the request timeout is 5 seconds, and the security identity is identified by *mySecurityIdentity* in this example:

  ```
  mqsicreateconfigurableservice MB7BROKER -c CICSConnection -o myCICSConnectionService
  -n cicsServer,clientApplid,clientQualifier,connectionTimeoutSecs,requestTimeoutSecs,
  securityIdentity
  -v tcp://test.cics.ibm.com:12345,BRKApp,BRKQual,10,5,mySecurityIdentity
  ```

- To change a configurable service, use the WebSphere Message Broker Explorer, or the **mqsichangeproperties** command, as shown in the following example. You must stop and start the execution group for the change of property value to take effect. This example changes the CICSRequest node that is configured to use the *myCICSConnectionService* configurable service. After you run this command, the CICSRequest node connects to the production system (`tcp://production.cics.ibm.com:12345`) instead of the test system (`tcp://test.cics.ibm.com:12345`).

  ```
  mqsichangeproperties MB7BROKER -c CICSConnection -o myCICSConnectionService
    -n cicsServer -v tcp://production.cics.ibm.com:12345
  ```

  See "Securing the connection to CICS Transaction Server for z/OS by using SSL" on page 547 for information about how to turn on SSL support in the broker by setting the `cicsServer` CICSConnection configurable service property.
- To display all CICSConnection configurable services, use the WebSphere Message Broker Explorer, or the **mqsireportproperties** command, as shown in the following example:

```
mqsireportproperties MB7BROKER -c CICSConnection -o AllReportableEntityNames -r
```

- You can delete a configurable service that you have created by using the WebSphere Message Broker Explorer, or the **mqsideleteconfigurableservice** command, as shown in the following example:

  ```
  mqsideleteconfigurableservice MB7BROKER -c CICSConnection -o myCICSConnectionService
  ```

  You can delete only custom-named configurable services; you cannot delete IBM-defined configurable services.

Related concepts:

"CICS Transaction Server for z/OS connectivity" on page 2174
Use the CICSRequest node to connect WebSphere Message Broker with CICS Transaction Server for z/OS applications.

"Configurable services" on page 1296
Configurable services are typically runtime properties. You can use them to define properties that are related to external services on which the broker relies; for example, an SMTP server or a JMS provider.

"Using the WebSphere Message Broker Explorer to work with configurable services" on page 644
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view, add, modify and delete configurable services.

Related reference:

"CICSRequest node" on page 4321
Use the CICSRequest node to call CICS Transaction Server for z/OS programs over TCP/IP-based IP InterCommunications (IPIC) protocol.

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

Preparing the environment for WebSphere MQ File Transfer Edition nodes:

Prepare the file system and queue managers, and determine the name of the broker agent.

About this task

- "Preparing the file system" on page 741
- "Preparing the queue manager" on page 741
- "Setting the coordination queue manager" on page 741
- "Naming execution groups" on page 742

- "Determining the agent name" on page 743

Preparing the file system:

WebSphere Message Broker uses a location in its work path to store transfers to remote agents. It uses another location as the default directory for received files. The high-level directory path for both locations is:

- *workpath*/common/FTE

Ensure that enough space is available here for files that you will transfer to and from the broker by WebSphere MQ File Transfer Edition.

Preparing the queue manager:

WebSphere Message Broker tries to create all the required artifacts on the queue manager for the agent, and the coordination queue manager, if appropriate. It might not be possible to create all artifacts, due to the configuration of your machine, or permissions. If it fails to create them, or you want to create them yourself in advance, see "Scripts to create artifacts required for WebSphere MQ File Transfer Edition" on page 744.

Setting the coordination queue manager:

When a message flow that contains a WebSphere MQ File Transfer Edition node is deployed to an execution group, an agent is automatically created and started in that execution group. By default, the agent uses the broker's local queue manager as the coordination queue manager.

- If the broker's queue manager is being used as the coordination queue manager, the broker configures it as a coordination queue manager.
- If you are using a different queue manager as the coordination queue manager, refer to the WebSphere File Transfer Edition Information Center for details of how to configure it as a coordination queue manager.

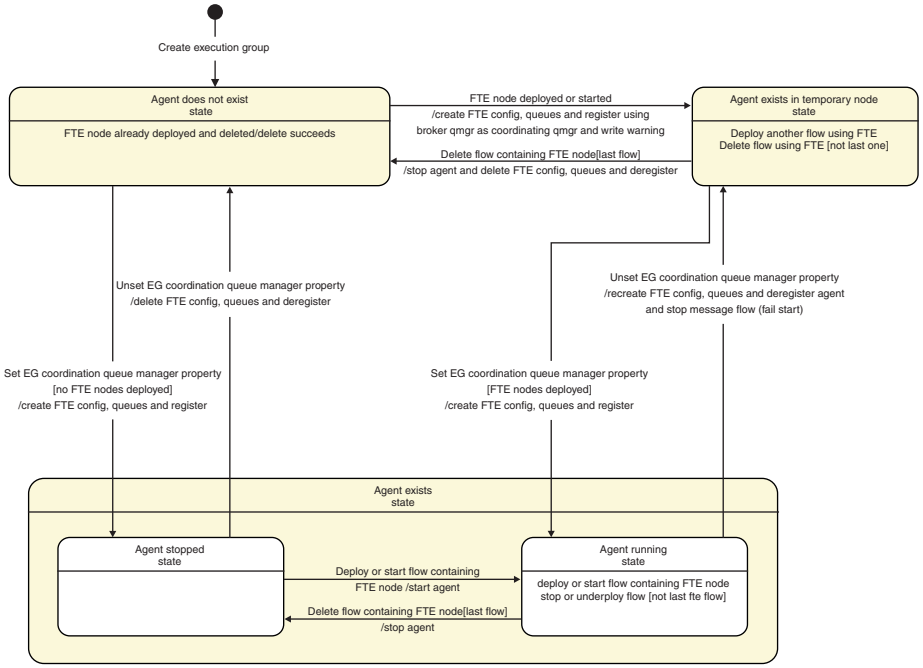
Unless you have previously defined the coordination queue manager, the agent is temporary; it is deleted when the flow is undeployed or the broker is stopped. This behavior is acceptable in a test environment. However, for production, the administrator must specify the coordination queue manager for the execution group. Specifying a coordination queue manager:

Ensures that the correct queue manager is used when the agent is created.

Makes the agent permanent. If a coordination queue manager has been defined, the agent is deleted only after you undefine the coordination queue manager (for example, by setting it to an empty string), and restart the execution group.

A warning is written to the log if the coordination queue manager is not changed from the default.

The following state diagram illustrates how the presence of nodes and a defined coordination queue manager affect the state of the agent.



About this task

Use one of the following methods to set the coordination queue manager.

Procedure

1. Optional: Use the **mqsichangeproperties** command.

   For example, to set the coordination queue manager to *QM2* for execution group myExecutionGroup in broker MB7BROKER:

   ```
   mqsichangeproperties MB7BROKER -e myExecutionGroup
   -o FTEAgent -n coordinationQMgr -v QM2
   ```

   After running this command, you must reload the execution group for the change to take effect.
2. Optional: Use the WebSphere Message Broker Explorer. Right-click the execution group, and select the appropriate options from the pop-up menu.

Naming execution groups:
About this task

The execution group name is used to form the queue name for WebSphere MQ File Transfer Edition queues. Consequently, the names of your execution groups must conform to the rules for naming WebSphere MQ objects. You cannot deploy a flow that contains a WebSphere MQ File Transfer Edition node unless this requirement is met. Permitted characters are:

- Uppercase A-Z
- Lowercase a-z (but there are restrictions on the use of lowercase letters for z/OS console support)

  On systems using EBCDIC Katakana you cannot use lowercase characters.
- Numerics 0-9
- Period (.)

- Forward slash (/)
- Underscore (_)
- Percent sign (%)

See the WebSphere MQ Version 7 Information Center online for full details of naming requirements.

Determining the agent name:
About this task

To send a file to a given execution group, users need to know the name of the agent that the broker creates. The agent name is derived from *Broker.ExecutionGroup*, and is not configurable. The total name length is limited to 28 characters, with a maximum 12 characters for the broker name, and 15 characters for the execution group. Broker and execution group names longer than these limits are truncated to form the agent name. The name must be a valid format for generating MQ Series queue name. Ensure that:

- The broker name is 12 characters or fewer (or at least unique in the first 12 characters).
- The execution group names are 15 characters or fewer (or at least unique in the first 15 characters).
- The broker and execution groups do not contain any characters that are invalid for queue names.
- The *broker.executiongroup* tuples are all unique, even if case is ignored.

The value used is written to the event log in message BIP3358. Use one of the following methods to determine the agent name.

Procedure

1. Optional: Use the **mqsireportproperties** command.

   For example, to display the FTE agent name for execution group FTESAMPLE in broker MB7BROKER:

   ```
   mqsireportproperties MB7BROKER -e FTESAMPLE -o FTEAgent -n agentName
   ```

   If the agent has been created, the command returns the agent name. If the agent has not been created, the command returns an empty string.
2. Optional: Use the WebSphere Message Broker Explorer. Right-click the execution group, and select the appropriate options from the pop-up menu.

Related concepts:

"Managed file transfers using WebSphere MQ File Transfer Edition" on page 1869
Transfer files, with file transfer metadata, in a timely and reliable manner.

Related tasks:

"Changing the location of the work path" on page 1011
The work path directory is the location where a component stores internal data, such as installation logs, component details, and trace output. The shared-classes directory is also located in the work path directory and is used for deployed Java code. If the work path directory does not have enough capacity, redirect the directory to another file system that has enough capacity.

Related reference:

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"FTEInput node" on page 4461
Use the FTEInput node to receive files using the WebSphere MQ File Transfer Edition.

"FTEOutput node" on page 4466
Use the FTEOutput node to write messages to files by using the WebSphere MQ File Transfer Edition.

Scripts to create artifacts required for WebSphere MQ File Transfer Edition:

WebSphere MQ script (MQSC) commands for use if WebSphere Message Broker cannot create all the required artifacts, or you want to create them yourself.

Queues for the FTE agent

Before running the following command, replace *Agent name* with the name of your FTE agent; see "Determining the agent name" on page 743.

```
DEFINE QLOCAL(SYSTEM.FTE.COMMAND.Agent name) +
 DEFPRTY(0) +
 DEFSOPT(SHARED) +
 GET(ENABLED) +
 MAXDEPTH(5000) +
 MAXMSGL(4194304) +
 MSGDLVSQ(PRIORITY) +
 PUT(ENABLED) +
 RETINTVL(999999999) +
 SHARE +
 NOTRIGGER +
 USAGE(NORMAL) +
 REPLACE
DEFINE QLOCAL(SYSTEM.FTE.DATA.Agent name) +
 DEFPRTY(0) +
 DEFSOPT(SHARED) +
 GET(ENABLED) +
 MAXDEPTH(5000) +
 MAXMSGL(4194304) +
 MSGDLVSQ(PRIORITY) +
 PUT(ENABLED) +
 RETINTVL(999999999) +
 SHARE +
 NOTRIGGER +
 USAGE(NORMAL) +
 REPLACE
DEFINE QLOCAL(SYSTEM.FTE.REPLY.Agent name) +
 DEFPRTY(0) +
 DEFSOPT(SHARED) +
 GET(ENABLED) +
 MAXDEPTH(5000) +
 MAXMSGL(4194304) +
 MSGDLVSQ(PRIORITY) +
 PUT(ENABLED) +
 RETINTVL(999999999) +
 SHARE +
```

```
 NOTRIGGER +
 USAGE(NORMAL) +
 REPLACE
DEFINE QLOCAL(SYSTEM.FTE.STATE.Agent name) +
 DEFPRTY(0) +
 DEFSOPT(SHARED) +
 GET(ENABLED) +
 MAXDEPTH(5000) +
 MAXMSGL(4194304) +
 MSGDLVSQ(PRIORITY) +
 PUT(ENABLED) +
 RETINTVL(999999999) +
 SHARE +
 NOTRIGGER +
 USAGE(NORMAL) +
 REPLACE
DEFINE QLOCAL(SYSTEM.FTE.EVENT.Agent name) +
 DEFPRTY(0) +
 DEFSOPT(SHARED) +
 GET(ENABLED) +
 MAXDEPTH(5000) +
 MAXMSGL(4194304) +
 MSGDLVSQ(PRIORITY) +
 PUT(ENABLED) +
 RETINTVL(999999999) +
 SHARE +
 NOTRIGGER +
 USAGE(NORMAL) +
 REPLACE
DEFINE QLOCAL(SYSTEM.FTE.AUTHAGT1.Agent name) +
 DEFPRTY(0) +
 DEFSOPT(SHARED) +
 GET(ENABLED) +
 MAXDEPTH(0) +
 MAXMSGL(0) +
 MSGDLVSQ(PRIORITY) +
 PUT(ENABLED) +
 RETINTVL(999999999) +
 SHARE +
 NOTRIGGER +
 USAGE(NORMAL) +
 REPLACE
DEFINE QLOCAL(SYSTEM.FTE.AUTHTRN1.Agent name) +
 DEFPRTY(0) +
 DEFSOPT(SHARED) +
 GET(ENABLED) +
 MAXDEPTH(0) +
 MAXMSGL(0) +
 MSGDLVSQ(PRIORITY) +
 PUT(ENABLED) +
 RETINTVL(999999999) +
 SHARE +
 NOTRIGGER +
 USAGE(NORMAL) +
 REPLACE
DEFINE QLOCAL(SYSTEM.FTE.AUTHOPS1.Agent name) +
```

```
 DEFPRTY(0) +
 DEFSOPT(SHARED) +
 GET(ENABLED) +
 MAXDEPTH(0) +
 MAXMSGL(0) +
 MSGDLVSQ(PRIORITY) +
 PUT(ENABLED) +
 RETINTVL(999999999) +
 SHARE +
 NOTRIGGER +
 USAGE(NORMAL) +
 REPLACE
DEFINE QLOCAL(SYSTEM.FTE.AUTHSCH1.Agent name) +
 DEFPRTY(0) +
 DEFSOPT(SHARED) +
 GET(ENABLED) +
 MAXDEPTH(0) +
 MAXMSGL(0) +
 MSGDLVSQ(PRIORITY) +
 PUT(ENABLED) +
 RETINTVL(999999999) +
 SHARE +
 NOTRIGGER +
 USAGE(NORMAL) +
 REPLACE
DEFINE QLOCAL(SYSTEM.FTE.AUTHMON1.Agent name) +
 DEFPRTY(0) +
 DEFSOPT(SHARED) +
 GET(ENABLED) +
 MAXDEPTH(0) +
 MAXMSGL(0) +
 MSGDLVSQ(PRIORITY) +
 PUT(ENABLED) +
 RETINTVL(999999999) +
 SHARE +
 NOTRIGGER +
 USAGE(NORMAL) +
 REPLACE
DEFINE QLOCAL(SYSTEM.FTE.AUTHADM1.Agent name) +
 DEFPRTY(0) +
 DEFSOPT(SHARED) +
 GET(ENABLED) +
 MAXDEPTH(0) +
 MAXMSGL(0) +
 MSGDLVSQ(PRIORITY) +
 PUT(ENABLED) +
 RETINTVL(999999999) +
 SHARE +
 NOTRIGGER +
 USAGE(NORMAL) +
 REPLACE
```

Queues for the coordination queue manager

```
DEFINE TOPIC('SYSTEM.FTE') TOPICSTR('SYSTEM.FTE') REPLACE
ALTER TOPIC('SYSTEM.FTE') NPMSGDLV(ALLAVAIL) PMSGDLV(ALLAVAIL)
```

```
DEFINE QLOCAL(SYSTEM.FTE) LIKE(SYSTEM.BROKER.DEFAULT.STREAM) REPLACE
ALTER QLOCAL(SYSTEM.FTE) DESCR('Stream for WMQFTE Pub/Sub interface')
* Altering namelist: SYSTEM.QPUBSUB.QUEUE.NAMELIST
* Value prior to alteration:
DISPLAY NAMELIST(SYSTEM.QPUBSUB.QUEUE.NAMELIST)
ALTER NAMELIST(SYSTEM.QPUBSUB.QUEUE.NAMELIST) +
 NAMES(SYSTEM.BROKER.DEFAULT.STREAM+
 ,SYSTEM.BROKER.ADMIN.STREAM,SYSTEM.FTE)
* Altering PSMODE.  Value prior to alteration:
DISPLAY QMGR PSMODE
ALTER QMGR PSMODE(ENABLED)
DEFINE QLOCAL(SYSTEM.FTE.DATABASELOGGER.REJECT) +
 DESCR('Messages rejected by the FTE database logger.') +
 DEFPRTY(0) +
 DEFSOPT(SHARED) +
 GET(ENABLED) +
 MAXDEPTH(999999999) +
 MAXMSGL(4194304) +
 MSGDLVSQ(PRIORITY) +
 PUT(ENABLED) +
 RETINTVL(999999999) +
 SHARE +
 NOTRIGGER +
 USAGE(NORMAL) +
 REPLACE
DEFINE QLOCAL(SYSTEM.FTE.DATABASELOGGER.COMMAND) +
 DESCR('Command messages to control the FTE database logger.') +
 DEFPRTY(0) +
 DEFSOPT(SHARED) +
 GET(ENABLED) +
 MAXDEPTH(999999999) +
 MAXMSGL(4194304) +
 MSGDLVSQ(PRIORITY) +
 PUT(ENABLED) +
 RETINTVL(5000) +
 SHARE +
 NOTRIGGER +
 USAGE(NORMAL) +
 REPLACE
```

Related tasks:

"Preparing the environment for WebSphere MQ File Transfer Edition nodes" on page 740
Prepare the file system and queue managers, and determine the name of the broker agent.

Configuring for JMS:

You can use configurable services to configure WebSphere Message Broker for JMS.

About this task

- "Configuring the broker to enable a JMS provider's proprietary API" on page 748
- "Connecting to different versions of the same JMS provider" on page 750
- "Configuring the JMSInput node for batch message processing" on page 751

Configuring the broker to enable a JMS provider's proprietary API:

Some JMS providers provide an alternative interface to the standard JMS specification for particular JMS API calls. In these cases, IBM supplies a Java class to interface with that proprietary API.

About this task

For example, BEA WebLogic uses a component called a *Client Interposed Transaction Manager* to allow a JMS client to obtain a reference to the XAResource that is associated with a user transaction.

If the WebSphere Message Broker JMS nodes use BEA WebLogic as the JMS provider, and the nodes must participate in a globally coordinated message flow, you must modify the configurable services properties that are associated with that vendor. The following table shows the properties that have been added to the configurable service for BEA WebLogic.

JMS provider	Property	Purpose	Default value
BEA_WebLogic	`proprietaryAPIHandler`	The name of the IBM supplied Java class to interface with a JMS provider's proprietary API.	`com.ibm.broker.apihandler. BEAWebLogicAPIHandler`
	`proprietaryAPIAttr1`	The Initial Context Factory class name for the vendor	`weblogic.jndi. WLInitialContextFactory`
	`proprietaryAPIAttr2`	The URL of the WebLogic bindings	*URL JNDI bindings*
	`proprietaryAPIAttr3`	The DNS name of the JMS server	*Server name*

In the list of JMS provider configurable services, the name of the IBM supplied Java class is set to the default value for the `proprietaryAPIHandler` property. Typically, you do not need to change this value, unless you are instructed to do so by an IBM Service team representative.

Procedure

- Use the **mqsichangeproperties** command to modify values of the properties for this JMS provider.

  The following example shows how to change the values of the properties `proprietaryAPIAttr2` and `proprietaryAPIAttr3` for the JMS provider configurable service definition called BEA_Weblogic, where these properties represent the URL of the WebLogic bindings and the DNS Server name of the BEA WebLogic JMS Server:

  ```
  mqsichangeproperties MB7BROKER -c JMSProviders -o BEA_Weblogic
  -n proprietaryAPIAttr2,proprietaryAPIAttr3 -v t3://9.20.94.16:7001,BEAServerName
  ```

- Use the **mqsireportproperties** command to display the properties for a JMS provider.

  The following example shows how to display the properties for all the broker's JMS provider resources (the default JMS provider resources and those configurable services that are defined with the **mqsicreateconfigurableservice** command):

  ```
  mqsireportproperties MB7BROKER -c JMSProviders -o BEA_WebLogic –r
  ```

  The result of this command has the following format:

```
ReportableEntityName=''
JMSProviders
  BEA_Weblogic=''
    jarsURL='default_Path'
    nativeLibs='default_Path'
    proprietaryAPIAttr1='weblogic.jndi.WLInitialContextFactory'
    proprietaryAPIAttr2='t3://9.20.94.16:7001'
    proprietaryAPIAttr3='BEAServerName'
    proprietaryAPIAttr4='default_none'
    proprietaryAPIAttr5='default_none'
    proprietaryAPIHandler='com.ibm.broker.apihandler.BEAWebLogicAPIHandler'
```

The default location for the JMS provider JAR files is the broker's shared-classes directory. You can specify an alternative location for the JAR files by using the **mqsichangeproperties** command, as shown in the following example:

```
mqsichangeproperties MB7BROKER -c JMSProviders -o BEA_WebLogic -n jarsURL
-v /var/mqsi/WebLogic
```

On Windows, the file location cannot be a mapped network drive on a remote Windows computer; the directory must be local or on a Storage Area Network (SAN) disk.

- Use the **mqsicreateconfigurableservice** command to add a JMS provider.

  The following example shows how to add a JMS provider called BEAV91 for broker MB7BROKER, specifying the name of an IBM supplied Java class called `com.ibm.broker.apihandler.BEAWebLogicAPIHandler` to handle vendor-specific API calls:

  ```
  mqsicreateconfigurableservice MB7BROKER -c JMSProviders -o BEAV91
  -n proprietaryAPIHandler,proprietaryAPIAttr1,proprietaryAPIAttr2,proprietaryAPIAttr3
  –v com.ibm.broker.apihandler.BEAWebLogicAPIHandler,weblogic.jndi.WLInitialContextFactory,
  t3://9.20.94.16:7001,BEAServerName
  ```

- If you have defined a user-defined JMS provider configurable service, set the value for the `proprietaryAPIHandler` property manually.

Related concepts:

"Configurable services" on page 1296
Configurable services are typically runtime properties. You can use them to define properties that are related to external services on which the broker relies; for example, an SMTP server or a JMS provider.

Related reference:

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"JMSInput node" on page 4532
Use the JMSInput node to receive messages from JMS destinations. JMS destinations are accessed through a connection to a JMS provider.

"JMSOutput node" on page 4549
Use the JMSOutput node to send messages to JMS destinations.

"SOAPInput node" on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

"SOAPRequest node" on page 4828
Use the SOAPRequest node to send a SOAP request to the remote Web service.

"SOAPAsyncRequest node" on page 4750
Use the SOAPAsyncRequest node with the SOAPAsyncResponse node to construct a pair of message flows that call a Web service asynchronously.

Connecting to different versions of the same JMS provider:

To use different versions of the same JMS provider, create a configurable service for the JMS provider, and set the jarsURL property to a unique path.

About this task

To connect to two versions of the same JMS provider (for example, JBoss), complete the following steps.

Procedure

1. Create a separate JMSProviders configurable service for each version of the JMS provider.

   For more information, see "**mqsicreateconfigurableservice** command" on page 3849 or "Creating a new configurable service" on page 645.
2. Set the **jarsURL** property of the JMSProviders configurable service to a unique path.

   For a description of the properties of the JMSProviders configurable service, see "JMSProviders configurable service" on page 3780.

Related concepts:

"Connection to different JMS providers" on page 1709
The JMSInput and JMSOutput nodes are compatible with all JMS providers that conform to the Java Message Service Specification, version 1.1.

Related tasks:

"Creating a new configurable service" on page 645
Use the WebSphere Message Broker Explorer to create a new configurable service to define properties for an external service on which the broker relies.

"Working with JMS" on page 1709
Learn about the concepts and tasks involved in configuring message flows to support JMS messages.

Related reference:

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

"Troubleshooting JMS nodes" on page 1730
Review possible problems with nodes using JMS transport.

"JMSInput node" on page 4532
Use the JMSInput node to receive messages from JMS destinations. JMS destinations are accessed through a connection to a JMS provider.

"JMSOutput node" on page 4549
Use the JMSOutput node to send messages to JMS destinations.

Configuring the JMSInput node for batch message processing:

Configure JMS message flows to send a batch acknowledgment for receipt of non-transactional JMS messages.

About this task

When the JMSInput node works in non-transactional mode, receipt and acknowledgment of messages take place in one step, followed by the message processing. In some scenarios, this acknowledgment response to the JMS server for each message might create an unacceptable level of network traffic. For example, using this messaging model to receive JMS messages across a wide area network that is already handling large volumes of traffic might result in non-optimal throughput rates for JMS messages.

The JMSInput node can acknowledge message receipt in batches rather than individually for non-transactional messages. Batch acknowledgment is enabled using the JMSProviders configurable service properties **clientAckBatchSize** and **clientAckBatchTime**. You can set these properties separately, or use them together, to tune the number of messages that are received and processed by the node before an acknowledgment response is returned to the source JMS server.

clientAckBatchSize
: This an integer value that represents the threshold number of messages received before the batch acknowledgment is sent.

clientAckBatchTime
: This is an integer value that represents the length, in milliseconds, of a repeating interval. At the end of each interval a batch acknowledgment is sent for all unacknowledged non-transactional JMS messages that were received during the preceding interval.

A batch acknowledgment is also sent when:

- There are no more input messages on the JMS server
- An error occurs during message processing. In this case, all previous messages in the batch that were successfully processed are first acknowledged, before handling the error.
- The message flow stops.

To disable batch acknowledgment, set both **clientAckBatchSize** and **clientAckBatchTime** to `0`.

Related concepts:

"Configurable services" on page 1296
Configurable services are typically runtime properties. You can use them to define properties that are related to external services on which the broker relies; for example, an SMTP server or a JMS provider.

Related reference:

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and

properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"JMSInput node" on page 4532
Use the JMSInput node to receive messages from JMS destinations. JMS destinations are accessed through a connection to a JMS provider.

Importing a policy set and policy set binding:

Use the **mqsichangeproperties** command to import a policy set and associated binding.

About this task

This topic shows how to import policy set *myPolicySet* to broker *myBroker* from a file called *myPolicySet.xml*. The associated binding is *myPolicySetBinding*, which you import from *myPolicySetBinding.xml*.

Procedure

1. Create a configurable service for the policy set, if one does not already exist.

   ```
   mqsicreateconfigurableservice myBroker -c PolicySets -o myPolicySet
   BIP8071I: Successful command completion.
   ```

2. Create a configurable service for the policy set binding, if one does not already exist.

   ```
   mqsicreateconfigurableservice myBroker -c PolicySetBindings -o myPolicySetBinding
   BIP8071I: Successful command completion.
   ```

3. Import the policy set.

   ```
   mqsichangeproperties myBroker -c PolicySets -o myPolicySet -n ws-security -p myPolicySet.xml
   ```

4. Import the policy set binding.

   ```
   mqsichangeproperties myBroker -c PolicySetBindings -o myPolicySetBinding
      -n ws-security -p myPolicySetBinding.xml
   ```

5. Change the value of the associatedPolicySet attribute. Set it to the name of the policy set with which this policy set binding was originally associated.

   ```
   mqsichangeproperties myBroker -c PolicySetBindings -o myPolicySetBinding
      -n associatedPolicySet -v myPolicySet
   ```

Related concepts:

"Policy sets" on page 774
Policy sets and bindings define and configure your WS-Security requirements, supported by WebSphere Message Broker, for the SOAPInput, SOAPReply, SOAPRequest, SOAPAsyncRequest, and SOAPAsyncResponse nodes.

Related tasks:

"Exporting a policy set and policy binding" on page 786
Use the **mqsireportproperties** command to export a policy set and associated binding to a file.

Related reference:

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

Configuring for email:

You can use configurable services to enable the EmailOutput node and EmailInput node in the broker runtime environment to connect with email servers.

About this task

The following topics describe how to prepare the environment to connect to email servers, and how to change the connection details without the need to redeploy your message flow.

- "Changing connection information for the EmailOutput node" on page 1798
- "Changing connection information for the EmailInput node" on page 1805

Configuring internal resources required by flows

You can use configurable services to configure the internal resources that are required by some message flows.

About this task

The following topics describe how to configure the storage of events for certain message flow nodes.

- "Configuring the storage of events for aggregation nodes"
- "Configuring the storage of events for Collector nodes" on page 755
- "Configuring the storage of events for Resequence nodes" on page 758
- "Configuring the storage of events for timeout nodes" on page 760
- "Configuring the XPath cache" on page 765
- "Configuring monitoring event sources using a monitoring profile" on page 762

Configuring the storage of events for aggregation nodes:

You can use an Aggregation configurable service to control the storage of events for AggregateControl and AggregateReply nodes.

About this task

By default, the storage queues used by all aggregation nodes are:

- SYSTEM.BROKER.AGGR.CONTROL
- SYSTEM.BROKER.AGGR.REPLY
- SYSTEM.BROKER.AGGR.REQUEST
- SYSTEM.BROKER.AGGR.UNKNOWN
- SYSTEM.BROKER.AGGR.TIMEOUT

However, you can control the queues that are used by different aggregation nodes by creating alternative queues containing a *QueuePrefix*, and using an Aggregation configurable service to specify the names of those queues for storing events.

Follow these steps to specify the queues that are used to store event states, and to set the expiry time of an aggregation:

Procedure

1. Create the storage queues to be used by the aggregation nodes. The following queues are required:

- SYSTEM.BROKER.AGGR.*QueuePrefix*.CONTROL
- SYSTEM.BROKER.AGGR.*QueuePrefix*.REPLY
- SYSTEM.BROKER.AGGR.*QueuePrefix*.REQUEST
- SYSTEM.BROKER.AGGR.*QueuePrefix*.UNKNOWN
- SYSTEM.BROKER.AGGR.*QueuePrefix*.TIMEOUT

The *QueuePrefix* variable can contain any characters that are valid in a WebSphere MQ queue name, but must be no longer than eight characters and must not begin or end with a period (.). For example, `SET1` and `SET.1` are valid queue prefixes, but `.SET1` and `SET1.` are invalid.

If you do not create the storage queues, WebSphere Message Broker creates the set of queues when the node is deployed; these queues are based on the default queues. If the queues cannot be created, the message flow is not deployed.

2. Use the **mqsicreateconfigurableservice** command to create an Aggregation configurable service. You can create a configurable service to be used with either a specific aggregation or with all aggregations in an execution group.
   a. If the configurable service is to be used with a specific aggregation, ensure that the name of the configurable service is the same as the name that you specify in the `Aggregate name` property on the AggregateControl and AggregateReply nodes. If the configurable service is to be used with all aggregations in an execution group, create the configurable service with the same name as the execution group.
   b. Set the **Queue prefix** property to the required value.
   c. Optional: Set the **Timeout** property to control the expiry time of an aggregation.

   For example, create a configurable service called `myAggregation`, which specifies queues prefixed with SYSTEM.BROKER.AGGR.SET1 and a timeout of 60 seconds:

   ```
   mqsicreateconfigurableservice MYBROKER -c Aggregation -o myAggregation
   -n queuePrefix,timeoutSeconds -v SET1,60
   ```

   You can use the **mqsideleteconfigurableservice** command to delete the Aggregation configurable service. However, the storage queues are not deleted automatically when the configurable service is deleted, so you must delete them separately.For more information, see “Configurable services properties” on page 3766
3. In the AggregateControl and AggregateReply nodes:
   a. Ensure that the name of the Aggregation configurable service is the same as the name specified in the `Aggregate name` property on the **Basic** tab; for example, `myAggregation`. If no Aggregation configurable service exists with the same name as the `Aggregate name` property, and if a configurable service exists with the same name as the execution group, that configurable service is used instead.
   b. Optional: Use the **mqsichangeproperties** and **mqsireportproperties** commands to change or view the properties of the configurable service. Alternatively, you can use the WebSphere Message Broker Explorer to view or modify a configurable service. For more information about working with configurable services, see “Using the WebSphere Message Broker Explorer to work with configurable services” on page 644.

What to do next

The properties for the configurable service are not used by the broker until you restart or redeploy the message flow, or restart the broker.

Related concepts:

"Using the WebSphere Message Broker Explorer to work with configurable services" on page 644
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view, add, modify and delete configurable services.

Related tasks:

"Viewing configurable services" on page 647
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view the properties of configurable services defined on your broker.

"Modifying an IBM defined configurable service" on page 648
You can create a new configurable service for an external service on which the broker relies, by modifying predefined configurable services provided by IBM. Use the WebSphere Message Broker Explorer to view and modify existing configurable services.

"Modifying a configurable service" on page 649
Use the WebSphere Message Broker Explorer to view and modify existing configurable services.

"Deleting a configurable service" on page 652
Use the WebSphere Message Broker Explorer to delete custom configurable services.

Related reference:

"AggregateControl node" on page 4296
Use the AggregateControl node to mark the beginning of a fan-out of requests that are part of an aggregation.

"AggregateRequest node" on page 4303
Use the AggregateRequest node to record the fact that request messages have been sent. This node also collects information that helps the AggregateReply node to construct the compound response message.

"AggregateReply node" on page 4299
Use the AggregateReply node to mark the end of an aggregation fan-in. This node collects replies and combines them into a single compound message.

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

Configuring the storage of events for Collector nodes:

You can use a Collector configurable service to control the storage of events for Collector nodes.

About this task

By default, the storage queues used by all Collector nodes are:

- SYSTEM.BROKER.EDA.EVENTS
- SYSTEM.BROKER.EDA.COLLECTIONS

These queues are also used by the Resequence node.

However, you can control the queues that are used by different Collector nodes by creating alternative queues that contain a *QueuePrefix* variable, and by using a Collector configurable service to specify the names of those queues for storing events.

Follow these steps to specify the queues that are used to store event states, and to set the expiry for the collection:

Procedure

1. Create the storage queues to be used by the Collector node. The following queues are required:
   - SYSTEM.BROKER.EDA.*QueuePrefix*.EVENTS
   - SYSTEM.BROKER.EDA.*QueuePrefix*.COLLECTIONS

   The *QueuePrefix* variable can contain any characters that are valid in a WebSphere MQ queue name, but must be no longer than eight characters and must not begin or end with a period (.). For example, `SET1` and `SET.1` are valid queue prefixes, but `.SET1` and `SET1.` are invalid.

   If you do not create the storage queues, WebSphere Message Broker creates the set of queues when the node is deployed; these queues are based on the default queues. If the queues cannot be created, the message flow is not deployed.
2. Use the **mqsicreateconfigurableservice** command to create a Collector configurable service. You can create a configurable service to be used with either a specific collection or with all collections in an execution group.
   a. If you are creating a configurable service to be used with a specific collection, ensure that the name of the configurable service is the same as the name that you specify in the `Configurable service` property on the Collector node. If you are creating a configurable service to be used with all collections in the execution group, ensure that the configurable service has the same name as the execution group.
   b. Set the **Queue prefix** property to the required value.
   c. Optional: Set the **Collection expiry** property.

   For example, create a Collector configurable service called `myCollectorService`, which uses queues prefixed with SYSTEM.BROKER.EDA.SET1, and with a collection expiry of 60 seconds:

   ```
   mqsicreateconfigurableservice MYBROKER -c Collector -o myCollectorService
   -n queuePrefix,collectionExpirySeconds -v SET1,60
   ```

   You can use the **mqsideleteconfigurableservice** command to delete the Collector configurable service. However, the storage queues are not deleted automatically when the configurable service is deleted, so you must delete them separately.

   For more information, see "Configurable services properties" on page 3766
3. In the Collector node:
   a. If the configurable service is to be used for a specific collection, specify the name of the configurable service in the `Configurable service` property on

the **Advanced** tab; for example, `myCollectorService`. If you do not set the `Configurable service` property, and if a configurable service exists with the same name as the execution group, that configurable service is used instead.

b. Optional: Use the **mqsichangeproperties** and **mqsireportproperties** commands to change or view the properties of the configurable service. Alternatively, you can use the WebSphere Message Broker Explorer to view or modify a configurable service. For more information about working with configurable services, see "Using the WebSphere Message Broker Explorer to work with configurable services" on page 644.

What to do next

The properties for the configurable service are not used by the broker until you restart or redeploy the message flow, or restart the broker.

Related concepts:

"Using the WebSphere Message Broker Explorer to work with configurable services" on page 644
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view, add, modify and delete configurable services.

Related tasks:

"Viewing configurable services" on page 647
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view the properties of configurable services defined on your broker.

"Modifying an IBM defined configurable service" on page 648
You can create a new configurable service for an external service on which the broker relies, by modifying predefined configurable services provided by IBM. Use the WebSphere Message Broker Explorer to view and modify existing configurable services.

"Modifying a configurable service" on page 649
Use the WebSphere Message Broker Explorer to view and modify existing configurable services.

"Deleting a configurable service" on page 652
Use the WebSphere Message Broker Explorer to delete custom configurable services.

Related reference:

"Collector node" on page 4333
Use the Collector node to create message collections based on rules that you configure in the node.

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

Configuring the storage of events for Resequence nodes:

You can use a Resequence configurable service to control the storage of events for Resequence nodes.

About this task

By default, the storage queues used by all Resequence nodes are:

- SYSTEM.BROKER.EDA.EVENTS
- SYSTEM.BROKER.EDA.COLLECTIONS

These queues are also used by the Collector node.

However, you can control the queues that are used by different Resequence nodes by creating alternative queues that contain a *QueuePrefix* variable, and by using a Resequence configurable service to specify the names of those queues for storing events.

Follow these steps to specify the queues that are used to store event states, and to set the timeout and the start and end of the sequence:

Procedure

1. Create the storage queues to be used by the Resequence node. The following queues are required:
   - SYSTEM.BROKER.EDA.*QueuePrefix*.EVENTS
   - SYSTEM.BROKER.EDA.*QueuePrefix*.COLLECTIONS

   The *QueuePrefix* variable can contain any characters that are valid in a WebSphere MQ queue name, but must be no longer than eight characters and must not begin or end with a period (.). For example, `SET1` and `SET.1` are valid queue prefixes, but `.SET1` and `SET1.` are invalid.

   If you do not create the storage queues, WebSphere Message Broker creates the set of queues when the node is deployed; these queues are based on the default queues. If the queues cannot be created, the message flow is not deployed.
2. Use the **mqsicreateconfigurableservice** command to create a Resequence configurable service. You can create a configurable service to be used with either a specific sequence or with all sequences in an execution group.
   a. If you are creating a configurable service to be used with a specific sequence, ensure that the name of the configurable service is the same as the name that you specify in the `Configurable service` property on the Resequence node. If you are creating a configurable service to be used with all sequences in the execution group, ensure that the configurable service has the same name as the execution group.
   b. Set the **Queue prefix** property to the required value.
   c. Optional: Set the **Missing message timeout**, **Start of sequence**, and **End of sequence** properties.

   For example, create a Resequence configurable service called `myResequenceService`, which uses queues prefixed with SYSTEM.BROKER.EDA.SET1, with a missing message timeout of 60 seconds, and which waits five seconds before determining the start and end numbers in a sequence:

```
mqsicreateconfigurableservice MYBROKER -c Resequence -o myResequenceService
-n queuePrefix,missingMessageTimeoutSeconds,startSequenceSeconds,endSequenceSeconds -v SET1,60,5,
```

You can use the **mqsideleteconfigurableservice** command to delete the Resequence configurable service. However, the storage queues are not deleted automatically when the configurable service is deleted, so you must delete them separately.For more information, see "Configurable services properties" on page 3766

3. In the Resequence node:
   a. If the configurable service is to be used for a specific sequence, specify the name of the configurable service on the **Advanced** tab; for example, `myResequenceService`. If you do not set the `Configurable service` property, and if a configurable service exists with the same name as the execution group, that configurable service is used instead.
   b. Optional: Use the **mqsichangeproperties** and **mqsireportproperties** commands to change or view the properties of the configurable service. Alternatively, you can use the WebSphere Message Broker Explorer to view or modify a configurable service. For more information about working with configurable services, see "Using the WebSphere Message Broker Explorer to work with configurable services" on page 644.

What to do next

The properties for the configurable service are not used by the broker until you restart or redeploy the message flow, or restart the broker.

Related concepts:

"Using the WebSphere Message Broker Explorer to work with configurable services" on page 644
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view, add, modify and delete configurable services.

Related tasks:

"Viewing configurable services" on page 647
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view the properties of configurable services defined on your broker.

"Modifying an IBM defined configurable service" on page 648
You can create a new configurable service for an external service on which the broker relies, by modifying predefined configurable services provided by IBM. Use the WebSphere Message Broker Explorer to view and modify existing configurable services.

"Modifying a configurable service" on page 649
Use the WebSphere Message Broker Explorer to view and modify existing configurable services.

"Deleting a configurable service" on page 652
Use the WebSphere Message Broker Explorer to delete custom configurable services.

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"Resequence node" on page 4651
Use the Resequence node to control the sequence in which a group (or groups) of incoming messages are propagated in a message flow.

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create,

are defined by their names and properties. You can use the supplied services.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

Configuring the storage of events for timeout nodes:

You can use a Timer configurable service to control the storage of events for TimeoutNotification and TimeoutControl nodes.

About this task

By default, the storage queue used by all timeout nodes is the SYSTEM.BROKER.TIMEOUT.QUEUE.

However, you can control the queues that are used by different timeout nodes by creating alternative queues that contain a *QueuePrefix* variable, and by using a Timer configurable service to specify the names of those queues for storing events.

Follow these steps to specify the queue that is used to store event states:

Procedure

1. Create the storage queue to be used by the timeout nodes. The following queue is required:
   - SYSTEM.BROKER.TIMEOUT.*QueuePrefix*.QUEUE

   The *QueuePrefix* variable can contain any characters that are valid in a WebSphere MQ queue name, but must be no longer than eight characters and must not begin or end with a period (.). For example, `SET1` and `SET.1` are valid queue prefixes, but `.SET1` and `SET1.` are invalid.

   If you do not create the storage queue, WebSphere Message Broker creates the queue when the node is deployed; this queue is based on the default queue. If the queue cannot be created, the message flow is not deployed.
2. Use the **mqsicreateconfigurableservice** command to create a Timer configurable service. You can create a configurable service to be used with either specific timeout requests or with all timeout requests in an execution group.
   a. If the configurable service is to be used with specific timeout requests, create the configurable service with the same name as the `Unique identifier` property on the TimeoutNotification and TimeoutControl nodes. If the configurable service is to be used with all timeout requests in an execution group, create the configurable service with the same name as the execution group.
   b. Set the **Queue prefix** property to the required value.

   For example, create a Timer configurable service that uses a queue prefixed with SYSTEM.BROKER.TIMEOUT.SET1:

```
mqsicreateconfigurableservice MB7BROKER -c Timer -o myTimer
-n queuePrefix -v SET1
```

You can use the **mqsideleteconfigurableservice** command to delete the Timer configurable service. However, the storage queue is not deleted automatically when the configurable service is deleted, so you must delete it separately.For more information, see "Configurable services properties" on page 3766.

3. In the TimeoutNotification and TimeoutControl nodes:
   a. Ensure that the name of the Timer configurable service is the same as the name specified in the `Unique Identifier` property on the **Basic** tab; for example, `myTimer`. If there is no Timer configurable service with the same name as the `Unique Identifier`, and if there is a configurable service with the same name as the execution group, that configurable service is used instead.
   b. Optional: Use the **mqsichangeproperties** and **mqsireportproperties** commands to change or view the properties of the configurable service. Alternatively, you can use the WebSphere Message Broker Explorer to view or modify a configurable service. For more information about working with configurable services, see "Using the WebSphere Message Broker Explorer to work with configurable services" on page 644.

What to do next

The properties for the configurable service are not used by the broker until you restart or redeploy the message flow, or restart the broker.

Related concepts:

"Using the WebSphere Message Broker Explorer to work with configurable services" on page 644
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view, add, modify and delete configurable services.

"Configuring timeout flows" on page 2809
Use the TimeoutControl and TimeoutNotification nodes in message flows to process timeout requests or to generate timeout notifications at specified intervals.

"Sending timeout request messages" on page 2810
To set a controlled timeout, send a message with a set of elements with well known names to a TimeoutControl node. These elements control the properties of the timeout to be created or deleted.

Related tasks:

"Viewing configurable services" on page 647
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view the properties of configurable services defined on your broker.

"Modifying an IBM defined configurable service" on page 648
You can create a new configurable service for an external service on which the broker relies, by modifying predefined configurable services provided by IBM. Use the WebSphere Message Broker Explorer to view and modify existing configurable services.

"Modifying a configurable service" on page 649
Use the WebSphere Message Broker Explorer to view and modify existing configurable services.

"Deleting a configurable service" on page 652
Use the WebSphere Message Broker Explorer to delete custom configurable services.

Related reference:

"TimeoutNotification node" on page 4936
Use the TimeoutNotification node to manage timeout-dependent message flows.

"TimeoutControl node" on page 4932
Use the TimeoutControl node to process an input message that contains a timeout request.

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

Configuring monitoring event sources using a monitoring profile:

You can create a monitoring profile and use the **mqsichangeflowmonitoring** command to configure your message flows to emit monitoring events.

Before you begin

Before you start:

Read the following topics:

- "Business-level monitoring" on page 3319
- "Monitoring basics" on page 3320

You must have a message flow that contains a node to which you want to add a monitoring event.

You can use XPath 1.0 expressions to configure a monitoring event.

Creating a monitoring profile:
About this task

First create a monitoring profile XML file. This is a file that lists the event sources in the message flow that will emit events, and defines the properties of each event.

Procedure

Follow the guidance at "Monitoring profile" on page 6768 to create your monitoring profile XML file.

Applying a monitoring profile:
About this task

When you have created a monitoring profile XML file, follow these steps to apply it.

Procedure

1. Use the **mqsicreateconfigurableservice** command to create a configurable service for the monitoring profile. In the following command example, replace *myBroker* with the name of your broker, and *myMonitoringProfile* with the name of your monitoring profile.

   ```
   mqsicreateconfigurableservice myBroker -c MonitoringProfiles
        -o myMonitoringProfile
   ```

2. Use the **mqsichangeproperties** command to associate your monitoring profile XML file with the configurable service. In the following command example, replace *myBroker* with the name of your broker, *myMonitoringProfile* with the name of your monitoring profile, and *myMonitoringProfile.xml* with the name of the monitoring profile XML file.

   ```
   mqsichangeproperties myBroker -c MonitoringProfiles -o myMonitoringProfile
        -n profileProperties -p myMonitoringProfile.xml
   ```

   Set the `useParserNameInMonitoringPayload` property to TRUE to force the `wmb:applicationData/wmb:complexContent/wmb:elementName` attribute to hold the name of the input node parser, if present. See "MonitoringProfiles configurable service" on page 3781 for details.

   ```
   mqsichangeproperties myBroker -c MonitoringProfiles -o myMonitoringProfile
        -n useParserNameInMonitoringPayload -v TRUE
   ```

3. Use the **mqsichangeflowmonitoring** command to apply a monitoring profile configurable service to one or more message flows.

   - Apply a monitoring profile to a single message flow *messageflow1* in execution group *EG1*:

     ```
     mqsichangeflowmonitoring myBroker -e EG1
          -f messageflow1 -m myMonitoringProfile
     ```

   - Apply a monitoring profile to all message flows in all execution groups:

     ```
     mqsichangeflowmonitoring myBroker -g -j -m myMonitoringProfile
     ```

   Monitoring for the flow is inactive; applying the monitoring profile does not activate it.

4. Alternatively, use the broker archive editor to apply a monitoring profile configurable service to one or more message flows, by setting message flow property **Monitoring Profile Name**.

   a. In the WebSphere Message Broker Toolkit, switch to the Broker Application Development perspective.

   b. In the Broker Development view, right-click the BAR file, then click **Open with** > **Broker Archive Editor**.

   c. Click the **Manage and Configure** tab.

   d. Click the message flow on which you want to set the monitoring profile configurable service. The properties that you can configure for the message flow are displayed in the **Properties** view.

   e. In the **Monitoring Profile Name** field, enter the name of a monitoring profile.

   f. Save the BAR file.

   g. Deploy the BAR file.

   Monitoring for the flow is inactive; deploying the BAR file does not activate it.

5. Activate monitoring for the flow using the `mqsichangeflowmonitoring -c` command.

   - Activate monitoring for a single message flow *messageflow1* in execution group *EG1*:

```
mqsichangeflowmonitoring myBroker -e EG1
     -f messageflow1 -c active
```

- Activate monitoring for all message flows in all execution groups:

```
mqsichangeflowmonitoring myBroker -g -j -c active
```

Updating a monitoring profile:

Procedure

1. Follow the guidance at "Monitoring profile" on page 6768 to update your monitoring profile XML file.
2. Use the **mqsichangeproperties** command to update the configurable service to use the new XML file. For example:

```
mqsichangeproperties myBroker -c MonitoringProfiles -o myMonitoringProfile
     -n profileProperties -p myMonitoringProfile.xml
```

Related concepts:

"Monitoring basics" on page 3320
Message flows can be configured to emit events. The events can be read and used by other applications for transaction monitoring, transaction auditing, and business process monitoring.

"Configurable services" on page 1296
Configurable services are typically runtime properties. You can use them to define properties that are related to external services on which the broker relies; for example, an SMTP server or a JMS provider.

Related tasks:

"Business-level monitoring" on page 3319
You can configure your message flow to emit event messages that can be used to support transaction monitoring and auditing, and business process monitoring.

"Activating monitoring" on page 3334
Use the **mqsichangeflowmonitoring** command to activate monitoring after you have configured monitoring event sources.

Related reference:

"Monitoring profile" on page 6768
To customize events after a message flow has been deployed, but without redeploying the flow, use a monitoring profile configurable service. By using this service, you can apply a monitoring profile to one or more message flows.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsichangeflowmonitoring** command" on page 3738
Use the **mqsichangeflowmonitoring** command to enable monitoring of message flows.

"**mqsireportflowmonitoring** command" on page 3924
Use the **mqsireportflowmonitoring** command to display the current options for monitoring that have been set using the **mqsichangeflowmonitoring** command.

"Example XPath expressions for event filtering" on page 6781
Use numeric, string, or Boolean expressions when configuring an event source, to determine whether the event is emitted.

Configuring the XPath cache:

The XPath cache has a default size of 100 elements. However, this default size might become a performance bottleneck for customers who use many XPath expressions with a single flow invocation completely invalidating the cache. Altering the size of the XPath cache might improve message flow performance.

An Execution Group keeps a cache of compiled XPath expressions to help reduce the processor usage of parsing and re-creating XPath expressions that are used repeatedly during Message Flow execution. This cache is shared by all Message Flows within an Execution Group. The default size of this cache is 100.

It might be necessary to alter the size of this cache for optimal message flow performance if many XPath expressions are created for each Message Flow invocation. In a highly multi-threaded environment where many XPath expressions are evaluated on each Message Flow invocation, it might be necessary to disable the cache to remove thread contention on the cache.

The property can be set by running the following **mqsichangeproperties** command:

```
mqsichangeproperties <broker> -e <eg> -o ExecutionGroup -n compiledXPathCacheSizeEntries -v <value
```

where *<value>* is the size of the cache to be set. The size can be set to any value greater than or equal to 100. A value of 0 means that the cache is disabled. The default value is 100.

The configured value can be reported by running the following **mqsireportproperties** command:

```
mqsireportproperties <broker> -e <eg> -o ExecutionGroup -n compiledXPathCacheSizeEntries
```

and can also be reported as part of the other ExecutionGroup level properties:

```
mqsireportproperties <broker> -e <eg> -o ExecutionGroup -a
```

To disable the cache for broker *BRK1* and execution group *EG1*, run the following command:

```
mqsichangeproperties BRK1 -e EG1 -o ExecutionGroup -n compiledXPathCacheSizeEntries -v 0
```

Related concepts:

"XPath overview" on page 1507
The XML Path Language (XPath) is used to uniquely identify or address parts of an XML document. An XPath expression can be used to search through an XML document, and extract information from any part of the document, such as an element or attribute (referred to as a *node* in XML) in it. XPath can be used alone or in conjunction with XSLT.

Related tasks:

"Using XPath" on page 1506
XPath provides an alternative method to ESQL for entering expressions in the property fields of specific built-in nodes.

WS-Security

Web Services Security (WS-Security) describes enhancements to SOAP messaging to provide quality of protection through message integrity, message confidentiality, and single message authentication. WS-Security mechanisms can be used to accommodate a wide variety of security models and encryption technologies.

WS-Security is a message-level standard that is based on securing SOAP messages through XML digital signature, confidentiality through XML encryption, and credential propagation through security tokens. The Web services security specification defines the facilities for protecting the integrity and confidentiality of a message and provides mechanisms for associating security-related claims with the message.

WS-Security provides a general-purpose mechanism for associating security tokens with messages. No specific type of security token is required by WS-Security. It is designed to be extensible, for example, to support multiple security token formats.

WS-Security also describes how to encode binary security tokens and attach them to SOAP messages. Specifically, the WS-Security profile specifications describe how to encode the following tokens:

- Username tokens
- X.509 certificates
- SAML assertions
- Kerberos tickets
- LTPA binary tokens

With WS-Security, the domain of these mechanisms can be extended by carrying authentication information in Web services requests. WS-Security also includes extensibility mechanisms that can be used to further describe the credentials that are included with a message. WS-Security is a building block that can be used in conjunction with other Web service protocols to address a wide variety of application security requirements.

There are numerous advantages to using WS-Security.

- Different parts of a message can be secured in a variety of ways. For example, you can use integrity on the security token (user ID and password) and confidentiality on the SOAP message body.
- Intermediaries can be used and end-to-end message-level security can be provided through any number of intermediaries.
- WS-Security works across multiple transports and is independent of the underlying transport protocol.
- Authentication of both individual users and multiple party identities is possible.

Traditional Web security mechanisms, such as HTTPS, might be insufficient to manage the security requirements of all Web service scenarios. For example, when an application sends a SOAP message using HTTPS, the message is secured only for the HTTPS connection, meaning during the transport of the message between the service requester (the client) and the service. However, the application might require that the message data be secured beyond the HTTPS connection, or even beyond the transport layer. By securing Web services at the message level, message-level security is capable of meeting these expanded requirements.

Message-level security, or securing Web services at the message level, addresses the same security requirements as for traditional Web security. These security requirements include: identity, authentication, authorization, integrity, confidentiality, nonrepudiation, and basic message exchange. Both traditional Web and message-level security share many of the same mechanisms for handling security, including digital certificates, encryption, and digital signatures.

With message-level security, the SOAP message itself either contains the information needed to secure the message or it contains information about where to get that information to handle security needs. The SOAP message also contains information relevant to the protocols and procedures for processing the specified message-level security. However, message-level security is not tied to any particular transport mechanism. Because the security information is part of the message, it is independent of a transport protocol, such as HTTPS.

The client adds to the SOAP message header security information that applies to that particular message. When the message is received, the Web service endpoint, using the security information in the header, verifies the secured message and validates it against the policy. For example, the service endpoint might verify the message signature and check that the message has not been tampered with. It is possible to add signature and encryption information to the SOAP message headers, as well as other information such as security tokens for identity (for example, an X.509 certificate) that are bound to the SOAP message content.

Without message-level security, the SOAP message is sent in clear text, and personal information such as a user ID or an account number is not protected. Without applying message-level security, there is only a SOAP body under the SOAP envelope in the SOAP message. By applying features from the WS-Security specification, the SOAP security header is inserted under the SOAP envelope in the SOAP message when the SOAP body is signed and encrypted.

To keep the integrity or confidentiality of the message, digital signatures and encryption are typically applied.

- Confidentiality specifies the confidentiality constraints that are applied to generated messages. This includes specifying which message parts within the generated message must be encrypted, and the message parts to attach encrypted Nonce and time stamp elements to.
- Integrity is provided by applying a digital signature to a SOAP message. Confidentiality is applied by SOAP message encryption.

You can add an authentication mechanism by inserting various types of security tokens, such as the Username token (element). When the Username token is received by the Web service server, the user name and password are extracted and verified. Only when the user name and password combination is valid, will the message be accepted and processed at the server. Using the Username token is just one of the ways of implementing authentication. This mechanism is also known as basic authentication.

The OASIS Web Services Security Specification provides a set of mechanisms to help developers of Web Services secure SOAP message exchanges. For details of the OASIS Web Services Security Specification, see OASIS Standard for WS-Security Specification.

Related concepts:

"WS-Security mechanisms" on page 768
The WS-Security specification provides three mechanisms for securing Web services at the message level: authentication, integrity, and confidentiality.

"WS-Security capabilities" on page 790
Web service security capabilities are supported by the broker.

"Public key cryptography" on page 354
All encryption systems rely on the concept of a key. A key is the basis for a transformation, usually mathematical, of an ordinary message into an unreadable message. For centuries, most encryption systems have relied on private key encryption. Public key encryption is the only challenge to private key encryption that has appeared in the last 30 years.

"Digital certificates" on page 356
Certificates provide a way of authenticating users. Instead of requiring each participant in an application to authenticate every user, third-party authentication relies on the use of digital certificates.

"Digital signatures" on page 360
A digital signature is a number that is attached to a document. For example, in an authentication system that uses public-key encryption, digital signatures are used to sign certificates.

"Policy sets" on page 774
Policy sets and bindings define and configure your WS-Security requirements, supported by WebSphere Message Broker, for the SOAPInput, SOAPReply, SOAPRequest, SOAPAsyncRequest, and SOAPAsyncResponse nodes.

WS-Security mechanisms

The WS-Security specification provides three mechanisms for securing Web services at the message level: authentication, integrity, and confidentiality.

Authentication

This mechanism uses a security token to validate the user and determine whether a client is valid in a particular context. A client can be a user, computer, or application. Without authentication, an attacker can use spoofing techniques to send a modified SOAP message to the service provider.

In authentication, a security token is inserted in the request message. Depending on the type of security token that is being used, the security token can also be inserted in the response message. The following types of security token are supported for authentication:

- Username tokens
- X.509 certificates
- SAML assertions
- Kerberos tickets
- LTPA binary tokens

Username tokens are used to validate user names and passwords. When a Web service server receives a username token, the user name and password are extracted and passed to a user registry for verification. If the user name and password combination is valid, the result is returned to the server and the message is accepted and processed. When used in authentication, username tokens are typically passed only in the request message, not the response message.

X.509 tokens are validated by using a certificate path.

The broker support for SAML assertions is restricted to passing the token to a WS-Trust security token server (STS) for validation.

Kerberos tickets are validated against the host's Kerberos `keytab` file.

The broker support for LTPA binary tokens is restricted to passing the token to a WS-Trust STS for validation.

All types of token must be protected. For this reason, if you send them over an untrusted network, take one of the following precautions:
- Use HTTPS
- Configure the policy set to protect the appropriate elements in the SOAP header

Integrity

This mechanism uses message signing to ensure that information is not changed, altered, or lost accidentally. When integrity is implemented, an XML digital signature is generated on the contents of a SOAP message. If unauthorized changes are made to the message data, the signature is not validated. Without integrity, an attacker can use tampering techniques to intercept a SOAP message between the Web service client and server, and modify it.

Confidentiality

This mechanism uses message encryption to ensure that no party or process can access or disclose the information in the message. When a SOAP message is encrypted, only a service that knows the appropriate key can decrypt and read the message.

Related concepts:

"WS-Security" on page 765
Web Services Security (WS-Security) describes enhancements to SOAP messaging to provide quality of protection through message integrity, message confidentiality, and single message authentication. WS-Security mechanisms can be used to accommodate a wide variety of security models and encryption technologies.

"Public key cryptography" on page 354
All encryption systems rely on the concept of a key. A key is the basis for a transformation, usually mathematical, of an ordinary message into an unreadable message. For centuries, most encryption systems have relied on private key encryption. Public key encryption is the only challenge to private key encryption that has appeared in the last 30 years.

"Digital certificates" on page 356
Certificates provide a way of authenticating users. Instead of requiring each participant in an application to authenticate every user, third-party authentication relies on the use of digital certificates.

"Digital signatures" on page 360
A digital signature is a number that is attached to a document. For example, in an authentication system that uses public-key encryption, digital signatures are used to sign certificates.

"Policy sets" on page 774
Policy sets and bindings define and configure your WS-Security requirements, supported by WebSphere Message Broker, for the SOAPInput, SOAPReply, SOAPRequest, SOAPAsyncRequest, and SOAPAsyncResponse nodes.

Implementing WS-Security

Configure authentication, XML encryption, XML signature, and message expiration by using the Policy Sets and Policy Set Bindings editor.

About this task

You use the Policy Sets and Policy Set Bindings editor in the WebSphere Message Broker Explorer to configure the following aspects of WS-Security:

"Authentication"

"Confidentiality" on page 771

"Integrity" on page 772

"Expiration" on page 773

Authentication:
About this task

The following tokens are supported:

- Username
- X.509
- SAML assertions
- Kerberos tickets
- LTPA binary tokens

Configuring authentication with username tokens:

1. In the WebSphere Message Broker Explorer, right-click on the broker with which you want to work, and click **Properties**.
2. In the Properties window, select the Security tab, and click **Policy Sets**.
3. Create a policy set and add UserName authentication tokens to it; see "Policy Sets and Policy Set Bindings editor: Authentication tokens panel" on page 6843.
4. Further configure any X.509 authentication tokens defined in the associated policy set; see "Policy Sets and Policy Set Bindings editor: Authentication and Protection Tokens panel" on page 6854.
5. Configure a security profile; see "Message flow security and security profiles" on page 788.
6. Associate the policy set with a message flow or node; see "Associating policy sets and bindings with message flows and nodes" on page 785.

Configuring authentication with X.509 tokens:

1. If you are using the broker's truststore to hold the trusted certificate, you must configure it; see "Viewing and setting keystore and truststore runtime properties at broker level" on page 780 or "Viewing and setting keystore and truststore runtime properties at execution group level" on page 783 depending on where you want to set keystore and truststore runtime properties.
2. Create a policy set and add UserName and X.509 authentication tokens to it; see "Policy Sets and Policy Set Bindings editor: Authentication tokens panel" on page 6843.
3. Configure the certificate mode for either broker truststore or an external security provider; see "Policy Sets and Policy Set Bindings editor: Authentication and Protection Tokens panel" on page 6854.
4. If you are using an external security provider, configure a security profile; see "Message flow security and security profiles" on page 788.
5. Associate the policy set with a message flow or node; see "Associating policy sets and bindings with message flows and nodes" on page 785

Configuring authentication with SAML assertions:

1. In the WebSphere Message Broker Explorer, right-click on the broker with which you want to work, and click **Properties**.
2. In the Properties window, select the Security tab, and click **Policy Sets**.
3. Create a policy set and add SAML pass-through 1.1 or SAML pass-through 2.0 tokens to it; see "Policy Sets and Policy Set Bindings editor: Authentication tokens panel" on page 6843. SAML pass-though does not enforce subject confirmation, but the assertion is simply provided as a token to be processed in the external Security Token Server specified in the security profile that is associated with the node.
4. Configure a security profile. The security profile must be configured to use a WS-Trust v1.3 STS. For more information, see "Message flow security and security profiles" on page 788.
5. Associate the policy set with a message flow or node; see "Associating policy sets and bindings with message flows and nodes" on page 785.

Configuring authentication with Kerberos tickets:

1. In the WebSphere Message Broker Explorer, right-click on the broker with which you want to work, and click **Properties**.
2. In the Properties window, select the Security tab, and click **Policy Sets**.
3. Create a policy set and add your Kerberos token type as symmetric tokens; see "Policy Sets and Policy Set Bindings editor: Message Level Protection panel" on page 6845.
4. Associate the policy set with a message flow or node; see "Associating policy sets and bindings with message flows and nodes" on page 785.
5. Configure the host's Kerberos `keytab` file. For more information about Kerberos configuration, see the documentation for your broker's host system. For example, for Windows, see the "Step-by-Step Guide to Kerberos 5 (krb5 1.0) Interoperability", which you can access at `http://technet.microsoft.com/en-us/library/`.

Configuring authentication with LTPA binary tokens:

1. In the WebSphere Message Broker Explorer, right-click on the broker with which you want to work, and click **Properties**.
2. In the Properties window, select the Security tab, and click **Policy Sets**.
3. Create a policy set and add LTPA tokens to it; see "Policy Sets and Policy Set Bindings editor: Authentication tokens panel" on page 6843. The LTPA binary token is passed through to the external Security Token Server (STS) specified in the security profile that is associated with the node.
4. Configure a security profile. The security profile must be configured to use a WS-Trust v1.3 STS. For more information, see "Message flow security and security profiles" on page 788.
5. Associate the policy set with a message flow or node; see "Associating policy sets and bindings with message flows and nodes" on page 785.

Confidentiality:
About this task

Confidentiality is provided by XML encryption, and requires either X.509 tokens or Kerberos tickets.

Configuring XML encryption with X.509 tokens:

1. If you are using the broker's truststore to hold the trusted certificate, you must configure it; see "Viewing and setting keystore and truststore

runtime properties at broker level" on page 780 or "Viewing and setting keystore and truststore runtime properties at execution group level" on page 783, depending on where you want to set keystore and truststore runtime properties.

2. In the WebSphere Message Broker Explorer, right-click on the broker with which you want to work, and click **Properties**.
3. In the Properties window, select the **Security** tab, and click **Policy Sets**.
4. Create a policy set, enable XML encryption, create encryption tokens, and select the encryption algorithms that you will use; see "Policy Sets and Policy Set Bindings editor: Message Level Protection panel" on page 6845.
5. Define which parts of a message are to be encrypted; see "Policy Sets and Policy Set Bindings editor: Message Part Protection panel" on page 6849.
6. Further configure message part encryption; see "Policy Sets and Policy Set Bindings editor: Message Part Policies panel" on page 6856.
7. Further configure the keystore and truststore; see "Policy Sets and Policy Set Bindings editor: Key Information panel" on page 6858.
8. Associate the policy set with a message flow or node; see "Associating policy sets and bindings with message flows and nodes" on page 785.

Configuring XML encryption with Kerberos tickets:

1. Configure your host for Kerberos, providing a `krb.conf` configuration file. This step is required on all operating systems, including Windows.
2. Provide the broker with the Kerberos client credentials for accessing the Kerberos Key Distribution Center (KDC). These credentials (which are required for SOAPRequest nodes) can be provided in the Broker properties tree, or by using the **`mqsisetdbparms`** command. The credentials are taken in order of priority:
   - The node has a security profile with the **`propagation`** property set to *True* and the Properties tree username and password token is present. If no Username and password token exists, an exception in thrown.
   - mqsisetdbparms kerberos::<realm>::<execution group name>
   - mqsisetdbparms kerberos::<realm>
   - mqsisetdbparms kerberos::kerberos
3. In the WebSphere Message Broker Explorer, right-click on the broker with which you want to work, and click **Properties**.
4. In the Properties window, select the **Security** tab, and click **Policy Sets**.
5. Create a policy set and add the required Kerberos token type as Symmetric Tokens; see "Policy Sets and Policy Set Bindings editor: Message Level Protection panel" on page 6845.

Integrity:
About this task

Integrity is provided by XML signature, and requires either X.509 tokens or Kerberos tickets.

Configuring XML signature with X.509 tokens:

1. If you are using the broker's truststore to hold the trusted certificate, you must configure it; see "Viewing and setting keystore and truststore runtime properties at broker level" on page 780 or "Viewing and

setting keystore and truststore runtime properties at execution group level" on page 783 depending on where you want to set keystore and truststore runtime properties.

2. In the WebSphere Message Broker Explorer, right-click on the broker with which you want to work, and click **Properties**.
3. In the Properties window, select the Security tab, and click **Policy Sets**.
4. Create a policy set, enable XML signature, and create signature tokens; see "Policy Sets and Policy Set Bindings editor: Message Level Protection panel" on page 6845.
5. Define which parts of a message are to be signed; see "Policy Sets and Policy Set Bindings editor: Message Part Protection panel" on page 6849.
6. Further configure message part signature; see "Policy Sets and Policy Set Bindings editor: Message Part Policies panel" on page 6856.
7. Further configure the keystore and truststore; see "Policy Sets and Policy Set Bindings editor: Key Information panel" on page 6858.
8. Associate the policy set with a message flow or node; see "Associating policy sets and bindings with message flows and nodes" on page 785.

Configuring XML signature with Kerberos tickets:

1. Configure your host for Kerberos, providing a `krb.conf` configuration file. This step is required on all operating systems, including Windows.
2. Provide the broker with the Kerberos client credentials for accessing the Kerberos Key Distribution Center (KDC). These credentials (which are required for SOAPRequest nodes) can be provided in the Broker properties tree, or by using the **mqsisetdbparms** command. The credentials are taken in the following order of priority:
   - The node has a security profile with the **propagation** property set to *True* and the Properties tree username and password token is present. If no Username and password token exists, an exception in thrown.
   - mqsisetdbparms kerberos::<realm>::<execution group name>
   - mqsisetdbparms kerberos::<realm>
   - mqsisetdbparms kerberos::kerberos
3. In the WebSphere Message Broker Explorer, right-click on the broker with which you want to work, and click **Properties**.
4. In the Properties window, select the **Security** tab, and click **Policy Sets**.
5. Create a policy set and add the required Kerberos token type as Symmetric Tokens; see "Policy Sets and Policy Set Bindings editor: Message Level Protection panel" on page 6845.

Expiration:
Procedure

To configure message expiration, see "Policy Sets and Policy Set Bindings editor: Message Expiration panel" on page 6862.

Related concepts:

"Policy sets" on page 774
Policy sets and bindings define and configure your WS-Security requirements, supported by WebSphere Message Broker, for the SOAPInput, SOAPReply, SOAPRequest, SOAPAsyncRequest, and SOAPAsyncResponse nodes.

"WS-Security capabilities" on page 790
Web service security capabilities are supported by the broker.

Related tasks:

"Viewing and setting keystore and truststore runtime properties at broker level" on page 780
Configure the message broker to refer to a keystore, a truststore, or both, before deploying any message flows that require policy set or bindings for signature, encryption, or X.509 Authentication.

"Viewing and setting keystore and truststore runtime properties at execution group level" on page 783
Configure an execution group to refer to a keystore, a truststore, or both, before deploying any message flows that require policy set or bindings for signature, encryption, or X.509 Authentication.

"Associating policy sets and bindings with message flows and nodes" on page 785
Use the Broker Archive editor to associate policy sets and bindings with message flows and nodes, so that they are available to the broker at run time.

Related reference:

"Policy Sets and Policy Set Bindings editor" on page 6841
The Policy Sets and Policy Set Bindings editor is the default editor for editing, saving, importing, and exporting a policy set or binding. You can define all the WS-Security policies for a single node, or a set of nodes. The policy set can be associated by the administrator with either a message flow or a node.

Policy sets

Policy sets and bindings define and configure your WS-Security requirements, supported by WebSphere Message Broker, for the SOAPInput, SOAPReply, SOAPRequest, SOAPAsyncRequest, and SOAPAsyncResponse nodes.

A *policy set* is a container for the WS-Security policy type.

A policy set *binding* is associated with a policy set and contains information that is specific to the environment and platform, such as information about keys.

Use policy sets and bindings to define the following items for both request and response SOAP messages:

- Authentication for the following tokens:
  - Username tokens (requires a security profile to specify the external security provider)
  - X.509 certificates (requires the broker keystore and truststore, or a security profile to specify the external security provider)
  - SAML assertions, using SAML 1.1 or 2.0 pass-through (requires a security profile to specify the external security provider)
  - LTPA tokens, using LTPA pass-through (requires a security profile to specify the external security provider)
- Asymmetric encryption (confidentiality) using X.509 certificates (requires the broker keystore and truststore)
- Symmetric encryption (confidentiality) using Kerberos tokens (requires the host to be configured for Kerberos)
- Asymmetric signature (integrity) (requires the broker keystore and truststore)

Either the whole SOAP message body, or specific parts of the SOAP message header and body can be encrypted and signed.

You administer policy sets and bindings from WebSphere Message Broker Explorer, which can add, delete, display and edit policy sets and bindings. Any changes to

policy sets or bindings in the toolkit are saved directly to the associated broker. You must stop and then restart the message flow for the new configuration information to take effect.

You can also export and import policy sets and bindings from a broker.

- "Exporting a policy set and policy binding" on page 786
- "Importing a policy set and policy set binding" on page 752

Policy sets and their associated bindings must be saved and restored together.

Policy sets are associated with a message flow, a node or both in the Broker Archive editor. For convenience, you can specify settings for *provider* and *consumer* at the message flow level. The provider setting applies to all SOAPInput and SOAPReply nodes in the message flow. The consumer setting applies to all SOAPRequest, SOAPAsyncRequest, and SOAPAsyncResponse nodes. Individual policy set and binding assignments can be applied at the node level in the Broker Archive editor, and these take precedence over the flow-level provider and consumer settings. The default setting is *none*, meaning that no policy set and bindings are to be used.

Several nodes in the same message flow can refer to the same policy set and bindings. It is the responsibility of the administrator to ensure that the required policy sets are available to the broker at run time. An error is reported if the broker cannot find the associated policy set or bindings.

The rest of this topic describes some of the terms that you will meet when configuring policy sets and bindings.

Default policy set and bindings

When a broker is created, a default policy set and bindings are created called WSS10Default. This default contains a limited security policy which specifies that a Username token is present in request messages (inbound) to SOAPInput nodes in the associated message flow.

The default policy set binding refers to the default WSS10Default policy set. They are not editable.

Consumer and provider nodes

Nodes are either consumers or providers.

Consumer nodes

SOAPRequest
SOAPAsyncRequest
SOAPAsyncResponse

Provider nodes

SOAPInput
SOAPReply

Request and response

Request and response is a message exchange pattern (MEP). It describes a client that sends a SOAP Request message to a Web services server, which in turn sends a Response SOAP message back to the client. The Request message is always the

SOAP message from the client to the server, and the Response message is always the SOAP message reply from server to the client. The following table describes this pattern in relation to the WebSphere Message Broker SOAP nodes:

Node	Broker viewpoint	Request	Response
SOAPInput	SOAP message inbound	Inbound message	Not applicable
SOAPReply	SOAP message outbound	Not applicable	Outbound message
SOAPRequest	SOAP message outbound followed by a SOAP message inbound	Outbound message	Inbound message
SOAPAsyncRequest	SOAP message outbound	Outbound message	Not applicable
SOAPAsyncResponse	SOAP message inbound	Not applicable	Inbound message

Initiator and recipient

Initiator and recipient are roles defined in the exchange of SOAP messages.

Initiator
The role that sends the initial message in a message exchange.

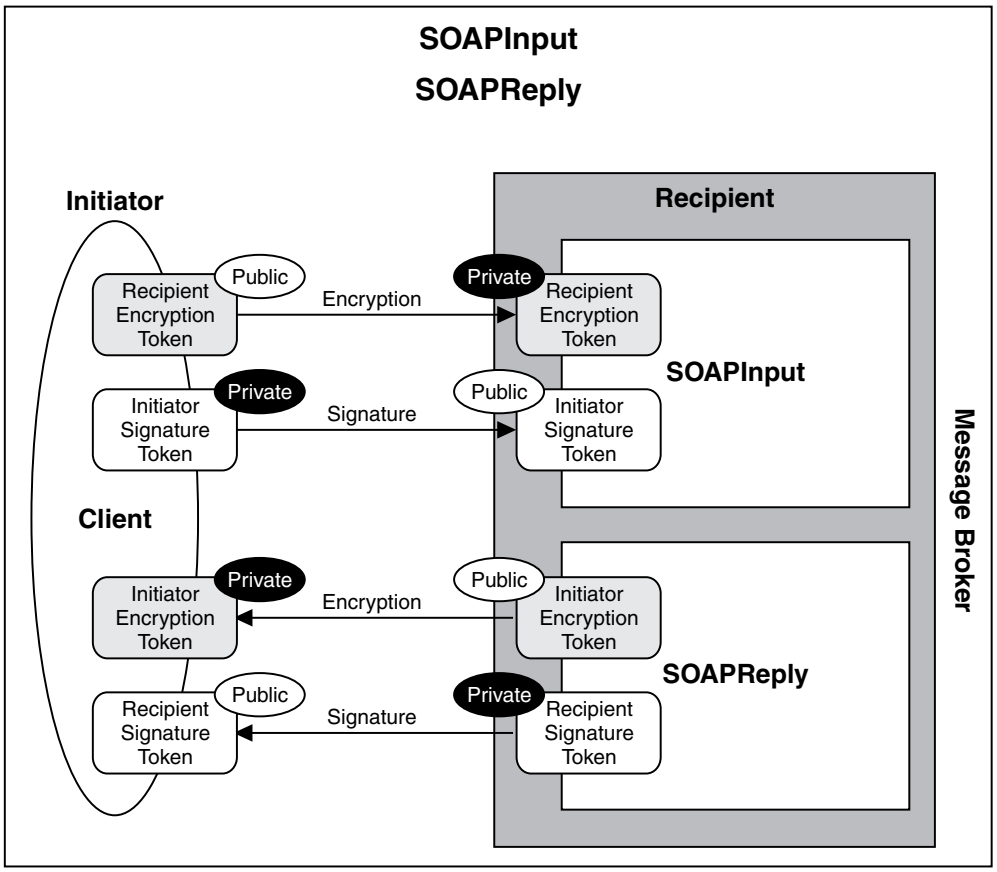
Recipient
The targeted role to process the initial message in a message exchange.

The following table describes these roles in relation to the Message Broker SOAP nodes:

Node	Broker viewpoint	Initiator	Recipient
SOAPInput	SOAP message inbound	External client sending SOAP message to the broker.	SOAPInput node
SOAPReply	SOAP message outbound	External client that sent the original SOAP message to the broker.	SOAPReply node
SOAPRequest (outbound)	SOAP message outbound followed by a SOAP message inbound	SOAPRequest node	External provider receiving the SOAP message
SOAPRequest (inbound)	SOAP message outbound followed by a SOAP message inbound	SOAPRequest node	External provider receiving the SOAP message
SOAPAsyncRequest	SOAP message outbound	SOAPAsyncRequest node	External provider receiving the SOAP message
SOAPAsyncResponse	SOAP message inbound	SOAPAsyncRequest node	External provider receiving the SOAP message

SOAPInput and SOAPReply nodes

In this diagram, the broker acts as recipient. A SOAPInput node receives a message from a client (initiator). A SOAPReply node replies. Inbound and outbound messages are signed and encrypted.



In the request:

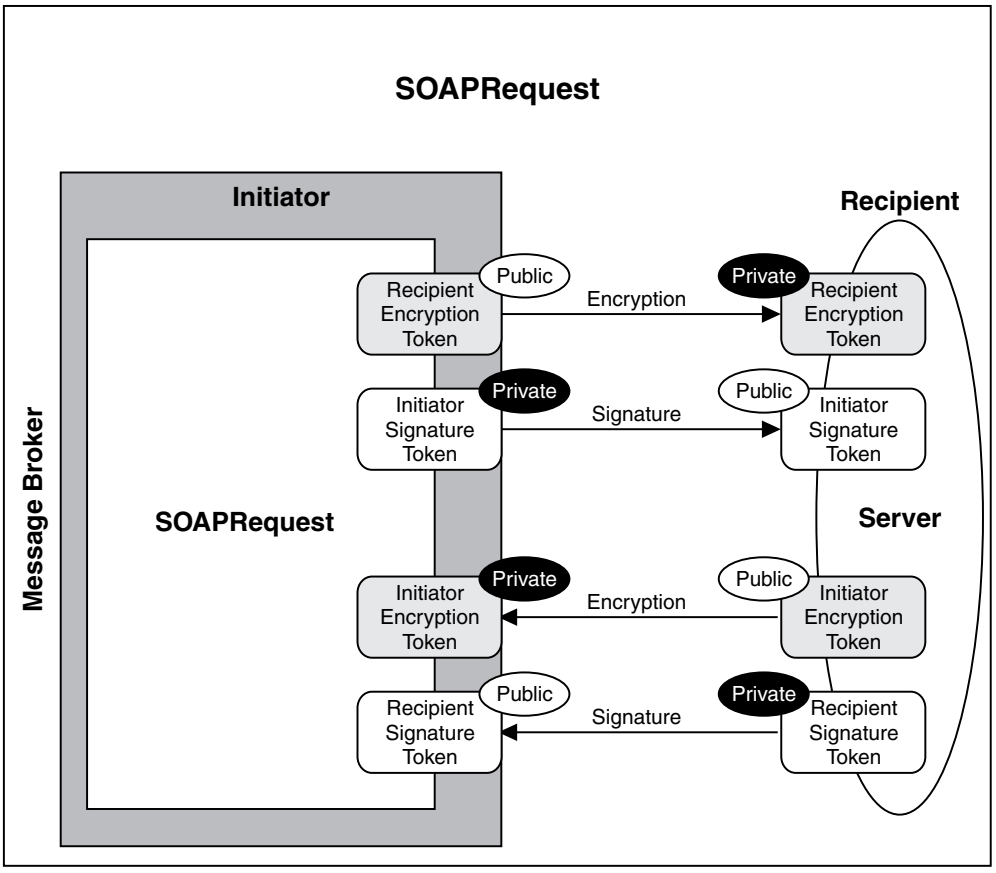
- The initiator uses the broker's public encryption token to encrypt the message, and its own private signature token to sign it.
- The broker uses its own private encryption token to decrypt the message, and the initiator's public signature token to verify the signature.

In the response:

- The broker uses the initiator's public encryption token to encrypt the message, and its own private signature token to sign the message.
- The initiator uses its own private encryption token to decrypt the message, and the broker's public signature token to verify the signature.

SOAPRequest node

This diagram shows the broker acting as an initiator. It uses the SOAPRequest node to make a synchronous request to an external provider (the recipient). Inbound and outbound messages are signed and encrypted. Use of tokens is similar to the example of the asynchronous SOAP nodes, shown earlier.



In the request:

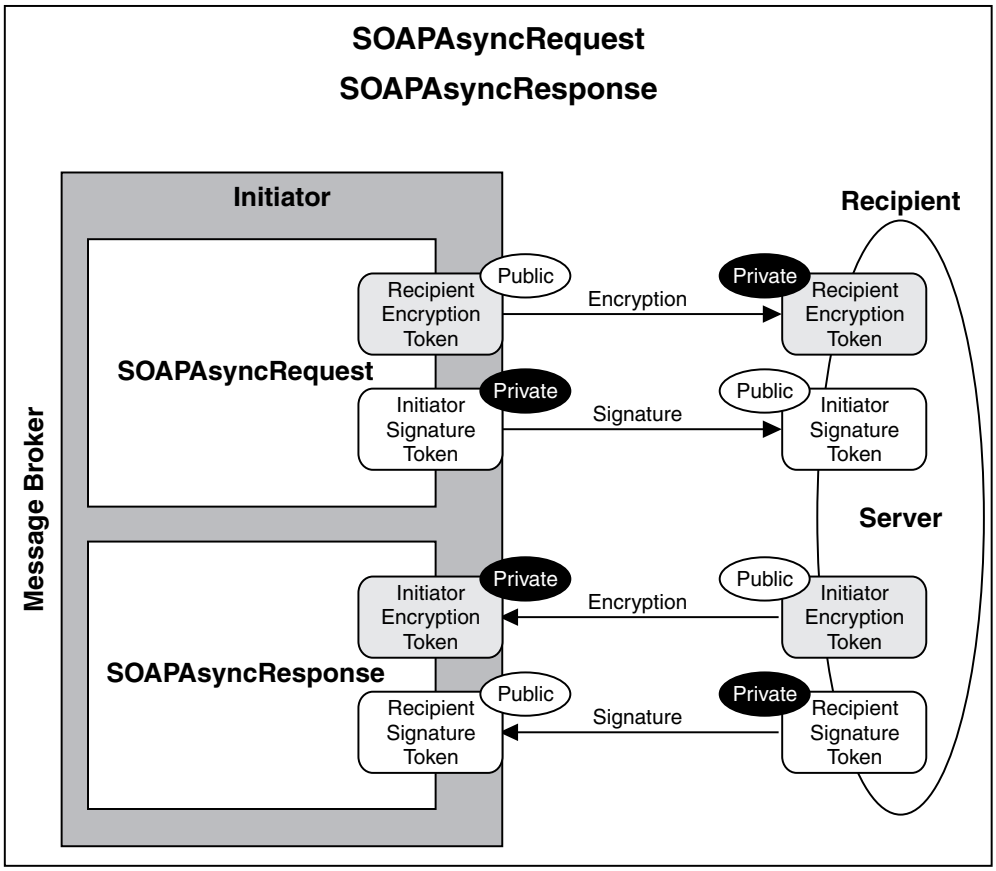
- The broker uses the recipient's public encryption token to encrypt the message, and its own private signature token to sign the message.
- The recipient uses its own private encryption token to decrypt the message, and the broker's public signature token to verify the signature.

In the response:

- The recipient uses the broker's public encryption token to encrypt the message, and its own private signature token to sign the message.
- The broker uses its own private encryption token to decrypt the message, and the initiator's public signature token to verify the signature.

Asynchronous SOAP nodes

This diagram shows the broker acting as an initiator. It uses the asynchronous SOAP nodes to make a request to an external provider (the recipient). Inbound and outbound messages are signed and encrypted.



In the request:

- The broker uses the recipient's public encryption token to encrypt the message, and its own private signature token to sign the message.
- The recipient uses its own private encryption token to decrypt the message, and the broker's public signature token to verify the signature.

In the response:

- The recipient uses the broker's public encryption token to encrypt the message, and its own private signature token to sign the message.
- The broker uses its own private encryption token to decrypt the message, and the initiator's public signature token to verify the signature.

Related concepts:

"WS-Security" on page 765
Web Services Security (WS-Security) describes enhancements to SOAP messaging to provide quality of protection through message integrity, message confidentiality, and single message authentication. WS-Security mechanisms can be used to accommodate a wide variety of security models and encryption technologies.

"WS-Security mechanisms" on page 768
The WS-Security specification provides three mechanisms for securing Web services at the message level: authentication, integrity, and confidentiality.

"Public key cryptography" on page 354
All encryption systems rely on the concept of a key. A key is the basis for a transformation, usually mathematical, of an ordinary message into an unreadable message. For centuries, most encryption systems have relied on private key encryption. Public key encryption is the only challenge to private key encryption that has appeared in the last 30 years.

"Digital certificates" on page 356
Certificates provide a way of authenticating users. Instead of requiring each participant in an application to authenticate every user, third-party authentication relies on the use of digital certificates.

"Digital signatures" on page 360
A digital signature is a number that is attached to a document. For example, in an authentication system that uses public-key encryption, digital signatures are used to sign certificates.

Related tasks:

"Viewing and setting keystore and truststore runtime properties at broker level"
Configure the message broker to refer to a keystore, a truststore, or both, before deploying any message flows that require policy set or bindings for signature, encryption, or X.509 Authentication.

"Viewing and setting keystore and truststore runtime properties at execution group level" on page 783
Configure an execution group to refer to a keystore, a truststore, or both, before deploying any message flows that require policy set or bindings for signature, encryption, or X.509 Authentication.

"Implementing WS-Security" on page 769
Configure authentication, XML encryption, XML signature, and message expiration by using the Policy Sets and Policy Set Bindings editor.

"Associating policy sets and bindings with message flows and nodes" on page 785
Use the Broker Archive editor to associate policy sets and bindings with message flows and nodes, so that they are available to the broker at run time.

"Exporting a policy set and policy binding" on page 786
Use the **mqsireportproperties** command to export a policy set and associated binding to a file.

"Importing a policy set and policy set binding" on page 752
Use the **mqsichangeproperties** command to import a policy set and associated binding.

Related reference:

"Policy Sets and Policy Set Bindings editor" on page 6841
The Policy Sets and Policy Set Bindings editor is the default editor for editing, saving, importing, and exporting a policy set or binding. You can define all the WS-Security policies for a single node, or a set of nodes. The policy set can be associated by the administrator with either a message flow or a node.

Viewing and setting keystore and truststore runtime properties at broker level:

Configure the message broker to refer to a keystore, a truststore, or both, before deploying any message flows that require policy set or bindings for signature, encryption, or X.509 Authentication.

About this task

Keystores and truststores are both keystores. They differ only in the way they are used.

Put all private keys and public key certificates (PKC) in the keystore.

Put all trusted root certificate authority (CA) certificates in the truststore. These certificates are used to establish the trust of any inbound public key certificates.

The only supported type of store is Java keystore (JKS).

Each instance of a broker can be configured to refer to one keystore and one truststore.

The following properties of the broker registry component must be defined correctly for policy sets and bindings:

brokerKeystoreFile
The directory and file location of the keystore.

brokerTruststoreFile
The directory and file location of the truststore.

Listing existing broker registry entries:
About this task

To display all broker registry values, run the command:

```
mqsireportproperties broker_name -o BrokerRegistry -a
```

This returns entries like these:

```
BrokerRegistry=''
  uuid='BrokerRegistry'
  brokerKeystoreType='JKS'
  brokerKeystoreFile=''
  brokerKeystorePass='brokerKeystore::password'
  brokerTruststoreType='JKS'
  brokerTruststoreFile=''
  brokerTruststorePass='brokerTruststore::password'
  httpConnectorPortRange=''
  httpsConnectorPortRange=''
```

Updating the broker reference to a keystore:
About this task

To update the broker reference to a keystore, use the following command:

```
mqsichangeproperties broker_name -o BrokerRegistry
  –n  brokerKeystoreFile
  -v c:\keystore\server.keystore
```

Where `c:\keystore\server.keystore` is the keystore to be referenced.

Updating the broker reference to a truststore:
About this task

To update the broker reference to a truststore, use the following command:

```
mqsichangeproperties broker_name -o BrokerRegistry
  –n  brokerTruststoreFile
  -v c:\truststore\server.truststore
```

Where `c:\truststore\server.truststore` is the truststore to be referenced.

Updating the broker with the keystore password:
About this task

Keystores and truststores normally require passwords for access. Use the **mqsisetdbparms** command to add these passwords to the broker runtime component.

```
mqsisetdbparms broker_name
  -n brokerKeystore::password
  -u temp -p pa55word
```

The user ID, which can be any value, is not required to access the keystore.

Updating the broker with the truststore password:
About this task

To update the broker with the truststore password, use the following command:

```
mqsisetdbparms broker_name
  -n brokerTruststore::password
  -u temp -p pa55word
```

The user ID, which can be any value, is not required to access the keystore.

Updating the broker with a private key password:
About this task

Private keys in the keystore might have their own individual passwords. These can be configured based on the alias name that is specified for the key in the Policy sets and bindings editor. If a key password based on the alias is not found, the keystore password is used. The following command updates the broker with the private key password for the key whose alias is *encKey*.

```
mqsisetdbparms broker_name
  -n brokerTruststore::keypass::encKey
  -u temp -p pa55word
```

The user ID, which can be any value, is not required to access the keystore.

Related concepts:

"Policy sets" on page 774
Policy sets and bindings define and configure your WS-Security requirements, supported by WebSphere Message Broker, for the SOAPInput, SOAPReply, SOAPRequest, SOAPAsyncRequest, and SOAPAsyncResponse nodes.

Related tasks:

"Implementing WS-Security" on page 769
Configure authentication, XML encryption, XML signature, and message expiration by using the Policy Sets and Policy Set Bindings editor.

"Associating policy sets and bindings with message flows and nodes" on page 785
Use the Broker Archive editor to associate policy sets and bindings with message flows and nodes, so that they are available to the broker at run time.

"Viewing and setting keystore and truststore runtime properties at execution group level" on page 783
Configure an execution group to refer to a keystore, a truststore, or both, before deploying any message flows that require policy set or bindings for signature, encryption, or X.509 Authentication.

Related reference:

"Policy Sets and Policy Set Bindings editor" on page 6841
The Policy Sets and Policy Set Bindings editor is the default editor for editing, saving, importing, and exporting a policy set or binding. You can define all the WS-Security policies for a single node, or a set of nodes. The policy set can be associated by the administrator with either a message flow or a node.

Viewing and setting keystore and truststore runtime properties at execution group level:

Configure an execution group to refer to a keystore, a truststore, or both, before deploying any message flows that require policy set or bindings for signature, encryption, or X.509 Authentication.

About this task

An execution group is a named grouping of message flows that have been assigned to a broker. The broker enforces a degree of isolation between message flows in distinct execution groups by ensuring that they run in separate address spaces, or as unique processes. For more information about execution groups, see "Execution groups" on page 53.

Execution group keystore and truststore runtime property values override equivalent property values on the broker, if any are set.

Keystores can contain two kinds of entries: key entries and trusted certificate entries. If a keystore is used to contain trusted certificates, it is typically referred to as a truststore. WebSphere Message Broker can refer to a keystore and a truststore per execution group. When the broker is encrypting or decrypting, it uses entries in its keystore; if the broker is verifying a signature or performing X.509 authentication, it uses entries in its truststore.

The following sample demonstrates the use of viewing and setting keystore and truststore runtime properties at execution group level:

- Address Book

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Displaying execution group level properties:
About this task

To display execution group level properties, run the command:

```
mqsireportproperties broker_name -o ComIbmJVMManager -a -e execution_group
```

Updating the execution group reference to a keystore:
About this task

To update the broker reference to a keystore at an execution group level, use the following command:

```
mqsichangeproperties broker_name -e execution_group -o ComIbmJVMManager
  -n  keystoreFile
  -v c:\keystore\server.keystore,JKS
```

where `c:\keystore\server.keystore,JKS` is a Java keystore (JKS).

Updating the execution group reference to a truststore:
About this task

To update the broker reference to a truststore at an execution group level, use the following command:

```
mqsichangeproperties broker_name -e execution_group -o ComIbmJVMManager
  –n  truststoreFile
  -v c:\truststore\server.truststore
```

where `c:\truststore\server.truststore` is the truststore to be referenced.

Updating the keystore and truststore passwords:
About this task

The commands used to update the keystore and truststore passwords at execution group level are the same as those used when setting keystore and truststore runtime properties at broker level.

- To update the broker with the keystore password; see "Updating the broker with the keystore password" on page 782.
- To update the broker with the truststore password; see "Updating the broker with the truststore password" on page 782.
- To update the broker with a private key password; see "Updating the broker with a private key password" on page 782.

To use the default broker password for the keystore, the keystorePass parameter must be blank, or it must be set to `brokerKeystore::password`. To use a password other than the default broker password, use the following commands:

```
mqsichangeproperties broker_name -e execution_group -o ComIbmJVMManager -n keystorePass
-v execution_group::keystorePass

mqsisetdbparms broker_name -n execution_group::keystorePass -u na -p password
```

To use the default broker password for the truststore, the truststorePass parameter must be blank, or it must be set to `brokerTruststore::password`. To use a password other than the default broker password, use the following commands:

```
mqsichangeproperties broker_name -e execution_group -o ComIbmJVMManager -n truststorePass
-v execution_group::truststorePass

mqsisetdbparms broker_name -n execution_group::truststorePass -u na -p password
```

Adding new certificates to a keystore or truststore:
About this task

If you add new certificates to a keystore or truststore, to ensure that the new certificates are picked up, you must reload the Java virtual machine (JVM). You can reload the JVM by restarting the execution group.

Related concepts:

"Policy sets" on page 774
Policy sets and bindings define and configure your WS-Security requirements, supported by WebSphere Message Broker, for the SOAPInput, SOAPReply, SOAPRequest, SOAPAsyncRequest, and SOAPAsyncResponse nodes.

Related tasks:

"Implementing WS-Security" on page 769
Configure authentication, XML encryption, XML signature, and message expiration by using the Policy Sets and Policy Set Bindings editor.

"Associating policy sets and bindings with message flows and nodes"
Use the Broker Archive editor to associate policy sets and bindings with message flows and nodes, so that they are available to the broker at run time.

"Viewing and setting keystore and truststore runtime properties at broker level" on page 780
Configure the message broker to refer to a keystore, a truststore, or both, before deploying any message flows that require policy set or bindings for signature, encryption, or X.509 Authentication.

"Stopping an execution group using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 944
You can stop all the message flows in an execution group by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer.

"Starting an execution group using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 941
You can start an execution group and all the deployed message flows in an execution group by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer.

Related reference:

"Policy Sets and Policy Set Bindings editor" on page 6841
The Policy Sets and Policy Set Bindings editor is the default editor for editing, saving, importing, and exporting a policy set or binding. You can define all the WS-Security policies for a single node, or a set of nodes. The policy set can be associated by the administrator with either a message flow or a node.

Associating policy sets and bindings with message flows and nodes:

Use the Broker Archive editor to associate policy sets and bindings with message flows and nodes, so that they are available to the broker at run time.

Before you begin

Before you start

Use the Policy Sets and Policy Set Bindings editor to create and configure policy sets and bindings.

About this task

Associations can be made between policy sets and message flow, or specific nodes. Associations made with a flow apply to all nodes described in the Policy Set and Bindings file. Associations at the flow level are defined as being either for consumer or provider nodes.

An association at the node level overrides any association made at the flow level. You do not enter information about consumer or provider for an association at node level.

Procedure

1. In the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer, open the BAR file in the Broker Archive editor.
2. Click the **Manage and Configure** tab.
3. Click the message flow or node that you want to associate with a policy set and binding. The properties that you can configure for the message flow or for the node are displayed in the **Properties** view.

4. If you are configuring a message flow, enter values in the following fields in the **Properties** view, as appropriate:

   Provider Policy Set Bindings

   Provider Policy Set

   Consumer Policy Set Bindings

   Consumer Policy Set

5. If you are configuring a node, enter values in the following fields in the **Properties** view:

   Policy Set

   Policy Set Bindings

Example

What to do next

For new associations to take effect, the BAR file must be redeployed and the message flows stopped and restarted.

Related concepts:

"Policy sets" on page 774
Policy sets and bindings define and configure your WS-Security requirements, supported by WebSphere Message Broker, for the SOAPInput, SOAPReply, SOAPRequest, SOAPAsyncRequest, and SOAPAsyncResponse nodes.

Related tasks:

"Viewing and setting keystore and truststore runtime properties at broker level" on page 780
Configure the message broker to refer to a keystore, a truststore, or both, before deploying any message flows that require policy set or bindings for signature, encryption, or X.509 Authentication.

"Viewing and setting keystore and truststore runtime properties at execution group level" on page 783
Configure an execution group to refer to a keystore, a truststore, or both, before deploying any message flows that require policy set or bindings for signature, encryption, or X.509 Authentication.

"Implementing WS-Security" on page 769
Configure authentication, XML encryption, XML signature, and message expiration by using the Policy Sets and Policy Set Bindings editor.

Related reference:

"Policy Sets and Policy Set Bindings editor" on page 6841
The Policy Sets and Policy Set Bindings editor is the default editor for editing, saving, importing, and exporting a policy set or binding. You can define all the WS-Security policies for a single node, or a set of nodes. The policy set can be associated by the administrator with either a message flow or a node.

Exporting a policy set and policy binding:

Use the **mqsireportproperties** command to export a policy set and associated binding to a file.

About this task

This topic shows how to export policy set *myPolicySet* from broker *myBroker* to a file called *myPolicySet.xml*. The associated binding is *myPolicySetBinding*, which you export to *myPolicySetBinding.xml*.

Procedure

1. Export the policy set to a file:

   ```
   mqsireportproperties myBroker -c PolicySets -o myPolicySet -n ws-security -p myPolicySet.xml
   ```

2. Export the policy set binding to a file:

   ```
   mqsireportproperties myBroker -c PolicySetBindings -o myPolicySetBinding
      -n ws-security -p myPolicySetBinding.xml
   ```

What to do next

Make a note of the policy set that is associated to the binding; you will need this information when you import the policy set and binding. This command displays the policy set associated with a binding:

```
mqsireportproperties myBroker -c PolicySetBindings -o myPolicySetBinding -n associatedPolicySet
```

This displays:

```
PolicySetBindings   myPolicySetBinding     associatedPolicySet='myPolicySet'
BIP8071I: Successful command completion.
```

Related tasks:

"Importing a policy set and policy set binding" on page 752
Use the **mqsichangeproperties** command to import a policy set and associated binding.

Related reference:

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

Importing a policy set and policy set binding:

Use the **mqsichangeproperties** command to import a policy set and associated binding.

About this task

This topic shows how to import policy set *myPolicySet* to broker *myBroker* from a file called *myPolicySet.xml*. The associated binding is *myPolicySetBinding*, which you import from *myPolicySetBinding.xml*.

Procedure

1. Create a configurable service for the policy set, if one does not already exist.

   ```
   mqsicreateconfigurableservice myBroker -c PolicySets -o myPolicySet
   BIP8071I: Successful command completion.
   ```

2. Create a configurable service for the policy set binding, if one does not already exist.

   ```
   mqsicreateconfigurableservice myBroker -c PolicySetBindings -o myPolicySetBinding
   BIP8071I: Successful command completion.
   ```

3. Import the policy set.

   ```
   mqsichangeproperties myBroker -c PolicySets -o myPolicySet -n ws-security -p myPolicySet.xml
   ```

4. Import the policy set binding.

   ```
   mqsichangeproperties myBroker -c PolicySetBindings -o myPolicySetBinding
      -n ws-security -p myPolicySetBinding.xml
   ```

5. Change the value of the associatedPolicySet attribute. Set it to the name of the policy set with which this policy set binding was originally associated.

   ```
   mqsichangeproperties myBroker -c PolicySetBindings -o myPolicySetBinding
      -n associatedPolicySet -v myPolicySet
   ```

Related concepts:

"Policy sets" on page 774
Policy sets and bindings define and configure your WS-Security requirements, supported by WebSphere Message Broker, for the SOAPInput, SOAPReply, SOAPRequest, SOAPAsyncRequest, and SOAPAsyncResponse nodes.

Related tasks:

"Exporting a policy set and policy binding" on page 786
Use the **mqsireportproperties** command to export a policy set and associated binding to a file.

Related reference:

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

Message flow security and security profiles

WebSphere Message Broker provides a security manager for implementing message flow security, so that end-to-end processing of a message through a message flow is secured based on an identity carried in that message instance.

For details of the supported external providers and the operation of the message flow security manager, see "Message flow security overview" on page 383. For information about the token types that are supported by the SOAP nodes and by external security providers, see "Identity" on page 390.

When the message flow is a Web service implemented by using "SOAP nodes" on page 1609 and the identity is to be taken from the "WS-Security" on page 765 SOAP headers, the SOAP nodes are the Policy Enforcement Point (PEP) and the external provider defined by the "Security profiles" on page 387 is the Policy Decision Point (PDP).

The following configuration is required to implement message flow security based on an identity carried in `WS_Security` tokens.

- "Policy sets" on page 774 define the type of tokens used for the identity.
  - To work with a Username and Password identity, configure the policy and binding for Username token "Authentication" on page 770.
  - To work with a X.509 Certificate identity, configure the policy and binding for X.509 certificate token "Authentication" on page 770.

    In the Policy Set Binding, set the X.509 certificate Authentication Token `certificates` mode to `Trust Any`. You set it this way (and not to `Trust Store`) so that the certificate is passed to the security provider defined by the Security Profile. Setting it to `Trust Store` will cause the certificate to

be validated in the local Broker Trust Store. For more details, see "Policy Sets and Policy Set Bindings editor: Authentication and Protection Tokens panel" on page 6854.

  - To work with a SAML assertion token, configure the policy and binding for SAML token "Authentication" on page 770.
- The message flow security operation and external provider are defined by the "Security profiles" on page 387

As an alternative to message flow security and an external PDP, the broker's truststore can be used as a local PDP for X.509 certificate authentication. For WS-Security signing and encryption using only the local broker capability, you must configure the broker's truststore. For details, see "Viewing and setting keystore and truststore runtime properties at broker level" on page 780, or "Viewing and setting keystore and truststore runtime properties at execution group level" on page 783.

Kerberos based WS-Security is supported in the SOAP nodes. When you use Kerberos for security, the SOAP node's WS-Security processing links directly with the host's Kerberos infrastructure. The broker host must be configured for Kerberos, providing a `krb.conf` file to define the Kerberos Key Distribution Center (KDC) and default realm. A Kerberos `keytab` file must also be configured. For more information about configuring Kerberos, see your host's Kerberos documentation.

To work with Kerberos WS-Security in SOAP nodes, create a policy set and bindings specifying Kerberos symmetric encryption tokens on the Message Level Protection panel; see "Policy Sets and Policy Set Bindings editor: Message Level Protection panel" on page 6845. Also configure the required settings on the Kerberos settings panel, as described in "Policy Sets and Policy Set Bindings editor: Kerberos settings panel" on page 6860, and then associate this policy set and bindings with the SOAP node. You can also associate SOAP nodes with a security profile that sets only propagation, so that Kerberos can be used to:

- Extract the service principal as a Username token from SOAP input nodes
- Propagate the Kerberos Key Distribution Center (KDC) credentials as a Username and password to SOAP request nodes.

Related concepts:

"Message flow security overview" on page 383
WebSphere Message Broker provides a security manager, which enables you to control access to individual messages in a message flow, using the identity of the message.

"WS-Security" on page 765
Web Services Security (WS-Security) describes enhancements to SOAP messaging to provide quality of protection through message integrity, message confidentiality, and single message authentication. WS-Security mechanisms can be used to accommodate a wide variety of security models and encryption technologies.

"Security profiles" on page 387
A security profile defines the security operations that are to be performed in a message flow at SecurityPEP nodes and security enabled input and output nodes.

"Policy sets" on page 774
Policy sets and bindings define and configure your WS-Security requirements, supported by WebSphere Message Broker, for the SOAPInput, SOAPReply, SOAPRequest, SOAPAsyncRequest, and SOAPAsyncResponse nodes.

"Username token capabilities" on page 791
This topic describes WS-Security username token capabilities of the broker.

"X.509 certificate token capabilities" on page 795
Various WS-Services Security X.509 certificate token profile standards are supported by WebSphere Message Broker.

Related tasks:

"Setting up message flow security" on page 431
Set up security on a message flow to control access based on the identity of a message passing through the message flow.

"Broker component security" on page 497
You must consider several security aspects when you are setting up brokers running on Windows, Linux, or UNIX platforms.

"Creating a security profile" on page 433
You can create a security profile for use with Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2. Support is also provided for TFIM V6.1, for compatibility with previous versions of WebSphere Message Broker. You can create the security profile by using either the **mqsicreateconfigurableservice** command or an editor in the WebSphere Message Broker Explorer.

"Viewing and setting keystore and truststore runtime properties at broker level" on page 780
Configure the message broker to refer to a keystore, a truststore, or both, before deploying any message flows that require policy set or bindings for signature, encryption, or X.509 Authentication.

Related reference:

"Policy Sets and Policy Set Bindings editor: Authentication and Protection Tokens panel" on page 6854
Use this panel, which is in the Policy Set Bindings section of the editor, to further configure any X.509 authentication tokens defined in the associated policy set.

WS-Security capabilities

Web service security capabilities are supported by the broker.

Web service security mechanisms are defined by OASIS standards. See OASIS Standard for WS-Security Specification

For information about the token profile standards, see:

- OASIS Web Services Security Username Token Profile
- OASIS Web Services Security X.509 Certificate Token Profile
- OASIS Web Services Security Kerberos Token Profile

SAML pass-through support is provided, which enables interoperability with WS-Security SAML profiles, without performing subject confirmation processing. This means that it does not provide validation of the trust relationship between the SAML subject and message content signatures. For information about the SAML token profile standards, see:

- OASIS Web Services Security SAML Token Profile 1.1
- SAML V2 Specification

LTPA pass-through support is also provided, which enables LTPA binary tokens to be passed to an external security token server (STS) for processing.

For more information about using the token profiles, see the following topics:

- "Username token capabilities" on page 791
- "X.509 certificate token capabilities" on page 795

- "SAML token capabilities" on page 804
- "Kerberos token capabilities" on page 809
- "LTPA token capabilities" on page 813

Related tasks:

"Setting up message flow security" on page 431
Set up security on a message flow to control access based on the identity of a message passing through the message flow.

"Broker component security" on page 497
You must consider several security aspects when you are setting up brokers running on Windows, Linux, or UNIX platforms.

OASIS Standard for WS-Security Specification

OASIS Web Services Security Username Token Profile

OASIS Web Services Security X.509 Certificate Token Profile

Username token capabilities:

This topic describes WS-Security username token capabilities of the broker.

For details of using WS-Security username token, see the following capabilities:

- "Username token capabilities for encryption, decryption, signing, and verifying"
- "Username token capabilities for authentication and authorization" on page 792
- "Username token capabilities for identity mapping" on page 793
- "Username token capabilities for extraction and propagation" on page 794

Related tasks:

"Setting up message flow security" on page 431
Set up security on a message flow to control access based on the identity of a message passing through the message flow.

"Broker component security" on page 497
You must consider several security aspects when you are setting up brokers running on Windows, Linux, or UNIX platforms.

Username token capabilities for encryption, decryption, signing, and verifying:

For Web services, you cannot complete encryption, decryption, signing, and verification by using username tokens.

The username token is not applicable, or supported, for the following in any configuration or direction:

- Encryption
- Decryption
- Signing
- Verification

Related concepts:

"WS-Security mechanisms" on page 768
The WS-Security specification provides three mechanisms for securing Web services at the message level: authentication, integrity, and confidentiality.

"Username token capabilities"
This topic describes WS-Security username token capabilities of the broker.

Related reference:

"Username token capabilities for authentication and authorization"
For Web services, you can complete authentication and authorization by using a Web Services Security username token.

"Username token capabilities for identity mapping" on page 793
For Web services, you can map an identity by using a username token.

"Username token capabilities for extraction and propagation" on page 794
This topic describes broker capability for extraction, propagation, or both using a username token in Web services.

Username token capabilities for authentication and authorization:

For Web services, you can complete authentication and authorization by using a Web Services Security username token.

For authentication, the Web Services Security username token must include both the username and the optional password.

The Web Services Security username token "Authentication" on page 768 and "Authorization" on page 401 is supported only in the following configuration:

Capability
- Authenticate
- Authorize

Policy Enforcement Point (PEP) and direction
- In (provider)

  "SOAPInput node" on page 4795

  Configured with a security policy and binding that defines that a Web Services Security username token is present for authentication; see "Authentication" on page 770. You can use the default policy and binding WSS10Default; see "Default policy set and bindings" on page 775.

  Configured with a security profile defining the Policy Decision Point (PDP); see the PDP section that follows.

Trust Store or PDP
- LDAP

  Configured by using an LDAP security profile specifying authentication, authorization, or both; see "Creating a security profile for LDAP" on page 435. For authentication, both a username and password are required.
- WS-Trust v1.3 STS

  Configured by using a WS-Trust v1.3 STS security profile specifying authentication, authorization or both; see "Creating a security profile for WS-Trust V1.3 (TFIM V6.2)" on page 440. For authentication, both a username and password are required.
- TFIM V6.1

  Configured by using a TFIM security profile specifying authentication, authorization or both; see "Creating a security profile for TFIM V6.1" on page 444. For authentication, both a username and password are required.

Related concepts:

"WS-Security mechanisms" on page 768
The WS-Security specification provides three mechanisms for securing Web services at the message level: authentication, integrity, and confidentiality.

"Username token capabilities" on page 791
This topic describes WS-Security username token capabilities of the broker.

"Authorization" on page 401
Authorization is the process of verifying that an identity token has permission to access a message flow.

Related reference:

"Username token capabilities for encryption, decryption, signing, and verifying" on page 791
For Web services, you cannot complete encryption, decryption, signing, and verification by using username tokens.

"Username token capabilities for identity mapping"
For Web services, you can map an identity by using a username token.

"Username token capabilities for extraction and propagation" on page 794
This topic describes broker capability for extraction, propagation, or both using a username token in Web services.

Username token capabilities for identity mapping:

For Web services, you can map an identity by using a username token.

"Identity mapping" on page 403 from a username identity token to a mapped username identity token is supported only in the following configurations:

Capability

- Identity mapping

Policy Enforcement Point (PEP) and direction

- In (provider)

  "SOAPInput node" on page 4795

  Configured with a security policy and binding that defines that a username taken is present. You can use the default policy and binding WSS10Default; see "Default policy set and bindings" on page 775.

  Configured with a security profile defining the external Policy Decision Point (PDP); see the PDP section that follows.

Trust store or PDP

- WS-Trust v1.3 STS

  Configured by using a WS-Trust v1.3 STS security profile that specifies identity mapping; see "Creating a security profile for WS-Trust V1.3 (TFIM V6.2)" on page 440.

- TFIM V6.1

  Configured by using a TFIM security profile that specifies identity mapping; see "Creating a security profile for TFIM V6.1" on page 444.

Related concepts:

"WS-Security mechanisms" on page 768
The WS-Security specification provides three mechanisms for securing Web services at the message level: authentication, integrity, and confidentiality.

"Username token capabilities" on page 791
This topic describes WS-Security username token capabilities of the broker.

"Identity mapping" on page 403
Identity mapping is the transformation of a security token from one format to another format, or the federation of an identity from one realm to an equivalent

identity in another realm.

"Identity" on page 390
In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

Related reference:

"Username token capabilities for encryption, decryption, signing, and verifying" on page 791
For Web services, you cannot complete encryption, decryption, signing, and verification by using username tokens.

"Username token capabilities for authentication and authorization" on page 792
For Web services, you can complete authentication and authorization by using a Web Services Security username token.

"Username token capabilities for extraction and propagation"
This topic describes broker capability for extraction, propagation, or both using a username token in Web services.

Username token capabilities for extraction and propagation:

This topic describes broker capability for extraction, propagation, or both using a username token in Web services.

The extraction of username into the Properties folder source "Identity" on page 390 fields, is supported in the following configurations:

Capability

- Extraction

Policy Enforcement Point (PEP) and direction

- In (provider)

  "SOAPInput node" on page 4795

  Configured with a security policy and binding which defines that a username taken is present. You can use the default policy and binding WSS10Default; see "Default policy set and bindings" on page 775.

  Configured with a security profile that defines propagation; see "Creating a security profile" on page 433

The propagation of a username token into the SOAP WS-Security header, from the token present in either the mapped or the source identity fields in the properties folder, is supported in the following configuration. See "Identity" on page 390.

Capability

- Propagate

Policy Enforcement Point (PEP) and direction

- Out (consumer)

  "SOAPRequest node" on page 4828

  "SOAPAsyncRequest node" on page 4750

  Configured with a security profile that defines propagation; for example, Default Propagation. See "Security profiles" on page 387

Related concepts:

"WS-Security mechanisms" on page 768
The WS-Security specification provides three mechanisms for securing Web

services at the message level: authentication, integrity, and confidentiality.

"Username token capabilities" on page 791
This topic describes WS-Security username token capabilities of the broker.

"Identity" on page 390
In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

"Identity and security token propagation" on page 426
Identity and security token propagation enables the identity and security tokens (associated with each message) to be propagated throughout a message flow, and on to target applications through output or request nodes.

Related tasks:

"Configuring the extraction of an identity or security token" on page 447
You can configure the SecurityPEP node or security enabled input nodes to extract the identity or security token from a message and store it in the properties tree identity fields, enabling it to be processed throughout the message flow and propagated at output or request nodes.

Related reference:

"Username token capabilities for encryption, decryption, signing, and verifying" on page 791
For Web services, you cannot complete encryption, decryption, signing, and verification by using username tokens.

"Username token capabilities for authentication and authorization" on page 792
For Web services, you can complete authentication and authorization by using a Web Services Security username token.

"Username token capabilities for identity mapping" on page 793
For Web services, you can map an identity by using a username token.

X.509 certificate token capabilities:

Various WS-Services Security X.509 certificate token profile standards are supported by WebSphere Message Broker.

For details of using the X.509 certificates, see the following capabilities:

- "X.509 certificate token capabilities for encryption" on page 796
- "X.509 certificate token capabilities for decryption" on page 797
- "X.509 certificate token capabilities for signing" on page 798
- "X.509 certificate token capabilities for verifying" on page 799
- "X.509 certificate token capabilities for authentication" on page 800
- "X.509 certificate token capabilities for authorization" on page 801
- "X.509 certificate token capabilities for identity mapping" on page 802
- "X.509 certificate token capabilities for extraction and propagation" on page 803

Related tasks:

"Setting up message flow security" on page 431
Set up security on a message flow to control access based on the identity of a message passing through the message flow.

"Broker component security" on page 497
You must consider several security aspects when you are setting up brokers running on Windows, Linux, or UNIX platforms.

X.509 certificate token capabilities for encryption:

For Web services, you can complete encryption by using an X.509 certificate token.

X.509 certificate token encryption for providing message "Confidentiality" on page 769 on outgoing SOAP messages from the broker is supported in the following configurations:

Capability

- Encrypt (by using a partner public key)

Policy Enforcement Point (PEP) and direction

- Out (consumer)

  "SOAPRequest node" on page 4828

  "SOAPAsyncRequest node" on page 4750
- Out (provider)

  "SOAPReply node" on page 4819

  Configured with a policy set and binding defining the message "Confidentiality" on page 771.

Trust Store or Policy Decision Point (PDP)

- Broker Truststore; for more details, see "Viewing and setting keystore and truststore runtime properties at broker level" on page 780.

Encryption is not supported with external PDPs such as TFIM or LDAP.

Related concepts:

"WS-Security mechanisms" on page 768
The WS-Security specification provides three mechanisms for securing Web services at the message level: authentication, integrity, and confidentiality.

"X.509 certificate token capabilities" on page 795
Various WS-Services Security X.509 certificate token profile standards are supported by WebSphere Message Broker.

Related tasks:

"Implementing WS-Security" on page 769
Configure authentication, XML encryption, XML signature, and message expiration by using the Policy Sets and Policy Set Bindings editor.

Related reference:

"X.509 certificate token capabilities for decryption" on page 797
For Web services, you can complete decryption by using an X.509 certificate token.

"X.509 certificate token capabilities for signing" on page 798
For Web services, you can use an X.509 certificate token for signing.

"X.509 certificate token capabilities for verifying" on page 799
For Web services, you can verify a signing by using an X.509 certificate token profile.

"X.509 certificate token capabilities for authentication" on page 800
For Web services, you can complete authentication by using an X.509 certificate token.

"X.509 certificate token capabilities for authorization" on page 801
The X.509 certificate token is not supported for authorization in any configuration or direction.

"X.509 certificate token capabilities for identity mapping" on page 802
For Web services, you can map an identity by using an X.509 certificate token.

"X.509 certificate token capabilities for extraction and propagation" on page 803
This topic describes broker Web services capability for extraction and propagation X.509 certificate token.

X.509 certificate token capabilities for decryption:

For Web services, you can complete decryption by using an X.509 certificate token.

X.509 certificate token decryption for incoming SOAP message "Confidentiality" on page 769 is supported in the following configurations:

Capability

- Decrypt (by using a broker private key)

Policy Enforcement Point (PEP) and direction.

- In (provider)

  "SOAPInput node" on page 4795
- In (consumer)

  "SOAPRequest node" on page 4828

  "SOAPAsyncResponse node" on page 4777

  Configured with a policy set and binding defining the message "Confidentiality" on page 771.

Trust Store or Policy Decision Point (PDP).

- Broker Truststore; for details, see "Viewing and setting keystore and truststore runtime properties at broker level" on page 780.

Decryption is not supported with external PDPs such as TFIM or LDAP.

Related concepts:

"WS-Security mechanisms" on page 768
The WS-Security specification provides three mechanisms for securing Web services at the message level: authentication, integrity, and confidentiality.

"X.509 certificate token capabilities" on page 795
Various WS-Services Security X.509 certificate token profile standards are supported by WebSphere Message Broker.

Related tasks:

"Implementing WS-Security" on page 769
Configure authentication, XML encryption, XML signature, and message expiration by using the Policy Sets and Policy Set Bindings editor.

Related reference:

"X.509 certificate token capabilities for encryption" on page 796
For Web services, you can complete encryption by using an X.509 certificate token.

"X.509 certificate token capabilities for signing" on page 798
For Web services, you can use an X.509 certificate token for signing.

"X.509 certificate token capabilities for verifying" on page 799
For Web services, you can verify a signing by using an X.509 certificate token profile.

"X.509 certificate token capabilities for authentication" on page 800
For Web services, you can complete authentication by using an X.509 certificate token.

"X.509 certificate token capabilities for authorization" on page 801
The X.509 certificate token is not supported for authorization in any configuration or direction.

"X.509 certificate token capabilities for identity mapping" on page 802
For Web services, you can map an identity by using an X.509 certificate token.

"X.509 certificate token capabilities for extraction and propagation" on page 803
This topic describes broker Web services capability for extraction and propagation X.509 certificate token.

X.509 certificate token capabilities for signing:

For Web services, you can use an X.509 certificate token for signing.

X.509 certificate token signing for outgoing SOAP message "Integrity" on page 769 is supported in the following configurations:

Capability

- Sign (by using a broker private key)

Policy Enforcement Point (PEP) and direction

- Out (consumer)

  "SOAPRequest node" on page 4828

  "SOAPAsyncRequest node" on page 4750
- Out (provider)

  "SOAPReply node" on page 4819

  Configured with a policy set and binding defining the message "Integrity" on page 772.

Trust Store or Policy Decision Point (PDP)

- Broker Truststore; for details, see "Viewing and setting keystore and truststore runtime properties at broker level" on page 780.

Signing is not supported with an external PDP such as TFIM or LDAP.

Related concepts:

"WS-Security mechanisms" on page 768
The WS-Security specification provides three mechanisms for securing Web services at the message level: authentication, integrity, and confidentiality.

"X.509 certificate token capabilities" on page 795
Various WS-Services Security X.509 certificate token profile standards are supported by WebSphere Message Broker.

Related tasks:

"Implementing WS-Security" on page 769
Configure authentication, XML encryption, XML signature, and message expiration by using the Policy Sets and Policy Set Bindings editor.

Related reference:

"X.509 certificate token capabilities for encryption" on page 796
For Web services, you can complete encryption by using an X.509 certificate token.

"X.509 certificate token capabilities for decryption" on page 797
For Web services, you can complete decryption by using an X.509 certificate token.

"X.509 certificate token capabilities for verifying"
For Web services, you can verify a signing by using an X.509 certificate token profile.

"X.509 certificate token capabilities for authentication" on page 800
For Web services, you can complete authentication by using an X.509 certificate token.

"X.509 certificate token capabilities for authorization" on page 801
The X.509 certificate token is not supported for authorization in any configuration or direction.

"X.509 certificate token capabilities for identity mapping" on page 802
For Web services, you can map an identity by using an X.509 certificate token.

"X.509 certificate token capabilities for extraction and propagation" on page 803
This topic describes broker Web services capability for extraction and propagation X.509 certificate token.

X.509 certificate token capabilities for verifying:

For Web services, you can verify a signing by using an X.509 certificate token profile.

X.509 certificate token verification of the "Integrity" on page 769 of a signed incoming SOAP message is supported in the following configurations:

Capability

- Verify signature (by using a partner public key)

Policy Enforcement Point (PEP) and direction

- In (provider)

  "SOAPInput node" on page 4795
- In (consumer)

  "SOAPRequest node" on page 4828

  "SOAPAsyncResponse node" on page 4777

  Configured with a policy set and binding defining the message "Integrity" on page 772

Trust Store or Policy Decision Point (PDP)

- Broker Trust store; for details, see "Viewing and setting keystore and truststore runtime properties at broker level" on page 780.

  Signature verification is not supported with an external PDP, such as TFIM or LDAP.

Related concepts:

"WS-Security mechanisms" on page 768
The WS-Security specification provides three mechanisms for securing Web services at the message level: authentication, integrity, and confidentiality.

"X.509 certificate token capabilities" on page 795
Various WS-Services Security X.509 certificate token profile standards are supported by WebSphere Message Broker.

Related tasks:

"Implementing WS-Security" on page 769
Configure authentication, XML encryption, XML signature, and message expiration by using the Policy Sets and Policy Set Bindings editor.

Related reference:

"X.509 certificate token capabilities for encryption" on page 796
For Web services, you can complete encryption by using an X.509 certificate token.

"X.509 certificate token capabilities for decryption" on page 797
For Web services, you can complete decryption by using an X.509 certificate token.

"X.509 certificate token capabilities for signing" on page 798
For Web services, you can use an X.509 certificate token for signing.

"X.509 certificate token capabilities for authentication"
For Web services, you can complete authentication by using an X.509 certificate token.

"X.509 certificate token capabilities for authorization" on page 801
The X.509 certificate token is not supported for authorization in any configuration or direction.

"X.509 certificate token capabilities for identity mapping" on page 802
For Web services, you can map an identity by using an X.509 certificate token.

"X.509 certificate token capabilities for extraction and propagation" on page 803
This topic describes broker Web services capability for extraction and propagation X.509 certificate token.

X.509 certificate token capabilities for authentication:

For Web services, you can complete authentication by using an X.509 certificate token.

The X.509 certificate token "Authentication" on page 768 of an incoming SOAP message is supported in the following configurations:

Capability

- Authenticate

Policy Enforcement Point (PEP) and direction

- In (provider)

  "SOAPInput node" on page 4795

  Configured with a policy set and binding defining the certificate "Authentication" on page 770.

  Optionally configured with a security profile defining an external Policy Decision Point (PDP); see the PDP section that follows.

Trust Store or PDP

- Broker Trust store; for details, see "Viewing and setting keystore and truststore runtime properties at broker level" on page 780.
- WS-Trust v1.3 STS

  Configured by using a WS-Trust v1.3 STS security profile specifying authentication; see "Creating a security profile for WS-Trust V1.3 (TFIM V6.2)" on page 440.
- TFIM V6.1

Configured by using a TFIM security profile specifying authentication; for details, see “Creating a security profile for TFIM V6.1” on page 444.

Certificate authentication with an external LDAP PDP is not supported.

Related concepts:

“WS-Security mechanisms” on page 768
The WS-Security specification provides three mechanisms for securing Web services at the message level: authentication, integrity, and confidentiality.

“X.509 certificate token capabilities” on page 795
Various WS-Services Security X.509 certificate token profile standards are supported by WebSphere Message Broker.

Related tasks:

“Implementing WS-Security” on page 769
Configure authentication, XML encryption, XML signature, and message expiration by using the Policy Sets and Policy Set Bindings editor.

Related reference:

“X.509 certificate token capabilities for encryption” on page 796
For Web services, you can complete encryption by using an X.509 certificate token.

“X.509 certificate token capabilities for decryption” on page 797
For Web services, you can complete decryption by using an X.509 certificate token.

“X.509 certificate token capabilities for signing” on page 798
For Web services, you can use an X.509 certificate token for signing.

“X.509 certificate token capabilities for verifying” on page 799
For Web services, you can verify a signing by using an X.509 certificate token profile.

“X.509 certificate token capabilities for authorization”
The X.509 certificate token is not supported for authorization in any configuration or direction.

“X.509 certificate token capabilities for identity mapping” on page 802
For Web services, you can map an identity by using an X.509 certificate token.

“X.509 certificate token capabilities for extraction and propagation” on page 803
This topic describes broker Web services capability for extraction and propagation X.509 certificate token.

X.509 certificate token capabilities for authorization:

The X.509 certificate token is not supported for authorization in any configuration or direction.

Related concepts:

“WS-Security mechanisms” on page 768
The WS-Security specification provides three mechanisms for securing Web services at the message level: authentication, integrity, and confidentiality.

“Authorization” on page 401
Authorization is the process of verifying that an identity token has permission to access a message flow.

“X.509 certificate token capabilities” on page 795
Various WS-Services Security X.509 certificate token profile standards are supported by WebSphere Message Broker.

Related tasks:

"Implementing WS-Security" on page 769
Configure authentication, XML encryption, XML signature, and message expiration by using the Policy Sets and Policy Set Bindings editor.

Related reference:

"X.509 certificate token capabilities for encryption" on page 796
For Web services, you can complete encryption by using an X.509 certificate token.

"X.509 certificate token capabilities for decryption" on page 797
For Web services, you can complete decryption by using an X.509 certificate token.

"X.509 certificate token capabilities for signing" on page 798
For Web services, you can use an X.509 certificate token for signing.

"X.509 certificate token capabilities for verifying" on page 799
For Web services, you can verify a signing by using an X.509 certificate token profile.

"X.509 certificate token capabilities for authentication" on page 800
For Web services, you can complete authentication by using an X.509 certificate token.

"X.509 certificate token capabilities for identity mapping"
For Web services, you can map an identity by using an X.509 certificate token.

"X.509 certificate token capabilities for extraction and propagation" on page 803
This topic describes broker Web services capability for extraction and propagation X.509 certificate token.

X.509 certificate token capabilities for identity mapping:

For Web services, you can map an identity by using an X.509 certificate token.

The broker supports "Identity mapping" on page 403 from an X.509 certificate token in an incoming SOAP message header to username tokens in the following configurations:

Capability

- Identity mapping

Policy Enforcement Point (PEP) and direction

- In (provider)

  "SOAPInput node" on page 4795

  Configured with a policy set and binding defining the certificate "Authentication" on page 770.

  Configured with a security profile defining an external Policy Decision Point (PDP); see the PDP section that follows.

Trust Store or PDP

- TFIM V6.1

  Configured by using a TFIM security profile specifying identity mapping; for details, see "Creating a security profile for TFIM V6.1" on page 444.

- WS-Trust v1.3 STS (TFIM V6.2)

  Configured by using a WS-Trust v1.3 STS security profile specifying identity mapping; for details, see "Creating a security profile for WS-Trust V1.3 (TFIM V6.2)" on page 440.

Identity mapping is not supported with LDAP, or at outbound nodes.

Username tokens only can be propagated.

Related concepts:

"WS-Security mechanisms" on page 768
The WS-Security specification provides three mechanisms for securing Web services at the message level: authentication, integrity, and confidentiality.

"Identity mapping" on page 403
Identity mapping is the transformation of a security token from one format to another format, or the federation of an identity from one realm to an equivalent identity in another realm.

"X.509 certificate token capabilities" on page 795
Various WS-Services Security X.509 certificate token profile standards are supported by WebSphere Message Broker.

Related tasks:

"Implementing WS-Security" on page 769
Configure authentication, XML encryption, XML signature, and message expiration by using the Policy Sets and Policy Set Bindings editor.

Related reference:

"X.509 certificate token capabilities for encryption" on page 796
For Web services, you can complete encryption by using an X.509 certificate token.

"X.509 certificate token capabilities for decryption" on page 797
For Web services, you can complete decryption by using an X.509 certificate token.

"X.509 certificate token capabilities for signing" on page 798
For Web services, you can use an X.509 certificate token for signing.

"X.509 certificate token capabilities for verifying" on page 799
For Web services, you can verify a signing by using an X.509 certificate token profile.

"X.509 certificate token capabilities for authentication" on page 800
For Web services, you can complete authentication by using an X.509 certificate token.

"X.509 certificate token capabilities for authorization" on page 801
The X.509 certificate token is not supported for authorization in any configuration or direction.

"X.509 certificate token capabilities for extraction and propagation"
This topic describes broker Web services capability for extraction and propagation X.509 certificate token.

X.509 certificate token capabilities for extraction and propagation:

This topic describes broker Web services capability for extraction and propagation X.509 certificate token.

The broker does not support propagation of an X.509 certificate.

The X.509 certificate token extraction is supported in the following configurations:

Capability

- Extraction

Policy Enforcement Point (PEP) and direction

- In (provider)

  "SOAPInput node" on page 4795

  Configured with a policy set and binding defining the X.509 certificate is present; see "Implementing WS-Security" on page 769.

  Configured with a security profile defining propagation; see the "Security profiles" on page 387.

Related concepts:

"WS-Security mechanisms" on page 768
The WS-Security specification provides three mechanisms for securing Web services at the message level: authentication, integrity, and confidentiality.

"Identity" on page 390
In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

"Identity and security token propagation" on page 426
Identity and security token propagation enables the identity and security tokens (associated with each message) to be propagated throughout a message flow, and on to target applications through output or request nodes.

"X.509 certificate token capabilities" on page 795
Various WS-Services Security X.509 certificate token profile standards are supported by WebSphere Message Broker.

Related tasks:

"Configuring the extraction of an identity or security token" on page 447
You can configure the SecurityPEP node or security enabled input nodes to extract the identity or security token from a message and store it in the properties tree identity fields, enabling it to be processed throughout the message flow and propagated at output or request nodes.

Related reference:

"X.509 certificate token capabilities for encryption" on page 796
For Web services, you can complete encryption by using an X.509 certificate token.

"X.509 certificate token capabilities for decryption" on page 797
For Web services, you can complete decryption by using an X.509 certificate token.

"X.509 certificate token capabilities for signing" on page 798
For Web services, you can use an X.509 certificate token for signing.

"X.509 certificate token capabilities for verifying" on page 799
For Web services, you can verify a signing by using an X.509 certificate token profile.

"X.509 certificate token capabilities for authentication" on page 800
For Web services, you can complete authentication by using an X.509 certificate token.

"X.509 certificate token capabilities for authorization" on page 801
The X.509 certificate token is not supported for authorization in any configuration or direction.

"X.509 certificate token capabilities for identity mapping" on page 802
For Web services, you can map an identity by using an X.509 certificate token.

SAML token capabilities:

This topic describes WS-Security SAML token capabilities of the broker.

The broker provides SAML pass-through support, which means that the mechanisms for subject confirmation are not enforced. The SAML token is extracted and passed to an external security token service (STS) for validation. The STS to be used is specified on a security profile.

For information about using WS-Security SAML tokens, see the following topics:

- "SAML token capabilities for encryption, decryption, signing, and verifying"
- "SAML token capabilities for authentication and authorization" on page 806
- "SAML token capabilities for identity mapping" on page 807
- "SAML token capabilities for extraction and propagation" on page 808

Related concepts:

"WS-Security mechanisms" on page 768
The WS-Security specification provides three mechanisms for securing Web services at the message level: authentication, integrity, and confidentiality.

"Identity" on page 390
In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

Related tasks:

"Setting up message flow security" on page 431
Set up security on a message flow to control access based on the identity of a message passing through the message flow.

"Broker component security" on page 497
You must consider several security aspects when you are setting up brokers running on Windows, Linux, or UNIX platforms.

SAML token capabilities for encryption, decryption, signing, and verifying:

The broker provides SAML pass-through support, which means that the encryption, decryption, signing, and verifying mechanisms for achieving SAML subject confirmation are not enforced by the broker.

Related concepts:

"WS-Security mechanisms" on page 768
The WS-Security specification provides three mechanisms for securing Web services at the message level: authentication, integrity, and confidentiality.

"SAML token capabilities" on page 804
This topic describes WS-Security SAML token capabilities of the broker.

"Identity" on page 390
In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

Related reference:

"SAML token capabilities for authentication and authorization" on page 806
For Web services, you can complete authentication and authorization using a SAML token.

"SAML token capabilities for identity mapping" on page 807
This topic describes the broker Web services capability for identity mapping using a SAML token.

"SAML token capabilities for extraction and propagation" on page 808
This topic describes broker capability for extraction, propagation, or both using a SAML token in Web services.

SAML token capabilities for authentication and authorization:

For Web services, you can complete authentication and authorization using a SAML token.

The SAML token "Authentication" on page 768 and "Authorization" on page 401 are supported only in the following configuration:

Capability

- Authenticate
- Authorize

Policy Enforcement Point (PEP) and direction

- In (provider)

  "SOAPInput node" on page 4795

  Configured with a security policy set and binding that defines that a SAML pass-through 1.1 or SAML pass-through 2.0 token is present for authentication; see "Authentication" on page 770. The broker provides only SAML pass-through support, which means that the SAML token is extracted and passed to an external security token service (STS) for validation. The STS to be used is specified in a security profile. The STS processing can be used to implement authentication based on the SAML principal, and authorization based on SAML attributes.

  Configured with a security profile defining the Policy Decision Point (PDP); see the PDP section that follows.

Trust Store or PDP

- WS-Trust v1.3 STS

  Configured by using a WS-Trust v1.3 STS security profile specifying authentication, authorization or both; see "Creating a security profile for WS-Trust V1.3 (TFIM V6.2)" on page 440.

Related concepts:

"WS-Security mechanisms" on page 768
The WS-Security specification provides three mechanisms for securing Web services at the message level: authentication, integrity, and confidentiality.

"SAML token capabilities" on page 804
This topic describes WS-Security SAML token capabilities of the broker.

"Authorization" on page 401
Authorization is the process of verifying that an identity token has permission to access a message flow.

"Authentication and validation" on page 398
Authentication is the process of establishing the identity of a user or system and verifying that the identity is valid. Applying authentication to a SAML security token involves validating the assertions that it carries and confirming that it is being processed within its validity period.

"Identity" on page 390
In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

Related reference:

"SAML token capabilities for encryption, decryption, signing, and verifying" on page 805
The broker provides SAML pass-through support, which means that the encryption, decryption, signing, and verifying mechanisms for achieving SAML subject confirmation are not enforced by the broker.

"SAML token capabilities for identity mapping"
This topic describes the broker Web services capability for identity mapping using a SAML token.

"SAML token capabilities for extraction and propagation" on page 808
This topic describes broker capability for extraction, propagation, or both using a SAML token in Web services.

SAML token capabilities for identity mapping:

This topic describes the broker Web services capability for identity mapping using a SAML token.

"Identity mapping" on page 403 from a SAML identity token to a mapped SAML identity token is supported only in the following configurations:

Capability

- Identity mapping

Policy Enforcement Point (PEP) and direction

- In (provider)

  "SOAPInput node" on page 4795

  Configured with a security policy set and bindings that specifies a SAML pass-though 1.1 or SAML pass-through 2.0 authentication token.

  Configured with a security profile defining the external Policy Decision Point (PDP); see the PDP section that follows.

Trust store or PDP

- WS-Trust v1.3 STS

  Configured by using a WS-Trust v1.3 STS security profile that specifies identity mapping; see "Creating a security profile for WS-Trust V1.3 (TFIM V6.2)" on page 440.

Related concepts:

"WS-Security mechanisms" on page 768
The WS-Security specification provides three mechanisms for securing Web services at the message level: authentication, integrity, and confidentiality.

"SAML token capabilities" on page 804
This topic describes WS-Security SAML token capabilities of the broker.

"Identity mapping" on page 403
Identity mapping is the transformation of a security token from one format to another format, or the federation of an identity from one realm to an equivalent identity in another realm.

"Identity" on page 390
In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

Related reference:

"SAML token capabilities for encryption, decryption, signing, and verifying" on page 805
The broker provides SAML pass-through support, which means that the encryption, decryption, signing, and verifying mechanisms for achieving SAML subject confirmation are not enforced by the broker.

"SAML token capabilities for authentication and authorization" on page 806
For Web services, you can complete authentication and authorization using a SAML token.

"SAML token capabilities for extraction and propagation"
This topic describes broker capability for extraction, propagation, or both using a SAML token in Web services.

SAML token capabilities for extraction and propagation:

This topic describes broker capability for extraction, propagation, or both using a SAML token in Web services.

The extraction of a SAML token into the Properties folder source "Identity" on page 390 fields, is supported in the following configurations:

Capability

- Extraction

Policy Enforcement Point (PEP) and direction

- In (provider)

  "SOAPInput node" on page 4795

  Configured with a security policy set and bindings that specifies a SAML pass-though 1.1 or SAML pass-through 2.0 authentication token.

  Configured with a security profile that defines propagation; see "Creating a security profile" on page 433

The propagation of a SAML token into the SOAP WS-Security header, from the token present in either the mapped or the source identity fields in the properties folder, is supported in the following configuration. For more information, see "Identity" on page 390.

Capability

- Propagate

Policy Enforcement Point (PEP) and direction

- Out (consumer)

  "SOAPRequest node" on page 4828

  "SOAPAsyncRequest node" on page 4750

  Configured with a security profile that defines propagation. For more information, see "Security profiles" on page 387.

  Configured with a security policy set and bindings that specifies SAML pass-though 1.1 or SAML pass-through 2.0 authentication token.

Related concepts:

"WS-Security mechanisms" on page 768
The WS-Security specification provides three mechanisms for securing Web services at the message level: authentication, integrity, and confidentiality.

"SAML token capabilities" on page 804
This topic describes WS-Security SAML token capabilities of the broker.

"Identity" on page 390
In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

"Identity and security token propagation" on page 426
Identity and security token propagation enables the identity and security tokens (associated with each message) to be propagated throughout a message flow, and on to target applications through output or request nodes.

Related tasks:

"Configuring the extraction of an identity or security token" on page 447
You can configure the SecurityPEP node or security enabled input nodes to extract the identity or security token from a message and store it in the properties tree identity fields, enabling it to be processed throughout the message flow and propagated at output or request nodes.

Related reference:

"SAML token capabilities for encryption, decryption, signing, and verifying" on page 805
The broker provides SAML pass-through support, which means that the encryption, decryption, signing, and verifying mechanisms for achieving SAML subject confirmation are not enforced by the broker.

"SAML token capabilities for identity mapping" on page 807
This topic describes the broker Web services capability for identity mapping using a SAML token.

"SAML token capabilities for authentication and authorization" on page 806
For Web services, you can complete authentication and authorization using a SAML token.

Kerberos token capabilities:

This topic describes WS-Security Kerberos token capabilities of the broker.

For details of using WS-Security Kerberos tokens, see the following topics:

- "Kerberos token capabilities for encryption, decryption, signing, and verifying" on page 810
- "Kerberos token capabilities for authentication and authorization" on page 811
- "Kerberos token capabilities for identity mapping" on page 812
- "Kerberos token capabilities for extraction and propagation" on page 812

Related concepts:

"WS-Security mechanisms" on page 768
The WS-Security specification provides three mechanisms for securing Web services at the message level: authentication, integrity, and confidentiality.

"Identity" on page 390
In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

Related tasks:

"Setting up message flow security" on page 431
Set up security on a message flow to control access based on the identity of a message passing through the message flow.

"Broker component security" on page 497
You must consider several security aspects when you are setting up brokers running on Windows, Linux, or UNIX platforms.

Kerberos token capabilities for encryption, decryption, signing, and verifying:

You can use Kerberos tokens for encryption, decryption, signing, and verifying.

Kerberos token encryption for providing message "Confidentiality" on page 769 and "Integrity" on page 769 on outgoing SOAP messages from the broker is supported in the following configurations:

Capability

- Encrypt, by using a Kerberos Key Distribution Center (KDC)
- Decrypt, by using the Kerberos `keytab` file

Policy Enforcement Point (PEP) and direction

- In (provider)

  "SOAPInput node" on page 4795
- In (consumer)

  "SOAPRequest node" on page 4828

  "SOAPAsyncResponse node" on page 4777

  Configured with a policy set and binding defining the message "Integrity" on page 772
- Out (consumer)

  "SOAPRequest node" on page 4828

  "SOAPAsyncRequest node" on page 4750
- Out (provider)

  "SOAPReply node" on page 4819

  Configured with a Kerberos policy set and binding.

Trust Store or Policy Decision Point (PDP)

- Kerberos KDC.

Related concepts:

"WS-Security mechanisms" on page 768
The WS-Security specification provides three mechanisms for securing Web services at the message level: authentication, integrity, and confidentiality.

"Identity" on page 390
In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

"Message flow security overview" on page 383
WebSphere Message Broker provides a security manager, which enables you to control access to individual messages in a message flow, using the identity of the message.

"Kerberos token capabilities" on page 809
This topic describes WS-Security Kerberos token capabilities of the broker.

Related reference:

"Kerberos token capabilities for authentication and authorization" on page 811
This topic describes the broker Web services capability for authentication, authorization, or both using a Kerberos token.

"Kerberos token capabilities for identity mapping" on page 812
This topic describes broker Web services capability for identity mapping using a Kerberos token.

"Kerberos token capabilities for extraction and propagation" on page 812
This topic describes broker capability for extraction, propagation, or both using a Kerberos token in Web services.

Kerberos token capabilities for authentication and authorization:

This topic describes the broker Web services capability for authentication, authorization, or both using a Kerberos token.

Kerberos is not applicable to authorization. Kerberos token encryption for providing message "Authentication" on page 770 on outgoing SOAP messages from the broker is supported in the following configurations:

Capability

- Authenticate using a Kerberos `keytab` file.

Policy Enforcement Point (PEP) and direction

- In (provider)

  "SOAPInput node" on page 4795
- In (consumer)

  "SOAPRequest node" on page 4828

  "SOAPAsyncResponse node" on page 4777

  Configured with a policy set and binding defining the message "Integrity" on page 772

Trust Store or Policy Decision Point (PDP)

- Kerberos `keytab` file.

Related concepts:

"WS-Security mechanisms" on page 768
The WS-Security specification provides three mechanisms for securing Web services at the message level: authentication, integrity, and confidentiality.

"Identity" on page 390
In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

"Message flow security overview" on page 383
WebSphere Message Broker provides a security manager, which enables you to control access to individual messages in a message flow, using the identity of the message.

"Kerberos token capabilities" on page 809
This topic describes WS-Security Kerberos token capabilities of the broker.

"Authorization" on page 401
Authorization is the process of verifying that an identity token has permission to access a message flow.

"Authentication and validation" on page 398
Authentication is the process of establishing the identity of a user or system and verifying that the identity is valid. Applying authentication to a SAML security token involves validating the assertions that it carries and confirming that it is being processed within its validity period.

Related reference:

"Kerberos token capabilities for encryption, decryption, signing, and verifying" on page 810
You can use Kerberos tokens for encryption, decryption, signing, and verifying.

"Kerberos token capabilities for identity mapping"
This topic describes broker Web services capability for identity mapping using a Kerberos token.

"Kerberos token capabilities for extraction and propagation"
This topic describes broker capability for extraction, propagation, or both using a Kerberos token in Web services.

Kerberos token capabilities for identity mapping:

This topic describes broker Web services capability for identity mapping using a Kerberos token.

Kerberos tickets from SOAP nodes are not supported for token mapping with an external security token server (STS) configured in the security profile.

On the Inbound route, with SOAPInput and SOAPAsyncResponse nodes, the presence of a security profile with propagation enabled causes the Kerberos Service Principal Name (SPN) to be placed in the properties tree as a Username token.

On the Outbound route, with SOAPRequest and SOAPAsyncRequest nodes, identity propagation can be used to provide the Kerberos Key Distribution Center (KDC) credentials. Arrange for the KDC credentials to be set as a Username and password token in the properties tree and associate the SOAP node with a security profile that specifies propagation; otherwise the KDC credentials are obtained using the Kerberos resource credentials that are created using the **mqsisetdbparms** command.

Related concepts:

"WS-Security mechanisms" on page 768
The WS-Security specification provides three mechanisms for securing Web services at the message level: authentication, integrity, and confidentiality.

"Kerberos token capabilities" on page 809
This topic describes WS-Security Kerberos token capabilities of the broker.

"Identity mapping" on page 403
Identity mapping is the transformation of a security token from one format to another format, or the federation of an identity from one realm to an equivalent identity in another realm.

"Identity" on page 390
In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

Related reference:

"Kerberos token capabilities for encryption, decryption, signing, and verifying" on page 810
You can use Kerberos tokens for encryption, decryption, signing, and verifying.

"Kerberos token capabilities for authentication and authorization" on page 811
This topic describes the broker Web services capability for authentication, authorization, or both using a Kerberos token.

"Kerberos token capabilities for extraction and propagation"
This topic describes broker capability for extraction, propagation, or both using a Kerberos token in Web services.

Kerberos token capabilities for extraction and propagation:

This topic describes broker capability for extraction, propagation, or both using a Kerberos token in Web services.

Kerberos tickets from SOAP nodes are not supported for token extraction and propagation with an external security token server (STS) configured in the security profile.

On the Inbound route, with SOAPInput and SOAPAsyncResponse nodes, the presence of a security profile with propagation enabled causes the Kerberos Service Principal Name (SPN) to be placed in the properties tree as a Username token.

On the Outbound route, with SOAPRequest and SOAPAsyncRequest nodes, identity propagation can be used to provide the Kerberos Key Distribution Center (KDC) credentials. Arrange for the KDC credentials to be set as a Username and password token in the properties tree and associate the SOAP node with a security profile that specifies propagation; otherwise the KDC credentials are obtained using the Kerberos resource credentials that are created using the **mqsisetdbparms** command.

Related concepts:

"WS-Security mechanisms" on page 768
The WS-Security specification provides three mechanisms for securing Web services at the message level: authentication, integrity, and confidentiality.

"Kerberos token capabilities" on page 809
This topic describes WS-Security Kerberos token capabilities of the broker.

"Identity" on page 390
In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

"Identity and security token propagation" on page 426
Identity and security token propagation enables the identity and security tokens (associated with each message) to be propagated throughout a message flow, and on to target applications through output or request nodes.

Related tasks:

"Configuring the extraction of an identity or security token" on page 447
You can configure the SecurityPEP node or security enabled input nodes to extract the identity or security token from a message and store it in the properties tree identity fields, enabling it to be processed throughout the message flow and propagated at output or request nodes.

Related reference:

"Kerberos token capabilities for encryption, decryption, signing, and verifying" on page 810
You can use Kerberos tokens for encryption, decryption, signing, and verifying.

"Kerberos token capabilities for authentication and authorization" on page 811
This topic describes the broker Web services capability for authentication, authorization, or both using a Kerberos token.

"Kerberos token capabilities for identity mapping" on page 812
This topic describes broker Web services capability for identity mapping using a Kerberos token.

LTPA token capabilities:

This topic describes WS-Security LTPA token capabilities of the broker.

The broker provides LTPA pass-through support, which means that the LTPA token is extracted and passed to an external security token service (STS) for validation. The STS to be used is specified on a security profile.

For information about using WS-Security LTPA tokens, see the following topics:

- "LTPA token capabilities for encryption, decryption, signing, and verifying"
- "LTPA token capabilities for authentication and authorization" on page 815
- "LTPA token capabilities for identity mapping" on page 816
- "LTPA token capabilities for extraction and propagation" on page 817

Related concepts:

"WS-Security mechanisms" on page 768
The WS-Security specification provides three mechanisms for securing Web services at the message level: authentication, integrity, and confidentiality.

"Identity" on page 390
In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

Related tasks:

"Setting up message flow security" on page 431
Set up security on a message flow to control access based on the identity of a message passing through the message flow.

"Broker component security" on page 497
You must consider several security aspects when you are setting up brokers running on Windows, Linux, or UNIX platforms.

LTPA token capabilities for encryption, decryption, signing, and verifying:

This topic describes broker Web services capability for encryption, decryption, signing, and verifying using LTPA tokens.

The LTPA token is not applicable, or supported, for the following in any configuration or direction:

- Encryption
- Decryption
- Signing
- Verifying

Related concepts:

"WS-Security mechanisms" on page 768
The WS-Security specification provides three mechanisms for securing Web services at the message level: authentication, integrity, and confidentiality.

"LTPA token capabilities" on page 813
This topic describes WS-Security LTPA token capabilities of the broker.

Related reference:

"LTPA token capabilities for authentication and authorization" on page 815
For Web services, you can complete authentication and authorization using an LTPA token.

"LTPA token capabilities for identity mapping" on page 816
This topic describes the broker Web services capability for identity mapping using an LTPA token.

"LTPA token capabilities for extraction and propagation" on page 817
This topic describes broker capability for extraction, propagation, or both using an LTPA token in Web services.

LTPA token capabilities for authentication and authorization:

For Web services, you can complete authentication and authorization using an LTPA token.

The LTPA token "Authentication" on page 768 and "Authorization" on page 401 are supported only in the following configuration:

Capability

- Authenticate
- Authorize

Policy Enforcement Point (PEP) and direction

- In (provider)

  "SOAPInput node" on page 4795

  Configured with a security policy set and binding that defines that an LTPA token is present for authentication; see "Authentication" on page 770. The broker provides only LTPA pass-through support, which means that the LTPA token is extracted and passed to an external security token service (STS) for validation. The STS to be used is specified in a security profile. The STS processing can be used to implement authentication and authorization based on the LTPA principal and realm.

  Configured with a security profile defining the Policy Decision Point (PDP); see the PDP section that follows.

Trust Store or PDP

- WS-Trust v1.3 STS

  Configured by using a WS-Trust v1.3 STS security profile specifying authentication, authorization or both; see "Creating a security profile for WS-Trust V1.3 (TFIM V6.2)" on page 440.

Related concepts:

"WS-Security mechanisms" on page 768
The WS-Security specification provides three mechanisms for securing Web services at the message level: authentication, integrity, and confidentiality.

"LTPA token capabilities" on page 813
This topic describes WS-Security LTPA token capabilities of the broker.

"Authorization" on page 401
Authorization is the process of verifying that an identity token has permission to access a message flow.

"Authentication and validation" on page 398
Authentication is the process of establishing the identity of a user or system and verifying that the identity is valid. Applying authentication to a SAML security token involves validating the assertions that it carries and confirming that it is being processed within its validity period.

"Identity" on page 390
In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

Related reference:

"LTPA token capabilities for identity mapping" on page 816
This topic describes the broker Web services capability for identity mapping using an LTPA token.

"LTPA token capabilities for extraction and propagation" on page 817
This topic describes broker capability for extraction, propagation, or both using an LTPA token in Web services.

"LTPA token capabilities for encryption, decryption, signing, and verifying" on page 814
This topic describes broker Web services capability for encryption, decryption, signing, and verifying using LTPA tokens.

LTPA token capabilities for identity mapping:

This topic describes the broker Web services capability for identity mapping using an LTPA token.

"Identity mapping" on page 403 from or to an LTPA identity token is supported only in the following configurations:

Capability

- Identity mapping

Policy Enforcement Point (PEP) and direction

- In (provider)

  "SOAPInput node" on page 4795

  Configured with a security policy set and bindings that specifies an LTPA pass-through authentication token.

  Configured with a security profile defining the external Policy Decision Point (PDP); see the PDP section that follows.

Trust store or PDP

- WS-Trust v1.3 STS

  Configured by using a WS-Trust v1.3 STS security profile that specifies identity mapping; see "Creating a security profile for WS-Trust V1.3 (TFIM V6.2)" on page 440.

Related concepts:

"WS-Security mechanisms" on page 768
The WS-Security specification provides three mechanisms for securing Web services at the message level: authentication, integrity, and confidentiality.

"LTPA token capabilities" on page 813
This topic describes WS-Security LTPA token capabilities of the broker.

"Identity mapping" on page 403
Identity mapping is the transformation of a security token from one format to another format, or the federation of an identity from one realm to an equivalent identity in another realm.

"Identity" on page 390
In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

Related reference:

"LTPA token capabilities for authentication and authorization" on page 815
For Web services, you can complete authentication and authorization using an LTPA token.

"LTPA token capabilities for extraction and propagation" on page 817
This topic describes broker capability for extraction, propagation, or both using an LTPA token in Web services.

"LTPA token capabilities for encryption, decryption, signing, and verifying" on page 814
This topic describes broker Web services capability for encryption, decryption, signing, and verifying using LTPA tokens.

LTPA token capabilities for extraction and propagation:

This topic describes broker capability for extraction, propagation, or both using an LTPA token in Web services.

The extraction of an LTPA token into the Properties folder source "Identity" on page 390 fields, is supported in the following configurations:

Capability

- Extraction

Policy Enforcement Point (PEP) and direction

- In (provider)

  "SOAPInput node" on page 4795

  Configured with a security policy set and bindings that specifies an LTPA pass-though authentication token.

  Configured with a security profile that defines propagation; see "Creating a security profile" on page 433

The propagation of an LTPA token into the SOAP WS-Security header, from the token present in either the mapped or the source identity fields in the properties folder, is supported in the following configuration. For more information, see "Identity" on page 390.

Capability

- Propagate

Policy Enforcement Point (PEP) and direction

- Out (consumer)

  "SOAPRequest node" on page 4828

  "SOAPAsyncRequest node" on page 4750

  Configured with a security profile that defines propagation. For more information, see "Security profiles" on page 387.

  Configured with a security policy set and bindings that specifies an LTPA pass-through authentication token.

Related concepts:

"WS-Security mechanisms" on page 768
The WS-Security specification provides three mechanisms for securing Web services at the message level: authentication, integrity, and confidentiality.

"LTPA token capabilities" on page 813
This topic describes WS-Security LTPA token capabilities of the broker.

"Identity" on page 390
In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

"Identity and security token propagation" on page 426
Identity and security token propagation enables the identity and security tokens (associated with each message) to be propagated throughout a message flow, and

on to target applications through output or request nodes.

Related tasks:

"Configuring the extraction of an identity or security token" on page 447
You can configure the SecurityPEP node or security enabled input nodes to extract the identity or security token from a message and store it in the properties tree identity fields, enabling it to be processed throughout the message flow and propagated at output or request nodes.

Related reference:

"LTPA token capabilities for authentication and authorization" on page 815
For Web services, you can complete authentication and authorization using an LTPA token.

"LTPA token capabilities for identity mapping" on page 816
This topic describes the broker Web services capability for identity mapping using an LTPA token.

"LTPA token capabilities for encryption, decryption, signing, and verifying" on page 814
This topic describes broker Web services capability for encryption, decryption, signing, and verifying using LTPA tokens.

Moving from WebSphere Message Broker on a distributed system to z/OS

Define resources for WebSphere Message Broker for z/OS and move your message flow applications.

About this task

Read the following topics for guidance on what action you might want to take in moving part of your operations to z/OS:

- "z/OS customization overview" on page 592
- "Customizing the z/OS environment" on page 591
- "Creating a broker on z/OS" on page 620
- "Administration in z/OS" on page 3979

After reviewing your requirements, re-create your broker on z/OS and deploy your message flows and execution groups to the broker on z/OS. If you have extended WebSphere Message Broker in a distributed environment with user-defined parsers or message processing nodes, port them to run under z/OS.

Also consider the following points:

- Floating point conversion: z/OS runs under z/OS floating point format, so floating point operations on z/OS run in a different range and accuracy from distributed systems.
- Administration commands are partially implemented as console commands and partially as JCL commands. Some commands provide both options.
- Event log messages: All address spaces have a JOBLOG where messages appear. In addition to this, all messages appear on the SYSLOG, with important operator messages being filtered to the console through MPF (Message Processing Facility).

For information about message flow transactionality, see "Message flow transactions" on page 1281.

Moving user applications

About this task

You can write your own applications to work with WebSphere Message Broker. If these applications use the common subset of functionality of all WebSphere Message Broker brokers, no migration is necessary. If you are using functionality that is available on some WebSphere Message Broker systems only, for example message segmentation and WebSphere MQ message groups, be aware that WebSphere Message Broker for z/OS does not provide support for this migration.

Related concepts:

"z/OS customization overview" on page 592
After you have used SMP/E to install WebSphere Message Broker for z/OS, the installed executable code is located inside the file system. JCL samples are located in the PDS `<hlq>.SBIPSAMP`, the JCL procedures are located in the PDS `<hlq>.SBIPPROC`, and load module for synchronizing statistics with SMF are located in the PDS `<hlq>.SBIPAUTH`.

Related tasks:

"Customizing the z/OS environment" on page 591
If you are planning to use a z/OS environment, consider whether to create your brokers on z/OS. You must also complete a number of tasks to configure your environment.

"Creating a broker on z/OS" on page 620
Create the broker component and the other resources on which it depends.

Related reference:

"Administration in z/OS" on page 3979
In the z/OS environment, commands are issued through the console and others in batch jobs.

Changing locales

You can change the locale for the system on which a runtime component is installed.

About this task

The way in which you change the locale depends on the operating system:

- "Changing your locale on Linux and UNIX systems" on page 820
- "Changing your locale on Windows" on page 822
- "Changing your locale on z/OS" on page 823

WebSphere Message Broker uses code page converters to support character sets from different environments. "Code page converters" on page 823 describes what a code page converter is, and how to generate new converters.

Related reference:

"Supported processors" on page 3583
WebSphere Message Broker is supported on multiple processors.

"Operating system requirements" on page 3590
WebSphere Message Broker is supported on multiple operating systems.

"Supported code pages" on page 4176
Application messages must conform to supported code pages.

"Locales" on page 3629
Message support is provided in a number of locales.

Changing your locale on Linux and UNIX systems

You can change your system locale on UNIX and Linux systems.

About this task

You can set environment variables to control the system locale. You can set these variables to be system-wide, or on a per-session basis:

LC_ALL Overrides all LC_* environment variables with the given value

LC_CTYPE
Character classification and case conversion

LC_COLLATE
Collation (sort) order

LC_TIME
Date and time formats

LC_NUMERIC
Non-monetary numeric formats

LC_MONETARY
Monetary formats

LC_MESSAGES
Formats of informative and diagnostic messages, and of interactive responses

LC_PAPER
Paper size

LC_NAME
Name formats

LC_ADDRESS
Address formats and location information

LC_TELEPHONE
Telephone number formats

LC_MEASUREMENT
Measurement units (Metric or Other)

LC_IDENTIFICATION
Metadata about the locale information

LANG The default value, which is used when either LC_ALL is not set, or an applicable value for LC_* is not set

NLSPATH
Delimited list of paths to search for message catalogs

TZ Time zone

LC_MESSAGES and NLSPATH are the most important variables to the broker. These variables define the language and location of response messages that the broker uses. The broker profile file, mqsiprofile, sets NLSPATH. Either you, or your system must set LC_MESSAGES. The value set in LC_MESSAGES must be a value that is installed on your machine and that the broker recognizes. LC_CTYPE is also important to the broker because it defines the character conversion that the broker performs when interacting with the local environment.

Before setting these variables, check that the language and code page are installed on your machine, and are supported by WebSphere Message Broker.

You can use the command **locale** to show your current locale. The command **locale -a** displays all the locales that are currently installed on the machine. Make sure that the locale you select for LANG and LC_ALL is in the list that is returned by the command **locale -a**. The values that locale uses and returns are case sensitive, therefore copy them exactly when assigning them to an environment variable.

For information on languages and code pages supported by WebSphere Message Broker, see "Locales" on page 3629 and "Supported code pages" on page 4176.

If you use common desktop environment (CDE), use this environment to set the locale instead of setting LANG and LC_ALL directly. The NLSPATH variable respects either method.

For example, to set WebSphere Message Broker to run in a UTF-8 environment set the following values in the profile:

```
LANG=en_US.utf-8
LC_ALL=en_US.utf-8
```

where en_US sets the language, and utf-8 sets the code page.

When you start a broker component, the locale of that component is inherited from the shell in which it is started. The broker component uses the LC_MESSAGES environment variable as the search path in the NLSPATH environment variable (LC_MESSAGES is set when variable LC_ALL is exported).

Messages are sent to the syslog in the code page set by this locale. If you have multiple brokers that write to this syslog, their messages are in the code page of the locale in which they were started, for example:

locale	syslog code page	ccsid
pt_BR	iso8859-1	819
pt_BR	ibm-850	850
pt_BR	utf-8	1208

Set the locale of the user ID that runs the syslog daemon to one that is compatible with the locales of all brokers that write to the syslog on that system, for example, utf-8. For compatibility, you can set the default locale. On Solaris, set the LANG and LC_ALL variables in /etc/default/init. On AIX and Linux, these variables are in /etc/environment. This task is not required on HP-UX.

For full-time zone support in the broker, set the TZ variable using Continent/City notation. For example set TZ to Europe/London to make London, England the time zone, or set it to America/New_York to make New York, America the time zone.

If you want to add a new locale, refer to the operating system documentation for information about how to complete that task. If the code page of the new locale is not supported by WebSphere Message Broker you must add it by "Generating a new code page converter" on page 824.

Related tasks:

"Generating a new code page converter" on page 824
Generate a code page converter to handle conversions of data that belongs to a code page that is not in the default set of code pages provided by WebSphere Message Broker.

Related reference:

"Supported processors" on page 3583
WebSphere Message Broker is supported on multiple processors.

"Operating system requirements" on page 3590
WebSphere Message Broker is supported on multiple operating systems.

"Supported code pages" on page 4176
Application messages must conform to supported code pages.

Changing your locale on Windows

Change your system locale on Windows to view objects and information in a different language or code page.

About this task

Brokers are started as services on Windows, and are therefore influenced by the system locale. The command-line functions are influenced by the locale that is set for the current user. WebSphere Message Broker on Windows has all locale information installed by default. However, you might have to install additional locale packages, if prompted to do so by the Windows operating system.

To change locale, use one of the following methods:

- Install a locale-specific operating system.
- Alter the system or user locale by selecting *Regional Settings* in the Control Panel.

Messages are sent to the Event Log in the code page set by the current locale.

You can use the **chcp** command to change the active console code page. Enter the command at a command prompt; if you enter **chcp** without a parameter, it displays the current setting. If you enter it with a code page, it changes the locale to that code page.

For example, to check the current code page setting:

```
C:\>chcp
Active code page: 437
```

The current page is displayed (437 represents US-ASCII). If you want to change the value to GB18030, enter:

```
C:\>chcp 54936
Active code page: 54936
```

Before you use a code page, search for `windows-`*`number`* where *`number`* is the active code page you want to use in the list of "Supported code pages" on page 4176. If the code page is not in the list, either use a code page that is in the list, or generate a new code page converter.

Related tasks:

"Generating a new code page converter" on page 824
Generate a code page converter to handle conversions of data that belongs to a code page that is not in the default set of code pages provided by WebSphere Message Broker.

Related reference:

"Supported processors" on page 3583
WebSphere Message Broker is supported on multiple processors.

"Operating system requirements" on page 3590
WebSphere Message Broker is supported on multiple operating systems.

"Supported code pages" on page 4176
Application messages must conform to supported code pages.

Changing your locale on z/OS

You can change your system locale on z/OS. If you want to change your system locale on z/OS, set the `LANG`, `LC_ALL`, and `NLSPATH` variables.

About this task

See "Installation information - broker" on page 621 for further information.

The locale is set in the broker profile (`BIPBPROF`) and you must run `BIPGEN` to create the broker `ENVFILE` and any execution group specific ENVFILE.

You can use the UNIX System Services (USS) executable `locale` to show your current locale. The command `locale -a` displays all the locales currently installed on the computer. Refer to the operating system documentation for information about adding new locales. If you add a new locale after you have installed WebSphere Message Broker, install that locale's message catalogs from the original install media.

You can set WebSphere Message Broker to operate with a specific code page. Set the code page after a period in the LANG and LC_ALL variable. This example sets the locale to `En_us` and the code page to `IBM-1140` (EBCDIC En_us with euro):

```
LANG=En_us.IBM-1140
LC_ALL=En_us.IBM-1140
```

Make sure that the selected code page is one of the "Supported code pages" on page 4176. If the code page is not in the list, either use a code page that is in the list, or generate a new code page converter.

Related tasks:

"Generating a new code page converter" on page 824
Generate a code page converter to handle conversions of data that belongs to a code page that is not in the default set of code pages provided by WebSphere Message Broker.

Related reference:

"Supported processors" on page 3583
WebSphere Message Broker is supported on multiple processors.

"Operating system requirements" on page 3590
WebSphere Message Broker is supported on multiple operating systems.

"Supported code pages" on page 4176
Application messages must conform to supported code pages.

Code page converters

Brokers complete string operations in Universal Character Set coded in 2 octets (UCS-2). If incoming strings are not encoded in UCS-2, they are converted to UCS-2 on arrival.

The broker uses international components for Unicode (ICU) code page converters to convert data. The Unicode Consortium has further information on Unicode.

A code page converter is a mapping from the byte sequence in one code page to a serialized representation of UCS-2, known as UCS Transformation Format 16 bit form (UTF-16). A code page converter allows the broker to create a UCS-2 representation of an incoming string.

When you handle UTF-16 data, CCSIDs 1200, 13488 and 17584 are treated differently to others. Traditionally, in ICU usage, the endian encoding of these CSSIDs was platform-specific, and WebSphere Message Broker uses an encoding parameter with these CSSIDs. You can specify the encoding parameter as MQENC_INTEGER_REVERSED to use these CCSIDs to explicitly produce little endian data.

Consider this example of the use of a code page converter. A message comes in on a queue from z/OS, with the WebSphere MQ CCSID field set to 1047 (LATIN-1 Open Systems without euro). The broker looks up ibm-1047 and uses the resulting converter to create a UCS-2 representation for internal use.

If you try to convert from a Unicode to a Non-Unicode character set, the following errors might occur:

- The target buffer is too small. This error causes a recoverable exception, which you can handle; alternatively, the message is rolled back.
- A code point in the source does not have an equivalent value in the target. At first, fallback mappings are attempted (for example, if you are converting to Japanese, a backslash (\) can be mapped to a yen (¥) if the conversion supplies it as a fallback mapping). If fallback mappings are not present, a recoverable exception is thrown. You can handle the exception, or the message is rolled back.

  The MRM parser substitutes invalid code points with substitution characters.

WebSphere Message Broker currently supports the code pages listed in “Supported code pages” on page 4176. If you need support for an additional code page, or if you require a different variant of a code page, you can extend the broker to support this code page.

Related tasks:

“Generating a new code page converter”
Generate a code page converter to handle conversions of data that belongs to a code page that is not in the default set of code pages provided by WebSphere Message Broker.

Related reference:

“Supported code pages” on page 4176
Application messages must conform to supported code pages.

Generating a new code page converter:

Generate a code page converter to handle conversions of data that belongs to a code page that is not in the default set of code pages provided by WebSphere Message Broker.

Before you begin

Before you start:

- Read “Code page converters” on page 823, which provides information about what a code page converter is, and about the code pages that WebSphere Message Broker supports.

About this task

To generate a new code page converter:

Procedure

1. Create or find a mapping data file with the file extension `.ucm` for the converter that you require. You can download `.ucm` files from the ICU Character set mapping files archive. These mapping data files are available and can be modified without restriction. An example mapping data file is `ibm-1284_P100-1996.ucm`. (ICU is an external open source project, not an IBM tool.)
2. Rename the `.ucm` to a file name with the format `ibm-`*`number`*`.ucm` where *number* is a number that you choose to identify the code page. Make sure that this number is not already used in one of the "Supported code pages" on page 4176. For example, you could rename `ibm-1284_P100-1996.ucm` to `ibm-1284.ucm`.
3. Go to ICU downloads and download the binary distribution for your system. An exact match is not important provided that the binary files are compatible. If you have problems building the converter, see the ICU user guide.
4. Extract the files from the binary distribution archive into a temporary directory.
5. Copy the library and binary files to a directory in the environment PATH and LIBPATH. (Alternatively, copy the library and binary files to directory that is not temporary and modify the environment PATH and LIBPATH to include this directory.)
6. One of the extracted files is `makeconv.exe`; use this makeconv tool to convert the mapping data file (`.ucm` files) into a binary converter file (`.cnv` file), by entering the following command:

   makeconv *mapping_file.ucm*

   where *mapping_file.ucm* is the mapping data file that you are using.

   The name of the binary converter file that **makeconv** produces is:

   mapping_file.cnv

   where *mapping_file.cnv* is the name of the mapping data file that was converted.

   To make the `.cnv` file for `ibm-1284.ucm`, use the following command:

   ```
   makeconv ibm-1284.ucm
   ```
7. Copy the file with the file extension `.cnv` for the code page that you need, into a directory that WebSphere Message Broker can access.

   The name and location of the file is of the form

   ibm-1284`.cnv`

   and is located in the *$ICU_DATA/icudt38<platform-suffix>* directory, where the *<platform-suffix>* is one of the following values:

   - l for little-endian ASCII platforms
   - b for big-endian ASCII platforms
   - e for EBCDIC platforms
8. Optional: If you do not want the new code page converter to be in the same location as other ICU data, you must associate the broker with the new directory where the converter is stored (the directory added must contain the full path, not including the icudt38x subdirectory):
   - To create a new broker that is associated with the converter, include the **-c** parameter on the **mqsicreatebroker** command.

- To alter an existing broker to recognize the converter, include the **-c** parameter on the **mqsichangebroker** command.
- To affect all the products and the broker command-line tools that are using ICU, add the *directory* to the **ICU_DATA** environment variable. If you have used either the **mqsicreatebroker** command or the **mqsichangebroker** command to specify the code page converter to be used, the broker ignores the **ICU_DATA** value.

Note: To ensure consistent behavior in all components, modify the **ICU_DATA** environment variable.

Related concepts:

"Code page converters" on page 823
Brokers complete string operations in Universal Character Set coded in 2 octets (UCS-2). If incoming strings are not encoded in UCS-2, they are converted to UCS-2 on arrival.

Related tasks:

"Changing locales" on page 819
You can change the locale for the system on which a runtime component is installed.

Related reference:

"Supported code pages" on page 4176
Application messages must conform to supported code pages.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

Configuring for high availability

If you want to operate your WebSphere Message Broker instances in a highly available configuration, you can set up your brokers to work either with a high availability manager, such as HACMP, or with WebSphere MQ multi-instance queue managers.

About this task

- "Configuring a WebSphere Message Broker to run in multi-instance mode" on page 827
- "Using a broker with an existing high availability manager" on page 843
- "Using a broker with an existing Windows Cluster (Windows Server 2003 or 2008)" on page 853

Related concepts:

"HTTP proxy servlet overview" on page 856
The HTTP proxy servlet is a Java servlet that you can use in an external web servlet container, such as IBM WebSphere Application Server or Apache Tomcat, to receive HTTP requests from web services client applications, to replace the support provided by the broker and execution group HTTP listeners.

Related tasks:

Chapter 7, “Configuring brokers for test and production environments,” on page 579
Create one or more brokers on one or more computers, and configure them on your test and production systems to process messages that contain your business data.

Configuring a WebSphere Message Broker to run in multi-instance mode

How you configure a WebSphere Message Broker to run in multi-instance mode.

Before you begin

Before you start:

Read the overview of multi-instance queue managers on WebSphere MQ; see “Multi-instance queue managers” on page 828. For further information on multi-instance queue managers see the WebSphere MQ information center.

About this task

The set of tasks describing how you configure a multi-instance broker on WebSphere Message Broker assumes the following:

- A network server is being configured to host the shared work path for the multi-instance broker and the shared directories for the multi-instance queue manager.
- The network server is shared between two client machines, each of which has a licensed copy of the WebSphere Message Broker and WebSphere MQ products installed.
- The platform for the illustration is assumed to be AIX, but the steps are true for all platforms supported by WebSphere Message Broker and WebSphere MQ except for z/OS.

  Multi-instance brokers and multi-instance queue managers are not supported on z/OS

There are three steps in configuring a WebSphere Message Broker to run in multi-instance mode:

Procedure

1. Create a Shared work path directory on an NFS or NAS server. On Windows there is the option to use a shared UNC path.
2. Create a WebSphere MQ multi-instance queue manager.
3. Create a Multi-instance broker

Related concepts:

“Multi-instance queue managers” on page 828
A multi-instance queue manager restarts automatically on a standby server.

Related tasks:

“Creating the shared directories” on page 829
How you create the shared directories that you need for your multi-instance WebSphere Message Broker.

“Creating the WebSphere MQ multi-instance queue manager” on page 830
How you create the WebSphere MQ multi-instance queue manager that you need for your multi-instance WebSphere Message Broker.

"Creating the multi-instance broker" on page 837
How you create the multi-instance broker.

Related information:

WebSphere MQ Version 7 Information Center online

Multi-instance queue managers

A multi-instance queue manager restarts automatically on a standby server.

Figure 1 shows a multi-instance configuration for QM1. WebSphere MQ is installed on two servers, one of which is a spare. One queue manager, QM1, has been created. One instance of QM1 is active, and is running on one server. The other instance of QM1 is running in standby on the other server, doing no active processing, but ready to take over from the active instance of QM1, if the active instance fails.

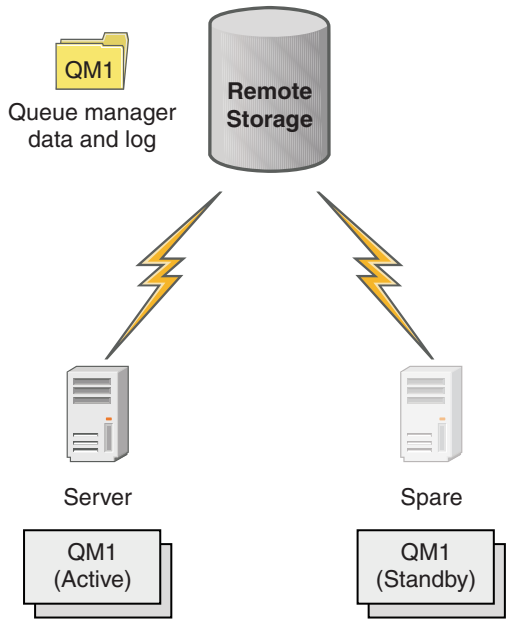


Figure 1. Multi-instance queue manager

When you intend to use a queue manager as a multi-instance queue manager, create a single queue manager on one of the servers using the WebSphere MQ **crtmqm** command, placing its queue manager data and logs in shared network storage. On the other server, rather than create the queue manager again, use the WebSphere MQ **addmqinf** command to create a reference to the queue manager data and logs on the network storage.

You can now run the queue manager from either of the servers. Each of the servers references the same queue manager data and logs; there is only one queue manager, and it is active on only one server at a time.

The queue manager can run either as a single instance queue manager, or as a multi-instance queue manager. In both cases only one instance of the queue manager is running, processing requests. The difference is that when running as a multi-instance queue manager, the server that is not running the active instance of the queue manager runs as a standby instance, ready to take over from the active instance automatically if the active server fails.

The only control you have over which instance becomes active first is the order in which you start the queue manager on the two servers. The first instance to acquire read/write locks to the queue manager data becomes the active instance.

You can swap the active instance to the other server, once it has started, by stopping the active instance using the switchover option to transfer control to the standby.

The active instance of QM1 has exclusive access to the shared queue manager data and logs folders when it is running. The standby instance of QM1 detects when the active instance has failed, and becomes the active instance. It takes over the QM1 data and logs in the state they were left by the active instance, and accepts reconnections from clients and channels.

The active instance might fail for various reasons that result in the standby taking over:

- Failure of the server hosting the active queue manager instance.
- Failure of connectivity between the server hosting the active queue manager instance and the file system.
- Unresponsiveness of queue manager processes, detected by WebSphere MQ, which then shuts down the queue manager.

You can add the queue manager configuration information to multiple servers, and choose any two servers to run as the active/standby pair.

A multi-instance queue manager is one part of a high availability solution. You need some additional components to build a useful high availability solution.

- Client and channel reconnection to transfer WebSphere MQ connections to the computer that takes over running the active queue manager instance.
- A high performance shared network file system that manages locks correctly and provides protection against media and file server failure.
- Resilient networks and power supplies to eliminate single points of failure in the basic infrastructure.
- Applications that tolerate failover. In particular you need to pay close attention to the behavior of transactional applications, and to applications that browse WebSphere MQ queues.
- Monitoring and management of the active and standby instances to ensure that they are running, and to restart active instances that have failed. Although multi-instance queue managers restart automatically, you need to be sure that your standby instances are running, ready to take over, and that failed instances are brought back online as new standby instances.

WebSphere MQ clients and channels reconnect automatically to the standby queue manager when it becomes active. Reconnection, and the other components in a high availability solution are discussed in related topics.

Creating the shared directories

How you create the shared directories that you need for your multi-instance WebSphere Message Broker.

Before you begin

Before you start:

Before you create the shared directory, read the documentation supplied with your NFS or NAS product used at your enterprise (or on Windows a shared UNC path). Note that, if you are intending to use an NFS server, this server must use NFSv4.

About this task

The following procedure gives a generalized set of instructions for a NFS shared path, because the specific instructions vary by product:

Procedure

1. Create an NFS Share on the server for the multi-instance WebSphere MQ queue manager.
2. Mount the NFS Share on both client nodes A and B, for the WebSphere MQ multi-instance queue manager, using a suitable location; for example, /Shared/Location/WMQ.
3. Ensure that the WMQ directories are owned by user and group mqm, and that the access permissions are set to rwx for user and group. For example **ls -al** displays:

   ```
   drwxrwxr-x mqm mqm 4096 Jun 14 14:38 WMQ
   ```

4. Create an NFS share on the server for the multi-instance broker.
5. Mount the NFS Share on both client nodes A and B, for the multi-instance broker, using a suitable location; for example, /Shared/Location/WMB.
6. Ensure that the WMB directories are owned by user and group mqbrkrs, and that the access permissions are set to rwx for user and group. For example **ls -al** displays:

   ```
   drwxrwxr-x mqbrkrs mqbrkrs 4096 Jun 14 14:39 WMB
   ```

7. If you are working on AIX you must turn off attribute caching so that you use the NFS server solely for multi-instance tasks.

   Use the following command, on both client nodes A and B, to turn off attribute caching for the example share locations used in the preceding instructions:

   ```
   nfs4cl setfsoptions /Shared/Location/WMQ noac
   nfs4cl setfsoptions /Shared/Location/WMB noac
   ```

   For further information, see AIX NFS commands

What to do next

Create a WebSphere MQ multi-instance queue manager.

Related tasks:

"Configuring a WebSphere Message Broker to run in multi-instance mode" on page 827
How you configure a WebSphere Message Broker to run in multi-instance mode.

"Creating the WebSphere MQ multi-instance queue manager"
How you create the WebSphere MQ multi-instance queue manager that you need for your multi-instance WebSphere Message Broker.

"Creating the multi-instance broker" on page 837
How you create the multi-instance broker.

Creating the WebSphere MQ multi-instance queue manager

How you create the WebSphere MQ multi-instance queue manager that you need for your multi-instance WebSphere Message Broker.

Before you begin

Before you start:

Create the shared directories that you require for the multi-instance queue manager; see "Creating the shared directories" on page 829.

About this task

The following procedure gives an overview of how you create a multi-instance WebSphere MQ queue manager. See "Creating a multi-instance queue manager" on page 832 for more information.

Procedure

1. Create a multi-instance WebSphere MQ queue manager called `QM1` on client node `A`. You do this by using the following command:

   ```
   -crtmqm -md /SharedLocation/WMQ/data
                   -ld /SharedLocation/WMQ/logs QM1
   ```

   where:

   md Is the name of the directory used to hold data files for a queue manager.

   ld Is the name of the directory used to hold log files.

   Note that it is important that the name of the queue manager goes at the end of the syntax. See the WebSphere MQ documentation for further information on the **crtmqm** command.

2. Add the details of WebSphere MQ queue manager `QM1` onto client node `B`. You do this by using the following command:

   ```
   -addmqinf  -v Name=QM1 -v Directory=WMQ -v Prefix=/var/mqm
                       -v DataPath=/SharedLocation/WMQ/data/QM1
   ```

   where:

   Name Is the name of the queue manager.

   Directory
   Is the name of the queue manager data directory.

   Prefix Is the directory path *under* which this queue manager data directory is stored by default.

   Data Path
   Is the data path where the queue manager data files are placed. The value of `Directory` is not appended automatically to this path; you must provide the transformed queue manager name as part of `DataPath`.

   The parameters listed above are all required parameters on Windows and UNIX platforms, with the exception of `DataPath`, which is optional on UNIX platforms only.

   See the WebSphere MQ documentation for further information on the **addmqinf** command.

3. Start queue manager `QM1` on client node `A` in multi-instance mode. You do this by using the following command:

   ```
   strmqm -x QM1
   ```

   See the WebSphere MQ documentation for further information on the **strmqm** command.

4. Observe the queue manager running in active mode. You do this by using the following command:

   ```
   dspmq -x
   ```

   See the WebSphere MQ documentation for further information on the **dspmq** command.
5. Start queue manager QM1 on client node B. Observe the queue manager running in standby mode.
6. Ensure that queue manager QM1 works as follows when simulating a failover from node A to node B:
   a. Stop queue manager QM1 on client node A. You do this by using the following command:

      ```
      endmqm -s QM1
      ```

      Observe on client node B, queue manager QM1 running in active mode, and on client node A that queue manger QM1 is now stopped.
   b. Restart queue manager QM1 on client node A. You do this by using the following command:

      ```
      strmqm -x QM1
      ```

      Observe on client node A, queue manager QM1 running in standby mode, and on client node B, queue manager QM1 running in active mode.

What to do next

Create a multi-instance broker.

Related concepts:

"Creating a multi-instance queue manager"
Create a multi-instance queue manager by creating the queue manager on one server, and configuring WebSphere MQ on another server to use the shared queue manager data and logs.

Related tasks:

"Configuring a WebSphere Message Broker to run in multi-instance mode" on page 827
How you configure a WebSphere Message Broker to run in multi-instance mode.

"Creating the shared directories" on page 829
How you create the shared directories that you need for your multi-instance WebSphere Message Broker.

"Creating the multi-instance broker" on page 837
How you create the multi-instance broker.

Creating a multi-instance queue manager:

Create a multi-instance queue manager by creating the queue manager on one server, and configuring WebSphere MQ on another server to use the shared queue manager data and logs.

Most of the work of setting up a multi-instance queue manager involves setting up the network storage that holds the queue manager data and log files, and making the files available to other servers using network shares. These tasks need to be performed by someone with administrative authority, such as *root* on UNIX systems. Once the shares are set up, and a normal queue manager has been created using the shares for its queue manager data and logs, you only need to configure WebSphere MQ on the other servers. You do not create the queue manager again on the other servers.

File access control

You need to take care that the user and group `mqm` on all other servers have permission to access the shares.

On UNIX platforms, you need to make the `uid` and `gid` of `mqm` the same on all the systems. You might need to edit `/etc/passwd` on each system to set a common `uid` and `gid` for `mqm`, then reboot your UNIX systems.

On Microsoft Windows, you must install WebSphere MQ on a domain server, and create a user to own WebSphere MQ resources. The user must either be a member of domain group `mqm` or a member of another global domain group which is directly or indirectly a member of `mqm`. Make this user the owner of the shared queue manager and log files. The `sid` of the user who owns the queue manager and log files is then the same as the `sid` of the user that runs instances of the queue manager.

Configuration information

Configure as many queue manager instances as you need by modifying the WebSphere MQ configuration information about each server that has WebSphere MQ installed and is to run a queue manager instance. The commands, **dspmqinf** and **addmqinf** help you to configure the additional queue manager instances, or you can edit the `mqs.ini` file on UNIX servers directly. The topics, "Create a multi-instance queue manager on Linux" on page 834 and "Create a multi-instance queue manager on Windows Server" on page 836 are examples showing how to configure a multi-instance queue manager.

On UNIX systems, you can share a single `mqs.ini` file by placing it on the network share and setting the **AMQ_MQS_INI_LOCATION** environment variable to point to it.

On Microsoft Windows, WebSphere MQ has for a number of releases kept its configuration information in the Windows registry. In release level 7.0.1 (or later), some configuration information is moved from the registry into configuration files.

Multi-instance queue manager configuration is moved out of the registry into `qm.ini` and `qmstatus.ini` files, which are located in the queue manager data directory. The WebSphere MQ configuration information (`mqs.ini` on UNIX platforms) remains in the registry on Windows as does all the configuration information for existing queue managers and new queue managers that are configured to use the default data directory.

Restrictions

1. Configure multiple instances of the same queue manager only on servers having the same operating system, architecture (both machines having 32-bit or 64-bit word size, for example), and endian settings.
2. All WebSphere MQ installations must be at release level 7.0.1 (or later).
3. Typically active and standby installations are maintained at the same maintenance level. Consult the maintenance instructions for each upgrade to check if you must upgrade all installations together.
4. The network share that contains queue manager data and logs can only be shared between queue managers that are configured with the same WebSphere MQ user, group, and access control mechanism. For example, the network share set up on a Linux server could contain separate queue manager data and logs for AIX, Solaris, HP-UX, and Linux queue managers.

5. On UNIX systems, configure the shared file system on networked storage with a `hard`, interruptible, mount rather than a `soft` mount. A hard interruptible mount forces the queue manager to hang until it is interrupted by a system call. Soft mounts do not guarantee data consistency after a server crash.
6. On Microsoft Windows, the domain group `mqm` must have full access to shared WebSphere MQ log and data directories.
7. The shared log and data directories cannot be stored on a FAT, or an NFSv3 file system.
8. z/OS does not support multi-instance queue managers; use queue sharing groups.
9. Multi-instance queue managers are supported only on the platform on which they are created.

Related concepts:

"Create a multi-instance queue manager on Linux"
An example shows how to set up a multi-instance queue manager on Linux. The setup is small to illustrate the concepts involved. The example is based on Linux Red Hat Enterprise 5. The steps differ on other UNIX platforms.

"Create a multi-instance queue manager on Windows Server" on page 836
An example shows how to set up an example multi-instance queue manager on Windows Server. The setup is small and simple, to demonstrate the concepts involved, rather than being production scale. The example is based on Windows Server 2003. The steps might differ on other versions of Windows Server.

Related tasks:

"Creating the shared directories" on page 829
How you create the shared directories that you need for your multi-instance WebSphere Message Broker.

Create a multi-instance queue manager on Linux:

An example shows how to set up a multi-instance queue manager on Linux. The setup is small to illustrate the concepts involved. The example is based on Linux Red Hat Enterprise 5. The steps differ on other UNIX platforms.

The example is set up on a 2 GHz notebook computer with 3 GB RAM running Windows XP Service Pack 2. Two VMware virtual machines run Linux Red Hat Enterprise 5 in 640 MB images. A WebSphere MQ client is installed on an additional 400 MB VMware image that runs Windows XP Service Pack 2 and runs the sample high availability applications. All the virtual machines are configured as part of a VMware host-only network for security reasons. Although it is generally recommended not to use an automatically generated IP address, it is acceptable to use one for a short demonstration, because the IP address is unlikely to change.

Example

Table 10. Illustrative multi-instance queue manager configuration on Linux

Server 1	Server 2
Log in as *root*	
Follow the instructions in WebSphere MQ Quick Beginnings for Linuxto install WebSphere MQ, create the mqm user and group, and define `/var/mqm`.	
Carry out the task, "Verifying shared file system locking" on page 837 to check the file system supports multi-instance queue managers.	

Table 10. Illustrative multi-instance queue manager configuration on Linux (continued)

Server 1	Server 2
Check what `uid` and `gid` `/etc/passwd` displays for mqm, for example, `mqm:x:501:100:MQ User:/var/mqm:/bin/bash`	Match the `uid` and `gid` for mqm in `/etc/passwd` and reboot if you have had to change the values.
Create log and data directories in a common folder, `/MQHA`, that is to be shared. For example, 1. **mkdir** */MQHA* 2. **mkdir** */MQHA/logs* 3. **mkdir** */MQHA/qmgrs*	Create the folder, `/MQHA`, to mount the shared file system. Keep the path the same as on server 1; for example: 1. **mkdir** */MQHA*
Ensure that the `MQHA` directories are owned by user and group mqm, and the access permissions are set to `rwx` for user and group; for example **ls -al** displays, `drwxrwxr–x mqm mqm 4096 Nov 27 14:38 MQDATA` 1. **chown -R** `mqm:mqm` */MQHA* 2. **chmod -R** `ug+rwx` */MQHA*	
Create the queue manager: **crtmqm -ld** */MQHA/logs* **-md** */MQHA/qmgrs* **-q** *QM1*	
Add(The `'*'` allows all machines that can reach this one mount /MQHA for read/write. Restrict access on a production machine.) `/MQHA` `*(rw,sync,no_wdelay,fsid=0)` to `/etc/exports`	
Start the NFS daemon: */etc/init.d/***nfs** `start`	
Discover the host name or IP address of server 1: **ifconfig**(Try the command **/sbin/ifconfig** if the simple **ifconfig** command does not work.)	Mount the exported file system `/MQHA`: **mount -hard -intr -t** `nfs4` *192.168.217.130:/ /MQHA*

Copy the queue manager configuration details from Server 1,
dspmqinf -o `command QM1`
 	
and copy the result to the clip board,
`addmqinf -s QueueManager`
`-v Name=QM1`
`-v Directory=QM1`
`-v Prefix=/var/mqm`
`-v DataPath=/MQHA/qmgrs/QM1`
	Paste the queue manager configuration command into Server 2,
	addmqinf -s `QueueManager`
	-v `Name=QM1`
	-v `Directory=QM1`
	-v `Prefix=/var/mqm`
	-v `DataPath=/MQHA/qmgrs/QM1`

Start the queue manager instances, in either order, with the **-x** parameter: **strmqm -x** `QM1`	

Related concepts:

"Creating a multi-instance queue manager" on page 832
Create a multi-instance queue manager by creating the queue manager on one server, and configuring WebSphere MQ on another server to use the shared queue manager data and logs.

"Create a multi-instance queue manager on Windows Server" on page 836
An example shows how to set up an example multi-instance queue manager on Windows Server. The setup is small and simple, to demonstrate the concepts involved, rather than being production scale. The example is based on Windows Server 2003. The steps might differ on other versions of Windows Server.

Create a multi-instance queue manager on Windows Server:

An example shows how to set up an example multi-instance queue manager on Windows Server. The setup is small and simple, to demonstrate the concepts involved, rather than being production scale. The example is based on Windows Server 2003. The steps might differ on other versions of Windows Server.

Before you begin

On Windows, multi-instance queue managers must run on a domain controller. For this reason, and for this example, install two Windows servers, and configure them as domain controllers in the same domain.

The example was set up on a 2 GHz notebook computer with 3 GB RAM running Windows XP Service Pack 2. Two VMware virtual machines ran Windows Server 2003 Service Pack 2 in 500 MB images. A WebSphere MQ client was installed on an additional 400 MB VMware image that ran Windows XP Service Pack 2 to run the sample high availability applications. All the virtual machines were configured as part of a VMware host-only network for security reasons. Although it is recommended not to use automatically generated IP address, for a short demonstration it is satisfactory as the IP addresses ought not to change. As part of the configuration you need to update the TCP/IP properties of the virtual machines with the address of the domain server you have chosen to be the DNS server.

If you configure a queue manager to run as a service, you need to check that the user the service logs on as is a member of the local group `mqm`.

Example

Table 11. Illustrative multi-instance queue manager configuration on Windows Server 2003

Server 1	Server 2
Log in with user who is a member of the local group `mqm`.	

Create log and data directories in a folder on an NTFS drive, `c:\MQHA`, making sure that the owner is a member of `mqm`, and `mqm` has full-control authority to the folders.
`C:\MQHA\data`
`C:\MQHA\log`	

Create a share `MQHA` for `C:\MQHA`.	Connect to `\\`*`hostname`*`\MQHA`
Using WebSphere MQ Explorer, create a queue manager, `QM1`, overriding the default paths to the data and log folders and use UNC names to refer to the data and log folders. `\\`*`hostname`*`\MQHA\data` and `\\`*`hostname`*`\MQHA\log`.	
On a command-line run, **`dspmqinf -o`** `command QM1` and copy the result to the clip board, `addmqinf -s QueueManager` **`-v`** `Name=QM1` **`-v`** `Directory=QM1` **`-v`** `Prefix="C:\IBM\MQ\Data"` **`-v`** `DataPath="\\`*`hostname`*`\MQHA\data\QM1"`	Paste and run the command, **`addmqinf -s`** `QueueManager` **`-v`** `Name=QM1` **`-v`** `Directory=QM1` **`-v`** `Prefix="C:\IBM\MQ\Data"` **`-v`** `DataPath="\\`*`hostname`*`\MQHA\data\QM1"`
Using WebSphere MQ Explorer, start the manager instances with the **Permit a standby instance** check box selected.	

Related concepts:

"Creating a multi-instance queue manager" on page 832
Create a multi-instance queue manager by creating the queue manager on one server, and configuring WebSphere MQ on another server to use the shared queue manager data and logs.

"Create a multi-instance queue manager on Linux" on page 834
An example shows how to set up a multi-instance queue manager on Linux. The setup is small to illustrate the concepts involved. The example is based on Linux Red Hat Enterprise 5. The steps differ on other UNIX platforms.

Verifying shared file system locking:

Run the WebSphere MQ **amqmfsck** command to check obtaining and releasing file locks, and concurrent writing on UNIX systems.

Before you begin

Before you start:

You need a machine with networked storage, and two other machines connected to it that have WebSphere MQ installed. You need to have administrator (root) authority to configure the file system, and be a WebSphere MQ administrator to run **amqmfsck**.

About this task

Create the directory or directories on the networked storage that you are going to use to share queue manager data and logs. The directory owner needs to be a WebSphere MQ administrator, that is, a member of the `mqm` group on UNIX. The user who runs the tests must have WebSphere MQ administrator authority.

Use the example of exporting and mounting a file system in "Create a multi-instance queue manager on Linux" on page 834 to help you through configuring the file system. Different file systems require different configuration steps. Refer to the file system documentation.

Procedure

1. Export the shared directory on the networked storage system and start the NFS daemon.
2. Mount the exported directory on the two WebSphere MQ servers.
3. Run **amqmfsck**, without any options, on each system to check basic locking
4. Run **amqmfsck** on both WebSphere MQ systems simultaneously, using the **-c** option, to test writing to the directory concurrently.
5. Run **amqmfsck** on both WebSphere MQ systems at the same time, using the **-w** option, to test waiting for and releasing a lock on the directory concurrently.

Creating the multi-instance broker

How you create the multi-instance broker.

Before you begin

Before you start:

1. Create the WebSphere MQ multi-instance queue manager; see "Creating the WebSphere MQ multi-instance queue manager" on page 830.

2. Create the shared directories that you require for the multi-instance broker; see "Creating the shared directories" on page 829.

About this task

A multi-instance broker can be created by using the **mqsicreatebroker** command only. It is not possible to convert an existing broker to a multi-instance broker by using the **mqsichangebroker** command, nor is it possible to migrate a broker from a previous release to a Version 7.0 multi-instance broker.

Similarly, a broker instance can be created by using the **mqsiaddbrokerinstance** command only.

You can configure a multi-instance broker to start as a WebSphere MQ service.

The following procedure gives an overview of how you create a multi-instance broker:

Procedure

1. Create a multi-instance broker called MB1 on client node A. Do this by using the following command:

   ```
   mqsicreatebroker MB1 –q QM1 –e /SharedLocation/WMB
   ```

   On Windows 7 and Windows Server 2008 systems, to run the **mqsicreatebroker** command you must open a command console with elevated privileges. To open a command console with elevated privileges, use the **mqsicommandconsole** command. For more information, see "**mqsicommandconsole** command" on page 3830.

   Specify `-d defined` on the **mqsicreatebroker** command to start the multi-instance broker as a WebSphere MQ service.

   You must ensure that the SharedLocation exists, and that your user ID has access to the shared location before you run this command.

   See the "**mqsicreatebroker** command" on page 3831 for further information.
2. Add the details of broker MB1 onto client node B. Do this by using the following command:

   ```
   mqsiaddbrokerinstance MB1 –e /SharedLocation/WMB
   ```

   See "**mqsiaddbrokerinstance** command" on page 3715 for further information.

   Note that the preceding example is for a UNIX system.
3. Start broker MB1 on client node A. Do this by using the following command:

   ```
   mqsistart MB1
   ```

   Observe the broker becoming active. Note that this assumes that queue manager QM1 is active on client node A.
4. Start broker MB1 on client node B. Observe that broker MB1 is running in standby mode against the standby queue managerQM1. Do this by using the following command:

   ```
   mqsilist
   ```
5. Ensure that broker MB1 works as follows:
   a. Stop broker MB1 and queue manager QM1 on client node A. Observe on client node B that broker MB1 and queue manager QM1 change from standby to active mode.

b. Restart queue manager QM1 and broker MB1 on client node A. Observe on client node B that queue manager QM1 and broker MB1 return to standby mode.

Related tasks:

"Configuring a WebSphere Message Broker to run in multi-instance mode" on page 827
How you configure a WebSphere Message Broker to run in multi-instance mode.

"Creating the shared directories" on page 829
How you create the shared directories that you need for your multi-instance WebSphere Message Broker.

"Creating the WebSphere MQ multi-instance queue manager" on page 830
How you create the WebSphere MQ multi-instance queue manager that you need for your multi-instance WebSphere Message Broker.

"Configuring a WebSphere Message Broker as a WebSphere MQ service" on page 894
If you want to operate your WebSphere Message Broker as a WebSphere MQ service.

Related reference:

"**mqsicommandconsole** command" on page 3830
Use the **mqsicommandconsole** command to launch an elevated command console from which commands that require elevation on Windows can be run.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsiaddbrokerinstance** command" on page 3715
Use the **mqsiaddbrokerinstance** command to create a multi-instance broker on a server where WebSphere Message Broker has been installed.

Deleting a multi-instance broker

How you delete a multi-instance broker.

About this task

The order of deletion of a multi-instance message broker and its associated instances is important. You should take care to use the correct command for each step in the process.

Ideally, you should remove all of the broker instances using the **mqsiremovebrokerinstance** command before you attempt to remove the broker itself. This command removes all local references to the broker instance, but does not affect the shared configuration for the multi-instance broker on the shared work path (which was specified with the -e option on the **mqsicreatebroker** and **mqsiaddbrokerinstance** commands).

Note that the **mqsiremovebrokerinstance** command cannot be issued against a standby broker instance. Stop the active broker instance before you run this command.

Now, remove the multi-instance broker using the **mqsideletebroker** command. This process removes all references to the broker on both the local and shared work paths.

If you make a mistake, and unintentionally remove a broker instance, it can be re-created using the **mqsiaddbrokerinstance** command, providing that the **mqsideletebroker** command has not been used to delete the broker.

If the multi-instance broker has been removed using the **mqsideletebroker** command before removing any associated broker instances, it will not be possible to start the broker instances.

To recover from this situation, re-create the multi-instance broker using the **mqsicreatebroker** command with the -e option, specifying the original shared work path location.

The following procedure gives an overview of how you delete a multi-instance broker:

Procedure

1. Stop the multi-instance broker `MB1` on client node `A`.
2. Stop the multi-instance broker `MB1` on client node `B`.
3. Remove the broker instance `MB1` on client node `B`. Do this by using the following command:

   ```
   -mqsiremovebrokerinstance MB1
   ```

   See "**mqsiremovebrokerinstance** command" on page 3918 for further information.
4. Remove the broker `MB1` on client node `A`. Do this by using the following command:

   ```
   -mqsideletebroker MB1
   ```

   See "**mqsideletebroker** command" on page 3863 for further information.

   Note, that you must remove broker instances created by using the **mqsiaddbrokerinstance** command before you remove the main broker created using the **mqsicreatebroker** command.

Related tasks:

"Configuring a WebSphere Message Broker to run in multi-instance mode" on page 827
How you configure a WebSphere Message Broker to run in multi-instance mode.

"Creating the shared directories" on page 829
How you create the shared directories that you need for your multi-instance WebSphere Message Broker.

"Creating the WebSphere MQ multi-instance queue manager" on page 830
How you create the WebSphere MQ multi-instance queue manager that you need for your multi-instance WebSphere Message Broker.

Related reference:

"**mqsideletebroker** command" on page 3863
Use the **mqsideletebroker** command to delete a named broker. The command also deletes the queues on the associated queue manager (created when the broker was created). You can also specify that the queue manager is to be deleted.

"**mqsiremovebrokerinstance** command" on page 3918
Use the **mqsiremovebrokerinstance** command to remove a multi-instance broker from a server where WebSphere Message Broker has been installed.

Deleting a multi-instance queue manager

To delete a multi-instance queue manager completely, you need to use the WebSphere MQ **dltmqm** command to delete the queue manager, then remove instances from other servers using either the WebSphere MQ **rmvmqinf** or WebSphere MQ **dltmqm** commands.

Run the WebSphere MQ **dltmqm** command to delete a queue manager that has instances defined on other servers, on any server where that queue manager is defined. You do not need to run the WebSphere MQ **dltmqm** command on the same server that you created it on. Then run the WebSphere MQ **rmvmqinf** or WebSphere MQ **dltmqm** command on all the other servers which have a definition of the queue manager.

You can only delete a queue manager when it is stopped. At the time you delete it no instances are running, and the queue manager, strictly speaking, is neither a single or a multi-instance queue manager; it is simply a queue manager that has its queue manager data and logs on a remote share. When you delete a queue manager, its queue manager data and logs are deleted, and the queue manager stanza is removed from the `mqs.ini` file on the server on which you issued the WebSphere MQ **dltmqm** command. You need to have access to the network share containing the queue manager data and logs when you delete the queue manager. On Windows there is no `mqs.ini` file; instead, the queue manager stanza is removed from the registry.

On other UNIX platform servers where you have previously created instances of the queue manager there are also entries in the `mqs.ini` files on those servers. You need to visit each server in turn, and remove the queue manager stanza by running the WebSphere MQ command **rmvmqinf** *Queue manager stanza name*. On Windows, run the same command, and the queue manager stanza is removed from the registry.

On UNIX platforms, if you have placed a common `mqs.ini` file in network storage and referenced it from all the servers by setting the `AMQ_MQS_INI_LOCATION` environment variable on each server, you need to delete the queue manager from only one of its servers as there is only one `mqs.ini` file to update.

Example

First server

```
dltmqm QM1
```

Other servers where instances are defined

```
rmvmqinf QM1, or

dltmqm QM1
```

Related concepts:

"Creating a multi-instance queue manager" on page 832
Create a multi-instance queue manager by creating the queue manager on one server, and configuring WebSphere MQ on another server to use the shared queue manager data and logs.

Related tasks:

"Configuring a WebSphere Message Broker to run in multi-instance mode" on page 827
How you configure a WebSphere Message Broker to run in multi-instance mode.

"Creating the shared directories" on page 829
How you create the shared directories that you need for your multi-instance

WebSphere Message Broker.

Listing a multi-instance broker

How you list a multi-instance broker.

Procedure

Use the **mqsilist** command to view multi-instance brokers. Do this by using the following command:

```
mqsilist
```

The output from the **mqsilist** command is of the following format:

```
BIP1284I: Broker 'BK1' on queue manager 'QM1' is running.
BIP1292I: The multi-instance Broker 'BKHA1' on multi-instance queue manager
 'QMHA1' has stopped.
BIP1294I   Broker 'BKHA2' is a multi-instance broker running in standby mode
 on multi-instance queue manager 'QMHA2'.
BIP1295I Broker 'BKHA3' is a multi-instance broker running in active mode on
 multi-instance queue manager 'QMHA3'
BIP8071I: Successful command completion.
```

Results

The preceding results show:

- A standard broker `BK1` running
- A multi-instance broker `BKHA1` stopped
- A multi-instance broker `BKHA2` started in Standby mode
- A multi-instance broker `BKHA3` running in Active mode.

Related tasks:

"Configuring a WebSphere Message Broker to run in multi-instance mode" on page 827
How you configure a WebSphere Message Broker to run in multi-instance mode.

"Creating the shared directories" on page 829
How you create the shared directories that you need for your multi-instance WebSphere Message Broker.

"Creating the WebSphere MQ multi-instance queue manager" on page 830
How you create the WebSphere MQ multi-instance queue manager that you need for your multi-instance WebSphere Message Broker.

Related reference:

"**mqsilist** command" on page 3882
Use the **mqsilist** command to list installed brokers and their associated resources.

Back up and restore a multi-instance broker

How you back up and restore a multi-instance broker.

Before you begin

Before you start:

To ensure that the backup is complete and correct, back up the broker when the broker is stopped, or when it is not processing a configuration change (such as a deployment or property change).

About this task

When you back up a multi-instance broker, you back up the registry and configuration from the shared work path of the broker and its associated instances.

Hence you perform the backup procedure described in this topic for the multi-instance broker only. There are no additional steps for backing up the associated instances of the broker.

When you restore a multi-instance broker, you re-create associated instances by using the **mqsiaddbrokerinstance** command.

Procedure

1. Back up a multi-instance broker in the standard way by using the **mqsibackupbroker** command. Do this by using the following command:

   ```
   mqsibackupbroker MB1 –d /BackupDirectory/WMB
   ```

   where:

   - `MB1` is the name of the broker.
   - `/BackupDirectory/WMB` is the directory in which the backup file is created.

   The command detects a multi-instance broker and backs up the registry and configuration from the shared work path of the broker.

   There are no additional backup steps for the associated instances of the broker.
2. Restore a multi-instance broker in the standard way by using the **mqsirestorebroker** command. Do this by using the following command:

   ```
   mqsirestorebroker MB1 –d /BackupDirectory/WMB -a 20091009.zip
   ```

   where:

   - `MB1` is the name of the broker.
   - `/BackupDirectory/WMB` is the directory in which the backup file is stored.
   - `20091009.zip` is the name of the backup (archive) file.

   The command detects a multi-instance broker and restores the registry and configuration to the shared work path of the broker.
3. Re-create associated instances of the multi-instance broker by using the **mqsiaddbrokerinstance** command. See the "**mqsiaddbrokerinstance** command" on page 3715 for further information.

Related tasks:

"Configuring a WebSphere Message Broker to run in multi-instance mode" on page 827
How you configure a WebSphere Message Broker to run in multi-instance mode.

Related reference:

"**mqsibackupbroker** command" on page 3720
Use the **mqsibackupbroker** command to back up the current configuration of a broker.

"**mqsirestorebroker** command" on page 3952
Use the **mqsirestorebroker** command to restore the broker configuration from a backup file.

Using a broker with an existing high availability manager

You can use WebSphere Message Broker Version 7.0 with an existing high availability manager, for example HACMP, HA/XD, VCS, or HP-UX Serviceguard.

About this task

With the introduction of multi-instance message brokers, configuring WebSphere Message Broker Version 7.0 with a high availability (HA) manager is much easier. Previously, support pack IC91 was provided to assist in configuring this requirement.

A number of scripts were provided, but most of these are no longer required because WebSphere Message Broker Version 7.0 has built in many of the functions as a result of the multi-instance work.

This topic summarizes how to complete the following tasks:

1. Create a broker.
2. Add a broker instance.
3. Start a broker.
4. Stop a broker.
5. Monitor a broker.
6. Delete a broker.

Procedure

1. To create a broker, mount the shared resource onto your primary node and use the following **mqsicreatebroker** command with the **-e** parameter to specify your shared resource location.

   ```
   mqsicreatebroker MyBroker -q MQ1 -e /MQHA/MyBroker/
   ```

   where:

   - `MyBroker` is the name of the broker.
   - `MQ1` is the name of the queue manager.
   - `/MQHA/MyBroker/` is the directory for your shared resource.

2. To add another broker instance, mount the shared resource onto your secondary nodes and use the following **mqsiaddbrokerinstance** command.

   ```
   mqsiaddbrokerinstance MyBroker –e /MQHA/MyBroker/
   ```

   where:

   - `MyBroker` is the name of the broker.
   - `/MQHA/MyBroker/` is the directory for your shared resource.

3. To start a broker, you can use one of the following script files:

   hamqsi_start_broker

   ```
   #!/bin/ksh
   # Module:
   #   hamqsi_start_broker
   #
   # Args:
   #   BROKER = name of broker to start
   #
   # Description:
   #   This script attempts to start the MQSI Broker
   #
   #   Runs as the userid which runs broker, and must have
   #   the user's environment (i.e. invoke from "su - $MQUSER ..")
   #

   BROKER=$1

   if [ -z "$BROKER" ]
   then
   ```

```
  echo "hamqsi_start_broker: ERROR! No Broker name supplied"
  echo "   Usage: hamqsi_start_broker <BROKER>"
  exit 1
fi

# Ensure that the broker is not already running. In this test
# we look for any broker-related processes, which might have
# been left around after a previous failure. Any that remain
# must now be terminated. This is a brutal means of mopping
# up broker processes.
# We stop the processes in the following order:
#   bipservice       - first so it cannot issue restarts
#   bipbroker        - next for same reason
#   biphttplistener
#   startDataFlowEngine
#   DataFlowEngine  - last
#
echo "hamqsi_start_broker: Ensure $BROKER not already running"
for process in bipservice bipbroker biphttplistener startDataFlowEngine DataFlowEngine
do
  # Output of kill redirected to /dev/null in case no processes
  ps -ef | grep "$process $BROKER" | grep -v grep | \
     awk '{print $2}'| xargs kill -9 > /dev/null 2>&1
done


# Start the Broker
echo "hamqsi_start_broker: Start Broker " $BROKER
mqsistart $BROKER > /dev/null 2>&1
if [ $? -ne "0" ]
then
  echo "hamqsi_start_broker: Bad result from mqsistart for $BROKER"
  exit 1
fi

# Check to see if the broker service has started. This loop
# uses a fixed online timeout of approx. 10 seconds.
TIMED_OUT=yes
i=0
while [ $i -lt 10 ]
do
  # Check for Broker start. We look for bipservice and
  # bipbroker to be running; there might be no message flows
  # deployed.
  # Look to see whether bipservice is running
  cnt=`ps -ef | grep "bipservice $BROKER" | grep -v grep | wc -l`
  if [ $cnt -gt 0 ]
  then
    # Look to see whether bipbroker is running
    cnt=`ps -ef | grep "bipbroker $BROKER" | grep -v grep | wc -l`
    if [ $cnt -gt 0 ]
    then
      # Broker is online
      echo "hamqsi_start_broker: ${BROKER} is running"
      TIMED_OUT=no
      break  # out of timing loop
    fi
  fi
  # Manage the loop counter
  i=`expr $i + 1`
  sleep 1
done

# Report error if broker failed to start in time
if [ ${TIMED_OUT} = "yes" ]
then
  echo "hamqsi_start_broker: Broker service failed to start: " $BROKER
```

```
  exit 1
fi

exit 0
```

hamqsi_start_broker_as

```
#
#!/bin/ksh
# Module:
#   hamqsi_start_broker_as
#
# Args:
#   broker = broker name
#   qm     = name of broker queue manager
#   mquser = user account under which QM and Broker are run
#
# Description:
#   Starting an MQSI Broker requires the following services:
#   (1) The MQSeries Queue Manager which supports the Broker
#   (2) The MQSI Broker service
#   This script provides a single source to initiate the required
#   services in sequence.
#
#   Queue Manager:
#   This script uses the strmqm script supplied by MQSeries V7
#
#   Broker:
#   This script then invokes the hamqsi_start_broker script which
#   checks that the broker is fully stopped and then starts it.
#
#   The hamqsi_start_broker_as script should be run as root.



# Check running as root
if [ `id -u`  -ne 0 ]
then
  echo "Must be running as root"
  exit 1
fi


BROKER=$1
QM=$2
MQUSER=$3

# Check all parameters exist

if [ -z "$BROKER" ]
then
  echo "hamqsi_start_broker_as: ERROR! No Broker name supplied"
  echo "  Usage: hamqsi_start_broker_as <BROKER> <QM> <MQUSER>"
  exit 1
fi

if [ -z "$QM" ]
then
  echo "hamqsi_start_broker_as: ERROR! No queue manager name supplied"
  echo "  Usage: hamqsi_start_broker_as <BROKER> <QM> <MQUSER>"
  exit 1
fi

if [ -z "$MQUSER" ]
then
  echo "hamqsi_start_broker_as: ERROR! No Userid supplied"
  echo "  Usage: hamqsi_start_broker_as <BROKER> <QM> <MQUSER>"
  exit 1
```

```
fi

# --------------------------------------------------------------------
# Start the Queue Manager
#
echo "hamqsi_start_broker_as: Start Queue manager " $QM
su $MQUSER -c "/opt/mqm/bin/strmqm $QM"
rc=$?
if [ $rc -ne 0 ]
then
  echo "hamqsi_start_broker_as: Could not start the queue manager"
  exit $rc
fi

# --------------------------------------------------------------------
# Start the Broker
#
# Ensure that the Broker is not already running and start the Broker
su - $MQUSER -c "/MQHA/bin/hamqsi_start_broker $BROKER"
rc=$?
if [ $rc -ne 0 ]
then
  echo "hamqsi_start_broker_as: Could not start the broker"
  exit $rc
fi

exit $rc
```

4. To stop a broker, you can use one of the following script files:

hamqsi_stop_broker

```
#!/bin/ksh
# Module:
#   hamqsi_stop_broker
#
# Args:
#   broker = name of broker
#   timeout = max time to allow for each phase of termination
#
# Description:
#   This script stops the broker, forcibly if necessary.
#   The script should be run by the user account under which
#   the broker is run, including environment.

BROKER=$1
TIMEOUT=$2

if [ -z "$BROKER" ]
then
  echo "hamqsi_stop_broker: ERROR! No broker name supplied"
  echo "   Usage: hamqsi_stop_broker <BROKER> <TIMEOUT>"
  exit 1
fi

if [ -z "$TIMEOUT" ]
then
  echo "hamqsi_stop_broker: ERROR! No timeout supplied"
  echo "   Usage: hamqsi_stop_broker <BROKER> <TIMEOUT>"
  exit 1
fi

for severity in normal immediate terminate
do
  # Issue the stop method in the background - we don't
  # want to risk having it hang us up, indefinitely. We
  # want to be able to concurrently run a TIMEOUT timer
  # to give up on the attempt, and try a more forceful
  # stop. If the kill version fails then there is nothing
```

```
  # more we can do here anyway.

  echo "hamqsi_stop_broker: Attempting ${severity} stop of ${BROKER}"
  case $severity in

  normal)
    # Minimum severity of stop is to issue mqsistop
    mqsistop $BROKER > /dev/null 2>&1 &
    ;;

  immediate)
    # This is an immediate stop.
    mqsistop $BROKER -i > /dev/null 2>&1 &
    ;;

  terminate)
    # This is a brutal means of mopping up Broker processes.
    # We stop the processes in the following order:
    #   bipservice       - first so it cannot issue restarts
    #   bipbroker        - next for same reason
    #   biphttplistener
    #   startDataFlowEngine
    #   DataFlowEngine  - last
    for process in bipservice bipbroker biphttplistener startDataFlowEngine DataFlowEngine
    do
      # Output of kill redirected to /dev/null in case no processes
      ps -ef | grep "$process $BROKER" | grep -v grep | \
        awk '{print $2}'| xargs kill -9 > /dev/null 2>&1
    done
    ;;

  esac

  echo "hamqsi_stop_broker: Waiting for ${severity} stop of ${BROKER} to complete"
  TIMED_OUT=yes
  SECONDS=0
  while (( $SECONDS < ${TIMEOUT} ))
  do
    # See whether there are any broker processes still running
    cnt=`ps -ef | \
      grep -E "bipservice $BROKER|bipbroker $BROKER|startDataFlowEngine
      $BROKER|DataFlowEngine $BROKER|biphttplistener $BROKER" | \
      grep -v grep | wc -l`
    if [ $cnt -gt 0 ]
    then
      # It's still running...wait for timeout
      sleep 1 # loop granularity
    else
      # It's stopped, as desired
      echo "${BROKER} has stopped"
      TIMED_OUT=no
      break # out of while ..offline timeout loop
    fi
  done # timeout loop

  if [ ${TIMED_OUT} = "yes" ]
  then
    continue        # to next level of urgency
  else
    break           # instance is stopped, job is done
  fi

done # next level of urgency

if [ ${TIMED_OUT} = "no" ]
then
  echo "hamqsi_stop_broker: Completed"
```

```
  exit 0
else
  echo "hamqsi_stop_broker: Completed with errors"
  exit 1
fi
```

hamqsi_stop_broker_as

```
#!/bin/ksh
# Module:
#   hamqsi_stop_broker_as
#
# Arguments are:
#   broker = name of broker
#   qm     = name of broker queue manager
#   mquser = user account under which QM and broker run
#   timeout = max time to allow each phase of stop processing
#
# Description:
#   This script stops the Broker, Queue Manager in that sequence.
#
#   Broker:
#   The script invokes the hamqsi_stop_broker script to stop the
#   broker, which checks that the broker is fully stopped.
#
#   Queue Manager:
#   This script uses the strmqm script supplied by MQSeries V7
#
#   The hamqsi_stop_broker_as script should be run as root.


# Check running as root
if [ `id -u`  -ne 0 ]
then
  echo "Must be running as root"
  exit 1
fi


BROKER=$1
QM=$2
MQUSER=$3
TIMEOUT=$4

# Check all parameters

if [ -z "$BROKER" ]
then
  echo "hamqsi_stop_broker_as: ERROR! No Broker name supplied"
  echo "   Usage: hamqsi_stop_broker_as <BROKER> <QM> <MQUSER> <TIMEOUT>"
  exit 1
fi

if [ -z "$QM" ]
then
  echo "hamqsi_stop_broker_as: ERROR! No queue manager name supplied"
  echo "   Usage: hamqsi_stop_broker_as <BROKER> <QM> <MQUSER> <TIMEOUT>"
  exit 1
fi

if [ -z "$MQUSER" ]
then
  echo "hamqsi_stop_broker_as: ERROR! No userid supplied"
  echo "   Usage: hamqsi_stop_broker_as <BROKER> <QM> <MQUSER> <TIMEOUT>"
  exit 1
fi

if [ -z "$TIMEOUT" ]
```

```
then
  echo "hamsi_stop_broker_as: ERROR! No Timeout value supplied"
  echo "   Usage: hamqsi_stop_broker_as <BROKER> <QM> <MQUSER> <TIMEOUT>"
 exit 1
fi


METHOD_STATUS="OK"

# ------------------------------------------------------------------
# Stop the BROKER
#
echo "hamqsi_stop_broker_as: Stop Broker " $BROKER
su - $MQUSER -c "/MQHA/bin/hamqsi_stop_broker $BROKER $TIMEOUT"
if [ $? -ne "0" ]
then
  # Even if the above operation failed, just report and then continue by
  # stopping other components
  echo "hamqsi_stop_broker_as: Attempt to stop broker $BROKER failed"
  METHOD_STATUS="Error"
fi

# ------------------------------------------------------------------
# Stop the Queue Manager, using script from MQ V7
#
echo "hamqsi_stop_broker_as: Stop Queue Manager $QM"
su $MQUSER -c "/opt/mqm/bin/endmqm -i $QM"
if [ $? -ne "0" ]
then
  # Even if the above operation failed, just report and then continue by
  # stopping other components
  echo "hamqsi_stop_broker_as: Attempt to stop queue manager $QM failed"
  METHOD_STATUS="Error"
fi

if [ ${METHOD_STATUS} = "OK" ]
then
  exit 0
else
  echo "hamqsi_stop_broker_as: Completed with errors"
  exit 1
fi
```

5. To monitor a broker, you can use the following script file that checks only for the existence of the main broker processes and provides a successful return code if they are found:

hamqsi_monitor_broker_as

```
#!/bin/ksh
# Module:
#   hamqsi_monitor_broker_as
#
# Args:
#   BROKER = name of broker in AppServer
#   QM     = name of queue manager in AppServer
#   MQUSER = userid under which queue manager and broker run
#
# Description:
#   This is the application monitor script used with HACMP/ES. It
#   needs to be invoked by a parameter-less wrapper script because
#   HACMP does not allow parameters to be passed to application
#   monitor scripts.
#
#   This hamqsi_monitor_broker_as script is run as root, and uses
#   su as needed to monitor the 3 components of the application server.
#
#   This script is tolerant of a queue manager that is still in
```

```
#   startup. If the queue manager is still starting this application
#   monitor script will exit with 0 - which indicates
#   to HACMP that there's nothing wrong. This is to allow for
#   startup time for the queue manager which might exceed the
#   Stabilisation Interval set for the Application Monitor in HACMP/ES.
#
#
# Exit codes:
#   0  => Broker & QM are all running OK or starting
#   >0 => One or more components are not responding.
#

# Check running as root
if [ `id -u`  -ne 0 ]
then
  echo "Must be running as root"
  exit 1
fi

BROKER=$1
QM=$2
MQUSER=$3

# Check the parameters

if [ -z "$BROKER" ]
then
  echo "hamqsi_monitor_broker_as: ERROR! No broker name supplied"
  exit 1
fi

if [ -z "$QM" ]
then
  echo "hamqsi_monitor_broker_as: ERROR! No queue manager name supplied"
  exit 1
fi

if [ -z "$MQUSER" ]
then
  echo "hamqsi_monitor_broker_as: ERROR! No mquser supplied"
  exit 1
fi

# Use a state variable to reflect the state of components as they
# are tested. Valid values are "stopped", "starting" and "started"
# Initialise it to "stopped" for safety.
STATE="stopped"

# ------------------------------------------------------------------
# Check that the queue manager is running or starting.
#
su - $MQUSER -c "echo 'ping qmgr' | runmqsc ${QM}" > /dev/null 2>&1
pingresult=$?
# pingresult will be 0 on success; non-zero on error (man runmqsc)
if [ $pingresult -eq 0 ]
then
  # ping succeeded
  echo "hamqsi_monitor_broker_as: Queue Manager ${QM} is responsive"
  STATE="started"
else
  # ping failed
  # Don't condemn the QM immediately, it might be in startup.
  # The following regexp includes a space and a tab, so use tab-friendly
  # editors.
  srchstr=" $QM[  ]*$"
  cnt=`ps -ef | grep strmqm | grep "$srchstr" | grep -v grep \
                | awk '{print $2}' | wc -l`
```

```
  if [ $cnt -gt 0 ]
  then
    # It appears that QM is still starting up, tolerate
    echo "hamqsi_monitor_broker_as: Queue Manager ${QM} is starting"
    STATE="starting"
  else
    # There is no sign of QM start process
    echo "hamqsi_monitor_broker_as: Queue Manager ${QM} is not responsive"
    STATE="stopped"
  fi
fi


# Decide whether to continue or to exit
case $STATE in
  stopped)
    echo "hamqsi_monitor_broker_as: Queue manager ($QM) is not running correctly"
    exit 1
    ;;
  starting)
    echo "hamqsi_monitor_broker_as: Queue manager ($QM) is starting"
    echo "hamqsi_monitor_broker_as: WARNING - Stabilisation Interval might be too short"
    echo "hamqsi_monitor_broker_as: WARNING - No test of broker $BROKER will be conducted"
    exit 0
    ;;
  started)
    echo "hamqsi_monitor_broker_as: Queue manager ($QM) is running"
    continue
    ;;
esac

# ------------------------------------------------------------------
# Check the MQSI Broker is running
#
# Re-initialise STATE for safety
STATE="stopped"
#
# The broker runs as a process called bipservice which is responsible
# for starting and re-starting the admin agent process (bipbroker).
# The bipbroker is responsible for starting any DataFlowEngines. The
# bipbroker starts the DataFlowEngines using the wrapper script
# startDataFlowEngine. If no execution groups have been assigned to
# the broker there will be no DataFlowEngine processes. There should
# always be a bipservice and bipbroker process pair. This monitor
# script only tests for bipservice, because bipservice should restart
# bipbroker if necessary - the monitor script should not attempt to
# restart bipbroker and it might be premature to report an absence
# of a bipbroker as a failure.
#
cnt=`ps -ef | grep "bipservice $BROKER" | grep -v grep | wc -l`
if [ $cnt -eq 0 ]
then
  echo "hamqsi_monitor_broker_as: MQSI Broker $BROKER is not running"
  STATE="stopped"
else
  echo "hamqsi_monitor_broker_as: MQSI Broker $BROKER is running"
  STATE="started"
fi

# Decide how to exit
case $STATE in
  stopped)
    echo "hamqsi_monitor_broker_as: Broker ($BROKER) is not running correctly"
    exit 1
    ;;
  started)
```

```
        echo "hamqsi_monitor_broker_as: Broker ($BROKER) is running"
        exit 0
      ;;
esac
```

If you require more information than that supplied by the preceding example, you can code your monitor to do the following actions:

- Subscribe to WebSphere Message Broker accounting and statistics, and analyze the results.
- Put a dummy message through the broker and analyze the results.

6. Before you delete the broker on the primary node, you must delete any brokers on the standby nodes. To delete a broker, use the **mqsiremovebrokerinstance** command on the secondary nodes, and the **mqsideletebroker** command on the primary node.

Results

For further information about configuring WebSphere MQ with a high availability manager, refer to the WebSphere MQ Version 7 Information Center online.

See the:

- **addmqinf** command
- **dspmqinf** command
- **rmvmqinf.** command

and the **-md** option on the **crtmqm** command.

Related tasks:

"Configuring a WebSphere Message Broker to run in multi-instance mode" on page 827
How you configure a WebSphere Message Broker to run in multi-instance mode.

Related reference:

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsiaddbrokerinstance** command" on page 3715
Use the **mqsiaddbrokerinstance** command to create a multi-instance broker on a server where WebSphere Message Broker has been installed.

"**mqsiremovebrokerinstance** command" on page 3918
Use the **mqsiremovebrokerinstance** command to remove a multi-instance broker from a server where WebSphere Message Broker has been installed.

"**mqsideletebroker** command" on page 3863
Use the **mqsideletebroker** command to delete a named broker. The command also deletes the queues on the associated queue manager (created when the broker was created). You can also specify that the queue manager is to be deleted.

Related information:

WebSphere MQ Version 7 Information Center online

Using a broker with an existing Windows Cluster (Windows Server 2003 or 2008)

You can use WebSphere Message Broker Version 7.0 with the existing high availability manager for Windows Server 2003 (Microsoft Cluster Service - MSCS) or Windows Server 2008 (Failover Cluster Manager).

About this task

With the introduction of multi-instance brokers, configuring WebSphere Message Broker Version 7.0 with a high availability (HA) manager is much easier. Previously, support pack IC91 was provided to assist in configuring this requirement.

A number of scripts were provided in support pack IC91, but most of these scripts are no longer required because WebSphere Message Broker Version 7.0 has built-in many of the functions as a result of the multi-instance work.

This topic summarizes how to complete the following tasks:

1. Complete the prerequisite setup.
2. Configure a local group.
3. Create a broker.
4. Add broker instances to the additional nodes.
5. Add the broker service to the cluster configuration
6. Start and stop a broker.
7. Delete a broker.

Procedure

1. To complete the prerequisite setup, complete the following steps:
   a. Validate your MSCS configuration (on Windows Server 2003) or Failover Cluster Manager configuration (on Windows Server 2008).
   b. Complete all of the steps documented in 'Supporting the Microsoft Cluster Service' of the WebSphere MQ documentation to create a cluster configuration that contains the queue manager on which you want the broker to run.
   c. Note the `haregtyp.exe` command as a prerequisite for creating WebSphere MQ resources in the cluster.
2. To configure a local group, ensure that the domain user under which you want the broker to run exists in the local *mqbrkrs* group on all nodes on which you want the broker to run.
3. To create a broker, use the following **mqsicreatebroker** command on the node on which your cluster is currently running.

   ```
   mqsicreatebroker MyBroker -q MQ1 -e E:\Broker\Workspace
   ```

   where:
   - `MyBroker` is the name of the broker.
   - `MQ1` is the name of the queue manager.
   - `E:\Broker\Workspace` is the directory of a shared disk (nonquorum) in your cluster configuration.
4. To add this broker instance to the other nodes in your cluster, switch your cluster to each node in turn. When a node is active, use the following **mqsiaddbrokerinstance** command.

   ```
   mqsiaddbrokerinstance MyBroker –e E:\Broker\Workspace
   ```

   where:
   - `MyBroker` is the name of the broker.
   - `E:\Broker\Workspace` is the directory of a shared disk (nonquorum) in your cluster configuration.

5. To add a broker generic service resource to the cluster which contains the broker queue manager, complete the following steps:
   a. Select the IBM WebSphere Message Broker component `MyBroker` service when asked. All other settings can be left unchanged.
   b. Add a dependency on the WebSphere MQ resource, to ensure that the queue manager is started before the broker.
6. To start and stop the broker resource, use the MSCS configuration (on Windows Server 2003) or Failover Cluster Manager (on Windows Server 2008).
7. To delete a broker resource from the MSCS configuration (on Windows Server 2003) or Failover Cluster Manager (on Windows Server 2008), complete the following steps:
   a. Delete the broker generic service from the cluster configuration.
   b. Use the **mqsiremovebrokerinstance** command on all nodes but one, moving the cluster between nodes before running each command.
   c. On the final node, use the **mqsideletebroker** command to completely remove the broker configuration.

Results

For further information about configuring WebSphere MQ with a high availability manager for Windows Server 2003 or Windows Server 2008, see the WebSphere MQ Version 7 Information Center online.

See the following topics:

- Introducing MSCS clusters
- Supporting the Microsoft Cluster Service (MSCS)
- Setting up WebSphere MQ for MSCS clustering

Related tasks:

"Using a broker with an existing high availability manager" on page 843
You can use WebSphere Message Broker Version 7.0 with an existing high availability manager, for example HACMP, HA/XD, VCS, or HP-UX Serviceguard.

"Configuring a WebSphere Message Broker to run in multi-instance mode" on page 827
How you configure a WebSphere Message Broker to run in multi-instance mode.

Related reference:

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsiaddbrokerinstance** command" on page 3715
Use the **mqsiaddbrokerinstance** command to create a multi-instance broker on a server where WebSphere Message Broker has been installed.

"**mqsiremovebrokerinstance** command" on page 3918
Use the **mqsiremovebrokerinstance** command to remove a multi-instance broker from a server where WebSphere Message Broker has been installed.

"**mqsideletebroker** command" on page 3863
Use the **mqsideletebroker** command to delete a named broker. The command also deletes the queues on the associated queue manager (created when the broker was created). You can also specify that the queue manager is to be deleted.

Related information:

WebSphere MQ Version 7 Information Center online

HTTP proxy servlet overview

The HTTP proxy servlet is a Java servlet that you can use in an external web servlet container, such as IBM WebSphere Application Server or Apache Tomcat, to receive HTTP requests from web services client applications, to replace the support provided by the broker and execution group HTTP listeners.

By using the HTTP proxy servlet in an external web servlet container, you can support a larger number of concurrent HTTP sessions, high availability, load distribution, and access to the broker from multiple IP addresses and ports.You can use the servlet to replace either the broker-wide listener, or an embedded listener in a specific execution group.

The HTTP proxy servlet supports SSL (HTTPS) secure protocol when it is deployed in a properly configured web servlet container.

You cannot use the HTTP proxy servlet if you configure your broker environment to use multi-instance WebSphere MQ queue managers; the servlet cannot connect to the standby queue manager when it becomes active.

For a detailed description of the proxy servlet, its location in the runtime directory path, and its function, see: "HTTP proxy servlet; proxy servlet component" on page 865

Before you install and test the HTTP proxy servlet, ensure you understand the following concepts:

- "HTTP traffic handling in WebSphere Message Broker" on page 857
- "HTTP traffic handling by using the proxy servlet in an external web servlet container" on page 859
- "HTTP proxy servlet; descriptions of required components" on page 863

When you have gained an understanding of the proxy servlet concept, read the following topics to help you install and test the HTTP proxy servlet:

- "Installing the proxy servlet" on page 872
- "Testing the proxy servlet" on page 892

Related concepts:

"HTTP traffic handling in WebSphere Message Broker" on page 857
Before reading about HTTP traffic handling using the proxy servlet in an external Web container, ensure that you understand the basic structure of HTTP traffic handling in WebSphere Message Broker.

"HTTP traffic handling by using the proxy servlet in an external web servlet container" on page 859
Before you install and test the HTTP proxy servlet, ensure that you understand the structure of HTTP traffic handling by using the proxy servlet in an external web servlet container, such as IBM WebSphere Application Server or Apache Tomcat.

"HTTP proxy servlet; descriptions of required components" on page 863
The HTTP proxy servlet requires a number of components such as message flows, a proxy servlet, a servlet container, web addresses, the Broker component, and web services clients.

Related tasks:

"Installing the proxy servlet" on page 872
The installation of the proxy servlet is made up of a number of tasks. Use this topic to complete each of the tasks to then move onto testing the proxy servlet.

"Testing the proxy servlet" on page 892
Test the proxy servlet with a message broker that receives HTTP requests and passes them to a message flow.

"Configuring a WebSphere Message Broker to run in multi-instance mode" on page 827
How you configure a WebSphere Message Broker to run in multi-instance mode.

Related reference:

"Proxy servlet configuration parameters" on page 878
Before you can deploy the proxy servlet `web.xml` to the servlet container, you must configure it with the following initialization parameters for the broker environment that the servlet connects to.

HTTP traffic handling in WebSphere Message Broker

Before reading about HTTP traffic handling using the proxy servlet in an external Web container, ensure that you understand the basic structure of HTTP traffic handling in WebSphere Message Broker.

WebSphere Message Broker supports Web services requests over HTTP. Brokers have HTTP listeners that receive requests, which are propagated to message flows that contain the following nodes:

- HTTPInput
- HTTPReply
- SOAPInput
- SOAPReply

WebSphere Message Broker has two types of listeners:

- A listener that is started and managed by the broker, which you can use for messages for the HTTP nodes. This listener supports two ports, one for HTTP and one for HTTPS (HTTP over SSL) messages.
- A listener that is started and managed by an execution group (an embedded listener), which you can use for messages for the SOAP and HTTP nodes. This listener also supports two ports for HTTP and HTTPS messages.

HTTP traffic handling is based on node type:

- HTTP nodes: The default configuration for HTTP nodes is for the broker listener to handle HTTP messages through port `7080`. Connections are made into the broker HTTP listener, which places requests on the SYSTEM.BROKER.WS.INPUT queue, from which the HTTPInput nodes read the data. After the flow has reached an HTTPReply node, the reply data is placed on the SYSTEM.BROKER.WS.REPLY queue, where the data is read by the HTTP listener and sent back to the HTTP client.

  If you want HTTP nodes to handle HTTPS messages, you must update the broker configuration to define a second port for HTTPS, and set the property to enable HTTPS processing, by using the **mqsichangeproperties** command. You can also change the port or ports on which the listener is listening by using this command.

  Because WebSphere MQ queues are used to couple the HTTP listener to the message flows, requests received by the HTTP listener can be processed by any message flow in any execution group (provided that the Web address selector matches), and the reply can come back from any execution group.

  You can also use the embedded execution group listener to process HTTP and HTTPS messages. You can change the configuration and port numbers by using the **mqsichangeproperties** command.

Connections are made directly to the execution groups, and requests are passed to the HTTPInput nodes in that execution group. The HTTP data is passed to the HTTPInput node, processed in the flow, and sent back directly from the HTTPReply node.

Because the network connection is made to a particular execution group, the reply must be sent back from that execution group.

- SOAP nodes: Connections are made directly to the execution group listener, and requests are passed to the SOAPInput nodes in that execution group. The HTTP data is passed to the SOAPInput node, processed in the flow, and sent back directly from the SOAPReply node.

  Because the network connection is made to a particular execution group, the reply must be sent back from that execution group.

For more details about these listeners, see "Processing HTTP messages" on page 1579.

Before you install and test the HTTP proxy servlet, ensure that you understand the following concepts:

- "HTTP proxy servlet overview" on page 856
- "HTTP traffic handling by using the proxy servlet in an external web servlet container" on page 859
- "HTTP proxy servlet; descriptions of required components" on page 863

When you have gained an understanding of the proxy servlet concept, read the following topics to help you to install and test the HTTP proxy servlet:

- "Installing the proxy servlet" on page 872
- "Testing the proxy servlet" on page 892

Related concepts:

"HTTP proxy servlet overview" on page 856
The HTTP proxy servlet is a Java servlet that you can use in an external web servlet container, such as IBM WebSphere Application Server or Apache Tomcat, to receive HTTP requests from web services client applications, to replace the support provided by the broker and execution group HTTP listeners.

"HTTP traffic handling by using the proxy servlet in an external web servlet container" on page 859
Before you install and test the HTTP proxy servlet, ensure that you understand the structure of HTTP traffic handling by using the proxy servlet in an external web servlet container, such as IBM WebSphere Application Server or Apache Tomcat.

"HTTP proxy servlet; descriptions of required components" on page 863
The HTTP proxy servlet requires a number of components such as message flows, a proxy servlet, a servlet container, web addresses, the Broker component, and web services clients.

"HTTP listeners" on page 1589
You can choose between broker-wide listeners and execution group (embedded) listeners to manage HTTP messages in your HTTP flows. Learn about the two types of listener, how ports are assigned to them, and how you can switch from one to the other for individual execution groups.

Related tasks:

"Installing the proxy servlet" on page 872
The installation of the proxy servlet is made up of a number of tasks. Use this topic to complete each of the tasks to then move onto testing the proxy servlet.

"Testing the proxy servlet" on page 892
Test the proxy servlet with a message broker that receives HTTP requests and passes them to a message flow.

Related reference:

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"HTTPInput node" on page 4474
Use the HTTPInput node to receive an HTTP message from an HTTP client for processing by a message flow.

"HTTPReply node" on page 4484
Use the HTTPReply node to return a response from the message flow to an HTTP client. This node generates the response to an HTTP client from which the input message was received by the HTTPInput node, and waits for confirmation that it has been sent.

"SOAPInput node" on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

"SOAPReply node" on page 4819
Use the SOAPReply node to send SOAP messages from the broker to the originating client in response to a message received by a SOAPInput node.

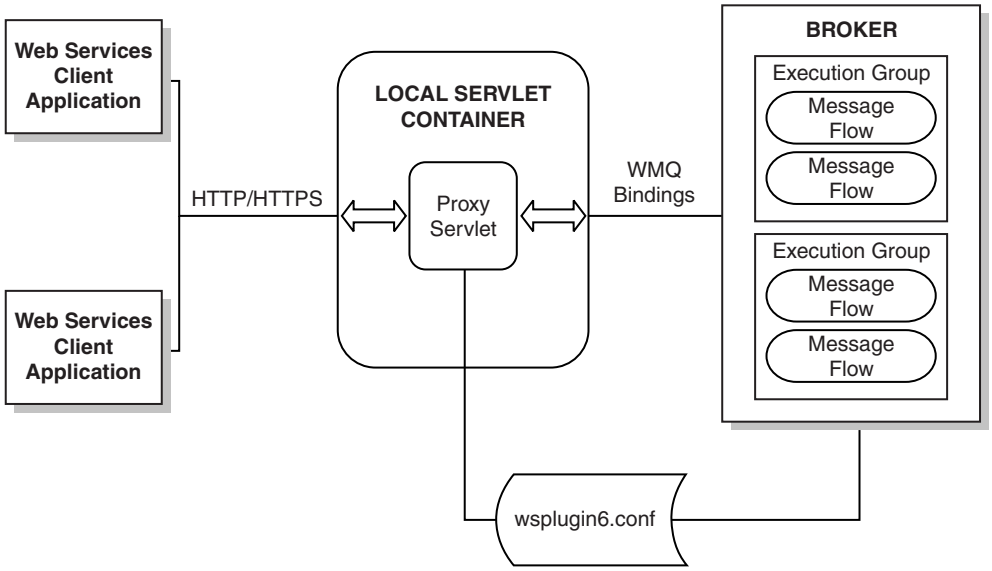
HTTP traffic handling by using the proxy servlet in an external web servlet container

Before you install and test the HTTP proxy servlet, ensure that you understand the structure of HTTP traffic handling by using the proxy servlet in an external web servlet container, such as IBM WebSphere Application Server or Apache Tomcat.

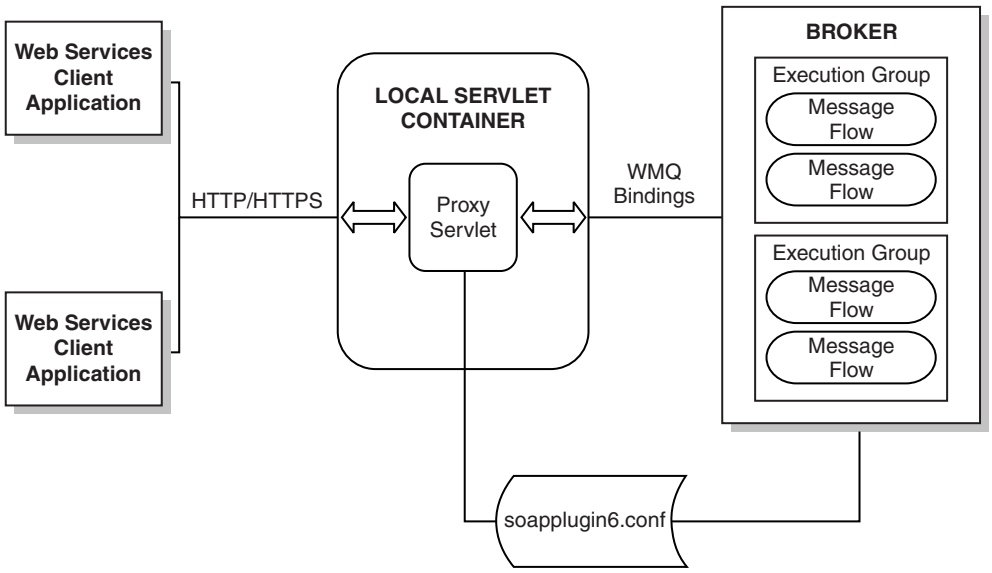
The proxy servlet is a Java servlet that produces the function of the broker HTTP listeners in an external web servlet container, such as WebSphere Application Server or Apache Tomcat. After the proxy servlet is deployed and running on the web servlet container, it uses the HTTP listener of the container to receive HTTP requests. If the web servlet container is configured to support SSL (HTTPS), web services requests are received by the message flows by using a secure communications protocol. For more information, see: "HTTP proxy servlet; proxy servlet component" on page 865

Components and configurations supported by the proxy servlet:

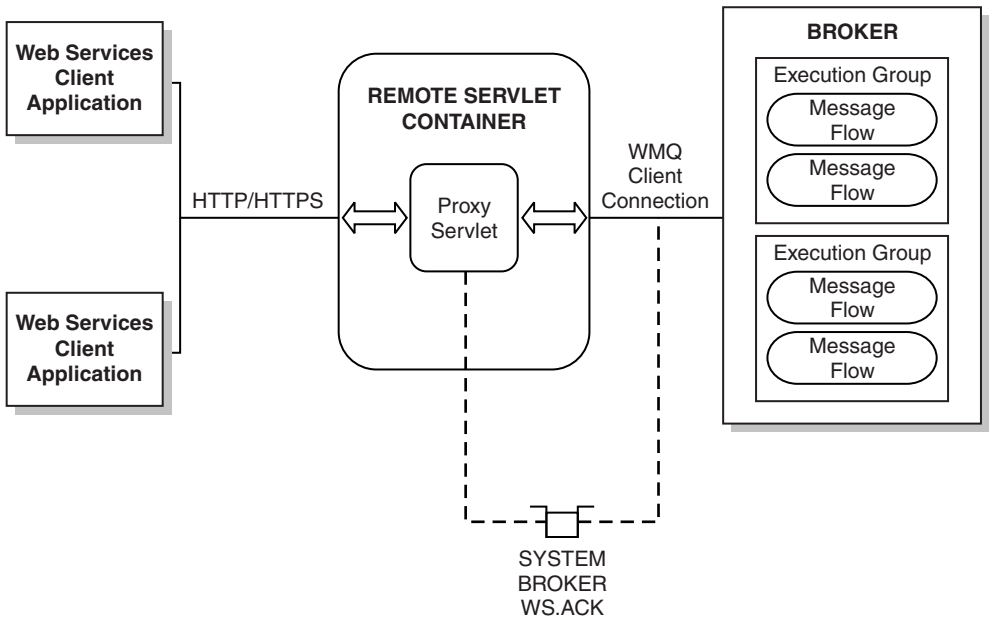
The following figures show the components and configurations that are supported by the proxy servlet, for descriptions of the components listed in these figures, see: "HTTP proxy servlet; descriptions of required components" on page 863



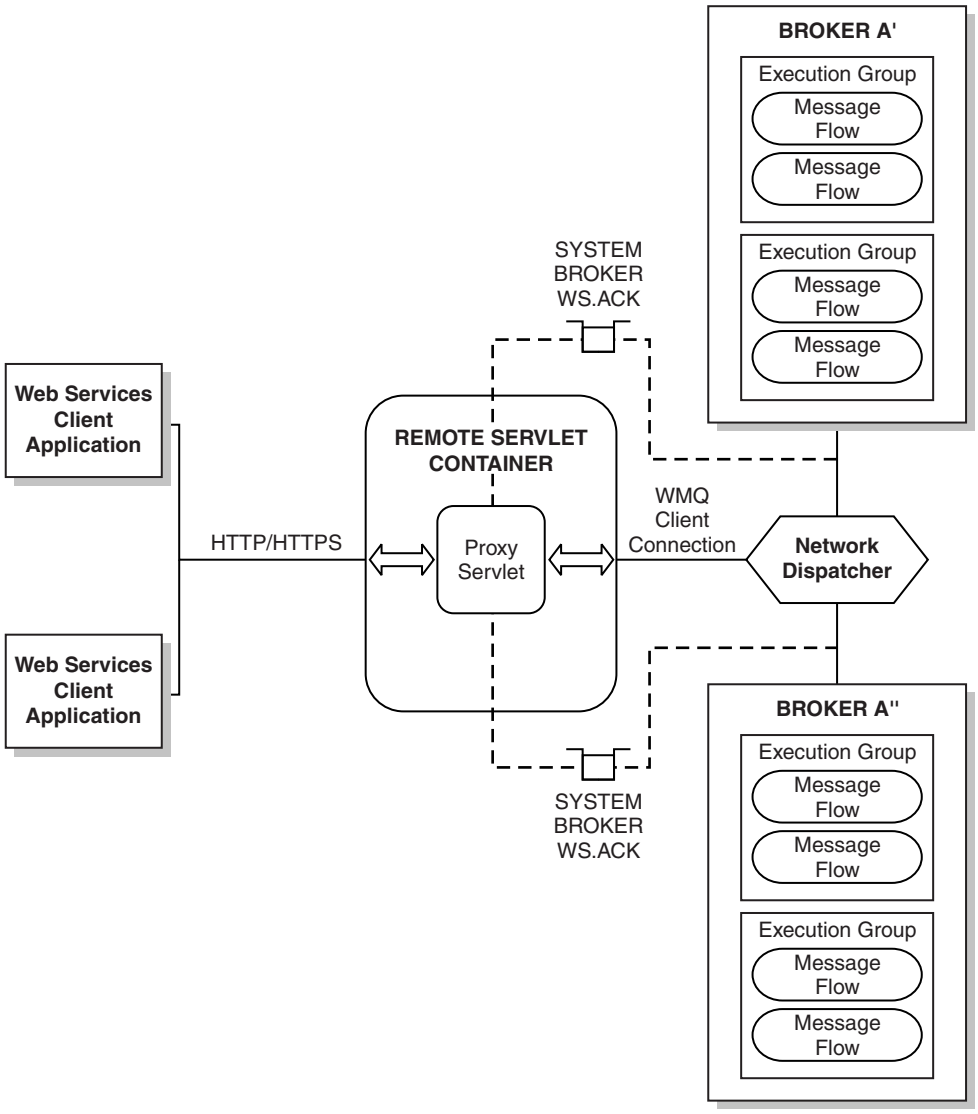
In the above figure the proxy servlet is running on the same server as the message broker. The proxy servlet connects to the broker queue manager by using bindings mode (local connection) and the servlet is configured to access only HTTP nodes and therefore has been configured to access the HTTP nodes configuration file `wsplugin6.conf`.



In the above figure the proxy servlet is running on the same server as the message broker. The proxy servlet connects to the broker queue manager by using bindings mode (local connection) and the servlet is configured to access only SOAP nodes and therefore has been configured to access the SOAP nodes configuration file `soapplugin6.conf`.



In the above figure the proxy servlet is running on a remote server to the message broker. The proxy servlet connects to the broker queue manager by using client mode (remote connection) and the servlet can be configured to access HTTP or SOAP nodes and the HTTP or SOAP node configuration is retrieved from the SYSTEM.BROKER.WS.ACK queue.



In the above figure the proxy servlet is configured to load balance WebSphere MQ connections across multiple message brokers. Network dispatchers or load balancers are required for this configuration to work.

When the proxy servlet is configured to connect to multiple brokers, the brokers must be identical clones of each other, which means that the same HTTP and SOAP flows are deployed with the same web addresses.

The proxy servlet sends the HTTP requests over the WebSphere MQ connections trying to distribute the load between the active broker connections.

Before you install and test the HTTP proxy servlet, ensure you understand the following concepts:

- "HTTP proxy servlet overview" on page 856
- "HTTP traffic handling in WebSphere Message Broker" on page 857

When you have gained an understanding of the proxy servlet concept, read the following topics to help you install and test the HTTP proxy servlet:

- “Installing the proxy servlet” on page 872
- “Testing the proxy servlet” on page 892

Related concepts:

“HTTP proxy servlet overview” on page 856
The HTTP proxy servlet is a Java servlet that you can use in an external web servlet container, such as IBM WebSphere Application Server or Apache Tomcat, to receive HTTP requests from web services client applications, to replace the support provided by the broker and execution group HTTP listeners.

“HTTP traffic handling in WebSphere Message Broker” on page 857
Before reading about HTTP traffic handling using the proxy servlet in an external Web container, ensure that you understand the basic structure of HTTP traffic handling in WebSphere Message Broker.

“HTTP proxy servlet; descriptions of required components”
The HTTP proxy servlet requires a number of components such as message flows, a proxy servlet, a servlet container, web addresses, the Broker component, and web services clients.

Related tasks:

“Installing the proxy servlet” on page 872
The installation of the proxy servlet is made up of a number of tasks. Use this topic to complete each of the tasks to then move onto testing the proxy servlet.

“Testing the proxy servlet” on page 892
Test the proxy servlet with a message broker that receives HTTP requests and passes them to a message flow.

HTTP proxy servlet; descriptions of required components

The HTTP proxy servlet requires a number of components such as message flows, a proxy servlet, a servlet container, web addresses, the Broker component, and web services clients.

Ensure that you are familiar with the following components and concepts required by the proxy servlet:

- “HTTP proxy servlet; message flows component” on page 864
- “HTTP proxy servlet; proxy servlet component” on page 865
- “HTTP proxy servlet; servlet container component” on page 867
- “HTTP proxy servlet; web addresses component” on page 868
- “HTTP proxy servlet; Broker component” on page 870
- “HTTP proxy servlet; web services clients component” on page 871

Before you install and test the HTTP proxy servlet, ensure you understand the following concepts:

- “HTTP proxy servlet overview” on page 856
- “HTTP traffic handling in WebSphere Message Broker” on page 857
- “HTTP traffic handling by using the proxy servlet in an external web servlet container” on page 859

When you have gained an understanding of the proxy servlet concept, read the following topics to help you install and test the HTTP proxy servlet:

- “Installing the proxy servlet” on page 872
- “Testing the proxy servlet” on page 892

Related concepts:

"HTTP proxy servlet overview" on page 856
The HTTP proxy servlet is a Java servlet that you can use in an external web servlet container, such as IBM WebSphere Application Server or Apache Tomcat, to receive HTTP requests from web services client applications, to replace the support provided by the broker and execution group HTTP listeners.

"HTTP traffic handling in WebSphere Message Broker" on page 857
Before reading about HTTP traffic handling using the proxy servlet in an external Web container, ensure that you understand the basic structure of HTTP traffic handling in WebSphere Message Broker.

"HTTP traffic handling by using the proxy servlet in an external web servlet container" on page 859
Before you install and test the HTTP proxy servlet, ensure that you understand the structure of HTTP traffic handling by using the proxy servlet in an external web servlet container, such as IBM WebSphere Application Server or Apache Tomcat.

Related tasks:

"Installing the proxy servlet" on page 872
The installation of the proxy servlet is made up of a number of tasks. Use this topic to complete each of the tasks to then move onto testing the proxy servlet.

"Testing the proxy servlet" on page 892
Test the proxy servlet with a message broker that receives HTTP requests and passes them to a message flow.

HTTP proxy servlet; message flows component:

The HTTP proxy servlet requires a number of components such as message flows, a proxy servlet, a servlet container, web addresses, the Broker component, and web services clients. Ensure that you are familiar with the message flows component.

Message flows execute the functions of transforming, logging, enriching, and routing messages. Message flows can use HTTP or SOAP input nodes to receive requests and HTTP or SOAP reply nodes to send responses. Each HTTP or SOAP input node has a web address configured in the node properties. The HTTP or SOAP input nodes only receive request messages addressed to the configured web address.

The proxy servlet is not aware of the execution groups that the message flows are deploying.

Before you install and test the HTTP proxy servlet, ensure you understand the following concepts:

- "HTTP proxy servlet overview" on page 856
- "HTTP traffic handling in WebSphere Message Broker" on page 857
- "HTTP traffic handling by using the proxy servlet in an external web servlet container" on page 859
- "HTTP proxy servlet; descriptions of required components" on page 863

When you have gained an understanding of the proxy servlet concept, read the following topics to help you install and test the HTTP proxy servlet:

- "Installing the proxy servlet" on page 872
- "Testing the proxy servlet" on page 892

Related concepts:

"HTTP proxy servlet overview" on page 856
The HTTP proxy servlet is a Java servlet that you can use in an external web servlet container, such as IBM WebSphere Application Server or Apache Tomcat, to receive HTTP requests from web services client applications, to replace the support provided by the broker and execution group HTTP listeners.

"HTTP traffic handling in WebSphere Message Broker" on page 857
Before reading about HTTP traffic handling using the proxy servlet in an external Web container, ensure that you understand the basic structure of HTTP traffic handling in WebSphere Message Broker.

"HTTP traffic handling by using the proxy servlet in an external web servlet container" on page 859
Before you install and test the HTTP proxy servlet, ensure that you understand the structure of HTTP traffic handling by using the proxy servlet in an external web servlet container, such as IBM WebSphere Application Server or Apache Tomcat.

"HTTP proxy servlet; descriptions of required components" on page 863
The HTTP proxy servlet requires a number of components such as message flows, a proxy servlet, a servlet container, web addresses, the Broker component, and web services clients.

Related tasks:

"Installing the proxy servlet" on page 872
The installation of the proxy servlet is made up of a number of tasks. Use this topic to complete each of the tasks to then move onto testing the proxy servlet.

"Testing the proxy servlet" on page 892
Test the proxy servlet with a message broker that receives HTTP requests and passes them to a message flow.

HTTP proxy servlet; proxy servlet component:

The HTTP proxy servlet requires a number of components such as message flows, a proxy servlet, a servlet container, web addresses, the Broker component, and web services clients. Ensure that you are familiar with the proxy servlet component.

A *Proxy servlet* is a Java Web Application Archive (WAR) file that is part of the runtime environment in WebSphere Message Broker Version 6.1 Fix Pack 3 (6.1.0.3) and above, and can be found in the following directory:

WMB61_runtime_install_path/tools. Where *WMB61_runtime_install_path* specifies the name of your runtime installation directory.

The proxy servlet is a Java servlet that receives HTTP requests. The proxy servlet matches the received web address with the web address that the HTTP or SOAP input nodes are monitoring, then passes the HTTPRequest to the correct HTTP or SOAP input node flow using WebSphere MQ.

The proxy servlet receives response messages from the HTTP or SOAP reply nodes and sends them back to the client applications over HTTP or HTTPS. The message broker has several internal WebSphere MQ queues, SYSTEM.BROKER.WS.* queues, that are used for the communication between the proxy servlet and the HTTP or SOAP input and reply nodes.

Each HTTP or SOAP input node monitors the arrival of requests associated to specific web addresses. The message broker has internal configuration files, and an internal WebSphere MQ queue SYSTEM.BROKER.WS.ACK, that contain the list of web addresses that are monitored by the different HTTP or SOAP input nodes in message flows deployed on any execution groups. The proxy servlet accesses the

internal file, or queue, to match the web addresses received in HTTP or HTTPS requests with the web addresses that the HTTP or SOAP input nodes that are waiting for.

The configuration files, or queue, have a unique correlation ID associated with each web address. The HTTP or SOAP input node uses this correlation ID to get the messages from the internal queue SYSTEM.BROKER.WS.INPUT, and the proxy servlet uses the same correlation ID to put the messages on this queue. This is the mechanism used to correlate the incoming HTTP or HTTPS requests and the HTTP or SOAP input nodes in the message flows.

The HTTP or SOAP input node copies the WebSphere MQ input message ID in the LocalEnvironment.Destination.HTTP.RequestIdentifier to be used by the HTTP or SOAP reply node for WebSphere MQ output message correlation ID. The proxy servlet does a selective GET by correlation ID to the reply queue SYSTEM.BROKER.WS.REPLY, and receives the response messages.

The proxy servlet accepts GET and POST HTTP or HTTPS requests.

Before you install and test the HTTP proxy servlet, ensure you understand the following concepts:

- "HTTP proxy servlet overview" on page 856
- "HTTP traffic handling in WebSphere Message Broker" on page 857
- "HTTP traffic handling by using the proxy servlet in an external web servlet container" on page 859
- "HTTP proxy servlet; descriptions of required components" on page 863

When you have gained an understanding of the proxy servlet concept, read the following topics to help you install and test the HTTP proxy servlet:

- "Installing the proxy servlet" on page 872
- "Testing the proxy servlet" on page 892

Related concepts:

"HTTP proxy servlet overview" on page 856
The HTTP proxy servlet is a Java servlet that you can use in an external web servlet container, such as IBM WebSphere Application Server or Apache Tomcat, to receive HTTP requests from web services client applications, to replace the support provided by the broker and execution group HTTP listeners.

"HTTP traffic handling in WebSphere Message Broker" on page 857
Before reading about HTTP traffic handling using the proxy servlet in an external Web container, ensure that you understand the basic structure of HTTP traffic handling in WebSphere Message Broker.

"HTTP traffic handling by using the proxy servlet in an external web servlet container" on page 859
Before you install and test the HTTP proxy servlet, ensure that you understand the structure of HTTP traffic handling by using the proxy servlet in an external web servlet container, such as IBM WebSphere Application Server or Apache Tomcat.

"HTTP proxy servlet; descriptions of required components" on page 863
The HTTP proxy servlet requires a number of components such as message flows, a proxy servlet, a servlet container, web addresses, the Broker component, and web services clients.

Related tasks:

"Installing the proxy servlet" on page 872
The installation of the proxy servlet is made up of a number of tasks. Use this

topic to complete each of the tasks to then move onto testing the proxy servlet.

"Testing the proxy servlet" on page 892
Test the proxy servlet with a message broker that receives HTTP requests and passes them to a message flow.

HTTP proxy servlet; servlet container component:

The HTTP proxy servlet requires a number of components such as message flows, a proxy servlet, a servlet container, web addresses, the Broker component, and web services clients. Ensure that you are familiar with the servlet container component.

A *Servlet container* is the runtime environment for servlets and Java Server Pages (JSP). WebSphere Application Server and Apache Tomcat are two examples of servlet containers (or web containers) that are available. The proxy servlet can be deployed in a local servlet container that is running on the same server as message broker or on a remote servlet container that is running on a remote server to message broker. The servlet container must allow the proxy servlet to configure and call the WebSphere MQ classes for Java.

The servlet container provides the SSL (HTTPS) listener support for web services applications. SSL must be configured and available in the container. The proxy servlet does not have to be configured for SSL but it enforces HTTP requests over SSL if the HTTP or SOAP input node is configured to use SSL.

Before you install and test the HTTP proxy servlet, ensure you understand the following concepts:

- "HTTP proxy servlet overview" on page 856
- "HTTP traffic handling in WebSphere Message Broker" on page 857
- "HTTP traffic handling by using the proxy servlet in an external web servlet container" on page 859
- "HTTP proxy servlet; descriptions of required components" on page 863

When you have gained an understanding of the proxy servlet concept, read the following topics to help you install and test the HTTP proxy servlet:

- "Installing the proxy servlet" on page 872
- "Testing the proxy servlet" on page 892

Related concepts:

"HTTP proxy servlet overview" on page 856
The HTTP proxy servlet is a Java servlet that you can use in an external web servlet container, such as IBM WebSphere Application Server or Apache Tomcat, to receive HTTP requests from web services client applications, to replace the support provided by the broker and execution group HTTP listeners.

"HTTP traffic handling in WebSphere Message Broker" on page 857
Before reading about HTTP traffic handling using the proxy servlet in an external Web container, ensure that you understand the basic structure of HTTP traffic handling in WebSphere Message Broker.

"HTTP traffic handling by using the proxy servlet in an external web servlet container" on page 859
Before you install and test the HTTP proxy servlet, ensure that you understand the structure of HTTP traffic handling by using the proxy servlet in an external web servlet container, such as IBM WebSphere Application Server or Apache Tomcat.

"HTTP proxy servlet; descriptions of required components" on page 863
The HTTP proxy servlet requires a number of components such as message flows, a proxy servlet, a servlet container, web addresses, the Broker component, and web services clients.

Related tasks:

"Installing the proxy servlet" on page 872
The installation of the proxy servlet is made up of a number of tasks. Use this topic to complete each of the tasks to then move onto testing the proxy servlet.

"Testing the proxy servlet" on page 892
Test the proxy servlet with a message broker that receives HTTP requests and passes them to a message flow.

HTTP proxy servlet; web addresses component:

The HTTP proxy servlet requires a number of components such as message flows, a proxy servlet, a servlet container, web addresses, the Broker component, and web services clients. Ensure that you are familiar with the web addresses component.

web addresses, or Universal Resource Locators (URLs), have an important role when HTTP or SSL (HTTPS) protocols are used. In WebSphere Message Broker, each HTTP or SOAP input node expects to receive requests from a specific web address, or web addresses when wildcard characters are used. The servlet container also uses the web address to locate the servlets that are going to process the HTTP or HTTPS requests received by the listener in the container.

The proxy servlet passes the requests from the servlet container to the broker and vice versa. web addresses have the dual function of locating servlets and locating HTTP or SOAP input nodes. This affects the format of the web addresses used for the broker.

A web address is made up of the following structure:

`<schema>://<hostname>:<port>/<url_path>`

web address structure definition:

- *<schema>*
- *<hostname>*
- *<port>*
- *<url_path>*

<schema> is HTTP or HTTPS.

<hostname> is the hostname, or IP address, of the server where the servlet container is running.

<port> is the port number that the servlet container is listening on.

<url_path> is a series of tokens separated by slashes /. These are used to indicate the location of the servlet and the location of the HTTP or SOAP input nodes.

Because the *<url_path>* is used for mapping two resources (instead of one with the broker internal listener) the format of the web address will change when the proxy servlet is used.

The broker structure of the *<url_path>* is:

/<url_path>=/<context_root>/<node_url_path>

Broker *<url_path>* structure definition:

- *<context_root>*
- *<node_url_path>*
- *<port>*
- *<url_path>*

<context_root> is the *<url_path>* allocated to the proxy servlet by the container when the servlet is installed and deployed.

<node_url_path> is the part of the web address path that is added to make the web address unique to a specific HTTP or SOAP input node.

The entire *<url_path>* has to be configured in the properties of the HTTPInput node.

In some web servlet containers, it is possible to configure the proxy servlet to receive all the HTTP or HTTPS requests arriving to the container (*<context_root>* = `"/*"`). In this case, the existing web addresses in the HTTP nodes do not have to change when the proxy servlet is implemented.

Before you install and test the HTTP proxy servlet, ensure you understand the following concepts:

- "HTTP proxy servlet overview" on page 856
- "HTTP traffic handling in WebSphere Message Broker" on page 857
- "HTTP traffic handling by using the proxy servlet in an external web servlet container" on page 859
- "HTTP proxy servlet; descriptions of required components" on page 863

When you have gained an understanding of the proxy servlet concept, read the following topics to help you install and test the HTTP proxy servlet:

- "Installing the proxy servlet" on page 872
- "Testing the proxy servlet" on page 892

Related concepts:

"HTTP proxy servlet overview" on page 856
The HTTP proxy servlet is a Java servlet that you can use in an external web servlet container, such as IBM WebSphere Application Server or Apache Tomcat, to receive HTTP requests from web services client applications, to replace the support provided by the broker and execution group HTTP listeners.

"HTTP traffic handling in WebSphere Message Broker" on page 857
Before reading about HTTP traffic handling using the proxy servlet in an external Web container, ensure that you understand the basic structure of HTTP traffic handling in WebSphere Message Broker.

"HTTP traffic handling by using the proxy servlet in an external web servlet container" on page 859
Before you install and test the HTTP proxy servlet, ensure that you understand the structure of HTTP traffic handling by using the proxy servlet in an external web servlet container, such as IBM WebSphere Application Server or Apache Tomcat.

"HTTP proxy servlet; descriptions of required components" on page 863
The HTTP proxy servlet requires a number of components such as message flows, a proxy servlet, a servlet container, web addresses, the Broker component, and web

services clients.

Related tasks:

"Installing the proxy servlet" on page 872
The installation of the proxy servlet is made up of a number of tasks. Use this topic to complete each of the tasks to then move onto testing the proxy servlet.

"Testing the proxy servlet" on page 892
Test the proxy servlet with a message broker that receives HTTP requests and passes them to a message flow.

HTTP proxy servlet; Broker component:

The HTTP proxy servlet requires a number of components such as message flows, a proxy servlet, a servlet container, web addresses, the Broker component, and Web services clients. Ensure that you are familiar with the Broker component.

The Broker component is the runtime environment for the message flows. Message flows can receive requests over HTTP when there are HTTP or SOAP input nodes in the flow.

WebSphere Message Broker has two types of HTTP listener: broker-wide listeners and execution group (embedded listeners). For more information about the two types of listener, see "HTTP listeners" on page 1589.

Each listener has its unique TCP/IP port to listen for HTTP and HTTPS requests.

The proxy servlet implements the HTTP or HTTPS listener function in an external servlet container.

Before you install and test the HTTP proxy servlet, ensure that you understand the following concepts:

- "HTTP proxy servlet overview" on page 856
- "HTTP traffic handling in WebSphere Message Broker" on page 857
- "HTTP traffic handling by using the proxy servlet in an external web servlet container" on page 859
- "HTTP proxy servlet; descriptions of required components" on page 863

When you have gained an understanding of the proxy servlet concept, read the following topics to help you install and test the HTTP proxy servlet:

- "Installing the proxy servlet" on page 872
- "Testing the proxy servlet" on page 892

Related concepts:

"HTTP proxy servlet overview" on page 856
The HTTP proxy servlet is a Java servlet that you can use in an external web servlet container, such as IBM WebSphere Application Server or Apache Tomcat, to receive HTTP requests from web services client applications, to replace the support provided by the broker and execution group HTTP listeners.

"HTTP traffic handling in WebSphere Message Broker" on page 857
Before reading about HTTP traffic handling using the proxy servlet in an external Web container, ensure that you understand the basic structure of HTTP traffic handling in WebSphere Message Broker.

"HTTP traffic handling by using the proxy servlet in an external web servlet container" on page 859
Before you install and test the HTTP proxy servlet, ensure that you understand the structure of HTTP traffic handling by using the proxy servlet in an external web servlet container, such as IBM WebSphere Application Server or Apache Tomcat.

"HTTP proxy servlet; descriptions of required components" on page 863
The HTTP proxy servlet requires a number of components such as message flows, a proxy servlet, a servlet container, web addresses, the Broker component, and web services clients.

Related tasks:

"Installing the proxy servlet" on page 872
The installation of the proxy servlet is made up of a number of tasks. Use this topic to complete each of the tasks to then move onto testing the proxy servlet.

"Testing the proxy servlet" on page 892
Test the proxy servlet with a message broker that receives HTTP requests and passes them to a message flow.

HTTP proxy servlet; web services clients component:

The HTTP proxy servlet requires a number of components such as message flows, a proxy servlet, a servlet container, web addresses, the Broker component, and web services clients. Ensure that you are familiar with the web services clients component.

web services clients are applications that send and receive SOAP (or simple HTTP) requests and responses to and from web services implemented in message flows running in a broker.

Before you install and test the HTTP proxy servlet, ensure you understand the following concepts:

- "HTTP proxy servlet overview" on page 856
- "HTTP traffic handling in WebSphere Message Broker" on page 857
- "HTTP traffic handling by using the proxy servlet in an external web servlet container" on page 859
- "HTTP proxy servlet; descriptions of required components" on page 863

When you have gained an understanding of the proxy servlet concept, read the following topics to help you install and test the HTTP proxy servlet:

- "Installing the proxy servlet" on page 872
- "Testing the proxy servlet" on page 892

Related concepts:

"HTTP proxy servlet overview" on page 856
The HTTP proxy servlet is a Java servlet that you can use in an external web servlet container, such as IBM WebSphere Application Server or Apache Tomcat, to receive HTTP requests from web services client applications, to replace the support provided by the broker and execution group HTTP listeners.

"HTTP traffic handling in WebSphere Message Broker" on page 857
Before reading about HTTP traffic handling using the proxy servlet in an external Web container, ensure that you understand the basic structure of HTTP traffic handling in WebSphere Message Broker.

"HTTP traffic handling by using the proxy servlet in an external web servlet container" on page 859
Before you install and test the HTTP proxy servlet, ensure that you understand the structure of HTTP traffic handling by using the proxy servlet in an external web servlet container, such as IBM WebSphere Application Server or Apache Tomcat.

"HTTP proxy servlet; descriptions of required components" on page 863
The HTTP proxy servlet requires a number of components such as message flows, a proxy servlet, a servlet container, web addresses, the Broker component, and web services clients.

Related tasks:

"Installing the proxy servlet"
The installation of the proxy servlet is made up of a number of tasks. Use this topic to complete each of the tasks to then move onto testing the proxy servlet.

"Testing the proxy servlet" on page 892
Test the proxy servlet with a message broker that receives HTTP requests and passes them to a message flow.

Installing the proxy servlet

The installation of the proxy servlet is made up of a number of tasks. Use this topic to complete each of the tasks to then move onto testing the proxy servlet.

Before you begin

Before you install the HTTP proxy servlet, ensure that you understand the following concepts:

- "HTTP traffic handling in WebSphere Message Broker" on page 857
- "HTTP traffic handling by using the proxy servlet in an external web servlet container" on page 859
- "HTTP proxy servlet; descriptions of required components" on page 863

About this task

The proxy servlet `proxyservlet.war` (WAR) file is part of the runtime environment in WebSphere Message Broker Version 6.1 Fix Pack 3 (6.1.0.3) and above, and can be found in the following directory:

WMB61_runtime_installation_path`/tools`. Where *WMB61_runtime_installation_path* specifies the name of your runtime installation directory.

Supported operating systems

The proxy servlet runs on any operating system that is supported by the servlet container. The documentation in this section describes the installation on Windows XP but the proxy servlet can be installed on other operating systems.

Prerequisites

The proxy servlet requires:

- WebSphere Message Broker Version 7.0, or later.
- A supported version of WebSphere MQ. For the latest details of all supported levels of hardware and software, visit the WebSphere Message Broker Requirements website.
- A servlet container and web server, such as WebSphere Application Server Version 6, or Apache Tomcat Version 6.

To install the proxy servlet:

Procedure

1. Install and customize a web servlet container, such as WebSphere Application Server or Apache Tomcat. For more information, see "Installing and customizing a web servlet container for the proxy servlet."
2. Configure the proxy servlet with the initialization parameters that are used by the proxy servlet. For more information, see "Configuring the proxy servlet" on page 876.
3. Enable the WebSphere MQ listener on each execution group where message flows with SOAP nodes are deployed so that the proxy servlet can access the SOAPInput and SOAPReply nodes. For more information, see "Enabling the WebSphere MQ listener for SOAP nodes for the proxy servlet to access" on page 889.
4. Deploy the proxy servlet in the web servlet container. For more information, see "Deploying the proxy servlet in the web servlet container" on page 891.

Results

You are now ready to test the proxy servlet. For information about how to complete this task, see "Testing the proxy servlet" on page 892.

Related concepts:

"HTTP proxy servlet overview" on page 856
The HTTP proxy servlet is a Java servlet that you can use in an external web servlet container, such as IBM WebSphere Application Server or Apache Tomcat, to receive HTTP requests from web services client applications, to replace the support provided by the broker and execution group HTTP listeners.

"HTTP traffic handling in WebSphere Message Broker" on page 857
Before reading about HTTP traffic handling using the proxy servlet in an external Web container, ensure that you understand the basic structure of HTTP traffic handling in WebSphere Message Broker.

"HTTP traffic handling by using the proxy servlet in an external web servlet container" on page 859
Before you install and test the HTTP proxy servlet, ensure that you understand the structure of HTTP traffic handling by using the proxy servlet in an external web servlet container, such as IBM WebSphere Application Server or Apache Tomcat.

"HTTP proxy servlet; descriptions of required components" on page 863
The HTTP proxy servlet requires a number of components such as message flows, a proxy servlet, a servlet container, web addresses, the Broker component, and web services clients.

Related tasks:

"Installing the proxy servlet" on page 872
The installation of the proxy servlet is made up of a number of tasks. Use this topic to complete each of the tasks to then move onto testing the proxy servlet.

"Testing the proxy servlet" on page 892
Test the proxy servlet with a message broker that receives HTTP requests and passes them to a message flow.

Installing and customizing a web servlet container for the proxy servlet:

Download, install, and customize a web servlet container, such as WebSphere Application Server or Apache Tomcat, for the proxy servlet to use to receive HTTP requests from web services client applications.

Before you begin

Before you install the servlet container, you must have completed the following tasks:
- Installed WebSphere Message Broker.
- If you are planning to install Apache Tomcat V6 as your web servlet, you must have installed a Java SDK that is compatible with Apache Tomcat V6.

About this task

The documentation in this section is based on an Apache Tomcat Version 6 installation, however this procedure can be used to guide you through the installation of a different web servlet container because the procedures are similar. This information describes the Apache Tomcat V6 installation on Windows XP and assumes WebSphere Message Broker is installed and running on the same server.

Windows **Windows**

Procedure

1. Download Apache Tomcat V6 to your local directory from the Apache Tomcat 6 Downloads website at: Apache Tomcat 6 download page
2. Run the Apache Tomcat V6 installer.
   a. Accept the license agreement.
   b. Select the required Apache Tomcat components and features to install.
   c. Select the destination directory for the Apache Tomcat installation. For example, `C:\Tomcat6`.
   d. Enter an HTTP listener port. For example, 8181.
   e. Type an administrator user ID and password. For example, *admin*.
   f. Browse and select a path for the J2SE 5.0 Java Runtime Environment (JRE). For example, `C:\Program Files\IBM\Java50\jre`.
   g. Click **Install** to start the installation.
3. Restart Windows. Apache Tomcat is now running as a Windows Service.
4. Open a web browser, such as Internet Explorer, and enter web address http://localhost:8181. Apache Tomcats home page is displayed.
5. Find file `catalina.properties` in your local directory. For example, <*tomcat_installation_path*>`/conf`. Where <*tomcat_installation_path*> specifies the name of your Apache Tomcat installation directory.
6. Edit `catalina.properties` and change **`shared.loader=`** to **`shared.loader= ${catalina.home}/shared/lib,${catalina.home}/shared/lib/*.jar`**.
7. Create a directory called <*tomcat_installation_path*>`/shared/lib`.
8. Copy the required WebSphere MQ JAR files into the Apache Tomcat shared loader directory. The proxy servlet uses WebSphere MQ to communicate with WebSphere Message Broker. It requires the following JAR files to exist in the Apache Tomcat shared loader directory: `com.ibm.mq.jar` and `connector.jar`. From WebSphere MQ Version 7.0 onwards, you also require the following JAR files:
   - `com.ibm.mq.commonservices.jar`
   - `com.ibm.mq.headers.jar`
   - `com.ibm.mq.pcf.jar`
   - `com.ibm.mq.jmqi.jar`

Copy the JAR files from <*WebSphere_MQ_installation_path*>\Java\lib to <*tomcat_installation_path*>\shared\lib.

Results

You have downloaded, installed, and customized Apache Tomcat V6 ready for use by the proxy servlet.

You must now configure the proxy servlet, see:

- "Configuring the proxy servlet" on page 876

Related concepts:

"HTTP proxy servlet overview" on page 856
The HTTP proxy servlet is a Java servlet that you can use in an external web servlet container, such as IBM WebSphere Application Server or Apache Tomcat, to receive HTTP requests from web services client applications, to replace the support provided by the broker and execution group HTTP listeners.

"HTTP traffic handling in WebSphere Message Broker" on page 857
Before reading about HTTP traffic handling using the proxy servlet in an external Web container, ensure that you understand the basic structure of HTTP traffic handling in WebSphere Message Broker.

"HTTP traffic handling by using the proxy servlet in an external web servlet container" on page 859
Before you install and test the HTTP proxy servlet, ensure that you understand the structure of HTTP traffic handling by using the proxy servlet in an external web servlet container, such as IBM WebSphere Application Server or Apache Tomcat.

"HTTP proxy servlet; descriptions of required components" on page 863
The HTTP proxy servlet requires a number of components such as message flows, a proxy servlet, a servlet container, web addresses, the Broker component, and web services clients.

Related tasks:

"Installing the proxy servlet" on page 872
The installation of the proxy servlet is made up of a number of tasks. Use this topic to complete each of the tasks to then move onto testing the proxy servlet.

"Configuring the proxy servlet" on page 876
Configure the proxy servlet with the initialization parameters that are used by the proxy servlet. These parameters need to be configured for the broker environment that the proxy servlet is connecting to before the proxy servlet can be deployed to the servlet container.

"Enabling the WebSphere MQ listener for SOAP nodes for the proxy servlet to access" on page 889
Enable the WebSphere MQ listener on each execution group where message flows with SOAP nodes are deployed so that the proxy servlet can access the SOAPInput and SOAPReply nodes.

"Deploying the proxy servlet in the web servlet container" on page 891
Load and install the proxy servlet file in the web servlet container, such as WebSphere Application Server or Apache Tomcat.

"Testing the proxy servlet" on page 892
Test the proxy servlet with a message broker that receives HTTP requests and passes them to a message flow.

Related reference:

"Proxy servlet configuration parameters" on page 878
Before you can deploy the proxy servlet `web.xml` to the servlet container, you must configure it with the following initialization parameters for the broker environment that the servlet connects to.

Configuring the proxy servlet:

Configure the proxy servlet with the initialization parameters that are used by the proxy servlet. These parameters need to be configured for the broker environment that the proxy servlet is connecting to before the proxy servlet can be deployed to the servlet container.

Before you begin

Before you configure the proxy servlet, you must complete the following task:
- "Installing and customizing a web servlet container for the proxy servlet" on page 873

About this task

The configuration of the proxy servlet is done by editing the Web Deployment Descriptor `web.xml` file that is packaged in the compressed `proxyservlet.war` (WAR) file. The WAR file can be found in the following directory:

<*WMB_runtime_install_path*>`/tools`.

You can either use the WebSphere Message Broker toolkit to edit the Web Deployment Descriptor `web.xml` file, or you can extract the `proxyservlet.war` file, find the `web.xml` file and edit it using an appropriate editor, such as Notepad. Both procedures are described in this section:

Procedure

1. To configure the Web Deployment Descriptor `web.xml` file by using the WebSphere Message Broker toolkit, start the toolkit, and switch to the J2EE perspective.
2. Click **File** > **Import**, expand the **Web** section, select **WAR file** in the list, and click **Next**.
3. Click **Browse** to find the WAR file in <*WMB_runtime_install_path*>`/tools`. Where <*WMB_runtime_install_path*> specifies the name of your runtime installation path. For example, in `C:\Program Files\IBM\MQSI\7.0\tools\proxyservlet.war`.
4. Set the name of the Web Project to `proxyservlet`.
5. Click **Finish**. The proxy servlet is now ready for configuring by using the J2EE perspective.
6. Expand **proxyservlet** in the Project Explorer view and double-click Deployment Descriptor to view the Web Deployment Descriptor.
7. Find the **Servlets and JSPs** section in the Web Deployment Descriptor, and click the servlet link called **WBIMBServlet** to display the servlet web address mappings and initialization parameters. The same parameters in the `web.xml` file can be configured through JNDI in WebSphere Application Server. This alternative method means that you set up at the application server side only once for any future deployment of the proxy servlet. This operation is possible because the JNDI configuration parameters take precedence over the initialization parameters in the `web.xml` file. For more information about

setting up the JNDI interface for the proxy servlet, see “Setting up the JNDI interface for the proxy servlet” on page 886

8. Click the Source tab, which is found at the bottom of the Web Deployment Descriptor view. You might need to click **>>** (Show List) to see the Source tab option. The source of the Web Deployment Descriptor `web.xml` displays the proxy servlet parameters.
9. Edit the proxy servlet parameters with initialization parameters specified at: “Proxy servlet configuration parameters” on page 878
10. When the configuration is complete, save the changes to the Deployment Descriptor `web.xml` file by pressing **Ctrl S**.
11. Export the configured proxy servlet ready for deployment to Tomcat. Click **File** > **Export**, expand the **Web** section, select **WAR file** in the list, and click **Next**.
12. Click **Browse**, specify a location for the configured WAR file, enter WAR file name `HTTPVSR1BKproxyserlet.war`, and click **Save**.
13. Enter the Web module name `proxyservlet`, and click **Finish**. You have now configured the proxy servlet by using the WebSphere Message Broker toolkit.

Results

To configure the Web Deployment Descriptor `web.xml` file directly, find and extract the `proxyservlet.war` file, find the `web.xml` file in the extracted contents, right-click the `web.xml` file, and select an appropriate editor, such as Notepad to edit the `web.xml` file with initialization parameters specified at: “Proxy servlet configuration parameters” on page 878.

You have now configured the proxy servlet with the initialization parameters.

In order that the proxy servlet can access the SOAPInput and SOAPReply nodes, you must now enable the WebSphere MQ listener on each execution group where message flows with SOAP nodes are deployed, see:

- “Enabling the WebSphere MQ listener for SOAP nodes for the proxy servlet to access” on page 889

Related concepts:

“HTTP proxy servlet overview” on page 856
The HTTP proxy servlet is a Java servlet that you can use in an external web servlet container, such as IBM WebSphere Application Server or Apache Tomcat, to receive HTTP requests from web services client applications, to replace the support provided by the broker and execution group HTTP listeners.

“HTTP traffic handling in WebSphere Message Broker” on page 857
Before reading about HTTP traffic handling using the proxy servlet in an external Web container, ensure that you understand the basic structure of HTTP traffic handling in WebSphere Message Broker.

“HTTP traffic handling by using the proxy servlet in an external web servlet container” on page 859
Before you install and test the HTTP proxy servlet, ensure that you understand the structure of HTTP traffic handling by using the proxy servlet in an external web servlet container, such as IBM WebSphere Application Server or Apache Tomcat.

“HTTP proxy servlet; descriptions of required components” on page 863
The HTTP proxy servlet requires a number of components such as message flows, a proxy servlet, a servlet container, web addresses, the Broker component, and web services clients.

Related tasks:

"Installing the proxy servlet" on page 872
The installation of the proxy servlet is made up of a number of tasks. Use this topic to complete each of the tasks to then move onto testing the proxy servlet.

"Installing and customizing a web servlet container for the proxy servlet" on page 873
Download, install, and customize a web servlet container, such as WebSphere Application Server or Apache Tomcat, for the proxy servlet to use to receive HTTP requests from web services client applications.

"Enabling the WebSphere MQ listener for SOAP nodes for the proxy servlet to access" on page 889
Enable the WebSphere MQ listener on each execution group where message flows with SOAP nodes are deployed so that the proxy servlet can access the SOAPInput and SOAPReply nodes.

"Deploying the proxy servlet in the web servlet container" on page 891
Load and install the proxy servlet file in the web servlet container, such as WebSphere Application Server or Apache Tomcat.

"Testing the proxy servlet" on page 892
Test the proxy servlet with a message broker that receives HTTP requests and passes them to a message flow.

"Setting up the JNDI interface for the proxy servlet" on page 886
The JNDI interface for the proxy servlet requires a one time setup of the WebSphere Application Server full profile.

Proxy servlet configuration parameters:

Before you can deploy the proxy servlet `web.xml` to the servlet container, you must configure it with the following initialization parameters for the broker environment that the servlet connects to.

This topic contains the following sections:
- "General options"
- "Information options" on page 880
- "ReplyToQ and QMgr options" on page 880
- "SSL connection options" on page 880
- "MQ connection options" on page 881

General options:

Parameter name	Default value	Description
brokerName	* (auto-detect from config data for broker)	*broker name* or "*" Use this parameter to set the name used for error messages; the value is auto-detected if set to "*". Set a value if several brokers are being proxied, and a single name is required for error messages.

Parameter name	Default value	Description
forwardingNodes	http	*http*, *soap*, or *both* If you configure the servlet to replace the broker-wide listener, set this property to <*http*>. If you configure the servlet to replace the embedded execution group listener, set this property to <*soap*>. Use *both* with caution, because the merging of web address selectors might lead to unexpected results in conflicting cases; for example, if an HTTP node and a SOAP node both use `/fnerble` as a web address selector.
configFilePath	`/var/mqsi/components/MB7BROKER/config/wsplugin6.conf`	*full path to config file* If the proxied broker is local, set this parameter to the `wsplugin6.conf` file (for HTTP nodes) or the `soapplugin6.conf` (for SOAP nodes) for the broker. This file is used only when the parameter **useQueueManagerDataInsteadOfConfigFile** is set to blank. The configuration file can be used only when the proxy servlet is running on the same server as the broker, and it has access to the file. In Windows, the file is stored in `C:\`*`install_dir`*`\config\wsplugin.conf` or `C:\Documents and Settings\All Users\Application Data\IBM\MQSI\components\`*`broker_ name`*`\config\wsplugin6.conf`. On Linux and UNIX, the file is stored in `/var/mqsi/config/wsplugin.conf` or in `/var/mqsi/components/`*`broker_ name`*`/config/wsplugin6.conf`.
useFastpathBindingsConnection	false	*true* or *false* Causes the servlet to connect in fastpath mode, if using a local queue manager.
traceFileName		*full path to trace file* Specify the location and name of the trace file. If this parameter is not specified the trace is sent to stdout.
turnTraceOn	0	*0*, *1*, or *2* Set 0 for no trace, 1 for normal trace, or 2 for debug trace.

Information options:

Parameter name	Default value	Description
enableStatusPage (WebSphere Message Broker V6.1 FP 4 or later)	false	*true* or *false* Switches display of the status page. When true, the page is visible at `http://hostname:port/proxy_context/messagebroker/httpproxy/statuspage`
enableInfoHeaders (WebSphere Message Broker V6.1 FP 4 or later)	false	*true* or *false* Causes the servlet to add extra headers in the response. These headers are: X-WMB-Broker-Name X-WMB-QM-Name X-WMB-MQ-URL-CorrelId and contain details of the configuration used for that message.

ReplyToQ and QMgr options:

Parameter name	Default value	Description
useClusterMode	false	*true* or *false* Set to true if the servlet is required to put reply-to queue and queue manager information in the MQMD of sent messages to enable the broker to respond to the correct queue manager in a cluster.
clusterModeQueueManagerName	SOME_OTHER_ QUEUE_MANAGER	*queue manager name* Queue manager name for initial MQCONN and ReplyToQMgr.
clusterModeReplyToQ	OUR.REPLYTO.QUEUE	*reply queue name* Queue name on which to listen.

SSL connection options:

Parameter name	Default value	Description
useSecuredChannel	false	*true* or *false* Set the value of `useSecuredChannel` to `true` if SSL is configured on MQ Channel. If set to `true`, the servlet attempts to establish a secured connection to the MQ Channel by using the `keyStore`, `keyStorePassword`, `trustStore`, `trustStorePassword`, and `cipherSuite` parameter values.

Parameter name	Default value	Description
keyStore		*full path to the keystore file* The fully qualified path to the key store file, which is of type "JKS". For example, in Windows: `C:\\Program Files\\IBM\\MQSI\\ keystore.jks` On Linux and UNIX: `/var/mqsi/keystore.jks`
keyStorePassword	changeit	The password to the keystore file.
trustStore		*full path to the truststore file*The fully qualified path to the truststore file, which is of type "JKS". For example, on Windows: `C:\\Program Files\\IBM\\MQSI\\ truststore.jks` On Linux and UNIX: `/var/mqsi/truststore.jks`This field is mandatory if `useSecuredChannel` is set to `true`.
trustStorePassword	changeit	The password to the truststore file.
cipherSuite		The encryption type that is configured in the MQ Channel. For example: `SSL_RSA_WITH_NULL_MD5` This field is mandatory if `useSecuredChannel` is set to `true`.

MQ connection options:

Parameter name	Default value	Description
useClientMode	false	*true* or *false* Use WebSphere MQ client (true) or bindings connection (false). Normally, **useQueueManagerDataInsteadOfConfigFile** would also be set to the broker queue manager if this parameter is set to true.
clientModeHostname	localhost	*hostname or IP address* Hostname or IP for the Queue Manager.
clientModeChannelName	SYSTEM.DEF. SVRCONN	*WebSphere MQ SVRCONN channel name* The name of the WebSphere MQ SVRCONN to use.
clientModePortNumber	1414	*port number* WebSphere MQ listener port number.

Parameter name	Default value	Description
clientModeConnectRetryCount (WebSphere Message Broker V6.1 FP 4 or later)	1	*integer* Number of times to retry the WebSphere MQ connect call. Use this parameter in cases where a network dispatcher or load balancer is being used to distribute work to a set of queue managers and one fails. A new connect might fail the first time, but succeed the second time. The retry count must be set to a high number to provide the greatest chance of success.
useQueueManagerDataInsteadOfConfigFile		*queue manager name*, "*", or *blank* Queue manager name, '*' (remote proxy), or blank for none (local proxy). This option causes the servlet to read Web address data from a queue, and avoid the need for a config file to be accessible from the servlet.
sleepBeforeGet	0	*time in seconds* Sleep time in seconds. This value causes the servlet to wait before issuing an MQGET for a response message from the broker.
disconnectBeforeSleep	true	*true* or *false* To release WebSphere MQ handle while sleeping. Useful for keeping the number of simultaneous WebSphere MQ connections down.
reconnectActiveLinksAge (WebSphere Message Broker V6.1 FP 4 or later)	-1	*time in seconds*, 0, or -1 If set to a number greater than zero, this parameter causes WebSphere MQ connections to be disconnected and reconnected if they have been inactive, because of low traffic volumes, for more than the specified number of seconds. Setting this to -1 prevents this reconnection. Setting it to 0 causes all connections to be used once only. This parameter is of most use if the connection to WebSphere MQ goes through a firewall that closes connections after a period of inactivity. Setting this parameter to a value less than the firewall timeout might prevent clients from getting WebSphere MQ 2009 (connection broken) errors.

Parameter name	Default value	Description
testConnectionBeforeReuse (WebSphere Message Broker V6.1 FP 4 or later)	false	*true* or *false* If set to true, the servlet attempts an MQINQ before doing the MQPUT of the HTTP data message. All problems with a cached WebSphere MQ client connection are detected at that point, and a new connection is established for the MQPUT of the actual data (and MQGET of the response). This parameter causes significant extra network traffic, and must be used only if problems have been seen with dropped connections, which are usually seen as WebSphere MQ 2009 errors, indicating connection broken.
maximumConnectionAge (WebSphere Message Broker V6.1 FP 6 or later)	-1	*time in seconds*, 0, or -1 If set to a number greater than zero, this parameter causes WebSphere MQ connections to be disconnected and reconnected if they are older than the specified number of seconds. Setting this parameter to -1 prevents these reconnections; setting this parameter to 0 causes all connections to be used only once. This parameter is of most use, if the frequent changes to the WebSphere MQ connection parameters are expected due to redeploys of the WebSphere Message Broker flows and you require the *ProxyServlet* to reflect these changes within the specified number of seconds.

You can define one or more mappings that are supported by the proxy servlet. These mappings are used by the servlet container to filter Web address requests before executing the correct instance of the proxy servlet.

The mappings are the */node_url_path* paths described in "HTTP proxy servlet; web addresses component" on page 868 (*/url_path=/context_root/node_url_path*).

You can define a */node_url_path* equal to "/*" to accept Web address paths similar to this example: /HTTPMyBrkServletProxy/*your_value*.

Example configuration scenarios:

In each of the following scenarios, **forwardingNodes** must be set for the HTTP and SOAP nodes:

- Scenario 1: The Web servlet container is on the same server as WebSphere Message Broker:

  In this configuration example, you must set **useClientMode** and **useQueueManagerDataInsteadOfConfigFile** to false, and the **configFile**

parameter must point to a valid file. The servlet attempts to connect to the local queue manager for the broker after reading the queue manager name from the config file.

- Scenario 2: The Web servlet container is on a different server to WebSphere Message Broker with an WebSphere MQ client link to the broker queue manager:

  In this configuration example, you must set **useClientMode** to true, **useQueueManagerDataInsteadOfConfigFile** to "*", or the broker queue manager name, and **clientModeHostname**, **clientModeChannelName**, and **clientModePortNumber** to the correct values defined in this section. The servlet attempts to connect to the remote queue manager for the broker, reading the required node configuration data from the broker from the WebSphere MQ client connection.

  You can copy the config file from the broker server to the Web servlet container server, and then use **configFile**, after ensuring that **useQueueManagerDataInsteadOfConfigFile** is set to blank to force reading from the config file. However, you must copy the config file every time it is changed by the broker.

- Scenario 3: The Web container is on a different server to WebSphere Message Broker with its own queue manager and a WebSphere MQ channel link to the broker queue manager:

  This configuration example is similar to scenario 1 in that client mode is not used, but you must set **useClusterMode** to true, **clusterModeQueueManagerName** to the queue manager of the Web servlet container, and **clusterModeReplyToQ** to a queue that exists on that queue manager.

  The servlet tries to open the queue SYSTEM.BROKER.WS.INPUT on the specified queue manager by using the queue manager name from the config file. Therefore, you must set up channels and transmit queues beforehand, to ensure the messages arrive on the broker queue manager.

  You must copy the config files from the broker server in this scenario.

- Scenario 4: This scenario is the same as scenario 2, but uses a network load-balancer for distributing work for several brokers:

  The configuration can be the same as scenario 2, with the network load-balancer IP address taking the place of the broker server. In general, config files cannot be used, because there are several brokers behind one virtual IP address, and each one has a different config file. The servlet loads information on a per-connection basis, and uses the correct configuration information for each broker.

  Because failover is often one of the reasons for this configuration, the following extra options can be useful:

  Set **clientModeConnectRetryCount** to ensure that a single failed server does not cause intermittent errors, even if the load-balancer does simple round connection dispatching. Setting the parameter to a high enough value to attempt to connect to all the brokers prevents problems in these cases, and the servlet uses the first available broker.

  Use **reconnectActiveLinksAge** to avoid reusing old connections that might have been discarded by firewalls in between the servlet and the load-balancer (or the load-balancer and the brokers). Set this parameter to a value less than the firewall timeout, to ensure connections are used only when they are valid.

  Use **testConnectionBeforeReuse** as an alternative way to handle dropped WebSphere MQ links between the Web servlet container and broker queue managers. This option causes an MQINQ to be performed before attempting to put any data to the broker. If the MQINQ fails, a new connection is established,

and the data is sent over the new connection. Because configuration adds another operation to the MQPUT and MQGET, it results in a significant overhead for every message; use this option only if no alternative options are available.

To finish completing the proxy servlet configuration, see "Configuring the proxy servlet" on page 876.

Related concepts:

"HTTP proxy servlet overview" on page 856
The HTTP proxy servlet is a Java servlet that you can use in an external web servlet container, such as IBM WebSphere Application Server or Apache Tomcat, to receive HTTP requests from web services client applications, to replace the support provided by the broker and execution group HTTP listeners.

"HTTP traffic handling in WebSphere Message Broker" on page 857
Before reading about HTTP traffic handling using the proxy servlet in an external Web container, ensure that you understand the basic structure of HTTP traffic handling in WebSphere Message Broker.

"HTTP traffic handling by using the proxy servlet in an external web servlet container" on page 859
Before you install and test the HTTP proxy servlet, ensure that you understand the structure of HTTP traffic handling by using the proxy servlet in an external web servlet container, such as IBM WebSphere Application Server or Apache Tomcat.

"HTTP proxy servlet; descriptions of required components" on page 863
The HTTP proxy servlet requires a number of components such as message flows, a proxy servlet, a servlet container, web addresses, the Broker component, and web services clients.

Related tasks:

"Installing the proxy servlet" on page 872
The installation of the proxy servlet is made up of a number of tasks. Use this topic to complete each of the tasks to then move onto testing the proxy servlet.

"Installing and customizing a web servlet container for the proxy servlet" on page 873
Download, install, and customize a web servlet container, such as WebSphere Application Server or Apache Tomcat, for the proxy servlet to use to receive HTTP requests from web services client applications.

"Configuring the proxy servlet" on page 876
Configure the proxy servlet with the initialization parameters that are used by the proxy servlet. These parameters need to be configured for the broker environment that the proxy servlet is connecting to before the proxy servlet can be deployed to the servlet container.

"Enabling the WebSphere MQ listener for SOAP nodes for the proxy servlet to access" on page 889
Enable the WebSphere MQ listener on each execution group where message flows with SOAP nodes are deployed so that the proxy servlet can access the SOAPInput and SOAPReply nodes.

"Deploying the proxy servlet in the web servlet container" on page 891
Load and install the proxy servlet file in the web servlet container, such as WebSphere Application Server or Apache Tomcat.

"Testing the proxy servlet" on page 892
Test the proxy servlet with a message broker that receives HTTP requests and passes them to a message flow.

Setting up the JNDI interface for the proxy servlet:

The JNDI interface for the proxy servlet requires a one time setup of the WebSphere Application Server full profile.

About this task

The proxy servlet initialization parameters must be configured for the broker environment that the proxy servlet is connecting to each time the proxy servlet is deployed to the servlet container. It is now possible to configure the `web.xml` parameters only once through the JNDI in WebSphere Application Server, regardless of how many future deployments there might be of the proxy servlet. Because the JNDI configuration parameters take precedence over the initialization parameters in the `web.xml` file, using this method means that you need to set up at the application server side only once for any future deployments of the proxy servlet.

These setup tasks must all be completed in the WebSphere Application Server administrative console.

Creating a resource environment provider:
About this task

Configure a resource environment provider, which encapsulates the referenceables that convert resource environment entry data into resource objects. These resource objects can then be accessed by applications.

Procedure

1. Select **Resources** > **Resource Environment** > **Resource Environment Providers**. The Resource environment providers wizard opens.
2. Click **New**. The Configuration panel opens so that you can configure a new resource environment provider.
3. Type a name for the resource environment provider in the **Name** filed. For example, `MyResourceEnvironmentProvider`. It is recommended that you enter a meaningful description in the **Description** field, but it is not required. Click **OK** to continue and then save the changes.

Results

The new resource environment provider is listed in the wizard.

Creating a referenceable object:
About this task

Configure a new referenceable, which specifies the factory class that converts data in the Java Naming and Directory Interface (JNDI) name space into an object that represents your resource to WebSphere Application Server.

Procedure

1. Select **Resources** > **Resource Environment** > **Resource Environment Providers**. The Resource environment providers wizard opens.
2. From the Resource environment providers panel, select the provider that you created in the previous task: In this example, it is called *MyResourceEnvironmentProvider*.
3. Click **Referenceables**. The Referenceables panel opens.

4. Click **New**. The Configuration panel opens.
5. In the Configuration panel, type the following values:
   a. In the **Factory class name** field, type:
   `com.ibm.broker.httpproxy.MQResourceConnectionFactory`
   b. In the **Class name** field, type:
   `com.ibm.broker.httpproxy.MQResourceConnection`
6. Click **OK** and save the changes.

Creating resource environment entries:
About this task

Configure resource environment entries, which are objects that contain information about a resource, and represent it in the JNDI name space.

Procedure

1. Select **Resources** > **Resource Environment** > **Resource Environment Providers** > ***MyResourceEnvironmentProvider***. The Resource environment providers wizard opens at the configuration panel for your provider.
2. Click **Resource environment entries**. The Resource environment entries panel opens.
3. Click **New**. The Configuration panel opens.
4. In the Configuration panel, type your values. In this example, the values are:
   a. In the **Name** field, type for example: `MQResourceReference`
   b. In the **JNDI name** field, type for example: `proxyservlet/reference/MQResourceReference` . This JNDI name is used during application deployment resource mapping.
   c. In the **Referenceables** drop-down list, ensure that `com.ibm.broker.httpproxy.MQResourceConnectionFactory` is selected.
5. Click **OK** and save the changes.

Creating custom properties or define the parameters to be configured:
About this task

Specify custom properties that your enterprise information system (IES) requires for the resource providers and resource factories that you configure. For example, most database vendors require extra custom properties for data sources that access the database.

Procedure

1. Select **Resources** > **Resource Environment** > **Resource Environment Providers** > ***MyResourceEnvironmentProvider*** > **ResourceEnvironment Entries** > ***MQResourceReference***. where *MyResourceEnvironmentProvider* and *MQResourceReference* are your own specified values. The Resource environment providers wizard opens at the configuration panel for your entry.
2. Click **Custom properties**. The Configuration panel opens.
3. Click **New**.
4. In the Configuration panel, you must type your values for all the resources that are administered through the administrative console. For example: The message broker name, the configuration file path, the client mode channel name, and the client mode port number. For each resource, you must provide values for **Name**, **Value**, **Description**, and **Type**. The following example is for the message broker name

Option	Description
Name	This field refers to the value in the "<param-name> </param-name>" tag in the web.xml. For example: brokerName
Value	This field refers to the value in the "<param-value> </param-value>" tag in the web.xml. For example: RRB
Description	Optional. It is recommended that you provide a meaningful description.
Type	Specify the type. For example: java.lang.String

5. Click **OK** to save the configuration.
6. Click **New** to create a new configuration for the remaining resources.

Results

You have a list of configured resources that is similar to the following table, but with the values that you specified:

Table 12. .

Name	Value	Description	Required
brokerName	RRB	Message Broker Name	false
configFilePath	C:\\Documents and Settings\\All Users.WINDOWS\\Application Data\\IBM\\MQSI\\components\\RRB\\config\soapplugin6.conf	Configuration File Path	false
clientModeChannelName	SYSTEM.DEF.SVR.CONN	Client Mode Channel Name	false
clientModePortNumber	2028	Client Mode Port Number	false

The WebSphere Application Server wizard does not provide an option to specify the **Required** attribute and so the default value is set to false. This attribute can be ignored.

What to do next

After you complete these tasks, you must deploy the proxyservlet.war file. During the deployment, you must provide the JNDI reference name for the **Resource Environment** reference that is defined in the web.xml file. For example: proxyservlet/reference/MQResourceReference.

After the deployment, proxyservlet gives precedence to the values configures in the Resource Environment Entries. If there is an environment change, the values must be modified in the custom properties of the Resource Environment Entries. After you modify a value, restart the Proxyservlet application in WebSphere Application Server for the new values to take effect.

Related concepts:

"HTTP proxy servlet overview" on page 856
The HTTP proxy servlet is a Java servlet that you can use in an external web servlet container, such as IBM WebSphere Application Server or Apache Tomcat, to receive HTTP requests from web services client applications, to replace the support provided by the broker and execution group HTTP listeners.

"HTTP traffic handling in WebSphere Message Broker" on page 857
Before reading about HTTP traffic handling using the proxy servlet in an external Web container, ensure that you understand the basic structure of HTTP traffic handling in WebSphere Message Broker.

"HTTP traffic handling by using the proxy servlet in an external web servlet container" on page 859
Before you install and test the HTTP proxy servlet, ensure that you understand the structure of HTTP traffic handling by using the proxy servlet in an external web servlet container, such as IBM WebSphere Application Server or Apache Tomcat.

"HTTP proxy servlet; descriptions of required components" on page 863
The HTTP proxy servlet requires a number of components such as message flows, a proxy servlet, a servlet container, web addresses, the Broker component, and web services clients.

Related tasks:

"Installing the proxy servlet" on page 872
The installation of the proxy servlet is made up of a number of tasks. Use this topic to complete each of the tasks to then move onto testing the proxy servlet.

"Configuring the proxy servlet" on page 876
Configure the proxy servlet with the initialization parameters that are used by the proxy servlet. These parameters need to be configured for the broker environment that the proxy servlet is connecting to before the proxy servlet can be deployed to the servlet container.

"Testing the proxy servlet" on page 892
Test the proxy servlet with a message broker that receives HTTP requests and passes them to a message flow.

Enabling the WebSphere MQ listener for SOAP nodes for the proxy servlet to access:

Enable the WebSphere MQ listener on each execution group where message flows with SOAP nodes are deployed so that the proxy servlet can access the SOAPInput and SOAPReply nodes.

Before you begin

Before you start:

Before you enable the WebSphere MQ listener, you must have completed the following tasks:
- "Installing and customizing a web servlet container for the proxy servlet" on page 873
- "Configuring the proxy servlet" on page 876

If your message flow uses HTTPS, you must also set the proxy servlet, browser, application server, and broker to use HTTPS.

Procedure

1. Open a command window that is configured for your environment and enter the following command:

   ```
   mqsichangeproperties <broker_name> -e <execution_group_name> -o HTTPConnector
   -n enableMQListener -v true
   ```

   For example:

   ```
   mqsichangeproperties VSR1BK -e default -o HTTPConnector -n enableMQListener -v true
   ```

2. Stop and start the broker.
3. Enter the following command to verify the property value:

   ```
   mqsireportproperties <broker_name> -e <execution_group_name> -o HTTPConnector
   -n enableMQListener
   ```

   For example:

   ```
   mqsireportproperties VSR1BK -e default -o HTTPConnector –n enableMQListener
   ```

4. Repeat these steps for all execution groups where message flows with SOAP nodes are deployed.

Results

You have enabled the WebSphere MQ listener for all execution groups where message flows with SOAP nodes are deployed.

You are now ready to deploy the proxy servlet in the Web servlet container. For information about how to complete this task, see “Deploying the proxy servlet in the web servlet container” on page 891.

Related concepts:

“HTTP proxy servlet overview” on page 856
The HTTP proxy servlet is a Java servlet that you can use in an external web servlet container, such as IBM WebSphere Application Server or Apache Tomcat, to receive HTTP requests from web services client applications, to replace the support provided by the broker and execution group HTTP listeners.

“HTTP traffic handling in WebSphere Message Broker” on page 857
Before reading about HTTP traffic handling using the proxy servlet in an external Web container, ensure that you understand the basic structure of HTTP traffic handling in WebSphere Message Broker.

“HTTP traffic handling by using the proxy servlet in an external web servlet container” on page 859
Before you install and test the HTTP proxy servlet, ensure that you understand the structure of HTTP traffic handling by using the proxy servlet in an external web servlet container, such as IBM WebSphere Application Server or Apache Tomcat.

“HTTP proxy servlet; descriptions of required components” on page 863
The HTTP proxy servlet requires a number of components such as message flows, a proxy servlet, a servlet container, web addresses, the Broker component, and web services clients.

Related tasks:

“Installing the proxy servlet” on page 872
The installation of the proxy servlet is made up of a number of tasks. Use this topic to complete each of the tasks to then move onto testing the proxy servlet.

“Installing and customizing a web servlet container for the proxy servlet” on page 873
Download, install, and customize a web servlet container, such as WebSphere Application Server or Apache Tomcat, for the proxy servlet to use to receive HTTP requests from web services client applications.

“Configuring the proxy servlet” on page 876
Configure the proxy servlet with the initialization parameters that are used by the proxy servlet. These parameters need to be configured for the broker environment that the proxy servlet is connecting to before the proxy servlet can be deployed to the servlet container.

“Deploying the proxy servlet in the web servlet container” on page 891
Load and install the proxy servlet file in the web servlet container, such as

WebSphere Application Server or Apache Tomcat.

"Testing the proxy servlet" on page 892
Test the proxy servlet with a message broker that receives HTTP requests and passes them to a message flow.

Deploying the proxy servlet in the web servlet container:

Load and install the proxy servlet file in the web servlet container, such as WebSphere Application Server or Apache Tomcat.

Before you begin

The documentation in this section is based on deploying the proxy servlet in Apache Tomcat Version 6 on Windows XP and assumes WebSphere Message Broker is installed and running on the same server.

Before you deploy the proxy servlet in Apache Tomcat, you must complete the following tasks:

- "Installing and customizing a web servlet container for the proxy servlet" on page 873
- "Configuring the proxy servlet" on page 876
- "Enabling the WebSphere MQ listener for SOAP nodes for the proxy servlet to access" on page 889

Procedure

1. Open a web browser and enter web address http://localhost:8181 to open the Tomcat home page.
2. Click**Tomcat Manager** and enter the *admin* user ID and password.
3. Scroll down to the section **WAR file to deploy** and click **Browse** and search for the `HTTPVSR1BKproxyservlet.war` file.
4. Click **Deploy**. The <*context_root*> associated to the proxy servlet is the same as the WAR file name. It is possible to rename the `.war` file before deploying it to Tomcat to have a different <*context_root*>. See "HTTP proxy servlet; web addresses component" on page 868 for a description of the <*context_root*>.
5. Click **/HTTPVSR1BKproxyservlet** on the Applications view to test if the servlet is active. If the proxy servlet trace is enabled, the trace file shows that the servlet has received the request and rejected it.

Results

The proxy servlet is now ready to be tested with a message broker that receives HTTP (not HTTPS) requests and passes them to a message flow. For information about how to complete this task see:

- "Testing the proxy servlet" on page 892

Related concepts:

"HTTP proxy servlet overview" on page 856
The HTTP proxy servlet is a Java servlet that you can use in an external web servlet container, such as IBM WebSphere Application Server or Apache Tomcat, to receive HTTP requests from web services client applications, to replace the support provided by the broker and execution group HTTP listeners.

"HTTP traffic handling in WebSphere Message Broker" on page 857
Before reading about HTTP traffic handling using the proxy servlet in an external Web container, ensure that you understand the basic structure of HTTP traffic handling in WebSphere Message Broker.

"HTTP traffic handling by using the proxy servlet in an external web servlet container" on page 859
Before you install and test the HTTP proxy servlet, ensure that you understand the structure of HTTP traffic handling by using the proxy servlet in an external web servlet container, such as IBM WebSphere Application Server or Apache Tomcat.

"HTTP proxy servlet; descriptions of required components" on page 863
The HTTP proxy servlet requires a number of components such as message flows, a proxy servlet, a servlet container, web addresses, the Broker component, and web services clients.

Related tasks:

"Installing the proxy servlet" on page 872
The installation of the proxy servlet is made up of a number of tasks. Use this topic to complete each of the tasks to then move onto testing the proxy servlet.

"Installing and customizing a web servlet container for the proxy servlet" on page 873
Download, install, and customize a web servlet container, such as WebSphere Application Server or Apache Tomcat, for the proxy servlet to use to receive HTTP requests from web services client applications.

"Configuring the proxy servlet" on page 876
Configure the proxy servlet with the initialization parameters that are used by the proxy servlet. These parameters need to be configured for the broker environment that the proxy servlet is connecting to before the proxy servlet can be deployed to the servlet container.

"Enabling the WebSphere MQ listener for SOAP nodes for the proxy servlet to access" on page 889
Enable the WebSphere MQ listener on each execution group where message flows with SOAP nodes are deployed so that the proxy servlet can access the SOAPInput and SOAPReply nodes.

"Testing the proxy servlet"
Test the proxy servlet with a message broker that receives HTTP requests and passes them to a message flow.

Testing the proxy servlet

Test the proxy servlet with a message broker that receives HTTP requests and passes them to a message flow.

Before you begin

To test the proxy servlet, use an existing web services client application or write your own SSL test client application using Java.

Before you test the proxy servlet, you must have completed the following tasks:
- "Installing and customizing a web servlet container for the proxy servlet" on page 873
- "Configuring the proxy servlet" on page 876
- "Enabling the WebSphere MQ listener for SOAP nodes for the proxy servlet to access" on page 889
- "Deploying the proxy servlet in the web servlet container" on page 891

Procedure

To test the proxy servlet, complete the following steps:

1. Install a client application that can send HTTP or SSL (HTTPS) requests. There are several options that can be used, such as:

- Nettool: An open source graphical utility available from the Internet at: Nettool downloads website. You can use this tool to send HTTP requests or to tunnel a request.
- OpenSSL: Another open source tool available from the Internet at: OpenSSL downloads website that can be used to send SSL (HTTPS) requests to the web servlet container.
- A broker message flow that has an HTTPRequest or SOAPRequest node can generate and send HTTP requests to an HTTP listener.
- A web browser using web pages or Java Server Pages (JSP) that can send HTTP POST requests. Most web browsers support HTTP and SSL (HTTPS).
- A client application that sends requests by using HTTP, or (SSL) HTTPS, or both.

2. Configure the proxy servlet to access the message broker in the `web.xml` file.
3. Configure a message flow with HTTP and SOAP input and reply nodes. The message flow receives the messages from the proxy servlet. If an HTTP and SOAP reply node is configured, responses are sent back to the proxy servlet.

Results

You have now finished testing the proxy servlet.

Related concepts:

"HTTP proxy servlet overview" on page 856
The HTTP proxy servlet is a Java servlet that you can use in an external web servlet container, such as IBM WebSphere Application Server or Apache Tomcat, to receive HTTP requests from web services client applications, to replace the support provided by the broker and execution group HTTP listeners.

"HTTP traffic handling in WebSphere Message Broker" on page 857
Before reading about HTTP traffic handling using the proxy servlet in an external Web container, ensure that you understand the basic structure of HTTP traffic handling in WebSphere Message Broker.

"HTTP traffic handling by using the proxy servlet in an external web servlet container" on page 859
Before you install and test the HTTP proxy servlet, ensure that you understand the structure of HTTP traffic handling by using the proxy servlet in an external web servlet container, such as IBM WebSphere Application Server or Apache Tomcat.

"HTTP proxy servlet; descriptions of required components" on page 863
The HTTP proxy servlet requires a number of components such as message flows, a proxy servlet, a servlet container, web addresses, the Broker component, and web services clients.

Related tasks:

"Installing the proxy servlet" on page 872
The installation of the proxy servlet is made up of a number of tasks. Use this topic to complete each of the tasks to then move onto testing the proxy servlet.

"Installing and customizing a web servlet container for the proxy servlet" on page 873
Download, install, and customize a web servlet container, such as WebSphere Application Server or Apache Tomcat, for the proxy servlet to use to receive HTTP requests from web services client applications.

"Configuring the proxy servlet" on page 876
Configure the proxy servlet with the initialization parameters that are used by the proxy servlet. These parameters need to be configured for the broker environment that the proxy servlet is connecting to before the proxy servlet can be deployed to

the servlet container.

"Enabling the WebSphere MQ listener for SOAP nodes for the proxy servlet to access" on page 889
Enable the WebSphere MQ listener on each execution group where message flows with SOAP nodes are deployed so that the proxy servlet can access the SOAPInput and SOAPReply nodes.

"Deploying the proxy servlet in the web servlet container" on page 891
Load and install the proxy servlet file in the web servlet container, such as WebSphere Application Server or Apache Tomcat.

Configuring a WebSphere Message Broker as a WebSphere MQ service

If you want to operate your WebSphere Message Broker as a WebSphere MQ service.

About this task

- "Starting and stopping a WebSphere Message Broker as a WebSphere MQ service"
- "Modifying the WebSphere MQ service for a broker" on page 896
- "Deleting the WebSphere MQ service for a broker" on page 897
- "Reporting and displaying the status of WebSphere Message Broker that runs as a WebSphere MQ service" on page 897

Related tasks:

Chapter 7, "Configuring brokers for test and production environments," on page 579
Create one or more brokers on one or more computers, and configure them on your test and production systems to process messages that contain your business data.

"Creating the multi-instance broker" on page 837
How you create the multi-instance broker.

Starting and stopping a WebSphere Message Broker as a WebSphere MQ service

Configuring a WebSphere Message Broker to start and stop as a WebSphere MQ service.

Before you begin

Before you start:

Ensure that you make the mqm user ID a member of the mqbrkrs group. On Windows, you must reboot your workstation for the change to take effect.

About this task

To configure a WebSphere Message Broker to run as a WebSphere MQ service, use one of the following options:

Procedure

- Create a broker to start as a WebSphere MQ service by using the **mqsicreatebroker** command with the **-d** parameter. For example,

```
mqsicreatebroker MyBroker –q MyQMGR –d defined
```

where

MyBroker
Is the name of the broker that you want to start and stop as a WebSphere MQ service.

MyQMGR
Is the name of the queue manager associated with the broker.

- Modify an existing broker to start as a WebSphere MQ service by using the **mqsichangebroker** command with the **-d** parameter. For example,

```
mqsichangebroker MyBroker –d defined
```

where

MyBroker
Is the name of the broker that you want to start and stop as a WebSphere MQ service.

Results

When you have configured a broker to start and stop as a WebSphere MQ service:

- The broker starts and stops automatically when its associated queue manager starts and stops. For a multi-instance broker, this action can occur during failover of the active queue manager.
- A multi-instance broker cannot be started in standby mode when its WebSphere MQ service is defined as active.
- You can stop the broker manually by using the **mqsistop** command, but the broker does not restart until the queue manager is stopped and started again.

  Alternatively, you can start the broker manually with the **mqsistart** command, which invokes the WebSphere MQ service to start the broker.
- On UNIX systems the broker environment is inherited from WebSphere MQ. Set any required environment variables (such as ODBCINI) by using a script in the `work_path/common/profiles` directory. See "Command environment: Linux and UNIX systems" on page 310 for more information.

Related tasks:

"Creating the WebSphere MQ multi-instance queue manager" on page 830
How you create the WebSphere MQ multi-instance queue manager that you need for your multi-instance WebSphere Message Broker.

"Creating the multi-instance broker" on page 837
How you create the multi-instance broker.

"Configuring a WebSphere Message Broker as a WebSphere MQ service" on page 894
If you want to operate your WebSphere Message Broker as a WebSphere MQ service.

Related reference:

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

Modifying the WebSphere MQ service for a broker

How you modify the WebSphere MQ service for a broker.

About this task

To alter the status of a WebSphere Message Broker so that it runs as a WebSphere MQ service use the **mqsichangebroker** command.

Carry out the following procedure:

Procedure

1. Stop the broker and its associated queue manager. If it is a multi-instance broker, you must stop all broker instances, and the multi-instance queue manager, if you want the changes to take affect immediately after the queue manager restarts.
2. Use the **mqsichangebroker** command with the **–d defined** option to activate the WebSphere MQ service as follows:

   ```
   mqsichangebroker MyBroker –d defined
   ```

   where

   MyBroker
   Is the name of the broker that you want to modify.

   If the WebSphere MQ service did not previously exist, it is defined, and the service CONTROL attribute is set to QMGR.

   You should no longer start the broker with the **mqsistart** command. The broker starts and stops automatically when the queue manager starts and stops.

   You can stop the broker manually with the **mqsistop** command, but the broker will not restart until the queue manager is stopped and started again. For a multi-instance broker this can occur during failover of the active queue manager.
3. Use the **mqsichangebroker** command with the **–d undefined** option to remove the WebSphere MQ service as follows:

   ```
   mqsichangebroker MyBroker –d undefined
   ```

   where

   MyBroker
   Is the name of the broker that you want to modify.

   When a WebSphere MQ service is being removed, and the queue manager is stopped, the **mqsichangebroker** command needs to start the queue manager to perform the deletion of the service. The command then stops the queue manager. The following example shows a possible output:

   ```
   >mqsichangebroker BK3 -d undefined
   WebSphere MQ queue manager 'QM3' starting.
   5 log records accessed on queue manager 'QM3' during the log replay phase.
   Log replay for queue manager 'QM3' complete.
   Transaction manager state recovered for queue manager 'QM3'.
   WebSphere MQ queue manager 'QM3' started.
   Waiting for queue manager 'QM3' to end.
   Waiting for queue manager 'QM3' to end.
   WebSphere MQ queue manager 'QM3' ended.
   BIP8071I: Successful command completion.
   ```

   You can start the broker manually using the **mqsistart** command, which invokes the WebSphere MQ service to restart the broker.

Related tasks:

"Starting and stopping a WebSphere Message Broker as a WebSphere MQ service" on page 894
Configuring a WebSphere Message Broker to start and stop as a WebSphere MQ service.

"Creating the multi-instance broker" on page 837
How you create the multi-instance broker.

"Configuring a WebSphere Message Broker as a WebSphere MQ service" on page 894
If you want to operate your WebSphere Message Broker as a WebSphere MQ service.

Related reference:

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

Deleting the WebSphere MQ service for a broker

How you delete a WebSphere MQ service for a broker.

About this task

You can use either of the following methods to delete the WebSphere MQ service from a WebSphere Message Broker:

Procedure

1. Use the **mqsichangebroker** command with the **–d undefined** option, as described in step 3 on page 896 of "Modifying the WebSphere MQ service for a broker" on page 896 This reverts the broker to manual stop and start operation.
2. Use the **mqsideletebroker** command. Using this command also removes a previously configured WebSphere MQ service when the broker is removed.

Related tasks:

"Modifying the WebSphere MQ service for a broker" on page 896
How you modify the WebSphere MQ service for a broker.

"Configuring a WebSphere Message Broker as a WebSphere MQ service" on page 894
If you want to operate your WebSphere Message Broker as a WebSphere MQ service.

Related reference:

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

Reporting and displaying the status of WebSphere Message Broker that runs as a WebSphere MQ service

How you report and display the status of a WebSphere Message Broker that runs as a WebSphere MQ service

Procedure

1. Use the **mqsireportbroker** command to show the property that displays the broker running as a WebSphere MQ service.

   The `StartAsMQService` property is set to **defined** if the broker is set to start as a WebSphere MQ service, otherwise this property is set to **undefined**.

   See the Windows, Linux, and UNIX systems version of the "**mqsireportbroker** command" on page 3919 for an example output from the command.
2. Use the **mqsilist** command to display the status of a broker that starts as a WebSphere MQ service. Message bip1297 is output when a broker is stopped, but will start as a WebSphere MQ service.

Related tasks:

"Creating the multi-instance broker" on page 837
How you create the multi-instance broker.

"Configuring a WebSphere Message Broker as a WebSphere MQ service" on page 894
If you want to operate your WebSphere Message Broker as a WebSphere MQ service.

Related reference:

"**mqsilist** command" on page 3882
Use the **mqsilist** command to list installed brokers and their associated resources.

"**mqsireportbroker** command" on page 3919
Use the **mqsireportbroker** command to display broker registry entries.

Chapter 8. Administering brokers and broker resources

Administering brokers and associated broker resources includes the tasks that you perform frequently to activate and manage those resources. Choose the method you prefer to administer your brokers and associated resources.

About this task

Administration of brokers includes the following tasks:

- "Managing brokers" on page 900
- "Managing execution groups" on page 935
- "Managing message flows" on page 950
- "Developing applications that use the Administration API" on page 956
- "Managing resources used by brokers" on page 1001
- "Administering Java applications" on page 1005
- "Accessing Administration log information" on page 1006
- "Changing the location of the work path" on page 1011
- "Backing up resources" on page 1013

These tasks can be performed by using one, or more, of the administrative techniques supported by WebSphere Message Broker:

- The WebSphere Message Broker Toolkit
- The WebSphere Message Broker Explorer
- The WebSphere Message Broker commands
- The Administration API for WebSphere Message Broker (also known as the CMP API)

For each task, the administrative techniques that you can use are identified.

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

"WebSphere Message Broker Toolkit" on page 31
The WebSphere Message Broker Toolkit is an integrated development environment and graphical user interface based on the Eclipse platform.

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

"The Administration API for WebSphere Message Broker" on page 54
The Administration API for WebSphere Message Broker is a programming interface that your applications can use to control brokers and their resources through a remote interface.

Related tasks:

"Deploying a user exit" on page 3116
Deploy your user exit to the broker.

"Monitoring message flow performance" on page 3279
You can configure your message flow to collect statistics about the operation and behavior of your message flows. Use this information to assess the performance of

your message flows.

Chapter 7, “Configuring brokers for test and production environments,” on page 579
Create one or more brokers on one or more computers, and configure them on your test and production systems to process messages that contain your business data.

Related reference:

“Administration in z/OS” on page 3979
In the z/OS environment, commands are issued through the console and others in batch jobs.

Managing brokers

Work with your existing brokers to manage their connections and their active status by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer. You can use the Administration API (also known as the CMP API) to complete some of these actions.

About this task

- “Connecting to a local broker using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer” on page 901
- “Connecting to a remote broker” on page 902
- “Connecting to a remote broker on z/OS in the WebSphere Message Broker Explorer” on page 904
- “Disconnecting from a broker in the WebSphere Message Broker Explorer” on page 909
- “Starting and stopping a broker” on page 921
- “Modifying a broker” on page 631
- “Viewing broker properties” on page 927
- “Preparing the environment for WebSphere Adapters nodes” on page 717
- “Preparing the environment for IMS nodes” on page 731
- “Preparing the environment for the CICSRequest node” on page 736
- “Changing the operation mode of your broker” on page 655
- “Checking the operation mode of your broker” on page 657
- “Moving from WebSphere Message Broker on a distributed system to z/OS” on page 818
- “Starting a WebSphere MQ queue manager as a Windows service” on page 929
- “Stopping a WebSphere MQ queue manager when you stop a broker” on page 929
- “Deleting a broker” on page 930

Related concepts:

“The broker environment” on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related tasks:

Chapter 6, “Configuring brokers for development environments,” on page 563
Set up application development environments on Linux on x86 or Windows to create, test, and deploy message flows and associated resources.

Chapter 7, “Configuring brokers for test and production environments,” on page 579
Create one or more brokers on one or more computers, and configure them on your test and production systems to process messages that contain your business data.

“Managing execution groups” on page 935
Work with your existing execution groups to manage the message flows that you have deployed to them by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer. You can use the command line, and the Administration API (also known as the CMP API) to complete some of these actions.

“Managing message flows” on page 950
Work with your existing messages that you have deployed to execution groups by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer. You can use the Administration API (also known as the CMP API) to complete some of these actions.

“Configuring a WebSphere Message Broker as a WebSphere MQ service” on page 894
If you want to operate your WebSphere Message Broker as a WebSphere MQ service.

Related information:

Administration API for WebSphere Message Broker (CMP API)

Connecting to a local broker using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer

To administer a broker by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer, you must be connected to the broker.

Before you begin

Before you start:

- Create a broker

About this task

In the WebSphere Message Broker Toolkit, local brokers are automatically connected. You must manually connect to remote brokers. In the WebSphere Message Broker Explorer you can choose to disconnect from both local and remote brokers. You can also choose to automatically reconnect to a broker when you start the WebSphere Message Broker Explorer.

To connect to a broker in the WebSphere Message Broker Explorer:

Procedure

1. In the Navigator view, expand the Brokers folder.
2. Right-click the broker you want to connect to, and click **Connect**.

Results

You can now view properties and configure your broker by using the WebSphere Message Broker Toolkit or the WebSphere Message Broker Explorer.

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

Related tasks:

"Connecting to a remote broker"
To administer your brokers by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer, you must connect to the broker.

"Automatically reconnecting to a broker" on page 908
You can configure each broker so that the WebSphere Message Broker Explorer automatically reconnects to it if the connection is lost; for example, if the network connection to a remote queue manager fails.

"Creating a broker using the WebSphere Message Broker Explorer" on page 618
On Linux on x86 or Windows, you can create brokers by using the WebSphere Message Broker Explorer.

"Creating an execution group using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 937
Use the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer to create execution groups on your broker.

Chapter 8, "Administering brokers and broker resources," on page 899
Administering brokers and associated broker resources includes the tasks that you perform frequently to activate and manage those resources. Choose the method you prefer to administer your brokers and associated resources.

Connecting to a remote broker

To administer your brokers by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer, you must connect to the broker.

Before you begin

Before you start:

- Create a broker

Before you can connect to a remote broker, the broker and its queue manager must be running. You must also complete the following steps:

- Ensure that a command server is running on the queue manager.
- Ensure a server-connection channel has been defined on the broker queue manager. When you create a broker, a default SVRCONN channel, called `SYSTEM.BKR.CONFIG`, is created. This channel supports connections from one or more remote clients to the broker.
- Create a TCP/IP listener on the broker queue manager.
- Ensure that the listener is running.
- Create a group and a user ID on the computer where the broker is running by using the local system facilities; for example, the **Groupadd** command on Linux on x86. Add the user ID to the group.
- Run the **setmqaut** command on the broker queue manager to grant authority to the group:

  ```
  setmqaut -m queueManager -t qmgr -g group +connect
  ```

About this task

You can create a connection to a remote broker, or you can create a connection to a remote broker by using settings from a `.broker` file. Right-click the Brokers folder,

and click **Connect to a Remote Broker Using *.broker File**. Select the `.broker` file, and click **Open**. The connection to the remote broker is created in your workspace.

To create a connection to a remote broker:

Procedure

1. Open the WebSphere Message Broker Explorer, or open the Brokers view in the WebSphere Message Broker Toolkit.
2. To create a connection to a remote broker, right-click the Brokers folder, and click **Connect to a Remote Broker**.
3. In the Connect to a broker wizard, enter the following values:
   a. The value for the **Queue Manager name** that your remote broker is using.
   b. The **Host** name or IP address of the computer on which your broker is running.
   c. The TCP **Port** on which the WebSphere MQ queue manager is listening (the default is 1414). This property must be a valid positive number.
   d. Optional: The name of the server-connection channel in the **SVRCONN Channel Name** field. The channel has a default name of `SYSTEM.BKR.CONFIG`.

      You can create more than one server-connection channel and define a different SSL certificate on each channel to enforce; for example, users with `view` access on to one channel and users with `deploy` access on to a different channel.

      You can then create WebSphere MQ exits on each channel to provide additional authentication of the WebSphere MQ message sent to the broker.

      You must create the server-connection channel manually on the broker queue manager by using one of the following options:

      - The WebSphere MQ **`runmqsc`** command to create a channel with options `CHLTYPE(SVRCONN)` and `TRPTYPE(TCP)`.
      - The WebSphere MQ Explorer to create a server-connection channel with the transmission protocol set to TCP.

      For more information see your WebSphere MQ documentation.

      If you do not change the name, or attempt to delete it, the default name of `SYSTEM.BKR.CONFIG` is assumed. The name of the server-connection channel is changed only if you enter another name in place of `SYSTEM.BKR.CONFIG`.
   e. Optional: The **Class** of the Security Exit required to connect to the WebSphere MQ queue manager. This property must be a valid Java class name, but you can leave this field blank if it does not apply to your domain connection.
   f. Optional: The **JAR File Location** for the Security Exit required to connect to the WebSphere MQ queue manager. Click **Browse** to find the file location. If this field does not apply to your domain connection, you can leave it blank. If you enter a Security Exit **Class**, you must provide a **JAR File Location**.
   g. Optional: The **Cipher Suite**, **Distinguished Names**, **CRL Name List**, **Key Store**, and **Trust Store** parameters are required to enable SSL. For more information, see "Implementing SSL authentication" on page 504. The **Cipher Suite** field lists available cipher suites. Click **More** to configure Custom SSL Cipher Suites in the Broker Administration Preferences window. If a **Cipher Suite** is not specified, all of the other fields in the SSL section are unavailable.

If you specify a keystore or truststore when you define the connection information, you are prompted for the keystore or truststore when you connect to the remote broker.

4. Click **Finish** to connect to the remote broker.

Results

You can now view properties and configure your broker using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer.

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

"Security exits" on page 354
Use security exit programs to verify that the partner at the other end of a connection is genuine.

Related tasks:

"Considering security for the WebSphere Message Broker Toolkit and WebSphere Message Broker Explorer" on page 500
Set up appropriate levels of security for the WebSphere Message Broker Toolkit and WebSphere Message Broker Explorer.

"Creating a broker using the WebSphere Message Broker Explorer" on page 618
On Linux on x86 or Windows, you can create brokers by using the WebSphere Message Broker Explorer.

"Creating an execution group using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 937
Use the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer to create execution groups on your broker.

Chapter 8, "Administering brokers and broker resources," on page 899
Administering brokers and associated broker resources includes the tasks that you perform frequently to activate and manage those resources. Choose the method you prefer to administer your brokers and associated resources.

Connecting to a remote broker on z/OS in the WebSphere Message Broker Explorer

To administer your brokers by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer, you must connect to the broker.

Before you begin

Before you start:

- Create a broker

About this task

To connect to a remote broker on z/OS:

Procedure

1. In the Navigator view, or Brokers view, right-click the Brokers folder, and click **Connect** > **Remote Broker**.
2. In the Connect to a broker wizard, enter the following values:
   a. The value for the **Queue Manager name** that your remote broker is using.

b. The **Host** name or IP address of the machine on which your broker is running.

c. The TCP **Port** on which the WebSphere MQ queue manager is listening (the default is 1414). This property must be a valid positive number.

d. Optional: The name of the server-connection channel in the **SVRCONN Channel Name** field. The channel has a default name of `SYSTEM.BKR.CONFIG`.

   You can create more than one server-connection channel and define a different SSL certificate on each channel to enforce; for example, users with `view` access on to one channel and users with `deploy` access on to a different channel.

   You can then create WebSphere MQ exits on each channel to provide additional authentication of the WebSphere MQ message sent to the broker.

   You must create the server-connection channel manually on the queue manager for the broker by using one of the following options:

   - The WebSphere MQ **runmqsc** command to create a channel with options `CHLTYPE(SVRCONN)` and `TRPTYPE(TCP)`.
   - The WebSphere MQ Explorer to create a server-connection channel with the transmission protocol set to TCP.

   For more information see your WebSphere MQ documentation.

   The default name of `SYSTEM.BKR.CONFIG` is assumed if you do not change the name, or attempt to delete it. The name of the server-connection channel is changed only if you enter another name in place of `SYSTEM.BKR.CONFIG`.

e. Optional: The **Class** of the Security Exit required to connect to the WebSphere MQ queue manager. This property must be a valid Java class name, but you can leave this field blank if it does not apply to your domain connection.

f. Optional: The **JAR File Location** for the Security Exit required to connect to the WebSphere MQ queue manager. Click **Browse** to find the file location. You can leave this field blank if it does not apply to your domain connection. You must provide a **JAR File Location** if you enter a Security Exit **Class**.

g. Optional: The **Cipher Suite**, **Distinguished Names**, **CRL Name List**, **Key Store**, and **Trust Store** parameters are required to enable SSL. For more information, see "Implementing SSL authentication" on page 504. The **Cipher Suite** field displays available cipher suites. Click **More** to configure Custom SSL Cipher Suites in the Broker Administration Preferences window. If a **Cipher Suite** is not specified, all of the other fields in the SSL section are unavailable.

3. Click **Finish** to connect to the remote broker.

Results

You can now view properties and configure your broker by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer.

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

"Security exits" on page 354
Use security exit programs to verify that the partner at the other end of a connection is genuine.

Related tasks:

"Considering security for the WebSphere Message Broker Toolkit and WebSphere Message Broker Explorer" on page 500
Set up appropriate levels of security for the WebSphere Message Broker Toolkit and WebSphere Message Broker Explorer.

"Creating a broker using the WebSphere Message Broker Explorer" on page 618
On Linux on x86 or Windows, you can create brokers by using the WebSphere Message Broker Explorer.

"Creating an execution group using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 937
Use the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer to create execution groups on your broker.

Chapter 8, "Administering brokers and broker resources," on page 899
Administering brokers and associated broker resources includes the tasks that you perform frequently to activate and manage those resources. Choose the method you prefer to administer your brokers and associated resources.

"Connecting to a remote broker" on page 902
To administer your brokers by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer, you must connect to the broker.

Importing broker definitions into the WebSphere Message Broker Explorer

Import broker connection details that have been created by another user into your session of the WebSphere Message Broker Explorer or WebSphere Message Broker Toolkit, to use the broker.

About this task

The properties of a broker connection can be exported from one instance of the WebSphere Message Broker Explorer, and imported into a different instance. You can also import a broker connection into an instance of the WebSphere Message Broker Toolkit. The properties of the broker connection are stored in a `.broker` file.

Importing a broker definition into the WebSphere Message Broker Toolkit

About this task

To import broker connection properties from a `.broker` file into the WebSphere Message Broker Toolkit:

Procedure

1. In the Brokers view, right-click the Brokers folder and click **Add Remote Broker From *.broker**.
2. Navigate to the broker connection files that you want to import, and click **Open**.

Results

The `.broker` file is imported, and the broker is displayed in the Brokers folder. You can now connect to the broker.

Importing a broker definition into the WebSphere Message Broker Explorer

About this task

To import broker connection properties from a `.broker` file into the WebSphere Message Broker Explorer:

Procedure

1. In the Navigator view, right-click the Brokers folder and click **Import *.broker**.
2. Navigate to the broker connection files that you want to import, and click **OK**.

Results

The `.broker` file is imported, and the broker is displayed in the Brokers folder. You can now connect to the broker.

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

Related tasks:

"Configuring brokers in the WebSphere Message Broker Explorer" on page 635
Configure your local and remote brokers by using the WebSphere Message Broker Explorer.

"Exporting broker definitions from the WebSphere Message Broker Explorer"
Use the WebSphere Message Broker Explorer to export broker connection details for another user.

"Connecting to a remote broker" on page 902
To administer your brokers by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer, you must connect to the broker.

"Connecting to a remote broker on z/OS in the WebSphere Message Broker Explorer" on page 904
To administer your brokers by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer, you must connect to the broker.

Exporting broker definitions from the WebSphere Message Broker Explorer

Use the WebSphere Message Broker Explorer to export broker connection details for another user.

About this task

The properties of a broker connection can be exported from one instance of the WebSphere Message Broker Explorer and imported into a different instance. The properties of the broker connection are stored in a `.broker` file.

To a export broker connection properties to a `.broker` file:

Procedure

1. In the Navigator view, expand the Brokers folder.
2. Select the broker for which you want to export the connection details for, and click **Export *.broker**.

3. Select a location to save the `.broker` file, and click **Save**.

Results

The `.broker` file is exported. Another user can now import the `.broker` file to connect to the broker.

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

Related tasks:

"Configuring brokers in the WebSphere Message Broker Explorer" on page 635
Configure your local and remote brokers by using the WebSphere Message Broker Explorer.

"Importing broker definitions into the WebSphere Message Broker Explorer" on page 906
Import broker connection details that have been created by another user into your session of the WebSphere Message Broker Explorer or WebSphere Message Broker Toolkit, to use the broker.

"Connecting to a remote broker" on page 902
To administer your brokers by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer, you must connect to the broker.

"Connecting to a remote broker on z/OS in the WebSphere Message Broker Explorer" on page 904
To administer your brokers by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer, you must connect to the broker.

Automatically reconnecting to a broker

You can configure each broker so that the WebSphere Message Broker Explorer automatically reconnects to it if the connection is lost; for example, if the network connection to a remote queue manager fails.

About this task

If you manually disconnect a broker, the broker is not automatically reconnected until the next time that you close and restart the WebSphere Message Broker Explorer. To configure a broker to automatically reconnect when you start the WebSphere Message Broker Explorer:

Procedure

1. In the Navigator view, expand the Brokers folder.
2. Right-click your broker, and select **Properties**.
3. Ensure **Autoreconnect** is selected in the General tab.
4. Click **OK**.

Results

The broker is automatically reconnected when you next start the WebSphere Message Broker Explorer.

What to do next

If you want to configure the broker so that it is not automatically reconnected when you start the WebSphere Message Broker Explorer, clear **Autoreconnect** in the broker properties window.

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

Related tasks:

"Creating a broker using the WebSphere Message Broker Explorer" on page 618
On Linux on x86 or Windows, you can create brokers by using the WebSphere Message Broker Explorer.

"Connecting to a local broker using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 901
To administer a broker by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer, you must be connected to the broker.

"Connecting to a remote broker" on page 902
To administer your brokers by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer, you must connect to the broker.

"Connecting to a remote broker on z/OS in the WebSphere Message Broker Explorer" on page 904
To administer your brokers by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer, you must connect to the broker.

Disconnecting from a broker in the WebSphere Message Broker Explorer

You can disconnect from all brokers that you are not currently configuring to improve performance in the WebSphere Message Broker Explorer.

About this task

To disconnect from a broker WebSphere Message Broker Explorer:

Procedure

1. In the Navigator view, expand the Brokers folder.
2. Right-click the broker you want to disconnect from, and click **Disconnect**.

Results

You have disconnected from the selected broker.

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

Related tasks:

"Creating a broker using the WebSphere Message Broker Explorer" on page 618
On Linux on x86 or Windows, you can create brokers by using the WebSphere Message Broker Explorer.

"Connecting to a local broker using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 901
To administer a broker by using the WebSphere Message Broker Toolkit or

WebSphere Message Broker Explorer, you must be connected to the broker.

"Creating an execution group using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 937
Use the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer to create execution groups on your broker.

Chapter 8, "Administering brokers and broker resources," on page 899
Administering brokers and associated broker resources includes the tasks that you perform frequently to activate and manage those resources. Choose the method you prefer to administer your brokers and associated resources.

Using the Administration Queue

The Administration Queue shows pending and submitted tasks that have been sent to the broker, and are waiting to be processed. Use the WebSphere Message Broker Explorer to view details of the tasks, and to cancel tasks in the queue.

About this task

Each broker that is connected in the WebSphere Message Broker Explorer displays an icon labeled **Administration Queue**. When you click the Administration Queue icon, tasks submitted to the broker are displayed in the Administration Queue QuickView in the Content pane. The Administration Queue icon changes to indicate the number of messages on the Administration Queue.

The following properties are displayed for each task on the Administration Queue:
- Order
- Status
- Username
- Operation Type
- Object Name
- Object Type
- Creation Time
- Elapsed Time
- Identifier

You can remove tasks from the Administration Queue if you want to cancel requests that have been sent to a broker. To remove tasks from the Administration Queue:

Procedure

1. In the Navigator view, expand the Brokers folder.
2. Expand the broker with which you want to work, and right-click **Administration Queue**.
3. Click **Cancel Work Items** The Administration Queue window is displayed.
4. Select the task or tasks that you want to remove from the Administration Queue, and click **Cancel Work Items**.

Results

The selected tasks are removed from the Administration Queue.

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

Related tasks:

"Configuring brokers in the WebSphere Message Broker Explorer" on page 635
Configure your local and remote brokers by using the WebSphere Message Broker Explorer.

Chapter 8, "Administering brokers and broker resources," on page 899
Administering brokers and associated broker resources includes the tasks that you perform frequently to activate and manage those resources. Choose the method you prefer to administer your brokers and associated resources.

Chapter 12, "Performance and monitoring," on page 3251
You can change various aspects of your broker configuration to tune brokers and message flows, and monitor message flows and publish/subscribe applications.

Working with Service Federation Management (SFM)

You can use a broker as a connectivity server that can be administered by a Service Federation Management (SFM) console provided with WebSphere Service Registry and Repository (WSRR) V7.0. The SFM console can then configure SFM proxies in the broker.

Your enterprise might have separate enterprise service buses (ESBs) in different business units. Each ESB and associated service registry constitute a separate domain of connected service applications. This configuration can result in expensive duplication of applications between domains and also in increased development effort to implement application connectivity across domains.

SFM, provided in WebSphere Service Registry and Repository (WSRR) V7.0, allows you to establish bridges between separate ESBs, allowing services and applications to be shared between domains. SFM provides:

- A federation model which provides a unifying view of federation-relevant content.
- A protocol, Service Control Management Protocol (SCMP), which accesses the service connectivity and registry components supporting a domain. SCMP is based on the Atom Publishing Protocol (Atom).
- A console for controlling service domains.

By using SFM, you can configure services in one domain so that they are available to service consumers in another domain; the service endpoints in one domain are manifested as service proxy endpoints in another domain.

You can use the SFM console to administer a broker as an SFM connectivity server to enable the creation of proxies running in a broker execution group. You can share services between domains by using proxies. You must use the **mqsichangeproperties** command to enable a broker to be administered by the SFM console. You can then use the SFM console federation facilities so that SCMP Atom requests can be issued to the broker on the configured HTTP or HTTPS port.

After you enable WebSphere Message Broker as an SFM connectivity server, the execution groups in a broker are available to act as SFM connectivity providers. Connectivity providers are added in the SFM console to domains to provide an ESB facility which can then create and host proxy instances to enable the sharing of services.

Service federation proxy instances are runtime resources that are separate from WebSphere Message Broker message flows, and are therefore not listed in the

execution group. The service federation proxy instances of an execution group can be reported by the Service Federation Management properties of that execution group.

To enable a broker for SFM, or configure the SFM properties of an execution, follow the appropriate link:

- "Enabling a broker for Service Federation Management"
- "Configuring the Service Federation Management properties of an execution group" on page 914

For more information about SFM, see the WebSphere Service Registry and Repository V7.0 Information Center at http://publib.boulder.ibm.com/infocenter/sr/v7r0/topic/com.ibm.sr.sfm.doc/wsrr_sfm_homepage.html.

Related tasks:

"Enabling a broker for Service Federation Management"
Use the **mqsichangeproperties** command to enable a broker for Service Federation Management (SFM).

"Configuring the Service Federation Management properties of an execution group" on page 914
Using the **mqsichangeproperties** command, you can configure the Service Federation Management (SFM) properties of an execution group in a broker that has been enabled for SFM. These properties determine how proxies are created.

Related information:

WebSphere Service Registry and Repository Version 7.0 Information Center online

Enabling a broker for Service Federation Management

Use the **mqsichangeproperties** command to enable a broker for Service Federation Management (SFM).

Before you begin

Before you start:

- Create a broker

About this task

Procedure

1. Run the **mqsichangeproperties** command to enable SFM in the broker.

   To enable SFM in the broker and to use HTTP as the protocol for communication between SFM and the broker, run the following command:

   ```
   mqsichangeproperties broker_name -b servicefederation -o scmp
   -n enabled,enableSSL -v true,false
   ```

   To enable SFM in the broker and to use HTTPS as the protocol for communication between SFM and the broker, run the following command:

   ```
   mqsichangeproperties broker_name -b servicefederation -o scmp
   -n enabled,enableSSL -v true,true
   ```

2. You must also set the hostname and context root for the service document URL that will be required when you add the broker to the Service Federation Console. Use the full domain qualified hostname for your broker to ensure reliable addressing.

   ```
   mqsichangeproperties broker_name -b servicefederation -o scmp
   -n hostname,contextRoot -v brokerhost.ibm.com,/scmp
   ```

3. Use the **mqsireportproperties** command to review the SFM properties that you have set by using the **mqsichangeproperties** command:

   ```
   mqsireportproperties broker_name -b servicefederation -o scmp -r
   ```

4. Run the **mqsichangeproperties** command to configure the properties relating to the HTTP or HTTPS port to be used. If you are using HTTP:

   ```
   mqsichangeproperties broker_name -b servicefederation -o HTTPConnector -n port
   -v 8085
   ```

   If you are using HTTPS:

   ```
   mqsichangeproperties broker_name -b servicefederation -o HTTPSConnector
   -n port,keystoreFile,keystorePass -v 8086,keys.jks,keypwd
   ```

5. If you need to secure the SFM HTTP or HTTPS port with basic access authentication user name and password credentials, run the **mqsisetdbparms** command and use a resource type of `sfm::scmp`:

   ```
   mqsisetdbparms broker_name -n sfm::scmp -u ConsoleUsername -p
   ConsolePwd
   ```

6. Stop the broker by using the **mqsistop** command.
7. Start the broker by using the **mqsistart** command. Review the system log to ensure that the Service Federation listener has successfully started. The message BIP3769 also confirms the service document URL value that must be configured into the SFM console.

   ```
   Service Federation SCMP listener initialized. Additional information :
   ''SCMP Service Document URL''
   ''http://brokerhost.ibm.com:8085/scmp/servicedocument''
   ```

What to do next

You have now enabled the broker for SFM. As a consequence of enabling the broker for SFM, the execution groups in the broker are also enabled for SFM; they can now create and host the proxy instances to provide service availability. You might want to configure the properties of the execution group that relate to the creation of proxy instances; see "Configuring the Service Federation Management properties of an execution group" on page 914.

Related concepts:

"Working with Service Federation Management (SFM)" on page 911
You can use a broker as a connectivity server that can be administered by a Service Federation Management (SFM) console provided with WebSphere Service Registry and Repository (WSRR) V7.0. The SFM console can then configure SFM proxies in the broker.

Related tasks:

"Configuring the Service Federation Management properties of an execution group" on page 914
Using the **mqsichangeproperties** command, you can configure the Service Federation Management (SFM) properties of an execution group in a broker that has been enabled for SFM. These properties determine how proxies are created.

Chapter 8, "Administering brokers and broker resources," on page 899
Administering brokers and associated broker resources includes the tasks that you perform frequently to activate and manage those resources. Choose the method you prefer to administer your brokers and associated resources.

"Starting and stopping a broker" on page 921
Run the appropriate command to start or stop a broker.

Related reference:

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and

properties of broker resources.

"Parameter values for the servicefederation component" on page 3816
Select the objects and properties associated with the servicefederation component that you want to change.

Configuring the Service Federation Management properties of an execution group

Using the **mqsichangeproperties** command, you can configure the Service Federation Management (SFM) properties of an execution group in a broker that has been enabled for SFM. These properties determine how proxies are created.

Before you begin

Before you start:

- Create a broker
- Start the broker
- Create an execution group
- Enable SFM in the broker

About this task

When you enable a broker for SFM, the execution groups in the broker are also enabled for SFM; they act as connectivity providers and create and host the proxy instances for use by service consumers in other domains.

Procedure

Run the **mqsichangeproperties** command to configure the properties that relate to the SFM capability of an execution group.
For example, to set a host name, insecure port, and secure port to be used in the endpoint address of SFM proxies created in an execution group:

```
mqsichangeproperties TEST -e exgroup1 -o ServiceFederationManager
-n proxyURLHostName,port,securePort -v mbhost.ibm.com,8811,8844
```

If an SFM proxy is configured to use SSL, its input proxy URL transport is HTTPS; it will need to use the keystore at either the execution group level or broker level to obtain its private key. If an SFM proxy has a target service that requires SSL, its target service URL transport is HTTPS; it will need to use the truststore at either the execution group level or the broker level to obtain the public key of the service. In such cases, you must configure either the execution group or the broker to reference keystores and truststores that have been loaded with these certificates. If both are set, the execution group level setting is used.
To review the properties that relate to the SFM capability of an execution group, run the **mqsireportproperties** command with the **-a** all properties option:

```
mqsireportproperties broker_name -e execution_group_name -o
ServiceFederationManager -a
```

To review details of SFM proxies that have been created in an execution group, run the **mqsireportproperties** command with the **-r** all recursive child properties option:

```
mqsireportproperties broker_name -e execution_group_name -o
ServiceFederationManager -r
```

Related concepts:

"Working with Service Federation Management (SFM)" on page 911
You can use a broker as a connectivity server that can be administered by a Service Federation Management (SFM) console provided with WebSphere Service Registry and Repository (WSRR) V7.0. The SFM console can then configure SFM proxies in the broker.

Related tasks:

"Enabling a broker for Service Federation Management" on page 912
Use the **mqsichangeproperties** command to enable a broker for Service Federation Management (SFM).

Chapter 8, "Administering brokers and broker resources," on page 899
Administering brokers and associated broker resources includes the tasks that you perform frequently to activate and manage those resources. Choose the method you prefer to administer your brokers and associated resources.

"Viewing and setting keystore and truststore runtime properties at broker level" on page 780
Configure the message broker to refer to a keystore, a truststore, or both, before deploying any message flows that require policy set or bindings for signature, encryption, or X.509 Authentication.

"Viewing and setting keystore and truststore runtime properties at execution group level" on page 783
Configure an execution group to refer to a keystore, a truststore, or both, before deploying any message flows that require policy set or bindings for signature, encryption, or X.509 Authentication.

Related reference:

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"ServiceFederationManager object property values" on page 3818
Select the properties associated with the ServiceFederationManager component that you want to change.

Grouping brokers by using broker sets

You can create a manual or an automatic broker set to visually group your brokers in the WebSphere Message Broker Explorer.

About this task

If you have a large number of brokers displayed in the WebSphere Message Broker Explorer, you might find it helpful to group the brokers by using broker sets. You can create manual broker sets, automatic broker sets, or a combination of both types. When you create a broker set for the first time, two sets are created; a set called All and a set with a name that you provided when you created it. The All broker set contains all your local brokers, and any remote brokers definitions in your workspace. If you delete all of the broker sets that you have created, the All broker set is automatically removed. You cannot rename, modify, or delete the All broker set manually.

A manual broker set is empty until you add brokers to the broker set from the All broker set or another broker set. You can add or remove brokers from the manual broker set at any time.

An automatic broker set uses broker tags to dynamically add and remove brokers from a set based on values you provide or the current state of the brokers.

See the following topics for more information about how to create and use broker sets in the WebSphere Message Broker Explorer:
- "Creating a manual broker set in the WebSphere Message Broker Explorer"
- "Adding and modifying broker tags in the WebSphere Message Broker Explorer" on page 917
- "Creating an automatic broker set in the WebSphere Message Broker Explorer" on page 918
- "Modifying broker sets in the WebSphere Message Broker Explorer" on page 920

Creating a manual broker set in the WebSphere Message Broker Explorer

You can create a manual broker set, and add brokers to visually group your brokers in the WebSphere Message Broker Explorer.

Before you begin

Before you start:

- Create a broker

About this task

A manual broker set is empty until you add brokers to the broker set from the All broker set, or from another broker set. You can add or remove brokers from the manual broker set at any time.

To create a manual broker set in the WebSphere Message Broker Explorer:

Procedure

1. In the Navigator view right-click the Brokers folder, and click **Broker Sets** > **New Broker Set**. The Create New Broker Set wizard is displayed.
2. Enter a name for the broker set.
3. Ensure **Manual** is selected as the set type.
4. Click **Finish**.

Results

The broker set is added to the Brokers folder. If this set was the first broker set that you have created, the All broker set is also added to the Brokers folder. The All broker set contains all your local brokers, and all remote brokers definitions in your workspace.

What to do next

You can now move brokers from your All broker set, or other broker sets, into the manual broker set that you have created.

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

Related tasks:

"Managing brokers" on page 900
Work with your existing brokers to manage their connections and their active status by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer. You can use the Administration API (also known as the CMP API)

to complete some of these actions.

"Configuring brokers in the WebSphere Message Broker Explorer" on page 635
Configure your local and remote brokers by using the WebSphere Message Broker Explorer.

"Grouping brokers by using broker sets" on page 915
You can create a manual or an automatic broker set to visually group your brokers in the WebSphere Message Broker Explorer.

"Creating an automatic broker set in the WebSphere Message Broker Explorer" on page 918
Create an automatic broker set in the WebSphere Message Broker Explorer to group brokers dynamically based on values you provide, or their current state.

Adding and modifying broker tags in the WebSphere Message Broker Explorer

Add or modify the broker tags on your brokers to change the brokers that are grouped in an automatic set in the WebSphere Message Broker Explorer.

Before you begin

Before you start:

- Create a broker

About this task

An automatic broker set uses broker tags to dynamically add and remove brokers from a set based on values you provide or the current state of the brokers. You must add the appropriate tags to your brokers before they can be dynamically added to an automatic broker set. You can select from the following included broker tags, or you can configure your own custom broker tags:

- brokerEnvironment:Development
- brokerEnvironment:Test
- brokerEnvironment:QA
- brokerEnvironment:Production

You can also remove tags that you have previously added to a broker to stop the broker being added to an automatic set.

To add or modify broker tags in the WebSphere Message Broker Explorer:

Procedure

1. In the Navigator view, expand the Brokers folder.
2. Right-click the broker for which you want to add or modify broker tags, and click **Modify Broker Tags**.
3. Select the broker tags to add to the broker:
   - Select broker tags from the available list.
   - Add a custom broker tag:
     a. Enter a name for the tag in the **Tags** field.
     b. Click **Add**.

        You can also select a tag and click **Remove** to remove it from the list of available broker tags.
   - Clear tags that you want to remove from the broker.
4. Click **Finish**.

Results

The selected broker tags are added to the broker. If the broker tags are matched in an automatic broker set, the broker appears in the broker set. If the broker tags no longer match in an automatic broker set, the broker no longer appears in the broker set.

What to do next

You can create or modify an automatic broker set to use one or more of the broker tags that you have created.

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

Related tasks:

"Grouping brokers by using broker sets" on page 915
You can create a manual or an automatic broker set to visually group your brokers in the WebSphere Message Broker Explorer.

"Creating a manual broker set in the WebSphere Message Broker Explorer" on page 916
You can create a manual broker set, and add brokers to visually group your brokers in the WebSphere Message Broker Explorer.

"Creating an automatic broker set in the WebSphere Message Broker Explorer"
Create an automatic broker set in the WebSphere Message Broker Explorer to group brokers dynamically based on values you provide, or their current state.

"Modifying broker sets in the WebSphere Message Broker Explorer" on page 920
Modify an automatic broker set in the WebSphere Message Broker Explorer to change or add broker tags used to filter brokers in the broker set.

Creating an automatic broker set in the WebSphere Message Broker Explorer

Create an automatic broker set in the WebSphere Message Broker Explorer to group brokers dynamically based on values you provide, or their current state.

Before you begin

Before you start:

- Create a broker
- Add broker tags to your brokers

About this task

An automatic broker set uses broker tags to dynamically add and remove brokers from a set based on values you provide or the current state of the brokers. You must add the appropriate tags to your brokers before they can be dynamically added to an automatic broker set. You can select a single tag, or multiple tags, for an automatic broker set.

You can select from the following included broker tags, or any custom tags you have added:

- brokerEnvironment:Development

- brokerEnvironment:Test
- brokerEnvironment:QA
- brokerEnvironment:Production
- brokerStatus:Started
- brokerStatus:Stopped
- brokerStatus:Connected
- brokerStatus:Disconnected

To create an automatic broker set:

Procedure

1. In the Navigator view right-click the Brokers folder, and click **Broker Sets** > **New Broker Set**. The Create New Broker Set wizard is displayed.
2. Enter a name for the broker set.
3. Ensure **Automatic** is selected as the set type.
4. Click **Next**.
5. Select the conditions under which you want the broker to be a part of the broker set from the list of available filters. Click the **Add** button to move them into the selected filters pane.
6. Decide if you want the broker to match all the selected conditions, or one or more of them:
   - If you want the broker to match all the selected conditions ensure **matches ALL of the selected filters** is selected.
   - If you want the broker to match one or more of the selected conditions ensure **matches ANY of the selected filters** is selected.
7. Click **Finish**.

Results

The broker set is added to the Brokers folder, and all brokers that match the selected filters are added to the broker set. If this set is the first broker set that you have created, the All broker set is also added to the Brokers folder. The All broker set contains all your local brokers, and all remote brokers definitions in your workspace.

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

Related tasks:

"Grouping brokers by using broker sets" on page 915
You can create a manual or an automatic broker set to visually group your brokers in the WebSphere Message Broker Explorer.

"Creating a manual broker set in the WebSphere Message Broker Explorer" on page 916
You can create a manual broker set, and add brokers to visually group your brokers in the WebSphere Message Broker Explorer.

"Adding and modifying broker tags in the WebSphere Message Broker Explorer" on page 917
Add or modify the broker tags on your brokers to change the brokers that are grouped in an automatic set in the WebSphere Message Broker Explorer.

"Modifying broker sets in the WebSphere Message Broker Explorer"
Modify an automatic broker set in the WebSphere Message Broker Explorer to change or add broker tags used to filter brokers in the broker set.

Modifying broker sets in the WebSphere Message Broker Explorer

Modify an automatic broker set in the WebSphere Message Broker Explorer to change or add broker tags used to filter brokers in the broker set.

Before you begin

Before you start:

- Create a broker
- Create an automatic broker set

About this task

An automatic broker set uses broker tags to dynamically add and remove brokers from a set based on values you provide or the current state of the brokers. You must add the appropriate tags to your brokers before they can be dynamically added to an automatic broker set. You can select from the following included broker tags or any custom tags you have added:

- brokerEnvironment:Development
- brokerEnvironment:Test
- brokerEnvironment:QA
- brokerEnvironment:Production
- brokerStatus:Started
- brokerStatus:Stopped
- brokerStatus:Connected
- brokerStatus:Disconnected

To modify an automatic broker set in the WebSphere Message Broker Explorer

Procedure

1. In the Navigator view expand the Brokers folder.
2. Right-click the broker set that you want to modify, and click **Modify Broker Set**.
3. Select the conditions under which you want the broker to be a part of the broker set from the list of available filters:
   - Click the **Add** button to move them into the selected filters pane.
   - Click the **Remove** button to remove them from the selected filters pane.
4. Decide if you want the broker to match all the selected conditions or one or more of them:
   - If you want the broker to match all the selected conditions ensure **matches ALL of the selected filters** is selected.
   - If you want the broker to match one or more of the selected conditions ensure **matches ANY of the selected filters** is selected.
5. Click **Finish**.

Results

The broker set is modified, and all brokers that match the selected filters are added to the broker set.

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

Related tasks:

"Grouping brokers by using broker sets" on page 915
You can create a manual or an automatic broker set to visually group your brokers in the WebSphere Message Broker Explorer.

"Creating a manual broker set in the WebSphere Message Broker Explorer" on page 916
You can create a manual broker set, and add brokers to visually group your brokers in the WebSphere Message Broker Explorer.

"Creating an automatic broker set in the WebSphere Message Broker Explorer" on page 918
Create an automatic broker set in the WebSphere Message Broker Explorer to group brokers dynamically based on values you provide, or their current state.

"Adding and modifying broker tags in the WebSphere Message Broker Explorer" on page 917
Add or modify the broker tags on your brokers to change the brokers that are grouped in an automatic set in the WebSphere Message Broker Explorer.

Starting and stopping a broker

Run the appropriate command to start or stop a broker.

Before you begin

Before you start:

- Ensure that your user ID has the correct authorizations to perform the task. Refer to "Security requirements for administrative tasks" on page 3644.
- On Linux, UNIX, and Windows systems, you must set up your command-line environment before performing this task, by running the product profile or console; refer to "Setting up a command environment" on page 213.

About this task

To start and stop a broker, you can use the **mqsistart** and **mqsistop** commands from the command line. Alternatively, on Windows and Linux, use the WebSphere Message Broker Toolkit or the WebSphere Message Broker Explorer to start and stop brokers.

To start or stop a broker using the WebSphere Message Broker Toolkit or the WebSphere Message Broker Explorer, follow the appropriate link:

- "Starting a local broker using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 925
- "Stopping a local broker using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 926

To start or stop a broker using commands from the command line, follow the link for the appropriate platform:

Procedure

- "Starting and stopping a broker on Linux and UNIX systems"
- "Starting and stopping a broker on Windows" on page 923
- "Starting and stopping a broker on z/OS" on page 924

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related tasks:

"Setting up a command environment" on page 213
After you have installed the product on one of the distributed systems, you must initialize the environment before you can use a runtime component or command.

Related reference:

"Security requirements for administrative tasks" on page 3644
You can configure access permissions to govern which users and groups can manipulate objects in the broker network. Security requirements for administrative tasks depend on the platform that you use.

"**mqsistart** command" on page 3965
Use the **mqsistart** command to start the specified broker if all initial verification tests complete successfully.

"**mqsistop** command" on page 3972
Use the **mqsistop** command to stop the specified component.

Starting and stopping a broker on Linux and UNIX systems

Run the appropriate command to start or stop a broker.

Procedure

1. Run '*./install_dir*/bin/mqsiprofile' to source the mqsiprofile script and set up the environment for a single targeted runtime environment. You must complete this setup before you can run a WebSphere Message Broker command.
2. To start a broker, enter the following command on the command line:

   mqsistart MB7BROKER

   Substitute your own broker name for MB7BROKER. The broker and its associated queue manager are started.

   Check the system log to ensure that the broker has initialized successfully. The log contains messages about verification procedures; if all tests are successful, only an initial start message is recorded. If any verification test is unsuccessful, the log also includes messages that provide details of the tests that have failed. If errors have been reported, review the messages and take the suggested actions to resolve these problems.
3. To stop a broker, enter the following command on the command line:

   mqsistop MB7BROKER

   Substitute your own broker name for MB7BROKER.

   You can also request that the broker queue manager is stopped by this command. Refer to "Stopping a WebSphere MQ queue manager when you stop a broker" on page 929.

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related tasks:

"Creating a broker" on page 611
You can create brokers on every platform that is supported by WebSphere Message Broker. The broker runs as a 64-bit application on all platforms except Linux on x86 and Windows on x86.

"Linux and UNIX systems: Configuring the syslog daemon" on page 3529
On Linux and UNIX systems, all WebSphere Message Broker messages (other than messages that are generated by the command-line utilities) are sent to the syslog subsystem.

"Stopping a WebSphere MQ queue manager when you stop a broker" on page 929
If you are preparing to stop a broker, you can stop the broker's WebSphere MQ queue manager at the same time.

Related reference:

"**mqsistart** command" on page 3965
Use the **mqsistart** command to start the specified broker if all initial verification tests complete successfully.

"**mqsistop** command" on page 3972
Use the **mqsistop** command to stop the specified component.

Starting and stopping a broker on Windows

Run the appropriate command to start or stop a broker.

Procedure

1. Open the WebSphere Message Broker command console. When you open the console, it sets up the environment that you need to run the WebSphere Message Broker commands.

   If you prefer, you can run the *install_dir*/bin/mqsiprofile command to set up the environment.

2. To start a broker, enter the following command on the command line:

   ```
   mqsistart MB7BROKER
   ```

   Substitute your own broker name for MB7BROKER.

   You can also request that the broker queue manager is started as a Windows service. Refer to "Starting a WebSphere MQ queue manager as a Windows service" on page 929.

   The broker and its associated queue manager are started. The command initiates the startup of the broker's Windows service.

   Check the Application Log in the Event Viewer to ensure that the broker has initialized successfully. The log contains messages about verification procedures; if all tests are successful, only an initial start message is recorded. If one or more verification tests are unsuccessful, the log also includes messages that provide details of the tests that have failed. If errors have been reported, review the messages and take the suggested actions to resolve these problems.

3. To stop a broker, enter the following command on the command line:

   ```
   mqsistop MB7BROKER
   ```

   Substitute your own broker name for MB7BROKER.

   You can also request that the broker queue manager is stopped by this command. Refer to "Stopping a WebSphere MQ queue manager when you stop a broker" on page 929.

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to

route, transform, and enrich in flight messages.

Related tasks:

"Creating a broker on Windows" on page 616
On Windows, you can create brokers on the command line.

"Starting a WebSphere MQ queue manager as a Windows service" on page 929
On Windows, you can start a WebSphere MQ queue manager as a Windows service to ensure the queue manager starts when you start your other components.

"Stopping a WebSphere MQ queue manager when you stop a broker" on page 929
If you are preparing to stop a broker, you can stop the broker's WebSphere MQ queue manager at the same time.

"Windows: Viewing the local error log" on page 3527
The Windows Event Viewer is where WebSphere Message Broker writes records to the local system. Use Windows system facilities to view this log.

Related reference:

"**mqsistart** command" on page 3965
Use the **mqsistart** command to start the specified broker if all initial verification tests complete successfully.

"**mqsistop** command" on page 3972
Use the **mqsistop** command to stop the specified component.

Starting and stopping a broker on z/OS

Run the appropriate command from SDSF to start or stop a broker.

Procedure

1. Start the component by using the command **/S <broker name>**. This command produces the following output, where MQP1BRK is the name of the broker:

   ```
   +BIP9141I MQP1BRK 0 The component was started
   ```

   Substitute your own broker name for MQP1BRK.

   The verification step runs, followed by starting the control process and any DataFlowEngine (execution group) address spaces.

   If the verification step fails, the errors are reported to the STDOUT stream in the JOBLOG. The control process and DataFlowEngine address spaces are not started. Review the messages to see what errors have been reported, and take the suggested actions to resolve these problems.

2. Alternatively, start the control process only by using the command:

   ```
   /S broker_name,STRTP=MAN
   ```

   If the verification step fails for any reason, the errors are reported to the STDOUT stream in the JOBLOG; the control process is not started. Review the messages to see what errors have been reported, and take the suggested actions to resolve these problems.

   No DataFlowEngine address spaces are started automatically if you specify STRTP=MAN. If the verification step is successful and the control process starts successfully, fully start the broker by issuing the console command:

   ```
   /F <broker name>, SC
   ```

3. To stop a broker, run the following command:

   ```
   /P <broker name>
   ```

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related tasks:

"Creating a broker on z/OS" on page 620
Create the broker component and the other resources on which it depends.

Related reference:

"START and STOP commands on z/OS" on page 3981

"Issuing commands to the z/OS console" on page 3980
You operate the broker using the z/OS START, STOP, and MODIFY commands.

"**mqsistart** command" on page 3965
Use the **mqsistart** command to start the specified broker if all initial verification tests complete successfully.

"**mqsistop** command" on page 3972
Use the **mqsistop** command to stop the specified component.

Starting a local broker using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer

You can start your local brokers by using the WebSphere Message Broker Toolkit or the WebSphere Message Broker Explorer.

About this task

On Windows and Linux use the WebSphere Message Broker Toolkit or the WebSphere Message Broker Explorer to start and stop brokers. Alternatively, you can use the **mqsistart** and **mqsistop** commands from the command line to start or stop a broker.

Starting a local broker using the WebSphere Message Broker Toolkit: About this task

To start a local broker by using the WebSphere Message Broker Toolkit:

Procedure

1. Open the WebSphere Message Broker Toolkit, and switch to the Broker Application Development perspective.
2. In the Brokers view, right-click the broker, and click **Start**.

Results

You have started the selected broker.

Starting a local broker using the WebSphere Message Broker Explorer: About this task

To start a local broker by using the WebSphere Message Broker Explorer:

Procedure

1. Open the WebSphere Message Broker Explorer.
2. In the Navigator view, expand the Brokers folder.
3. Right-click the broker you want to start, and click **Start**.

Results

You have started the selected broker.

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

Related tasks:

"Creating a broker using the WebSphere Message Broker Explorer" on page 618
On Linux on x86 or Windows, you can create brokers by using the WebSphere Message Broker Explorer.

"Connecting to a local broker using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 901
To administer a broker by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer, you must be connected to the broker.

"Starting and stopping a broker" on page 921
Run the appropriate command to start or stop a broker.

"Creating an execution group using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 937
Use the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer to create execution groups on your broker.

"Stopping a local broker using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer"
You can stop your local brokers by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer.

Chapter 8, "Administering brokers and broker resources," on page 899
Administering brokers and associated broker resources includes the tasks that you perform frequently to activate and manage those resources. Choose the method you prefer to administer your brokers and associated resources.

Related reference:

"**mqsistart** command" on page 3965
Use the **mqsistart** command to start the specified broker if all initial verification tests complete successfully.

"**mqsistop** command" on page 3972
Use the **mqsistop** command to stop the specified component.

Stopping a local broker using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer

You can stop your local brokers by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer.

About this task

On Windows and Linux on x86, use the WebSphere Message Broker Toolkit or the WebSphere Message Broker Explorer to start and stop brokers. Alternatively, you can use the **mqsistop** command from the command line to stop a broker.

Stopping a local broker using the WebSphere Message Broker Toolkit: About this task

To stop a local broker by using the WebSphere Message Broker Toolkit:

Procedure

1. Open the WebSphere Message Broker Toolkit, and switch to the Broker Application Development perspective.
2. In the Brokers view, right-click the broker, and click **Stop**.

Results

You have stopped the selected broker.

Stopping a local broker using the WebSphere Message Broker Explorer: About this task

To stop a local broker by using the WebSphere Message Broker Explorer:

Procedure

1. Open the WebSphere Message Broker Explorer.
2. In the Navigator view, expand the Brokers folder.
3. Right-click the broker you want to stop, and click **Stop** > **Broker**. You can also click **Stop** > **Broker Immediately** if you have already tried, and failed, to stop the broker in a controlled fashion.

Results

You have stopped the selected broker.

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

Related tasks:

"Creating a broker using the WebSphere Message Broker Explorer" on page 618
On Linux on x86 or Windows, you can create brokers by using the WebSphere Message Broker Explorer.

"Connecting to a local broker using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 901
To administer a broker by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer, you must be connected to the broker.

"Starting and stopping a broker" on page 921
Run the appropriate command to start or stop a broker.

"Creating an execution group using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 937
Use the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer to create execution groups on your broker.

"Starting a local broker using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 925
You can start your local brokers by using the WebSphere Message Broker Toolkit or the WebSphere Message Broker Explorer.

Chapter 8, "Administering brokers and broker resources," on page 899
Administering brokers and associated broker resources includes the tasks that you perform frequently to activate and manage those resources. Choose the method you prefer to administer your brokers and associated resources.

Viewing broker properties

You can view broker properties by using the **mqsireportbroker** command. You can also use the WebSphere Message Broker Explorer to view broker properties.

Before you begin

Before you start:

You must have completed the following tasks:

- Ensure that your user ID has the correct authorizations to perform the task; see "Security requirements for administrative tasks" on page 3644.
- Create a broker.
- On Windows, Linux, and UNIX systems, you must set up your command-line environment before performing this task by running the product profile or console; see "Setting up a command environment" on page 213.

About this task

Use the **mqsireportbroker** command or domain to view all the properties associated with a broker. The command shows both parameters entered on the command line and viewable in the workbench.

Procedure

1. Run the **mqsireportbroker** command. For example, to view the properties of the broker SOAPBR, run the following command:

   ```
   mqsireportbroker SOAPBR
   ```

2. View responses of the **mqsireportbroker** command to check on current settings. Examples are given in "**mqsireportbroker** command" on page 3919.
3. If you want to make any changes, run the **mqsichangebroker** command, specifying the required parameters.

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related tasks:

"Creating a broker" on page 611
You can create brokers on every platform that is supported by WebSphere Message Broker. The broker runs as a 64-bit application on all platforms except Linux on x86 and Windows on x86.

"Starting and stopping a broker" on page 921
Run the appropriate command to start or stop a broker.

"Configuring brokers in the WebSphere Message Broker Explorer" on page 635
Configure your local and remote brokers by using the WebSphere Message Broker Explorer.

Related reference:

"Security requirements for administrative tasks" on page 3644
You can configure access permissions to govern which users and groups can manipulate objects in the broker network. Security requirements for administrative tasks depend on the platform that you use.

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

Starting a WebSphere MQ queue manager as a Windows service

On Windows, you can start a WebSphere MQ queue manager as a Windows service to ensure the queue manager starts when you start your other components.

Before you begin

Before you start:

You must complete the following task:

- Stop the queue manager for the Broker component by using the **endmqm** command. If you prefer, you can use the WebSphere MQ Explorer.

About this task

When you start a broker, the "**mqsistart** command" on page 3965 starts the associated queue manager if it is not already running.

When you start a broker on Windows, the broker starts as a service on Windows, but the associated queue manager does not.

You can change the properties of the queue manager service to set the startup type to automatic to enable the queue manager to run as a Windows service.

This change ensures that the operation of the queue manager is independent of the logged-on status of the user that starts the broker.

To start a WebSphere MQ queue manager as a Windows service:

Procedure

1. Click **Start** > **Programs** > **IBM WebSphere MQ** > **WebSphere MQ Explorer**.
2. In the left pane, right-click the queue manager and select **Properties**. The Properties dialog opens. The General properties are displayed.
3. Find the Startup property and set it to *Automatic*.
4. Click **OK**. The Properties dialog closes and the change is applied.
5. Restart the queue manager for the broker by using the **strmqm** command or WebSphere MQ Explorer. The changes to the queue manager's startup type take effect when you restart Windows.
6. Start the broker by using the **mqsistart** command.

Related tasks:

"Starting and stopping a broker" on page 921
Run the appropriate command to start or stop a broker.

Related reference:

"**mqsistart** command" on page 3965
Use the **mqsistart** command to start the specified broker if all initial verification tests complete successfully.

Stopping a WebSphere MQ queue manager when you stop a broker

If you are preparing to stop a broker, you can stop the broker's WebSphere MQ queue manager at the same time.

About this task

You can specify a `-q` parameter on the **mqsistop** command to initiate a controlled shutdown of the queue manager for a broker.

Procedure

To stop a WebSphere MQ queue manager enter the following command on the command line:

```
mqsistop MB7BROKER  -q
```

where:
`MB7BROKER` is the name of the broker.
`-q` stops the WebSphere MQ queue manager associated with the component. The command cannot complete until shutdown of the queue manager has completed.

Related tasks:

"Starting and stopping a broker" on page 921
Run the appropriate command to start or stop a broker.

Related reference:

"**mqsistop** command" on page 3972
Use the **mqsistop** command to stop the specified component.

Deleting a broker

Delete a broker using the command line on the system where the broker component is installed.

Before you begin

Before you start: Check that your user ID has the correct authorizations to perform the task; for details, see "Security requirements for administrative tasks" on page 3644.

About this task

On Windows and Linux systems, you can delete a broker using the WebSphere Message Broker Toolkit or the WebSphere Message Broker Explorer. For instructions on deleting the broker using the WebSphere Message Broker Toolkit or the WebSphere Message Broker Explorer, see "Deleting a broker using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 933.

On Windows, Linux, and UNIX systems, you must set up your command-line environment before deleting a broker, by running the product profile or console; see "Setting up a command environment" on page 213.

Follow the link for the appropriate platform:

Procedure

- "Deleting a broker on Linux and UNIX systems" on page 931
- "Deleting a broker on Windows" on page 932
- "Deleting a broker on z/OS" on page 933

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related tasks:

"Creating a broker" on page 611
You can create brokers on every platform that is supported by WebSphere Message Broker. The broker runs as a 64-bit application on all platforms except Linux on x86 and Windows on x86.

"Modifying a broker" on page 631
Modify a broker by using the command line on the system where the broker component is installed.

Related reference:

"Security requirements for administrative tasks" on page 3644
You can configure access permissions to govern which users and groups can manipulate objects in the broker network. Security requirements for administrative tasks depend on the platform that you use.

"**mqsideletebroker** command" on page 3863
Use the **mqsideletebroker** command to delete a named broker. The command also deletes the queues on the associated queue manager (created when the broker was created). You can also specify that the queue manager is to be deleted.

Deleting a broker on Linux and UNIX systems

Delete the physical broker component.

About this task

To delete a broker on Linux and UNIX systems:

Procedure

1. Stop the broker by using the **mqsistop** command.
2. Enter the following command to delete the broker:

   ```
   mqsideletebroker MB7BROKER
   ```

   where `MB7BROKER` is the broker name.
3. If you have created execution group profiles for this broker, delete these profiles if they are no longer required.
4. If you have created execution group `shared-classes` directories for this broker, delete these directories if they are no longer required.

Results

On completion of this task, you have:

- Removed the broker data from the database.
- Removed the record for the component in the broker registry. It is therefore removed from the list of components that are displayed when you run the **mqsilist** command.
- Removed any execution group profile scripts that are no longer needed.
- Removed any execution group `shared-classes` directories that are no longer needed.

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related tasks:

"Starting and stopping a broker" on page 921
Run the appropriate command to start or stop a broker.

Related reference:

"**mqsideletebroker** command" on page 3863
Use the **mqsideletebroker** command to delete a named broker. The command also deletes the queues on the associated queue manager (created when the broker was created). You can also specify that the queue manager is to be deleted.

"**mqsistop** command" on page 3972
Use the **mqsistop** command to stop the specified component.

"**mqsilist** command" on page 3882
Use the **mqsilist** command to list installed brokers and their associated resources.

Deleting a broker on Windows

Delete the physical broker component.

About this task

Windows To delete a broker:

Procedure

1. Stop the broker by using the **mqsistop** command.
2. Enter the following command to delete the broker:

   ```
   mqsideletebroker MB7BROKER
   ```

   where MB7BROKER is the broker name.
3. If you have created execution group profiles for this broker, delete these profiles if they are no longer required.
4. If you have created execution group `shared-classes` directories for this broker, delete these directories if they are no longer required.

Results

On completion of this task, you have:

- Stopped the Windows service that runs the broker.
- Removed the record for the component in the broker registry. It is therefore removed from the list of components that are displayed when you run the **mqsilist** command.
- Removed any execution group profile scripts that are no longer needed.
- Removed any execution group `shared-classes` directories that are no longer needed.

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related tasks:

"Starting and stopping a broker" on page 921
Run the appropriate command to start or stop a broker.

Related reference:

"**mqsideletebroker** command" on page 3863
Use the **mqsideletebroker** command to delete a named broker. The command also deletes the queues on the associated queue manager (created when the broker was

created). You can also specify that the queue manager is to be deleted.

"**mqsistop** command" on page 3972
Use the **mqsistop** command to stop the specified component.

"**mqsilist** command" on page 3882
Use the **mqsilist** command to list installed brokers and their associated resources.

Deleting a broker on z/OS

Delete the physical broker component.

About this task

To delete a broker:

Procedure

1. Stop the broker, by stopping the started task.
2. Customize and submit the delete job `BIPDLBK` in your component PDSE to delete the broker component and WebSphere MQ. This action does not delete all files from the component directory in the file system.
3. If you have created an execution group `shared-classes` directory for this broker, delete this directory if it is no longer required.

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related tasks:

"Creating a broker on z/OS" on page 620
Create the broker component and the other resources on which it depends.

"Modifying a broker on z/OS" on page 634
Use the **mqsichangebroker** command on z/OS to modify your broker.

Related reference:

"**mqsideletebroker** command" on page 3863
Use the **mqsideletebroker** command to delete a named broker. The command also deletes the queues on the associated queue manager (created when the broker was created). You can also specify that the queue manager is to be deleted.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

Deleting a broker using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer

On Windows and Linux on x86 you can delete brokers by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer.

Before you begin

Before you start: Check that your user ID has the correct authorizations to perform the task; for details, see "Security requirements for administrative tasks" on page 3644.

About this task

You can delete a local broker using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer. You cannot delete a remote broker using the

WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer, but you can remove the connection to the remote broker from your workspace. To remove the connection to a remote broker in the WebSphere Message Broker Toolkit, right-click the broker in the Brokers view, and click **Remove Connection**. To remove the connection to a remote broker in the WebSphere Message Broker Explorer, right-click the broker in the Navigator view, and click **Remove**.

Deleting a broker using the WebSphere Message Broker Toolkit:
About this task

To delete a broker using the WebSphere Message Broker Toolkit:

Procedure

1. Open the WebSphere Message Broker Toolkit, and switch to the Broker Application Development perspective.
2. In the Brokers view, right-click the broker, and click **Delete**. If the broker is a remote broker, the connection to the broker is removed from your workspace.

   If the broker is a local broker, the Delete Local Broker wizard is displayed:

   a. Optional: If you want to delete the queue manager associated with the broker, ensure **Delete the broker's queue manager** is selected.
   b. Click **Finish** to delete the broker. The wizard stops the broker if it is running.

Results

You have deleted the broker.

Deleting a broker using the WebSphere Message Broker Explorer:
About this task

To delete a broker using the WebSphere Message Broker Explorer:

Procedure

1. Open the WebSphere Message Broker Explorer.
2. In the Navigator view, expand the broker's queue manager, or the Brokers folder.
3. Right-click the broker, and click **Delete** > **Broker**. The Delete Broker wizard is displayed.
4. If you want to delete the queue manager associated with the broker, ensure **Delete the broker's queue manager** is selected.
5. Click **Next**. The wizard stops the broker if it is running.
6. Click **Finish** to close the wizard.

Results

You have deleted the broker. If you choose to delete the queue manager, the queue manager continues to be displayed in the Navigator view until the views are refreshed. The time this takes depends on your queue manager refresh settings, but the default refresh value is 15 seconds.

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

Related tasks:

"Deleting a broker" on page 930
Delete a broker using the command line on the system where the broker component is installed.

"Creating a broker" on page 611
You can create brokers on every platform that is supported by WebSphere Message Broker. The broker runs as a 64-bit application on all platforms except Linux on x86 and Windows on x86.

"Modifying a broker" on page 631
Modify a broker by using the command line on the system where the broker component is installed.

Related reference:

"Security requirements for administrative tasks" on page 3644
You can configure access permissions to govern which users and groups can manipulate objects in the broker network. Security requirements for administrative tasks depend on the platform that you use.

"**mqsideletebroker** command" on page 3863
Use the **mqsideletebroker** command to delete a named broker. The command also deletes the queues on the associated queue manager (created when the broker was created). You can also specify that the queue manager is to be deleted.

Managing execution groups

Work with your existing execution groups to manage the message flows that you have deployed to them by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer. You can use the command line, and the Administration API (also known as the CMP API) to complete some of these actions.

About this task

- "Creating an execution group" on page 936
- "Renaming an execution group using the WebSphere Message Broker Explorer" on page 940
- "Starting an execution group using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 941
- "Stopping an execution group using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 944
- "Deleting an execution group" on page 946

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related tasks:

Chapter 6, "Configuring brokers for development environments," on page 563
Set up application development environments on Linux on x86 or Windows to create, test, and deploy message flows and associated resources.

Chapter 7, "Configuring brokers for test and production environments," on page 579
Create one or more brokers on one or more computers, and configure them on your test and production systems to process messages that contain your business data.

"Managing brokers" on page 900
Work with your existing brokers to manage their connections and their active status by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer. You can use the Administration API (also known as the CMP API) to complete some of these actions.

"Managing message flows" on page 950
Work with your existing messages that you have deployed to execution groups by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer. You can use the Administration API (also known as the CMP API) to complete some of these actions.

Related information:

Administration API for WebSphere Message Broker (CMP API)

Creating an execution group

Create execution groups by using WebSphere Message Broker Toolkit, the WebSphere Message Broker Explorer, or the command line.

Before you begin

Before you start:

Complete the following tasks:

- Create a broker
- Connect to the broker:
  - "Connecting to a local broker using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 901
  - "Connecting to a remote broker" on page 902
  - "Connecting to a remote broker on z/OS in the WebSphere Message Broker Explorer" on page 904

About this task

The mode that your broker is working in can affect the number of execution groups that you can use; see "Restrictions that apply in each operation mode" on page 3657.

Use one of the following methods to complete this task:

- The WebSphere Message Broker Toolkit or the WebSphere Message Broker Explorer. See "Creating an execution group using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 937 for more information.
- The **mqsicreateexecutiongroup** command. Details are provided in "Creating an execution group using the **mqsicreateexecutiongroup** command" on page 939.

You can also use the CMP API to create execution groups on all platforms; see "Developing applications that use the Administration API" on page 956.

For information about why you might want to create multiple execution groups, see "Execution groups" on page 53.

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to

route, transform, and enrich in flight messages.

Related tasks:

Chapter 6, "Configuring brokers for development environments," on page 563
Set up application development environments on Linux on x86 or Windows to create, test, and deploy message flows and associated resources.

Chapter 7, "Configuring brokers for test and production environments," on page 579
Create one or more brokers on one or more computers, and configure them on your test and production systems to process messages that contain your business data.

"Managing brokers" on page 900
Work with your existing brokers to manage their connections and their active status by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer. You can use the Administration API (also known as the CMP API) to complete some of these actions.

"Managing message flows" on page 950
Work with your existing messages that you have deployed to execution groups by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer. You can use the Administration API (also known as the CMP API) to complete some of these actions.

Related reference:

"**mqsicreateexecutiongroup** command" on page 3854
Use the **mqsicreateexecutiongroup** command to add a new execution group to a broker.

Related information:

Administration API for WebSphere Message Broker (CMP API)

Creating an execution group using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer

Use the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer to create execution groups on your broker.

Before you begin

Before you start: Check that you have completed the prerequisite tasks, and see what other options you can use to create execution groups, in "Creating an execution group" on page 936.

About this task

You must create an execution group before you can deploy message flows and related resources to a broker.

Creating an execution group using the WebSphere Message Broker Toolkit: Before you begin

Before you start:

- Create a broker
- Optional: "Connecting to a remote broker" on page 902

About this task

To add an execution group to a broker:

Procedure

1. In the Brokers view, right-click the broker to which you want to add an execution group, and click **New Execution Group**.
2. In the New Execution Group dialog, enter the **Execution Group name**.
3. Click **OK** to add the execution group to the broker.

Results

The execution group is added to the appropriate broker, and can be viewed as a child of the selected broker.

Creating an execution group using the WebSphere Message Broker Explorer: Before you begin

Before you start:

- Create a broker
- Connect to a local broker or "Connecting to a remote broker" on page 902

About this task

To add an execution group to a broker:

Procedure

1. In the Navigator view, expand the Brokers folder.
2. Right-click the broker to which you want to add an execution group, and click **New** > **Execution Group**.
3. In the New Execution Group dialog, enter the **Execution Group name**.
4. Click **OK** to add the execution group to the broker.

Results

The execution group is added to the appropriate broker, and can be viewed as a child of the selected broker.

Note: If you do not immediately see the execution group that you added, you can refresh the broker by right-clicking on it and selecting **Refresh**.

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

Related tasks:

"Creating a broker using the WebSphere Message Broker Explorer" on page 618
On Linux on x86 or Windows, you can create brokers by using the WebSphere Message Broker Explorer.

"Renaming an execution group using the WebSphere Message Broker Explorer" on page 940
You can rename an execution group by using the WebSphere Message Broker Explorer.

"Importing a broker archive file to the WebSphere Message Broker Explorer" on page 3242
Before you can deploy broker resources to your brokers by using the WebSphere Message Broker Explorer, you must first import a broker archive file into a Broker

Archive Folder.

"Deploying a broker archive file" on page 3235
After you have created and populated a broker archive (BAR) file, deploy the file to an execution group on a broker, so that the file contents can be used in the broker.

"Connecting to a local broker using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 901
To administer a broker by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer, you must be connected to the broker.

"Connecting to a remote broker" on page 902
To administer your brokers by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer, you must connect to the broker.

"Deleting an execution group by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 947
You can delete an execution group by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer.

Creating an execution group using the `mqsicreateexecutiongroup` command

Use the **mqsicreateexecutiongroup** command to create execution groups on your broker.

Before you begin

Before you start: Check that you have completed the prerequisite tasks, and see what other options you can use to create execution groups, in "Creating an execution group" on page 936.

You must create an execution group before you can deploy message flows and related resources to a broker.

Procedure

1. If you are creating an execution group on Linux, UNIX, and Windows systems:
   a. Open a command prompt that has the environment configured for your current installation.
   b. Enter the **mqsicreateexecutiongroup** command, specifying the parameters for the execution group that you want to create.
      - If the broker is local, specify the broker name. For example:

        ```
        mqsicreateexecutiongroup MB7BROKER -e EGroup_2
        ```

      - If the broker is remote, you can specify a configuration file. For example:

        ```
        mqsicreateexecutiongroup -n MB7BROKER.broker -e EGroup_2
        ```

      - If the broker is remote, you can alternatively specify one or more of the connection parameters **-i**, **-p**, **-q**. For example:

        ```
        mqsicreateexecutiongroup -q MB7QMGR -e EGroup_2
        ```

      See the **mqsicreateexecutiongroup** command description for more details about these options.
2. If you are creating an execution group on z/OS:
   a. Configure the `BIPCREG` job to specify the properties for the execution group that you want to create.
   b. Run the `BIPCREG` job.

Results

On completion of this task, the execution group has been created on the specified broker.

What to do next

Next: You can deploy message flows to the execution group by using the WebSphere Message Broker Toolkit or the WebSphere Message Broker Explorer.

Related concepts:

"Execution groups" on page 53
An execution group is a named grouping of message flows that have been assigned to a broker. The broker enforces a degree of isolation between message flows in distinct execution groups by ensuring that they run in separate address spaces, or as unique processes.

Related tasks:

"Creating an execution group using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 937
Use the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer to create execution groups on your broker.

"Renaming an execution group using the WebSphere Message Broker Explorer"
You can rename an execution group by using the WebSphere Message Broker Explorer.

"Deleting an execution group by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 947
You can delete an execution group by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer.

"Deleting an execution group by using the **mqsideleteexecutiongroup** command" on page 949
Use the command line to delete an execution group from the broker.

Related reference:

"**mqsicreateexecutiongroup** command" on page 3854
Use the **mqsicreateexecutiongroup** command to add a new execution group to a broker.

"**mqsideleteexecutiongroup** command" on page 3869
Use the **mqsideleteexecutiongroup** command to remove an execution group from a broker.

Renaming an execution group using the WebSphere Message Broker Explorer

You can rename an execution group by using the WebSphere Message Broker Explorer.

Before you begin

Before you start:

- Create a broker
- Start the broker
- Connect to a local broker, connect to a remote broker, or connect to a remote broker on z/OS
- Create an execution group

About this task

To rename an execution group:

Procedure

1. Open the WebSphere Message Broker Explorer.
2. In the Navigator view, expand the Brokers folder.
3. Expand your broker.
4. Right-click the execution group that you want to rename, and click **Rename**.
5. Enter a name for the execution group, and click **OK**.

Results

You have renamed the selected execution group.

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

Related tasks:

"Creating a broker using the WebSphere Message Broker Explorer" on page 618
On Linux on x86 or Windows, you can create brokers by using the WebSphere Message Broker Explorer.

"Starting a local broker using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 925
You can start your local brokers by using the WebSphere Message Broker Toolkit or the WebSphere Message Broker Explorer.

"Connecting to a local broker using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 901
To administer a broker by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer, you must be connected to the broker.

"Creating an execution group using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 937
Use the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer to create execution groups on your broker.

Chapter 8, "Administering brokers and broker resources," on page 899
Administering brokers and associated broker resources includes the tasks that you perform frequently to activate and manage those resources. Choose the method you prefer to administer your brokers and associated resources.

Starting an execution group using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer

You can start an execution group and all the deployed message flows in an execution group by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer.

About this task

If resources are deployed to your execution group, the started or stopped state of the message flows is retained when you next start the execution group. You can use the menu to start or stop all message flows after the execution group has been started.

If you prefer, you can start execution groups by using the **mqsistartmsgflow** command.

Starting an execution group with the WebSphere Message Broker Toolkit

About this task

To start an execution group by using the WebSphere Message Broker Toolkit:

Procedure

1. Open the WebSphere Message Broker Toolkit, and switch to the Broker Application Development perspective.
2. In the Brokers view, expand your broker.
3. Right-click the execution group, and click **Start**.

Results

A message is sent to the broker to start the selected execution group. Right-click the message flows that you want to start, and click **Start** to start the message flows deployed to the execution group.

Starting an execution group with the WebSphere Message Broker Explorer

About this task

To start an execution group by using the WebSphere Message Broker Explorer:

Procedure

1. Open the WebSphere Message Broker Explorer.
2. In the Navigator view, expand the Brokers folder.
3. Expand your broker.
4. Right-click the execution group, and click **Start** > **Execution Group**.

Results

A message is sent to the broker to start the selected execution group. Click **Start** > **All Flows** to start all the message flows deployed to the execution group.

Starting an execution group with the `mqsistartmsgflow` command

About this task

To start an execution group by using the **mqsistartmsgflow** command:

Procedure

1. Open a command prompt that has the environment configured for your current installation.
2. Enter the **mqsistartmsgflow** command, specifying the parameters for the execution group that you want to start.
   - If the broker is local, specify the broker name. For example:

     ```
     mqsistartmsgflow MB7BROKER -e EGroup_2
     ```

   - If the broker is remote, you can specify a configuration file. For example:

     ```
     mqsistartmsgflow -n MB7BROKER.broker -e EGroup_2
     ```

- If the broker is remote, you can alternatively specify one or more of the connection parameters **-i**, **-p**, **-q**. For example:

  ```
  mqsistartmsgflow -q MB7QMGR -e EGroup_2
  ```

What to do next

If you want to start all the execution groups on a broker, specify the **-g** flag instead of **-e**.

See the **mqsistartmsgflow** command description for more details about these options.

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

Related tasks:

"Creating a broker using the WebSphere Message Broker Explorer" on page 618
On Linux on x86 or Windows, you can create brokers by using the WebSphere Message Broker Explorer.

"Starting a local broker using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 925
You can start your local brokers by using the WebSphere Message Broker Toolkit or the WebSphere Message Broker Explorer.

"Connecting to a local broker using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 901
To administer a broker by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer, you must be connected to the broker.

"Creating an execution group using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 937
Use the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer to create execution groups on your broker.

"Starting a message flow by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 951
Use the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer to start a message flow.

"Stopping a message flow using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 952
Use the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer to stop a message flow.

Chapter 8, "Administering brokers and broker resources," on page 899
Administering brokers and associated broker resources includes the tasks that you perform frequently to activate and manage those resources. Choose the method you prefer to administer your brokers and associated resources.

Related reference:

"**mqsistartmsgflow** command" on page 3969
Use the **mqsistartmsgflow** command to start execution groups and message flows.

Stopping an execution group using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer

You can stop all the message flows in an execution group by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer.

About this task

If resources are deployed to your execution group, the started or stopped state of the message flows is remembered when you stop the execution group. You can use the menu to start or stop all message flows before the execution group has been stopped.

If you prefer, you can stop execution groups by using the **mqsistopmsgflow** command.

The broker processes all inflight messages and associated transactions for each message flow before stopping it. See "Message flow transactions" on page 1281 for information on how outstanding units of work are handled in this situation.

Stopping an execution group with the WebSphere Message Broker Toolkit

About this task

To stop an execution group by using the WebSphere Message Broker Toolkit:

Procedure

1. Open the WebSphere Message Broker Toolkit, and switch to the Broker Application Development perspective.
2. In the Brokers view, expand your broker.
3. Right-click the execution group, and click **Stop**.

Results

A message is sent to the broker to stop the selected execution group.

Stopping an execution group with the WebSphere Message Broker Explorer

About this task

To stop an execution group by using the WebSphere Message Broker Explorer:

Procedure

1. Open the WebSphere Message Broker Explorer.
2. In the Navigator view, expand the Brokers folder.
3. Expand your broker.
4. Right-click the execution group, and click **Stop** > **Execution Group**.

Results

A message is sent to the broker to stop the selected execution group.

Stopping an execution group with the `mqsistopmsgflow` command

About this task

To start an execution group by using the **mqsistopmsgflow** command:

Procedure

1. Open a command prompt that has the environment configured for your current installation.
2. Enter the **mqsistopmsgflow** command, specifying the parameters for the execution group that you want to stop.
   - If the broker is local, specify the broker name. For example:

     ```
     mqsistopmsgflow MB7BROKER -e EGroup_2
     ```
   - If the broker is remote, you can specify a configuration file. For example:

     ```
     mqsistopmsgflow -n MB7BROKER.broker -e EGroup_2
     ```
   - If the broker is remote, you can alternatively specify one or more of the connection parameters **-i**, **-p**, **-q**. For example:

     ```
     mqsistopmsgflow -q MB7QMGR -e EGroup_2
     ```

What to do next

If you want to stop all the execution groups on a broker, specify the **-g** flag instead of **-e**.

See the **mqsistopmsgflow** command description for more details about these options.

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

Related tasks:

"Creating a broker using the WebSphere Message Broker Explorer" on page 618
On Linux on x86 or Windows, you can create brokers by using the WebSphere Message Broker Explorer.

"Starting a local broker using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 925
You can start your local brokers by using the WebSphere Message Broker Toolkit or the WebSphere Message Broker Explorer.

"Connecting to a local broker using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 901
To administer a broker by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer, you must be connected to the broker.

"Creating an execution group using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 937
Use the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer to create execution groups on your broker.

"Starting an execution group using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 941
You can start an execution group and all the deployed message flows in an execution group by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer.

Chapter 8, “Administering brokers and broker resources,” on page 899
Administering brokers and associated broker resources includes the tasks that you perform frequently to activate and manage those resources. Choose the method you prefer to administer your brokers and associated resources.

Related reference:

“**mqsistopmsgflow** command” on page 3975
Use the **mqsistopmsgflow** command to stop execution groups and message flows.

Deleting an execution group

Delete execution groups by using WebSphere Message Broker Toolkit, the WebSphere Message Broker Explorer, or the command line.

Before you begin

Before you start:

Complete the following tasks:

- Create an execution group
- Check that the execution group is running; you cannot delete an execution group that is stopped.

About this task

Use one of the following methods to complete this task:

- The WebSphere Message Broker Toolkit or the WebSphere Message Broker Explorer. See “Deleting an execution group by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer” on page 947 for more information.
- The **mqsideleteexecutiongroup** command. Details are provided in “Deleting an execution group by using the **mqsideleteexecutiongroup** command” on page 949.

You can also use the CMP API to delete execution groups on all platforms; see “Developing applications that use the Administration API” on page 956.

For information about why you might want to create multiple execution groups, see “Execution groups” on page 53.

Related concepts:

“The broker environment” on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related tasks:

Chapter 6, “Configuring brokers for development environments,” on page 563
Set up application development environments on Linux on x86 or Windows to create, test, and deploy message flows and associated resources.

Chapter 7, “Configuring brokers for test and production environments,” on page 579
Create one or more brokers on one or more computers, and configure them on your test and production systems to process messages that contain your business data.

"Managing brokers" on page 900
Work with your existing brokers to manage their connections and their active status by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer. You can use the Administration API (also known as the CMP API) to complete some of these actions.

"Managing message flows" on page 950
Work with your existing messages that you have deployed to execution groups by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer. You can use the Administration API (also known as the CMP API) to complete some of these actions.

Related reference:

"**mqsideleteexecutiongroup** command" on page 3869
Use the **mqsideleteexecutiongroup** command to remove an execution group from a broker.

Related information:

Administration API for WebSphere Message Broker (CMP API)

Deleting an execution group by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer

You can delete an execution group by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer.

Before you begin

Before you start: Check that you have completed the prerequisite tasks, and see what other options you can use to delete execution groups, in "Deleting an execution group" on page 946.

Deleting an execution group by using the WebSphere Message Broker Toolkit: About this task

To delete an execution group by using the WebSphere Message Broker Toolkit

Procedure

1. Open the WebSphere Message Broker Toolkit, and switch to the Broker Application Development perspective.
2. In the Brokers view, expand your broker.
3. Right-click the execution group that you want to delete, and click **Delete** > **Execution Group**.

Results

Your execution group and all resources deployed to it are deleted. If you have created any execution group profiles, or an execution group `shared-classes` directory, for this execution group, delete them manually if they are no longer required.

Deleting an execution group by using the WebSphere Message Broker Explorer: About this task

To delete an execution group by using the WebSphere Message Broker Explorer:

Procedure

1. Open the WebSphere Message Broker Explorer.
2. In the Navigator view, expand the Brokers folder.
3. Expand your broker.
4. Right-click the execution group, and click **Delete** > **Execution Group**.
5. If you have deployed resources, you must confirm that you want to remove all deployed resources before the execution group is deleted. Click **OK** to remove the deployed resources.

Results

Your execution group and all resources deployed to it are deleted. If you have created any execution group profiles, or an execution group `shared-classes` directory, for this execution group, delete them manually if they are no longer required.

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

Related tasks:

"Creating a broker using the WebSphere Message Broker Explorer" on page 618
On Linux on x86 or Windows, you can create brokers by using the WebSphere Message Broker Explorer.

"Starting a local broker using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 925
You can start your local brokers by using the WebSphere Message Broker Toolkit or the WebSphere Message Broker Explorer.

"Connecting to a local broker using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 901
To administer a broker by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer, you must be connected to the broker.

"Creating an execution group using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 937
Use the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer to create execution groups on your broker.

"Deleting an execution group by using the **mqsideleteexecutiongroup** command" on page 949
Use the command line to delete an execution group from the broker.

"Starting a message flow by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 951
Use the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer to start a message flow.

"Stopping a message flow using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 952
Use the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer to stop a message flow.

Chapter 8, "Administering brokers and broker resources," on page 899
Administering brokers and associated broker resources includes the tasks that you perform frequently to activate and manage those resources. Choose the method you prefer to administer your brokers and associated resources.

Deleting an execution group by using the mqsideleteexecutiongroup command

Use the command line to delete an execution group from the broker.

Before you begin

Before you start: Check that you have completed the prerequisite tasks, and see what other options you can use to delete execution groups, in "Deleting an execution group" on page 946.

Procedure

1. If you are deleting an execution group on Linux, UNIX, and Windows systems:
   a. Open a command prompt that has the environment configured for your current installation.
   b. Enter the **mqsideleteexecutiongroup** command, specifying the parameters for the execution group that you want to delete.
      - If the broker is local, specify the broker name; for example:
        ```
        mqsideleteexecutiongroup MB7BROKER -e EGroup_2
        ```
      - If the broker is remote, you can specify a configuration file; for example:
        ```
        mqsideleteexecutiongroup -n MB7BROKER.broker -e EGroup_2
        ```
      - If the broker is remote, you can alternatively specify one or more of the connection parameters **-i**, **-p**, and **-q**; for example:
        ```
        mqsideleteexecutiongroup -q MB7QMGR -e EGroup_2
        ```

      See the **mqsideleteexecutiongroup** command description for more details about these options.
   c. If you have created execution group profiles and they are no longer required, delete them manually.
   d. If you have created an execution group `shared-classes` directory and it is no longer required, delete it manually.
2. If you are deleting an execution group on z/OS:
   a. Configure the `BIPDLEG` job to specify the properties for the execution group that you want to delete.
   b. Run the `BIPDLEG` job.
   c. If you have created an execution group `shared-classes` directory and it is no longer required, delete it manually.

Results

On completion of this task, the execution group has been deleted from the specified broker. In addition:

- All message flows that were running on the execution group are stopped.
- Any execution group profile scripts that are no longer needed have been removed.
- Any execution group `shared-classes` directory that is no longer needed has been removed.

Related concepts:

"Execution groups" on page 53
An execution group is a named grouping of message flows that have been assigned to a broker. The broker enforces a degree of isolation between message flows in distinct execution groups by ensuring that they run in separate address spaces, or as unique processes.

Related tasks:

"Creating an execution group using the **mqsicreateexecutiongroup** command" on page 939
Use the **mqsicreateexecutiongroup** command to create execution groups on your broker.

"Creating an execution group using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 937
Use the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer to create execution groups on your broker.

"Deleting an execution group by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 947
You can delete an execution group by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer.

"Developing applications that use the Administration API" on page 956
Develop Java applications that use the Administration API (also known as the CMP API) to communicate with, deploy to, and manage brokers and their associated resources.

Related reference:

"**mqsicreateexecutiongroup** command" on page 3854
Use the **mqsicreateexecutiongroup** command to add a new execution group to a broker.

"**mqsideleteexecutiongroup** command" on page 3869
Use the **mqsideleteexecutiongroup** command to remove an execution group from a broker.

Managing message flows

Work with your existing messages that you have deployed to execution groups by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer. You can use the Administration API (also known as the CMP API) to complete some of these actions.

About this task

- "Starting a message flow by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 951
- "Stopping a message flow using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 952
- "Deleting a message flow using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 953
- "Setting user-defined properties dynamically at run time using the WebSphere Message Broker Explorer" on page 954

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related tasks:

Chapter 6, "Configuring brokers for development environments," on page 563
Set up application development environments on Linux on x86 or Windows to create, test, and deploy message flows and associated resources.

Chapter 7, “Configuring brokers for test and production environments,” on page 579
Create one or more brokers on one or more computers, and configure them on your test and production systems to process messages that contain your business data.

“Managing brokers” on page 900
Work with your existing brokers to manage their connections and their active status by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer. You can use the Administration API (also known as the CMP API) to complete some of these actions.

“Managing execution groups” on page 935
Work with your existing execution groups to manage the message flows that you have deployed to them by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer. You can use the command line, and the Administration API (also known as the CMP API) to complete some of these actions.

Related information:

Administration API for WebSphere Message Broker (CMP API)

Starting a message flow by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer

Use the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer to start a message flow.

About this task

You can start an individual message flow in the WebSphere Message Broker Toolkit, or multiple message flows in the WebSphere Message Broker Explorer. If you stop the execution group in which the message flow is running, the started or stopped status of the message flow is retained.

Starting a message flow by using the WebSphere Message Broker Toolkit

Procedure

1. In the Brokers view, expand the broker and execution group.
2. Right-click the message flow that you want to start, and click **Start**.

Results

A message is sent to the broker to start the message flow.

Starting a message flow by using the WebSphere Message Broker Explorer

Procedure

1. In the Navigator view, expand the Brokers folder.
2. Expand the broker and execution group, and right-click the message flow or message flows that you want to start, and click **Start**.

Results

A message is sent to the broker to start the message flows.

Note: A message flow can also be started using the command line utility mqsistartmsgflow. For more information see mqsistartmsgflow command.

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

Related tasks:

"Connecting to a local broker using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 901
To administer a broker by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer, you must be connected to the broker.

"Stopping a message flow using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer"
Use the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer to stop a message flow.

Chapter 8, "Administering brokers and broker resources," on page 899
Administering brokers and associated broker resources includes the tasks that you perform frequently to activate and manage those resources. Choose the method you prefer to administer your brokers and associated resources.

Stopping a message flow using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer

Use the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer to stop a message flow.

About this task

You can stop an individual message flow in the WebSphere Message Broker Toolkit, or multiple message flows in the WebSphere Message Broker Explorer. If you stop the execution group in which the message flow is running, the started or stopped status of the message flow is remembered.

You can also use the **mqsistopmsgflow** command to stop a message flow.

Stopping a message flow using the WebSphere Message Broker Toolkit

Procedure

1. In the Brokers view, expand the broker and execution group.
2. Right-click the message flow that you want to stop, and click **Stop**.

Results

A message is sent to the broker to stop the selected message flow.

Stopping a message flow using the WebSphere Message Broker Explorer

Procedure

1. In the Navigator view, expand the Brokers folder.
2. Expand the broker and execution group, and right-click the message flow or message flows that you want to stop, and click **Stop**.

Results

A message is sent to the broker to stop the selected message flows.

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

Related tasks:

"Connecting to a local broker using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 901
To administer a broker by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer, you must be connected to the broker.

"Starting a message flow by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 951
Use the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer to start a message flow.

Chapter 8, "Administering brokers and broker resources," on page 899
Administering brokers and associated broker resources includes the tasks that you perform frequently to activate and manage those resources. Choose the method you prefer to administer your brokers and associated resources.

Related reference:

"**mqsistopmsgflow** command" on page 3975
Use the **mqsistopmsgflow** command to stop execution groups and message flows.

Deleting a message flow using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer

Use the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer to remove message flows from an execution group.

About this task

You can delete an individual message flow in the WebSphere Message Broker Toolkit, or multiple message flows in the WebSphere Message Broker Explorer.

Deleting a message flow using the WebSphere Message Broker Toolkit

Procedure

1. In the Brokers view, expand the broker and execution group.
2. Right-click the message flow that you want to delete, and click **Delete**.

Results

A message is sent to the broker to delete the selected message flow.

Deleting a message flow using the WebSphere Message Broker Explorer

Procedure

1. In the Navigator view, expand the Brokers folder.
2. Expand the broker and execution group, and right-click the message flow or message flows that you want to delete, and click **Delete**.

Results

A message is sent to the broker to delete the selected message flows.

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

Related tasks:

"Creating a broker using the WebSphere Message Broker Explorer" on page 618
On Linux on x86 or Windows, you can create brokers by using the WebSphere Message Broker Explorer.

"Starting a local broker using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 925
You can start your local brokers by using the WebSphere Message Broker Toolkit or the WebSphere Message Broker Explorer.

"Connecting to a local broker using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 901
To administer a broker by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer, you must be connected to the broker.

"Creating an execution group using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 937
Use the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer to create execution groups on your broker.

"Starting a message flow by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 951
Use the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer to start a message flow.

"Stopping a message flow using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 952
Use the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer to stop a message flow.

Chapter 8, "Administering brokers and broker resources," on page 899
Administering brokers and associated broker resources includes the tasks that you perform frequently to activate and manage those resources. Choose the method you prefer to administer your brokers and associated resources.

Setting user-defined properties dynamically at run time using the WebSphere Message Broker Explorer

Use the WebSphere Message Broker Explorer to view and set user-defined properties on a message flow dynamically at run time.

Before you begin

For user-defined properties on a message flow to be discoverable, the message flow must comply with the following conditions:

- The message flow must contain at least one of the following nodes:
  - JavaCompute
  - Compute
  - Database
  - Filter
  - PHPCompute
- The message flow must define the relevant user-defined property and provide an override value.

About this task

Tip: Use meaningful names and values for the properties that you define, so that you can understand their purpose and intent quickly. For example, a user-defined property named `property01`, with an initial value of `valueA` is not as useful as a property named `RouteToAorB` with an initial value of `RouteA`.

Use the following instructions to view and modify user-defined properties on your message flows using the WebSphere Message Broker Explorer:

Procedure

1. In the Navigator view, expand the Brokers folder, and navigate to the message flow on which you want view the user-defined properties.
2. Right-click the message flow, and click **Properties**.
3. Click **User Defined Properties** to display a list of the user-defined properties and their value defined on the message flow.
4. Optional: if you want to modify the user-defined properties, change the value in the appropriate rows in the Value column, and click **Apply**. A message is sent to the broker to change the value of the user-defined property.
5. Click **OK** to close the Properties window.

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

"User-defined properties" on page 1147
A user-defined property (UDP) is a property that is defined when you construct a message flow by using the Message Flow editor. This property can be used by the ESQL or Java program inside message flow nodes, such as a Compute node.

"User-defined properties in ESQL" on page 2376
Access user-defined properties (UDPs) as variables in your ESQL program by specifying the EXTERNAL keyword on a DECLARE statement. For example, the ESQL statement `DECLARE today EXTERNAL CHARACTER 'monday'` defines a user-defined property called today with an initial value monday.

Related tasks:

"Configuring a message flow at deployment time with user-defined properties" on page 2626
Use user-defined properties (UDPs) to configure message flows at deployment and run time, without modifying program code. You can give a UDP an initial value when you declare it in your program, or when you use the Message Flow editor to create or modify a message flow.

"Accessing message flow user-defined properties from a JavaCompute node" on page 2659
Customize a JavaCompute node to access properties that you have associated with the message flow in which the node is included.

Related reference:

"ESQL variables" on page 5048
ESQL variables can be described as *external variables*, *normal variables*, or *shared variables*; their use is defined in the DECLARE statement.

"Configurable message flow properties" on page 4020
When you add a message flow to a broker archive (BAR) file in preparation for deploying it to a broker, you can set additional properties that influence its run time operation. These properties are available for review and update when you

select the **Manage and Configure** tab for the broker archive file.

Developing applications that use the Administration API

Develop Java applications that use the Administration API (also known as the CMP API) to communicate with, deploy to, and manage brokers and their associated resources.

Before you begin

Before you start:

Read an overview of the "The Administration API for WebSphere Message Broker" on page 54, and how you can use it to manage the resources associated with your brokers.

Samples are provided to demonstrate typical CMP scenarios. Run and explore the samples to learn about what you can do with the CMP; see "The Administration API samples" on page 958.

About this task

To write administrative applications that run tasks in your brokers, see the following information:

- Configuring the environment
- Connecting to a broker
- Navigating brokers and broker resources
- Deploying resources
- Setting message flow user-defined properties at run time
- Working with UserDefined configurable service properties in JavaCompute nodes
- Managing brokers
- Managing brokers from JavaCompute nodes
- Working with resource manager statistics
- Submitting batch requests

Results

When you have created your applications, test their operation in your test environment. Check the results of the operations that the programs have performed by using the Deployment Log view.

What to do next

When you have written and tested your applications, distribute them to the computers from which you want your administrators to perform the tasks that you have developed.

Related concepts:

"The Administration API for WebSphere Message Broker" on page 54
The Administration API for WebSphere Message Broker is a programming interface that your applications can use to control brokers and their resources through a remote interface.

"The Administration API samples" on page 958
Explore the samples to learn the basic features that are provided by the Administration API (also known as the CMP API). Run the samples to deploy a BAR file or manage a broker, or use the CMP API Exerciser to implement various tasks.

Related tasks:

"Configuring an environment for developing and running CMP applications" on page 969
Prepare the environment in which you want to run your CMP applications.

Chapter 11, "Packaging and deploying," on page 3209
Package resources that you create in the WebSphere Message Broker Toolkit, such as message flows, and deploy them to execution groups on brokers.

Related information:

Administration API for WebSphere Message Broker (CMP API)

The Administration API for WebSphere Message Broker

The Administration API for WebSphere Message Broker is a programming interface that your applications can use to control brokers and their resources through a remote interface.

> The Administration API for WebSphere Message Broker is also known as the Configuration Manager Proxy, or CMP API
>
> The Configuration Manager has been removed from Version 7, and the full name of the API has changed. However, the terms CMP application and CMP API have been retained, and are used in this information center to refer to the Administration API, for continuity and consistency with the JAR file `ConfigManagerProxy.jar` that supplies all the classes.

The Administration API for WebSphere Message Broker (CMP API) consists solely of a Java implementation, and is referred to as the Message Broker Java API. Your applications have complete access to the broker functions and resources through the set of Java classes that constitute the CMP. Use the CMP API to interact with the broker to perform the following tasks:

- Deploy BAR files
- Change the broker configuration properties
- Create, modify, and delete execution groups
- Inquire and set the status of the broker and its associated resources, and to be informed if status changes
  - Execution groups
  - Deployed message flows
  - Deployed files used by the message flows (for example, JAR files)
- View the Administration log

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

"Execution groups" on page 53
An execution group is a named grouping of message flows that have been assigned to a broker. The broker enforces a degree of isolation between message flows in distinct execution groups by ensuring that they run in separate address spaces, or as unique processes.

"WebSphere Message Broker Toolkit" on page 31
The WebSphere Message Broker Toolkit is an integrated development environment and graphical user interface based on the Eclipse platform.

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

"The Administration API samples"
Explore the samples to learn the basic features that are provided by the Administration API (also known as the CMP API). Run the samples to deploy a BAR file or manage a broker, or use the CMP API Exerciser to implement various tasks.

Related tasks:

"Developing applications that use the Administration API" on page 956
Develop Java applications that use the Administration API (also known as the CMP API) to communicate with, deploy to, and manage brokers and their associated resources.

Related information:

Administration API for WebSphere Message Broker (CMP API)

The Administration API samples

Explore the samples to learn the basic features that are provided by the Administration API (also known as the CMP API). Run the samples to deploy a BAR file or manage a broker, or use the CMP API Exerciser to implement various tasks.

Deploy BAR

The Deploy BAR sample deploys a BAR file to an execution group, and displays the outcome. Read about this sample in "Running the CMP Deploy BAR sample" on page 959.

Broker management

The broker management sample uses the CMP API to display to the screen the complete run state of the broker. "Running the CMP broker management sample" on page 960 describes this sample in more detail.

The CMP API Exerciser

The CMP API Exerciser is a graphical interface to the CMP that you can use to view and manipulate brokers. Use the CMP API Exerciser sample to view and manage a broker, or record and play back configuration scripts. You can also customize the CMP API Exerciser to tailor the tasks to suit your requirements. See "Running the CMP API Exerciser sample" on page 962 for more information.

Modify CMP API samples

You can modify the CMP API samples, and change various parameters that affect how the samples run. Read "Modifying the CMP samples" on page 968 and follow the guidance given about possible changes.

Related concepts:

"The Administration API for WebSphere Message Broker" on page 54
The Administration API for WebSphere Message Broker is a programming interface that your applications can use to control brokers and their resources through a remote interface.

Related tasks:

"Running the CMP Deploy BAR sample"
Run the CMP Deploy BAR sample to deploy a broker archive file to a broker.

"Running the CMP broker management sample" on page 960
Run the broker management sample to display the complete run state of a broker.

"Running the CMP API Exerciser sample" on page 962
Run the CMP API Exerciser sample to view and manage a broker, customize the CMP API Exerciser, or record and play back configuration scripts.

"Modifying the CMP samples" on page 968
Modify the CMP samples to change the parameters that they use to complete their tasks.

Related information:

Administration API for WebSphere Message Broker (CMP API)

Running the CMP Deploy BAR sample

Run the CMP Deploy BAR sample to deploy a broker archive file to a broker.

Before you begin

Before you start:

The BAR file and execution group name, and the connection details that define the target broker, are hard coded into the application. You can run this sample unchanged, or you can modify the values and parameters that it uses to apply them to your own configuration.

If you modify the Deploy BAR sample, you must update and recompile the source file before you run it. The source file for this sample is located in the following directory:

install_dir/sample/ConfigManagerProxy/cmp/DeployBar.java

About this task

Use the Deploy BAR sample to deploy a BAR file to an execution group, and display the outcome.

Procedure

1. Run the Deploy BAR sample by entering the appropriate command for your platform:
   - Windows On Windows, open a Command Console and run the following command:

     install_dir\sample\ConfigManagerProxy\StartDeployBAR.bat
   - Linux UNIX z/OS On other platforms:
     a. Start a broker command environment by running **mqsiprofile**, or follow the guidance provided in the StartDeployBar shell script to configure the correct CLASSPATH for your environment.
     b. Run the shell script:

        install_dir\sample\ConfigManagerProxy\StartDeployBAR

The default connection parameters used by the sample are shown in the following table.

Connection parameter	Description
`"localhost"`	Host name of the computer where the broker is running.
`"MB7QMGR"`	Name of the broker queue manager.
`2414`	Port on which the broker queue manager is listening
`"default"`	Name of the execution group.
`"mybar.bar"`	Fully qualified name of the BAR file to deploy.

The CMP connects to the broker that is running on the local computer (defined by `localhost`). The queue manager `MB7QMGR` must be listening on port `2414`. Next, the CMP deploys the file `mybar.bar` to the predefined execution group `default`.

2. Check the results of the sample by viewing the broker in the WebSphere Message Broker Toolkit, or by using the CMP API Exerciser.

What to do next

Next: Run another sample, or work with the CMP API Exerciser.

Related concepts:

"The Administration API for WebSphere Message Broker" on page 54
The Administration API for WebSphere Message Broker is a programming interface that your applications can use to control brokers and their resources through a remote interface.

"The Administration API samples" on page 958
Explore the samples to learn the basic features that are provided by the Administration API (also known as the CMP API). Run the samples to deploy a BAR file or manage a broker, or use the CMP API Exerciser to implement various tasks.

Related tasks:

Chapter 11, "Packaging and deploying," on page 3209
Package resources that you create in the WebSphere Message Broker Toolkit, such as message flows, and deploy them to execution groups on brokers.

"Running the CMP broker management sample"
Run the broker management sample to display the complete run state of a broker.

"Running the CMP API Exerciser sample" on page 962
Run the CMP API Exerciser sample to view and manage a broker, customize the CMP API Exerciser, or record and play back configuration scripts.

"Modifying the CMP samples" on page 968
Modify the CMP samples to change the parameters that they use to complete their tasks.

Related information:

Administration API for WebSphere Message Broker (CMP API)

Running the CMP broker management sample

Run the broker management sample to display the complete run state of a broker.

Before you begin

Before you start:

You can run this sample unchanged, or you can modify the values and parameters that it uses to apply them to your own configuration.

If you modify this management sample, you must update and recompile the source file before you run it. The source file for this sample is located in the following directory:

install_dir/sample/ConfigManagerProxy/cmp/BrokerInfo.java

About this task

Use the broker management sample to display the complete run state of the target broker.

Procedure

- Run the broker management sample by entering the appropriate command for your platform. Replace *connection_file* with the fully qualified file path to the broker file (with extension `.broker`).
  - Windows On Windows, use the Command Console to run the following command:

    install_dir\sample\ConfigManagerProxy\StartBrokerInfo.bat *connection_file*
  - Linux UNIX z/OS On other platforms:
    1. Start a broker command environment by running **mqsiprofile**, or follow the guidance provided in the StartBrokerInfo shell script to configure the correct CLASSPATH for your environment.
    2. Run the shell script:

       install_dir\sample\ConfigManagerProxy\StartBrokerInfo *connection_file*

  The complete run state of the broker is displayed. You can see responses like the following output:

```
(28/01/09 11:47:43) Connecting. Please wait...
(28/01/09 11:47:46) Successfully connected to the broker's queue manager.
(28/01/09 11:47:49) Successfully connected to the broker.
(28/01/09 11:47:49) Broker 'MB7BROKER' is running.
(28/01/09 11:47:50)   Execution group 'default' on 'MB7BROKER' is running.
(28/01/09 11:47:51)     Message flow 'simpleflow' on 'default' on 'MB7BROKER' is running.
(28/01/09 11:47:51) Disconnected.
```

- If you prefer, you can run this sample in interactive mode, which causes the sample to listen for changes to the broker.
  1. To run the sample in interactive mode, specify the **-i** option on the command.

     For example:

```
\sample\ConfigManagerProxy\StartBrokerInfo.bat
 c:\myBroker.broker  -i
```

     In interactive mode, you see the output shown earlier, and the additional response:

```
(13/08/08 15:53:46) Listening for changes to the broker...
```

  2. To stop the sample when it is running in interactive mode, force it to end by using CTRL+C.

What to do next

Next: Run another sample, or work with the CMP API Exerciser.

Related concepts:

"The Administration API for WebSphere Message Broker" on page 54
The Administration API for WebSphere Message Broker is a programming interface that your applications can use to control brokers and their resources through a remote interface.

"The Administration API samples" on page 958
Explore the samples to learn the basic features that are provided by the Administration API (also known as the CMP API). Run the samples to deploy a BAR file or manage a broker, or use the CMP API Exerciser to implement various tasks.

Related tasks:

"Running the CMP Deploy BAR sample" on page 959
Run the CMP Deploy BAR sample to deploy a broker archive file to a broker.

"Running the CMP API Exerciser sample"
Run the CMP API Exerciser sample to view and manage a broker, customize the CMP API Exerciser, or record and play back configuration scripts.

"Modifying the CMP samples" on page 968
Modify the CMP samples to change the parameters that they use to complete their tasks.

Related information:

Administration API for WebSphere Message Broker (CMP API)

Running the CMP API Exerciser sample

Run the CMP API Exerciser sample to view and manage a broker, customize the CMP API Exerciser, or record and play back configuration scripts.

Before you begin

Before you start:

You can run this sample unchanged, or you can modify the values and parameters that it uses to apply them to your own configuration.

If you modify the CMP API Exerciser sample, you must update and recompile the source file before you run it. The source file for this sample is located in the following directory:

install_dir/sample/ConfigManagerProxy/cmp/exerciser

About this task

Use the CMP API Exerciser to complete the following tasks:

- "Viewing and managing a broker in the CMP API Exerciser" on page 963
- "Customizing the CMP API Exerciser" on page 965
- "Recording and playing back configuration scripts using the CMP API Exerciser" on page 967

Related concepts:

"The Administration API for WebSphere Message Broker" on page 54
The Administration API for WebSphere Message Broker is a programming interface that your applications can use to control brokers and their resources through a

remote interface.

"The Administration API samples" on page 958
Explore the samples to learn the basic features that are provided by the Administration API (also known as the CMP API). Run the samples to deploy a BAR file or manage a broker, or use the CMP API Exerciser to implement various tasks.

Related tasks:

"Modifying the CMP samples" on page 968
Modify the CMP samples to change the parameters that they use to complete their tasks.

Related information:

Administration API for WebSphere Message Broker (CMP API)

Viewing and managing a broker in the CMP API Exerciser:

Use the CMP API Exerciser sample to view and manage a broker.

About this task

To view and manage a broker by using the CMP API Exerciser, complete the following steps:

Procedure

1. Start the CMP API Exerciser:
   - Windows On Windows, click **Start** > **All Programs** > **IBM WebSphere Message Broker 7.0** > **Java Programming APIs** > **CMP API Exerciser**.
   - Linux UNIX z/OS On other platforms:
     a. Start a broker command environment by running **mqsiprofile**, or follow the guidance provided in the `StartConfigManagerProxyExerciser` shell script to configure the correct CLASSPATH for your environment.
     b. Ensure that your user ID has writer permission to the current directory. The CMP API Exerciser stores its configuration settings in a file in this directory.
     c. Run the shell script:
        install_dir`\sample\ConfigManagerProxy\StartConfigManagerProxyExerciser`

   The CMP API Exerciser window opens.
2. Connect to a running broker by clicking either **File** > **Connect to Local Broker** or **File** > **Connect to Remote Broker**.

   The Connect to a Broker dialog opens.
3. Enter the connection parameters to the broker, then click **Submit**.

   Broker information is retrieved and displayed in the CMP API Exerciser window. You have now connected to the broker.

   The upper left of the screen contains a hierarchical representation of the broker to which you are connected. Selecting objects in the tree causes the table on the right to change, reflecting the attributes of the object that you select. The Method column lists CMP API methods that you can call in your own Java applications, and the Result column indicates the data that is returned by calling the CMP API method on the selected object.
4. Run a CMP API method against a broker object. CMP API methods are used to manage objects in a broker.

a. In the navigation tree, right-click a broker.

   A pop-up menu opens to show all the available CMP API methods.

b. Select **Create execution group**. The Create execution group dialog opens.

c. Enter the name for a new execution group and click **Submit**. The output from the method is displayed in the log window at the bottom of the screen. For example:

```
(31/03/09 16:53:50) ----> cmp.exerciser.ClassTesterForBrokerProxy.testCreateEG
 (MB7BROKER, "eg1")
(31/03/09 16:53:50) The request was successfully sent to the broker.
(31/03/09 16:53:50) <---- cmp.exerciser.ClassTesterForBrokerProxy.testCreateEG
```

You also see messages that are returned from the broker when the method is called. For example:

```
(31/03/09 16:53:50) ----> cmp.exerciser.ExerciserAdministeredObjectListener.processActionResponse(...)
(31/03/09 16:53:50) affectedObject = MB7BROKER
(31/03/09 16:53:50) completionCode = success
(31/03/09 16:53:50) (Reference property) commsmessage.lastinbatch=true
(31/03/09 16:53:50) (Reference property) uuid=595e1d10-3875-11d4-a485-000629be5bf8
(31/03/09 16:53:50) (Reference property) child.uuid=1d8b3c5d-2001-0000-0080-c2000502e620
(31/03/09 16:53:50) (Reference property) configmanagerproxy.osname=Windows XP
(31/03/09 16:53:50) (Reference property) child.name=eg1
(31/03/09 16:53:50) (Reference property) userid=Matt
(31/03/09 16:53:50) (Reference property) configmanagerproxy.hostname=lucas
(31/03/09 16:53:50) (Reference property) commsmessage.configobjecttype=Broker
(31/03/09 16:53:50) (Reference property) type=Broker
(31/03/09 16:53:50) (Reference property) child.type=ExecutionGroup
(31/03/09 16:53:50) (Reference property) commsmessage.operationtype=createchild
(31/03/09 16:53:50) (Reference property) configmanagerproxy.noeventlog=false
(31/03/09 16:53:50) (Reference property) eg.arch=0
(31/03/09 16:53:50) <---- cmp.exerciser.ExerciserAdministeredObjectListener.processActionResponse()
```

In this example, `completionCode = success` means that the request to create the execution group is successful. The lines marked `(Reference property)` describe the request to which the response refers.

Results

During these steps you connected to a broker, viewed the broker information, and performed a management task by using the CMP API Exerciser.

What to do next

Next: Continue to work with the CMP API Exerciser, or run another sample.

Related concepts:

"The Administration API for WebSphere Message Broker" on page 54
The Administration API for WebSphere Message Broker is a programming interface that your applications can use to control brokers and their resources through a remote interface.

"The Administration API samples" on page 958
Explore the samples to learn the basic features that are provided by the Administration API (also known as the CMP API). Run the samples to deploy a BAR file or manage a broker, or use the CMP API Exerciser to implement various tasks.

Related tasks:

"Running the CMP API Exerciser sample" on page 962
Run the CMP API Exerciser sample to view and manage a broker, customize the CMP API Exerciser, or record and play back configuration scripts.

"Customizing the CMP API Exerciser"
Enable or disable a selection of options to customize the CMP API Exerciser to meet your requirements.

"Recording and playing back configuration scripts using the CMP API Exerciser" on page 967
Use the CMP API Exerciser to record and play back configuration scripts.

"Modifying the CMP samples" on page 968
Modify the CMP samples to change the parameters that they use to complete their tasks.

Related information:

Administration API for WebSphere Message Broker (CMP API)

Customizing the CMP API Exerciser:

Enable or disable a selection of options to customize the CMP API Exerciser to meet your requirements.

About this task

To customize the CMP API Exerciser, complete the following steps:

Procedure

1. Start the CMP API Exerciser.
   - Windows On Windows, click **Start** > **All Programs** > **IBM WebSphere Message Broker 7.0** > **Java Programming APIs** > **CMP API Exerciser**.
   - Linux UNIX z/OS On other operating systems, run the following shell script:

     ```
     install_dir\sample\ConfigManagerProxy\StartConfigManagerProxyExerciser
     ```

   The CMP API Exerciser window opens.
2. Customize the CMP API Exerciser by selecting one or more of the following options from the **File** menu.
   a. Optional: Click **File** > **Discover Subcomponent Tree Recursively** to enable or disable this option.
      - If you enable this option, the CMP API Exerciser connects to a broker and discovers all defined broker objects.
      - If you disable this option, only the top-level objects are discovered; you must select the context-sensitive option, **Discover subcomponents**, to iterate down the object hierarchy.

   b. Optional: Click **File** > **Use Incremental Deployment** to enable or disable this option.
      - If you enable this option, all deploy operations cause a delta (incremental) deploy where relevant.
      - If you disable this option, all deploy operations cause a complete deploy.

   c. Optional: Click **File** > **Show Advanced Properties** to enable or disable this option.
      - If you enable this option, output from all available methods is displayed in the right pane of the CMP API Exerciser.
      - If you disable this option, output from a subset of the available methods is displayed in the right pane of the CMP API Exerciser.

   d. Optional: Click **File** > **Connect Using .broker Properties File** to enable or disable this option.

- If you enable this option, a file dialog opens when you connect to a broker. Use the file dialog to navigate to a file with a `.broker` extension, which provides the connection parameters to the queue manager that hosts the broker.
- If you disable this option, you are prompted to enter the queue manager connection parameters, security exit parameters, host name, and port.

e. Optional: Click **File** > **Enable MQ Java Client Service Trace** to enable or disable this option.
   - If you enable this option, a level 5 service trace of the WebSphere MQ Classes for Java runs. A trace dialog opens; specify a file name to which trace records are written.
   - If you disable this option, level 5 service tracing of the WebSphere MQ Classes for Java is not done.

f. Optional: Click **File** > **Enable Broker Administration Service Trace** to enable or disable this option.
   - If you enable this option, a service trace of the CMP API run. A trace dialog opens; specify a file name to which trace records are written.
   - If you disable this option, service tracing of the CMP API is not done.

g. Optional: Click **File** > **Set Timeout Characteristics**.

   Specify the time, in seconds, that the CMP API Exerciser waits for responses from the broker. The default wait interval is 6 seconds.

What to do next

Next: Continue to work with the CMP API Exerciser, or run another sample.

Related concepts:

"The Administration API for WebSphere Message Broker" on page 54
The Administration API for WebSphere Message Broker is a programming interface that your applications can use to control brokers and their resources through a remote interface.

"The Administration API samples" on page 958
Explore the samples to learn the basic features that are provided by the Administration API (also known as the CMP API). Run the samples to deploy a BAR file or manage a broker, or use the CMP API Exerciser to implement various tasks.

Related tasks:

"Modifying the CMP samples" on page 968
Modify the CMP samples to change the parameters that they use to complete their tasks.

"Running the CMP API Exerciser sample" on page 962
Run the CMP API Exerciser sample to view and manage a broker, customize the CMP API Exerciser, or record and play back configuration scripts.

"Viewing and managing a broker in the CMP API Exerciser" on page 963
Use the CMP API Exerciser sample to view and manage a broker.

"Recording and playing back configuration scripts using the CMP API Exerciser" on page 967
Use the CMP API Exerciser to record and play back configuration scripts.

Related information:

Administration API for WebSphere Message Broker (CMP API)

Recording and playing back configuration scripts using the CMP API Exerciser:

Use the CMP API Exerciser to record and play back configuration scripts.

About this task

You can run a script file from the command line, a shell window, or from a batch file. If you run the script by using one of these methods, ensure the first action completed by the script is to connect to a broker.

Procedure

1. Start the CMP API Exerciser.
   - Windows On Windows, click **Start** > **All Programs** > **IBM WebSphere Message Broker 7.0** > **Java Programming APIs** > **CMP API Exerciser**.
   - Linux UNIX z/OS On other operating systems, run the following shell script:

     ```
     install_dir\sample\ConfigManagerProxy\StartConfigManagerProxyExerciser
     ```

   The CMP API Exerciser window opens.
2. Start recording a script by clicking **Scripting** > **Record New Script**. The Save dialog opens. Type a name for the script file, select an appropriate file location, and click **Save**.
   a. Complete one or more actions on a broker by using the CMP API Exerciser.

      If you start recording before you run actions against the broker, the first action taken is to connect to the broker; however, you can start recording a script at any point during the management of a broker.
   b. Optional: Insert a pause by clicking **Scripting** > **Insert a pause**.

      The CMP API Exerciser pauses so that responses can be returned before the next action is issued. Use this option to avoid naming conflicts if you are deleting and recreating objects of the same name.

      The Insert a pause dialog opens; specify the duration of the pause in seconds.
   c. Stop recording the script by clicking **Scripting** > **Stop Recording**.

      Information about the actions taken are saved to the script file.
3. To replay the script file:
   a. Click **Scripting** > **Play Back Recorded Script**.

      The Open dialog opens.
   b. Select the appropriate script file and click **Open**. The script file is replayed.

What to do next

Next: Continue to work with the CMP API Exerciser, or run another sample.

Related concepts:

"The Administration API for WebSphere Message Broker" on page 54
The Administration API for WebSphere Message Broker is a programming interface that your applications can use to control brokers and their resources through a remote interface.

"The Administration API samples" on page 958
Explore the samples to learn the basic features that are provided by the Administration API (also known as the CMP API). Run the samples to deploy a BAR file or manage a broker, or use the CMP API Exerciser to implement various

tasks.

Related tasks:

"Running the CMP API Exerciser sample" on page 962
Run the CMP API Exerciser sample to view and manage a broker, customize the CMP API Exerciser, or record and play back configuration scripts.

"Viewing and managing a broker in the CMP API Exerciser" on page 963
Use the CMP API Exerciser sample to view and manage a broker.

"Customizing the CMP API Exerciser" on page 965
Enable or disable a selection of options to customize the CMP API Exerciser to meet your requirements.

"Modifying the CMP samples"
Modify the CMP samples to change the parameters that they use to complete their tasks.

Related information:

Administration API for WebSphere Message Broker (CMP API)

Modifying the CMP samples

Modify the CMP samples to change the parameters that they use to complete their tasks.

About this task

If you change a sample, you must recompile the source code to implement those changes. For example, you might choose to change the default connection parameters. If you modify these and other values, you change the behavior of the sample.

To modify a sample, perform the following steps:

Procedure

1. Set up the environment by following the instructions provided in "Configuring an environment for developing and running CMP applications" on page 969.
2. Locate the source file for the sample that you want to change.

   The source files for the samples are located in the following directories:

   Deploy BAR sample
   install_dir/sample/ConfigManagerProxy/cmp/DeployBar.java

   Broker management sample
   install_dir/sample/ConfigManagerProxy/cmp/BrokerInfo.java

   CMP API Exerciser sample
   install_dir/sample/ConfigManagerProxy/cmp/exerciser.java
3. Open the source file, and modify the appropriate parameters.
4. Save and recompile the source file.

What to do next

Next: Run the modified sample by following the instructions in the appropriate link:

- "Running the CMP Deploy BAR sample" on page 959
- "Running the CMP broker management sample" on page 960
- "Running the CMP API Exerciser sample" on page 962

Related concepts:

"The Administration API for WebSphere Message Broker" on page 54
The Administration API for WebSphere Message Broker is a programming interface that your applications can use to control brokers and their resources through a remote interface.

"The Administration API samples" on page 958
Explore the samples to learn the basic features that are provided by the Administration API (also known as the CMP API). Run the samples to deploy a BAR file or manage a broker, or use the CMP API Exerciser to implement various tasks.

Related information:

Administration API for WebSphere Message Broker (CMP API)

Configuring an environment for developing and running CMP applications

Prepare the environment in which you want to run your CMP applications.

Before you begin

Before you start:

To develop and run Java applications that use the CMP API, you must install the following prerequisite software in your local computer environment:

- The WebSphere MQ Classes for Java.

  These classes provide the internal wire protocol that your applications use to communicate with the broker.

  You can install the classes from the media supplied with WebSphere Message Broker.
- An IBM Java Development Kit (JDK) at a supported Java level. Java support is defined in "Additional software requirements" on page 3598.

  WebSphere Message Broker does not supply a JDK; you must acquire and install a suitable product yourself.

  You must use an IBM JDK to develop CMP applications and an IBM Java Runtime Environment (JRE) to run these applications.

About this task

To set up your computer in preparation for building and running CMP applications, you must configure your class path environment variable so that it includes the WebSphere MQ Classes for Java, and the JAR file that defines the CMP.

Follow the instructions provided for the appropriate environment:

- "Configuring the Windows command-line environment to run CMP applications" on page 970
- "Configuring Linux, UNIX, and z/OS command-line environments to run CMP applications" on page 971
- "Configuring the Eclipse environment to run CMP applications" on page 972

You can also run CMP applications, and therefore control one or more broker components, from computers on which you have not installed WebSphere Message Broker. For more information, see "Configuring environments without the broker component installed" on page 973.

The JAR file `ConfigManagerProxy.jar` contains the English message catalog for displaying broker (BIP) messages from the Administration log of the broker. If you want a CMP application to display broker messages in a language other than English, you must also add the directory that contains the localized message catalogs to your class path; for example, `C:\Program Files\IBM\MQSI\v.r\messages` on Windows operating systems. The default directory includes the version and release of the product, in the format v.r (version.release).

You can also use the CMP to display or log messages from a catalog that you create yourself.

Related concepts:

"The Administration API for WebSphere Message Broker" on page 54
The Administration API for WebSphere Message Broker is a programming interface that your applications can use to control brokers and their resources through a remote interface.

Related tasks:

"Configuring the Windows command-line environment to run CMP applications"
Use system facilities to configure environment variables on your Windows computers to run your CMP applications.

"Configuring Linux, UNIX, and z/OS command-line environments to run CMP applications" on page 971
Use system facilities to configure environment variables on your Linux, UNIX, and z/OS computers to run your CMP applications.

"Configuring the Eclipse environment to run CMP applications" on page 972
Use Eclipse facilities to configure the environment to run your CMP applications.

"Configuring environments without the broker component installed" on page 973
Install and run CMP applications and other utilities on computers, when you are using a local ID only, on which you have not installed the broker component.

"Creating message catalogs" on page 3138
Create your own message catalogs to write tailored entries to the local error log.

Related reference:

"Additional software requirements" on page 3598
WebSphere Message Broker requires additional software products to run successfully.

Related information:

Administration API for WebSphere Message Broker (CMP API)

WebSphere MQ Version 7 Information Center online

Configuring the Windows command-line environment to run CMP applications

Use system facilities to configure environment variables on your Windows computers to run your CMP applications.

About this task

Update the CLASSPATH environment variable:

Procedure

1. Add the CMP JAR file to your CLASSPATH. For example:

```
set CLASSPATH = %CLASSPATH%;%install_dir%\classes\ConfigManagerProxy.jar
```

2. Add the WebSphere MQ Classes for Java JAR file `com.ibm.mq.jar` to your CLASSPATH in the same way.

   On 32-bit operating system editions, the file is typically stored in the directory `C:\Program Files\IBM\WebSphere MQ\java\lib`. On 64-bit operating system editions, the file is typically stored in the directory `C:\Program Files (x86)\IBM\WebSphere MQ\java\lib`.

   For more information about the WebSphere MQ Classes for Java, see the *Using Java* section in the WebSphere MQ information center.
3. Add your Java development directory to the CLASSPATH in the same way.

What to do next

Next: Use the tools that are provided by your JDK to build and run your CMP applications.

Related concepts:

"The Administration API for WebSphere Message Broker" on page 54
The Administration API for WebSphere Message Broker is a programming interface that your applications can use to control brokers and their resources through a remote interface.

Related tasks:

"Configuring an environment for developing and running CMP applications" on page 969
Prepare the environment in which you want to run your CMP applications.

"Configuring Linux, UNIX, and z/OS command-line environments to run CMP applications"
Use system facilities to configure environment variables on your Linux, UNIX, and z/OS computers to run your CMP applications.

"Configuring the Eclipse environment to run CMP applications" on page 972
Use Eclipse facilities to configure the environment to run your CMP applications.

Related information:

Administration API for WebSphere Message Broker (CMP API)

WebSphere MQ Version 7 Information Center online

Configuring Linux, UNIX, and z/OS command-line environments to run CMP applications

Use system facilities to configure environment variables on your Linux, UNIX, and z/OS computers to run your CMP applications.

About this task

Update the CLASSPATH environment variable:

Procedure

1. Add the CMP JAR to your CLASSPATH. For example:

```
export CLASSPATH = $CLASSPATH%;$install_dir/sample/ConfigManagerProxy/ConfigManagerProxy.jar
```

2. Add the WebSphere MQ Classes for Java JAR file `com.ibm.mq.jar` to your CLASSPATH in the same way.

   On Linux and UNIX systems, the file is typically installed in the directory *MQ_INSTALLATION_PATH*`/java/lib`.

   For more information about the WebSphere MQ Classes for Java, see the *Using Java* section in the WebSphere MQ information center.

3. Add your Java development directory to the CLASSPATH in the same way.

What to do next

Next: Use the tools that are provided by your JDK to build and run your CMP applications.

Related concepts:

"The Administration API for WebSphere Message Broker" on page 54
The Administration API for WebSphere Message Broker is a programming interface that your applications can use to control brokers and their resources through a remote interface.

Related tasks:

"Configuring an environment for developing and running CMP applications" on page 969
Prepare the environment in which you want to run your CMP applications.

"Configuring the Windows command-line environment to run CMP applications" on page 970
Use system facilities to configure environment variables on your Windows computers to run your CMP applications.

"Configuring the Eclipse environment to run CMP applications"
Use Eclipse facilities to configure the environment to run your CMP applications.

Related information:

Administration API for WebSphere Message Broker (CMP API)

WebSphere MQ Version 7 Information Center online

Configuring the Eclipse environment to run CMP applications

Use Eclipse facilities to configure the environment to run your CMP applications.

About this task

Update the CLASSPATH environment variable:

Procedure

1. In your Eclipse environment, select **File** > **New** > **Project**. The New Project wizard opens.
2. Select **Java Project** from the options displayed. Click **Next**. The New Java Project window opens.
3. Enter a name for your new project. Click **Next**.
4. Select the **Libraries** tab, and click **Add External Jars**.
5. To set up the build environment, navigate to the *install_dir*/classes subdirectory. For example, for Version 7.0 on Windows 32-bit operating system editions, navigate to the directory C:\Program Files\IBM\MQSI\7.0\classes. Select the file ConfigManagerProxy.jar, and click **Open**. The file is added to the list in the window for the **Libraries** tab.
6. When you have added the file ConfigManagerProxy.jar to the build path, set up the runtime environment. Click **Add External Jars** again, and navigate to the *WebSphere_MQ_installation_directory*/java/lib subdirectory. For example, on Linux on x86, navigate to the directory /opt/mqm/java/lib. Select the file com.ibm.mq.jar, and click **Open**. This file is also added to the list.

   For more information about the WebSphere MQ Classes for Java, see the *Using Java* section in the WebSphere MQ information center.

7. Click **Finish** to close the New Project wizard.

What to do next

Next: Use the Eclipse development tools to build and run your CMP applications.

Related concepts:

"The Administration API for WebSphere Message Broker" on page 54
The Administration API for WebSphere Message Broker is a programming interface that your applications can use to control brokers and their resources through a remote interface.

Related tasks:

"Configuring an environment for developing and running CMP applications" on page 969
Prepare the environment in which you want to run your CMP applications.

"Configuring the Windows command-line environment to run CMP applications" on page 970
Use system facilities to configure environment variables on your Windows computers to run your CMP applications.

"Configuring Linux, UNIX, and z/OS command-line environments to run CMP applications" on page 971
Use system facilities to configure environment variables on your Linux, UNIX, and z/OS computers to run your CMP applications.

Related information:

Administration API for WebSphere Message Broker (CMP API)

WebSphere MQ Version 7 Information Center online

Configuring environments without the broker component installed

Install and run CMP applications and other utilities on computers, when you are using a local ID only, on which you have not installed the broker component.

About this task

You can run Java applications that use the CMP API even where you have not installed the broker component. These CMP applications include your own applications, and the following command utilities:

- **mqsicreateexecutiongroup**
- **mqsideleteexecutiongroup**
- **mqsimode**
- **mqsireloadsecurity**; the CMP application version of **mqsireloadsecurity** is called mqsireloadsecurityscript.
- **mqsistartmsgflow**
- **mqsistopmsgflow**

To install CMP applications in an environment that does not have the broker component installed, complete the following steps:

Procedure

1. Ensure that the target computer has a compatible Java Runtime Environment (JRE). Because you are not installing the broker component, which includes a JRE, you must use an alternative option.

Note, that you must also set MQSI_JREPATH to the installation path of your JRE.

Java support is defined in "Additional software requirements" on page 3598.

2. Copy the following set of files from a computer that has the broker component installed to the target computer:
   a. `ConfigManagerProxy.jar` from the `classes` directory.
   b. The WebSphere MQ Classes for Java.
      - On Windows, these classes are located in the file `com.ibm.mq.jar`.
      - On other platforms, these classes are located in the component's installation image.
   c. Your CMP application and all configuration files, for example `.broker` files.
   d. If you want to run the broker commands on the target computer, complete the following steps:
      1) Copy `brokerutil.jar` from the `classes` directory.
      2) Copy the required utility bat files, or shell scripts, from the `bin` directory. Shell scripts have a `.bat` extension on Windows or no extension on UNIX platforms:
         - **`mqsicreateexecutiongroup`** or **`mqsicreateexecutiongroup.`**`bat`
         - **`mqsideleteexecutiongroup`** or **`mqsideleteexecutiongroup.`**`bat`
         - **`mqsimode`** or **`mqsimode.`**`bat`
         - **`mqsireloadsecurity`** or **`mqsireloadsecurity`**`script.bat`
         - **`mqsistartmsgflow`** or **`mqsistartmsgflow.`**`bat`
         - **`mqsistopmsgflow`** or **`mqsistopmsgflow.`**`bat`
   e. If you want to display broker (BIP) messages in English environments other than US English, copy all BIPmsgs*.properties files from the `messages` directory.
3. On the target computer, use system facilities to update the CLASSPATH environment variable to include the following files:
   - The JAR file that contains the definitions of the CMP classes, `ConfigManagerProxy.jar`.
   - Your applications that import the CMP classes.
   - The WebSphere MQ Classes for Java, `com.ibm.mq.jar`, and any additional JAR files required by this package.
   - Any other required JAR files and directories. For example, if you require any of the available command utilities on the target computer, include `brokerutil.jar`; if you require the broker (BIP) messages to be displayed in locales other than US English, include a directory that contains BIPmsgs*.properties.
4. Ensure that the user ID that the target computer uses has the following authorities:
   - Authority to connect to the queue manager that the broker uses.
   - Authority to manipulate broker objects.

What to do next

Next: You can run your CMP applications, and the specified command utilities, on the target computer.

Related concepts:

"The Administration API for WebSphere Message Broker" on page 54
The Administration API for WebSphere Message Broker is a programming interface that your applications can use to control brokers and their resources through a remote interface.

Related tasks:

"Configuring an environment for developing and running CMP applications" on page 969
Prepare the environment in which you want to run your CMP applications.

"Managing brokers in a CMP application" on page 989
Manage the brokers and their resources from a CMP application.

Related reference:

"Additional software requirements" on page 3598
WebSphere Message Broker requires additional software products to run successfully.

"**mqsicreateexecutiongroup** command" on page 3854
Use the **mqsicreateexecutiongroup** command to add a new execution group to a broker.

"**mqsideleteexecutiongroup** command" on page 3869
Use the **mqsideleteexecutiongroup** command to remove an execution group from a broker.

"**mqsimode** command" on page 3899
Use the **mqsimode** command to configure and retrieve operation mode information.

"**mqsireloadsecurity** command" on page 3911
Use the **mqsireloadsecurity** command to force the immediate expiry of some or all the entries in the security cache.

"**mqsistartmsgflow** command" on page 3969
Use the **mqsistartmsgflow** command to start execution groups and message flows.

"**mqsistopmsgflow** command" on page 3975
Use the **mqsistopmsgflow** command to stop execution groups and message flows.

Related information:

WebSphere MQ Version 7 Information Center online

Connecting to a broker from a CMP application

Connect an application that uses the CMP API to a broker, to send requests about its status and its resources.

Before you begin

Before you start

Before starting this step, you must have completed "Configuring an environment for developing and running CMP applications" on page 969.

About this task

Consider the following program `BrokerRunStateChecker.java`. It connects to a broker that is running on the default queue manager of the local computer.

```
import com.ibm.broker.config.proxy.*;

public class BrokerRunStateChecker {
```

```
    public static void main(String[] args) {

      // The ip address of where the broker is running
       // and the port number of the queue manager listener.
        displayBrokerRunState("localhost", 2414, "");
    }

    public static void displayBrokerRunState(String hostname,
                                             int port,
                                             String qmgr) {
        BrokerProxy b = null;
        try {
            BrokerConnectionParameters bcp =
                new MQBrokerConnectionParameters(hostname, port, qmgr);
            b = BrokerProxy.getInstance(bcp);
            String brokerName = b.getName();

            System.out.println("Broker '"+brokerName+
                "' is available!");
            b.disconnect();
        } catch (ConfigManagerProxyException ex) {
            System.out.println("Broker is NOT available"+
                " because "+ex);
        }
    }
}
```

The first line of the program requests Java to import the CMP API classes, which are supplied in the package `com.ibm.broker.config.proxy`.

The first line of code inside the `try` block of the displayBrokerRunState() method instantiates a `BrokerConnectionParameters` object. This method is an interface which states that implementing classes are able to provide the parameters to connect to a broker.

The class MQBrokerConnectionParameters implements this interface; it defines a set of WebSphere MQ connection parameters. The constructor used here takes three parameters:

1. The host name of the computer that the broker is running on
2. The port on which the WebSphere MQ listener service for the broker is listening
3. The name of the queue manager that is associated with the broker

When you have defined this object, you can connect to the queue manager that is defined by those characteristics. The connection is achieved by the static getInstance() factory method just inside the `try` block. When a valid handle to the queue manager is returned, the application requests the name of the broker by using b.getName(), and displays it.

getName(), and other methods that request information from the broker, block until the information is supplied, or a timeout occurs. Therefore, if the broker is not running, the application hangs for a period. You can control the timeout period by using the BrokerProxy.setRetryCharacteristics() method. Typically, blocking only occurs when a given resource is accessed for the first time within an application.

Finally, the program calls the disconnect() method. This method frees up resources associated with the connection in both the CMP and the broker.

When a `BrokerProxy` handle is first returned from the getInstance() method, the broker service does not have to be running. It is only when the application uses the handle (by calling getName() in this example) that the application can be assured that a two-way connection with the broker is active.

Related concepts:

"The Administration API for WebSphere Message Broker" on page 54
The Administration API for WebSphere Message Broker is a programming interface that your applications can use to control brokers and their resources through a remote interface.

Related tasks:

"Configuring an environment for developing and running CMP applications" on page 969
Prepare the environment in which you want to run your CMP applications.

Related information:

Administration API for WebSphere Message Broker (CMP API)

Navigating brokers and broker resources in a CMP application

Explore the status and attributes of the broker that your CMP application is connected to, and discover information about its resources.

Before you begin

Before you start

Before starting this step, you must have completed "Connecting to a broker from a CMP application" on page 975.

About this task

Each resource that the broker can control is represented as a single object in the CMP API. An application can request status and other information about the following objects:

- Brokers
- Execution groups
- Deployed message flows
- Administration log

The CMP also handles deployed message sets; these resources are handled as attributes of deployed execution groups.

Collectively known as *administered objects*, these objects provide most of the interface to the broker, and are therefore fundamental to an understanding of the CMP.

Each administered object is an instance of a Java class that describes the underlying type of object in the broker. The main Java classes are shown in the following table.

Java class	Class function
BrokerProxy	Describes brokers.
ExecutionGroupProxy	Describes execution groups.

Java class	Class function
MessageFlowProxy	Describes message flows that have already been deployed to execution groups; does not describe message flows in the Broker Application Development perspective of the WebSphere Message Broker Toolkit.
LogProxy	Represents the log of recent administrative activity on the broker, for all users.

Each administered object describes a single object that can be controlled by the broker. For example, every execution group within a broker has one `ExecutionGroupProxy` instance that represents it within the application.

The `LogProxy` object includes messages, created as `LogEntry` objects, that record recent changes made to the administered objects. These messages include the following information:

- Administration requests; for example, a request to deploy a BAR file
- Results of administration requests
- Changes that have been made to objects; for example, a change to a property value through an administration task
- Changes that have been made to objects as a result of starting a broker at a new fix pack level or version.

You can use the `AdminQueueProxy` object to examine what items of work are in progress or are waiting for processing by the broker. The following code shows how you can access the queue:

```
BrokerConnectionParameters bcp =
  new MQBrokerConnectionParameters("localhost", 1414, "QMGR");
BrokerProxy b = BrokerProxy.getInstance(bcp);
AdminQueueProxy l = b.getAdministrationQueue();
```

A set of public methods is available for each administered object, which applications can use to inquire and manipulate properties of the underlying broker to which the instance refers. To access an administered object through its API, your application must first request a handle to that object from the object that logically owns it.

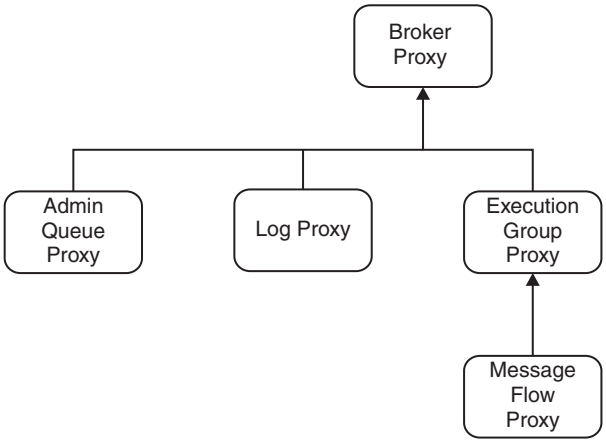
For example, because brokers logically own execution groups, to gain a handle to execution group `EG1` running on broker `B1`, the application must ask the `BrokerProxy` object represented by `B1` for a handle to the `ExecutionGroupProxy` object represented by `EG1`.

On a `BrokerProxy` object that refers to broker `B1`, the application can call methods that cause the broker to reveal its run-state, or cause it to start all its message flows. You can write applications to track the changes that are made to the broker objects by reading the messages maintained as `LogEntry` objects.

In the following example, a handle is requested to the `BrokerProxy` object. The `BrokerProxy` is logically the root of the administered object tree, therefore your application can access all other objects in the broker directly, or indirectly.

The broker directly owns the execution groups, therefore applications can call a method on the `BrokerProxy` object to gain a handle to the `ExectionGroupProxy` objects. Similarly, the execution group logically contains the set of all message flows, therefore the application can call methods on the `ExecutionGroupProxy` object to access the `MessageFlowProxy` objects.

The complete hierarchy of these access relationships is shown in the following diagram.



The following application traverses the administered object hierarchy to discover the run-state of a deployed message flow. The application assumes that message flow MF1 is deployed to EG1 on broker B1; you can substitute these values in the code for other values that are valid in the broker.

```
import com.ibm.broker.config.proxy.*;

public class GetMessageFlowRunState {

  public static void main(String[] args) {

    BrokerProxy b = null;
    try {
      BrokerConnectionParameters bcp =
         new MQBrokerConnectionParameters(
           "localhost",
           1414,
           "");
      b = BrokerProxy.getInstance(bcp);
    } catch (ConfigManagerProxyException cmpex) {
      System.out.println("Error connecting: "+cmpex);
    }

    if (b != null) {
      System.out.println("Connected to broker!");
      displayMessageFlowRunState(b, "EG1", "MF1");
      b.disconnect();
    }
  }

  private static void displayMessageFlowRunState(
                                 BrokerProxy b,
                                 String egName,
                                 String flowName) {
    try {
      ExecutionGroupProxy eg =
        b.getExecutionGroupByName(egName);

      if (eg != null) {
        MessageFlowProxy mf =
          eg.getMessageFlowByName(flowName);

        if (mf != null) {
          boolean isRunning = mf.isRunning();
```

```
          System.out.print("Flow "+flowName+" on " +
            egName+" on "+b.getName()+" is ");

          if (isRunning) {
            System.out.println("running");
          } else {
            System.out.println("stopped");
          }
        } else {
          System.err.println("No such flow "+flowName);
        }
      } else {
        System.err.println("No such exegrp "+egName+"!");
      }

    } catch(ConfigManagerProxyPropertyNotInitializedException
                                                    ex) {
      System.err.println("Comms problem! "+ex);
    }
  }
}
```

The method displayMessageFlowRunState() does most of the work. This method takes the valid `BrokerProxy` handle gained previously, and discovers the run-state of the message flow in the following way:

1. The `BrokerProxy` instance is used to gain a handle to the `ExecutionGroupProxy` object with the name described by the string `egName`
2. If a valid execution group is returned, the `ExecutionGroupProxy` instance is used to gain a handle to the `MessageFlowProxy` object with the name described by the string `flowName`.
3. If a valid message flow is returned, the run-state of the `MessageFlowProxy` object is queried, and the result is displayed.

The application does not have to know the names of objects that it can manipulate. Each administered object contains methods to return sets of objects that it logically owns. The following example demonstrates this technique by looking up the names of all execution groups within the broker.

```
import java.util.Enumeration;
import com.ibm.broker.config.proxy.*;

public class DisplayExecutionGroupNames {

  public static void main(String[] args) {

    BrokerProxy b = null;
    try {
      BrokerConnectionParameters bcp =
         new MQBrokerConnectionParameters(
           "localhost",
           1414,
           "");
      b = BrokerProxy.getInstance(bcp);
    } catch (ConfigManagerProxyException cmpex) {
      System.out.println("Error connecting: "+cmpex);
    }

    if (b != null) {
      System.out.println("Connected to broker!");
      displayExecutionGroupNames(b);
      b.disconnect();
    }
  }
```

```
  private static void displayExecutionGroupNames(BrokerProxy b)
  {
    try {
      Enumeration<ExecutionGroupProxy> allEGs = b.getExecutionGroups(null);

      while (allEGs.hasMoreElements()) {
        ExecutionGroupProxy thisEG =
          allEGs.nextElement();
        System.out.println("Found EG: "+thisEG.getName());
      }
    } catch(ConfigManagerProxyPropertyNotInitializedException
                                                        ex) {
        System.err.println("Comms problem! "+ex);
    }
  }
}
```

The key method is BrokerProxy.getExecutionGroups(Properties). When supplied with a null argument, this method returns an enumeration of all the `ExecutionGroupProxy` objects in the broker. The application uses this method to look at each `ExecutionGroupProxy` in turn, and display its name.

The Properties argument of the method BrokerProxy.getExecutionGroups(Properties) can be used to exactly specify the characteristics of the execution groups that are sought. The application can use this argument for nearly all the methods that return administered objects, and is a powerful way of filtering those objects with which the application needs to work.

Examples of the characteristics that can be used to filter object lookups are the run-state and short description, as well as more obvious properties such as the name and UUID. To write logic to achieve filtered lookups, you must understand how each administered object stores its information.

The properties of each administered object are stored locally inside the object by using a hash table, where each property is represented as a {`key, value`} tuple. Each key is the name of an attribute (for example, `name`) and each value is the value (for example, `BROKER1`).

Each key name must be expressed by using a constant from the `AttributeConstants` class (`com.ibm.broker.config.proxy`). A complete set of keys and possible values for each administered object is described in the Java documentation for the `AttributesConstant` class, or by using the `Show raw property table for this object` function in the CMP API Exerciser sample application. The latter displays the complete list of {`key, value`} pairs for each administered object.

The Properties argument that is supplied to the lookup methods is a set of those {`key, value`} pairs that must exist in each administered object in the returned enumeration. Consider the following code fragment:

```
Properties p = new Properties();

p.setProperty(AttributeConstants.OBJECT_RUNSTATE_PROPERTY,
              AttributeConstants.OBJECT_RUNSTATE_RUNNING);

Enumeration<MessageFlowProxy> mf = executionGroup.getMessageFlows(p);
```

Providing that the variable `executionGroup` is a valid `ExecutionGroupProxy` object, the returned enumeration contains only active message flows (that is, OBJECT_RUN_STATE_PROPERTY equal to OBJECT_RUNSTATE_RUNNING).

When property filtering is applied to a method that returns a single administered object rather than an enumeration of objects, only the first result is returned (which is non-deterministic if more than one match applies). Therefore the following code:

```
Properties p = new Properties();
p.setProperty(AttributeConstants.NAME_PROPERTY,
              "EG1");
ExecutionGroupProxy eg1 = brokerProxy.getExecutionGroup(p);
```

is an alternative to the following statement:

```
ExecutionGroupProxy eg1 = brokerProxy.getTopicByName("EG1");
```

If multiple {key, value} pairs are added to a property filter, all properties must be present in the child object for an object to match. If you want a method to perform a logical OR, or a logical NOT, on a filter, you must write specific application code for this purpose.

When AdministeredObjects are first instantiated in an application, the CMP API asks the broker for the current set of properties for that object. This action happens asynchronously, therefore the first time a property is requested, the CMP API might pause while it waits for the information to be supplied by the broker. If the information does not arrive within a certain time (for example, if the broker is not running), a ConfigManagerProxyPropertyNotInitializedException is thrown. Your application can control the maximum time that the CMP API waits by using the BrokerProxy.setRetryCharacteristics() method.

Related concepts:

"The Administration API for WebSphere Message Broker" on page 54
The Administration API for WebSphere Message Broker is a programming interface that your applications can use to control brokers and their resources through a remote interface.

Related tasks:

"Configuring an environment for developing and running CMP applications" on page 969
Prepare the environment in which you want to run your CMP applications.

"Connecting to a broker from a CMP application" on page 975
Connect an application that uses the CMP API to a broker, to send requests about its status and its resources.

Related information:

Administration API for WebSphere Message Broker (CMP API)

Deploying resources to a broker from a CMP application

Deploy BAR files to the brokers in your broker network from a CMP application.

About this task

You can also check the result of a deployment by using the CMP.

Example

An example

This example connects to a broker that is running on the local computer. Its queue manager, MB7QMGR, is listening on port 2414. The code deploys a BAR file called MyBAR.bar to an execution group called default that is running on the broker, and displays the result.

```
import com.ibm.broker.config.proxy.*;
public class DeployBAR {

  public static void main(String[] args) {
    BrokerConnectionParameters bcp =
       new MQBrokerConnectionParameters("localhost", 2414, "MB7QMGR");
    try {
      BrokerProxy b = BrokerProxy.getInstance(bcp);
      ExecutionGroupProxy eg = b.getExecutionGroupByName("default");
      DeployResult dr = eg.deploy("MyBAR.bar", true, 30000);
      System.out.println("Result = "+dr.getCompletionCode());
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

CMP API Exerciser

About this task

You can also use the CMP API Exerciser to deploy files and check the results. For example:

Procedure

1. Connect to the broker by clicking **File** > **Connect to Broker**. This action opens the Connect to Broker dialog.
2. Enter the relevant connection parameters in the dialog. A hierarchical representation of the broker and its resources is displayed.
3. Complete one or more of the following operations:
   - Click an object in the tree to display the attributes of that object.
   - Right-click an object in the tree to call CMP API methods that manipulate that object. For example, right-clicking a broker opens a menu that includes the items **Start user trace**.
   - Use the log pane at the bottom of the screen to view useful information that relates to the operation in progress.

Related concepts:

"Packaging and deployment overview" on page 3210
Deployment is the process of transferring data to an execution group on a broker so that it can take effect in the broker. Message flows and associated resources are packaged in broker archive (BAR) files for deployment.

"The Administration API for WebSphere Message Broker" on page 54
The Administration API for WebSphere Message Broker is a programming interface that your applications can use to control brokers and their resources through a remote interface.

Related tasks:

Chapter 11, "Packaging and deploying," on page 3209
Package resources that you create in the WebSphere Message Broker Toolkit, such as message flows, and deploy them to execution groups on brokers.

"Checking the results of deployment in a CMP application" on page 984
When you deploy from a CMP application, you can check the results of that action.

"Developing applications that use the Administration API" on page 956
Develop Java applications that use the Administration API (also known as the CMP API) to communicate with, deploy to, and manage brokers and their associated resources.

"Administration API (CMP) trace" on page 3554
Enable or disable service trace for the Administration API for WebSphere Message Broker (also known as the CMP API).

"Resolving problems when developing Administration API applications" on page 3510
Use the advice given here to help you to resolve problems that can arise when developing Administration API for WebSphere Message Broker (also known as the CMP API) applications.

Related information:

Administration API for WebSphere Message Broker (CMP API)

Checking the results of deployment in a CMP application

When you deploy from a CMP application, you can check the results of that action.

About this task

Code your application to test the results of the deploy actions that it takes. You can use code like the following snippet:

```
DeployResult dr = eg.deploy("MyBAR.bar", true, 30000);
System.out.println("Overall result = "+dr.getCompletionCode());

// Display log messages
Enumeration logEntries = dr.getLogEntries();
while (logEntries.hasMoreElements()) {
  LogEntry le = (LogEntry)logEntries.nextElement();
  System.out.println("General message: " + le.getDetail());
}
```

The deploy method blocks other processes until the broker has responded to the deployment request. When the method returns, the `DeployResult` object represents the outcome of the deployment at the time when the method returned; the object is not updated by the CMP API.

If the deployment message could not be sent to the broker, a `ConfigManagerProxyLoggedException` exception is thrown at the time of the deployment. If the broker receives the deployment message, log messages for the overall deployment are displayed, followed by completion codes specific to each broker that is affected by the deployment. The completion code, shown in the following table, is one of the static instances from the `CompletionCodeType` class.

Completion code	Description
pending	The deployment is held in a batch and is not sent until you call BrokerProxy.sendUpdates().
submitted	The deploy message was sent to the broker, but no response was received before the timeout period expired.
success	The broker has successfully completed the deployment.
failure	The broker has generated one or more errors during deployment. You can call the getLogEntries() method of the `DeployResult` class to get more information about the deployment failure. This method returns an enumeration of available `LogEntry` objects.

Related concepts:

"Packaging and deployment overview" on page 3210
Deployment is the process of transferring data to an execution group on a broker so that it can take effect in the broker. Message flows and associated resources are

packaged in broker archive (BAR) files for deployment.

"The Administration API for WebSphere Message Broker" on page 54
The Administration API for WebSphere Message Broker is a programming interface that your applications can use to control brokers and their resources through a remote interface.

Related tasks:

Chapter 11, "Packaging and deploying," on page 3209
Package resources that you create in the WebSphere Message Broker Toolkit, such as message flows, and deploy them to execution groups on brokers.

"Viewing Administration log information" on page 1007
You can view Administration log information by using either the WebSphere Message Broker Explorer, or the CMP API.

"Administration API (CMP) trace" on page 3554
Enable or disable service trace for the Administration API for WebSphere Message Broker (also known as the CMP API).

"Resolving problems when deploying message flows or message sets" on page 3436
Use the advice given here to help you to resolve common problems that can arise when you deploy message flows or message sets.

"Resolving problems when developing Administration API applications" on page 3510
Use the advice given here to help you to resolve problems that can arise when developing Administration API for WebSphere Message Broker (also known as the CMP API) applications.

Related information:

Administration API for WebSphere Message Broker (CMP API)

Setting message flow user-defined properties at run time in a CMP application

Use the CMP API to query, discover, and set message flow user-defined properties dynamically at run time. You can use the CMP API to set properties with a data type of character.

Before you begin

For user-defined properties on a message flow to be discoverable, the message flow must comply with the following conditions:

- The message flow must contain at least one of the following nodes:
  - JavaCompute
  - Compute
  - Database
  - Filter
  - PHPCompute
- The message flow must define the relevant user-defined property and provide an override value.

About this task

Tip: Use meaningful names and values for the properties that you define, so that you can understand their purpose and intent quickly. For example, a user-defined property named `property01`, with an initial value of `valueA` is not as useful as a property named `RouteToAorB` with an initial value of `RouteA`.

To query, discover, and set user-defined properties on a message flow, use the CMP API to issue the following calls. For details about the calls, including the syntax and parameters to use, see the CMP API documentation ("Administration API" on page 3672).

Procedure

1. Call MessageFlowProxy.getUserDefinedPropertyNames() to retrieve a list of all the user-defined properties that were defined by the Message Flow editor on the message flow or subflows.

   A string array is returned that contains the property names.
2. Call MessageFlowProxy.getUserDefinedProperty() to retrieve the value of the specified user-defined property.

   The value of the property is returned as a Java.lang.String value.
3. Call MessageFlowProxy.setUserDefinedProperty() to set a new value for the specified user-defined property.

   The property must exist. You cannot change the data type of the existing user-defined property (Java.lang.String); therefore, you must ensure that the new value complies with the existing data type. The value that you set with the MessageFlowProxy.setUserDefinedProperty() call is populated to all relevant nodes in the message flow, including nodes in subflows.

Related concepts:

"User-defined properties" on page 1147
A user-defined property (UDP) is a property that is defined when you construct a message flow by using the Message Flow editor. This property can be used by the ESQL or Java program inside message flow nodes, such as a Compute node.

"User-defined properties in ESQL" on page 2376
Access user-defined properties (UDPs) as variables in your ESQL program by specifying the EXTERNAL keyword on a DECLARE statement. For example, the ESQL statement `DECLARE today EXTERNAL CHARACTER 'monday'` defines a user-defined property called today with an initial value monday.

"The Administration API for WebSphere Message Broker" on page 54
The Administration API for WebSphere Message Broker is a programming interface that your applications can use to control brokers and their resources through a remote interface.

"ESQL variables" on page 2374
An ESQL variable is a data field that is used to help process a message.

Related tasks:

"Developing applications that use the Administration API" on page 956
Develop Java applications that use the Administration API (also known as the CMP API) to communicate with, deploy to, and manage brokers and their associated resources.

"Configuring a message flow at deployment time with user-defined properties" on page 2626
Use user-defined properties (UDPs) to configure message flows at deployment and run time, without modifying program code. You can give a UDP an initial value when you declare it in your program, or when you use the Message Flow editor to create or modify a message flow.

"Accessing message flow user-defined properties from a JavaCompute node" on page 2659
Customize a JavaCompute node to access properties that you have associated with the message flow in which the node is included.

Related reference:

"Administration API" on page 3672
Use the Administration API for WebSphere Message Broker (CMP API) Java classes and methods to develop CMP applications.

"Configurable message flow properties" on page 4020
When you add a message flow to a broker archive (BAR) file in preparation for deploying it to a broker, you can set additional properties that influence its run time operation. These properties are available for review and update when you select the **Manage and Configure** tab for the broker archive file.

Working with properties of a configurable service of type UserDefined at run time in a JavaCompute node

Use the CMP API in a JavaCompute node to query, set, create, and delete properties dynamically at run time in configurable services that you have defined with type UserDefined.

Before you begin

Before you start:

Complete the following tasks.

- Read the concept topic, "Configurable services" on page 1296.
- "Creating a message flow" on page 1431.

About this task

If you have created a UserDefined configurable service, and created properties for that service, you can work with those properties in a JavaCompute node. For example, you can create a UserDefined configurable service to set timeouts for processing HTTP messages.

You can create and delete configurable services in JavaCompute nodes, in the WebSphere Message Broker Explorer, and by using the **mqsicreateconfigurableservice** and **mqsideleteconfigurableservice** commands.

Procedure

1. Right-click the JavaCompute node and click **Open Java** to create and open a Java file in the Editor view, or open an existing file.
2. Create the Java class for the node in which you want to include CMP methods.
3. Add the CMP JAR file *install_dir*/classes/ConfigManagerProxy.jar to the Java build path for the associated Java project.
4. Import com.ibm.broker.config.proxy.* in your code.
5. Add the following static method to the class you have created:

   ```
   BrokerProxy b = BrokerProxy.getLocalInstance();
   ```

   This method returns an instance of the BrokerProxy object for the broker to which the message flow (that contains this node) is deployed.
6. To ensure that the BrokerProxy object has been populated with data from the broker before you access the configurable service, add the following code:

   ```
   while(!b.hasBeenPopulatedByBroker()) { Thread.sleep(100); }
   ```
7. Access the appropriate UserDefined configurable service:
   a. If you know the name of the configurable service, use the following code to access it:

      ```
      ConfigurableService myUDCS = b.getConfigurableService("UserDefined", "UD1");
      ```

b. If you want to select from a set of UserDefined configurable services, use the following code to get a list of all services of a particular type:

```
ConfigurableService[] UD_set = b.getConfigurableServices("UserDefined");
```

8. Add further code to access and use the specific properties that you are interested in. For example:
   - Retrieve the properties that are defined to that service:

     ```
     String[] props = myUDCS.getProperties();
     ```

   - Create a new property:

     ```
     String newprop = 'VerifyRequestTimeout';
     String newval = '15';
     myUDCS.setProperty(newprop, newval);
     ```

   - Delete a property:

     ```
     myUDCS.deleteProperty(newprop);
     ```

     You can also use the deleteProperties() method to delete more than one property.

     You can delete properties in UserDefined configurable services only. If you use this method on a configurable service of a different type, a `ConfigManagerProxyLoggedException` is generated.

9. Deploy the JAR file, and all associated message flows, in a BAR file. You do not have to deploy the `ConfigManagerProxy.jar` file to the target execution group, because the broker can access these classes independently.

Related concepts:

"The Administration API for WebSphere Message Broker" on page 54
The Administration API for WebSphere Message Broker is a programming interface that your applications can use to control brokers and their resources through a remote interface.

"Using the WebSphere Message Broker Explorer to work with configurable services" on page 644
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view, add, modify and delete configurable services.

"Configurable services" on page 1296
Configurable services are typically runtime properties. You can use them to define properties that are related to external services on which the broker relies; for example, an SMTP server or a JMS provider.

Related tasks:

"Developing applications that use the Administration API" on page 956
Develop Java applications that use the Administration API (also known as the CMP API) to communicate with, deploy to, and manage brokers and their associated resources.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

"Using timeouts with HTTP and SOAP nodes" on page 1595
Connect the HTTP Timeout terminal of the HTTPInput or SOAPInput nodes to further nodes to process timeouts.

Related reference:

"Administration API" on page 3672
Use the Administration API for WebSphere Message Broker (CMP API) Java classes and methods to develop CMP applications.

"JavaCompute node" on page 4514
Use the JavaCompute node to work with messages by using the Java language.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

Managing brokers in a CMP application

Manage the brokers and their resources from a CMP application.

Before you begin

Before you start

Before you start this task, you must have completed the task "Connecting to a broker from a CMP application" on page 975.

About this task

Use the CMP API to change the state of objects in the domain; you can create, delete, modify, and deploy objects stored within it. The following example sets the long description field of the broker:

```
import com.ibm.broker.config.proxy.*;

public class SetLongDescription {

  public static void main(String[] args) {

    BrokerProxy b = null;
    try {
      BrokerConnectionParameters bcp =
         new MQBrokerConnectionParameters(
           "localhost",
           1414,
           "");
      b = BrokerProxy.getInstance(bcp);
      b.setLongDescription("this is my broker");
      b.disconnect();
    } catch (ConfigManagerProxyException cmpex) {
      System.out.println("Error connecting: "+cmpex);
    }
  }
}
```

The broker processes requests to change properties from the CMP asynchronously, therefore if your application calls getLongDescription() immediately following the call to setLongDescription(), the response might return the old value of the property. For more information, see "Checking the results of broker management with the most recent completion code in a CMP application" on page 992.

What to do next

Next:

Most state-changing CMP methods return control immediately without informing the calling application of the outcome of the request. To discover this information, see “Checking the results of broker management in a CMP application” on page 991.

Related concepts:

“The Administration API for WebSphere Message Broker” on page 54
The Administration API for WebSphere Message Broker is a programming interface that your applications can use to control brokers and their resources through a remote interface.

Related tasks:

“Configuring an environment for developing and running CMP applications” on page 969
Prepare the environment in which you want to run your CMP applications.

“Connecting to a broker from a CMP application” on page 975
Connect an application that uses the CMP API to a broker, to send requests about its status and its resources.

“Managing brokers from JavaCompute nodes” on page 997
You can use the CMP API to manage brokers and their associated resources from JavaCompute nodes in deployed message flows.

Related information:

Administration API for WebSphere Message Broker (CMP API)

Creating objects in a CMP application

Create new objects associated with a broker.

About this task

The following example adds an execution group called `EG2` to the connected broker.

```
import com.ibm.broker.config.proxy.*;

public class AddExecutionGroup {

  public static void main(String[] args) {

    BrokerProxy b = null;
    try {
      BrokerConnectionParameters bcp =
         new MQBrokerConnectionParameters(
           "localhost",
           1414,
           "");
      b = BrokerProxy.getInstance(bcp);
      ExecutionGroupProxy e = b.createExecutionGroup("EG2");
      b.disconnect();
    } catch (ConfigManagerProxyException cmpex) {
      System.out.println("Error connecting: "+cmpex);
    }
  }
}
```

Because requests are processed asynchronously by the broker, the `ExecutionGroupProxy` object that is returned from the createExecutionGroup() method is a skeleton object when it is returned to your application, because it refers to an object that might not yet exist in the broker. The application can manipulate the object as if it existed on the broker, although the actual creation of the underlying object might not happen for some time.

If the request to create the object described by the skeleton fails, all requests that use the skeleton also fail. Therefore, if execution group EG2 cannot be created, all subsequent requests that concern the skeleton object fail. However, unless the application explicitly checks for errors, it works in the same way as it does in the successful case, because no exception is thrown unless, because of a communication problem, a message cannot be sent to the broker.

See "Checking the results of broker management in a CMP application" for further information about how to detect problems such as these.

Related concepts:

"The Administration API for WebSphere Message Broker" on page 54
The Administration API for WebSphere Message Broker is a programming interface that your applications can use to control brokers and their resources through a remote interface.

Related tasks:

"Configuring an environment for developing and running CMP applications" on page 969
Prepare the environment in which you want to run your CMP applications.

"Checking the results of broker management in a CMP application"
When you have made a change to the broker, use one of the supplied methods to check if the change has been successful.

Related reference:

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

Related information:

Administration API for WebSphere Message Broker (CMP API)

Checking the results of broker management in a CMP application

When you have made a change to the broker, use one of the supplied methods to check if the change has been successful.

About this task

Choose one of three methods in the CMP API to determine the outcome of requests to create, delete, modify, and deploy resources:

- If you have initiated a deployment method, you can use the return code from the deployment API; this technique is shown in "Checking the results of deployment in a CMP application" on page 984.
- You can query an object's most recent completion code; this option is shown in "Checking the results of broker management with the most recent completion code in a CMP application" on page 992.
- You can use the administered object notification mechanism; by using this approach, you can code specific routines to handle the responses and take appropriate action, and improve the efficiency of your program. See "Checking the results of broker management with object notification in a CMP application" on page 993.
- If you prefer to make property changes synchronously, call the method BrokerProxy.setSynchronous() before making your changes.

  If subsequent property change methods give a successful return, the request has been successfully completed by the broker. If the changes are rejected by the broker, or time out (they cannot be completed successfully by the broker before

the timeout period expires), the methods throw either a `ConfigManagerProxyRequestFailure` or a `ConfigManagerProxyRequestTimeout` exception.

For more information on synchronous property changes, see the description of the BrokerProxy.setSynchronous() method in the CMP API Javadoc information.

Related concepts:

"The Administration API for WebSphere Message Broker" on page 54
The Administration API for WebSphere Message Broker is a programming interface that your applications can use to control brokers and their resources through a remote interface.

Related tasks:

"Configuring an environment for developing and running CMP applications" on page 969
Prepare the environment in which you want to run your CMP applications.

"Checking the results of broker management with the most recent completion code in a CMP application"
Use an object's most recent completion code to determine the outcome of a request that your application made against that object.

"Checking the results of broker management with object notification in a CMP application" on page 993
Use object notification to determine the outcome of a request that your application made against the object.

Related information:

Administration API for WebSphere Message Broker (CMP API)

Checking the results of broker management with the most recent completion code in a CMP application:

Use an object's most recent completion code to determine the outcome of a request that your application made against that object.

About this task

Most state-changing methods in the CMP API do not provide return code that indicates the success or failure of a specific action. For these methods, you must write different code to discover the outcome of the action. Assuming that administered objects are not shared across threads, the following code fragment can be used to discover the outcome of a request to modify a broker's `LongDescription`, where `b` is an instance of a `BrokerProxy` object:

```
GregorianCalendar oldCCTime =
                         b.getTimeOfLastCompletionCode();
b.setLongDescription(newDesc);
GregorianCalendar newCCTime = oldCCTime;
while ((newCCTime == null) || (newCCTime.equals(oldCCTime))) {
  newCCTime = b.getTimeOfLastCompletionCode());
  Thread.sleep(1000);
}
CompletionCodeType ccType = b.getLastCompletionCode();
if (ccType == CompletionCodeType.success) {
  // etc.
}
```

In this example, the application initially determines when an action on the broker was last completed, using the getTimeOfLastCompletionCode() method. This method returns the time that the topology last received a completion code or, if no

return codes have been received, a null value. The application updates the broker's `LongDescription`, then continually monitors the topology, waiting for the results of the setLongDescription() command to be returned to the CMP. When the results are returned, control breaks out of the `while` loop and the last completion code is determined.

As well as being unsuitable for a multi-threaded application, this algorithm for determining the outcome of commands is inefficient, because it causes the CMP application to wait while the broker processes the request.

For a more efficient application, and one that is suitable for a multi-threaded environment, code the alternative approach that uses administered object notifications; see "Checking the results of broker management with object notification in a CMP application."

If you prefer, you can make property changes synchronously by using the methodBrokerProxy.setSynchronous() method. When you make synchronous property changes, methods such as setLongDescription() do not return until the change has been processed by the broker. For more information on synchronous property changes, see the description of the BrokerProxy.setSynchronous()) method in the CMP API Javadoc information.

Related concepts:

"The Administration API for WebSphere Message Broker" on page 54
The Administration API for WebSphere Message Broker is a programming interface that your applications can use to control brokers and their resources through a remote interface.

Related tasks:

"Checking the results of broker management in a CMP application" on page 991
When you have made a change to the broker, use one of the supplied methods to check if the change has been successful.

"Checking the results of broker management with object notification in a CMP application"
Use object notification to determine the outcome of a request that your application made against the object.

Related information:

Administration API for WebSphere Message Broker (CMP API)

Checking the results of broker management with object notification in a CMP application:

Use object notification to determine the outcome of a request that your application made against the object.

About this task

The CMP can notify applications whenever commands complete, or whenever changes occur to administered objects. By making use of the `OBSERVER` design pattern, your CMP application can define a handle to a user-supplied object that has a specific method that is called if an object is modified or deleted, or whenever a response to a previously submitted action is returned from the broker.

The user-supplied code must implement the `AdministeredObjectListener` interface. This interface defines methods that are invoked by the CMP when an event occurs on an administered object to which the listener is registered. The following methods are defined:

processModify(...)

processModify(...) is called when the administered object to which the listener is registered has one or more of its attributes modified by the broker. The following information is included in the notification through the use of the processModify() method arguments:

1. A handle to the `AdministeredObject` to which the notification refers.
2. A list of strings that contain the key names that have been changed.
3. A list of strings that describe new subcomponents that have been created for the object; for example, new execution groups in a broker.
4. A list of strings that describe subcomponents that have been removed from the object.

The format of the strings passed to the final two parameters is an internal representation of the administered object. You can turn this representation into an administered object type by using the getManagedSubcomponentFromStringRepresentation() method.

Consider the following additional information:

1. Strings are passed within these lists to enhance performance; the CMP does not use resource instantiating administered objects, unless they are specifically requested by the calling application.
2. The first time you call the processModify() method for a listener, the `changed attributes` parameter can include a complete set of attribute names for the object, if the application is using a batch method, or if the CMP is experiencing communication problems with the broker.

processDelete(...)

processDelete(...) is called if the object with which the listener is registered is completely removed from the broker. Supplied to `processDelete(...)` is one parameter – a handle to the administered object that has been deleted; when this method returns, the administered object handle might no longer be valid. At about the same time that a processDelete(...) event occurs, a processModify(...) event is sent to listeners of the deleted object's parent, to announce a change in the parent's list of subcomponents.

processActionResponse(...)

processActionResponse(...) is the event that informs the application that a previous action submitted by that application is complete. Only one processActionResponse(...) event is received for each state-changing operation issued by the CMP application. This event contains the following items of information:

1. A handle to the administered object for which a request was submitted.
2. The completion code of the request.
3. A set of zero, or more, informational (BIP) messages associated with the result.
4. A set of (`key, value`) pairs that describes the submitted request in more detail. Check the documentation for information about how to parse these pairs.

To register a listener, each administered object has a registerListener() method that is used to tell the CMP to call the supplied code whenever an event occurs on that object. You can register the same `AdministeredObjectListener` for notifications from multiple administered objects. You can also register multiple `AdministeredObjectListeners` against the same administered object.

The following example demonstrates this technique by registering a listener on the broker object, and displaying a message whenever it is modified:

```
import com.ibm.broker.config.proxy.*;
import com.ibm.broker.config.proxy.CompletionCodeType;
import java.util.List;
import java.util.ListIterator;
import java.util.Properties;

public class MonitorBroker implements AdministeredObjectListener {

  public static void main(String[] args) {

    BrokerProxy b = null;
    try {
      BrokerConnectionParameters bcp =
         new MQBrokerConnectionParameters(
           "localhost",
           1414,
           "");
      b = BrokerProxy.getInstance(bcp);
    } catch (ConfigManagerProxyException cmpex) {
      System.out.println("Error connecting: "+cmpex);
    }

    if (b != null) {
      System.out.println("Connected to broker");
      listenForChanges(b);
      b.disconnect();
    }
  }

  private static void listenForChanges(AdministeredObject obj)
  {
    if (obj != null) {
      obj.registerListener(new MonitorBroker());
      while(true) {
        // thread could do something else here instead
        try {
          Thread.sleep(10000);
        } catch (InterruptedException ex) {
          // ignore
        }
      }
    }
  }

  public void processActionResponse(AdministeredObject obj,
                                    CompletionCodeType cc,
                                    List bipMessages,
                                    Properties refProperties) {
    // Event ignored in this example
  }

  public void processDelete(AdministeredObject deletedObject) {
         // Event ignored in this example
  }

  public void processModify(AdministeredObject affectedObject,
                            List changedAttributes,
```

```
                            List newChildren,
                            List removedChildren) {
    try {
      System.out.println(affectedObject+" has changed:");
      ListIterator e = changedAttributes.listIterator();
      while (e.hasNext()) {
        String changedAttribute = (String) e.next();
        System.out.println("Changed: "+changedAttribute);
      }
      ListIterator e2 = newChildren.listIterator();
      while (e2.hasNext()) {
        String newChildStr = (String) e2.next();
        AdministeredObject newChild =
          affectedObject.getManagedSubcomponentFromStringRepresentation(newChildStr);
        System.out.println("New child: "+newChild);
      }
      ListIterator e3 = removedChildren.listIterator();
      while (e3.hasNext()) {
        String remChildStr = (String) e3.next();
        AdministeredObject removedChild =
          affectedObject.getManagedSubcomponentFromStringRepresentation(remChildStr);
        System.out.println("Removed child: "+removedChild);
      }
    } catch (ConfigManagerProxyPropertyNotInitializedException ex) {
      ex.printStackTrace();
    }
  }
}
```

The listenForChanges() method attempts to register an instance of the `MonitorBroker` class for notifications of broker changes. If successful, the main thread pauses indefinitely to prevent the application from ending when the method returns. When the listener is registered, whenever the broker changes (for example, if an execution group is added), the processModify() method is called. This method displays details of each notification on the screen.

You must register separate listeners for each administered object on which you want to receive notifications. You can use the same listener instance for multiple administered objects.

You can stop receiving notifications in three ways:

- AdministeredObject.deregisterListener(AdministeredObjectListener)
- ConfigManagerProxy.deregisterListeners()
- ConfigManagerProxy.disconnect()

The first method de-registers a single listener from a single administered object; the other two methods deregister all listeners connected with that `BrokerProxy` instance. In addition, the final method shows that all listeners are implicitly removed when connection to the broker is stopped.

You can also implement the AdvancedAdministeredObjectListener interface which, when registered, yields additional information to applications.

Related concepts:

"The Administration API for WebSphere Message Broker" on page 54
The Administration API for WebSphere Message Broker is a programming interface that your applications can use to control brokers and their resources through a remote interface.

Related tasks:

"Checking the results of broker management in a CMP application" on page 991
When you have made a change to the broker, use one of the supplied methods to check if the change has been successful.

"Checking the results of broker management with the most recent completion code in a CMP application" on page 992
Use an object's most recent completion code to determine the outcome of a request that your application made against that object.

Related information:

Administration API for WebSphere Message Broker (CMP API)

Managing brokers from JavaCompute nodes

You can use the CMP API to manage brokers and their associated resources from JavaCompute nodes in deployed message flows.

Before you begin

Before you start

Before starting this step, you must create a "JavaCompute node" on page 4514 in a message flow.

About this task

Use CMP methods and classes in your JavaCompute node to explore and manage brokers and other resources.

Procedure

1. Create the Java class for the node in which you want to include CMP methods.
2. Add the CMP JAR file *install_dir*/classes/ConfigManagerProxy.jar to the Java build path for the associated Java project.
3. Add the following static method to the class:

   ```
   BrokerProxy thisBroker = BrokerProxy.getLocalInstance();
   ```

   This method returns an instance of the BrokerProxy object for the broker to which the message flow (that contains this node) is deployed.
4. To work with an execution group on this broker, add the following static method to your code:

   ```
   ExecutionGroupProxy thisEG = ExecutionGroupProxy.getLocalInstance();
   ```

   This method returns an instance of the ExecutionGroupProxy object for the execution group to which the message flow is deployed.
5. If you want to connect to a different broker that you have created on the computer to which your node and message flow are deployed, you can use a variant of this class:

   ```
   BrokerProxy secondBroker = BrokerProxy.getLocalInstance(string)
   ```

   Specify the name of the alternative local broker as the value of the variable *string*. Your code can manage this second broker, and its associated resources, by using the BrokerProxy object that is returned by this call.
6. Include additional CMP methods in your Java code to run the operations that you want against the broker or execution group by using the objects obtained in previous steps. You can follow the guidance that is provided in other topics in this section for further information and examples that show how to use CMP methods in CMP applications.

If you include methods that affect the message flow in which your CMP application is running, it might not be able to receive all notifications that these operations have successfully completed. Stopping, deleting, and redeploying the message flow are examples in this category; consider carefully the consequences of using these methods.

7. Deploy the JAR file, and all associated message flows, in a BAR file. You do not have to deploy the `ConfigManagerProxy.jar` file to the target execution group, because the broker can access these classes independently.

Related concepts:

"The Administration API for WebSphere Message Broker" on page 54
The Administration API for WebSphere Message Broker is a programming interface that your applications can use to control brokers and their resources through a remote interface.

Related tasks:

"Configuring an environment for developing and running CMP applications" on page 969
Prepare the environment in which you want to run your CMP applications.

"Creating Java code for a JavaCompute node" on page 2629
Use these instructions to associate Java code with your JavaCompute node.

Related reference:

"JavaCompute node" on page 4514
Use the JavaCompute node to work with messages by using the Java language.

Related information:

Administration API for WebSphere Message Broker (CMP API)

Working with resource statistics in a CMP application

Start, stop, and review status of resource statistics collection in your CMP applications.

Before you begin

Before you start:

- Read the concept topic about resource statistics.

About this task

You can create CMP applications to examine and control the collection of resource statistics.

Checking what resource types can return statistics

```
////////////////////////////////////////////////////
// Sample CMP API code that connects to a local broker
// called 'testbrk' and writes out available
// resource types on the broker that have the
// ability to emit resource-level statistics.
BrokerProxy b = null;
try {
    b = BrokerProxy.getLocalInstance("testbrk");
 String[] resourceNames = b.getResourceTypeNames();
 for (String thisResource : resourceNames) {
  System.out.println(thisResource);
 }
} catch (ConfigManagerProxyLoggedException e) {
```

```
 e.printStackTrace();
} catch (ConfigManagerProxyPropertyNotInitializedException e) {
 e.printStackTrace();
}
```

Checking for resource names associated with a specific resource type

```
//////////////////////////////////////////////////////
// Sample CMP API code that connects to a local broker
// called 'testbrk' and writes out resource property
// names reported for a specific resource type.
BrokerProxy b = null;
try {
    b = BrokerProxy.getLocalInstance("testbrk");
 String[] resourcePropertyNames =
    b.getResourceTypeStatisticsPropertyNames("JVM");
 for (String thisResourceProperty : resourcePropertyNames) {
  System.out.println(thisResourceProperty);
 }
} catch (ConfigManagerProxyLoggedException e) {
 e.printStackTrace();
} catch (ConfigManagerProxyPropertyNotInitializedException e) {
 e.printStackTrace();
}
```

Starting statistics collection

```
//////////////////////////////////////////////////////
// Sample CMP API code that connects to a local broker
// called 'testbrk' and gets a reference to the execution
// group called 'default'. It then enables resource
// statistics for all the execution group's resource types.
BrokerProxy b = null;
try {
    b = BrokerProxy.getLocalInstance("testbrk");
 ExecutionGroupProxy eg = b.getExecutionGroupByName("default");
 if (eg != null) {
  eg.setResourceStatisticsEnabled(null, true);
 }
} catch (ConfigManagerProxyLoggedException e) {
 e.printStackTrace();
} catch (ConfigManagerProxyPropertyNotInitializedException e) {
 e.printStackTrace();
}
```

Stopping statistics collection

```
//////////////////////////////////////////////////////
// Sample CMP API code that connects to a local broker
// called 'testbrk' and gets a reference to the execution
// group called 'default'. It then disables resource
// statistics for all the execution group's resource types.
BrokerProxy b = null;
try {
    b = BrokerProxy.getLocalInstance("testbrk");
 ExecutionGroupProxy eg = b.getExecutionGroupByName("default");
 if (eg != null) {
  eg.setResourceStatisticsEnabled(null, false);
 }
} catch (ConfigManagerProxyLoggedException e) {
 e.printStackTrace();
} catch (ConfigManagerProxyPropertyNotInitializedException e) {
 e.printStackTrace();
}
```

Viewing statistics collection status

```
//////////////////////////////////////////////////////
// Sample CMP API code that connects to a local broker
// called 'testbrk' and gets a reference to the execution
// group called 'default'. It then writes out if resource
// statistics is enabled.
```

```
BrokerProxy b = null;
try {
    b = BrokerProxy.getLocalInstance("testbrk");
 ExecutionGroupProxy eg = b.getExecutionGroupByName("default");
 if (eg != null) {
  System.out.println(eg.getResourceStatisticsEnabled(null));
 }
} catch (ConfigManagerProxyLoggedException e) {
 e.printStackTrace();
} catch (ConfigManagerProxyPropertyNotInitializedException e) {
 e.printStackTrace();
}
```

Related concepts:

"Resource statistics" on page 3306
Resource statistics are collected by a broker to record performance and operating details of resources that are used by execution groups.

Related tasks:

"Developing applications that use the Administration API" on page 956
Develop Java applications that use the Administration API (also known as the CMP API) to communicate with, deploy to, and manage brokers and their associated resources.

"Monitoring resource performance" on page 3305
You can collect statistics to assess the performance of certain resources used by execution groups.

Related reference:

"Resource statistics data" on page 6745
Learn about the measurements for which data is returned when you activate resource statistics collection.

Submitting batch requests from a CMP application

Use the CMP API to group multiple requests that are destined for the same broker, and submit them as a single unit of work.

About this task

To start a batch, your application must call the beginUpdates() method on the `BrokerProxy` handle. The CMP API delays submitting any state-changing requests to the broker until it is told a batch of requests is ready to be sent.

The sendUpdates() method tells the CMP API to submit as a batch all requests received since the last beginUpdates() call. The clearUpdates() method can be used to discard a batch without submitting it to the broker. The application can check if a batch is currently in progress by using the isBatching() method. Only one batch for a CMP API handle can be in progress at any one time.

One advantage of using a batch method is that it provides an assurance that no other applications can have messages processed by the broker during the batch. When a broker receives a batch of requests, it processes each request in the batch in the order it was added to the batch (FIFO), and requests from no other CMP application are processed until the entire batch is completed.

Consider the following sequence of commands:

```
ExecutionGroupProxy e = b.createExecutionGroup("EG2");
e.deploy("mybar.bar");
```

Without using a batch method, the application cannot guarantee the success of these actions. For example, even if each command would otherwise succeed, a second (possibly remote) application might delete the execution group EG2 after it has been created by the first application, but before the other command is processed.

If the sequence is extended to use a batch method, the broker is now guaranteed to process all the commands together, therefore no other application can disrupt the logic intended by the application.

```
b.startUpdates();
ExecutionGroupProxy e = b.createExecutionGroup("EG2");
e.deploy("mybar.bar");
b.sendUpdates();
```

Another advantage of using a batch method is performance. The CMP typically sends one WebSphere MQ message to the broker for each request.

In a situation that requires lots of requests to be sent in quick succession, the use of a batch has a significant effect on performance, reducing both time taken to process the requests, and the memory used. For example, your application might create a number of execution groups on a single broker. Each batch of requests is sent in a single WebSphere MQ message, therefore reducing the processing that is required for each method.

Batch mode does not provide transactional (commit and backout) capability; some requests in a batch might succeed and others fail. If the broker processes a request in a batch that fails, it continues to process the next request in the batch until it has attempted all requests in the batch.

Related concepts:

"The Administration API for WebSphere Message Broker" on page 54
The Administration API for WebSphere Message Broker is a programming interface that your applications can use to control brokers and their resources through a remote interface.

Related tasks:

"Configuring an environment for developing and running CMP applications" on page 969
Prepare the environment in which you want to run your CMP applications.

"Connecting to a broker from a CMP application" on page 975
Connect an application that uses the CMP API to a broker, to send requests about its status and its resources.

Related information:

Administration API for WebSphere Message Broker (CMP API)

Managing resources used by brokers

Manage the resources used by brokers.

About this task

- "Listing database connections that the broker holds" on page 1002
- "Quiescing a database" on page 1002
- "Using a JDBC connection pool to manage database resources used by an execution group" on page 1003

Related tasks:

"Monitoring resource performance" on page 3305
You can collect statistics to assess the performance of certain resources used by execution groups.

Listing database connections that the broker holds

The broker does not provide an interface that you can use to list the connections that it has to a database. You must use the facilities of the database suppliers to list connections.

For further information about how to list database connections, refer to the documentation that is provided by your database vendor.

Related concepts:

"User database connections" on page 2110
User databases contain business data that is written and accessed by deployed message flows. You must create connections from the broker to the user database by using ODBC or JDBC.

Related tasks:

"Working with databases" on page 2109
Create and configure databases to use with your message flow applications.

Related reference:

"Supported databases" on page 3591
You can optionally configure databases to contain data that is accessed by your message flows. Databases from IBM and other suppliers are supported at specific versions on supported operating systems.

"Quiescing a database"
The broker and database exhibit specific behaviors when you quiesce the database.

"WebSphere MQ connections" on page 4222
The number of WebSphere MQ connections a broker requires to its queue manager depends on the actions of the message flows that access the WebSphere MQ resource.

Related information:

DB2 V9.1 Information Center (distributed systems)

DB2 V9.5 Information Center (distributed systems)

DB2 Information Center (z/OS)

Quiescing a database

The broker and database exhibit specific behaviors when you quiesce the database.

If you access databases from one or more message flows, your database administrator might occasionally want to issue the quiesce instruction on a database. This action is a function of the database, not of the broker.

The following three assumptions are made for the database that you are quiescing:

- The database can be quiesced (not all databases support this function).
- New connections to the database are blocked by the database when it is quiescing.
- Message flows that access the database eventually become idle.

The following list shows the behavior that is expected while a database is quiescing:

- Run the command that quiesces the database. When this command starts, the connections that are in use remain in use, but no new connections to the database are allowed.
- Messages that are being processed by message flows, which use existing connections to the database, continue to use their connections until the connections become idle. Therefore if messages continue to be received by the message flow, it might be a long time before the quiesce occurs. To ensure that messages are no longer processed, stop the message flow. Stopping the message flow stops messages being processed, and releases the database connections that the flow was using. This action ensures that the database connections that the flow holds become idle.
- Database connections for the message flow become idle. This situation causes the broker to release the connections to the user databases that the message flow is using. When all connections to the database from the broker, and from any other applications that are using the database, are released, the database can complete its quiesce function.

For more information, see "User database connections" on page 2110.

Related concepts:

"User database connections" on page 2110
User databases contain business data that is written and accessed by deployed message flows. You must create connections from the broker to the user database by using ODBC or JDBC.

Related tasks:

"Enabling ODBC connections to the databases" on page 668
Set up the resources and environment that the broker requires for Open Database Connectivity (ODBC) connections to user databases on distributed systems.

"Enabling JDBC connections to the databases" on page 683
Configure connections to a user database through a JDBCProvider service.

"Authorizing access to user databases" on page 662
When you have created a user database, you must authorize the broker and its execution groups to access it.

Related reference:

"Supported databases" on page 3591
You can optionally configure databases to contain data that is accessed by your message flows. Databases from IBM and other suppliers are supported at specific versions on supported operating systems.

"Listing database connections that the broker holds" on page 1002
The broker does not provide an interface that you can use to list the connections that it has to a database. You must use the facilities of the database suppliers to list connections.

"WebSphere MQ connections" on page 4222
The number of WebSphere MQ connections a broker requires to its queue manager depends on the actions of the message flows that access the WebSphere MQ resource.

Using a JDBC connection pool to manage database resources used by an execution group

Use broker JDBC provider resources to configure the use of thread pools independently of message flow and input node thread pools.

About this task

WebSphere Message Broker manages JDBC connections in the following ways:

- Non-pooled connections:
  - WebSphere Message Broker creates a JDBC connection on demand for each message flow instance that requires one.
  - Each JDBC connection is associated with the message flow instance for which it was created. This association is maintained until the connection is closed.
  - Each JDBC connection that is idle for 60 seconds is closed, and is no longer associated with a message flow instance.
  - After a JDBC connection that was associated with a message flow instance is closed, if the same message flow instance requires a JDBC connection, WebSphere Message Broker creates a new JDBC connection on demand.
- Pooled connections:
  - When a message flow instance requires a JDBC connection, WebSphere Message Broker assigns an unused connection from the pool.
  - If all pooled JDBC connections are being used, and the maximum pool size has not been reached, WebSphere Message Broker creates a new pooled JDBC connection. The maximum pool size is specified in the `maxConnectionPoolSize` property of the "JDBCProviders configurable service" on page 3778.
  - Each pooled JDBC connection remains associated with a message flow instance only for the processing of one input message.
  - When a message flow instance completes the processing of an input message, the association with a JDBC connection is removed, and the JDBC connection is returned to the pool.
  - Each pooled JDBC connection that is idle for 15 minutes is closed, and is removed from the pool.
  - Pooled JDBC connections are not applicable to the DatabaseRetrieve and DatabaseRoute nodes.

Using a JDBC connection pool enable you to scale database access independently of the number of message flow threads.

You can create a JDBC connection pool by setting the `maxConnectionPoolSize` property of the "JDBCProviders configurable service" on page 3778 to a non-zero integer value. The `maxConnectionPoolSize` property acts at the execution group level to specify the maximum number of JDBC connection threads that can be used. A value of zero defaults to the standard WebSphere Message Broker Version 7.0 behavior, where one JDBC connection is created for each message flow thread.

All message flows within an execution group that use the same `JDBCProviders` configurable service also share a connection pool. You can monitor the behavior of a JDBC connection pool by using broker resource statistics

The `maxConnectionPoolSize` property is applicable to JDBC connections obtained using the `getJDBCType4Connection()` API of the JavaCompute node.

Note: The `maxConnectionPoolSize` property does not apply to the JDBC connections used by the DatabaseRetrieve or DatabaseRoute nodes.

Related tasks:

"Viewing resource statistics data in the WebSphere Message Broker Explorer" on page 3313
Use the WebSphere Message Broker Explorer to view resource statistics data for your execution groups in the Broker Resources and Broker Resources Graph views.

"Viewing the status of resource statistics collection in the WebSphere Message Broker Explorer" on page 3317
Use the WebSphere Message Broker Explorer to view the status of resource statistics collection in the Broker Resources view.

"Managing resources used by brokers" on page 1001
Manage the resources used by brokers.

Related reference:

"Resource statistics data" on page 6745
Learn about the measurements for which data is returned when you activate resource statistics collection.

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

Administering Java applications

Manage the Java applications that are deployed to a broker.

About this task

You can deploy message flows that contain Java applications to a broker. The Java code is run within a JVM that is created by the execution group. The JVM runs in the same process as other broker components such as message parsing and nodes that are not Java-based.

Administration of Java applications includes the following tasks:

- "Tuning JVM parameters"
- "Configuring classloaders for Java user-defined nodes"
- "Configuring classloaders for JavaCompute nodes" on page 1006

Tuning JVM parameters

About this task

Use the **mqsichangeproperties** command to tune the JVM parameters to ensure that there are sufficient resources for all the Java applications that are deployed to the execution group. For more details, see "**mqsichangeproperties** command" on page 3756 and "JVM parameter values" on page 3813.

Configuring classloaders for Java user-defined nodes

About this task

Java user-defined nodes are manually installed onto a broker as either a PAR file or a JAR file. Because PAR and JAR files are only loaded by the broker on startup, the broker must be restarted. For more details, see "Packaging a Java user-defined node" on page 3118.

A PAR file is given its own Java classloader, ensuring that the node classes are isolated from any other node classes.

For more details, see “User-defined node class loading” on page 3120.

Configuring classloaders for JavaCompute nodes

About this task

A JavaCompute node is deployed to an execution group as part of a BAR file. A JavaCompute node can specify a JavaClassLoader configurable service to be used by the node. A JavaClassLoader configurable service defines the behavior of the classloaders that are used by the node. For more details, see “JavaCompute node classloading” on page 2635.

If a JavaCompute node specifies a JavaClassloader configurable service, you must define a configurable service with the name specified by the node on the broker. For more details, see “JavaCompute node classloading using a configurable service” on page 2636.

Related concepts:

“The broker environment” on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

“JavaCompute node classloading” on page 2635
Details the default Java classloader options and the precedence order of each type.

“JavaCompute node classloading using a configurable service” on page 2636
Details alternative configurable Java classloader options and the precedence order of each type.

“User-defined node class loading” on page 3120
Details the Java classes packaging options and loading order precedence for user-defined nodes.

Related tasks:

Chapter 8, “Administering brokers and broker resources,” on page 899
Administering brokers and associated broker resources includes the tasks that you perform frequently to activate and manage those resources. Choose the method you prefer to administer your brokers and associated resources.

Related reference:

“JavaCompute node” on page 4514
Use the JavaCompute node to work with messages by using the Java language.

Accessing Administration log information

Active brokers record information about their operations in the Administration log. Access the events that are written by the broker by using the WebSphere Message Broker Explorer and the WebSphere Message Broker Toolkit. For some actions, you can also use the Administration API (also known as the CMP API).

About this task

- “Viewing Administration log information” on page 1007
- “Saving Administration log information” on page 1008
- “Clearing Administration log information” on page 1009
- “Changing Administration Log view preferences” on page 1010

Related concepts:

“The broker environment” on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related reference:

"Administration Log view" on page 6840
The Administration Log view shows administration requests and their results, changes made to objects, and the result of deployment actions on brokers. The Administration Log view can also describe configuration changes that have been automatically applied as a result of moving the broker from one version to another; for example, after applying maintenance.

Related information:

Administration API for WebSphere Message Broker (CMP API)

Viewing Administration log information

You can view Administration log information by using either the WebSphere Message Broker Explorer, or the CMP API.

About this task

Follow the instructions in this topic to use the WebSphere Message Broker Explorer. If you prefer to use the CMP API, see "Developing applications that use the Administration API" on page 956 and "Administration API" on page 3672.

Administration log information is written to the Administration Log view in the WebSphere Message Broker Explorer. The Administration Log view displays actions performed on the broker by all users. Alternatively, you can view the results of deployment actions from a single user using the Deployment Log view in the WebSphere Message Broker Toolkit, see "Deployment Log view" on page 6797.

The Administration Log view contains information about events that occur within your brokers. These events can be information, errors, or warnings and relate to your own actions. To view events for a particular broker, look for the name of the broker in the Source column.

Each event contains the following information:

- Message: The event number.
- Source: Where the event has come from.
- TimeStamp: The date and time that the event occurred. Time stamps are taken from the computer that is hosting the broker.
- Details: What has caused the event and what action is needed to rectify it.

You can only view Administration log information for a specific broker using the WebSphere Message Broker Explorer if the broker is both running and connected. Local brokers are automatically connected, but you must manually connect to remote brokers.

Procedure

1. To view Administration log information in the WebSphere Message Broker Explorer, ensure that the Administration Log view is visible. If the Administration Log view is not visible, from the menu, click **Window** > **Show View** > **Administration Log**.
2. Expand the Brokers folder, and select the broker with which you want to work. The Administration log information for the selected broker is displayed in the Administration Log view. The Administration log information is displayed in time and date order.

3. Double-click the message to display the full details for a message. The message is opened in a new window.

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

"The Administration API for WebSphere Message Broker" on page 54
The Administration API for WebSphere Message Broker is a programming interface that your applications can use to control brokers and their resources through a remote interface.

Related tasks:

"Clearing Administration log information" on page 1009
Clear Administration log information to reduce the size of the log by using either the WebSphere Message Broker Explorer or the CMP API.

"Saving Administration log information"
Save the Administration log information that is written to the Administration Log view in the WebSphere Message Broker Explorer.

Related reference:

"Administration Log view" on page 6840
The Administration Log view shows administration requests and their results, changes made to objects, and the result of deployment actions on brokers. The Administration Log view can also describe configuration changes that have been automatically applied as a result of moving the broker from one version to another; for example, after applying maintenance.

Saving Administration log information

Save the Administration log information that is written to the Administration Log view in the WebSphere Message Broker Explorer.

About this task

Administration log information is deleted automatically from the broker when the broker restarts. You can save the log contents to file if you want to retain them. You cannot save the Administration log information from the Deployment Log view in the WebSphere Message Broker Toolkit.

Procedure

1. In the WebSphere Message Broker Explorer, expand the Brokers folder in the Navigator view.
2. Select the broker with which you want to work, to display the Administration log information in the Administration Log view.
3. Right-click in the Administration Log view, and click **Save Log As**.
4. Enter an appropriate directory in which to save the log information.
5. Enter a name for the log file, and click **Save Log**.

Results

Each message recorded in the Administration log is written to the text file with the same information that is detailed in the Administration log itself.

To view the saved log, open the log file in an appropriate text editor.

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related tasks:

"Viewing Administration log information" on page 1007
You can view Administration log information by using either the WebSphere Message Broker Explorer, or the CMP API.

"Clearing Administration log information"
Clear Administration log information to reduce the size of the log by using either the WebSphere Message Broker Explorer or the CMP API.

Related reference:

"Administration Log view" on page 6840
The Administration Log view shows administration requests and their results, changes made to objects, and the result of deployment actions on brokers. The Administration Log view can also describe configuration changes that have been automatically applied as a result of moving the broker from one version to another; for example, after applying maintenance.

Clearing Administration log information

Clear Administration log information to reduce the size of the log by using either the WebSphere Message Broker Explorer or the CMP API.

About this task

Follow the instructions in this topic to use the WebSphere Message Broker Explorer to clear the log. If you prefer to use the CMP API, see "Developing applications that use the Administration API" on page 956 and "Administration API" on page 3672. You can empty the contents of the Deployment Log view in the WebSphere Message Broker Toolkit by right-clicking in the Deployment Log view, and clicking **Clean Log**. Emptying the contents of the Deployment Log view does not remove the entries from the Administration log.

To clear all the Administration log information from the Administration log:

Procedure

1. Expand the Brokers folder in the Navigator view.
2. Select the broker with which you want to work, to display the Administration log information in the Administration Log view.
3. Right-click in the Administration Log view, and click **Clear Administration Log**.

   If you have set the Broker Explorer preference to *warn before deleting log events*, a prompt asks you to confirm deletion. Click **OK**.

   If you have not set the Broker Explorer preference to *warn before deleting log events*, the Administration log is cleared automatically.

   All log entries will be deleted for this user.

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related tasks:

"Viewing Administration log information" on page 1007
You can view Administration log information by using either the WebSphere Message Broker Explorer, or the CMP API.

"Saving Administration log information" on page 1008
Save the Administration log information that is written to the Administration Log view in the WebSphere Message Broker Explorer.

Related reference:

"Administration Log view" on page 6840
The Administration Log view shows administration requests and their results, changes made to objects, and the result of deployment actions on brokers. The Administration Log view can also describe configuration changes that have been automatically applied as a result of moving the broker from one version to another; for example, after applying maintenance.

Related information:

Administration API for WebSphere Message Broker (CMP API)

Changing Administration Log view preferences

You can change preferences for the Administration Log view by using the Broker Explorer preferences page in the WebSphere Message Broker Explorer.

About this task

You can choose not to display a warning before deleting log events. The default is to display a warning. You can also choose how long to wait for responses from the broker after a deployment.

To change preferences:

Procedure

1. Click **Window>Preferences**.
2. Expand the **Broker Explorer** category in the left pane.
3. Make your selections.
4. Click **OK**.

Related tasks:

"Accessing Administration log information" on page 1006
Active brokers record information about their operations in the Administration log. Access the events that are written by the broker by using the WebSphere Message Broker Explorer and the WebSphere Message Broker Toolkit. For some actions, you can also use the Administration API (also known as the CMP API).

Related reference:

"Administration Log view" on page 6840
The Administration Log view shows administration requests and their results, changes made to objects, and the result of deployment actions on brokers. The Administration Log view can also describe configuration changes that have been automatically applied as a result of moving the broker from one version to another; for example, after applying maintenance.

Changing the location of the work path

The work path directory is the location where a component stores internal data, such as installation logs, component details, and trace output. The shared-classes directory is also located in the work path directory and is used for deployed Java code. If the work path directory does not have enough capacity, redirect the directory to another file system that has enough capacity.

About this task

The work path is fixed at installation time so that WebSphere Message Broker can always find the information that it needs, and always knows where to store new information.

If you need to change the location (for example, if you do not have enough capacity on the automatically-designated file system), do not change the path to the directory; instead, redirect the old work path directory to a new location.

Changing the location of the work path on Windows systems

About this task

When you change the location of the work path, you mount the new partition at the location of the old work path directory.

To change the location of the work path on Windows:

Procedure

1. Shut down all WebSphere Message Broker services and processes.
2. Create a new partition on the system. The new partition can be on the same drive as the old work path, or on a different drive.
3. Locate the work path directory for your installation on the local system by running the following command:
   ```
   echo %MQSI_WORKPATH%
   ```
4. Copy the contents of the work path directory to the new partition.
5. Delete the contents of the old work path directory.
6. Open the Computer Management dialog: click **Start** > **Settings** > **Control Panel** > **Administrative Tools** > **Computer Management**; the Computer Management dialog opens.
7. In the left pane of the Computer Management dialog, click **Disk Management**. The new partition that you added, and any existing partitions, are listed in the right pane.
8. Right-click the new partition, then click **Change Drive Letter and Paths**. The Change Drive Letter and Paths dialog opens.
9. Click **Add**. The Add Drive Letter or Path dialog opens.
10. Ensure that **Mount in the following empty NTFS folder** is selected, then browse to the old work path location.

11. Click **OK**, then click **OK** again.

Results

Any files that WebSphere Message Broker creates in the work path location are stored on the new partition.

Changing the location of the work path on Linux and UNIX systems

About this task

When you change the location of the work path, you can either mount the new partition at the location of the old work path directory, or you can replace the old work path directory with a soft link that points to the new work path directory.

To change the location of the work path on UNIX and Linux:

Procedure

1. Shut down all WebSphere Message Broker services and processes.
2. Create a new directory on a suitable file system.
3. Locate the work path directory for your installation on the local system by running the following command:
   ```
   echo $MQSI_WORKPATH
   ```
4. Copy the contents of the work path directory to the new partition.
5. Delete the contents of the old work path directory.
6. Perform one of the following tasks so that the WebSphere Message Broker installation uses the new work path location:
   - Use the **mount** command to mount the new work path directory at the location of the old work path directory.
   - Delete the old work path directory and replace it with a soft link. Give the soft link the same name as the old work path directory and point the link to the new work path directory.

Results

Any files that WebSphere Message Broker creates in the work path location are stored in the new location.

Related tasks:

"Can you see all of your files and folders?" on page 3351
How to show all files in Windows Explorer:

Related reference:

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

Backing up resources

Back up your broker components, and the working files associated with brokers, the WebSphere Message Broker Explorer, and the WebSphere Message Broker Toolkit, so that you can restore these resources if required.

About this task

Back up your broker components regularly to ensure that you can return to a known operational state if necessary. In addition to configuration and operation state, the broker maintains additional resources in its work path, and you can request that these resources are also backed up.

Back up the workspaces and connection files that you have created in the WebSphere Message Broker Explorer.

Back up the workspaces you have created in the WebSphere Message Broker Toolkit; these resources contain your application development resources; for example message flows and message sets. If you use a development repository to store application resources, such as Rational ClearCase, see the documentation associated with that repository to check how you can back up this data.

The following topics tell you how to back up and restore brokers and the WebSphere Message Broker Toolkit workspace:

- "Backing up the broker"
- "Restoring the broker" on page 1015
- "Backing up the WebSphere Message Broker Explorer and WebSphere Message Broker Toolkit workspace" on page 1016

On distributed systems, you can use the backed up components to restore the broker only on an identical operating environment. The operating system must be at the same level, and the broker and queue manager names must be identical.

Related concepts:

"Development repository" on page 45
Use a development repository to benefit from features such as version control and access control of files, which make it easier for teams to work on shared resources.

Related tasks:

"Recovering after failure" on page 3574
Follow a set of procedures to recover after a serious problem.

Related reference:

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

Backing up the broker

Back up the broker configuration and all associated resources.

Before you begin

Before you start: Create the broker.

About this task

You can back up the broker and its resources to preserve the current state of the broker configuration. You can use the backup file that is created to restore a broker in an identical operating environment: the operating system must be at the same level, and the broker and queue manager names must be identical.

You can run this command for a broker that is active. However, you must not take a backup while the broker is processing configuration changes and deployments; the backup file created might contain incomplete information. If the file contains partial records, you cannot use it to restore the broker at a later time.

To ensure that the backup is complete and correct, take a backup either when the broker is not processing a configuration change (such as a deployment or change property) or when the broker is stopped.

Procedure

1. If you want to back up an active broker, check that no configuration change requests are in progress. For example, if you are changing broker properties, or have initiated a deployment, wait for these actions to complete before you back up the broker. Active message flows are unaffected by the backup process.

   If you prefer, you can stop the broker before you take a backup by using the **mqsistop** command.
2. Back up the broker. Specify the broker name and the location in which the backup file is created. You can also optionally specify the name of the backup file, and the name of a file to which a detailed trace is written.
   - Linux UNIX Windows Run the **mqsibackupbroker** command, specifying the broker name and the directory to which the backup file is written.

     For example, to back up a broker on Windows, enter the following command:

     ```
     mqsibackupbroker MB7BROKER -d c:\MQSI\BACKUP
     ```
   - z/OS Customize and submit the JCL member BIPBUBK.
3. When the command has completed successfully, you can continue to use the broker. If you stopped the broker, restart it by using the **mqsistart** command.

Results

The current broker configuration is saved in the backup file. Keep the file safe so that you can restore the broker at a later date if required.

Related tasks:

"Restoring the broker" on page 1015
Restore a broker configuration that you backed up previously.

"Customizing the broker JCL" on page 625
This subtask is part of the larger task of creating a broker on z/OS.

Related reference:

"**mqsibackupbroker** command" on page 3720
Use the **mqsibackupbroker** command to back up the current configuration of a broker.

"**mqsirestorebroker** command" on page 3952
Use the **mqsirestorebroker** command to restore the broker configuration from a

backup file.

"Sample BIPBUBK file" on page 4004
The sample BIPBUBK file that is shipped with WebSphere Message Broker is included here for your reference.

"Sample BIPRSBK file" on page 4012
The sample BIPRSBK file that is shipped with WebSphere Message Broker is included here for your reference.

"**mqsistart** command" on page 3965
Use the **mqsistart** command to start the specified broker if all initial verification tests complete successfully.

"**mqsistop** command" on page 3972
Use the **mqsistop** command to stop the specified component.

Restoring the broker

Restore a broker configuration that you backed up previously.

Before you begin

Before you start: Back up the broker.

About this task

You can restore a broker on a computer that has an identical configuration by using the backup file that you created. The operating system must be at the same level, and the broker and queue manager names must be identical.

Procedure

1. If you have deleted the broker and it no longer exists, or if you are restoring it on a different computer, create it by using the **mqsicreatebroker** command. Use the same name and parameters that you used for the broker that you backed up, including the name of the queue manager.
2. If the broker is running, stop it by using the **mqsistop** command. If you intend to restore common configuration information from other brokers that you have configured on this computer, you must also stop all the brokers that share this common information. For example, you can restore profile information from common files.
3. Restore the broker. Specify the broker name and the name and location of the backup file. If you want to restore common configuration information, or if you want a trace of the actions taken, specify the appropriate parameters for your platform.
   - Linux UNIX Windows Run the **mqsirestorebroker** command.

     For example, to restore a broker on Windows, enter the following command:

     ```
     mqsirestorebroker WBRK_BROKER -d c:\MQSI\BACKUP -a mybroker.zip
     ```
   - z/OS Customize and submit the JCL member BIPRSBK.
4. When the command has completed successfully, start the broker by using the **mqsistart** command.

What to do next

Next: The broker configuration has been restored; you can continue your work with this broker.

Related tasks:

"Backing up the broker" on page 1013
Back up the broker configuration and all associated resources.

"Customizing the broker JCL" on page 625
This subtask is part of the larger task of creating a broker on z/OS.

Related reference:

"**mqsibackupbroker** command" on page 3720
Use the **mqsibackupbroker** command to back up the current configuration of a broker.

"**mqsirestorebroker** command" on page 3952
Use the **mqsirestorebroker** command to restore the broker configuration from a backup file.

"Sample BIPBUBK file" on page 4004
The sample `BIPBUBK` file that is shipped with WebSphere Message Broker is included here for your reference.

"Sample BIPRSBK file" on page 4012
The sample `BIPRSBK` file that is shipped with WebSphere Message Broker is included here for your reference.

"**mqsistart** command" on page 3965
Use the **mqsistart** command to start the specified broker if all initial verification tests complete successfully.

"**mqsistop** command" on page 3972
Use the **mqsistop** command to stop the specified component.

Backing up the WebSphere Message Broker Explorer and WebSphere Message Broker Toolkit workspace

The WebSphere Message Broker Explorer and WebSphere Message Broker Toolkit workspaces contain your personal settings and data, such as message flow and message set resources. You can have multiple workspaces in different locations, and you can also have references to projects that are in other locations, therefore consider all these locations when you back up your resources.

About this task

The default workspace directory for the WebSphere Message Broker Explorer depends on the platform on which it is running:

- Windows On Windows XP and Windows Server 2003: `C:\Documents and Settings\<userid>\Application Data\IBM\MQ Explorer\<server project>\`.
- Windows On Windows Vista and Windows Server 2008, the default workspace directory is created at `C:\Users\`*`user_ID`*`\IBM\MQ Explorer\<server project>\`.
- Linux On Linux, the default workspace directory is created at `/home/`*`user_ID`*`/IBM/MQ Explorer\/<server project>/`.

where *user_ID* is the user name with which you are logged on. Back up files in these locations, and in all other locations in which you have saved workspace files.

The default workspace directory for the WebSphere Message Broker Toolkit depends on the platform on which it is running:

- Windows On Windows XP and Windows Server 2003: `C:\Documents and Settings\`*`user_ID`*`\IBM\wmbt70\workspace`.
- Windows On Windows Vista and Windows Server 2008: `C:\Users\`*`user_ID`*`\IBM\wmbt70\workspace`.

- Linux On Linux: `/home/user_ID/IBM/wmqi70/workspace`.

where *user_ID* is the user name with which you are logged on. Back up files in these locations, and in all other locations in which you have saved workspace files.

The WebSphere Message Broker Toolkit workspace directory contains a directory called `.metadata`, which contains your personal settings and preferences for the WebSphere Message Broker Toolkit. If the `.metadata` directory gets corrupted, you lose these settings, and the WebSphere Message Broker Toolkit reverts to the default layout and preferences. If you have not backed up the `.metadata` directory, you must manually set all preferences again, and import all projects, such as message flow projects, that were displayed in the Broker Development view. To back up the `.metadata` directory, take a copy of the directory.

The WebSphere Message Broker Toolkit workspace also contains a directory for each project (for example, a message flow project) that you have created in the WebSphere Message Broker Toolkit. These directories contain your data, which you must back up.

Use one of the following methods to back up the data in your workspace; the instructions are the same for both the WebSphere Message Broker Explorer and the WebSphere Message Broker Toolkit.

Procedure

- Export your working projects. You can export the projects directly as a compressed file. For further information, see Exporting in the Eclipse Workbench User Guide.
- Copy the project directories from the workspace directory to another location.
- WebSphere Message Broker Explorer only: Export all `.broker` connection files to save the details of connections to all your brokers.
- WebSphere Message Broker Toolkit only: Take copies of all your BAR files to back up their contents. Include all the associated source files when you build your broker archive (BAR) files; you can then save all content by saving only the BAR file. To add resources to a BAR file that is ready for deployment, select **Include source files**, which adds the message flow and message set source files, and the compiled files.

What to do next

If you want to restore the resources, copy the directories back into your workspace directory and import the projects. For instructions, see Importing in the Eclipse Workbench User Guide.

Related concepts:

"Resources" on page 36
The projects, folders, and files that you work with in the WebSphere Message Broker Toolkit workspace are called *resources*. By default, these resources are stored with their metadata in the workspace directory in your local file system. The workspace directory is created the first time that you start the WebSphere Message Broker Toolkit.

Related tasks:

"Backing up resources" on page 1013
Back up your broker components, and the working files associated with brokers, the WebSphere Message Broker Explorer, and the WebSphere Message Broker Toolkit, so that you can restore these resources if required.

Related information:

Workbench User Guide - Customizing the Workbench

Chapter 9. Developing message flow applications

Develop message flows to process your business messages and data.

About this task

WebSphere Message Broker provides several ways in which you can develop the message flow applications that you need to support your business processes. Your client applications can use many different techniques to interact with a broker, and the message flows that you have deployed to it.

The following steps provide a typical route that you might take through this section of the information center to understand the concepts associated with message flows, develop your own, and establish the ways in which your client applications can use message flows. When you are familiar with the basic concepts and behavior, you can progress to more complex options by adding further resources for your message flows to use.

Procedure

1. If you are not familiar with WebSphere Message Broker, read "Processing messages" on page 1021. This section defines the concepts associated with message flows, and is essential reading whatever your planned use of the broker. It also covers the general processing performed by a broker on all messages, regardless of origin. For example, it explains how a broker handles errors in your message flows, and how you can set up coordinated transactions.
2. When you have reached an understanding of what a message flow is, and how it can process your messages and data, you can review the different ways in which you can create your own message flows. Choose from the following options:
   - "Developing message flow applications by using patterns" on page 1309. The supplied patterns provide reusable solutions that encapsulate a tested approach to solving a common architecture, design, or deployment task in a particular context. Much of the work of design and development has been done for you when you use a pattern.

     You can use these patterns unchanged, or modify them to meet your own requirements. You must create additional resources to complement the pattern and complete the solution.
   - "Developing message flow applications by using samples" on page 1406. The supplied samples provide tested message flows that focus on a particular feature or function supported by WebSphere Message Broker. They are more limited in scope than patterns, but provide comprehensive examples of typical message processing in particular scenarios. They are stand-alone applications; you can use them without having to create and configure additional resources.

     Because they are created to demonstrate a particular facet of the product, the samples are not always designed to use the preferred techniques for a particular task. Therefore, use them as examples to learn how particular functions work, not as complete production-level solutions. You might find them helpful as a starting point to developing your own message flows, or as part of a larger solution.

- "Developing message flow applications from a wizard" on page 1408. You can run one of several Quick Start wizards in the WebSphere Message Broker Toolkit. These wizards create message flows and associated resources that are dependent on particular requirements; for example, adapter connections. You can use the messages flows created by these wizards as the basis for other message flows that change or add to the original result.
- "Developing message flow applications from scratch" on page 1423. If you prefer, you can create a message flow by using the basic building blocks available in the Message Flow editor. You decide which nodes are appropriate, and how to connect them together, to provide the processing that your messages require.

3. Connect your business process applications and data to your message flows. You can use a number of different protocols to communicate with the broker. You can also interact from your message flows with other products and services. Find out what options are available in "Connecting client applications" on page 1537.
4. You can design your message flows to handle your messages and data in different ways. You can choose from a range of nodes that support:
   - "Routing messages" on page 2209
   - "Transforming and enriching messages" on page 2227
   - "Processing events" on page 2717
   - "Handling errors in message flows" on page 2823
5. Message flows can process self-defining messages, predefined messages, or both. Predefined messages might provide extra value in your environment, and offer additional processing options within your message flows, Learn about these benefits, and how you can define your own message models, in "Constructing message models" on page 2838.
6. Your message processing environment might need some special processing that is not provided by WebSphere Message Broker. You can explore further options in "Developing user-defined extensions" on page 2970, and learn about how to customize your broker in additional ways.

What to do next

When you develop your message flows, and when you run them in the broker, you can tune the way the message flows work to improve message flow performance. For further information, see "Message flow performance" on page 3260.

When you create a project, it is recommended to avoid using spaces in the project name. Although using spaces in the name is valid, you might encounter problems. For example: A scenario where an XML schema file located in project X references an XML schema file project Y, either through import or include statements. If the referenced schema in the project contain spaces in project name, the name does not resolve, and you receive errors

Related tasks:

Chapter 10, "Testing and debugging message flow applications," on page 3143
Use the flow debugger to track messages through your message flows and check for errors, or use the Test Client to test your message flows in a safe environment before they are used on a production system.

Chapter 11, "Packaging and deploying," on page 3209
Package resources that you create in the WebSphere Message Broker Toolkit, such as message flows, and deploy them to execution groups on brokers.

Processing messages

Process your business messages and data by interacting with a broker, which you can configure to provide services and to communicate with other applications and systems.

About this task

You can use WebSphere Message Broker to integrate client applications that use different protocols and message formats. A broker provides a flexible environment that you can configure to route and transform messages in many ways from one application to all other applications. In many cases, you can reuse existing applications with little or no change, to use the services provided by a broker.

A broker manages three sets of resources to integrate your applications, messages, and data:

- Message flows
- Nodes
- Message models

You can configure message flows to support your applications that use one or both of the supported communication models, point-to-point and publish/subscribe. You can connect your client applications to the message flows that are running in a broker by using one of several supported protocols.

This section introduces you to conceptual details of message flows, message models, and their associated resources. It also describes the default behavior taken by a broker when it runs message flows. Read this section to gain an overall understanding of applications, messages, and brokers, and understand the generic tasks that are independent of the protocol your applications use.

- "Message flows overview" on page 1022
- "Message modeling" on page 1154
- "Message flow behavior" on page 1277

Some of the examples used in these topics have characteristics that apply to specific environments; these characteristics are clearly stated where applicable. Exceptions to the default behavior of the broker for particular protocols, or message formats, are stated in the sections where they apply. Additional characteristics of specific protocols of messages are also described where they apply.

Related tasks:

"Developing message flow applications by using patterns" on page 1309
Create resources that are used to solve a specific business problem by using patterns.

"Developing message flow applications by using samples" on page 1406
Use the samples to learn more about the features that are available in WebSphere Message Broker, and how to use them.

"Developing message flow applications from a wizard" on page 1408
A Quick Start wizard sets up the basic resources that are required to develop a message flow application. The wizard sets up and gives names to containers for the resources in which you then develop your message flow.

"Developing message flow applications from scratch" on page 1423
Design, create, and configure message flows by using the WebSphere Message Broker Toolkit.

"Connecting client applications" on page 1537
Connect your client applications to the broker by using one or more of the supported protocols from other resources and software servers in your network.

"Routing messages" on page 2209
Route messages through your message flow or the broker by using one or more of the techniques described in this section.

"Transforming and enriching messages" on page 2227
Transform and enrich messages by using one or more of the techniques described in this section.

Message flows overview

A message flow is a sequence of processing steps that run in the broker when an input message is received.

You define a message flow in the WebSphere Message Broker Toolkit by including a number of message flow nodes, each of which represents a set of actions that define a processing step. The way in which you join the message flow nodes together determine which processing steps are carried out, in which order, and under which conditions. The path that you create between one node and another is known as a connection.

A message flow must include an input node that provides the source of the messages that are processed. You can process the message in one or more ways, and optionally deliver it through one or more output nodes; see "Connecting client applications" on page 1537. The message is received as a bit stream, and is converted by a parser into a tree structure that is used internally in the message flow. Before the message is delivered to a final destination, it is converted back into a bit stream. For more information about these conversions, see "Parsers" on page 1072 and "The message tree" on page 1041.

When you want to exchange messages between multiple applications, you might find that the applications do not understand or expect messages in the same format. You might must provide some processing between the sending and receiving applications that ensures that both can continue to work unchanged, but can exchange messages successfully. For more information about the options available, see "Transforming and enriching messages" on page 2227.

You define the processing that is required when you create and configure a message flow. You can include built-in nodes, nodes that are supplied by a vendor, nodes that you have created yourself (user-defined nodes), or other message flows (known as subflows).

The processing that you set up determines what actions are completed on a message when it is received, the order in which the actions are completed, and the final destinations of the message. All these actions manage the route that a message takes through a message flow; more information about these actions is provided in "Routing messages" on page 2209. To complete more complex processing involving more than one message, you can use the nodes described in "Processing events" on page 2717.

You can configure additional properties to make your message flow transactional, or multithreaded. You can also add error paths that ensure every message is handled in an appropriate way.

When you want to run a message flow to process messages, you deploy it to a broker, where it is run in an execution group.

The mode in which your broker is working can affect the number of execution groups and message flows that you can deploy, and the types of node that you can use. See "Restrictions that apply in each operation mode" on page 3657.

The following topics describe the concepts that you must understand to design, create, and configure a message flow and its associated resources:

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

"Execution groups" on page 53
An execution group is a named grouping of message flows that have been assigned to a broker. The broker enforces a degree of isolation between message flows in distinct execution groups by ensuring that they run in separate address spaces, or as unique processes.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"Message flow behavior" on page 1277
Message flow behavior is initially defined by the broker, but you can change or add to that behavior in some situations.

"Nodes for connectivity" on page 1028
WebSphere Message Broker supports direct connections from applications, and can send direct requests to other application endpoints. WebSphere Message Broker can also connect to various subsystems including WebSphere MQ, files, and databases, to read and write existing application data.

"Properties" on page 1143
You can view and change properties that define broker characteristics, and those properties of associated resources such as message flows.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"Message flows" on page 4015
Use the reference information in this section to develop your message flows and related resources.

Message flow nodes

A message flow node is a processing step in a message flow. It can be a built-in node, a user-defined node, or a subflow node.

A message flow node receives a message, performs a set of actions against the message, and optionally passes the original message, and none or more other messages, to the next node in the message flow.

A message flow node has a fixed number of input and output points known as terminals. You can make connections between the terminals to define the routes that a message can take through a message flow. Message flow nodes are displayed in the node palette that is associated with the Message Flow editor. The palette is arranged in categories, which group together nodes that provide related processing; for example, transformation.

Input nodes do not have input terminals. The message flow starts when a message is retrieved from an input device; for example, a WebSphere MQ queue. The message flow ends when none or more output messages have been sent by one or more output nodes, and control returns back to the input node. The input node either commits or rolls back the transaction. Input and output nodes can be protocol-specific, to interact with particular systems such as Web Services.

Most nodes are processing nodes, that you can include between your input and output nodes and connect together to define the flow of control. These nodes typically transform a message from one format to another, or route a message along a particular path, or provide more complex options such as aggregation or filtering.

You can configure a node by setting or changing the values for its properties. Some nodes have *mandatory properties*, for which you must set a value. Other properties must have a value, but are assigned a default value which you can leave unchanged. The remaining properties are *optional properties*; no value is required.

When you develop a message flow, the way in which you set the properties of the nodes in that flow influences the way in which the messages are processed by that flow. For example, by setting properties that define input and output WebSphere MQ queue names, you determine where the message flow receives the message from, and where it delivers the message.

You can also configure nodes by using *promoted properties*; promote one or more node properties to become properties of the message flow that contains those nodes. You can then change these properties at the flow level, rather than having to update one or more individual nodes. You can also promote equivalent properties from more than one node to the same message flow property; for example, you might use this technique to set, at the flow level, the name of the database that all the nodes in the message flow must connect to.

A subset of node properties is *configurable properties*; that is, you can change their values when you deploy the message flow to a broker for execution. You might find this ability useful if you deploy a message flow to more than one broker, and want it to behave in a slightly different way on each broker. For example, when you deploy the message flow to a test broker, you can set a configurable property to force the flow to interact with a test database. When you deploy the same message flow to a production broker, you can set the same property to the value of a production database, without having to update the message flow itself.

The mode that your broker is working in can affect the types of node that you can use; see "Restrictions that apply in each operation mode" on page 3657.

You can add nodes of three types into your message flows:

Built-in node
A built-in node is a message flow node that is supplied by WebSphere Message Broker. The built-in nodes provide input and output, manipulation and transformation, decision making, collating requests, and error handling and reporting functions.

For information about all of the built-in nodes supplied by WebSphere Message Broker, see "Built-in nodes" on page 4293.

For information about the nodes that you can use to connect WebSphere Message Broker to your applications, see "Nodes for connectivity" on page 1028.

User-defined node
A user-defined node is an extension to the broker that provides a new message flow node in addition to the nodes that are supplied with the product. A user-defined node must be written to the user-defined node API provided by WebSphere Message Broker for both C and Java languages. The following sample demonstrates how you can write your own nodes in both C and Java languages.

- User-defined Extension

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Subflow
A subflow is a directed graph that is composed of message flow nodes and connectors and is designed to be embedded in a message flow or in another subflow. To connect your subflow to other nodes in the main flow, you can add Input and Output nodes to the subflow. A subflow must include at least one Input node or one Output node. A subflow can be executed by a broker only as part of the message flow in which it is embedded, and therefore cannot be independently deployed.

A message is received by an Input node and processed according to the definition of the subflow. That might include being stored through a Warehouse node, or delivered to another message target, for example through an MQOutput node. If required, the message can be passed through an Output node back to the main flow for further processing.

The subflow, when it is embedded in a main flow, is represented by a subflow node, which has a unique icon. The icon is displayed with the correct number of terminals to represent the Input and Output nodes that you have included in the subflow definition.

The most common use of a subflow is to provide processing that is required in many places within a message flow, or is to be shared between several message flows. For example, you might code some error processing in a subflow, or create a subflow to provide an audit trail (storing the entire message and writing a trace entry).

For more information, see "Subflows" on page 1030.

A node does not always produce an output message for every output terminal: often it produces one output for a single terminal based on the message received or the result of the operation of the node. For example, a Filter node typically sends a message on either the True terminal or the False terminal, but not both.

If you have connected more than one terminal to another node, the processing in the node determines the order in which the message is propagated to the nodes that it is connected to; you cannot change this order. The node sends the output message on each terminal, but sends on the next terminal only when the processing has completed for the current terminal.

Updates to a message are never propagated to nodes which have been previously executed, only to nodes that follow the node in which the update has been made. The order in which the message is propagated to the different output terminals is determined by the broker; you cannot change this order. The only exception to this rule is the FlowOrder node, in which the terminals indicate the order in which the message is propagated to each.

All built-in nodes include error handling as part of their processing. If an error is detected within the node, the message is propagated to the failure terminal. What happens then depends on the structure of your message flow. You can use only the basic error handling provided by the broker, or you can enhance your flow by adding error processing nodes and flows to provide more comprehensive failure processing. For more information about these options, see “Handling errors in message flows” on page 2823.

The message flow can accept a new message for processing only when all paths through the message flow (that is, all connected nodes from all output terminals) have been completed, and control has returned to the input node which commits or rolls back the transaction.

The following sample uses environment variables in the XML_Reservation sample to store information that has been taken from a database table and to pass that information to a node downstream in the message flow.

- Airline Reservations

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Related concepts:

“Message flow projects” on page 1035
A message flow project is a specialized container in which you create and maintain all the resources associated with one or more message flows.

“Message flow node palette” on page 1027
The palette in the Message Flow editor contains all the built-in nodes, which are organized into categories, or drawers. A drawer is a container for a list of icons, such as the **Favorites** drawer.

“Message flow connections” on page 1032
A connection is an entity that connects an output terminal of one message flow node to an input terminal of another. The connection represents the flow of control and data between two message flow nodes.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Defining a promoted property" on page 1297
When you create a message flow, you can promote properties from individual nodes in that message flow to the message flow level. Properties promoted in this way override the property values that you have set for the individual nodes.

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"Message flow projects and files" on page 6822

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Related information:

Java user-defined extensions API

Message flow node palette:

The palette in the Message Flow editor contains all the built-in nodes, which are organized into categories, or drawers. A drawer is a container for a list of icons, such as the **Favorites** drawer.

When you first open the WebSphere Message Broker Toolkit, the default drawers contain built-in nodes that are related in function. For example, one drawer contains all the nodes that handle input and output to WebSphere MQ queues. Another drawer, **Transformation**, groups the nodes that you can use to convert the input message into a different form, including the Compute and JavaCompute nodes.

You can drag the nodes that you use most often into the **Favorites** drawer for easy access. If you create your own nodes, you can also add them to the palette. You can drag a node from the palette onto the canvas, and create a connection between two nodes. You can also use the palette to create an annotation on a message flow or node.

Right-click the palette to add a selected node to the canvas, or customize the appearance and behavior of the palette.

Use the Customize Palette dialog box to reorder node categories, set the drawer behavior for individual categories, and rename or hide nodes or categories.

You cannot move a category above the Favorites category. You can hide the Favorites category, but you cannot delete or rename it.

Use the Palette Settings dialog box to set the palette layout, determine the behavior of palette drawers, and choose a particular font.

The following topics explain how to change the palette layout and settings:

- "Changing the palette layout" on page 1490
- "Changing the palette settings" on page 1490

- "Customizing the palette" on page 1491

Related tasks:

"Adding nodes to the Favorites category on the palette" on page 1492
The nodes on the palette are organized in categories. The first category is Favorites, which is usually empty. You can drag the nodes that you use most often to the Favorites category.

"Adding a message flow node" on page 1494
When you have created a message flow, add nodes to define its function.

"Creating a user-defined node in the WebSphere Message Broker Toolkit" on page 3081
Create the representation of a user-defined node created in Java and C only, in the WebSphere Message Broker Toolkit.

"Adding annotations to a message flow or node" on page 1531
You can add annotations to a message flow, a node, or multiple nodes. You can use these annotations to record reminders, issues arising during the development of a message flow, or informal documentation to facilitate team development.

Related reference:

"Message Flow editor" on page 6810
The Message Flow editor is the default editor provided by the Broker Application Development perspective for defining a graphical representation of a message flow in the WebSphere Message Broker Toolkit, and for setting properties for individual message flow nodes.

"Plug-in Development projects and files" on page 6825

Nodes for connectivity:

WebSphere Message Broker supports direct connections from applications, and can send direct requests to other application endpoints. WebSphere Message Broker can also connect to various subsystems including WebSphere MQ, files, and databases, to read and write existing application data.

You can connect WebSphere Message Broker to your applications by adding the appropriate nodes to your message flow. The nodes you use can be tailored to support the protocols and subsystems that your applications already use. WebSphere Message Broker supplies nodes to support different protocols and subsystems; you can also create your own nodes to support additional protocols and subsystems if required.

- Nodes are supplied to support the protocols that are described in "Connecting client applications" on page 1537; for example, WebSphere MQ and HTTP.
- Nodes are supplied to support the subsystems listed in "External systems and resources" on page 61.

Within your message flow, you can include the following types of nodes to communicate with your applications. The icons for the nodes in each group described are based on a common appearance, which is shown with that group.

Input nodes

The input node reads data from a subsystem or input application, which might be in the form of a message, or a record (for example, from a file). The input node calls a parser to interpret the data and create an internal message tree structure. The node can split the input message into records if required. When the message is ready, the input node sends it to the rest of the message flow for processing.

Input nodes are represented by icons that conform to this template: 

Output nodes

An output node takes data from the message tree, calls a parser to serialize the tree into the appropriate message or record format, and writes out the message or record to one or more specified end applications or subsystems. If appropriate, you can configure your message flow to continue processing a message after it has generated one or more output messages through output nodes.

Output nodes are represented by icons that conform to this template:

Reply nodes

A reply node is a specialized form of an output node. Typically, the reply node is associated with an input node in the same flow, and uses context information from that input node to decide where to send the reply. Depending on the protocol of the input node, the context information might be created for you by that node; for other protocols, the context information might be contained within the message itself.

Reply nodes are represented by icons that conform to this template: 

Get and receive nodes

A get (receive) node reads extra data from a subsystem, and includes it in the current message tree, during message flow processing.

Get nodes are represented by icons that conform to this template: 

Request nodes

A request node writes a request to an external system, reads the response, and incorporates some or all that response data into the current message tree.

Request nodes are represented by icons that conform to this template:

Asynchronous request and response nodes

These two nodes are specialized form of a request node, in which you can generate a request and handle the response in a second message flow. Typically, you use these nodes when you are making a request that might take some time to complete. By using this technique, you can have several outstanding requests, without suspending flow processing.

Asynchronous request nodes are represented by icons that conform to this template: 

Asynchronous response nodes are represented by icons that conform to this template: 

You can connect applications that use different protocols by choosing an appropriate mix of input and output nodes. You must also include nodes that can transform the input message into the appropriate output format between the input and output nodes.

Because of the extent of the support provided by the nodes in WebSphere Message Broker, you can configure your brokers to act as clients to many different subsystems, and to interact with them based on the contents of the message tree created for each individual input message.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Processing messages" on page 1021
Process your business messages and data by interacting with a broker, which you can configure to provide services and to communicate with other applications and systems.

"Developing message flow applications from scratch" on page 1423
Design, create, and configure message flows by using the WebSphere Message Broker Toolkit.

"Connecting client applications" on page 1537
Connect your client applications to the broker by using one or more of the supported protocols from other resources and software servers in your network.

"Routing messages" on page 2209
Route messages through your message flow or the broker by using one or more of the techniques described in this section.

"Transforming and enriching messages" on page 2227
Transform and enrich messages by using one or more of the techniques described in this section.

Subflows

You can include subflows in your message flows in the same way as you include built-in or user-defined nodes. You can also connect subflows to other nodes in the same way.

You can define a subflow once, and use it in more than one message flow, and in more than one message flow project. Therefore, a subflow provides the following benefits:

- Reusability and reduced development time.
- Consistency and increased maintainability of your message flows (consider a subflow as analogous to a programming macro, or to inline code that is written once but used in many places).
- Flexibility to tailor a subflow to a specific context (for example, by updating the output queue or data source information).

However, remember that a subflow is not a single node, and its inclusion increases the number of nodes in the message flow, which might affect its performance.

Consider these examples of subflow use:

- You can define a subflow that provides a common sequence of actions that applies to several message flows if an error is encountered. For example, you might have a common error routine that writes the message to a database through the Warehouse node, and puts it to a queue for processing by an error recovery routine. The use of this routine in multiple message flows, or in several places within one message flow, provides an efficient and consistent use of resources and avoids reinventing such routines every time an error is encountered.
- You might want to perform a common calculation on messages that pass through several different message flows; for example, you might access currency exchange rates from a database and apply them to calculate prices in several different currencies. You can include the currency calculator subflow in each of the message flows in which it is appropriate.

Use the Passthrough node to enable version control of a subflow at run time. By including a Passthrough node, you can add a label to your message flow or subflow. By combining this label with keyword replacement from your version control system, you can identify which version of a subflow is included in a deployed message flow. You can use this label for your own purposes. If you have included the correct version keywords in the label, you can see the value of the label:

- Stored in the broker archive (BAR) file, by using the **mqsireadbar** command
- As last deployed to a particular broker, on the properties of a deployed message flow in the WebSphere Message Broker Toolkit
- In the runtime environment, if you enable user trace for that message flow

The message that it propagates on its Out terminal is the same message that it received on its In terminal; for example, if you develop an error processing subflow to include in several message flows, you might want to modify that subflow. However, you might want to introduce the modified version initially to just a subset of the message flows in which it is included. Set a value for the instance of the Passthrough node that identifies which version of the subflow you have included.

The use of subflows is demonstrated in the following samples:

- Error Handler
- Coordinated Request Reply

Error Handler uses a subflow to trap information about errors and store the information in a database. Coordinated Request Reply uses a subflow to encapsulate the storage of the ReplyToQ and ReplyToQMgr values in a WebSphere MQ message so that the processing logic can be reused in other message flows, and to allow alternative implementations to be substituted.

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Promoted properties" on page 1145
A promoted property is a message flow node property that has been promoted to the level of the message flow in which it is included.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Adding a subflow" on page 1501
In a message flow, you can include an embedded message flow, also known as a *subflow*. For example, you might define a subflow that provides error handling, and include it in a message flow connected to a failure terminal on a node that can generate an error in some situations.

"Defining a promoted property" on page 1297
When you create a message flow, you can promote properties from individual nodes in that message flow to the message flow level. Properties promoted in this way override the property values that you have set for the individual nodes.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Message flow connections

A connection is an entity that connects an output terminal of one message flow node to an input terminal of another. The connection represents the flow of control and data between two message flow nodes.

Most nodes have one input terminal, and many have more than one output terminal. You can connect an output terminal to more than one target node, so that the same message can be processed in a number of different ways. For example, you might want to send the same information to more than one remote application for further processing.

The connections of the message flow, represented by black lines in the Message Flow editor, determine the path that a message takes through the message flow. You can add bend points to the connection to alter the way in which it is displayed.

For a description of bend points, see "Bend points" on page 1033. For a description of terminals, see "Message flow node terminals" on page 1034.

Related concepts:

"Bend points" on page 1033
A bend point is added to a connection between two message flow nodes where the line that represents the connection changes direction.

"Message flow node terminals" on page 1034
A terminal is the point at which one node in a message flow is connected to another node.

"Message flow nodes" on page 1024
A message flow node is a processing step in a message flow. It can be a built-in node, a user-defined node, or a subflow node.

"Message flow projects" on page 1035
A message flow project is a specialized container in which you create and maintain all the resources associated with one or more message flows.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"Message flow projects and files" on page 6822

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Bend points:

A bend point is added to a connection between two message flow nodes where the line that represents the connection changes direction.

Use bend points to change the visual path of a connection to display node alignment and processing logic more clearly and effectively. Bend points have no effect on the behavior of the message flow; they are visual modifications only.

A connection is initially made as a straight line between the two connected nodes or brokers. Use bend points to move the representation of the connection, without moving its start and end points.

Related concepts:

"Message flow projects" on page 1035
A message flow project is a specialized container in which you create and maintain all the resources associated with one or more message flows.

"Message flow nodes" on page 1024
A message flow node is a processing step in a message flow. It can be a built-in node, a user-defined node, or a subflow node.

"Message flow node terminals" on page 1034
A terminal is the point at which one node in a message flow is connected to another node.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"Message Flow editor" on page 6810
The Message Flow editor is the default editor provided by the Broker Application Development perspective for defining a graphical representation of a message flow in the WebSphere Message Broker Toolkit, and for setting properties for individual message flow nodes.

"Message flow projects and files" on page 6822

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Message flow node terminals:

A terminal is the point at which one node in a message flow is connected to another node.

Use terminals to control the route that a message takes, depending on whether the operation that is performed by a node on that message is successful. Terminals are wired to other node terminals by using message flow node connections to indicate the flow of control.

Every built-in node has a number of terminals to which you can connect other nodes. Input nodes (for example, MQInput) do not have input terminals; all other nodes have at least one input terminal through which to receive messages to be processed. Most built-in nodes have failure terminals that you can use to manage the handling of errors in the message flow. Most nodes have output terminals through which the message can flow to a subsequent node.

If you have any user-defined nodes, these might also have terminals that you can connect to other built-in or user-defined node terminals.

Dynamic terminals are terminals that you can add to certain nodes after you have added them to a message flow in the Message Flow editor. For example, you can add dynamic output terminals to the PHPCompute, Route, and DatabaseRoute nodes, or you can add dynamic input terminals to the Collector node. You can also delete and rename dynamic terminals. If a node has five or more terminals, they are displayed in a group. For example, the following example shows a node with more than four output terminals. 

Related concepts:

"Message flow nodes" on page 1024
A message flow node is a processing step in a message flow. It can be a built-in node, a user-defined node, or a subflow node.

"Bend points" on page 1033
A bend point is added to a connection between two message flow nodes where the line that represents the connection changes direction.

Related tasks:

"Using dynamic terminals" on page 1518
You can add, rename, and remove dynamic terminals on a node in the Message Flow editor.

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"Message Flow editor" on page 6810
The Message Flow editor is the default editor provided by the Broker Application Development perspective for defining a graphical representation of a message flow in the WebSphere Message Broker Toolkit, and for setting properties for individual message flow nodes.

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"DatabaseRoute node" on page 4373
Use the DatabaseRoute node to route messages using information from a database

in conjunction with XPath expressions.

"Route node" on page 4669
Use the Route node to direct messages that meet certain criteria down different paths of a message flow.

"Collector node" on page 4333
Use the Collector node to create message collections based on rules that you configure in the node.

"PHPCompute node" on page 4639
Use the PHPCompute node to route and transform an incoming message, using the PHP scripting language.

Message flow projects

A message flow project is a specialized container in which you create and maintain all the resources associated with one or more message flows.

You can create a message flow project to contain a single message flow and its resources, or you can group together related message flows and resources in a single message flow project to provide an organizational structure to your message flow resources.

Message flow project resources are created as files, and are displayed in the project in the Broker Development view. These resources define the content of the message flow, and additional objects that contain detailed configuration information. For example, a project might contain ESQL modules or message mappings, used by one or more nodes in the message flow.

Import one of the following samples to see how the message flow resources of the sample are stored in a message flow project. If the sample has a message set, the message set resources are stored in a Message Set project.

- Video Rental
- Comma Separated Value (CSV)

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Related concepts:

"Project references" on page 44
When a projects refers to other projects in the workspace, this is called a *project reference*. When one project references another, the files in the referenced project are available for use by the referring project. For example to configure certain nodes, to create libraries of reusable message flows, and to enable Content Assist in the ESQL editor.

"Message flow nodes" on page 1024
A message flow node is a processing step in a message flow. It can be a built-in node, a user-defined node, or a subflow node.

"Message flow connections" on page 1032
A connection is an entity that connects an output terminal of one message flow node to an input terminal of another. The connection represents the flow of control and data between two message flow nodes.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

Chapter 9, “Developing message flow applications,” on page 1019
Develop message flows to process your business messages and data.

Related reference:

“Message flow projects and files” on page 6822

“Built-in nodes” on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Broker schemas

A broker schema is a symbol space that defines the scope of uniqueness of the names of resources defined within it. The resources include message flows and other optional resources such as ESQL files, Java files, and mapping files.

The broker schema is defined as the relative path from the project source directory to the flow name. When you first create a message flow project, a default broker schema named `(default)` is created within the project.

You can create new broker schemas to provide separate symbol spaces within the same message flow project. A broker schema is implemented as a folder, or subdirectory, within the project, and provides organization within that project. You can also use project references to spread the scope of a single broker schema across multiple projects to create an application symbol space that provides a scope for all resources associated with an application suite.

If you create a new broker schema while in category mode, the empty schema is not visible in the Broker Development view. To show the empty schema in the Broker Development view, click **Hide Categories**  on the toolbar.

A broker schema name must be a character string that starts with a Unicode character followed by zero or more Unicode characters or digits, and the underscore. You can use the period to provide a structure to the name, for example `Stock.Common`. A directory is created in the project directory to represent the schema, and if the schema is structured using periods, further subdirectories are defined. For example, the broker schema `Stock.Common` results in a directory `Common` within a directory `Stock` within the message flow project directory.

If you create a resource (for example, a message flow) in the default broker schema within a project, the file or files associated with that resource are created in the directory that represents the project. If you create a resource in another broker schema, the files are created within the schema directory.

For example, if you create a message flow in the default schema in the message flow project Project1, its associated files are stored in the Project1 directory. If you create another message flow in the `Stock.Common` broker schema within the project Project1, its associated files are created in the directory `Project1\Stock\Common`.

Because each broker schema represents a unique name scope, you can create two message flows that share the same name within two broker schemas. The broker schemas ensure that these two message flows are recognized as separate resources. The two message flows, despite having the same name, are considered unique.

If you move a message flow from one project to another, you can continue to use the message flow within the original project if you preserve the broker schema. If you do this, you must update the list of dependent projects for the original project by adding the target project. If, however, you do not preserve the broker schema, the flow becomes a different flow because the schema name is part of the fully qualified message flow name, and it is no longer recognized by other projects. This action results in broken links that you must manually correct. For further information about correcting errors after moving a message flow, see "Moving a message flow" on page 1439.

Do not move resources by moving their associated files in the file system; you must use the WebSphere Message Broker Toolkit to move resources to ensure that all references are corrected to reflect the new organization.

The following scope and reuse conditions apply when you create functions, procedures, and constants in a broker schema:

Functions

- Functions are locally reusable and can be called by module-scope subroutines or mappings within the same schema.
- Functions are globally reusable and can be called by other functions or procedures in ESQL or mapping files within any schema defined in the same or another project.

Procedures

- Procedures are locally reusable and can be called from module-scope subroutines in ESQL files within the same schema.
- Procedures are globally reusable and can be called by other functions or procedures in ESQL files within any schema defined in the same or another project.

Procedures cannot be used in mapping files.

Constants

- Constants are locally reusable and can be used where they are defined in any ESQL or mapping file within the same broker schema.
- Constants are not globally reusable; you cannot use a constant that is declared in another schema.

If you want to reuse functions or procedures globally:

- Specify the path of the function or procedure:
  - If you want to reuse a function or procedure in an ESQL file, either provide a fully-qualified reference, or include a PATH statement that defines the path.

    If you define the path, code the PATH statement in the same ESQL file as that in which the function is coded, but not within any MODULE.
  - If you want to reuse a function in a mapping file, do one of the following:
    - Qualify the function in the Composition Expression editor.
    - Select **Organize Schema References** in the outline view. This detects dependent PATHs and automatically updates the reference.
    - Select **Modify Schema References** in the outline view. You can then select the schema in which the function is defined.

    (You cannot reuse a procedure in a mapping file.)
- Set up references between the projects in which the functions and procedures are defined and used.

Related concepts:

"By name linking" on page 43
You identify objects using a combination of a *namespace* and a name, referred to as a *fully qualified name*. The use of fully qualified names, called *by name linking*, makes it easy to identify and locate objects, and to correct broken references.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Creating a broker schema" on page 1429
To organize your message flow project resources, and to define the scope of resource names to ensure uniqueness, you can create broker schemas. A default schema is created when you create the message flow project, but you can create additional schemas.

"Using message mappings" on page 2228
Message mappings define the blueprint for creating a message.

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Client application programming interfaces

You can configure the nodes in your message flows to customize the behavior of those nodes by using one or more of the supported programming interfaces.

You can use one of the following options for converting message formats in your message flows:

Mappings
: The Message Mapping editor in the WebSphere Message Broker Toolkit is a graphical interface that displays a visual representation of messages. You can use this editor to:

  - Drag fields from a source message to a target message
  - Map data from databases to the message structure
  - Apply functions

  You can use mappings in the DataDelete, DataInsert, DataUpdate, Mapping, and Warehouse nodes.

  Use this option if the transformations you want to perform are not complex, or you do not want to use a programming language.

ESQL Use ESQL to manipulate data in both messages and database tables. ESQL

is a programming language that is specific to WebSphere Message Broker, and based on SQL. You can code ESQL statements to create, reference, and update message fields and database content. ESQL provides a rich set of statements and functions that you can use to achieve sophisticated transformations.

You can use ESQL in the Compute, Database, and Filter nodes.

Use this option if the transformations you want to perform are complex, and your message flow designers are familiar with procedural programming languages.

Java Use the Java programming language to route or transform your messages. You can use XPath to create, reference, and update message fields. You can also use JDBC to access database tables.

You can use Java only in the JavaCompute node.

Use this option if your message flow designers are familiar with the Java programming language.

XSL style sheets
Use standard XSL style sheets to convert XML messages to other formats supported by the broker.

You can use XSL only in theXSLTransform node.

Use this option if your message flows are processing XML messages, and your message flow designers are familiar with the XSL style sheets.

PHP Use the PHP programming language to route or transform your messages. You can use PHP to create, reference, and update message fields.

You can use PHP only in the PHPCompute node.

Use this option if your message flow designers are familiar with the PHP scripts and the programming language.

When you configure your message flows and nodes, you create a set of files that are stored in your workspace.

The files created are of the following types:

- A message flow definition file, `<message_flow_name>.msgflow`. This file is mandatory, and is created automatically for you. It contains details about the message flow characteristics and contents; for example, what nodes it includes and its promoted properties.
- One or more message mappings files, `<message_flow_name><_nodename>.msgmap`. A unique file is required for each node in the message flow that uses the Message Mapping editor. This file is required only if your message flow contains one or more of the nodes that you can customize by using mappings. You can create this file yourself, or you can cause it to be created for you by requesting specific actions against a node.

  You can customize the following built-in nodes by specifying how input values map to output values.

Node	Usage
"DataDelete node" on page 4382	Use this node to delete one or more rows from a database table without creating an output message.

Node	Usage
"DataInsert node" on page 4386	Use this node to insert one or more rows in a database table without creating an output message.
"DataUpdate node" on page 4390	Use this node to update one or more rows in a database table without creating an output message.
"Mapping node" on page 4571	Use this node to construct output messages and populate them with information that is new, modified from the input message, or taken from a database. You can also use the Mapping node to update, insert, or delete rows in a database table.
"Warehouse node" on page 4963	Use this node to store all or part of a message in a database table without creating an output message.

You can use built-in ESQL functions and statements to define message mappings, and you can use your own ESQL functions.

The "Extract node" on page 4412 also uses mappings to create an output message that contains a subset of the contents of the input message. However, the Extract node is deprecated in WebSphere Message Broker Version 6.0 and later releases. Although message flows that contain an Extract node remain valid in WebSphere Message Broker Version 6.0, redesign your message flows to replace Extract nodes by Mapping nodes to take advantage of later enhancements.

- One or more ESQL resources files, `<message_flow_name>.esql`. An ESQL file is required only if your message flow includes one or more of the nodes that must be customized by using ESQL modules, or contains functions that are called by your message mappings. You can create this file yourself, or you can cause it to be created for you by requesting specific actions against a node.

  You can customize the following built-in nodes by creating free-form ESQL statements that use the built-in ESQL statements and functions, and your own user-defined functions:
  - Compute
  - Database
  - Filter
- One or more Java programming files, `<message_flow_name>.java`. A Java file is required only if your message flow includes one or more JavaCompute nodes. You can create this file yourself, or you can cause it to be created for you by requesting specific actions against a node.
- One or more XSL stylesheets, `<message_flow_name>.xslt`. An XSLT file is required only if your message flow includes one or more XSLTransform nodes. You can create this file yourself, or you can cause it to be created for you by requesting specific actions against a node.
- One or more PHP scripts, `<message_flow_name>.php`. A PHP script file is required only if your message flow includes one or more PHPCompute nodes. You can create this file yourself, or you can cause it to be created for you by requesting specific actions against a node.

You can include other files in your message flow project so that they are deployed to the broker with your message flow. The broker stores these extra files but does not process them in any way.

For details of how to create the files to support these transform options, and create their content, see "Transforming and enriching messages" on page 2227.

Related tasks:

"Transforming and enriching messages" on page 2227
Transform and enrich messages by using one or more of the techniques described in this section.

Related reference:

"Transformation interfaces" on page 4980
View the reference material associated with the different ways in which you can transform messages in message flows.

"Mapping node" on page 4571
Use the Mapping node to construct one or more new messages and populate them with various types of information.

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"Filter node" on page 4452
Use the Filter node to route a message according to message content.

"JavaCompute node" on page 4514
Use the JavaCompute node to work with messages by using the Java language.

"XSLTransform node" on page 4968
Use the XSLTransform node to transform an XML message to another form of message, according to the rules provided by an XSL (Extensible Stylesheet Language) style sheet, and to set the `Message domain`, `Message set`, `Message type`, and `Message format` for the generated message.

"PHPCompute node" on page 4639
Use the PHPCompute node to route and transform an incoming message, using the PHP scripting language.

The message tree

A message tree is a structure that is created, either by one or more parsers when an input message bit stream is received by a message flow, or by the action of a message flow node.

A message is used to describe:

- A set of business data that is exchanged by applications
- A set of elements that are arranged in a predefined structure
- A structured sequence of bytes

WebSphere Message Broker routes and manipulates messages after converting them into a logical tree. The process of conversion, called parsing, makes obvious the content and structure of a message, and simplifies later operations. After the message has been processed, the parser converts it back into a bit stream.

WebSphere Message Broker supplies a range of parsers to handle the many different messaging standards in use. See Parsers.

After a message has been parsed, it can be processed in a message flow.

The logical tree has contents that are identical to the message, but the logical tree is easier to manipulate in the message flow. The message flow nodes provide an interface to query, update, or create the content of the tree.

Related concepts:

"Parsers" on page 1072
A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from the internal message tree representation.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

Logical tree structure:

The logical tree structure is the internal (broker) representation of a message. It is also known as the message assembly.

When a message arrives at a broker, it is received by an input node that you have configured in a message flow. Before the message can be processed by the message flow, the message must be interpreted by one or more parsers that create a logical tree representation from the bit stream of the message data.

The tree format contains identical content to the bit stream from which it is created, but it is easier to manipulate in the message flow. Many of the built-in message flow nodes provide an interface for you to query and update message content in the tree, and perform other actions against messages and databases to help you to provide the required function in each node.

Several interfaces are provided:

- ESQL, a programming language that you can code in the Compute, Database, and Filter nodes.
- Java, a programming language that you can code in the JavaCompute node.
- Mappings, a graphical method of achieving transformation from input to output structures, available in the DataDelete, DataInsert, DataUpdate, Mapping, and Warehouse nodes.
- XSL, a language for transforming XML that you can use in the XSLTransform node.
- PHP, a scripting language that you can code in the PHPCompute node.

The tree structure that is created by the parsers is largely independent of any message format (for example, XML). The exception to this is the subtree that is created as part of the message tree to represent the message body. This subtree is message dependent, and its content is not further described here.

The input node creates this message assembly, which consists of four trees:
- "Message tree structure" on page 1045
- "Environment tree structure" on page 1055
- "Local environment tree structure" on page 1056
- "Exception list tree structure" on page 1066

The first of these trees is populated with the contents of the input message bit stream, as described in "How the message tree is populated" on page 1047; the remaining three trees are initially empty.

Each of the four trees created has a root element (with a name that is specific to each tree). Each tree is made up of a number of discrete pieces of information called *elements*. The root element has no *parent* and no *siblings* (siblings are elements that share a single parent). The root is parent to a number of *child* elements. Each child must have a parent, can have zero or more siblings, and can have zero or more children.

The four trees are created for both built-in and user-defined input nodes and parsers.

The input node passes the message assembly that it has created to subsequent message processing nodes in the message flow:

- All message processing nodes can read the four trees.
- You can code ESQL in the Database and Filter nodes, or use mappings in the nodes that support that interface to modify the Environment and LocalEnvironment trees only.
- The Compute node differs from other nodes in that it has both an input message assembly and at least one output message assembly. Configure the Compute node to determine which trees are included in the output message assembly; the Environment tree is an exception in that it is always retained from input message assembly to output message assembly.

  To determine which of the other trees are included, you must specify a value for the `Compute mode` property of the node (displayed on the Advanced tab). The default action is for only the message to be created. You can specify any combination of message, LocalEnvironment, and ExceptionList trees to be created in the output message assembly.

  If you want the output message assembly to contain a complete copy of the input message tree, you can code a single ESQL SET statement to make the copy. If you want the output message to contain a subset of the input message tree, code ESQL to copy those parts that you want. In both cases, your choice of `Compute mode` must include `Message`.

  If you want the output message assembly to contain all or part of the input LocalEnvironment or ExceptionList tree contents, code the appropriate ESQL to copy information you want to retain in that tree. Your choice of `Compute mode` must include LocalEnvironment, or Exception, or both.

  You can also code ESQL to populate the output message, Environment, LocalEnvironment, or ExceptionList tree with information that is not copied from the input tree. For example, you can retrieve data from a database, or calculate content from the input message data.
- You can produce similar results in the JavaCompute node. See "Writing Java" on page 2638 for more information.
- You can also achieve results like these in the PHPCompute node. See "Using PHP" on page 2670 for more information.

Related concepts:

"Message tree structure" on page 1045
The message tree is a part of the logical message tree in which the broker stores its internal representation of the message body.

"How the message tree is populated" on page 1047
The message tree is initially populated by the input node of the message flow.

"Environment tree structure" on page 1055
The environment tree is a part of the logical message tree in which you can store information while the message passes through the message flow.

"Local environment tree structure" on page 1056
The local environment tree is a part of the logical message tree in which you can store information while the message flow processes the message.

"Exception list tree structure" on page 1066
The exception list tree is a part of the logical message tree in which the message flow writes information about exceptions that occur when a message is processed.

"Correlation names" on page 1069
A correlation name is a field reference that identifies a well-defined starting point in the logical message tree and is used in field references to describe a standard part of the tree format.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Manipulating message body content" on page 2418
The message body is always the last child of root, and is identified by its parser name, for example XML or MRM. You can refer to, modify, and create message body data.

"Manipulating other parts of the message tree" on page 2452
You can access message tree headers, the properties tree, the local environment tree, the environment tree and the exception list tree.

"Writing ESQL" on page 2413
How you can use ESQL to customize nodes.

"Writing Java" on page 2638
When you create a message flow, you include input nodes that receive messages and, optionally, output nodes that send out new or updated messages. If the message processing requires it, you can include other nodes after the input node that are customized in Java to complete the actions that your applications need.

"Using PHP" on page 2670
You can use the PHP scripting language for message routing and transformation.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"User-defined nodes" on page 6415
You can define your own nodes to use in WebSphere Message Broker message flows.

"MQRFH2 header" on page 6397
The MQRFH2 header is used to pass messages to and from a message broker that belongs to WebSphere Message Broker.

Message tree structure:

The message tree is a part of the logical message tree in which the broker stores its internal representation of the message body.

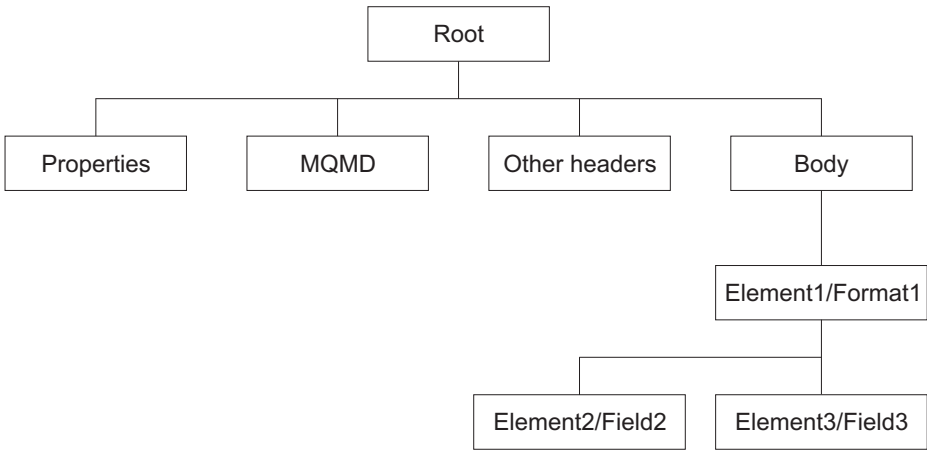
The root of a message tree is called Root. The message tree is always present, and is passed from node to node in a single instance of a message flow.

The message tree includes all the headers that are present in the message, in addition to the message body. The tree also includes the Properties subtree (described in "Parsers" on page 1072), if that is created by the parser. If a supplied parser has created the message tree, the element that represents the Properties subtree is followed by zero or more headers.

If the message has been received across the WebSphere MQ Enterprise Transport or WebSphere MQ Mobile Transport, the first header (the second element) must be the MQMD. Any additional headers that are included in the message appear in the tree in the same order as in the message. The last element beneath the root of the message tree is always the message body.

If a user-defined parser has created the message tree, the Properties tree, if present, is followed by the message body.

The message tree structure is shown in the following section. If the input message is not a WebSphere MQ message, the headers that are shown might not be present. If the parser that created this tree is a user-defined parser, the Properties tree might not be present.



The Body tree is a structure of child elements that represents the message content (data), and reflects the logical structure of that content. The Body tree is created by a body parser (either a supplied parser or a user-defined parser), as described in "Parsers" on page 1072.

Each element in the parsed tree is one of three types:

Name element

A name element has a string associated with it, which is the name of the element. An example of a name element is `XMLElement`, as described in "XML element" on page 4267. A name element also has a second string associated with it, which is the namespace of the element; this string might be empty.

Value element

A value element has a value associated with it. An example of a value element is `XMLContent`, as described in "XML content" on page 4267.

Name-value element

A name-value element is an optimization of the case where a name element contains only a value element and nothing else. The element contains both a name and a value. An example of a name-value element is `XMLAttribute`, as described in "XML attribute" on page 4264.

For information about how the message tree is populated, see "How the message tree is populated" on page 1047.

Properties folder: The Properties folder is the first element of the message tree and holds information about the characteristics of the message.

The root of the Properties folder is called Properties. It is the first element under Root. All message trees that are generated by the built-in parsers include a Properties folder for the message. If you create your own user-defined parser, you can choose whether the parser creates a Properties folder. However, for consistency, you should include this action in the user-defined parser.

The Properties folder contains a set of standard properties that you can manipulate in the message flow nodes in the same way as any other property. Some of these fields map to fields in the supported WebSphere MQ headers, if present, and are passed to the appropriate parser when a message is delivered from one node to another.

For example, the MQRFH2 header contains information about the message set, message type, and message format. These values are stored in the Properties folder as MessageSet, MessageType, and MessageFormat. To access these values using ESQL or Java in the message processing nodes, refer to these values in the Properties folder; do not refer directly to the fields in the headers from which they are derived.

The Properties parser ensures that the values in the header fields match the values in the Properties folder on input to, and output from, every node. For any field, if only one header is changed (the Properties header or a specific message header), that value is used. If both the Properties header and the specific message header are changed, the value from the Properties folder is used.

When the message flow processing is complete, the Properties folder is discarded.

Related concepts:

"Correlation names" on page 1069
A correlation name is a field reference that identifies a well-defined starting point in the logical message tree and is used in field references to describe a standard part of the tree format.

"Parsers" on page 1072
A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from the internal message tree representation.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Accessing the Properties tree" on page 2460
The Properties tree has its own correlation name, Properties, and you must use this in all ESQL statements that refer to or set the content of this tree.

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Writing ESQL" on page 2413
How you can use ESQL to customize nodes.

"Writing Java" on page 2638
When you create a message flow, you include input nodes that receive messages and, optionally, output nodes that send out new or updated messages. If the message processing requires it, you can include other nodes after the input node that are customized in Java to complete the actions that your applications need.

"Manipulating message body content" on page 2418
The message body is always the last child of root, and is identified by its parser name, for example XML or MRM. You can refer to, modify, and create message body data.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

How the message tree is populated:

The message tree is initially populated by the input node of the message flow.

When the input node receives the input message, it creates and completes the Properties tree (the first subtree of the message tree). See "Message tree structure" on page 1045.

The node then examines the contents of the input message bit stream and creates the remainder of the message tree to reflect those contents. This process depends to some extent on the input node itself, which is governed by the transport across which the message is received:

WebSphere MQ Enterprise Transport protocol
: If your application communicates with the broker across these protocols, and your message flow includes the corresponding MQInput node, all messages that are received must start with a Message Queue Message Descriptor (MQMD) header. If a valid MQMD is not present at the start of the message, the message is rejected, and no further processing takes place.

  The input node first invokes the MQMD parser and creates the subtree for that header.

  A message can have zero or more additional headers following the MQMD. These headers are chained together, with the **Format** field of one header defining the format of the following header, up to and including

the last header, which defines the format of the message body. If an MQRFH and an MQRFH2 header exist in the chain, the name and value data in either of these two headers can also contain information about the format of the following data. If the value that is specified in **Format** is a recognized parser, this value always takes precedence over the name and value data.

The broker invokes the appropriate parser to interpret each header, following the chain in the message. Each header is parsed independently. The fields in a single header are parsed in an order that is governed by the parser. You cannot predict the order that is chosen, but the order in which fields are parsed does not affect the order in which the fields are displayed in the header.

The broker ensures that the integrity of the headers that precede a message body is maintained. The format of each part of the message is defined, either by the **Format** field in the immediately preceding header (if the following part is a recognized WebSphere MQ format), or by the values that are set in the MQRFH or MQRFH2 header:

- The format of the first header is known because it must be MQMD.
- The format of any subsequent header in the message is set in the **Format** field in the preceding header.
- The format of the body corresponds to the message domain and the parser that must be invoked for the message body (for example, XMLNSC). This information is set either in the MQRFH or MQRFH2 header, or in the *Message Domain* property of the input node that receives the message.

This process is repeated as many times as required by the number of headers that precede the message body. You do not need to populate these fields yourself; the broker handles this sequence for you.

The broker completes this process to ensure that **Format** fields in headers correctly identify each part of the message. If the broker does not complete this process, WebSphere MQ might be unable to deliver the message. The message body parser is not a recognized WebSphere MQ header format, therefore the broker replaces this value in the last headers **Format** field with the value MQFMT_NONE. The original value in that field is stored in the **Domain** field in the MQRFH or MQRFH2 header to retain the information about the contents of the message body.

For example, if the MQRFH2 header immediately precedes the message body, and its **Format** field is set to XMLNSC, which indicates that the message body must be parsed by the XMLNSC parser, the MQRFH2 **Domain** field is set to XMLNSC, and its **Format** field is reset to MQFMT_NONE.

These actions might result in information that is stored explicitly by an ESQL or Java expression being replaced by the broker.

The **CodedCharSetId** and **Encoding** fields are not populated in the same way as the **Format** field. In particular, the message body is not used to determine the **CodedCharSetId** and **Encoding** values. Rather, these values affect the way in which the message body is written. This can cause unexpected results if an intermediate header (for example MQRFH2) is removed without updating the preceding header in the chain with the removed header values.

When all the headers have been parsed, and the corresponding sub-trees have been created in the message tree, the input node associates the specified parser with the message body. Specify the parser that is to be associated with the message body content, either in a header in the message (for example, the <mcd> folder in the MQRFH2 header), or in the input node properties (if the message does not include headers). The input node makes the association as described in the following list:

- If the message has an MQRFH or MQRFH2 header, the domain that is identified in the header (either in **Format** or the name and value data) determines the parser that is associated with this message.
- If the message does not have an MQRFH or MQRFH2 header, or if the header does not identify the domain, the *Message Domain* property of the input node indicates the domain of the message, and the parser that is to be used. You can specify a user-defined domain and parser.
- If the message domain cannot be identified by header values or by the *Message Domain* property of the input node, the message is handled as a binary object (BLOB). The BLOB parser is associated with the message. A BLOB can be interpreted as a string of hexadecimal characters, and can be modified or examined in the message flow by specifying the location of the subset of the string.

By default, the message body is not parsed straight away, for performance reasons. The message body might not need to be parsed during the message flow. It is parsed only when a reference is made to its contents.

For example, the message body is parsed when you refer to a field in the message body, for example: `Root.XMLNSC.MyDoc.MyField`. Depending on the paths that are taken in the message flow, this can take place at different points. This parsing when first needed approach is also referred to as "partial parsing" or "on-demand parsing", and in typical processing does not affect the logic of a message flow. However, there are some implications for error handling scenarios; see “Handling errors in message flows” on page 2823.

If you want a message flow to accept messages from more than one message domain, include an MQRFH2 header in your message from which the input nodes extract the message domain and related message definition information (message set, message type, and message format).

If you set up the message headers or the input node properties to identify a user-defined domain and parser, the way in which it interprets the message and constructs the logical tree might differ from that described here.

WebSphere Broker File Transport, WebSphere Broker Adapters Transport, WebSphere MQ Web Services Transport, and WebSphere Broker JMS Transport protocols

If your application communicates with the broker across these supported protocols, and your message flow includes the corresponding input nodes, messages that are received do not have to include a particular header. If recognized headers are included, the input node invokes the appropriate parsers to interpret the headers and to build the relevant parts of the message tree, as described for the other supported protocols.

If there are no headers, or these headers do not specify the parser for the message body, set the input node properties to define the message body parser. If you do not set the node properties in this way, the message is treated as a BLOB. You can specify a user-defined parser.

The specified parser is associated with the message body by the input node (in the same way as it is for the WebSphere MQ Enterprise Transport protocol), and by default the message body is not parsed immediately.

If you set up the message headers or the input node properties to identify a user-defined domain and parser, the way in which it interprets the message and constructs the logical tree might differ from that described here.

All other protocols

If you want your message flow to accept messages from a transport protocol for which WebSphere Message Broker does not provide built-in support, or you want it to provide some specific processing on receipt of a message, use either the Java or the C language programming interface to create a new user-defined input node.

This interface does not automatically generate a Properties subtree for a message (this subtree is discussed in "Message tree structure" on page 1045). A message does not need to have a Properties subtree, but you might find it useful to create one to provide a consistent message tree structure, regardless of input node. If you are using a user-defined input node, you must create a Properties subtree in the message tree yourself.

To process messages that do not conform to any of the defined message domains, use the C language programming interface to create a new user-defined parser.

Refer to the node interface to understand how it uses parsers, and whether you can configure it to modify its behavior. If the node uses a user-defined parser, the tree structure that is created for the message might differ slightly from that created for built-in parsers. A user-defined input node can parse an input message completely, or it can participate in partial parsing in which the message body is parsed only when it is required.

You can also create your own output and message processing nodes in C or Java.

Properties versus MQMD folder behavior for various transports

Differences exist in the way the Properties folder and the MQMD folder are treated with respect to which folder takes precedence for the same fields. This treatment is characterized by the transport type (for example, HTTP or WebSphere MQ) that you use.

When the message flow is sourced by an MQInput node, you have an MQMD to parse. In this case, the Properties folder is sourced by the MQMD values and so the MQMD folder takes precedence over the Properties folder in terms of value propagation between the folders. This scenario means that you can perform ESQL, for example, `SET OutputRoot.MQMD.CorrelId` and this command updates the Properties folder value.

When the message flow is sourced from an input node that is not the MQInput node (such as the HTTPInput node or a user-defined input node), no MQMD is parsed. In this scenario, the Properties folder is not sourced from an input MQMD; it is created and populated with transport values that come from other transport specific headers. When you create an MQMD folder in a message flow that was not sourced from the WebSphere MQ transport, the MQMD header does not take

precedence over the Properties folder because the WebSphere MQ transport did not populate the Properties folder. Therefore, in this case, the Properties folder overwrites any values in MQMD.

The Properties folder is constructed and represents a message received on the transport. In this scenario two entirely different transports are being used which have different meanings and, therefore, different requirements of the Properties folder. When sourced from an HTTPInput node, the HTTP headers have control over the Properties folder for like fields. When sourced from an MQInput node the MQMD has control over the Properties folder for like fields.

Therefore, when you add an MQMD folder to a tree that was created by the HTTP Transport, this MQMD folder does not have control over the Properties folder, and the value propagation direction is not MQMD to Properties, it is Properties to MQMD. The correct approach is to set the **replyIdentifier** field of the Properties folder and to use it to populate the MQMD:

```
SET OutputRoot.Properties.ReplyIdentifier = X' ..... ';
```

The behavior is not unique to just the **CorrelId** to **ReplyIdentifier** fields. It applies for all like fields between the MQMD and Properties folder:

- **CorrelId**
- **Encoding**
- **CodedCharSetId**
- **Persistence**
- **Expiry**
- **Priority**

In summary:

1. When your message flow is sourced by an MQInput node, the MQMD takes precedence over the Properties folder in terms of value propagation between the folders.
2. When your message flow is sourced from an input node that is not the MQInput node (such as the HTTPInput node or a user-defined input node), the MQMD header does not take precedence over Properties folder .
3. When an MQMD folder is added in a tree that was created by the HTTP Transport, this MQMD does not have control over the Properties folder and the value propagation direction is not MQMD to Properties; it is Properties to MQMD.

Example

```
SET OutputRoot.Properties = InputRoot.Properties;
SET OutputRoot.MQMD.Version = 2;
SET OutputRoot.MQMD.CorrelId = X'4d454e53414a4533202020202020202020202020202020202020';
SET OutputRoot.MQMD.Encoding = 785;
SET OutputRoot.MQMD.CodedCharSetId = 500;
SET OutputRoot.MQMD.Persistence = 1;
SET OutputRoot.MQMD.Expiry = 10000;
SET OutputRoot.MQMD.Priority = 9;
SET OutputRoot.BLOB.BLOB = X'01';
```

When sourced from an HTTPInput node none of these changes take effect and the MQMD output from the Compute node is:

```
(0x01000000):MQMD       = (
    (0x03000000):Version        = 2
    (0x03000000):CorrelId       = X'000000000000000000000000000000000000000000000000'
```

```
(0x03000000):Encoding       = 546
(0x03000000):CodedCharSetId = 1208
(0x03000000):Persistence    = FALSE
(0x03000000):Expiry         = -1
(0x03000000):Priority       = 0
```

Related concepts:

"Logical tree structure" on page 1042
The logical tree structure is the internal (broker) representation of a message. It is also known as the message assembly.

"Message tree structure" on page 1045
The message tree is a part of the logical message tree in which the broker stores its internal representation of the message body.

"Environment tree structure" on page 1055
The environment tree is a part of the logical message tree in which you can store information while the message passes through the message flow.

"Local environment tree structure" on page 1056
The local environment tree is a part of the logical message tree in which you can store information while the message flow processes the message.

"Exception list tree structure" on page 1066
The exception list tree is a part of the logical message tree in which the message flow writes information about exceptions that occur when a message is processed.

"Correlation names" on page 1069
A correlation name is a field reference that identifies a well-defined starting point in the logical message tree and is used in field references to describe a standard part of the tree format.

Related tasks:

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"User-defined nodes" on page 6415
You can define your own nodes to use in WebSphere Message Broker message flows.

"MQRFH2 header" on page 6397
The MQRFH2 header is used to pass messages to and from a message broker that belongs to WebSphere Message Broker.

"Parsing on demand" on page 4173
On-demand parsing, referred to as partial parsing, is used to parse an input message bit stream only as far as is necessary to satisfy the current reference. The parsers that can perform partial parsing of input messages are the JSON, MRM, XML, XMLNS, and XMLNSC parsers.

Message tree contents after an exception:

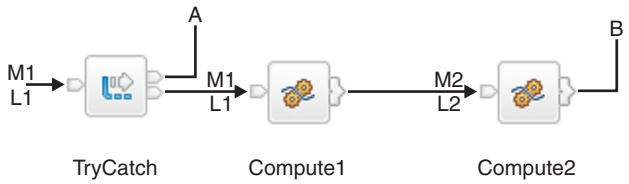
The contents of the message tree are updated if an exception is raised.

If no exception occurs while processing the message, the tree structure and content received by an individual node is determined by the action of previous nodes in the flow. If an exception occurs in the message flow, the content of the four trees depends on the following factors:

- If the exception is returned to the input node, and the input node Catch terminal is not connected, the trees are discarded. If the message is in a transaction, it is returned to the input queue for further processing. When the message is processed again, a new tree structure is created. If the message is not in a transaction, it is discarded.
- If the exception is returned to the input node and the Catch terminal is connected, the message and local environment trees that were created originally by the input node, and propagated through the Out terminal, are restored, and any updates that you made to their content in the nodes that followed the input node are lost. The environment tree is not restored, and its contents are preserved. If the nodes following the input node include a Compute node that creates a new local environment or message tree, those trees are lost. The exception list tree reflects the one or more exceptions that have been recorded.
- If the exception is caught in the message flow by a TryCatch node, the message and local environment trees that were previously propagated through the Try terminal of the TryCatch node are restored and propagated through the Catch terminal. Any updates that you made to their content in the nodes that followed the TryCatch node are lost. The environment tree is not restored, and its contents are preserved. If the nodes following the TryCatch node include a Compute node that creates a new local environment or message tree, those trees are lost. The exception list tree reflects the one or more exceptions that have been recorded.

Exception handling paths in a message flow: Exception handling paths start at a failure terminal (most message processing nodes have these), the Catch terminal of an input node, a TryCatch node, or an AggregateReply node, but are no different in principle from a normal message flow path. Such a flow consists of a sequence of nodes connected together by the designer of the message flow. The exception handling paths differ in the kind of processing that they do to record or react to the exception. For example, they might examine the exception list to determine the nature of the error, and take appropriate action or log data from the message or exception.

The local environment and message tree that are propagated to the exception handling message flow path are those at the start of the exception path, not those at the point when the exception is thrown. The following figure illustrates this point:



- A message (M1) and local environment (L1) are being processed by a message flow. They are passed through the TryCatch node to Compute1.
- Compute1 updates the message and local environment and propagates a new message (M2) and local environment (L2) to the next node, Compute2.

- An exception is thrown in Compute2. If the failure terminal of Compute2 is not connected (point **B**), the exception is propagated back to the TryCatch node, but the message and local environment are not. The exception handling path starting at point **A** has access to the first message and local environment, M1 and L1. The environment tree is also available and retains the content it had when the exception occurred.
- If the failure terminal of Compute2 is connected (point **B**), the message and local environment M2 and L2 are propagated to the node connected to that failure terminal. The environment tree is also available and retains the content it had when the exception occurred.

Related concepts:

"Message tree structure" on page 1045
The message tree is a part of the logical message tree in which the broker stores its internal representation of the message body.

"Environment tree structure" on page 1055
The environment tree is a part of the logical message tree in which you can store information while the message passes through the message flow.

"Local environment tree structure" on page 1056
The local environment tree is a part of the logical message tree in which you can store information while the message flow processes the message.

"Exception list tree structure" on page 1066
The exception list tree is a part of the logical message tree in which the message flow writes information about exceptions that occur when a message is processed.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Accessing the ExceptionList tree using ESQL" on page 2471
The ExceptionList tree has its own correlation name, ExceptionList, and you must use this in all ESQL statements that refer to or set the content of this tree.

"Throwing an exception" on page 2511
If you detect an error or other situation in your message flow in which you want message processing to be ended, you can throw an exception in a message flow in two ways.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"AggregateReply node" on page 4299
Use the AggregateReply node to mark the end of an aggregation fan-in. This node collects replies and combines them into a single compound message.

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"TryCatch node" on page 4949
Use the TryCatch node to provide a special handler for exception processing.

Environment tree structure:

The environment tree is a part of the logical message tree in which you can store information while the message passes through the message flow.

The root of the environment tree is called Environment. This tree is always present in the input message; an empty environment tree is created when a message is received and parsed by the input node. You can use this tree as you choose, and create both its content and structure.

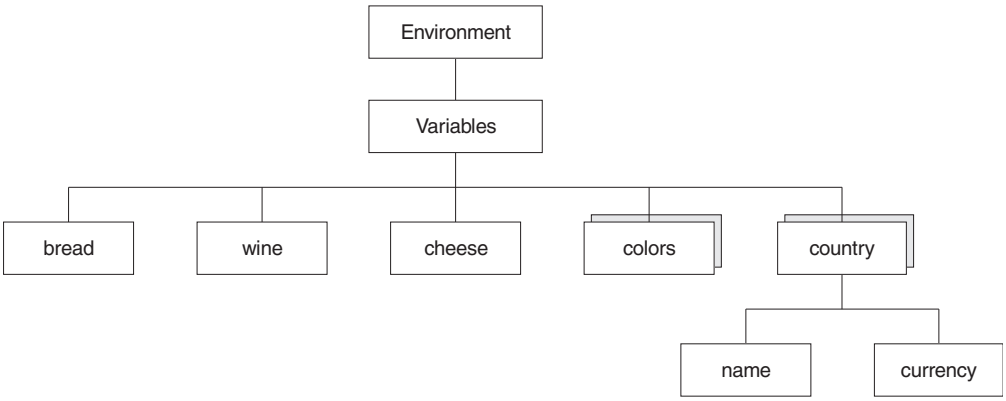
WebSphere Message Broker uses fields in the Environment tree in only two situations. (Contrast this with the "Local environment tree structure" on page 1056, which the broker uses in many situations):

- If you have requested data collection for message flow accounting and statistics, and have indicated that accounting origin basic support is required, the broker checks for the existence of the field Environment.Broker.AccountingOrigin. If the field exists, the broker uses its value to set the accounting origin for the current data record. For further information about the use of this field, see "Setting message flow accounting and statistics accounting origin" on page 3290.
- If you have activated a message flow to emit monitoring events the broker stores correlation attributes in the Environment tree. For further information, see "Correlation and monitoring events" on page 6778

The environment tree differs from the local environment tree in that a single instance of it is maintained throughout the message flow. If you include a Compute node, a Mapping node, or a JavaCompute node in your message flow, you do not have to specify whether you want the environment tree to be included in the output message. The environment tree is included automatically, and the entire contents of the input environment tree are retained in the output environment tree, subject to any modifications that you make in the node. Any changes that you make are available to subsequent nodes in the message flow, and to previous nodes if the message flows back (for example, to a FlowOrder or TryCatch node).

If you want to create your own information, create it in the environment tree in a subtree called Variables.

The following figure shown an example of an environment tree:



You could use the following ESQL statements to create the content shown above.

```
SET Environment.Variables =
      ROW('granary' AS bread, 'riesling' AS wine, 'stilton' AS cheese);
SET Environment.Variables.Colors[] =
      LIST{'yellow', 'green', 'blue', 'red', 'black'};
SET Environment.Variables.Country[] = LIST{ROW('UK' AS name, 'pound' AS currency),
     ROW('USA' AS name, 'dollar' AS currency)};
```

When the message flow processing is complete, the Environment tree is discarded.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"Local environment tree structure"
The local environment tree is a part of the logical message tree in which you can store information while the message flow processes the message.

"Correlation names" on page 1069
A correlation name is a field reference that identifies a well-defined starting point in the logical message tree and is used in field references to describe a standard part of the tree format.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Accessing the environment tree" on page 2469
The environment tree has its own correlation name, Environment, and you must use this name in all ESQL statements that refer to, or set, the content of this tree.

"Setting message flow accounting and statistics accounting origin" on page 3290
When you request accounting origin support for collecting message flow accounting and statistics data on the **mqsichangeflowstats** command, you must also configure your message flows to provide the correct identification values that indicate what the data is associated with.

"Writing ESQL" on page 2413
How you can use ESQL to customize nodes.

"Writing Java" on page 2638
When you create a message flow, you include input nodes that receive messages and, optionally, output nodes that send out new or updated messages. If the message processing requires it, you can include other nodes after the input node that are customized in Java to complete the actions that your applications need.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Local environment tree structure:

The local environment tree is a part of the logical message tree in which you can store information while the message flow processes the message.

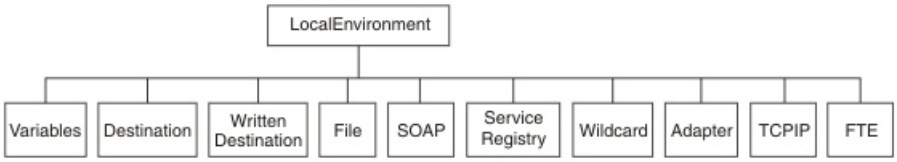
The root of the local environment tree is called `LocalEnvironment`. This tree is always present in the input message, it is created when a message is received by the input node. Some input nodes create local environment fields, others leave it empty.

The local environment tree is made up of the following structure:

- Anything in the format of `LocalEnvironment.Destination.`*`output_or_request_node_name`*, for example; LocalEnvironment.Destination.Email, decides what happens when information is going into an output or request node.
- Anything in the format of `LocalEnvironment.WrittenDestination.`*`output_or_request_node_name`*, for example; LocalEnvironment.WrittenDestination.FTE, gives you information about the processed output of an output or a request node.
- Anything in the format of `LocalEnvironment.`*`input_node_name`*`.input`, for example; LocalEnvironment.Adapter.Input, contains information that has been stored by an input node.

Use the local environment tree to store variables that can be referred to and updated by message processing nodes that occur later in the message flow. You can also use the local environment tree to define destinations (that are internal and external to the message flow) to which a message is sent. WebSphere Message Broker also stores information in LocalEnvironment in some circumstances, and references it to access values that you might have set for destinations. (Contrast this to the Environment tree structure, which the broker uses only in specific situations, see "Environment tree structure" on page 1055.)

The following figure shows an example of the local environment tree structure. The children of Destination are protocol-dependent.



In the tree structure shown, LocalEnvironment has several children:

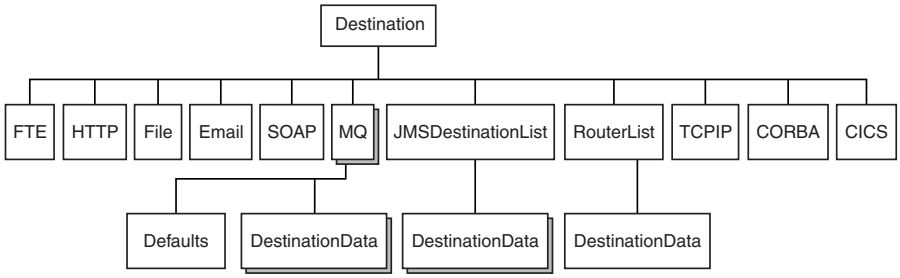
LocalEnvironment.Variables

This subtree is optional. If you create local environment variables, store them in a subtree called Variables. It provides a work area that you can use to pass information between nodes. This subtree is never inspected or modified by any supplied node.

Variables in the local environment can be changed by any subsequent message processing node, and the variables persist until the node that created them goes out of scope.

The variables in this subtree are persistent only within a single instance of a message flow. If you have multiple instances of a message passing through the message flow, and need to pass information between them, you must use an external database.

LocalEnvironment.Destination



This subtree consists of a number of children that indicate the transport types to which the message is directed (the Transport identifiers), or the target Label nodes that are used by a RouteToLabel node.

- Transport information

  Transport information is used by some input and output nodes, including Email, File, FTE, HTTP, JMS, MQ, SOAP, and TCPIP.

  LocalEnvironment.Destination.CICS

  If the message flow includes a CICSRequest node, you can override the following properties with elements in this subtree:
  - `Program name`
  - `Commarea length`
  - `Mirror transaction ID`
  - `Set EIBTRNID only`
  - `Message domain`
  - `Message set`
  - `Message type`
  - `Message format`
  - `Message coded character set ID`
  - `Message encoding`

  For more information, see "Local environment overrides for the CICSRequest node" on page 2191.

  You can also set LocalEnvironment values for CICS channels and containers. For more information, see "COMMAREA or channel data structures" on page 2183.

  LocalEnvironment.Destination.CORBA

  If the message flow includes a CORBARequest node, you can override its `Operation name` property by specifying a value in the following location:

  ```
  $LocalEnvironment/Destination/CORBA/Request/OperationName
  ```

  For more information about the operation name, see "CORBARequest node" on page 4349.

  LocalEnvironment.Destination.Email

  If the message flow includes an EmailOutput node, then information defined in this subtree will specify or override the SMTP server connection information and attachments associated with each email sent by the node. Multiple attachments can be specified for inclusion in the email sent, including the specification of the attachment name, content, and type. See "EmailOutput node" on page 4400.

  LocalEnvironment.Destination.File

If the message flow includes a FileOutput node, you can override its `directory` and `name` properties with elements in this subtree. See "Using local environment variables with file nodes" on page 1820.

LocalEnvironment.Destination.FTE

If the message flow includes an FTEOutput node, you can override its properties with elements in this subtree. See "Using local environment variables with file nodes" on page 1820.

LocalEnvironment.Destination.HTTP

If the message flow starts with an HTTPInput node, a single name element HTTP is added to Destination. The element HTTP.RequestIdentifier is created and initialized so that it can be used by an HTTPReply node. You can also create other fields in the HTTP structure for use by the HTTPRequest node; for example, the URL of the service to which the request is sent. For more information, see *Local environment overrides* in the topic "HTTPRequest node" on page 4488, and examples in "Populating Destination in the local environment tree" on page 2467.

LocalEnvironment.Destination.JMSDestinationList

A JMSOutput node can be configured to send to multiple JMS Queues or to publish to multiple JMS Topics using a destination list created in the local environment by a transformation node.

The JMSOutput node searches the local environment for data elements called DestinationData under the folder Destination.JMSDestinationList. The node sends the JMS message to each DestinationData entry found in that folder. See the example in "Populating Destination in the local environment tree" on page 2467.

LocalEnvironment.Destination.MQ

If the message flow includes an MQOutput node, each element is a name element, *MQ* (A deprecated alternative exists, called *MQDestinationList*. Use *MQ* for all new message flows). If more than one element exists, each is processed sequentially by the node. See the example in "Populating Destination in the local environment tree" on page 2467.

You can configure MQOutput nodes to examine the list of destinations and send the message to those destinations, by setting the property `Destination Mode` to Destination List. If you do so, you must create this subtree and its contents to define those destinations, giving it the name Destination. If you do not do so, the MQOutput node cannot deliver the messages.

If you prefer, you can configure the MQOutput node to send messages to a single fixed destination, by setting the property `Destination Mode` to Queue Name or Reply To Queue. If you select either of these fixed options, the destination list has no effect on broker operations and you do not have to create this subtree.

You can construct the MQ element to contain a single optional Defaults element. The Defaults element, if created, must be the first child and must contain a set of name-value elements that give default values for the message destination and its PUT options for that parent.

You can also create a number of elements called DestinationData within MQ. Each of these can be set up with a set of name-value elements that defines a message destination and its PUT options.

The set of elements that define a destination is described in "Data types for elements in the MQ DestinationData subtree" on page 4240.

The content of each instance of DestinationData is the same as the content of Defaults for each protocol, and can be used to override the default values in Defaults. You can set up Defaults to contain values that are common to all destinations, and set only the unique values in each DestinationData subtree. If you do not set a value either in DestinationData or Defaults, the value that you have set for the corresponding node property is used. Similarly, if you specify a field name or value with the wrong spelling or case, it is ignored, and the value that you have set for the corresponding node property is used.

The information that you insert into DestinationData depends on the characteristic of the corresponding node property: This information is described in "Accessing the local environment tree" on page 2463.

LocalEnvironment.Destination.SOAP

You can place outbound WS-Addressing header information in the local environment to override the defaults that are generated by the SOAPReply, SOAPRequest, or SOAPAsyncRequest nodes. See "WS-Addressing information in the local environment" on page 1656.

If the message flow includes SOAPRequest or SOAPAsyncRequest nodes, you can override their HTTP Transport and JMS transport properties in this subtree. See "SOAPRequest node" on page 4828, "SOAPAsyncRequest node" on page 4750, or "Local environment overrides for the SOAPRequest node" on page 4850.

If the message flow includes a SOAPAsyncRequest node, you can use this subtree to pass state and correlation information to a SOAPAsyncResponse node in another message flow. See "WS-Addressing with the SOAPAsyncRequest and SOAPAsyncResponse nodes" on page 1655.

If the message flow includes SOAPReply, SOAPRequest, or SOAPAsyncRequest nodes, you can override their use of outbound MTOM messages in this subtree. See "Using SOAP MTOM with the SOAPReply, SOAPRequest, and SOAPAsyncRequest nodes" on page 1678.

LocalEnvironment.Destination.TCPIP

If the message flow includes a TCPIPClientOutput node or a TCPIPServerOutput node, you can override its TCPIP connection with elements in this subtree. See "TCPIPClientOutput node" on page 4867 and "TCPIPServerOutput node" on page 4903.

- Routing information

  The child of Destination is *RouterList*. It has a single child element called *DestinationData*, which has a single entry called labelName. If you are using a dynamic routing scenario involving the RouteToLabel and Label nodes, you must set up the Destination subtree with a RouterList that contains the reference labels.

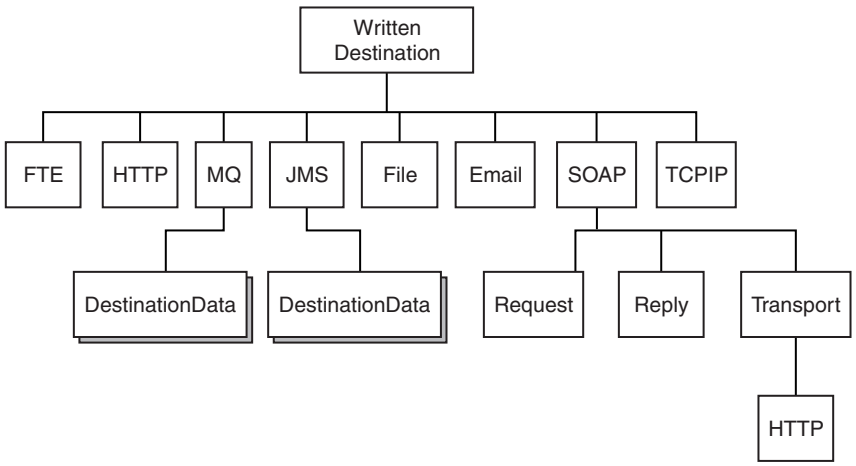
LocalEnvironment.Wildcard

This subtree contains information about the wildcard characters that are stored by the FileInput node.

On the FileInput node you can specify a file name pattern that contains wildcard characters.

More details about the information that is stored in this subtree are in "Using local environment variables with file nodes" on page 1820.

LocalEnvironment.WrittenDestination



This subtree contains the addresses to which the message has been written. Its name is fixed and it is created by the message flow when a message is propagated through the Out terminal of a request, output, or reply node. The subtree includes transport-specific information (for example, if the output message has been put to a WebSphere MQ queue, it includes the queue manager and queue names).

You can use one of the following methods to obtain information about the details of a message after it has been sent by the nodes:

- Connect a transformation node to the Out terminal.
- Configure a user exit to process an output message callback event, as described in "Exploiting user exits" on page 2985.

The topic for each node that supports WrittenDestination information contains details about the data that it contains.

LocalEnvironment.Adapter

This subtree contains information that is stored by the WebSphere Adapters nodes.

For a WebSphere Adapters input node:

- **MethodName** is the name of the business method that corresponds to the Enterprise Information System (EIS) event that triggered this message delivery.

  The bindings for EIS events or business methods are created by the Adapter Connection wizard.
- **Type** describes the type of adapter that is being used:
  - SAP
  - Siebel
  - PeopleSoft
  - JD Edwards

For a WebSphere Adapters request node:

MethodName is the name of the business method that the request node must use.

LocalEnvironment.CD and LocalEnvironment.CD.Transfer

These subtrees contain information that is stored by the CDInput node. The LocalEnvironment.CD subtree contains information about the current

record. The LocalEnvironment.CD.Transfer subtree contains information received from IBM Sterling Connect:Direct regarding the file.

More details about the information that is stored in these subtrees are in "Using local environment variables with file nodes" on page 1820.

LocalEnvironment.Database
This subtree contains information that is propagated from the DatabaseInput node.

The LocalEnvironment.Database.Input.Event.Usr subtree contains user-defined data associated with an event. It is initialized in the ReadEvents procedure of the ESQL module associated with the DatabaseInput node.

LocalEnvironment.Database.Input.Event.Key holds a unique key for an event. It is set in the ReadEvents procedure of the ESQL module associated with the DatabaseInput node.

LocalEnvironment.Database.Input.Event.FailureCount contains a value for the number of times an attempt to process an event has failed. This count includes all unhandled exceptions occurring either in the ESQL module or the message flow.

LocalEnvironment.File
This subtree contains information that is stored by the FileInput node.

This information describes the file, and also contains data about the current record.

More details about the information that is stored in this subtree are in "Using local environment variables with file nodes" on page 1820.

LocalEnvironment.FTE and LocalEnvironment.FTE.Transfer
These subtrees contain information that is stored by the FTEInput node. The LocalEnvironment.FTE subtree contains information about the current record. The LocalEnvironment.FTE.Transfer subtree contains information received from WebSphere MQ File Transfer Edition regarding the file.

More details about the information that is stored in these subtrees are in "Using local environment variables with file nodes" on page 1820.

LocalEnvironment.ServiceRegistry
This subtree contains information for queries by the EndpointLookup and RegistryLookup nodes.

More details about the information that is stored in this subtree are in "Dynamically defining the search criteria" on page 1891, "EndpointLookup node output" on page 1894, and "RegistryLookup node output" on page 1897.

LocalEnvironment.SOAP
This subtree contains information that is stored by SOAPInput, SOAPAsyncResponse, or SOAPRequest nodes.

More details about the information that is stored in this subtree are in "WS-Addressing information in the local environment" on page 1656.

If the message flow includes a SOAPAsyncResponse node, you can use this subtree to receive state and correlation information passed by a SOAPAsyncRequest node in another message flow.

More details about the information that is stored in this subtree are in "WS-Addressing with the SOAPAsyncRequest and SOAPAsyncResponse nodes" on page 1655.

LocalEnvironment.TCPIP

If the message flow includes a TCPIPClientReceive node or a TCPIPServerReceive node, you can override its TCPIP connection with elements in this subtree. See "TCPIPClientReceive node" on page 4877 and "TCPIPServerReceive node" on page 4913.

This subtree contains information that is stored by the TCPIPClientInput, TCPIPClientReceive, TCPIPServerInput, and TCPIPServerReceive nodes.

This information describes the connection that the node is using.

More details about the information that is stored in this subtree are in "TCPIPClientInput node" on page 4854, "TCPIPClientReceive node" on page 4877, "TCPIPServerInput node" on page 4890, and "TCPIPServerReceive node" on page 4913.

When the message flow processing is complete, the local environment tree is discarded.

The following samples demonstrate how to use the local environment to dynamically route messages based on the destination list:

- Airline Reservations
- Message Routing
- Managed File Transfer

The following samples use the local environment tree to store information that is later added to the output message that is created by the message flow:

- User-defined Extension
- Managed File Transfer

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"Environment tree structure" on page 1055
The environment tree is a part of the logical message tree in which you can store information while the message passes through the message flow.

"Using local environment variables with file nodes" on page 1820
You can use fields in the local environment to dynamically alter the behavior of the FileInput, FileOutput, FTEInput, and FTEOutput nodes. You can also find what values the output nodes used to process the file.

"WS-Addressing information in the local environment" on page 1656
WS-Addressing header information can be placed in the local environment tree where it is visible to a message flow. WS-Addressing header information is only processed by the SOAP nodes.

"Dynamically defining the search criteria" on page 1891
You can use the RegistryLookup and EndpointLookup nodes to issue WebSphere Service Registry and Repository (WSRR) queries specified in the local environment.

"EndpointLookup node output" on page 1894
Use an EndpointLookup node to retrieve the endpoint addresses for WSDL service definitions held in WebSphere Service Registry and Repository (WSRR).

"RegistryLookup node output" on page 1897
Use the RegistryLookup node to retrieve any type of entity held in the WebSphere Service Registry and Repository (WSRR). The entities that match the specified search criteria are stored in the local environment. The input message is not modified.

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Creating destination lists" on page 1477
Create a list of destinations to indicate where a message is sent.

"Accessing the local environment tree" on page 2463
The local environment tree has its own correlation name, LocalEnvironment, and you must use this name in all ESQL statements that refer to or set the content of this tree.

"Using scratchpad areas in the local environment" on page 2465
The local environment tree includes a subtree called variables. This subtree is always created, but is never populated by the message flow. Use this area for your own purposes; for example, to pass information from one node to another. You can create other subtrees of the local environment tree.

"Populating Destination in the local environment tree" on page 2467
Use the Destination subtree to set up the target destinations that are used by output nodes, the HTTPRequest node, the SOAPRequest node, the SOAPAsyncRequest node, and the RouteToLabel node. The following examples show how you can create and use an ESQL procedure to perform the task of setting up values for each of these uses.

"Writing ESQL" on page 2413
How you can use ESQL to customize nodes.

"Writing Java" on page 2638
When you create a message flow, you include input nodes that receive messages and, optionally, output nodes that send out new or updated messages. If the message processing requires it, you can include other nodes after the input node that are customized in Java to complete the actions that your applications need.

Related reference:

"CICSRequest node" on page 4321
Use the CICSRequest node to call CICS Transaction Server for z/OS programs over TCP/IP-based IP InterCommunications (IPIC) protocol.

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"CORBARequest node" on page 4349
Use the CORBARequest node to call an external CORBA application over Internet Inter-Orb Protocol (IIOP).

Exception list tree structure:

The exception list tree is a part of the logical message tree in which the message flow writes information about exceptions that occur when a message is processed.

The root of the exception list tree is called ExceptionList, and the tree consists of a set of zero or more exception descriptions. The exception list tree is populated by the message flow if an exception occurs. If no exception conditions occur during message flow processing, the exception list that is associated with that message consists of a root element only. This list is, in effect, an empty list of exceptions.

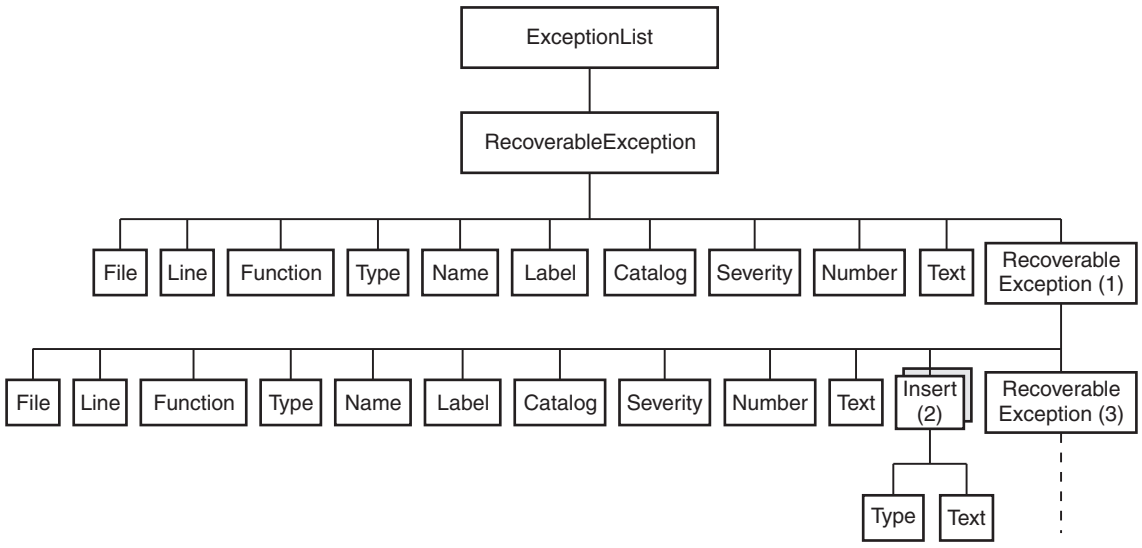
The exception list tree can be accessed by other nodes in the message flow that receive the message after the exception has occurred. You can modify the contents of the exception list tree only in a node that provides an interface to modify the outbound message tree; for example, the Compute node.

If an exception condition occurs, message processing is suspended and an exception is thrown. Control is passed back to a higher level; that is, an enclosing catch block. An exception list is built to describe the failure condition, and the whole message, together with the local environment tree, and the newly-populated exception list, is propagated through an exception-handling message flow path.

The child of ExceptionList may be any one of the exception types that is included in the following list. Typically, only one child of the root is created, although more than one might be generated in some circumstances. The child of ExceptionList contains a number of children, each of which may also be any one of the types in the following list. The last of these children provides further information specific to the type of exception.

- FatalException
- RecoverableException
- ConfigurationException
- SecurityException
- ParserException
- ConversionException
- DatabaseException
- UserException
- CastException
- MessageException
- SqlException
- SocketException
- SocketTimeoutException
- UnknownException

The following figure shows the structure of the exception list tree for a recoverable exception:



The exception description structure can be both repeated and nested to produce an exception list tree. In this tree:

- The depth (that is, the number of parent-child steps from the root) represents increasingly detailed information for the same exception.
- The width of the tree represents the number of separate exception conditions that occurred before processing was abandoned. This number is usually one, and results in an exception list tree that consists of a number of exception descriptions that are connected as children of each other.
- At the numbered points in the tree:
  1. This child can be any one of the exception types that are listed earlier in this topic. All of these elements have the children shown; if present, the last child is the same element as its parent.
  2. This element might be repeated.
  3. If present, this child contains the same children as its parent.

The children in the tree take the form of a number of name-value elements that give details of the exception, and zero or more name elements whose name is Insert. The NLS (National Language Support) message number identified in a name-value element identifies a WebSphere Message Broker error message. The Insert values are used to replace the variables in this message and provide further detail about the cause of the exception.

The name-value elements in the exception list shown in the figure above are described in the following table.

Name	Type	Description
File[1]	String	C++ source file name
Line[1]	Integer	C++ source file line number
Function[1]	String	C++ source function name
Type[2]	String	Source object type
Name[2]	String	Source object name
Label[2]	String	Source object label
Text[1]	String	Optional non-NLS text

Name		Type	Description
Catalog[3]		String	NLS message catalog name[4]
Severity[3]		Integer	1 = information 2 = warning 3 = error
Number[3]		Integer	NLS message number[4]
Insert[3]	Type	Integer	The data type of the value: 0 = Unknown 1 = Boolean 2 = Integer 3 = Float 4 = Decimal 5 = Character 6 = Time 7 = GMT Time 8 = Date 9 = Timestamp 10 = GMT Timestamp 11 = Interval 12 = BLOB 13 = Bit Array 14 = Pointer
	Text	String	The data value

Notes:

1. Do not use the File, Line, Function, and Text elements for exception handling decision making. These elements ensure that information can be written to a log for use by IBM Service personnel, and are subject to change in both content and order.
2. The Type, Name, and Label elements define the object (usually a message flow node) that was processing the message when the exception condition occurred.
3. The Catalog, Severity, and Number elements define an NLS message: the Insert elements that contain the two name-value elements shown define the inserts into that NLS message.
4. NLS message catalog name and NLS message number refer to a translatable message catalog and message number.

When the message flow processing is complete, the exception list tree is discarded.

The following sample uses the exception list in the XML_Reservation message flow to pass error information to the Throw node, which generates an error message that includes the information from ExceptionList:

- Airline Reservations

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an

input message is received.

"Message tree contents after an exception" on page 1052
The contents of the message tree are updated if an exception is raised.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Accessing the ExceptionList tree using ESQL" on page 2471
The ExceptionList tree has its own correlation name, ExceptionList, and you must use this in all ESQL statements that refer to or set the content of this tree.

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Throwing an exception" on page 2511
If you detect an error or other situation in your message flow in which you want message processing to be ended, you can throw an exception in a message flow in two ways.

"Writing ESQL" on page 2413
How you can use ESQL to customize nodes.

"Writing Java" on page 2638
When you create a message flow, you include input nodes that receive messages and, optionally, output nodes that send out new or updated messages. If the message processing requires it, you can include other nodes after the input node that are customized in Java to complete the actions that your applications need.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

Correlation names:

A correlation name is a field reference that identifies a well-defined starting point in the logical message tree and is used in field references to describe a standard part of the tree format.

When you access data in any of the four trees (message, environment, local environment, or exception list), the correlation names that you can use depend on the node for which you create ESQL or mappings, and whether the node creates an output message. For example, a Trace node does not alter the content of the message as it passes through the node, but a Compute node can construct a new output message.

You can introduce new correlation names with SELECT expressions, quantified predicates, and FOR statements. You can create non-correlation names in a node by using reference variables.

Correlation names in nodes that do not create an output message: Most message flow nodes do not create an output message; all ESQL expressions that you write in ESQL modules or in mappings in these nodes refer to just the input message. Use the following correlation names in the ESQL modules that you write for Database and Filter nodes:

Root The root of the message passing through the node.

Body The last child of the root of the message; that is, the body of the message. This name is an alias for `Root.*[<]`.

For a description of how to use the asterisk (*) in field references, see "Using anonymous field references" on page 2430.

Properties
The standard properties of the input message.

Environment
The structure that contains the current global environment variables that are available to the node. Environment can be read and updated from any node for which you can create ESQL code or mappings.

LocalEnvironment
The structure that contains the current local environment variables that are available to the node. LocalEnvironment can be read and updated from any node for which you can create ESQL code or mappings.

DestinationList
The structure that contains the current local environment variables available to the node. Its preferred name is LocalEnvironment, although the DestinationList correlation name can be used for compatibility with earlier versions.

ExceptionList
The structure that contains the current exception list to which the node has access.

You cannot use these correlation names in the expression of any mapping for a Mapping, Extract, Warehouse, DataInsert, DataUpdate, or DataDelete node.

Correlation names in nodes that create an output message: If you are coding ESQL for a Compute node, the correlation names must distinguish between the two message trees involved: the input message and the output message. The correlation names in ESQL in these nodes are:

InputBody
The last child of the root of the input message. This name is an alias for `InputRoot.*[<]`.

For a description of how to use *, see "Using anonymous field references" on page 2430.

InputRoot
The root of the input message.

InputProperties
The standard properties of the input message.

Environment
The structure that contains the current global environment variables that are available to the node. Environment can be read and updated.

InputLocalEnvironment
The structure that contains the local environment variables for the message passing through the node.

InputDestinationList
The structure that contains the local environment variables for the message passing through the node. Use the correlation name InputDestinationList for compatibility with earlier versions; if compatibility is not required, use the preferred name InputLocalEnvironment

InputExceptionList
The structure that contains the exception list for the message passing through the node.

OutputRoot
The root of the output message.

In a Compute node, the correlation name OutputBody is not valid.

OutputLocalEnvironment
The structure that contains the local environment variables that are sent out from the node.

While this correlation name is always valid, it has meaning only when the `Compute Mode` property of the Compute node indicates that the Compute node is propagating the LocalEnvironment.

OutputDestinationList
The structure that contains the local environment variables that are sent out from the node. Use the correlation name OutputDestinationList for compatibility with earlier versions; if compatibility is not required, use the preferred name OutputLocalEnvironment

OutputExceptionList
The structure that contains the exception list that the node is generating.

While this correlation name is always valid, it has meaning only when the `Compute Mode` property of the Compute node indicates that the Compute node is propagating the ExceptionList.

Related concepts:

"Environment tree structure" on page 1055
The environment tree is a part of the logical message tree in which you can store information while the message passes through the message flow.

"Local environment tree structure" on page 1056
The local environment tree is a part of the logical message tree in which you can store information while the message flow processes the message.

"Exception list tree structure" on page 1066
The exception list tree is a part of the logical message tree in which the message flow writes information about exceptions that occur when a message is processed.

"ESQL field references" on page 2381
An ESQL field reference is a sequence of period-separated values that identify a specific field (which might be a structure) within a message tree or a database table.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

Chapter 9, “Developing message flow applications,” on page 1019
Develop message flows to process your business messages and data.

“Manipulating message body content” on page 2418
The message body is always the last child of root, and is identified by its parser name, for example XML or MRM. You can refer to, modify, and create message body data.

“Manipulating other parts of the message tree” on page 2452
You can access message tree headers, the properties tree, the local environment tree, the environment tree and the exception list tree.

“Using anonymous field references” on page 2430
You can refer to the array of all children of a particular element by using a path element of *.

Related reference:

“Built-in nodes” on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

“Compute node” on page 4340
Use the Compute node to construct one or more new output messages.

“Database node” on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

“Filter node” on page 4452
Use the Filter node to route a message according to message content.

Parsers

A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from the internal message tree representation.

A parser is called when the bit stream that represents an input message is converted to the internal form that can be handled by the broker; this invocation of the parser is known as *parsing*. The internal form, a logical tree structure, is described in “Logical tree structure” on page 1042. It is described as a tree because messages are typically hierarchical in structure; a good example of this structure is XML. The way in which the parser interprets the bit stream is unique to that parser; therefore, the logical message tree that is created from the bit stream varies from parser to parser.

The parser that is called depends on the structure of a message, referred to as the *message template*. Message template information comprises the *message domain*, *message set*, *message type*, and *physical format* of the message. Together, these values identify the structure of the data that the message contains.

A parser is also called when a logical tree that represents an output message is converted into a bit stream; this action by the parser is known as *writing*. Typically, an output message is generated by an output node at the end of the message flow. However, you can connect more nodes to an output node to continue processing of the message.

The message domain identifies the parser that is used to parse and write instances of the message. The remaining parts of the message template, message set, message type, and physical format, are optional, and are used by model-driven parsers such as the MRM parser.

The logical structure of the message typically maps to the business content of the message; for example, it contains a customer name, address, and account number. It is only when you send a message across a connection that the physical characteristics are important, and influence the construction of the bit stream.

The broker requires access to a parser for every message domain to which your input messages and output messages belong. In addition, the broker requires a parser for every identifiable message header that is included in the input or output message. Parsers are called when required by the message flow.

Body parsers

WebSphere Message Broker provides built-in support for messages in the following message domains by providing message body parsers:

- MRM ("MRM parser and domain" on page 1111)
- XMLNSC, XMLNS, and XML ("XML parsers and domains" on page 1084)
- SOAP ("SOAP parser and domain" on page 1082)
- DataObject ("DataObject parser and domain" on page 1114)
- JMSMap and JMSStream ("JMS parsers and domains" on page 1116)
- MIME ("MIME parser and domain" on page 1117)
- BLOB ("BLOB parser and domain" on page 1124)
- IDOC ("IDOC parser and domain" on page 1126)
- JSON ("JSON parser and domain" on page 1128)

See "Which body parser should you use?" on page 1077 for a discussion about which message body parser to use under what circumstances.

You specify which message domain to use for your message at the place in the message flow where parsing or writing is initiated.

- To parse a message bit stream, typically you set the *Message Domain* property of the input node that receives the message. But, if you are initiating the parse operation in ESQL, use the **DOMAIN** clause of the **CREATE** statement.

  The message tree that is created is described in "Message tree structure" on page 1045. Its exact form might change as it progresses through the message flow, depending on what the nodes are doing.

  The last child element of the Root element of the message tree takes the name of the body parser that created the tree. For example, if the *Message Domain* property was set to `MRM`, the last child element of Root is called MRM, which indicates that the message tree is owned by the MRM parser.
- To write a message, the broker calls the owning body parser to create the message bit stream from the message tree.

Some body parsers are *model-driven*, which means that they use predefined messages from a message set when parsing and writing. The MRM, SOAP, DataObject, IDOC, and (optionally) XMLNSC parsers are model-driven parsers. To use these parsers, messages must be modeled in a message set and deployed to the broker from the WebSphere Message Broker Toolkit.

Other body parsers are *programmatic*, which means that the messages that they parse and write are *self-defining* messages, and no message set is required. See "Predefined and self-defining messages" on page 1076.

When you use a model-driven parser, you must also specify the *Message Set* and, optionally, the *Message Type* and *Message Format* so that the parser can locate the deployed message definition with which to guide the parsing or writing of the message.

To parse a message bit stream, typically you set the *Message Set*, *Message Type*, and *Message Format* properties of the input node that receives the message. Or, if you are initiating the parse operation in ESQL, you use the **SETTYPE**, and **FORMAT** clauses of the **CREATE** statement. This information is copied into the *Properties* folder of the message tree.

To write a message, the broker calls the owning body parser to create the message bit stream from the message tree. If the parser is a model-driven parser, it uses the *MessageSet*, *MessageType*, and *MessageFormat* fields in the Properties folder.

Whether *Message Type* or *Message Format* are needed depends on the message domain.

Even if the body parser is not model-driven, it is good practice to create and use a message set in the WebSphere Message Broker Toolkit, because it simplifies the development of your message flow applications, even though the message set is not deployed in the broker runtime environment. See "Why model messages?" on page 1158 for information about the advantages of creating a message set.

Header parsers

WebSphere Message Broker also provides parsers for the following message headers, which your applications can include in input or output messages:

- WMQ MQMD ("The MQMD parser" on page 4249)
- WMQ MQMDE ("The MQMDE parser" on page 4251)
- WMQ MQCFH ("The MQCFH parser" on page 4245)
- WMQ MQCIH ("The MQCIH parser" on page 4246)
- WMQ MQDLH ("The MQDLH parser" on page 4248)
- WMQ MQIIH ("The MQIIH parser" on page 4248)
- WMQ MQRFH ("The MQRFH parser" on page 4252)
- WMQ MQRFH2 and MQRFH2C ("The MQRFH2 and MQRFH2C parsers" on page 4253)
- WMQ MQRMH ("The MQRMH parser" on page 4253)
- WMQ MQSAPH ("The MQSAPH parser" on page 4254)
- WMQ MQWIH ("The MQWIH parser" on page 4255)
- WMQ SMQ_BMH ("The SMQ_BMH parser" on page 4256)
- JMS header ("Representation of messages in the JMS Transport" on page 1691)
- HTTP headers ("HTTP headers" on page 1583)

All header parsers are programmatic and do not use a message set when parsing or writing.

User-defined parsers

To parse or write message body data or headers that the supplied parsers do not handle, you can create user-defined parsers that use the WebSphere Message Broker user-defined parser programming interface.

Tip: No parser is provided for messages, or parts of messages, in the WMQ format `MQFMT_IMS_VAR_STRING`. Data in this format is often preceded by an MQIIH header (format `MQFMT_IMS`). WebSphere Message Broker treats such data as a BLOB message. If you change the CodedCharSetId or the encoding of such a message in a message flow, the `MQFMT_IMS_VAR_STRING` data is not converted, and the message descriptor or preceding header does not correctly describe that part of the message. If you need the data in these messages to be converted, use the MRM domain and create a message set to model the message content, or provide a user-defined parser.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

"Why model messages?" on page 1158
WebSphere Message Broker supplies a range of parsers to parse and write message formats. Some message formats are self-defining and can be parsed without reference to a model. Most message formats, however, are not self-defining, and the parser must have access to a predefined model that describes the message, if it is to parse it correctly.

"Message domains and parsers" on page 1159
WebSphere Message Broker supplies a range of *parsers* to parse and write message formats.

"Which body parser should you use?" on page 1077
The characteristics of the messages that your applications exchange indicate which body parser you must use.

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"Developing user-defined extensions" on page 2970
A user-defined extension is a component that you design and implement to extend the function of WebSphere Message Broker.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"Parsing on demand" on page 4173
On-demand parsing, referred to as partial parsing, is used to parse an input message bit stream only as far as is necessary to satisfy the current reference. The parsers that can perform partial parsing of input messages are the JSON, MRM, XML, XMLNS, and XMLNSC parsers.

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"C language user-defined parser API" on page 6538
The C language user-defined parser API consists of two complementary sets of functions that you can use to define the function of your parser.

Predefined and self-defining messages:

Both predefined and self-defining messages are supported.

Each message that flows through a broker has a specific structure that is meaningful to the applications that send or receive that message.

You can use:

- Messages that you have modeled in the Broker Application Development perspective of the WebSphere Message Broker Toolkit; these messages are referred to as *predefined messages*. A model-driven parser requires predefined messages.
- Messages that can be parsed without a model; these are called *self-defining messages*.

Predefined messages: When you create a message in the WebSphere Message Broker Toolkit, you define the fields (*Elements*) in the message, along with special field types that you might need, and specific values (Value Constraints) to which the fields might be restricted.

Every message that you model in the WebSphere Message Broker Toolkit must be a member of a message set. You can group related messages together in a message set; for example, request and response messages for a bank account query can be defined in a single message set.

When you deploy a message set to a broker, the message model is sent to the broker in a form appropriate to the parser that is used to parse and write the message.

For information about the benefits of predefining messages, see "Why model messages?" on page 1158

The following samples demonstrate how to model messages in XML, CWF and TDS formats:

- Video Rental
- Comma Separated Value (CSV)

The following samples provide message sets for industry-standard message formats:

- EDIFACT
- FIX
- SWIFT
- X12

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Self-defining messages: You can create and route messages that are self-defining. The best example of a self-defining message is an XML document.

You can also model self-defining messages in the WebSphere Message Broker Toolkit. However, you do not need to deploy these message sets to the brokers that

support those message flows. For further information about why you might want to model these messages, see “Why model messages?” on page 1158.

Several samples, including the following, use self-defining XML messages and do not require a message set, consequently fewer resources need to be defined:

- Large Messaging
- Airline Reservations

The following sample demonstrates how you can transform a message from self-defining XML to a predefined binary format:

- Coordinated Request Reply

The following sample demonstrates how you can extract information from an XML message and transform it into BLOB format for storage in a database:

- Data Warehouse

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Related concepts:

“Message modeling” on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

“Parsers” on page 1072
A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from the internal message tree representation.

“Why model messages?” on page 1158
WebSphere Message Broker supplies a range of parsers to parse and write message formats. Some message formats are self-defining and can be parsed without reference to a model. Most message formats, however, are not self-defining, and the parser must have access to a predefined model that describes the message, if it is to parse it correctly.

Related tasks:

Chapter 9, “Developing message flow applications,” on page 1019
Develop message flows to process your business messages and data.

“Constructing message models” on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

Related reference:

“Built-in nodes” on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Which body parser should you use?:

The characteristics of the messages that your applications exchange indicate which body parser you must use.

WebSphere Message Broker provides a range of message parsers. Each parser processes either message body data for messages in a particular message domain (for example, XML), or particular message headers (for example, the MQMD).

Review the messages that your applications send to the broker, and determine to which message domain the message body data belongs, using the following criteria as a guide.

If your application data uses SOAP-based web services, including SOAP with Attachments (MIME) or MTOM
Use the SOAP domain. The SOAP domain has built-in support for WS-Addressing and WS-Security standards.

If your application data uses JSON format, as maybe used in RESTful web services
Use the JSON domain.

If your application data is in XML format other than SOAP
The domain that you use depends on the nature of the XML documents and the processing that you want to perform. See "Which XML parser should you use?" on page 1080

If your application data comes from a C or COBOL application, or consists of fixed-format binary data
Use the MRM domain with a Custom Wire Format (CWF) physical format.

If your application data consists of formatted text, perhaps with field content that is identified by tags, or separated by specific delimiters, or both
Use the MRM domain with a Tagged/Delimited String (TDS) physical format.

If your application data is created using the JMS API
The domain that you use depends on the type of the JMS message. For a full description of JMS message processing, see "JMS message as input" on page 1693.

If your application data is from a WebSphere Adapter such as the adapters for SAP, PeopleSoft, or Siebel
Use the DataObject domain.

If your application data is in SAP text IDoc format, such as those exported using the WebSphere MQ Link for R3
Use the MRM domain with a Tagged/Delimited String (TDS) physical format.

If your application data is in MIME format other than SOAP with Attachments (for example, RosettaNet)
Use the MIME domain. If the message is multipart MIME, you might have to parse specific parts of the message with other parsers. For example, you might use the XMLNSC parser to parse the XML content of a RosettaNet message.

If you do not know, or do not have to know, the content of your application data Use the BLOB domain.

Related concepts:

"SOAP parser and domain" on page 1082
You can use the SOAP parser to create a common WSDL-based logical tree format for working with Web services, independent of the physical bitstream format.

"XML parsers and domains" on page 1084
You can use XML domains to parse and write messages that conform to the W3C

XML standard.

"MRM parser and domain" on page 1111
You can use the MRM domain to parse and write a wide range of message formats.

"JMS parsers and domains" on page 1116
The JMSMap and JMSStream domains can be used for modeling messages that are produced by the implementations of the Java Messaging Service standard.

"MIME parser and domain" on page 1117
Use the MIME domain if your messages use the MIME standard for multipart messages.

"BLOB parser and domain" on page 1124
The BLOB message domain includes all the messages with content that cannot be interpreted and subdivided into smaller sections of information.

"JSON parser and domain" on page 1128
JSON (JavaScript Object Notation) is a simple data-interchange format based on a subset of the JavaScript programming language.

"DataObject parser and domain" on page 1114
Use the DataObject domain to parse and write messages for WebSphere Adapters and CORBA applications.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"Which XML parser should you use?" on page 1080
If your messages are general-purpose XML documents, you can use one of the dedicated XML domains (XMLNSC or XMLNS) to parse the message, or you can use the MRM domain to parse the message.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Developing Java" on page 2628
When you use the JavaCompute node, you customize it to determine the exact processing that it provides.

Related reference:

"WebSphere Broker JMS Transport" on page 1681
The WebSphere Broker JMS Transport is a service that connects applications that send and receive messages that conform to the Java Message Service (JMS) standard.

"JMS message as input" on page 1693
The JMS message is a Java object, and therefore you cannot parse the message as a bit stream. When the message is received, the header data, property data, and payload data are extracted by using the JMS API.

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Which XML parser should you use?:

If your messages are general-purpose XML documents, you can use one of the dedicated XML domains (XMLNSC or XMLNS) to parse the message, or you can use the MRM domain to parse the message.

Note: Although SOAP XML can be parsed using any namespace-aware XML parser, use the dedicated SOAP domain to parse SOAP XML because the SOAP domain provides full support for SOAP with Attachments, and standards such as WS-Addressing and WS-Security.

Note: The XML domain is deprecated. Do not use it for developing new message flows. The XML domain still works with existing message flows.

Which XML parser you choose depends on the nature of your XML messages, and the transformation logic that you want to use. The differentiating features of each domain are:

- The XMLNSC parser has a new architecture that gives significant performance improvements over the XMLNS and XML parsers.
- The XMLNSC parser can be used with or without an XML Schema that is generated from a message set. Using a message set with the XMLNSC parser allows the parser to operate in validating mode which provides the following capabilities:
  - XML Schema 1.0 compliant validation when parsing and writing.
  - The XML Schema indicates the real data type of a field in the message instead of always treating the field as a character string.
  - Base64 binary data can be automatically decoded.
- The MRM parser must be used with a message dictionary that is generated from a message set. This message dictionary enables the MRM parser to provide the following capabilities: For example:
  - Validation against the dictionary when parsing and writing. Note that validation is not XML Schema 1.0 compliant.
  - The dictionary indicates the real data type of a field in the message instead of always treating the field as a character string.
  - Base64 binary data can be automatically decoded.
- The XMLNS parser is programmatic and does not use a model when parsing.

  This means that:
  - All data in an XML message is treated as character strings.
  - Validation is not possible when parsing and writing.
- The MRM parser uses information from the XML physical format of a message set to simplify the task of creating transformation logic:
  - Date and time information can be extracted from a data value using a specified format string.
  - When creating output messages, the MRM parser can automatically generate the XML declaration, and other XML constraints.
- The XMLNSC and XMLNS parsers do not use XML physical format information from a message set. Transformation logic must explicitly create all constructs in an output message.

- The MRM parser discards some parts of an XML message when parsing; for example, white space formatting, XML comments, XML processing instructions, and inline DTDs. If you use this parser, you cannot create these constructs when building an output message.
- The XMLNSC parser, by default, discards white space formatting, XML comments, XML processing instructions, and inline DTDs. However, options are provided to preserve all of these constructs, except inline DTDs. You can create them all, except inline DTDs, when constructing an output message.
- The XMLNS parser preserves all parts of an XML document, including white space formatting. You can create all XML constructs when constructing an output message.
- The XMLNSC and MRM parsers build compact message trees that use fewer syntax elements than the XMLNS parser for attributes and simple elements. This makes these parsers more suitable than the XMLNS parser for parsing large XML messages.
- The XMLNS parser builds a message tree that conforms more closely than the XMLNSC or MRM parsers to the XML Data Model. You might want to use this parser if you are using certain XPath expressions to access the message tree, and the relative position of parent and child nodes is important, or if you are accessing text nodes directly.
- The XMLNS parser holds the text content of an element as a child of the element, whereas the XMLNSC parser builds a compact tree. To update a text value in the compact tree of the XMLNSC parser, use ESQL to set the VALUE of the target node. For example:

  ```
  Set OutputRoot.XMLNSC.TestMsg.Value3 VALUE = InputRoot.XMLNSC.TestMsg.Value2;
  ```

Tip: If performance is critical, use the XMLNSC domain.

Tip: If you need to validate the content and values in XML messages, use the XMLNSC domain.

Tip: If you need to preserve formatting in XML messages on output, use the XMLNSC domain with the option to retain mixed content.

Tip: If you require the message tree to conform as closely as possible to the XML data model, perhaps because you are using certain XPath expressions to access the message tree, use the XMLNS domain.

Tip: If you are taking non-XML data that has been parsed by the CWF or TDS formats of the MRM domain, and merely transforming the data to the equivalent XML, use the MRM domain. You can achieve this by adding an XML physical format to the message set with default values, and changing the Message Format in the Properties folder of the message tree.

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

"Why model messages?" on page 1158
WebSphere Message Broker supplies a range of parsers to parse and write message formats. Some message formats are self-defining and can be parsed without reference to a model. Most message formats, however, are not self-defining, and the parser must have access to a predefined model that describes the message, if it is to parse it correctly.

"The message model" on page 1160
The message model consists of the following components.

"Which body parser should you use?" on page 1077
The characteristics of the messages that your applications exchange indicate which body parser you must use.

"XML parsers and domains" on page 1084
You can use XML domains to parse and write messages that conform to the W3C XML standard.

"MRM parser and domain" on page 1111
You can use the MRM domain to parse and write a wide range of message formats.

"SOAP parser and domain"
You can use the SOAP parser to create a common WSDL-based logical tree format for working with Web services, independent of the physical bitstream format.

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

SOAP parser and domain:

You can use the SOAP parser to create a common WSDL-based logical tree format for working with Web services, independent of the physical bitstream format.

Use the SOAP parser in conjunction with the SOAP nodes in your message flow.

Messages in the SOAP domain are processed by the SOAP parser. The SOAP parser creates a common logical tree representation for all SOAP-based Web services and validates the message against a WSDL definition. If a runtime message is not allowed by the WSDL, an exception is thrown, otherwise the `portType` and `operation` names from the WSDL are saved in the logical tree.

The SOAP domain offers `WS-*` processing, together with a canonical tree shape that is independent of the wire format (XML or MIME).

The standards supported are:

- WSDL 1.1
- SOAP 1.1 and 1.2
- MIME 1.0
- Message Transmission Optimization Mechanism (MTOM) 1.0

A WSDL 1.1 definition must be deployed to describe the web service messages that the SOAP domain needs to parse and write at run time. Therefore, the SOAP parser is always model-driven. The bitstream format for these runtime messages can be SOAP 1.1 or SOAP 1.2, optionally wrapped by MIME as an SwA (SOAP with Attachments) or MTOM message.

When a message set that supports the SOAP domain is added to a broker archive (BAR) file, XML schemas are created automatically from the message definition files in the message set, and any WSDL files in the message set are added to the BAR file. The WSDL and XML schema are deployed to the broker and used by the SOAP parser.

If you want the SOAP domain to parse your SOAP Web service, complete the following steps:

1. Create a new message set, or locate an existing message set.
2. Ensure that either the message set has its *Default message domain* project set to `SOAP`, or the *SOAP* check box (under *Supported message domains*) is selected, to indicate that the message set supports the SOAP domain.
3. Import your WSDL file to create a message definition file. The WSDL is also added to the message set. Message definition files for the SOAP envelope and the SOAP logical tree are also added to the message set automatically.
4. Add the message set to a broker archive (BAR) file, which generates the required XML Schema and WSDL in a file with extension .xsdzip, and deploy the BAR file to the broker.
5. If you associate your WSDL with a SOAP node in your message flow, the *Message Set* property is automatically set in the node. The *Message domain* property is always pre-selected as `SOAP`.

Tip: The SOAP parser invokes the XMLNSC parser to parse and validate the XML content of the SOAP Web service. See "XMLNSC parser" on page 1090.

Related concepts:

"What is SOAP?" on page 1604
SOAP is an XML message format used in Web service interactions. SOAP messages are typically sent over HTTP or JMS, but other transport protocols can be used. The use of SOAP in a specific Web service is described by a WSDL definition.

"SOAP message details" on page 1084
A SOAP message consists of an `<Envelope>`, which is the root element in every SOAP message, and this contains two child elements, an optional `<Header>` and a mandatory `<Body>`.

"SOAP tree overview" on page 1611
This tree format allows you to access the key parts of the SOAP message in a convenient way.

"Parsers" on page 1072
A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from the internal message tree representation.

"XMLNSC parser" on page 1090
The XMLNSC parser is a flexible, general-purpose XML parser that offers high performance XML parsing and optional XML Schema validation.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

SOAP message details:

A SOAP message consists of an `<Envelope>`, which is the root element in every SOAP message, and this contains two child elements, an optional `<Header>` and a mandatory `<Body>`.

If the SOAP message has attachments, the 'envelope' is wrapped by MIME, or is encoded as MTOM.

For further information on the structure of a SOAP message, see "The structure of a SOAP message" on page 1605.

Related concepts:

"The SOAP body" on page 1608
The SOAP body (the <Body> element) is a mandatory sub-element of the SOAP envelope, which contains information intended for the ultimate recipient of the message.

"The SOAP header" on page 1606
The SOAP header (the <Header> element) is an optional sub-element of the SOAP envelope, and is used to pass application-related information that is processed by SOAP nodes along the message flow.

"What is SOAP?" on page 1604
SOAP is an XML message format used in Web service interactions. SOAP messages are typically sent over HTTP or JMS, but other transport protocols can be used. The use of SOAP in a specific Web service is described by a WSDL definition.

XML parsers and domains:

You can use XML domains to parse and write messages that conform to the W3C XML standard.

The term *XML domains* refers to a group of three domains that are used by WebSphere Message Broker to parse XML documents.

When reading an XML message, the parser that is associated with the domain builds a message tree from the input bit stream. The input bit stream must be a well-formed XML document that conforms to the W3C XML Specification (version 1.0 or 1.1).

When writing a message, the parser creates an XML bit stream from a message tree.

The domains have different characteristics, for guidance about which domain to choose, see "Which XML parser should you use?" on page 1080.

XMLNSC domain

The XMLNSC domain is the preferred domain for parsing all general purpose XML messages, including those messages that use XML namespaces. This parser is the preferred parser for the following reasons:

- The XMLNSC parser has an architecture that results in ultra-high performance when parsing all kinds of XML.
- The XMLNSC parser reduces the amount of memory that is used by the logical message tree that is created from the parsed message. The default behavior of the parser is to discard non-significant white space and

mixed content, comments, processing instructions, and embedded DTDs; however controls are provided to retain mixed content, comments, and processing instructions, if required.

- The XMLNSC parser can operate as a model-driven parser, and can validate XML messages against XML Schemas generated from a message set, to ensure that your XML messages are correct.

XMLNS domain
If the XMLNSC domain does not meet your requirements, use the alternative namespace-aware domain and parser.

XML domain
The XML domain is not namespace-aware. It is deprecated and must not be used to develop new message flows.

The MRM domain also provides XML parsing and writing facilities. For guidance on when you might use MRM XML instead of one of the XML parsers, see "Which XML parser should you use?" on page 1080.

By default, the three XML parsers are *programmatic* parsers and do not use a message set at run time when parsing and writing. However, the XMLNSC parser can operate as a model-driven parser and can validate XML messages for correctness against XML Schemas generated from a message set.

When you use the XMLNS or XML parsers, or the XMLNSC parser without a message set, it is good practice to create and use a message set in the WebSphere Message Broker Toolkit; this action simplifies the development of your message flow applications, even though the message set is not deployed to the broker run time.

For the advantages of creating a message set, see "Why model messages?" on page 1158.

The XML parsers are on-demand parsers. For more information, see "Parsing on demand" on page 4173.

The topics in this information center provide a summary of XML terminology, concepts, and message constructs. These aspects are important when you use XML messages in your message flows.

Tip: For more detailed information about XML, see the World Wide Web Consortium (W3C) Web site.

Example XML message parsing: A simple XML message might take the following form:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE Envelope
PUBLIC "http://www.ibm.com/dtds" "example.dtd"
[<!ENTITY Example_ID "ST_TimeoutNodes Timeout Request Input Test Message">]
>
<Envelope version="1.0">
 <Header>
  <Example>&Example_ID;</Example>
  <!-- This is a comment  -->
 </Header>
 <Body  version="1.0">
  <Element01>Value01</Element01>
  <Element02/>
  <Element03>
```

```
   <Repeated>ValueA</Repeated>
   <Repeated>ValueB</Repeated>
  </Element03>
  <Element04><P>This is <B>bold</B> text</P></Element04>
 </Body>
</Envelope>
```

The following sections show the output that is created by the Trace node when this example message has been parsed in the XMLNS and XMLNSC parsers. They demonstrate the differences in the internal structures that are used to represent the data as it is processed by the broker.

Example XML Message parsed in the XMLNS domain: In the following example, the white space elements within the tree are present because of the space, tab, and line breaks that format the original XML document; for clarity, the actual characters in the trace have been replaced with `'WhiteSpace'`. White space within an XML element does have business meaning, and is represented by using the Content syntax element. The XmlDecl, DTD, and comments, are represented in the XML domain using explicit syntax elements with specific field types.

```
(0x01000010):XMLNS        = (
    (0x05000018):XML      = (
      (0x06000011): = '1.0'
      (0x06000012): = 'UTF-8'
      (0x06000014): = 'no'
    )
    (0x06000002):         = 'WhiteSpace'
    (0x05000020):Envelope = (
      (0x06000004): = 'http://www.ibm.com/dtds'
      (0x06000008): = 'example.dtd'
      (0x05000021): = (
        (0x05000011):Example_ID = (
          (0x06000041): = 'ST_TimeoutNodes Timeout Request Input Test Message'
        )
      )
    )
    (0x06000002):         = 'WhiteSpace'
    (0x01000000):Envelope = (
      (0x03000000):version = '1.0'
      (0x02000000):        = 'WhiteSpace'
      (0x01000000):Header  = (
        (0x02000000):        = 'WhiteSpace'
        (0x01000000):Example = (
          (0x06000020): = 'Example_ID'
          (0x02000000): = 'ST_TimeoutNodes Timeout Request Input Test Message'
          (0x06000021): = 'Example_ID'
        )
        (0x02000000):        = 'WhiteSpace'
        (0x06000018):        = ' This is a comment  '
        (0x02000000):        = 'WhiteSpace'
      )
      (0x02000000):        = 'WhiteSpace'
      (0x01000000):Body    = (
        (0x03000000):version   = '1.0'
        (0x02000000):          = 'WhiteSpace'
        (0x01000000):Element01 = (
          (0x02000000): = 'Value01'
        )
        (0x02000000):          = 'WhiteSpace'
        (0x01000000):Element02 =
        (0x02000000):          = 'WhiteSpace'
        (0x01000000):Element03 = (
          (0x02000000):         = 'WhiteSpace'
          (0x01000000):Repeated = (
            (0x02000000): = 'ValueA'
```

```
      )
      (0x02000000):         = 'WhiteSpace'
      (0x01000000):Repeated = (
        (0x02000000): = 'ValueB'
      )
      (0x02000000):         = 'WhiteSpace'
    )
    (0x02000000):          = 'WhiteSpace'
    (0x01000000):Element04 = (
      (0x01000000):P = (
        (0x02000000):  = 'This is '
        (0x01000000):B = (
          (0x02000000): = 'bold'
        )
        (0x02000000):  = ' text'
      )
    )
    (0x02000000):          = 'WhiteSpace'
  )
  (0x02000000):        = 'WhiteSpace'
)
```

Example XML Message parsed in the XMLNSC domain: The following trace shows the elements that are created to represent the same XML structure within the compact XMLNSC parser in its default mode. In this mode, the compact parser does not retain comments, processing instructions, or mixed text.

The example illustrates the significant saving in the number of syntax elements that are used to represent the same business content of the example XML message when using the compact parser.

By not retaining mixed text, all of the white space elements that have no business data content are no longer taking any space in the broker message tree at run time. However, the mixed text in `Element04.P` is also discarded, and only the value of the child folder, `Element04.P.B`, is held in the tree; the text `This is` and `text` in `P` is discarded. This type of XML structure is not typically associated with business data formats; therefore, use of the compact XMLNSC parser is typically desirable. However, if you want to this type of processing, either do not use the XMLNSC parser, or use it with *Retain mixed text mode* enabled.

The handling of the XML declaration is also different in the XMLNSC parser. The version, encoding, and stand-alone attributes are held as child entities of the XmlDeclaration, rather than as elements with a particular field type.

```
(0x01000000):XMLNSC     = (
    (0x01000400):XmlDeclaration = (
      (0x03000100):Version    = '1.0'
      (0x03000100):Encoding   = 'UTF-8'
      (0x03000100):StandAlone = 'no'
    )
    (0x01000000):Envelope       = (
      (0x03000100):version = '1.0'
      (0x01000000):Header  = (
        (0x03000000):Example = 'ST_TimeoutNodes Timeout Request Input Test Message'
      )
      (0x01000000):Body    = (
        (0x03000100):version   = '1.0'
        (0x03000000):Element01 = 'Value01'
        (0x01000000):Element02 =
        (0x01000000):Element03 = (
          (0x03000000):Repeated = 'ValueA'
          (0x03000000):Repeated = 'ValueB'
        )
```

```
        (0x01000000):Element04 = (
          (0x01000000):P = (
            (0x03000000):B = 'bold'
          )
        )
  )
```

The following samples use the XML parser to process messages:

- Coordinated Request Reply
- Large Messaging
- Message Routing

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Some predefined message models are supplied with the WebSphere Message Broker Toolkit and can be imported by using the New Message Definition File wizard and selecting the IBM supplied message option. See "IBM supplied messages that you can import" on page 6367.

Related concepts:

"Parsers" on page 1072
A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from the internal message tree representation.

"XMLNSC parser" on page 1090
The XMLNSC parser is a flexible, general-purpose XML parser that offers high performance XML parsing and optional XML Schema validation.

"XMLNS parser" on page 1104
The XMLNS parser is a flexible, general-purpose XML parser.

"XML parser" on page 1110
The XML domain is very similar to the XMLNS domain, but the XML domain has no support for XML namespaces or opaque parsing.

"MRM parser and domain" on page 1111
You can use the MRM domain to parse and write a wide range of message formats.

"Which XML parser should you use?" on page 1080
If your messages are general-purpose XML documents, you can use one of the dedicated XML domains (XMLNSC or XMLNS) to parse the message, or you can use the MRM domain to parse the message.

"Why model messages?" on page 1158
WebSphere Message Broker supplies a range of parsers to parse and write message formats. Some message formats are self-defining and can be parsed without reference to a model. Most message formats, however, are not self-defining, and the parser must have access to a predefined model that describes the message, if it is to parse it correctly.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"JMS parsers and domains" on page 1116
The JMSMap and JMSStream domains can be used for modeling messages that are

produced by the implementations of the Java Messaging Service standard.

"SOAP parser and domain" on page 1082
You can use the SOAP parser to create a common WSDL-based logical tree format for working with Web services, independent of the physical bitstream format.

"Logical tree structure" on page 1042
The logical tree structure is the internal (broker) representation of a message. It is also known as the message assembly.

"Correlation names" on page 1069
A correlation name is a field reference that identifies a well-defined starting point in the logical message tree and is used in field references to describe a standard part of the tree format.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Manipulating message body content" on page 2418
The message body is always the last child of root, and is identified by its parser name, for example XML or MRM. You can refer to, modify, and create message body data.

"Manipulating messages in the XMLNSC domain" on page 2546
If you are writing ESQL to process messages in the XMLNSC domain, it is helpful to learn about the structure of the message tree that the XMLNSC parser builds.

"Manipulating messages in the XMLNS domain" on page 2563
When you write ESQL for processing messages in the XMLNS domain, it is helpful to understand the structure of the message tree that the XMLNS parser builds.

Related reference:

"IBM supplied messages that you can import" on page 6367
You can import IBM supplied messages to create a new message definition file.

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"XML constructs" on page 4257
A self-defining XML message carries the information about its content and structure within the message in the form of a document that adheres to the XML specification. Its definition is not held anywhere else.

"Parsing on demand" on page 4173
On-demand parsing, referred to as partial parsing, is used to parse an input message bit stream only as far as is necessary to satisfy the current reference. The parsers that can perform partial parsing of input messages are the JSON, MRM, XML, XMLNS, and XMLNSC parsers.

developerWorks

XMLNSC parser:

The XMLNSC parser is a flexible, general-purpose XML parser that offers high performance XML parsing and optional XML Schema validation.

The XMLNSC parser has a range of options that make it suitable for most XML processing requirements. Some of these options are only available in the XMLNSC parser.

Although the XMLNSC parser is capable of parsing XML documents without an XML Schema, extra features of the parser become available when it operates in model-driven mode. In model-driven mode, the XMLNSC parser is guided by an XML Schema, which describes the shape of the message tree (the logical model).

XML Schemas are created automatically from the content of a message set when the message set is added to a broker archive (BAR) file. The XML Schemas are deployed to the broker and used by the XMLNSC parser to validate your XML messages. Validation is fully compliant with the XML Schema 1.0 specification.

For guidance on when to use the XMLNSC domain and parser, see "Which XML parser should you use?" on page 1080.

If you want the XMLNSC domain to parse a message, select *Message Domain* as `XMLNSC` on the appropriate node in the message flow. Additionally, if you want the XMLNSC parser to validate your messages, perform the additional steps that are described in "XMLNSC validation" on page 1099.

Features of the XMLNSC parser

Feature	Present	Description
Namespace support	Yes	Namespace information is used if it is present. No user configuration is required. See "XML parsers namespace support" on page 1109.
On-demand parsing	Yes	See "Parsing on demand" on page 4173.
Compact message tree	Yes	Less memory is used when building a message tree from an XML document. See "Manipulating messages in the XMLNSC domain" on page 2546.
Opaque parsing	Yes	One or more elements can be parsed opaquely. See "XMLNSC opaque parsing" on page 1097.
Ultra high performance	Yes	The architecture of the XMLNSC parser means that the parser's use of processor resources is significantly less than that of the other XML parsers.
Validation	Yes	See the table that follows this one.

Feature	Present	Description
Inline DTD support	Partial	Inline DTDs are processed but discarded. See “XMLNSC DTD support” on page 1103.
XML Data Model compliance	Partial	The compact nature of the message tree means that some XPath queries are not supported.

The following features are only available when message validation is enabled. See “XMLNSC validation” on page 1099.

Feature	Description
Message validation	Validates compliance with the XML Schema 1.0 specification.
xsi:nil support	Sets the value of an element to NULL if it has xsi:nil=”true” and the XML Schema indicates that it is nillable.
Default value support	Sets the value of an empty element, or missing attribute, to its default value, according to XML Schema rules.
Use correct simple types	Allows the use of the simple types that are defined in the XML Schema when building the message tree.
Base64 support	Converts base64 data to BLOB when parsing. Converts BLOB to base64 when writing.

If you specify the SOAP domain as the owner of a SOAP Web Services message, the SOAP parser invokes the XMLNSC parser in model-driven mode to parse the XML content of the SOAP message.

If you specify the DataObject domain as the owner of a WebSphere Adapter message, and the message is written to a destination other than a WebSphere Adapter, the DataObject parser invokes the XMLNSC parser to write the message as XML.

Related concepts:

“XML parsers and domains” on page 1084
You can use XML domains to parse and write messages that conform to the W3C XML standard.

“XML parsers namespace support” on page 1109
Namespaces in XML messages are supported by the XMLNSC and XMLNS parsers. Namespaces are not supported by the XML parser.

“XMLNSC opaque parsing” on page 1097
Opaque parsing is a performance feature that is offered by the XMLNSC domain.

“XMLNS parser” on page 1104
The XMLNS parser is a flexible, general-purpose XML parser.

“SOAP parser and domain” on page 1082
You can use the SOAP parser to create a common WSDL-based logical tree format for working with Web services, independent of the physical bitstream format.

"DataObject parser and domain" on page 1114
Use the DataObject domain to parse and write messages for WebSphere Adapters and CORBA applications.

"Which XML parser should you use?" on page 1080
If your messages are general-purpose XML documents, you can use one of the dedicated XML domains (XMLNSC or XMLNS) to parse the message, or you can use the MRM domain to parse the message.

"Why model messages?" on page 1158
WebSphere Message Broker supplies a range of parsers to parse and write message formats. Some message formats are self-defining and can be parsed without reference to a model. Most message formats, however, are not self-defining, and the parser must have access to a predefined model that describes the message, if it is to parse it correctly.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Manipulating messages in the XMLNSC domain" on page 2546
If you are writing ESQL to process messages in the XMLNSC domain, it is helpful to learn about the structure of the message tree that the XMLNSC parser builds.

Related reference:

"Parsing on demand" on page 4173
On-demand parsing, referred to as partial parsing, is used to parse an input message bit stream only as far as is necessary to satisfy the current reference. The parsers that can perform partial parsing of input messages are the JSON, MRM, XML, XMLNS, and XMLNSC parsers.

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

XMLNSC empty elements and null values:

Empty elements and null values occur frequently in XML documents.

A robust message flow must be able to recognize and handle empty elements and null values. Similarly, elements in a message tree might have a NULL value, an empty value, or no value at all. This topic explains the parsing and writing of these values by the XMLNSC domain. For details on processing null values in Graphical Data Maps and ESQL, see "Handling null values" on page 1140.

Parsing

Description	XML input parsed by XMLNSC	Value of 'element' in message tree
Empty element value	`<element/>`	Empty string
Empty element value	`<element></element>`	Empty string
Folder with child elements	`<element><childElement/></element>`	No value
Nil element value	`<element xsi:nil="true"/>`	No value and a child `xsi:nil` attribute with the value 'true'.

Both forms of an empty element result in the same value in the message tree.

Writing

Description	Value of 'element' in message tree	XML output from XMLNSC parser
Empty element value	Empty string	`<element/>`
Null element value	NULL	`<element/>`
Folder with child elements	No value	`<element><childElement/></element>`
Nil element value	Empty string, NULL, or Folder	`<element xsi:nil="true"/>` Note that the XMLNSC parser produces only `xsi:nil` attributes that are already in the message tree. It does not automatically produce `xsi:nil` attributes for all message tree elements that have a NULL value and are 'nillable'.

Empty elements

An empty element can take two forms in an XML document:

- `<element/>`
- `<element></element>`

The XMLNSC parser treats both forms in the same way. The element is added to the message tree with a value of "" (the empty string).

When a message tree is produced by the XMLNSC parser, it always uses the first form for elements that have a value of "" (the empty string) in the input XML message.

Elements with an `xsi:nil` attribute

If validation is enabled for the flow, the XMLNSC parser performs the following validations:

- The 'nillable' property of an element definition in the XML schema is set to 'true'.

  If the element in the document has an `xsi:nil` attribute with the value 'true', the element must not have a value, or contain any child elements.
- The 'nillable' property of an element definition in the XML schema is set to 'false'.

  The element in the input document must not have an `xsi:nil` attribute.

If an element in the input document has an `xsi:nil` attribute, the XMLNSC parser creates an element in the message tree with no value, and a child `xsi:nil` attribute with the value 'true' or 'false'.

When a message tree is written to a bit stream by the XMLNSC parser, if the value of the element is empty, NULL, or no value, and the element has no child

elements, the element is written as <element/>. If the element has an xsi:nil attribute, it is written exactly like any other attribute.

Note that the XMLNSC parser produces only xsi:nil attributes that are already in the message tree. It does not automatically produce xsi:nil attributes for all message tree elements that have a NULL value and are 'nillable'.

Related concepts:

"XMLNSC parser" on page 1090
The XMLNSC parser is a flexible, general-purpose XML parser that offers high performance XML parsing and optional XML Schema validation.

"XML parsers namespace support" on page 1109
Namespaces in XML messages are supported by the XMLNSC and XMLNS parsers. Namespaces are not supported by the XML parser.

"XMLNSC data types" on page 1102
Mapping between XML Schema simple types and the data types that the XMLNSC parser uses in the message tree when Build tree using XML Schema types is specified.

"XMLNSC opaque parsing" on page 1097
Opaque parsing is a performance feature that is offered by the XMLNSC domain.

"XMLNSC validation" on page 1099
The XMLNSC parser offers high-performance, standards-compliant XML Schema validation at any point in a message flow.

"XMLNSC message tree options" on page 1101
The XMLNSC options that are described in this section affect the parsing of an XML document by the XMLNSC parser. They have no effect on XML output.

"XMLNSC DTD support" on page 1103
The input XML message might contain an inline DTD.

"Handling null values" on page 1140
A business message might contain fields that are either empty or have a specific out-of-range value. In these cases, the application that receives the message is expected to treat the field as if it did not have a value. The logical message tree supports this concept by enabling the value of any element to be set to NULL.

XMLNSC: Using field types:

The XMLNSC parser sets the field type on every syntax element that it creates.

The field type indicates the type of XML construct that the element represents. The XMLNSC parser uses the field type when writing a message tree. The field type can be set by using ESQL or Java to control the output XML. The field types that are used by the XMLNSC parser must be referenced by using constants with names that are prefixed by 'XMLNSC.'

Tip: Field type constants that have the prefix 'XML.' are for use with the XMLNS and XML parsers only, and are not valid with the XMLNSC or MRM parsers.

Field types for creating syntax elements

Use the following field type constants to create syntax elements in the message tree. The XMLNSC parser uses these values when creating a message tree from an input message.

XML construct	XMLNSC Field Type constant	Value
Simple Element	XMLNSC.Field XMLNSC.CDataField	0x03000000 0x03000001
Attribute	XMLNSC.SingleAttribute XMLNSC.Attribute	0x03000101 0x03000100
Mixed content	XMLNSC.Value XMLNSC.CDataValue	0x02000000 0x02000001
Namespace Declaration	XMLNSC.SingleNamespaceDecl XMLNSC.NamespaceDecl	0x03000102 0x03000103
Complex element	XMLNSC.Folder	0x01000000
Inline DTD	XMLNSC.DocumentType	0x01000300
XML declaration	XMLNSC.XmlDeclaration	0x01000400
Entity reference	XMLNSC.EntityReference	0x02000100
Entity definition	XMLNSC.SingleEntityDefinition XMLNSC.EntityDefinition	0x03000301 0x03000300
Comment	XMLNSC.Comment	0x03000400
Processing Instruction	XMLNSC.ProcessingInstruction	0x03000401

Field types for path expressions (generic field types)

Use the following field type constants when querying the message tree by using a path expression; for example:

```
SET str = FIELDVALUE(InputRoot.e1.(XMLNSC.Attribute)attr1)
```

It is good practice to specify field types when querying a message tree built by the XMLNSC parser. This makes your ESQL code more specific and more readable, and it avoids incorrect results in some cases. However, care is required when choosing which field type constant to use. When you use the XMLNSC parser, use a generic field type constant when querying the message tree. This allows your path expression to tolerate variations in the input XML.

The generic field type constants are listed in the following table:

XML construct	XMLNSC Field Type constant	Purpose
Tag	XMLNSC.Element	Matches any tag, whether it contains child tags (XMLNSC.Folder) or a value (XMLNSC.Field)
Element	XMLNSC.Field	Matches a tag which contains normal text, CData, or a mixture of both. Does not match tags which contain child tags.
Attribute	XMLNSC.Attribute	Matches single-quoted and double-quoted attributes
Mixed content	XMLNSC.Value	Matches normal text, CData, or a mixture of both
XML Declaration	XMLNSC.NamespaceDecl	Matches single- and double-quoted declarations

If you write

```
InputRoot.e1.(XMLNSC.DoubleAttribute)attrName
```

your path expression does not match a single-quoted attribute. If you use the generic field type constant *XMLNSC.Attribute*, your message flow works with either single-quoted or double-quoted attributes.

Note that you should always use the field type constants and not their numeric values.

Field types for controlling output format

The following field types are provided for XML Schema and base64 support. Do not use these field type constants in path expressions; use them in conjunction with XMLNSC.Attribute and XMLNSC.Field to indicate the required output format for DATE and BLOB values. See "XMLNSC: XML Schema support" on page 2559 for further information.

XMLNSC Field Type constant	Purpose	Value
XMLNSC.gYear	The value must be a DATE. If the field type includes this value, the DATE value is produced by using the XML Schema gYear format.	0x00000010
XMLNSC.gYearMonth	The value must be a DATE. If the field type includes this value, the DATE value is produced by using the XML Schema gYearMonth format.	0x00000040
XMLNSC.gMonth	The value must be a DATE. If the field type includes this value, the DATE value is produced by using the XML Schema gMonth format.	0x00000020
XMLNSC.gMonthDay	The value must be a DATE. If the field type includes this value, the DATE value is produced by using the XML Schema gMonthDay format.	0x00000050
XMLNSC.gDay	The value must be a DATE. If the field type includes this value, the DATE value is produced by using the XML Schema gDay format.	0x00000030
XMLNSC.base64Binary	The value must be a BLOB. The value is produced with base64 encoding.	0x00000060
XMLNSC.List	The element must be XMLNSC.Attribute or XMLNSC.Field. If the field type includes this value, the values of all child elements in the message tree are produced as a space-separated list.	0x00000070

Field types for direct output

Use the following field types to produce pre-constructed segments of an XML document. Character escaping is not done; therefore, take extra care not to construct a badly-formed output document. Use these constants only after carefully exploring alternative solutions.

XMLNSC Field Type constant	Purpose	Value
XMLNSC.Bitstream	The value of this syntax element must be a BLOB. The value is written directly to the output bit stream. For more information about its usage, see "Working with large XML messages" on page 2543.	0x03000200

XMLNSC Field Type constant	Purpose	Value
XMLNSC.AsisElementContent	The value of this syntax element must be CHARACTER. The value is written directly to the output bit stream. No character substitutions are performed. Use this element with care.	0x03000600

Related concepts:

"XMLNSC: The message body" on page 2549
The XMLNSC parser builds a message tree from the body of an XML document.

"XMLNSC: Element values and mixed content" on page 2556
The XMLNSC parser is a compact parser; therefore, an element with single content is parsed as a single syntax element. When an element has both child elements and some text, the text is called *mixed content*.

"XMLNSC: Comments and Processing Instructions" on page 2558
The XMLNSC parser discards comments and processing instructions because both comments and processing instructions are auxiliary information with no business meaning.

Related reference:

"XMLNSC: Attributes and elements" on page 2552
The XMLNSC parser uses field types to represent attributes and elements.

"XMLNSC: Namespace declarations" on page 2554
The XMLNSC parser provides full support for namespaces.

XMLNSC opaque parsing:

Opaque parsing is a performance feature that is offered by the XMLNSC domain.

If you are designing a message flow and you know that certain elements in a message are never referenced by the message flow, you can specify that these elements are parsed opaquely. This reduces the costs of parsing and writing the message, and might improve performance in other parts of the message flow.

Use the property *Opaque Elements* on the *Parser options* page of the relevant message flow node to specify the elements that you want to be parsed opaquely. This property specifies a list of element names. If an element in the input XML message is named in this list, the element is parsed as a single string.

An opaque element cannot be queried like an ordinary element; its value is the segment of the XML bit stream that belongs to the element, and it has no child elements in the message tree, even though it can represent a large subtree in the XML document.

When an opaque element is serialized, the value of the element is copied directly into the output bit stream. The string is converted to the correct code page, but no other changes are made. Because this might produce a bit stream that is not valid XML, some care is required.

Do not parse an element opaquely in any of the following cases:

- The message flow must access one of its child elements.
- The message flow changes the namespace prefix in a way that affects the opaque element or one of its child elements **and** the element is to be copied to the output bit stream.

- The element, or any child element, contains a reference to an internal entity that is defined in an inline DTD **and** the element is to be copied to the output bit stream.
- The element contains child attributes that have default values that are defined in an inline DTD **and** the element is to be copied to the output bit stream.

Make sure that you check the above points before you specify an element for opaque parsing.

Using opaque parsing has some drawbacks. When opaque parsing is enabled, some parts of the message are never parsed, and XML that is either badly formed or not valid is allowed to pass through the message flow without being detected. For this reason, if you enable validation, you cannot use opaque parsing.

The XMLNS domain offers a more limited opaque parsing facility, but this is provided only to support existing applications. Use the XMLNSC domain in new message flows if you want to use opaque parsing.

Related tasks:

"Specifying opaque elements for the XMLNSC parser"
Specify an element as an opaque element so that its content is ignored by the XMLNSC parser.

Specifying opaque elements for the XMLNSC parser:

Specify an element as an opaque element so that its content is ignored by the XMLNSC parser.

Before you begin

About this task

To specify the elements that are to be skipped by the XMLNSC parser:

Procedure

1. Right-click the selected message flow node, click **Properties** and select **Parser Options**.
2. At the bottom of the `XMLNSC Parser Options` panel is an area that lists the elements that have already been selected as opaque elements. Click **Add** to add an element to this list. A new pane **Add Opaque elements Entry** opens.
3. In the **Add Opaque elements Entry** pane, specify the new XML element that you want to be opaquely parsed. Each opaque element must be specified as an ESQL element name or an XPath expression of the form `//prefix:name` (or `//name`, if your input document does not contain namespaces).

   Note: A prefix is used rather than a full URI to identify the namespace; for further information, see "XPath namespace support" on page 2650.

What to do next

Click **Edit** or **Delete** to edit the list of opaque elements.

Related concepts:

"XMLNSC opaque parsing" on page 1097
Opaque parsing is a performance feature that is offered by the XMLNSC domain.

"Extracting information from a message by using XPath 1.0 and a JavaCompute node" on page 2648
XPath is a query language designed for use with XML documents, but you can use it with any tree structure to query contents.

XMLNSC validation:

The XMLNSC parser offers high-performance, standards-compliant XML Schema validation at any point in a message flow.

Validation of the input XML message or the message tree is performed against the XML Schemas that are deployed.

Validation is not the same as *parsing*. When parsing, the XMLNSC parser always checks that the input document is well-formed XML, according to the XML specification. If validation is enabled, the XMLNSC parser also checks that the XML document obeys the rules in the XML Schema.

Enabling XML Schema validation in a message flow

You must complete the following tasks to construct a message flow that validates an XML document in accordance with an XML Schema:

- Enable validation at the appropriate point in the message flow. This is typically achieved by setting the *Validate* property of the appropriate node to `Content` or `Content and Value`. See "Validating messages" on page 1478.
- Ensure that all required XML Schema files are deployed. See "Deploying XML Schemas" later in this section.
- Specify the message set in which the XML Schemas are deployed. Typically, you specify the message set by selecting the `Message Set` property on the node.

Deploying XML Schemas

All XML Schemas that are used by WebSphere Message Broker must be created as message definition files within a message set.

To create and deploy a message set for XML Schema validation:

1. Create a new message set or locate an existing message set.
2. Ensure that the message set has its *Default message domain* set to `XMLNSC`, or that the XMLNSC check box under *Supported message domains* is selected, to indicate that the message set supports the XMLNSC domain.
3. Create a message definition file in the message set to represent your message. If you have an existing XML Schema or DTD that describes your message, you can import it. You can repeat this step for each message that you want to validate.
4. Add the message set to a broker archive (BAR) file, which generates the required XML Schema in a file with extension .xsdzip, and deploy the BAR file to the broker.

Standards compliant validation

XMLNSC validation complies fully with XML Schema v1.0 as defined in the specifications that are available at http://www.w3.org/TR/xmlschema-1/ and http://www.w3.org/TR/xmlschema-2/, with the following minor exceptions:

- Any floating point value that is smaller than 10E-43 is treated as zero.

- Any member of a group or complex type, that has both minOccurs > 1024 and maxOccurs > 1024, is validated as if minOccurs = 0 and maxOccurs is unbounded.

Validating XML v1.1 documents

You can validate documents that conform to the XML v1.1 specification, but support is limited by the fact that the XML Schema v1.0 documents must conform to XML v1.0.

As an example, you cannot always declare an XML v1.1 tag name in XML Schema v1.0. This limitation is not imposed by the XMLNSC parser implementation; it is a limitation of XML Schema v1.0.

Interpreting validation errors

A validation error is an error that results when the XML document breaks the rules that are defined in the XML schema. The XML Schema standard specifies exactly what these rules are, and how they should be applied. Validation errors that the XMLNSC parser issues contain information that links the error to the XML Schema rule that has been violated.

All validation errors are reported in BIP5025 or BIP5026 messages. Both messages begin with text in the following form:

```
XML schema validation error '[cvc-error key: error description]'
```

Examples:

```
'cvc-minInclusive-valid: The value "2" is not valid with respect to the minInclusive facet
with value "3" for type "po:itemCountType".'
```

```
'cvc-complex-type.2.4.a: Expecting element with local name "numItems" but saw "totalValue".'
```

To find the XML Schema rule that has been violated, open the XML Schema specification and search for the error key.

Example 1: Open http://www.w3.org/TR/xmlschema-1/ and search for ‘cvc-minInclusive-valid'. Follow the link to the XML Schema rules for the minInclusive facet.

Example 2: Open http://www.w3.org/TR/xmlschema-1/ and search for ‘cvc-complex-type'. Follow the link to the XML Schema rules for validating the content of a complex type. In this case, the error key contains extra information. The ‘2.4.a' refers to the exact sub-rule that was violated. It should not be included when searching for the rule.

If the XML Schema specification does not provide enough information, you can find more information using a search engine. The XML Schema standard is very widely used, and many online tutorials and other resources are available.

Related concepts:

“XMLNSC parser” on page 1090
The XMLNSC parser is a flexible, general-purpose XML parser that offers high performance XML parsing and optional XML Schema validation.

“XML parsers namespace support” on page 1109
Namespaces in XML messages are supported by the XMLNSC and XMLNS parsers. Namespaces are not supported by the XML parser.

"XMLNSC empty elements and null values" on page 1092
Empty elements and null values occur frequently in XML documents.

"XMLNSC message tree options"
The XMLNSC options that are described in this section affect the parsing of an XML document by the XMLNSC parser. They have no effect on XML output.

"XMLNSC data types" on page 1102
Mapping between XML Schema simple types and the data types that the XMLNSC parser uses in the message tree when `Build tree using XML Schema types` is specified.

"XMLNSC DTD support" on page 1103
The input XML message might contain an inline DTD.

"XMLNSC opaque parsing" on page 1097
Opaque parsing is a performance feature that is offered by the XMLNSC domain.

"Handling null values" on page 1140
A business message might contain fields that are either empty or have a specific out-of-range value. In these cases, the application that receives the message is expected to treat the field as if it did not have a value. The logical message tree supports this concept by enabling the value of any element to be set to NULL.

Related tasks:

"Validating messages" on page 1478
The broker provides validation based on the message model for predefined messages.

Related reference:

"Validation properties" on page 4169
You can control validation by setting properties on the Validate and Parser Options tabs for the nodes that are listed in the following table.

XMLNSC message tree options:

The XMLNSC options that are described in this section affect the parsing of an XML document by the XMLNSC parser. They have no effect on XML output.

Retain Mixed Content

Mixed content is XML text which occurs between elements.

```
<parent>
     <childElement1>Not mixed content</childElement1>
     This text is mixed content
     <childElement2>Not mixed content</childElement2>
</parent>
```

By default, the XMLNSC parser discards all mixed content. Mixed content is retained in the message tree if you select *Retain mixed content* in the Parser options page of the input node. For further information, see 'Element with mixed content' in "XMLNSC: Element values and mixed content" on page 2556.

Retain Comments

By default, the XMLNSC parser discards all comments in the input XML. Comments are retained in the message tree if you select *Retain comments* in the Parser options page of the input node. For further information, see 'Comments' in "XMLNSC: Comments and Processing Instructions" on page 2558.

Retain Processing Instructions

By default, the XMLNSC parser discards all processing instructions in the input XML. Processing instructions are retained in the message tree if you select *Retain processing instructions* in the Parser options page of the input node. For further information, see 'Processing Instructions' in “XMLNSC: Comments and Processing Instructions” on page 2558.

Build tree using XML Schema data types

By default, the XMLNSC parser uses the CHARACTER data type for all element and attribute values that the parser creates in the message tree. However, if you are using the XMLNSC parser to validate the XML document, you can select *Build tree using XML Schema data types* in the Parser options page of the input node. This causes element and attribute values to be cast to the message broker data type that most closely matches their XML Schema simple type. The exact mapping between XML schema types and message broker types can be found in “XMLNSC data types.”

Related concepts:

“XMLNSC parser” on page 1090
The XMLNSC parser is a flexible, general-purpose XML parser that offers high performance XML parsing and optional XML Schema validation.

“XML parsers namespace support” on page 1109
Namespaces in XML messages are supported by the XMLNSC and XMLNS parsers. Namespaces are not supported by the XML parser.

“XMLNSC opaque parsing” on page 1097
Opaque parsing is a performance feature that is offered by the XMLNSC domain.

“XMLNSC validation” on page 1099
The XMLNSC parser offers high-performance, standards-compliant XML Schema validation at any point in a message flow.

“XMLNSC data types”
Mapping between XML Schema simple types and the data types that the XMLNSC parser uses in the message tree when `Build tree using XML Schema types` is specified.

“XMLNSC DTD support” on page 1103
The input XML message might contain an inline DTD.

Related tasks:

“Manipulating messages in the XMLNSC domain” on page 2546
If you are writing ESQL to process messages in the XMLNSC domain, it is helpful to learn about the structure of the message tree that the XMLNSC parser builds.

XMLNSC data types:

Mapping between XML Schema simple types and the data types that the XMLNSC parser uses in the message tree when `Build tree using XML Schema types` is specified.

For mapping details, see “ESQL to XML Schema data type mapping” on page 5044. Base64 decoding is automatically performed by the XMLNSC parser.

List types

In the message tree, a list type is represented as a parent node with an anonymous child node for each list item. This allows repeating lists to be handled without any loss of information.

If a list element repeats, the occurrences appear as siblings of one another, and each occurrence has its own set of child nodes representing its own list items.

Related concepts:

"XMLNSC parser" on page 1090
The XMLNSC parser is a flexible, general-purpose XML parser that offers high performance XML parsing and optional XML Schema validation.

"XML parsers namespace support" on page 1109
Namespaces in XML messages are supported by the XMLNSC and XMLNS parsers. Namespaces are not supported by the XML parser.

"XMLNSC opaque parsing" on page 1097
Opaque parsing is a performance feature that is offered by the XMLNSC domain.

"XMLNSC validation" on page 1099
The XMLNSC parser offers high-performance, standards-compliant XML Schema validation at any point in a message flow.

"XMLNSC message tree options" on page 1101
The XMLNSC options that are described in this section affect the parsing of an XML document by the XMLNSC parser. They have no effect on XML output.

"XMLNSC DTD support"
The input XML message might contain an inline DTD.

Related reference:

"ESQL to XML Schema data type mapping" on page 5044
Mapping from XML Schema simple type to ESQL message tree data type.

XMLNSC DTD support:

The input XML message might contain an inline DTD.

Parsing

If the input XML document has an inline DTD, the XMLNSC parser reads and uses information in the DTD while parsing, but does not add the DTD information to the message tree.

Internal entity definitions in the DTD are used to automatically expand entity references that are encountered in the body of the document.

Attributes that are missing from the input document are automatically supplied with the default value specified in the DTD.

The XMLNSC parser never adds the DTD to the message tree because the information that it contains has already been used during the parse. This behavior keeps the message tree compact and reduces CPU usage, and means that the XMLNSC parser does not always produce exactly the same document as it parsed. However, the business meaning of the output document is not altered.

If these restrictions are a problem, the XMLNS domain and parser provide full support for parsing and writing of the DTD. See "XMLNS DTD support" on page 1108.

Writing

The XMLNSC parser can produce a DTD that contains entity definitions only. This behavior allows the XMLNSC parser to be used for writing out XML documents that use internal entities (the most common reason for using a DTD). See "Manipulating messages in the XMLNSC domain" on page 2546 for further details.

External DTDs

No support is offered for external DTDs

Related concepts:

"XMLNSC parser" on page 1090
The XMLNSC parser is a flexible, general-purpose XML parser that offers high performance XML parsing and optional XML Schema validation.

"XML parsers namespace support" on page 1109
Namespaces in XML messages are supported by the XMLNSC and XMLNS parsers. Namespaces are not supported by the XML parser.

"XMLNSC empty elements and null values" on page 1092
Empty elements and null values occur frequently in XML documents.

"XMLNSC validation" on page 1099
The XMLNSC parser offers high-performance, standards-compliant XML Schema validation at any point in a message flow.

"XMLNSC message tree options" on page 1101
The XMLNSC options that are described in this section affect the parsing of an XML document by the XMLNSC parser. They have no effect on XML output.

"XMLNSC data types" on page 1102
Mapping between XML Schema simple types and the data types that the XMLNSC parser uses in the message tree when `Build tree using XML Schema types` is specified.

"XMLNSC opaque parsing" on page 1097
Opaque parsing is a performance feature that is offered by the XMLNSC domain.

"Handling null values" on page 1140
A business message might contain fields that are either empty or have a specific out-of-range value. In these cases, the application that receives the message is expected to treat the field as if it did not have a value. The logical message tree supports this concept by enabling the value of any element to be set to NULL.

Related tasks:

"Manipulating messages in the XMLNSC domain" on page 2546
If you are writing ESQL to process messages in the XMLNSC domain, it is helpful to learn about the structure of the message tree that the XMLNSC parser builds.

XMLNS parser:

The XMLNS parser is a flexible, general-purpose XML parser.

The XMLNS parser is not model-driven and does not use an XML Schema when parsing XML documents.

For guidance on when to use the XMLNS domain and parser, see "Which XML parser should you use?" on page 1080.

If you want the XMLNS domain to parse a particular message, you must select *Message Domain* as `XMLNS` on the appropriate node in the message flow.

Features of the XMLNS parser

Feature	Present	Description
Namespace support	Yes	Namespace information is used if it is present. No user configuration is required. See "Namespace support" on page 1508.
On-demand parsing	Yes	See "Parsing on demand" on page 4173.
Compact message tree	No	
Opaque parsing	Partial	Limited support from ESQL only for parsing a single element opaquely. See "XMLNS opaque parsing" on page 1107.
Ultra high performance	No	
Validation	No	
Inline DTD support	Yes	Inline DTDs are processed and retained in the message tree. See "XMLNS DTD support" on page 1108.
XML Data Model compliance	Yes	The resultant message tree conforms to the XML Data Model.

Related concepts:

"XML parsers and domains" on page 1084
You can use XML domains to parse and write messages that conform to the W3C XML standard.

"Namespace support" on page 1508
The XPath Expression builder provides qualified support for namespaces.

"XMLNSC parser" on page 1090
The XMLNSC parser is a flexible, general-purpose XML parser that offers high performance XML parsing and optional XML Schema validation.

"Which XML parser should you use?" on page 1080
If your messages are general-purpose XML documents, you can use one of the dedicated XML domains (XMLNSC or XMLNS) to parse the message, or you can use the MRM domain to parse the message.

"Why model messages?" on page 1158
WebSphere Message Broker supplies a range of parsers to parse and write message formats. Some message formats are self-defining and can be parsed without reference to a model. Most message formats, however, are not self-defining, and the parser must have access to a predefined model that describes the message, if it is to parse it correctly.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Manipulating messages in the XMLNS domain" on page 2563
When you write ESQL for processing messages in the XMLNS domain, it is helpful to understand the structure of the message tree that the XMLNS parser builds.

Related reference:

"Parsing on demand" on page 4173
On-demand parsing, referred to as partial parsing, is used to parse an input message bit stream only as far as is necessary to satisfy the current reference. The parsers that can perform partial parsing of input messages are the JSON, MRM, XML, XMLNS, and XMLNSC parsers.

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

XMLNS empty elements and null values:

Empty elements and null values occur frequently in XML documents.

A robust message flow must be able to recognize and handle empty elements and null values. Similarly, elements in a message tree might have a NULL value, an empty value, or no value at all. This topic explains the parsing and writing of these values by the XMLNS domain. For advice on good ESQL or Java coding practices see "Handling null values" on page 1140.

Parsing

Description	XML input parsed by XMLNS	Value of 'element' in message tree
Empty element value	<element/>	Empty string
Empty element value	<element></element>	Empty string
Folder with child elements	<element><childElement/ ></element>	No value
Nil element value	<element xsi:nil="true"/>	Empty string

Note that both forms of an empty element result in the same value in the message tree.

Note also that a NULL value is never put into the message tree by the XMLNS parser.

Writing

Description	Value of 'element' in message tree	XML output from XMLNS parser
Empty element value	Empty string	<element/>
Null element value	NULL	<element/>
Folder with child elements	No value	<element><childElement/ ></element>

Empty elements

An empty element can take two forms in an XML document:

```
- <element/>
- <element></element>
```

The XMLNS parser treats both forms in the same way. The element is added to the message tree with a value of "" (the empty string).

When a message tree is produced by the XMLNS parser, it always uses the first form for elements that have a value of "" (the empty string).

Elements with an xsi:nil attribute

The XMLNS parser treats the xsi:nil attribute exactly like any other attribute. When xsi:nil is encountered while parsing, it does not set the value of the parent element to NULL. If you require this behavior you should use the XMLNSC parser. When writing a message tree, if an xsi:nil attribute exists it will be produced in the same way as any other attribute.

Related concepts:

"XMLNS parser" on page 1104
The XMLNS parser is a flexible, general-purpose XML parser.

"XMLNS opaque parsing"
Opaque parsing is a performance feature that is offered by the XMLNS domain.

"Handling null values" on page 1140
A business message might contain fields that are either empty or have a specific out-of-range value. In these cases, the application that receives the message is expected to treat the field as if it did not have a value. The logical message tree supports this concept by enabling the value of any element to be set to NULL.

XMLNS opaque parsing:

Opaque parsing is a performance feature that is offered by the XMLNS domain.

XMLNS opaque parsing has been superseded by the opaque parsing feature of the XMLNSC domain. Do not use the XMLNS parser for opaque parsing unless your message flow requires features that are only offered by the XMLNS parser.

If you are designing a message flow, and you know that a particular element in a message is never referenced by the message flow, you can specify that that element is to be parsed opaquely. This reduces the costs of parsing and writing the message, and might improve performance in other parts of the message flow.

To specify that an XML element is to be parsed opaquely, use an **ESQL CREATE** statement with a PARSE clause to parse the XML document. Set the FORMAT qualifier of the PARSE clause to the constant, case-sensitive string 'XMLNS_OPAQUE' and set the TYPE qualifier of the PARSE clause to the name of the XML element that is to be parsed in an opaque manner.

The TYPE clause can specify the element name with no namespace (to match any namespace), or with a namespace prefix or full namespace URI (to match a specific namespace).

XMLNS opaque elements cannot be specified via the node properties.

Consider the following example:

```
DECLARE soap NAMESPACE 'http://schemas.xmlsoap.org/soap/envelope/';

DECLARE BitStream BLOB ASBITSTREAM(InputRoot.XMLNS
                  ENCODING InputRoot.Properties.Encoding
                  CCSID InputRoot.Properties.CodedCharSetId);
--No Namespace
```

```
  CREATE LASTCHILD OF OutputRoot
    DOMAIN('XMLNS')
          PARSE (BitStream
                 ENCODING InputRoot.Properties.Encoding
                 CCSID InputRoot.Properties.CodedCharSetId
                 FORMAT 'XMLNS_OPAQUE'
                 TYPE 'Body');

--Namespace Prefix
  CREATE LASTCHILD OF OutputRoot
    DOMAIN('XMLNS')
          PARSE (BitStream
                 ENCODING InputRoot.Properties.Encoding
                 CCSID InputRoot.Properties.CodedCharSetId
                 FORMAT 'XMLNS_OPAQUE'
                 TYPE 'soap:Body');

--Namespace URI
  CREATE LASTCHILD OF OutputRoot
    DOMAIN('XMLNS')
          PARSE (BitStream
                 ENCODING InputRoot.Properties.Encoding
                 CCSID InputRoot.Properties.CodedCharSetId
                 FORMAT 'XMLNS_OPAQUE'
                 TYPE '{http://schemas.xmlsoap.org/soap/envelope/}:Body');
```

Related concepts:

"XMLNS parser" on page 1104
The XMLNS parser is a flexible, general-purpose XML parser.

"XMLNSC opaque parsing" on page 1097
Opaque parsing is a performance feature that is offered by the XMLNSC domain.

Related reference:

"CREATE statement" on page 5082
The CREATE statement creates a new message field.

XMLNS DTD support:

The input XML might contain an inline DTD.

Parsing

If the input XML document has an inline DTD, the XMLNS parser reads and uses information in the DTD while parsing, and adds the DTD information to the message tree.

Internal entity definitions in the DTD are used to automatically expand entity references that are encountered in the body of the document.

Attributes that are missing from the input document are automatically supplied with the default value specified in the DTD.

Writing

The XMLNS parser can produce any inline DTD that has been constructed in the message tree.

External DTDs

No support is offered for external DTDs

Related concepts:

"XMLNS parser" on page 1104
The XMLNS parser is a flexible, general-purpose XML parser.

"XMLNSC opaque parsing" on page 1097
Opaque parsing is a performance feature that is offered by the XMLNSC domain.

Related reference:

"CREATE statement" on page 5082
The CREATE statement creates a new message field.

XML parsers namespace support:

Namespaces in XML messages are supported by the XMLNSC and XMLNS parsers. Namespaces are not supported by the XML parser.

Parsing

The XMLNS and XMLNSC parsers can parse any well-formed XML document, whether or not the document contains namespaces. If elements or attributes have namespaces, those namespaces are applied to the elements and attributes in the message tree. Namespace prefix mappings are also carried in the message tree, and are used when serializing the message tree back to XML.

- If an element or attribute in the input XML has a namespace, the corresponding node in the message tree also has that namespace.
- If an element contains a namespace declaration (an xmlns attribute), a child element that contains its prefix and namespace URI is created in the message tree.

While the message is passing through a message flow, namespaces and namespace mappings can be modified using ESQL or any of the other transformation technologies that are offered by message broker.

Writing

Namespaces and their prefixes are preserved in the message tree when parsing, and are used when the XMLNS and XMLNSC parsers convert a message tree to an XML bit stream.

- When serializing a message tree, the parser scans for namespace declarations on each XML element. If any are found, it uses them to select the namespace prefixes in the output document.
- If an element in the message tree has a namespace, but there is no in-scope namespace declaration for its namespace URI, a valid namespace prefix is automatically generated and used in the output XML. Auto-generated prefixes have the form NS1, NS2, and so on.

Tip: If an element in the message tree has a child element that is a 'default namespace' declaration, every child of that element (whether an XML element or an XML attribute, at any nesting depth) must have a namespace. If this rule is not enforced message broker cannot generate correct XML output for the message tree.

Related concepts:

"XML parsers and domains" on page 1084
You can use XML domains to parse and write messages that conform to the W3C XML standard.

"XMLNSC parser" on page 1090
The XMLNSC parser is a flexible, general-purpose XML parser that offers high performance XML parsing and optional XML Schema validation.

"XMLNS parser" on page 1104
The XMLNS parser is a flexible, general-purpose XML parser.

"Namespaces in the message model" on page 1201
Use namespaces to qualify message model object names.

Related tasks:

"Manipulating messages in the XMLNSC domain" on page 2546
If you are writing ESQL to process messages in the XMLNSC domain, it is helpful to learn about the structure of the message tree that the XMLNSC parser builds.

"Manipulating messages in the XMLNS domain" on page 2563
When you write ESQL for processing messages in the XMLNS domain, it is helpful to understand the structure of the message tree that the XMLNS parser builds.

XML parser:

The XML domain is very similar to the XMLNS domain, but the XML domain has no support for XML namespaces or opaque parsing.

The XML domain is deprecated, but existing message flows that use the XML domain continue to work. Use the XMLNSC domain when developing new message flows.

The XML parser is not model-driven and does not use an XML Schema when parsing XML documents.

If you want the XML domain to parse a particular message, you must select *Message Domain* as `XML` on the appropriate node in the message flow.

Tip: The XMLNSC and XMLNS parsers both support XML messages that do not use namespaces, with no extra configuration.

Features of the XML parser

Feature	Present	Description
Namespace support	No	
On-demand parsing	Yes	See "Parsing on demand" on page 4173.
Compact message tree	No	
Opaque parsing	No	
Ultra high performance	No	
Validation	No	
Inline DTD support	Yes	Inline DTDs are processed and retained in the message tree.
XML Data Model compliance	Yes	The resultant message tree conforms to the XML Data Model.

Related concepts:

"XML parsers and domains" on page 1084
You can use XML domains to parse and write messages that conform to the W3C XML standard.

"XMLNS parser" on page 1104
The XMLNS parser is a flexible, general-purpose XML parser.

"XMLNSC parser" on page 1090
The XMLNSC parser is a flexible, general-purpose XML parser that offers high performance XML parsing and optional XML Schema validation.

"Which XML parser should you use?" on page 1080
If your messages are general-purpose XML documents, you can use one of the dedicated XML domains (XMLNSC or XMLNS) to parse the message, or you can use the MRM domain to parse the message.

"Why model messages?" on page 1158
WebSphere Message Broker supplies a range of parsers to parse and write message formats. Some message formats are self-defining and can be parsed without reference to a model. Most message formats, however, are not self-defining, and the parser must have access to a predefined model that describes the message, if it is to parse it correctly.

Related tasks:

"Manipulating messages in the XML domains" on page 2535
You can manipulate messages in the XML, XMLNS, and XMLNSC domains.

Related reference:

"Parsing on demand" on page 4173
On-demand parsing, referred to as partial parsing, is used to parse an input message bit stream only as far as is necessary to satisfy the current reference. The parsers that can perform partial parsing of input messages are the JSON, MRM, XML, XMLNS, and XMLNSC parsers.

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

MRM parser and domain:

You can use the MRM domain to parse and write a wide range of message formats.

The MRM domain can be used to parse and write a wide variety of message formats. It is primarily intended for non-XML message formats, but it can also parse and write XML. For guidance on when to consider using the MRM parser, instead of one of the XML parsers, to parse XML, see "Which XML parser should you use?" on page 1080

The key features of the MRM domain are:

- Support for messages from applications that are written in C, COBOL, PL/I and other languages, by using the Custom Wire Format (CWF) physical format. This support includes the ability to create a message model directly from a C header file or COBOL copybook.
- Support for text messages, perhaps with field content that is identified by tags, separated by specific delimiters, or both, by using the Tagged Delimited String (TDS) physical format. This includes industry standards such as CSV, HL7, SWIFT, EDIFACT, and X12.
- Support for XML messages, including those that use XML namespaces, by using the XML physical format.

WebSphere Message Broker uses the MRM parser to read and write messages that belong to the MRM domain. When reading a message, the MRM parser constructs

a message tree from a bit stream. When writing a message, the MRM parser creates a bit stream from a message tree. The MRM parser is always model-driven, and it is guided by a message dictionary that describes the shape of the message tree (the logical model) and the physical layout of the bytes or characters in the bit stream (the physical format). A message dictionary is created automatically from the content of a message set when it is added to the broker archive (BAR) file. Therefore, when you create a message set for use with the MRM domain, you must define both the logical model and the appropriate physical format information.

The operation of the parser depends on the physical format that you have associated with the input or output message:

- For a binary message, the parser reads a set sequence of bytes according to information in the CWF physical format, and translates them into the fields and values in the message tree.
- For a text message, the parser uses a key piece of TDS physical format information called *Data Element Separation* to decide how to parse each portion of the message bit stream. This informs the parser whether the message uses delimiters, tags, fixed length elements, patterns, and so on. The parser then reads the data according to information in the TDS physical format, and translates it into the fields and values in the message tree.
- For an XML message, the parser reads the XML markup language (element tags and attributes), guided by information in the XML physical format, and translates them into the fields and values in the message tree.

Because the MRM parser is model-driven, it can perform validation of messages against the model that is defined in the deployed dictionary. The level of validation that is performed by the MRM parser is similar to that defined by XML Schema 1.0, but is not fully compliant. If you use XML messages, and you want fully compliant XML Schema 1.0 validation, use the XMLNSC domain.

The MRM parser is an on-demand parser. See "Parsing on demand" on page 4173.

If you want to use the MRM domain to parse a particular message:

1. Create a new message set with an appropriate CWF, TDS, or XML physical format; or locate an existing message set.
2. Ensure that the message set has its *Default message domain* set to `MRM`, or that the *MRM* check box under *Supported message domains* is selected to indicate that the message set supports the MRM domain.
3. Create a message definition file in the message set to represent your message, ensuring that both logical and physical format information is provided. If you have an existing C, COBOL, XML Schema, or DTD description of your message, you can import the description by using a wizard.
4. Add the message set to a broker archive (BAR) file which will generate a message dictionary for use by the MRM parser, and deploy the BAR file to the broker.
5. Select `MRM` as *Message Domain* on the appropriate node in your message flow.
6. Additionally set values for *Message Set*, *Message Type*, and *Message Format* on the node. *Message Type* is the name of the message in the message definition file.

The following samples all use the MRM parser to process messages:

- Video Rental
- Comma Separated Value (CSV)
- EDIFACT

- FIX
- SWIFT
- X12

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Some predefined message models are supplied with the WebSphere Message Broker Toolkit and can be imported by using the New Message Definition File From IBM supplied Message wizard. The CSV, ALE IDoc, and File IDoc models are specifically for use with the MRM domain. See "IBM supplied messages that you can import" on page 6367.

IBM supplies predefined message sets for industry standard formats SWIFT, X12, EDIFACT, and FIX. For more information, contact Dublin Adapters at dubadapt@ie.ibm.com.

Related concepts:

"Parsers" on page 1072
A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from the internal message tree representation.

"Which XML parser should you use?" on page 1080
If your messages are general-purpose XML documents, you can use one of the dedicated XML domains (XMLNSC or XMLNS) to parse the message, or you can use the MRM domain to parse the message.

"XML parsers and domains" on page 1084
You can use XML domains to parse and write messages that conform to the W3C XML standard.

"Correlation names" on page 1069
A correlation name is a field reference that identifies a well-defined starting point in the logical message tree and is used in field references to describe a standard part of the tree format.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and

Filter nodes in your message flows by coding ESQL.

"Developing Java" on page 2628
When you use the JavaCompute node, you customize it to determine the exact processing that it provides.

"Manipulating message body content" on page 2418
The message body is always the last child of root, and is identified by its parser name, for example XML or MRM. You can refer to, modify, and create message body data.

"Manipulating messages in the MRM domain" on page 2581
How to use messages that have been modeled in the MRM domain, and that are parsed by the MRM parser.

Related reference:

"IBM supplied messages that you can import" on page 6367
You can import IBM supplied messages to create a new message definition file.

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Parsing on demand" on page 4173
On-demand parsing, referred to as partial parsing, is used to parse an input message bit stream only as far as is necessary to satisfy the current reference. The parsers that can perform partial parsing of input messages are the JSON, MRM, XML, XMLNS, and XMLNSC parsers.

DataObject parser and domain:

Use the DataObject domain to parse and write messages for WebSphere Adapters and CORBA applications.

- "Using the DataObject domain with WebSphere Adapters"
- "Using the DataObject domain with CORBA" on page 1115

Using the DataObject domain with WebSphere Adapters: You must use the DataObject domain when you use WebSphere Adapter nodes in your message flow.

WebSphere Message Broker uses the DataObject parser to read and write message from Enterprise Information Systems (EIS) such as SAP, PeopleSoft, and Siebel. Such messages belong to the DataObject domain.

When it receives a message from an adapter, the DataObject parser constructs a message tree from the business object that it receives from the EIS. When it writes a message, the DataObject parser creates from the message tree the business object that it sends to the EIS. The DataObject parser is always *model-driven*, and it is guided by the XML Schemas that model the EIS business objects. The XML Schemas are created automatically from the content of a message set when the message set is added to the broker archive (BAR) file.

If you want to parse a message using the DataObject domain, you must:

1. Create a message set, or locate an existing message set.
2. Ensure that either the message set has its *Default message domain* project set to `DataObject`, or the *DataObject* check box (under *Supported message domains*) is selected, to indicate that the message set supports the DataObject domain.
3. Create a message definition file in the message set to represent your EIS business object. Use the *New adapter connection* wizard to connect to the EIS and retrieve the Business object metadata.

4. Add the message set to a broker archive (BAR) file, which generates XML Schema for the DataObject parser to use, and deploy the BAR file to the broker.
5. If you associate your adapter inbound or outbound message with an adapter node in your message flow, the *Message Set* property is automatically set in the node. The *Message domain* property is always pre-selected as `DataObject`.

Tip: If a message that belongs to the DataObject domain is written to a destination other than a WebSphere Adapter, the DataObject parser invokes the XMLNSC parser to write the message as XML.

The following adapter samples use the DataObject parser to process messages:

- SAP Connectivity
- Twineball Example EIS Adapter

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Using the DataObject domain with CORBA: You must use the DataObject domain when you use CORBA nodes in your message flow.

WebSphere Message Broker uses the DataObject parser to read and write message from CORBA applications. Such messages belong to the DataObject domain.

For information about how to build the tree under the DataObject domain for use with CORBA, see "CORBA operation parameters" on page 2156 and "IDL data types" on page 2150.

Related concepts:

"Parsers" on page 1072
A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from the internal message tree representation.

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"CORBA nodes" on page 2147
Use CORBA nodes to connect WebSphere Message Broker with CORBA Internet Inter-Orb Protocol (IIOP) applications.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Connecting to an external CORBA application" on page 2159
Connecting to an external CORBA application involves importing an IDL file, creating a message flow, building a message, and processing the response from the CORBARequest node.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

"SiebelInput node" on page 4740
Use the SiebelInput node to interact with a Siebel application.

"SiebelRequest node" on page 4745
Use the SiebelRequest node to interact with a Siebel application.

"PeopleSoftInput node" on page 4630
Use the PeopleSoftInput node to interact with a PeopleSoft application.

"PeopleSoftRequest node" on page 4635
Use the PeopleSoftRequest node to interact with a PeopleSoft application.

"TwineballInput node" on page 4951
Use the TwineballInput node to discover how the WebSphere Adapters nodes work.

"TwineballRequest node" on page 4955
Use the TwineballRequest node to discover out how WebSphere Adapters nodes work.

"CORBARequest node" on page 4349
Use the CORBARequest node to call an external CORBA application over Internet Inter-Orb Protocol (IIOP).

JMS parsers and domains:

The JMSMap and JMSStream domains can be used for modeling messages that are produced by the implementations of the Java Messaging Service standard.

Use the JMSMap domain when handling JMS messages of type MapMessage. Use the JMSStream domain when handling JMS messages of type StreamMessage.

These message types appear in the broker in XML format, and are therefore supported in an identical way to XML domain messages.

For a full description of JMS MapMessage and StreamMessage processing, see "WebSphere Broker JMS Transport" on page 1681.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Developing Java" on page 2628
When you use the JavaCompute node, you customize it to determine the exact processing that it provides.

"Manipulating messages in the JMS domains" on page 2609
This topic provides information specific to dealing with messages that belong to the JMSMap and JMSStream domains. These messages are parsed by the generic XML parser.

Related reference:

"WebSphere Broker JMS Transport" on page 1681
The WebSphere Broker JMS Transport is a service that connects applications that send and receive messages that conform to the Java Message Service (JMS) standard.

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

MIME parser and domain:

Use the MIME domain if your messages use the MIME standard for multipart messages.

The MIME (Multipurpose Internet Mail Extension) parser does not support the full MIME standard, but does support common uses of MIME. You can send the messages to the broker over HTTP or over other transport types, such as WebSphere MQ. Use the MIME domain if your messages use the MIME standard for multipart messages.

The MIME domain does not support Content-Type values with a media type of *message*.

To specify that a message uses the MIME domain, select `MIME` as the *Message Domain* on the relevant message flow node.

Use the MIME domain and parser to parse and write MIME messages. The MIME parser creates a logical tree, and sets up the broker ContentType property. You can use Compute nodes and JavaCompute nodes to manipulate the logical tree. Set the Content-Type value using the ContentType property in the MIME domain.

Example MIME message

The following example shows a simple multipart MIME message. The message shown is a SOAP with Attachments message with two parts: the root part and one attachment part. The boundary string *MIME_boundary* delimits the parts.

```
MIME-Version: 1.0
Content-Type: Multipart/Related; boundary=MIME_boundary; type=text/xml
Content-Description: Optional description of message.

Optional preamble text
--MIME_boundary
Content-Type: text/xml; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-ID: <rootpart@example.com>

<?xml version='1.0' ?>
<SOAP-ENV:Envelope
          xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">

    <SOAP-ENV:Header xmlns:ins="http://myInsurers.com">
       <ins:ClaimReference>abc-123</ins:ClaimReference>
    </SOAP-ENV:Header>

    <SOAP-ENV:Body xmlns:ins="http://myInsurers.com">
       <ins:SendClaim>
          <ins:ClaimDetail>myClaimDetails</ins:ClaimDetail>
          <ins:ClaimPhoto>
            <href>cid:claimphoto@example.com</href>
          </ins:ClaimPhoto>
       </ins:SendClaim>
    </SOAP-ENV:Body>

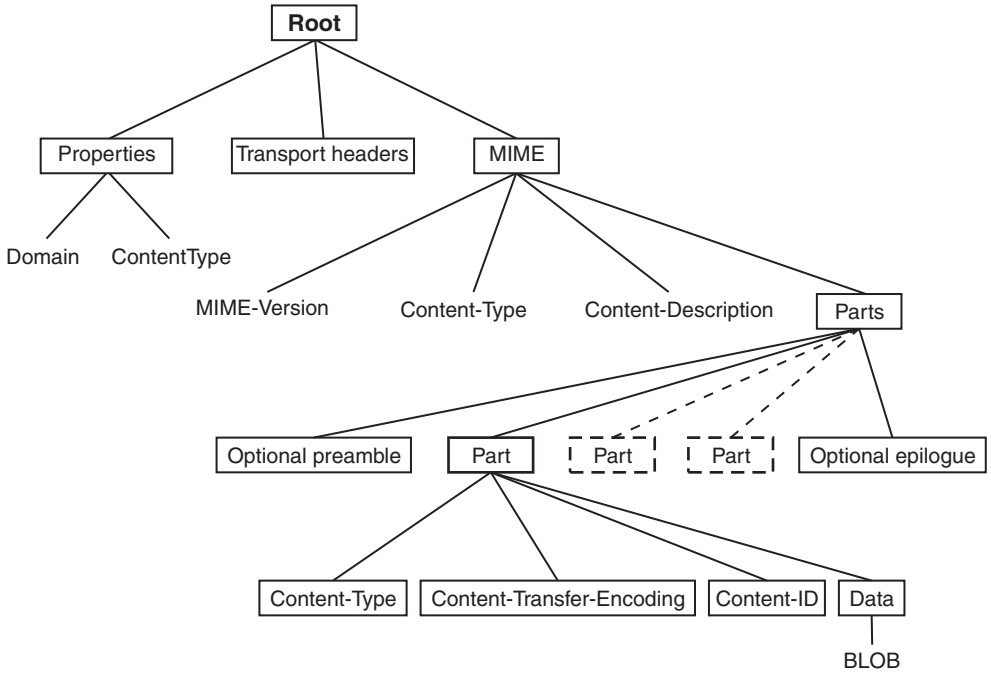
</SOAP-ENV:Envelope>

--MIME_boundary
Content-Type: application/octet-stream
Content-Transfer-Encoding: binary
Content-ID: <claimphoto@example.com>

myBinaryData
--MIME_boundary--
Optional epilogue text
```

Example MIME logical tree

The following diagram shows a MIME logical tree, which is described in "MIME tree details" on page 1123. A MIME logical tree does not need to contain all of the children that are shown in the diagram. The value of the Content-Type header of a MIME message is the same as the ContentType field in the Properties subtree. The Transport-headers are headers from the transport that is used, such as an MQMD or HTTP.



You can further parse the BLOB data in the tree (for example, by using an ESQL CREATE statement) if you know about the format of that MIME part. You might be able to find information about the format from its Content-Type field in the logical tree. Alternatively, you might know the format that your MIME messages take, and be able to parse them appropriately. For example, you might know that the first MIME Part is always an XML message, and that the second MIME Part is a binary security signature.

When the EmailInput node receives an email from an email server that supports Post Office Protocol 3 (POP3) or Internet Message Access Protocol (IMAP), the body of the email message, and any attachments, are propagated in the MIME domain. All other information relating to the email is stored in the `Root.`*`Transport headers`* MIME logical tree; for example, `Root.EmailInputHeader.To`. Where `To` is the storage location of one of the email elements. For a complete list of the email elements that are propagated in the MIME logical tree when you use an EmailInput node, see “EmailInput node” on page 4394.

You must specify how to parse other message formats, such as tagged delimited or binary data, within your message flow, because the MIME parser does not do this. You must also specify how to handle encoded and signed message parts, because the MIME parser does not process these.

Some pre-defined MIME message models are supplied with the WebSphere Message Broker Toolkit and can be imported using the New Message Definition From IBM Supplied Message wizard.

Related concepts:

“Parsers” on page 1072
A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from the internal message tree representation.

"MIME tree details" on page 1123
A MIME message is represented in the broker as a logical tree. When it writes a message, the MIME parser creates a message bit stream by using the logical message tree.

"MIME messages"
A MIME message consists of both data and metadata. MIME metadata consists of HTTP-style headers and MIME boundary delimiters.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Developing Java" on page 2628
When you use the JavaCompute node, you customize it to determine the exact processing that it provides.

"Processing email messages" on page 1786
You can configure the EmailOutput node to deliver an email from a message flow to an email server that supports Simple Mail Transfer Protocol (SMTP). You can also configure the EmailInput node to retrieve an email from an email server that supports Post Office Protocol 3 (POP3) or Internet Message Access Protocol (IMAP).

Related reference:

"IBM supplied messages that you can import" on page 6367
You can import IBM supplied messages to create a new message definition file.

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

MIME messages:

A MIME message consists of both data and metadata. MIME metadata consists of HTTP-style headers and MIME boundary delimiters.

MIME headers

Each header is a colon-separated name-value pair on a line. The ASCII sequence <CR><LF> terminates the line. A sequence of these headers, called a header block, is terminated by a blank line: <CR><LF><CR><LF>. Any headers that are in this HTTP style can appear in a MIME document. Some common MIME headers are described in "MIME standard header fields" on page 6323.

Content-Type

The only header that must be present is the *Content-Type* header. This header specifies the type of the data in the message. If the Content-Type value starts with "multipart", the message is a multipart MIME message. For multipart messages the Content-Type header must also include a boundary attribute that gives the text that is used to delimit the message parts. Each MIME part has its own

Content-Type field that specifies the type of the data in the part. This can also be multipart, which allows multipart messages to be nested. MIME parts with any other Content-Type values are handled as BLOB data.

If a MIME document is sent over HTTP, the Content-Type header appears in the HTTP header block rather than in the MIME message body. For this reason, the broker manages the value of the Content-Type header as the ContentType property in the *Properties* folder of the logical tree. This allows the MIME parser to obtain the Content-Type value for a MIME document that is received over HTTP. If you need to either create a new MIME tree or modify the value of the Content-Type, set the Content-Type value using the ContentType property in the MIME domain. If you set the Content-Type value directly in the MIME tree or HTTP tree, this value might be ignored or used inconsistently. The following ESQL is an example of how to set the broker ContentType property:

```
SET OutputRoot.Properties.ContentType = 'text/plain';
```

Parsing

The MIME domain does not enforce the full MIME specification. Therefore, you can work with messages that might not be valid in other applications. For example, the MIME parser does not insist on a **MIME-Version** header. The MIME parser imposes the following constraints:

- The MIME headers must be properly formatted:
  - Each header is a colon-separated name-value pair, on a line of its own, terminated by the ASCII sequence <CR><LF>.
  - The header line must use 7-bit ASCII.
  - Semicolons are used to separate parameters:

    ```
    Content-Type: Multipart/Related; boundary=MIME_boundary; type=text/xml
    ```

  - A header might contain a comment in parentheses, for example:

    ```
    MIME-Version: 1.0 (Generated by XYZ)
    ```

- A line that starts with white space is treated as a continuation of the previous line. Therefore, a long header can be split across more than one line.
- If two or more headers in a header block have the same name, their values are concatenated into a comma-separated list.
- A top-level MIME Content-Type header must be available. The header is not case-sensitive. If the transport is HTTP, any Content-Type value in the HTTP header is used as the top-level Content-Type. If the transport is not HTTP, the Content-Type must appear in the initial header block of the MIME message.
- The Content-Type value is a media type followed by the / character and a subtype. Examples of this are `text/xml` and `multipart/related`. The parser does not validate subtypes. The Content-Type value can be followed by one or more parameters that are separated by semicolons.
- If the media type of a message is multipart, a boundary attribute must provide the text that is used to delimit the separate MIME parts.
- Each individual MIME part can have its own Content-Type header. The part header can have a media type of multipart, so that multipart messages can be nested. In this case, a valid boundary attribute must be provided, and its value must be different from any that has been previously defined in the message. MIME parts that have any other Content-Type value are handled as BLOB data.
- MIME multipart boundary delimiters are represented in 7-bit ASCII. The boundary delimiter consists of a line starting with a hyphen pair, followed by a boundary string. This sequence must not occur within the MIME message at any

point other than as a boundary. A MIME end-delimiter is a hyphen pair, followed by the MIME boundary string, followed by a further hyphen pair. All delimiter lines must end in the ASCII sequence <CR><LF>. An example of a delimited message is:

```
--MIME_boundary
message data
--MIME_boundary
message data
--MIME_boundary--
```

where *MIME_boundary* is the boundary delimiter string, and *message data* represents message data.

- The MIME media type *message* is not supported and results in an error at run time.
- Any preamble data (text between the initial MIME header block and the first boundary delimiter) or epilogue data (text after the final boundary delimiter) is stored in the logical tree as a value-only element. Preamble data and epilogue data can appear only as the first and last children, respectively, of a Parts node.
- The MIME parser does not support on-demand parsing and ignores the *Parse Timing* property. The parser does not validate MIME messages against a message model, and ignores the WebSphere Message Broker Toolkit *Validate* property.

Special cases of multipart MIME

The MIME parser is intended primarily for use with multipart MIME messages. However, the parser also handles some special cases:

- Multipart MIME with just one part. The logical tree for the MIME part saves the Content-Type and other information as usual, but the Data element for the attachment is empty.
- Single-part MIME. For single-part MIME, the logical tree has no Parts child. The last child of the MIME tree is the Data element. The Data element is the parent of the BLOB that contains the message data.
- MIME parts with no content.

Secure MIME (S/MIME)

S/MIME is a standard for sending secure email messages. S/MIME has an outer level Content-Type of *multipart/signed* with parameters **protocol** and **micalg** that define the algorithms that are used to encrypt the message. One or more MIME parts can have encoded content. These parts have Content-Type values such as *application/pkcs7-signature* and a Content-Transfer-Encoding of *base64*. The MIME domain does not attempt to interpret or verify whether the message is signed.

Related concepts:

"MIME tree details" on page 1123
A MIME message is represented in the broker as a logical tree. When it writes a message, the MIME parser creates a message bit stream by using the logical message tree.

"Logical tree structure" on page 1042
The logical tree structure is the internal (broker) representation of a message. It is also known as the message assembly.

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

Related tasks:

"Manipulating message body content" on page 2418
The message body is always the last child of root, and is identified by its parser name, for example XML or MRM. You can refer to, modify, and create message body data.

Related reference:

"MIME standard header fields" on page 6323
Check this quick reference to the common MIME headers.

MIME tree details:

A MIME message is represented in the broker as a logical tree. When it writes a message, the MIME parser creates a message bit stream by using the logical message tree.

Logical tree elements

A MIME message is represented in the broker as a logical tree with the following elements:

- The root of the tree is a node called MIME.
- All correctly formatted headers are stored in the logical tree, regardless of whether they conform to the MIME standard. The headers appear in the logical tree as name=value, as shown here:

  ```
  Content-Type=text/xml
  ```
- A multipart MIME message is represented by a subtree with a root node called Parts.
- Any preamble or epilogue data associated with a multipart MIME message is represented by value-only elements appearing as the first and last children of Parts.
- In the special case of single-part MIME, the content is represented by a subtree with the root called Data.
- Each part of a multipart MIME message is represented by an element called Part with a child element for each MIME header, and a last child called Data.
- The Data element represents the content of a MIME part. This makes it easier to test for the presence of body content by using ESQL because the Data element is always the last child of its parent.

Writing MIME messages

When it writes a message, the MIME parser creates a message bit stream by using the logical message tree. The MIME domain does not enforce all of the constraints that the MIME specification requires, therefore it might generate MIME messages that do not conform to the MIME specification. The constraints that the MIME parser imposes are:

- The tree must have a root called MIME, and constituent Parts, Part, and Data elements, as described in "Logical tree elements."
- Exactly one Content-Type header must be present at the top level of the tree, or be available by using the ContentType property. Media subtypes are not validated.
- If the media type is *multipart*, a valid boundary parameter must also exist.
- Any constituent MIME parts can have exactly one Content-Type header. If the value of this header starts with *multipart*, it must also include a valid boundary parameter. The value of this boundary parameter must not be the same as other boundary parameter values in the definition.

- The MIME Content-Type value "message" is not supported and results in an error at run time.
- All name-value elements in the tree are written as `name: value` followed by the ASCII sequence `<CR><LF>`.

If you have other elements in the tree, the parser behaves in the same way as the HTTP header parser:

- A name-only element or a NameValue element with a NULL value results in `Name: NULL` .
- Any children of a name-value element are ignored.

The message flow must serialize subtrees if they exist; you can use the ESQL command **ASBITSTREAM**.

Related concepts:

"MIME parser and domain" on page 1117
Use the MIME domain if your messages use the MIME standard for multipart messages.

"MIME messages" on page 1120
A MIME message consists of both data and metadata. MIME metadata consists of HTTP-style headers and MIME boundary delimiters.

"Logical tree structure" on page 1042
The logical tree structure is the internal (broker) representation of a message. It is also known as the message assembly.

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

Related tasks:

"Manipulating message body content" on page 2418
The message body is always the last child of root, and is identified by its parser name, for example XML or MRM. You can refer to, modify, and create message body data.

BLOB parser and domain:

The BLOB message domain includes all the messages with content that cannot be interpreted and subdivided into smaller sections of information.

Messages in this domain are processed by the BLOB parser. The BLOB parser is a program that interprets a bit stream or message tree that represents a message that belongs to the BLOB domain. The parser then generates the corresponding tree from the bit stream on input, or a bit stream from the tree on output.

A BLOB message is handled as a single string of bytes, and although you can manipulate it, you cannot identify specific pieces of the byte string using a field reference, in the way that you can with messages in other domains.

You can process messages in the BLOB domain in the following ways:

- You can refer to the message content if you know the location (offset) of particular information within the message. You can specify offset values in ESQL statements within nodes in a message flow to manipulate the information.
- You can store the message in an external database, in whole or in part (where the part is identified by the offset of the data that is to be stored).

- You can use the Mapping node to map to and from a predefined BLOB message, and to map to and from items of BLOB data. The BLOB message cannot be:
  - The message content in a message where `Content Validation` is defined as Open or Open Defined (for example, the message body of a SOAP envelope)
  - The message represented by a wildcard inside another message

  The UnknownParserName field is ignored.

The BLOB message body parser does not create a tree structure in the same way that other message body parsers do. It has a root element BLOB, which has a child element, also called BLOB, which contains the data.

For example, `InputBody.BLOB.BLOB[10]` identifies the tenth byte of the message body; `substring(InputBody.BLOB.BLOB from 10 for 10)` references 10 bytes of the message data starting at offset 10.

If you want to use the BLOB parser to parse a particular message, select `BLOB` as the `Message Domain` on the relevant node in your message flow.

The following sample demonstrates how you can extract information from an XML message and transform it into BLOB format to store it in a database.

- Data Warehouse

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Related concepts:

"Parsers" on page 1072
A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from the internal message tree representation.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Logical tree structure" on page 1042
The logical tree structure is the internal (broker) representation of a message. It is also known as the message assembly.

"Correlation names" on page 1069
A correlation name is a field reference that identifies a well-defined starting point in the logical message tree and is used in field references to describe a standard part of the tree format.

Related tasks:

"Manipulating messages in the BLOB domain" on page 2615
How to deal with messages that belong to the BLOB domain, and that are parsed by the BLOB parser.

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Developing Java" on page 2628
When you use the JavaCompute node, you customize it to determine the exact processing that it provides.

"Manipulating message body content" on page 2418
The message body is always the last child of root, and is identified by its parser name, for example XML or MRM. You can refer to, modify, and create message body data.

"Creating a BLOB output message using a message map" on page 2262
Use the Message Mapping editor to create a bit stream from a message source, and create it as a BLOB output message.

"Storing a BLOB message in a database table using a message map" on page 2297
Use the Message Mapping editor to create a bit stream from a BLOB message, and store it in a database table.

"Mapping from a BLOB message to an output message" on page 2263
Use the Message Mapping editor to parse a BLOB message.

"Mapping from a BLOB field in a database table to an output message" on page 2298
Use the Message Mapping editor to parse a bit stream from a field in a database table into a folder in a target message.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

IDOC parser and domain:

The IDOC domain can be used to process messages that are sent to the broker by SAP R3 clients across the WebSphere MQ link for R3. Such messages are known as SAP ALE IDocs.

Note: The IDOC domain is deprecated and is not recommended for developing new message flows. Instead use the MRM domain with a TDS physical format. See "MRM parser and domain" on page 1111.

A typical ALE IDoc message that has been sent from SAP to the WebSphere MQ link for R3 consists of an MQMD header, an MQSAPH header, and the ALE IDoc itself. The IDoc is made up of fixed size structures:

- The first structure is the Control Structure (DC). This is a complex element 524 bytes long that contains a fixed set of SAP-defined simple elements.
- One or more Data Structures (DDs). Each DD is a complex element 1063 bytes long that contains a fixed set of SAP-defined simple elements that occupies 63 bytes, followed by 1000 bytes of user-defined segment data.

WebSphere Message Broker uses the IDOC parser to read and write ALE IDocs that belong to the IDOC domain. When reading a message, the IDOC parser constructs a message tree from a bit stream. When writing a message, the IDOC parser creates a bit stream from a message tree.

The IDOC parser processes the SAP-defined elements in the DC, then, for each DD, the IDOC parser processes the SAP-defined elements, then calls the MRM parser to process the user-defined segment data, using its CWF physical format.

The IDOC parser is therefore a model-driven parser, and requires that you create a message set in which to model the IDoc message, and deploy it to the broker.

If you want the IDOC domain to parse a particular message, you must:

1. Create a new message set with a CWF physical format, or locate an existing message set.
2. Ensure that either the message set has its *Default message domain* project set to `IDOC`, or the *IDOC* check box (under *Supported message domains*) is selected, to indicate that the message set supports the IDOC domain.
3. Create message definition files in the message set to represent your message. See "Building the message model for the IDOC parser" on page 6330 for the steps involved.
4. Add the message set to a broker archive (BAR) file which generates a message dictionary for use by the MRM parser, and deploy the BAR file to the broker.
5. Select *Message Domain* as `IDOC` on the appropriate node in your message flow.
6. Additionally, select *Message Set* and *Message Format* on the node. (You do not need to select *Message Type*).

Related concepts:

"Parsers" on page 1072
A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from the internal message tree representation.

"MRM parser and domain" on page 1111
You can use the MRM domain to parse and write a wide range of message formats.

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

Related tasks:

"Building the message model for the IDOC parser" on page 6330
The ALE IDoc messages that are sent to, and received from, SAP applications by using the WebSphere MQ Link for R3, can be processed by the IDOC parser, which requires a message model to interpret the data correctly. This topic describes how to build the message model.

"Manipulating messages in the IDOC domain" on page 2610
Use ESQL from a Compute node to copy the incoming IDoc to the outgoing IDoc, and manipulate the message.

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Developing Java" on page 2628
When you use the JavaCompute node, you customize it to determine the exact processing that it provides.

"Manipulating messages in the MRM domain" on page 2581
How to use messages that have been modeled in the MRM domain, and that are parsed by the MRM parser.

Related reference:

"Field names of the IDOC parser structures" on page 6333
The field names of the Control Structure (DC) and the Data Structure (DD) that are used by the IDOC parser.

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

JSON parser and domain:

JSON (JavaScript Object Notation) is a simple data-interchange format based on a subset of the JavaScript programming language.

WebSphere Message Broker provides support for a JSON domain. Messages in the JSON domain are processed by the JSON parser and serializer. The JSON parser interprets a bit stream using the JSON grammar, and generates a corresponding JSON domain logical message tree in the broker. When processing data for output, the JSON serializer generates a JSON formatted bit stream from a JSON domain logical message tree.

The JSON parser and serializer do not support message validation because JSON message modeling is not supported by the broker.

To enable function that becomes available in WebSphere Message Broker fix packs, use the **-f** parameter on the **mqsichangebroker** command. For more information, see "**mqsichangebroker** command" on page 3723.

JSON is a language-independent text format, based on two structures:

- Objects (name-value pairs) with the following types:
  - String
  - Number
  - Boolean
  - Null
- Ordered collections of values (arrays)

Objects and arrays can be nested.

For more information about JSON message structure, see "JSON message details" on page 1135.

JSON data streams can be parsed from any coded character set ID (CCSID) that is supported by the broker. The data stream is parsed according to the CCSID that is defined by the transport when the JSON parser is invoked from an input or request node, or defined by the CCSID parameter in a PARSE clause on a CREATE function call. If no CCSID is specified, or if a value of 0 is set, the parser attempts to detect (by examining the first few bytes of the data stream) whether one of the following Unicode encodings is being used:

- UTF-8
- UTF-16BE
- UTF-16LE
- UTF-32BE
- UTF-32LE

If a UTF-* CCSID is explicitly specified, the JSON parser tolerates the corresponding Byte Order Mark (BOM) at the beginning of the data stream.

JSON data streams are parsed into a logical message tree under the `Data` element below the `JSON` parser root. The logical tree structure is shown in the "Example JSON message" on page 1130.

The `Data` element can be accessed and manipulated from ESQL as `JSON.Data`, from Java as `JSON/Data`, from PHP as `JSON->Data`, or from XPath as `$Body/Data`. The JSON parser issues an error if a bit stream is not formatted according to the JSON grammar.

The JSON serializer serializes message trees into a JSON format data stream. The CCSID can be defined by either the broker properties tree, or by transport headers in the message assembly. If no CCSID is defined, the serializer defaults to the queue manager default CCSID for all nodes except HTTP nodes, which default to UTF-8 encoding.

When the JSON serializer is invoked through an `ASBITSTREAM` function call, the CCSID is defined by the CCSID parameter. If no CCSID parameter is provided, or if the value is set to 0, the JSON serializer defaults to using UTF-8 encoding.

To process messages with the JSON parser, select `JSON` as the *Message Domain* on the relevant node in the message flow. The Mapping node and the XSLTransform node do not support the JSON domain.

The broker sets the HTTP Content-Type header to `application/json` when serializing a JSON message tree, unless an explicit value is set by the message flow.

JSON objects are modeled in the broker message tree as a sequence of *NameValue* elements. The parser builds the message tree in the order in which it encounters the members in the bit stream. The serializer writes the object members into the bit stream in the tree order.

JSON arrays are modeled in the broker message tree as a *Name* element with a JSON parser-specific type flag of JSON.Array and ordered children. The children can be any of the following types of array:

- An array containing simple values; for example:

  ```
  "array1" : [ "thing1", 1 ]
  ```

  The following message tree is produced:

  ```
  (0x01001000:Array): array1 = (
      (0x03000000:NameValue):Item = 'thing1' (CHARACTER)
      (0x03000000:NameValue):Item = 1 (INTEGER)
   )
  ```

  The JSON parser assigns the name `Item` to the *NameValue* elements.
- An array containing objects; for example:

  ```
  "array2" : [ {"a" : 1}, {"b" : 2} ]
  ```

  The following message tree is produced:

  ```
  (0x01001000:Array):array2 = (
    (0x01000000:Object):Item = (
      (0x03000000:NameValue):a = 1 (INTEGER)
    )
    (0x01000000:Object):Item = (
      (0x03000000:NameValue):b = 2 (INTEGER)
    )
  )
  ```
- A multidimensional array; for example:

  ```
  "array3" : [ [1.1], [2.1] ]
  ```

The following message tree is produced:

```
(0x01001000:Array):array3 = (
  (0x01001000:Array):Item = (
    (0x03000000:NameValue):Item = 1.1E+0 (FLOAT)
  )
  (0x01001000:Array):Item = (
    (0x03000000:NameValue):Item = 2.1E+0 (FLOAT)
  )
)
```

Example JSON message

The following example shows a simple JSON message:

```
{
    "name" : "John Doe",
    "age" : -1.0,
    "known" : false,
    "address" : { "street" : null,
                  "city" : "unknown" },
    "belongings" : ["item1", "item2", "item3"]
}
```

This JSON input produces the following broker logical message tree:

```
(0x01000000:Object):JSON            = ( ['json' : 0xd55fc8]
  (0x01000000:Object):Data = (
    (0x03000000:NameValue):name       = 'John Doe' (CHARACTER)
    (0x03000000:NameValue):age        = -1E+0 (FLOAT)
    (0x03000000:NameValue):known      = FALSE (BOOLEAN)
    (0x01000000:Object   ):address    = (
      (0x03000000:NameValue):street = NULL
      (0x03000000:NameValue):city   = 'unknown' (CHARACTER)
    )
    (0x01001000:Array    ):belongings = (
      (0x03000000:NameValue):Item = 'item1' (CHARACTER)
      (0x03000000:NameValue):Item = 'item2' (CHARACTER)
      (0x03000000:NameValue):Item = 'item3' (CHARACTER)
    )
  )
)
```

Related concepts:

"Parsers" on page 1072
A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from the internal message tree representation.

"JSON message details" on page 1135
A JSON message consists of name-value pairs (objects), and ordered collections of values (arrays). Objects, arrays, or both structures can be nested.

"Message tree mapping in the JSON domain" on page 1138
When reading a JSON message, the parser builds a message tree from the input bit stream by mapping JSON values to corresponding message tree element types. When serializing a message tree into the output bit stream, tree element types are mapped to JSON value types.

"Manipulating messages in the JSON domain" on page 2617
You can manipulate messages that belong to the JSON domain, which are parsed by the JSON parser.

"Which body parser should you use?" on page 1077
The characteristics of the messages that your applications exchange indicate which body parser you must use.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

Related tasks:

"Changing the parser used in a message flow" on page 1485
If you need to change the parser that is used in a message flow, use the ResetContentDescriptor node to request that the message is reparsed by a different parser.

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Developing Java" on page 2628
When you use the JavaCompute node, you customize it to determine the exact processing that it provides.

"Manipulating message body content" on page 2418
The message body is always the last child of root, and is identified by its parser name, for example XML or MRM. You can refer to, modify, and create message body data.

"Controlling the functional level of WebSphere Message Broker" on page 51
You can control the functional level of your broker to enable new functionality added in WebSphere Message Broker fix packs.

Related reference:

"Available parsers" on page 6689
A parser is called by the broker only when that parser is required. The parser that is called depends upon the parser that has been specified.

"IBM supplied messages that you can import" on page 6367
You can import IBM supplied messages to create a new message definition file.

"Message set properties" on page 5371
Message sets have properties that you can set to define their characteristics and the way in which they are processed.

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

JSONP support in the JSON domain:

JSONP (JavaScript Object Notation with Padding) is an extension of the JavaScript Object Notation (JSON) format.

WebSphere Message Broker provides support for JSONP services. A JSONP service, or Remote JSON Service, is a Web service that returns JSON data padded with a user-defined JavaScript function call. The JSONP response message can be interpreted as an executable script, so this functionality can be used to create cross-domain function calls.

For example:

```
http://brokerhost:7080/flowUrlPathSuffix?jsonp=scriptFn
```

This URL includes a query string, where:

- **jsonp** tells the JSONP service that any response from the URL must be returned as a JSONP message
- **scriptFn** is the name of a client-side executable function

Responses to the URL would therefore be in the JSONP format:

```
scriptFn(response)
```

The JSON message tree provides a top level `Padding` element, into which the JSON parser places the name of the client-side JSONP function. Similarly, the JSON serializer pads a JSON message if the top-level element `Padding` is present in the tree.

For more information about JSON, see "JSON parser and domain" on page 1128.

For information about how to use WebSphere Message Broker to provide a JSONP service, see "Providing a JSONP service" on page 1133.

For information about how to use WebSphere Message Broker to consume a JSONP service response, see "Consuming a JSONP service response" on page 1134.

Example JSONP message

The following example shows a simple JSONP message:

```
scriptFn (
    {
        "name" : "John Doe",
        "age" : -1.0,
        "known" : false,
        "address" : { "street" : null,
                      "city" : "unknown" },
        "belongings" : ["item1", "item2", "item3"]
    }
)
```

This JSONP input produces the following broker logical message tree:

```
  (0x01000000:Object):JSON            = ( ['json' : 0xd55fc8]
    (0x03000000:NameValue):Padding = 'scriptFn' (CHARACTER)
    (0x01000000:Object   ):Data    = (
      (0x03000000:NameValue):name       = 'John Doe' (CHARACTER)
      (0x03000000:NameValue):age        = -1E+0 (FLOAT)
      (0x03000000:NameValue):known      = FALSE (BOOLEAN)
      (0x01000000:Object   ):address    = (
        (0x03000000:NameValue):street = NULL
        (0x03000000:NameValue):city   = 'unknown' (CHARACTER)
      )
      (0x01001000:Array    ):belongings = (
        (0x03000000:NameValue):Item = 'item1' (CHARACTER)
        (0x03000000:NameValue):Item = 'item2' (CHARACTER)
        (0x03000000:NameValue):Item = 'item3' (CHARACTER)
      )
    )
  )
```

Related concepts:

"JSON parser and domain" on page 1128
JSON (JavaScript Object Notation) is a simple data-interchange format based on a subset of the JavaScript programming language.

"Manipulating messages in the JSON domain" on page 2617
You can manipulate messages that belong to the JSON domain, which are parsed by the JSON parser.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"What is a Web service?" on page 1603
A Web service is defined by the World Wide Web Consortium (W3C) as a software system designed to support interoperable machine-to-machine interaction over a network.

Related tasks:

"Providing a JSONP service"
Configure your WebSphere Message Broker message flow to provide a JSONP service response.

"Consuming a JSONP service response" on page 1134
When a message flow is configured to use the JSON domain, the JSON parser automatically detects JSONP messages. The JSON parser puts the JSON padding in the top level `Padding` element, and the JSON data under the `Data` element.

"Manipulating message body content" on page 2418
The message body is always the last child of root, and is identified by its parser name, for example XML or MRM. You can refer to, modify, and create message body data.

Providing a JSONP service:

Configure your WebSphere Message Broker message flow to provide a JSONP service response.

Before you begin

Before you start:

Before completing this task, read the following overview topics about JSON:
- "JSON parser and domain" on page 1128
- "JSONP support in the JSON domain" on page 1131

About this task

You can use ESQL, PHP, or Java to configure your message flow to provide a JSONP response.

The code examples in this task assume that the client application provides a JavaScript call in the following format:

```
<script type="text/javascript"
 src="http://brokerhost:7080/flowUrlPathSuffix?jsonp=scriptFn">
</script>
```

Procedure

1. On the **Advanced** tab of your HTTPInput node, select `Parse Query String`. This option enables you to access the JSONP script prefix that is included in the incoming URL, for example `scriptFn`, from the local environment tree.
2. Insert the following code, as appropriate:
   - If your message flow uses a Compute node:

     ```
     SET OutputRoot.JSON.Padding = InputLocalEnvironment.HTTP.Input.QueryString.jsonp;
     SET OutputRoot.JSON.Data.objectName = 'thing1';
     ```

   - If your message flow uses a PHPCompute node:

```
$output_assembly->JSON->Padding =
     $input_assembly[MB_LOCAL_ENVIRONMENT]->HTTP->input->QueryString->jsonp;
$output_assembly->JSON->Data->ObjectName = "thing1";
```

- If your message flow uses a JavaCompute node:

```
MbMessage outMessage = new MbMessage();
MbElement outRoot = outMessage.getRootElement();
MbElement outParser = outRoot.createElementAsLastChild(MbJSON.PARSER_NAME);
String paddingString =
    assembly.getLocalEnvironment().getRootElement().getFirstElementByPath("HTTP/Input/QueryStr
MbElement padding = outParser.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "Padding", pa
MbElement data = outParser.createElementAsLastChild(MbElement.TYPE_NAME, "Data", null);
data.createElementAsLastChild(MbElement.TYPE_NAME_VALUE,"objectName","thing1");
```

This code generates the following bit stream, sent as the HTTP reply:

```
scriptFn( {"objectName":"thing1"} )
```

This bit stream causes the JavaScript function `scriptFn` to be called with the JSON object as a parameter.

Related concepts:

"JSON parser and domain" on page 1128
JSON (JavaScript Object Notation) is a simple data-interchange format based on a subset of the JavaScript programming language.

"Manipulating messages in the JSON domain" on page 2617
You can manipulate messages that belong to the JSON domain, which are parsed by the JSON parser.

"What is a Web service?" on page 1603
A Web service is defined by the World Wide Web Consortium (W3C) as a software system designed to support interoperable machine-to-machine interaction over a network.

Related tasks:

"Consuming a JSONP service response"
When a message flow is configured to use the JSON domain, the JSON parser automatically detects JSONP messages. The JSON parser puts the JSON padding in the top level `Padding` element, and the JSON data under the `Data` element.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"PHPCompute node" on page 4639
Use the PHPCompute node to route and transform an incoming message, using the PHP scripting language.

"JavaCompute node" on page 4514
Use the JavaCompute node to work with messages by using the Java language.

Consuming a JSONP service response:

When a message flow is configured to use the JSON domain, the JSON parser automatically detects JSONP messages. The JSON parser puts the JSON padding in the top level `Padding` element, and the JSON data under the `Data` element.

Before you begin

Before you start:

Before completing this task, read the following overview topics about JSON:
- "JSON parser and domain" on page 1128
- "JSONP support in the JSON domain" on page 1131

About this task

You can process both JSON and JSONP messages in a single message flow, because the parser puts the JSON data under the `Data` element in the message tree. If JSONP padding is detected, the name of the client-side script is placed in the top level `Padding` element.

Follow these steps to test for the presence of padding:

Procedure

1. Create a message flow with an HTTPInput node, an HTTPReply node, and your choice of a Compute, PHPCompute, or JavaCompute node.
2. On the **Input Message Parsing** tab of your HTTPInput node, set the `Message domain` property to `JSON : For JavaScript Object Notation messages`.
3. Insert the following code, as appropriate:
   - If your message flow uses a Compute node:

     ```
     DECLARE PaddingRef REFERENCE TO InputRoot.JSON.Padding
     IF LASTMOVE(PaddingRef) THEN
        -- JSON Padding is present
     ELSE
         -- No JSON Padding present
     END IF;
     ```

   - If your message flow uses a PHPCompute node:

     ```
     if ($input_assembly->JSON->Padding != null)
        // JSON Padding is present
     else
         // No JSON Padding
     ```

   - If your message flow uses a JavaCompute node:

     ```
     if (message.getRootElement().getFirstElementByPath("JSON/Padding") != null){
         //JSON Padding is present
     }
     else{
         //No JSON Padding
     }
     ```

Related concepts:

"JSON parser and domain" on page 1128
JSON (JavaScript Object Notation) is a simple data-interchange format based on a subset of the JavaScript programming language.

"Manipulating messages in the JSON domain" on page 2617
You can manipulate messages that belong to the JSON domain, which are parsed by the JSON parser.

"What is a Web service?" on page 1603
A Web service is defined by the World Wide Web Consortium (W3C) as a software system designed to support interoperable machine-to-machine interaction over a network.

Related tasks:

"Providing a JSONP service" on page 1133
Configure your WebSphere Message Broker message flow to provide a JSONP service response.

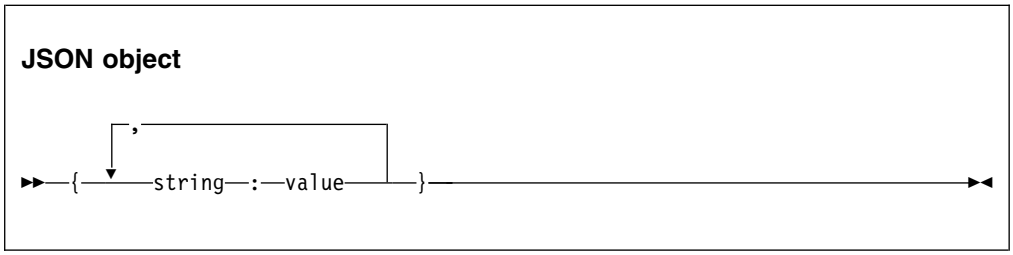
JSON message details:

A JSON message consists of name-value pairs (objects), and ordered collections of values (arrays). Objects, arrays, or both structures can be nested.

For more detailed information about JSON, see the JavaScript Object Notation (JSON) web site.

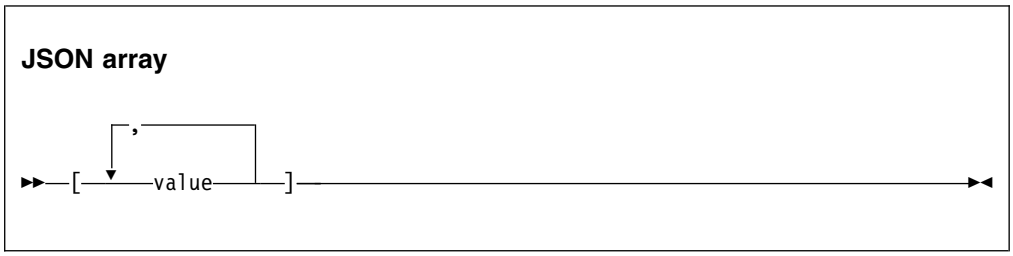
JSON object

In a JSON message, an object is an unordered set of comma-separated name-value pairs that begins with a left brace ({) and ends with a right brace (}). Each name is followed by a colon (:).



JSON array

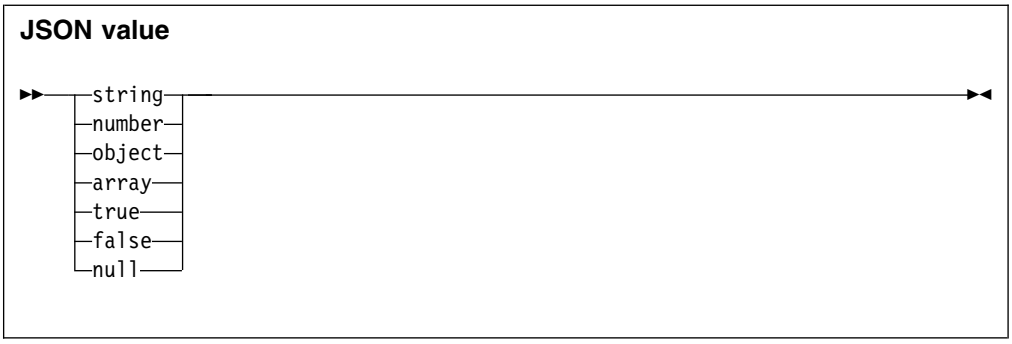
A JSON array is an ordered collection of comma-separated values that begins with left bracket ([) and ends with right bracket (]).



JSON value

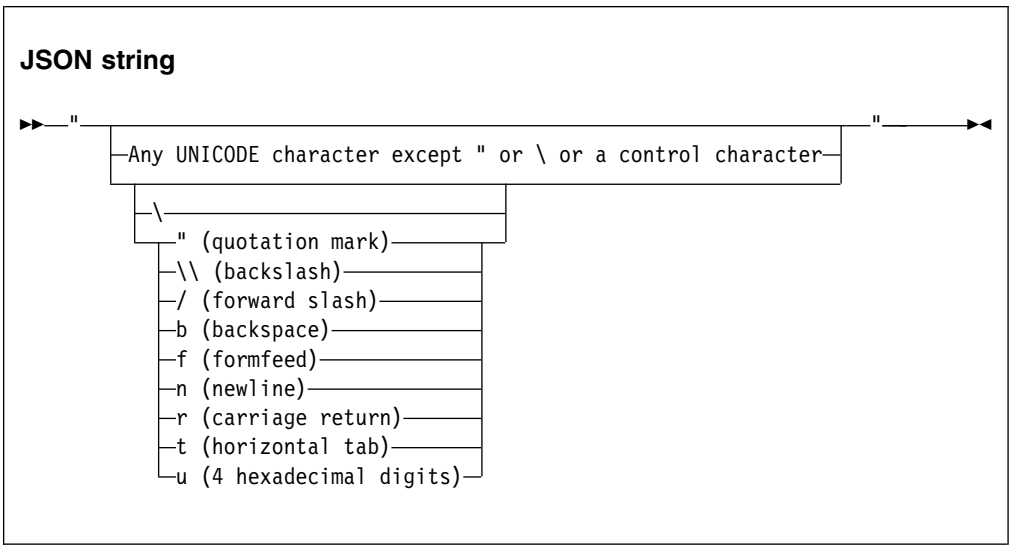
A JSON value can be any of the following structures, any of which can be nested:

- A string in double quotation marks
- A number
- Boolean
- Null
- An object
- An array



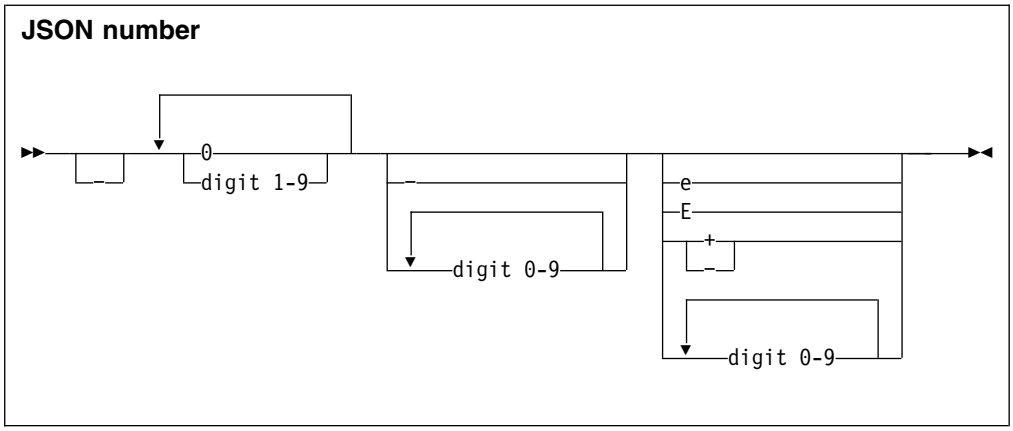
JSON string

A JSON string is very much like a C or Java string. A string is a collection of zero or more Unicode characters, wrapped in double quotation marks, using backslash escapes. A character is represented as a single character string.



JSON number

A JSON number is the same as a C or Java number, except that the octal and hexadecimal formats are not used.



Whitespace can be inserted between any pair of tokens.

Related concepts:

"JSON parser and domain" on page 1128
JSON (JavaScript Object Notation) is a simple data-interchange format based on a subset of the JavaScript programming language.

"Manipulating messages in the JSON domain" on page 2617
You can manipulate messages that belong to the JSON domain, which are parsed by the JSON parser.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Manipulating message body content" on page 2418
The message body is always the last child of root, and is identified by its parser name, for example XML or MRM. You can refer to, modify, and create message body data.

Message tree mapping in the JSON domain:

When reading a JSON message, the parser builds a message tree from the input bit stream by mapping JSON values to corresponding message tree element types. When serializing a message tree into the output bit stream, tree element types are mapped to JSON value types.

The following tables show how JSON message values are mapped to message tree element types:

- "JSON bit stream to message tree value mapping"
- "Message tree to JSON message value mapping" on page 1139

JSON bit stream to message tree value mapping

The JSON parser maps JSON values to message tree element types according to the rules in the following table:

JSON value present in bit stream	Parsed as
String	CHARACTER

JSON value present in bit stream	Parsed as
JSON Number value presented as: • A minus sign prefix if the value is negative • At least one digit, which can be zero • No dot fractional part • No exponent	INTEGER
JSON Number value presented as: • A minus sign prefix if the number is negative • At least one digit in the integer part, which can be zero • Either or both of: • A dot and at least one digit in the fractional part, which can be zero • An exponent using uppercase 'E' or lowercase 'e', an optional plus or minus sign, and at least one digit	FLOAT
Boolean	BOOLEAN
Null	NULL

Message tree to JSON message value mapping

The JSON serializer maps message tree elements to JSON value types according to the rules in the following table:

Message tree Element type	JSON Domain serializes as	
	JSON type	Format
BIT	String	Any number of 0 and 1s
BLOB	String	Even number of hexadecimal digits
CHARACTER	String	Char data with JSON string escaping for any double quotation mark (") or backslash (\) character within the CHAR
DATE	String	The standard ESQL string representation, 'yyyy-mm-dd'
TIME, GMTTIME	String	The standard ESQL string representation, 'hh:mm:ss.ffffff'
INTEGER	Number	• A minus sign prefix if the number is negative • At least one digit, which can be zero • No dot fractional • No exponent
FLOAT	Number	• A minus sign prefix if the number is negative • At least one digit in the integer part, which can be zero • At least one digit in the fractional part, which can be zero • An exponent part using uppercase 'E', a plus or minus sign, and at least one digit
DECIMAL	Number	• A minus sign prefix if the number is negative • At least one digit in the integer part, which can be zero • At least one digit in the fractional part, which can be zero Decimal literals ‘NAN', ‘INF', and so on, are not supported when serializing to JSON.
BOOLEAN	Boolean	true or false The serializer only serializes Boolean logical tree elements with true or false values; unknown is not supported

Message tree Element type	JSON Domain serializes as	
NULL	Null	Null

ROW	Object	Note: Assigning a ROW directly to a JSON Domain tree does not produce JSON arrays.
		`DECLARE myRow ROW;`
		`SET myRow.rowData[1].fieldOne = 'Row1Field1';`
		`SET myRow.rowData[1].fieldTwo = 'Row1Field2';`
		`SET myRow.rowData[2].fieldOne = 'Row2Field1';`
		`SET myRow.rowData[2].fieldTwo = 'Row2Field2';`
		`SET OutputRoot.JSON.Data.aRow = myRow;`
		Produces the following JSON bit stream
		`"aRow":{`
		`  "rowData": {"fieldOne":"Row1Field1","fieldTwo":"Row1Field2"},"`
		`  "rowData": {"fieldOne":"Row2Field1","fieldTwo":"Row2Field2"}`
		`}`

ROW	Array	To produce a JSON array from a ROW type the JSON.Array field would also be set.
		`SET OutputRoot.JSON.Data.aRow = myRow;`
		`SET OutputRoot.JSON.Data.aRow TYPE = JSON.Array;`
		Produces the following JSON bit stream
		`"aRow":[`
		`  {"fieldOne":"Row1Field1","fieldTwo":"Row1Field2"},`
		`  {"fieldOne":"Row2Field1","fieldTwo":"Row2Field2"}`
		`]`

Related concepts:

"JSON parser and domain" on page 1128
JSON (JavaScript Object Notation) is a simple data-interchange format based on a subset of the JavaScript programming language.

"Manipulating messages in the JSON domain" on page 2617
You can manipulate messages that belong to the JSON domain, which are parsed by the JSON parser.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Manipulating message body content" on page 2418
The message body is always the last child of root, and is identified by its parser name, for example XML or MRM. You can refer to, modify, and create message body data.

Handling null values:

A business message might contain fields that are either empty or have a specific out-of-range value. In these cases, the application that receives the message is expected to treat the field as if it did not have a value. The logical message tree supports this concept by enabling the value of any element to be set to NULL.

Ways to represent a null value: In an XML document, the usual way to represent a null value is to leave the element or attribute empty.

For example: `<price></price>` or `<element price=""/>`

The xsi:nil attribute provides a way to make this more explicit:
`price=<xsi:nil="true"/>`

Some business messages use a special value to represent a null value:
`<price>-999</price>`

This style of null representation is supported only by the MRM parser.

ESQL support for null values: Using ESQL, you can set the value of a message tree element to NULL:

```
SET OutputRoot.XMLNSC.myField VALUE = NULL;
```

Note that this is different from `SET OutputRoot.XMLNSC.myField = NULL;` which would cause myField to be deleted from the message tree.

The same effect can be achieved using Java or a Mapping node.

XMLNSC parser support for null values: Typically, the XML parsers (XMLNSC, XMLNS, and XML) do not create null values in the message tree; an empty element or an empty attribute value merely produces an empty string value in the message tree.

If validation is enabled, the XMLNSC parser detects and processes any xsi:nil attributes in the input document. If the xsi:nil attribute is set to 'true', and the element is nullable, the attribute's parent element in the message tree is given a null value.

For more information about XML parser support for empty elements and null values, see “XMLNSC empty elements and null values” on page 1092 and “XMLNS empty elements and null values” on page 1106.

MRM parser support for null values: **XML physical format**

When parsing, the MRM XML parser can detect and process xsi:nil attributes in the input XML document. If the xsi:nil attribute is set to 'true', and the element is nullable, the attribute's parent element in the message tree is given a null value.

For information about enabling xsi:nil support in the MRM parser, see “XML Null handling options” on page 6258.

The following topics provide more information about handling null values in the MRM parser:
- “MRM Custom wire format: NULL handling” on page 1216
- “MRM XML physical format: NULL handling” on page 1249
- “MRM TDS format: NULL handling” on page 1240

All physical formats

The MRM parser can detect a null value that is represented by an out-of-range value. The null value must be specified in the physical format of the message set.

While parsing, the MRM parser checks the null value for each element in the message. If the value in the bit stream matches the null value in the message set, the MRM parser sets the value in the message tree to NULL.

The same check is performed when converting a message tree to a bit stream. If the value in the message tree is NULL, the MRM parser outputs the null value from the message set.

JSON parser support for null values: The JSON format supports NULL as a JSON value type.

When a JSON message includes an object with a null value, the JSON parser sets the value in the message tree to NULL.

When serializing an element in the message tree with a null value, the JSON bit stream is constructed as a JSON object with a value of null.

Related concepts:

"Logical tree structure" on page 1042
The logical tree structure is the internal (broker) representation of a message. It is also known as the message assembly.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"MRM Custom wire format: NULL handling" on page 1216
CWF supports the handling of explicit NULL values within messages, if the logical nillable property of the element is set.

"MRM XML physical format: NULL handling" on page 1249
The purpose of null handling is to specify how messages deal with null values; that is, the absence of a meaningful value for an element.

"MRM TDS format: NULL handling" on page 1240
NULL handling dictates the way in which the MRM parser for TDS messages handles elements that have been set to Null.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Manipulating message body content" on page 2418
The message body is always the last child of root, and is identified by its parser name, for example XML or MRM. You can refer to, modify, and create message body data.

"Querying null values in a message in the MRM domain" on page 2597
You can use an ESQL statement to compare an element to NULL.

"Setting null values in a message in the MRM domain" on page 2599
You can use implicit or explicit null processing to set the value of an element to NULL in an output message.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message

models.

Properties

You can view and change properties that define broker characteristics, and those properties of associated resources such as message flows.

Broker properties

Each broker has a set of properties that define certain characteristics that you can control:

- You can set some broker properties only at the time that you create the broker by using the **mqsicreatebroker** command; for example, its associated queue manager.
- You can change some of the broker properties by using the **mqsichangebroker** command; for example, timeout values for processing configuration requests.
- You can view all broker properties by using the **mqsireportbroker** command.
- You can access some broker properties from ESQL and Java programs that you run in message flow nodes. For details of these options, see "Broker properties" on page 1144.

Node properties

Each built-in (supplied) node has at least one property that you can configure; some properties are mandatory, others are optional. In addition to setting these properties for each node when you add it to a message flow, you can also use the following configuration techniques:

- You can promote properties, so that you can set them at the message flow level. For information about how and why you might want to use this technique, see "Promoted properties" on page 1145.
- You can configure properties when you add the message flow to a BAR file for deployment. In each node description, the subset of node properties that you can use in this way are identified. For more information, see "Editing configurable properties" on page 3227.

Node properties are discussed in "Message flow nodes" on page 1024.

User-defined properties

You can create your own properties when you create a message flow, and access those properties from ESQL and Java programs in your message flow nodes. For further details, see "User-defined properties" on page 1147

Configurable service properties

You can modify some of the properties of the configurable services that are supplied to enhance your message flow processing options. For example, configurable services are supplied to communicate with enterprise information systems such as SAP.

You can also create your own configurable services, as variants of the supplied services, or to provide additional options.

For further information, see "Configurable services" on page 1296.

Related concepts:

"Message flow nodes" on page 1024
A message flow node is a processing step in a message flow. It can be a built-in node, a user-defined node, or a subflow node.

"Broker properties"
For each broker, WebSphere Message Broker maintains a set of properties. You can access some of these properties from your ESQL programs. A subset of the properties is also accessible from Java programs. It can be useful, at run time, to have real-time access to details of a specific node, flow, or broker.

"Promoted properties" on page 1145
A promoted property is a message flow node property that has been promoted to the level of the message flow in which it is included.

"User-defined properties" on page 1147
A user-defined property (UDP) is a property that is defined when you construct a message flow by using the Message Flow editor. This property can be used by the ESQL or Java program inside message flow nodes, such as a Compute node.

"Configurable services" on page 1296
Configurable services are typically runtime properties. You can use them to define properties that are related to external services on which the broker relies; for example, an SMTP server or a JMS provider.

Related reference:

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsireportbroker** command" on page 3919
Use the **mqsireportbroker** command to display broker registry entries.

Broker properties:

For each broker, WebSphere Message Broker maintains a set of properties. You can access some of these properties from your ESQL programs. A subset of the properties is also accessible from Java programs. It can be useful, at run time, to have real-time access to details of a specific node, flow, or broker.

Broker properties are divided into four categories:

- Properties that relate to a specific node
- Properties that relate to nodes in general
- Properties that relate to a message flow
- Properties that relate to the execution group

For a description of the broker, flow, and node properties that are accessible from ESQL and Java, see "Broker properties that are accessible from ESQL and Java" on page 5302.

Broker properties have the following characteristics.

- They are grouped by broker, execution group, flow, and node.
- They are case-sensitive. Their names always start with an uppercase letter.
- They return NULL if they do not contain a value.

All nodes for which user programs can edit ESQL support access to broker properties. These nodes are:

- Compute nodes
- Database nodes
- DatabaseInput nodes

- Filter nodes
- All derivatives of these nodes

User-defined properties can be queried, discovered, and set at run time to dynamically change the behavior of a message flow. You can use the Administration API for WebSphere Message Broker(CMP API) to manipulate these properties, which can be used by a systems monitoring tool to perform automated actions in response to situations that it detects in the monitored systems. For more information, see "User-defined properties" on page 1147.

A *complex property* is a property to which you can assign multiple values. Complex properties are displayed in a table in the Properties view, where you can add, edit, and delete values, and change the order of the values in the table. You cannot promote complex properties; therefore, they are not shown in the Promote properties dialog box. Nor can you configure complex properties; therefore, they are not supported in the Broker Archive editor. For an example of a complex property, see the `Query elements` property of the DatabaseRoute node.

For more information about editing the properties of a node, see "Configuring a message flow node" on page 1503.

Related concepts:

"User-defined properties in ESQL" on page 2376
Access user-defined properties (UDPs) as variables in your ESQL program by specifying the EXTERNAL keyword on a DECLARE statement. For example, the ESQL statement `DECLARE today EXTERNAL CHARACTER 'monday'` defines a user-defined property called today with an initial value monday.

"User-defined properties" on page 1147
A user-defined property (UDP) is a property that is defined when you construct a message flow by using the Message Flow editor. This property can be used by the ESQL or Java program inside message flow nodes, such as a Compute node.

Related tasks:

"Accessing broker properties from ESQL" on page 2625
You can access broker properties, at run time, from the ESQL modules in your message flow nodes.

"Accessing broker properties from the JavaCompute node" on page 2658
For each broker, WebSphere Message Broker maintains a set of properties. You can access some of these properties from your Java programs. It can be useful, during the run time of your code, to have real-time access to details of a specific node, flow, or broker.

Related reference:

"Broker properties that are accessible from ESQL and Java" on page 5302
You can access broker, message flow, and node properties from ESQL and Java.

"DatabaseRoute node" on page 4373
Use the DatabaseRoute node to route messages using information from a database in conjunction with XPath expressions.

Promoted properties:

A promoted property is a message flow node property that has been promoted to the level of the message flow in which it is included.

A message flow contains one or more message flow nodes. You can promote the properties of a message flow node to apply to the message flow to which it

belongs. In this case, any user of the message flow can override these promoted properties at the message flow level, without being aware of the message flow's internal structure.

You can promote compatible properties (that is, properties that represent comparable values) from more than one node to the same promoted property; you can then set a single property that affects multiple nodes.

For example, you might want to set the name of a data source as a property of the message flow, rather than a property of each individual node in the message flow that references that data source. You create a message flow that accesses a database called SALESDATA. However, while you are testing the message flow, you want to use a test database called TESTDATA. If you set the data source properties of each individual node in the message flow to refer to SALESDATA, you can promote the data source property for each node in the flow that refers to it, and update the property to have the value TESTDATA; this value overrides the data source properties on the nodes while you test the message flow (the promoted property always takes precedence over the settings for the properties in any relevant nodes).

A subset of message flow node properties is also configurable (that is, the properties can be updated at deployment time). You can promote configurable properties: if you do so, the promoted property (which can have a different name from the property or properties that it represents) is the one that is available to update at deployment time. Configurable properties are those associated with system resources; for example, queues and data sources. An administrator can set these properties at deployment time, without the need for a message flow developer.

You cannot promote a complex property, so it does not appear in the Promote properties dialog box. For more information about complex properties, see “Broker properties” on page 1144.

Related concepts:

“Message flows overview” on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

“Packaging and deployment overview” on page 3210
Deployment is the process of transferring data to an execution group on a broker so that it can take effect in the broker. Message flows and associated resources are packaged in broker archive (BAR) files for deployment.

“Broker properties” on page 1144
For each broker, WebSphere Message Broker maintains a set of properties. You can access some of these properties from your ESQL programs. A subset of the properties is also accessible from Java programs. It can be useful, at run time, to have real-time access to details of a specific node, flow, or broker.

Related tasks:

“Promoting a property” on page 1298
You can promote a node property to the message flow level to simplify the maintenance of the message flow and its nodes, and to provide common values for multiple nodes in the flow by converging promoted properties.

“Renaming a promoted property” on page 1302
If you have promoted a property from the node to the message flow level, it is initially assigned the same name that it has at the node level. You can rename the property to have a more meaningful name in the context of the message flow.

"Removing a promoted property" on page 1304
If you have promoted a property from the node to the message flow level, you can remove (delete) it if you no longer want to specify its value at the message flow level. The property reverts to the value that you specified at the node level. If you remove a promoted property that is a mandatory property, ensure that you have set a value at the node level. If you have not, you cannot successfully deploy a broker archive file that includes this message flow.

"Converging multiple properties" on page 1306
You can promote properties from several nodes in a message flow to define a single promoted property, which applies to all those nodes.

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

User-defined properties:

A user-defined property (UDP) is a property that is defined when you construct a message flow by using the Message Flow editor. This property can be used by the ESQL or Java program inside message flow nodes, such as a Compute node.

The advantage of UDPs is that their values can be changed by operational staff at deployment and run time. You do not need to change your application programs. For example, if you use the UDPs to hold data about your computer center, you can configure a message flow for a particular computer, task, or environment at deployment time, without having to change the code at the message node level.

When you launch the Message Flow editor to either create a message flow or modify an existing message flow, as well as deciding which nodes are required in the message flow, you also have the option (provided by the tab) of defining and giving initial values to some user-defined properties. Use the **User Defined Properties** tab at the bottom of the edit window. See "Message Flow editor" on page 6810 for more information.

As well as being defined using the Message flow editor, you must also define a UDP either by using a DECLARE statement with the EXTERNAL keyword in all ESQL programs that use it, or by calling the getUserDefinedAttribute method in all JavaCompute nodes that use it.

See the "DECLARE statement" on page 5117 for details of the DECLARE statement, and see "Accessing message flow user-defined properties from a JavaCompute node" on page 2659 for more information about how to use a UDP in a JavaCompute node.

Values that you give to a UDP when you define it in a message flow override the value of that variable in your ESQL program.

You can also modify the value of a UDP at deployment time by using the Broker Archive editor to edit the BAR file. This value overrides the value that was set when you defined the message flow.

The value of the UDP is set at the message flow level, and is the same for all eligible nodes that are contained in the flow. An *eligible node* is a node that supports UDPs and is within the scope of the declaration that declares the UDP to your application. For example, if you use the Message Flow editor to change the value of a user property called `timezone`, which is declared in a schema called

mySchema, in a message flow called *myFlow*, the UDP is available at run time to all the nodes in *myFlow* that support UDPs and that fall within *mySchema*.

Similarly, if you use the Message Flow editor to change the value of a user-defined property in a subflow, the newly edited property is available to all the nodes in the subflow that support UDPs, and that are within the scope of the declaration. The property is not available, for example, to nodes in the parent flow.

A UDP has global scope and is not specific to a particular subflow. If you reuse a subflow in a message flow, and those subflows have identical UDPs, you cannot set the UDPs to different values.

Controlling user-defined properties at run time

User-defined properties can be queried, discovered, and set at run time to dynamically change the behavior of a message flow. You can use the Administration API for WebSphere Message Broker (also known as the CMP API) or WebSphere Message Broker Explorer to manipulate these properties, which can be used by a systems monitoring tool to perform automated actions in response to situations that it detects in the monitored systems.

For example, a message flow contains a Route node, which is used to differentiate between the classes of customer that are defined in the message. The Route node has a user-defined property called `ProcessClasses`, which is set with an initial value of `All`. When `ProcessClasses` is set to `All`, the node routes messages from all classes of customer to its first terminal for immediate processing.

When certain conditions are detected (for example, the monitoring system detects that the request load is causing the service level agreement to fall below its target), the Route node must be set to pass requests from only "Gold" class customers for immediate processing, while other customer requests are sent to another output terminal, which queues them for later batch processing. Therefore, the monitoring application sets `ProcessClasses` to `Gold`, so that the Route node routes the less critical messages to the second terminal.

To make it easier to know what a user-defined property does, and what values it can have, adopt a suitable naming convention. For example, a user-defined property named `property01`, with an initial value of `valueA` is not as useful as a property named `RouteToAorB` with an initial value of `RouteA`.

For more information, see “Setting message flow user-defined properties at run time in a CMP application” on page 985 or “Setting user-defined properties dynamically at run time using the WebSphere Message Broker Explorer” on page 954.

Precedence of UDP value overriding

You can define a user-defined property in the following ways:
- In the ESQL code
- In the Message Flow editor
- Through a BAR file override
- By using the CMP API

A BAR file override takes precedence over changes in the Message Flow editor, and changes in the Message Flow editor take precedence over changes in the ESQL code.

The precedence of the values for user-defined properties is shown in the following sequence:

1. The user-defined property `ProcessClasses` is set to `All` in a message flow BAR file. After deployment of the BAR file, the value of `ProcessClasses` is `All`.
2. The same user-defined property (`ProcessClasses`) is set to `Gold` by using the CMP API to issue the call setUserDefinedProperty("ProcessClasses", "Gold"). After successful completion of this method, the value of `ProcessClasses` is `Gold`.
3. The broker is shut down and restarted. The value of `ProcessClasses` is still `Gold`.
4. The original flow BAR file is redeployed. After deployment, the value of `ProcessClasses` is `All`.

You can find more information about defining user-defined properties in the following related topics.

Related concepts:

"User-defined properties in ESQL" on page 2376
Access user-defined properties (UDPs) as variables in your ESQL program by specifying the EXTERNAL keyword on a DECLARE statement. For example, the ESQL statement `DECLARE today EXTERNAL CHARACTER 'monday'` defines a user-defined property called today with an initial value monday.

"ESQL variables" on page 2374
An ESQL variable is a data field that is used to help process a message.

Related tasks:

"Configuring a message flow at deployment time with user-defined properties" on page 2626
Use user-defined properties (UDPs) to configure message flows at deployment and run time, without modifying program code. You can give a UDP an initial value when you declare it in your program, or when you use the Message Flow editor to create or modify a message flow.

"Setting message flow user-defined properties at run time in a CMP application" on page 985
Use the CMP API to query, discover, and set message flow user-defined properties dynamically at run time. You can use the CMP API to set properties with a data type of character.

"Setting user-defined properties dynamically at run time using the WebSphere Message Broker Explorer" on page 954
Use the WebSphere Message Broker Explorer to view and set user-defined properties on a message flow dynamically at run time.

"Accessing message flow user-defined properties from a JavaCompute node" on page 2659
Customize a JavaCompute node to access properties that you have associated with the message flow in which the node is included.

Related reference:

"Broker Archive editor" on page 6794
Use the Broker Archive editor to create and manage broker archive (BAR) files.

"DECLARE statement" on page 5117
Use the DECLARE statement to define a variable, the data type of the variable and, optionally, its initial value.

"Message Flow editor" on page 6810
The Message Flow editor is the default editor provided by the Broker Application Development perspective for defining a graphical representation of a message flow in the WebSphere Message Broker Toolkit, and for setting properties for individual message flow nodes.

"ESQL variables" on page 5048
ESQL variables can be described as *external variables*, *normal variables*, or *shared variables*; their use is defined in the DECLARE statement.

Impact analysis: analyzing the effects of planned changes to your applications

When developing an application, you might want to change the names of some artifacts, or move them. Impact analysis reports the artifacts that are likely to be affected by a particular change.

You can analyze the effects of renaming the following artifacts:

- Files
  - MXSD files
  - Deployable WSDL files
  - Map files
  - Message flow files (including subflows)
- Global artifacts in message sets (MXSD)
  - Element definitions
  - Complex and simple type definitions
  - Model groups
  - Attribute groups
  - Attributes
  - Message definitions
- ESQL modules
- Schema-scope ESQL constants
- Maps, including submaps
- Schema-scope ESQL routines

You can analyze the effects of moving the following artifacts within the originating project:

- Schema-scope ESQL routines
- Schema-scope ESQL constants
- ESQL modules
- Map files
- Message flow files

Impact analysis lists two types of impact. *Primary impacts* are the immediate consequences if a change is performed. For example, if you analyze the impact of renaming a file, the primary impact is the renaming of that specific file. *Secondary impacts* are any other artifacts or contents of an artifact that are affected by the primary change. A single change can create many secondary impacts. If another artifact called `S2` references the name of the file that you want to rename, there is a secondary impact upon artifact `S2`. Impact analysis does not report third-level impacts; artifacts that depend on the validity of `S2` are not listed.

For example, when a global artifact in a message set, such as message or global element, is renamed, impact analysis generates a list of artifacts that are likely to be affected:

MXSDs

Flows

Deployable WSDLs

Maps

ESQL paths in the ESQL module

See “Impact analysis: reference” on page 4174 for details of artifacts and paths that are not reported under secondary analysis.

Note: Impact analysis does not change any resources.

Impact analysis results are shown either in the Impact Analysis dialog box, or in the Impact Analysis view in the WebSphere Message Broker Toolkit. For more information about how to view information about selected resources, mark changes as complete, and view previous results, see “Impact Analysis view” on page 6801.

Indexing

Indexing must be enabled to support impact analysis. The act of indexing determines what definitions and relationships exist in WebSphere Message Broker artifacts. These definitions and relationships are used to perform impact analysis. By default, indexing is disabled, for performance reasons.

Related tasks:

“Enabling and disabling indexing” on page 1454
Enable indexing to support impact analysis.

“Analyzing planned changes to message model objects” on page 2897
Use impact analysis to analyze the effect of renaming message model objects.

“Analyzing planned changes to ESQL objects” on page 2403
Use impact analysis to analyze the effect of renaming or moving ESQL objects, or moving or changing the broker schema of an ESQL file.

“Analyzing the impact of changes to message maps” on page 2234
Use impact analysis to analyze the effect of renaming or moving message maps.

“Analyzing planned changes to message flows” on page 1436
Use impact analysis to analyze the effect of renaming or moving message flows, including subflows.

“Analyzing planned changes to message set resources” on page 1165
Use impact analysis to analyze the effect of renaming message definition or deployable WSDL files.

Related reference:

“Impact analysis: reference” on page 4174
Some artifacts are excluded from secondary analysis.

Data conversion

Convert data that your message flows are transferring between different environments by using WebSphere MQ or WebSphere Message Broker facilities.

Data conversion is the process by which data is transformed from the format recognized by one operating system into the format recognized by a second operating system with different characteristics such as numeric order.

If you are using a network of systems that use different methods for storing numeric values, or you need to communicate between users who view data in different code pages, you must consider how to implement data conversion.

Code page conversions

Code page conversion might be required for one or more of the following reasons:

- ASCII versus EBCDIC
- Code pages that are specific to national language
- Code pages that are specific to operating systems

In WebSphere MQ, these factors are handled by the CCSID field in the MQMD header. For more information about the MQMD header, see "MQMD - Message descriptor" in the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online. For more information about code page support, see "Code page conversion", also in the *Application Programming Reference* section.

Encoding

Encoding (byte order) conversion might be required for one or both of the following reasons:

- Big endian versus little endian

  Endian is an attribute of data that describes whether it is stored in computer memory or transmitted with the most significant byte first (big endian) or last (little endian).

- Floating point number representations

In WebSphere MQ, these factors are handled by the Encoding field in the MQMD header. For more information about the MQMD header, see "MQMD - Message descriptor" in the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online. For more information about encoding, see "Machine encoding", also in the *Application Programming Reference* section.

If you are configuring a message flow to receive messages:

- Messages received across a WebSphere MQ protocol that uses WebSphere MQ headers, contain code page encoding characteristics in the MQMD header, and optionally in other WebSphere MQ headers.
- Messages received across protocols that do not use WebSphere MQ headers do not include these characteristics. Configure these characteristics by using properties on the nodes in your message flows. For example, set the Message coded character set ID and Message encoding properties on the FileInput node.

If you are configuring a message flow to send messages to other applications or systems:

- Messages sent across a WebSphere MQ protocol contain code page encoding characteristics in the MQMD header, and optionally in other WebSphere MQ headers.
- Messages sent across protocols that do not use WebSphere MQ headers must be modified to include these characteristics in the Properties folder in the logical message tree structure. The parser called by the output node uses these values to generate the correct bit stream.

When you use WebSphere Message Broker, you can use the data conversion facilities of WebSphere Message Broker, WebSphere MQ, or both.

WebSphere Message Broker facilities

You can model your messages in the MRM through the WebSphere Message Broker Toolkit. Predefined elements of the messages are converted according to their type and physical layer characteristics. For further details, see “Configuring physical properties” on page 2912. You can also use self-defining messages. You can then use the Compute, JavaCompute, or PHPCompute node to configure encoding and CCSIDs. You do not need WebSphere MQ data conversion exits.

- String data is converted according to the CCSID setting.
- Decimal integer and float extended decimal types are converted according to the CCSID setting.
- Decimal integer and float (other physical data types) are converted according to the Encoding setting.
- Binary and Boolean data is not converted.

WebSphere Message Broker can also convert the WebSphere MQ headers for which parsers are provided.

When you use WebSphere Message Broker facilities, the whole message is not converted to the specified encoding and CCSID: you can specify a different encoding, or CCSID, or both, in each header to perform a different conversion for the following part of the message. The encoding and CCSID in the last header defines the values for the message body.

WebSphere MQ facilities

Headers and message body are converted according to the values set in the appropriate MQMD fields, and other header format names. You might need to set up data conversion exits to convert the body of your messages.

When you use WebSphere MQ facilities, the whole message is converted to the specified encoding and CCSID, according to the setting of the format in the WebSphere MQ header.

For more detail about data conversion using WebSphere MQ facilities, see "Data conversion" in the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online.

Related concepts:

“Message flows overview” on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

“Parsers” on page 1072
A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from the internal message tree representation.

Related tasks:

“Configuring message flows for data conversion” on page 1293
If you exchange messages between applications that run on systems that are incompatible in some way, you can configure your system to provide data conversion as the message passes through the broker.

“Converting code page and message encoding” on page 2476
You can use ESQL within a Compute node to convert data for code page and message encoding.

“Configuring physical properties” on page 2912
Working with the physical properties of message model objects.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"Supported code pages" on page 4176
Application messages must conform to supported code pages.

Related information:

WebSphere MQ Version 7 Information Center online

Message modeling

Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Applications typically use a combination of messages, including those that are defined by the following structures or standards:

- C and COBOL data structures
- Industry standards such as SWIFT or EDIFACT
- XML DTD or Schema

You can model a wide variety of message formats so that they can be understood by WebSphere Message Broker message flows.

When the message format is known, the broker can parse an incoming message bit stream and convert it into a logical message tree for manipulation by a message flow. After the message has been processed by the message flow, the broker converts the message tree back into a message bit stream.

The following topics together give an overview of Message modeling:

- "Message modeling concepts" on page 1155
- "Why model messages?" on page 1158
- "Message domains and parsers" on page 1159
- "The message model" on page 1160
- "Physical formats in the MRM domain" on page 1211
- "Ways to create message definitions" on page 1253
- "Generate model representations" on page 1270

You can import either of the following samples to explore message set projects to understand how the sample's messages are modeled in different formats.

- Video Rental
- Comma Separated Value (CSV)

The following samples also have message sets supplied:

- EDIFACT
- FIX
- SWIFT
- X12

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online

information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

If you want to analyze the effect of changing a message model object, see "Analyzing planned changes to message model objects" on page 2897.

Related concepts:

"Message modeling concepts"
Message modeling is a way of predefining the message formats that are used by your applications.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Parsers" on page 1072
A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from the internal message tree representation.

"Impact analysis: analyzing the effects of planned changes to your applications" on page 1150
When developing an application, you might want to change the names of some artifacts, or move them. Impact analysis reports the artifacts that are likely to be affected by a particular change.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Analyzing planned changes to message model objects" on page 2897
Use impact analysis to analyze the effect of renaming message model objects.

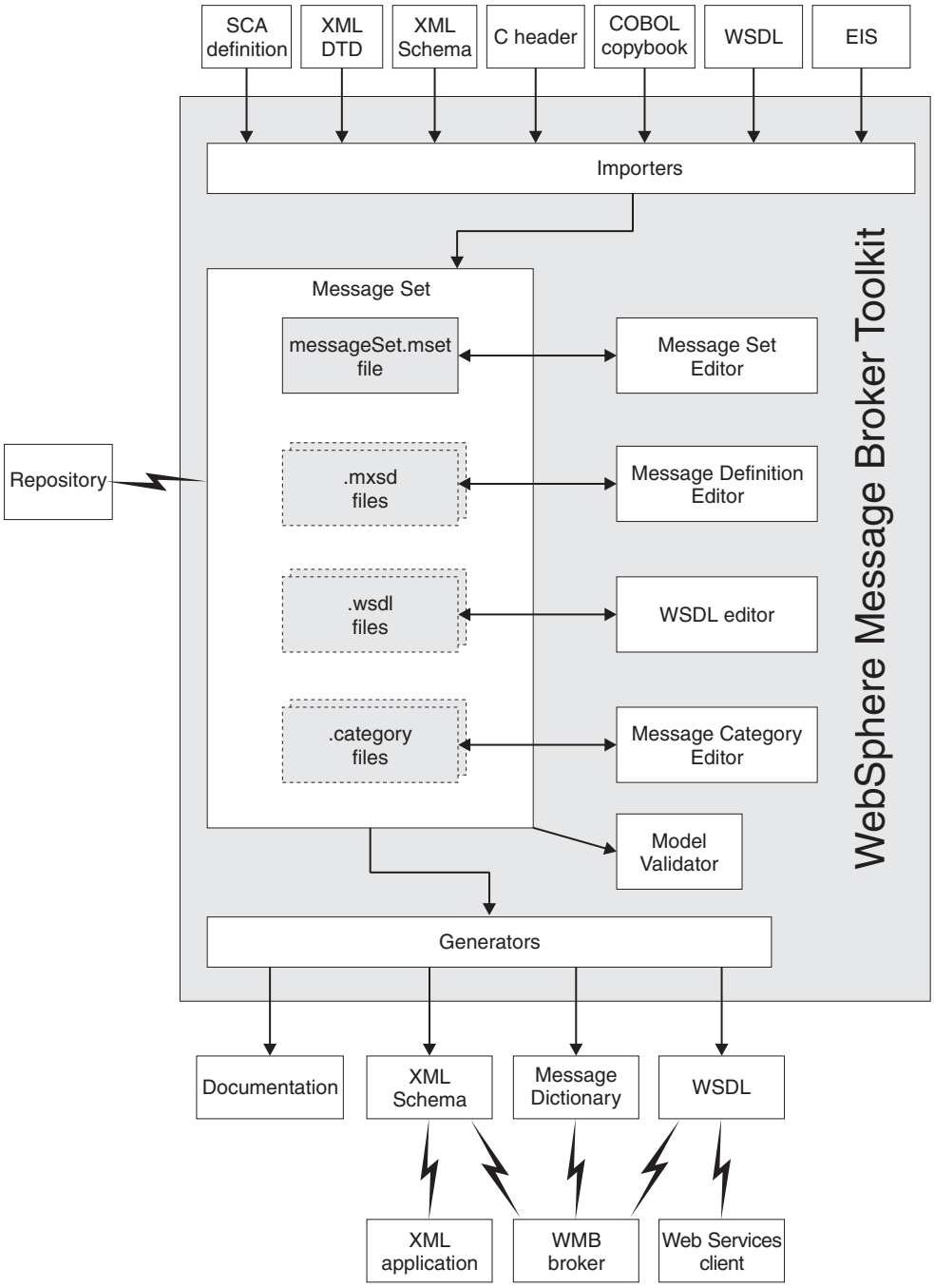
Message modeling concepts

Message modeling is a way of predefining the message formats that are used by your applications.

When you have created your message models, include them in your broker archive (BAR) file with the message flows that use those models. Deploy the BAR file to the broker, which uses your message models to automatically parse and write your message formats.

When you model messages, you must understand the following concepts:

- *Message set projects*
- *Message sets*
- *Message definition files*
- *Web Services Description Language (WSDL) files*
- *SCA definition files*
- *Message categories*
- *Model importers*
- *Model editors*
- *Model generators*
- *Model validator*
- *Domains and parsers*



A *message set project* is a specialized project (container) in which you create and maintain all the resources that are associated with exactly one *message set*.

A message set is a logical grouping of your messages and the objects that comprise them (elements, types, groups). A message set contains the following files:

- Exactly one message set file
- Zero or more message definition files
- Zero or more WSDL files
- Zero or more message category files

The message set file provides message model information that is common to all the messages in the message set. You can create this information using the *message set editor*.

When you have created a message set, you typically import application message formats described by XML DTD, XML Schema, WSDL files, C structures, COBOL structures, or EIS systems, creating and populating *message definition files*. You can then edit the logical structure of your messages, and create and edit physical formats that describe the precise appearance of your message bit stream during transmission, using the *message definition editor*. Alternatively, you can create an empty message definition file and create your messages using just the editor.

When your message definition files are complete, you can then generate the message set in a form that can be used by a broker, parser, or application. This might be in one of the following forms:

- A message dictionary for deployment to a broker
- An XML Schema for use by an application to validate XML messages, or for deployment to a broker
- Web Services Description Language (WSDL) for a Web services client, or for deployment to a broker
- Documentation to give to programmers or business analysts

Messages can be optionally grouped into *message categories* for convenience . You can add messages to message categories using the *message category editor*.

Each time you save a message set file, message definition file, or message category file, the content is *validated* to ensure that the message model that you are creating follows certain rules. There are rules for both the logical structure and the physical formats. This 'model validation' ensures the integrity of your model, but does not necessarily prevent you from saving a message model file that is not valid.

WebSphere Message Broker supplies a range of parsers to parse and write message formats. Each parser is suited to a particular class of messages (for example, fixed-length binary, delimited text, or XML) known as a *message domain*. When you create a message set, you specify which domains the message set supports. This determines which parsers can be used when you parse and write messages that are defined within that message set.

Related concepts:

"Why model messages?" on page 1158
WebSphere Message Broker supplies a range of parsers to parse and write message formats. Some message formats are self-defining and can be parsed without reference to a model. Most message formats, however, are not self-defining, and the parser must have access to a predefined model that describes the message, if it is to parse it correctly.

"Message set projects" on page 1161
A *message set project* is a specialized container in which you create and maintain all the resources associated with one message set.

"Message sets overview" on page 1162
A message set is a container for grouping messages and associated message resources (elements, types, groups).

"Message definition files" on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

"Message categories" on page 1200
Message category files have the suffix `.category`. These files are optional. You can have many message category files in a message set.

"Message model integrity" on page 1210
When you create your message model, it is important that it is internally consistent.

"Message domains and parsers" on page 1159
WebSphere Message Broker supplies a range of *parsers* to parse and write message formats.

Related reference:

"Message set editor" on page 6819
The Message set editor is the default editor provided by the Broker Application Development perspective for editing message set (`messageSet.mset`) files.

"Message Definition editor" on page 6804
The Message Definition editor is the default editor provided by the Broker Application Development perspective for editing message definition (`.mxsd`) files.

"Message Category editor" on page 6802
The Message Category editor is the default editor provided by the Broker Application Development perspective for working with message category (`.category`) files in a message set.

Why model messages?

WebSphere Message Broker supplies a range of parsers to parse and write message formats. Some message formats are self-defining and can be parsed without reference to a model. Most message formats, however, are not self-defining, and the parser must have access to a predefined model that describes the message, if it is to parse it correctly.

An example of a self-defining message format is XML. In XML the message itself contains metadata as well as data values, enabling an XML parser to understand an XML message even if no model is available.

Examples of messages that do not have a self-defining message format are binary messages that originate from a COBOL program, and from SWIFT formatted text messages. Neither contain sufficient metadata to enable a parser to understand the messages.

Even if your messages are self-defining and do not require modeling, the following advantages of modeling them might be useful:

- Runtime validation of messages. Without a model, a parser cannot check whether input and output messages have the correct structure and data values.
- Enhanced parsing of XML messages. Although XML is self-defining, without a model, all data values are treated as strings. If a model is used, the parser knows the data type of data values, and can cast the data accordingly.
- Improved productivity when writing ESQL. When you are creating ESQL programs for WebSphere Message Broker message flows, the ESQL editor can use message models to provide code completion assistance.
- Drag-and-drop operations on message maps. When you are creating message maps for WebSphere Message Broker message flows, the Mapping editor uses the message model to populate its source and target views. Without message models, you cannot use the Mapping editor.
- Reuse of message models, in whole or in part, by creating additional messages that are based on existing messages.

- Generation of documentation.
- Provision of version control and access control for message models by storing them in a central repository.

To make full use of the facilities that are offered by WebSphere Message Broker, model your message formats.

To speed up the creation of message models, importers are provided to read metadata such as C header files, COBOL copybooks, XML Schema and DTDs, WSDL files, and EIS metadata, and create message models from that metadata. Additionally, predefined models are available for common industry standard message formats such as SWIFT, EDIFACT, X12, FIX, HL7, and TLOG.

Related concepts:

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

"Which body parser should you use?" on page 1077
The characteristics of the messages that your applications exchange indicate which body parser you must use.

"The message model" on page 1160
The message model consists of the following components.

"Project references" on page 44
When a projects refers to other projects in the workspace, this is called a *project reference*. When one project references another, the files in the referenced project are available for use by the referring project. For example to configure certain nodes, to create libraries of reusable message flows, and to enable Content Assist in the ESQL editor.

"Ways to create message definitions" on page 1253
When you have created a message set, you must populate the message set with message definitions.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

Message domains and parsers

WebSphere Message Broker supplies a range of *parsers* to parse and write message formats.

A parser is invoked when the bit stream that represents an input message is converted to the internal form that can be handled by the broker. The internal form, a logical tree structure, is described in "Logical tree structure" on page 1042. Similarly, a parser is invoked to convert a logical tree back into a bit stream.

Each parser is suited to a particular class of messages (for example, fixed-length binary, delimited text, or XML) known as a *message domain*.

When you create a message set, you specify which message domains the message set supports. This determines which parsers are used when you parse and write messages that are defined within that message set.

The parsers that are supplied with WebSphere Message Broker are described in "Parsers" on page 1072.

Related concepts:

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

"Why model messages?" on page 1158
WebSphere Message Broker supplies a range of parsers to parse and write message formats. Some message formats are self-defining and can be parsed without reference to a model. Most message formats, however, are not self-defining, and the parser must have access to a predefined model that describes the message, if it is to parse it correctly.

"Parsers" on page 1072
A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from the internal message tree representation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

The message model

The message model consists of the following components.

- Message set projects
- Message sets
- Message definition files
- WSDL files
- Message categories

See "Message modeling concepts" on page 1155 for a summary of these components, and the relationship between them. See Related Concepts later in this section for a detailed description of each component.

The majority of your model content is described by message definition files. These files use XML Schema to represent your messages. XML Schema is an international standard that defines a language for describing the structure of XML documents. It is ideally suited to describing the messages that flow between business applications, and it is widely used in the business community for this purpose. WebSphere Message Broker uses XML Schema to describe the structure of all kinds of message format, not just XML.

Each message definition file describes both the *logical structure* of your messages, and the *physical formats* that describe the appearance of your message bit stream during transmission. If you are using the MRM or IDOC domains, you *must* provide physical format information. This tells the parser exactly how to parse the message bit stream. If you are not using the MRM or IDOC domains, physical format information is not needed

To understand the different ways that you create and populate message definition files, see "Ways to create message definitions" on page 1253. See "Physical formats in the MRM domain" on page 1211 for a description of the physical formats that are available to you.

Related Concepts

"Message set projects"
A *message set project* is a specialized container in which you create and maintain all the resources associated with one message set.

"Message sets overview" on page 1162
A message set is a container for grouping messages and associated message resources (elements, types, groups).

"Message definition files" on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

"XML Schema" on page 1172
XML Schema is an international standard from the World Wide Web Consortium (W3C) that defines a language for describing the structure of XML documents.

"What is WSDL?" on page 1615
WSDL is an XML notation for describing a web service. A WSDL definition tells a client how to compose a web service request and describes the interface that is provided by the web service provider.

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"Message categories" on page 1200
Message category files have the suffix `.category`. These files are optional. You can have many message category files in a message set.

"Ways to create message definitions" on page 1253
When you have created a message set, you must populate the message set with message definitions.

"Generate model representations" on page 1270
After you have created and populated a message set, you can generate a message model in several different representations for use by a broker, a parser, or your applications.

Related Tasks

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

Message set projects:

A *message set project* is a specialized container in which you create and maintain all the resources associated with one message set.

The content of a message set project is a single message set folder. If the message set is modeling messages from EIS systems, the name of the message set project provides the name of the message set and, optionally, a single Adapters folder. You can create a message set project using the following methods:

- The New Message Set wizard.
- The Quick Start wizards.

These restrictions apply to message set projects:

- A message set project must contain just one message set.
- A message set project cannot refer to any other message set.

Import either of the following samples to see how message set resources are stored in a message set project. The sample's message flow resources are stored separately in a Message Flow project.

- Video Rental
- Comma Separated Value (CSV)

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

"Message sets overview"
A message set is a container for grouping messages and associated message resources (elements, types, groups).

"Message definition files" on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

"Message categories" on page 1200
Message category files have the suffix `.category`. These files are optional. You can have many message category files in a message set.

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Quick Start wizards overview" on page 1409
You can use a Quick Start wizard to set up the basic resources that are required to develop a message flow application. The wizard sets up and gives names to containers for the resources that you need when you develop a message flow.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Deleting a message set project" on page 2839
Delete a message set project and, optionally, the contents of the associated project directory.

Message sets overview:

A message set is a container for grouping messages and associated message resources (elements, types, groups).

A message set is a folder in a message set project that contains a `messageSet.mset` file. The name of the folder is the name of the message set. A message set project can contain just one message set.

When you create a new message set, a new message set project is automatically created with a name that is the same as that of the message set.

You can base your new message set on an existing message set. In this case, all the definitions in the existing message set are copied into the new message set.

When you have created your message set, you must specify the following key properties:

Supported message domains
The message domains that are supported by the message set. The supported domains determine what is generated for deployment to a broker, and are used when parsing and writing the messages that are defined within the message set.

Default message domain
The default domain of the message set.

Use namespaces
Indicates whether the message definitions that you create within the message set are XML namespace aware.

Related concepts:

"Message set projects" on page 1161
A *message set project* is a specialized container in which you create and maintain all the resources associated with one message set.

"Message set resources"
Resources in a message set are created as files, and are displayed under the message set folder in the Broker Development view.

"Message set identification" on page 1167
A message set is identified by the name of the message set folder in the message set.

"Message set limitations" on page 1168
A message set is the original container for message models used by WebSphere Message Broker.

"Namespaces in the message model" on page 1201
Use namespaces to qualify message model object names.

"Message definition files" on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

"Message domains and parsers" on page 1159
WebSphere Message Broker supplies a range of *parsers* to parse and write message formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

Related reference:

"Message set editor" on page 6819
The Message set editor is the default editor provided by the Broker Application Development perspective for editing message set (`messageSet.mset`) files.

"Message set properties" on page 5371
Message sets have properties that you can set to define their characteristics and the way in which they are processed.

Message set resources:

Resources in a message set are created as files, and are displayed under the message set folder in the Broker Development view.

- Message set file `messageSet.mset`

There is always one, and only one, `messageSet.mset` file in a message set. This file contains message model properties that are common to all the content of the message set. It is also where you define the physical formats that you want for this message set. These can be Custom Wire Format (CWF), Tagged Delimited String Format (TDS), and XML Wire Format (XML).

The file is created for you when a new message set is created, and you manipulate its content with the Message Set Editor.

- Message definition files that have the suffix `.mxsd`

  You can have many message definition files in a message set. Each file contains the logical model and the associated physical model, in XML Schema form, for a group of related messages.
- Deployable WSDL files that have the suffix `.wsdl`

  These files are used by the SOAP domain. You can have many WSDL files in a message set.
- Message category files that have the suffix `.category`

  These files are optional. You can have many message category files in a message set. A message category provides another way of grouping your messages, perhaps for documentation purposes.

When you have completed the resources in your message set, you can generate the content of the message set in a form that can be used by a broker parser or an application. This might be:

- a message dictionary for deployment to a broker
- XML Schema for use by an application building XML messages, or for deployment to a broker
- Web Services Description Language (WSDL) for a web services client, or for deployment to a broker
- documentation to give to programmers or business analysts

You can analyze the effects of changes to certain resources; see "Analyzing planned changes to message set resources" on page 1165.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

"Message set projects" on page 1161
A *message set project* is a specialized container in which you create and maintain all the resources associated with one message set.

"Message sets overview" on page 1162
A message set is a container for grouping messages and associated message resources (elements, types, groups).

"Message definition files" on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

"Message categories" on page 1200
Message category files have the suffix `.category`. These files are optional. You can have many message category files in a message set.

"Message domains and parsers" on page 1159
WebSphere Message Broker supplies a range of *parsers* to parse and write message formats.

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"Generate model representations" on page 1270
After you have created and populated a message set, you can generate a message model in several different representations for use by a broker, a parser, or your applications.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Deleting a message set project" on page 2839
Delete a message set project and, optionally, the contents of the associated project directory.

"Adding a Custom Wire Format (CWF)" on page 2848
You can add a Custom Wire Format (CWF) physical format layer to a message set by using the Message Set editor.

"Adding an XML wire format" on page 2853
You can add an XML wire format physical format layer to a message set by using the Message Set editor.

"Adding a TDS physical format" on page 2851
Add a Tagged/Delimited String (TDS) physical format layer to a message set using the Message Set editor.

Related reference:

"What is WSDL?" on page 1615
WSDL is an XML notation for describing a web service. A WSDL definition tells a client how to compose a web service request and describes the interface that is provided by the web service provider.

"Message set editor" on page 6819
The Message set editor is the default editor provided by the Broker Application Development perspective for editing message set (`messageSet.mset`) files.

"Message set properties" on page 5371
Message sets have properties that you can set to define their characteristics and the way in which they are processed.

Analyzing planned changes to message set resources:

Use impact analysis to analyze the effect of renaming message definition or deployable WSDL files.

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 2842
- "Enabling and disabling indexing" on page 1454

You must also have created message set content, for example by importing a WSDL file.

About this task

You can analyze the effect of renaming the following objects:

- Message definition files (`.mxsd`)
- Deployable WSDL files (`.wsdl`)

Procedure

1. In the Broker Development view, right-click the file that you want to rename, then click **Impact Analysis** > **Rename**.
2. In the Impact Analysis - Rename Artifact window, type the new name of the file, then click **Analyze Impact**.

   The Rename Artifact dialog box shows the results of impact analysis, listing primary and secondary impacts.

   You can view the results of impact analysis in the Impact Analysis dialog box, or the Impact Analysis view of the WebSphere Message Broker Toolkit. For more information about how to view information about selected resources, mark changes as complete, copy results to an external application, and view previous results, see "Impact Analysis view" on page 6801.

:

Related concepts:

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

"Impact analysis: analyzing the effects of planned changes to your applications" on page 1150
When developing an application, you might want to change the names of some artifacts, or move them. Impact analysis reports the artifacts that are likely to be affected by a particular change.

Related tasks:

"Enabling and disabling indexing" on page 1454
Enable indexing to support impact analysis.

"Analyzing planned changes to ESQL objects" on page 2403
Use impact analysis to analyze the effect of renaming or moving ESQL objects, or moving or changing the broker schema of an ESQL file.

"Analyzing the impact of changes to message maps" on page 2234
Use impact analysis to analyze the effect of renaming or moving message maps.

"Analyzing planned changes to message flows" on page 1436
Use impact analysis to analyze the effect of renaming or moving message flows, including subflows.

"Analyzing planned changes to message model objects" on page 2897
Use impact analysis to analyze the effect of renaming message model objects.

Related reference:

"Impact analysis: reference" on page 4174
Some artifacts are excluded from secondary analysis.

"Impact Analysis view" on page 6801
The Impact Analysis view shows information about selected resources, which changes have been marked as complete, and previous results. You can also use this view to copy results to the clipboard.

Message set identification:

A message set is identified by the name of the message set folder in the message set.

When you must refer to a message set from a message flow (for example, when setting the Message Set property of an input node), use the message set name.

A message set also has a 13-character identifier that is guaranteed to be unique. You can use this identifier, instead of the message set name, to refer to a message set, but only if you are using the MRM or IDOC domains. Other domains do not recognize the identifier.

A message set also has an alias. An alias can be used only with MRM multipart messages.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

"Message set projects" on page 1161
A *message set project* is a specialized container in which you create and maintain all the resources associated with one message set.

"Message sets overview" on page 1162
A message set is a container for grouping messages and associated message resources (elements, types, groups).

"Multipart messages" on page 1191
A *multipart message* contains one or more other messages within its structure. The contained message is sometimes referred to as an *embedded message*.

"Message definition files" on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

"Message categories" on page 1200
Message category files have the suffix `.category`. These files are optional. You can have many message category files in a message set.

"Message domains and parsers" on page 1159
WebSphere Message Broker supplies a range of *parsers* to parse and write message formats.

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Creating a message set" on page 2842
Use the New Message Set wizard to create a message set.

"Deleting a message set project" on page 2839
Delete a message set project and, optionally, the contents of the associated project directory.

"Adding a Custom Wire Format (CWF)" on page 2848
You can add a Custom Wire Format (CWF) physical format layer to a message set by using the Message Set editor.

"Adding an XML wire format" on page 2853
You can add an XML wire format physical format layer to a message set by using the Message Set editor.

"Adding a TDS physical format" on page 2851
Add a Tagged/Delimited String (TDS) physical format layer to a message set using the Message Set editor.

Related reference:

"Message set editor" on page 6819
The Message set editor is the default editor provided by the Broker Application Development perspective for editing message set (`messageSet.mset`) files.

"Message set properties" on page 5371
Message sets have properties that you can set to define their characteristics and the way in which they are processed.

Message set limitations:

A message set is the original container for message models used by WebSphere Message Broker.

You can have as many message definition files as you want within one message set, but you must limit your message sets to a few related message definition files that share the same physical formats.

There are several reasons for the suggested limitations:

- Generation of a message dictionary and other representations is quicker.
- Generated documentation is more manageable.
- MRM physical formats apply to *all* objects within the message set.

  Therefore, for example, if you are using the MRM domain and have an XML message and an unrelated CWF message modeled in the same message set, CWF physical format properties are present for all objects. But the CWF properties are of no interest to the XML message and therefore take default values in those objects, which can result in unwanted task list warnings.
- You cannot use recursion for MRM CWF and TDS physical formats.

  Therefore, if you are modeling XML messages that have a recursive structure, you must ensure that recursive XML messages do not share a message set with MRM CWF or TDS physical formats.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

"Message set projects" on page 1161
A *message set project* is a specialized container in which you create and maintain all the resources associated with one message set.

"Message sets overview" on page 1162
A message set is a container for grouping messages and associated message resources (elements, types, groups).

"Message definition files" on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

"Message categories" on page 1200
Message category files have the suffix `.category`. These files are optional. You can have many message category files in a message set.

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Deleting a message set project" on page 2839
Delete a message set project and, optionally, the contents of the associated project directory.

"Adding a Custom Wire Format (CWF)" on page 2848
You can add a Custom Wire Format (CWF) physical format layer to a message set by using the Message Set editor.

"Adding an XML wire format" on page 2853
You can add an XML wire format physical format layer to a message set by using the Message Set editor.

"Adding a TDS physical format" on page 2851
Add a Tagged/Delimited String (TDS) physical format layer to a message set using the Message Set editor.

Related reference:

"Message set editor" on page 6819
The Message set editor is the default editor provided by the Broker Application Development perspective for editing message set (`messageSet.mset`) files.

"Message set properties" on page 5371
Message sets have properties that you can set to define their characteristics and the way in which they are processed.

Message set version and keywords:

When you develop a message set, you can define the version of the message set, and other key information that you want to be associated with it.

After you have deployed the message set in a BAR file, you can view the message set properties in the WebSphere Message Broker Toolkit and WebSphere Message Broker Explorer. The properties include the deployment and modification dates and times (the default information that is displayed), and the additional version or keyword information that you have set.

You can define information to give details of the message set that has been deployed; therefore, you can check that it is the message set that you expect.

Version

You can set the version of the message set in the `Version` property.

You can also define a default message set version in the `Default version tag` of the message set preferences. All message sets that you create after you have set this property have this default applied to the `Version` property at the message set level.

Keywords

You must define keywords in the `Documentation` property of the message set. Keywords follow certain rules to ensure that the information can be parsed. The following example shows the type of information that you can define in the `Documentation` property:

```
$MQSI Author=John Smith MQSI$
```

The following table contains the information that is displayed by the WebSphere Message Broker Toolkit or Explorer:

Message set name	
Deployment Time	28-Aug-2004 15:04
Modification Time	28-Aug-2004 14:27
Version	1.0
Author	John Smith

In this display, the version information has been defined by using the `Version` property of the object. If you have not defined version information by using the `Version` property, it is omitted from this display.

Restrictions within keywords

Do not use the following characters within keywords, because they cause unpredictable behavior:

```
^ $ . | \ < > ? + * = & [ ] ( )
```

You can use these characters in the values that are associated with keywords; for example:

- `$MQSI RCSVER=$id$ MQSI$` is acceptable
- `$MQSI $name=Fred MQSI$` is not acceptable

Related concepts:

"Message sets overview" on page 1162
A message set is a container for grouping messages and associated message resources (elements, types, groups).

"Message definition files" on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

"Version and keyword information for deployable objects" on page 1443
Use the Broker Archive editor to view the version and keyword information of deployable objects.

"Message flow version and keywords" on page 1445
When you are developing a message flow, you can define the version of the message flow as well as other key information that you want to be associated with it.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Deleting a message set project" on page 2839
Delete a message set project and, optionally, the contents of the associated project

directory.

Related reference:

"Message set properties" on page 5371
Message sets have properties that you can set to define their characteristics and the way in which they are processed.

"Guidance for defining keywords" on page 4017
You can add extra information to an object in the form of one or more keywords.

"Message set preferences" on page 5366
Preferences for message sets.

Message definition files:

A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

Every message set requires at least one message definition file to describe its messages. Message definition files use the XML Schema language to describe the *logical format* of one or more messages. Extra information in the form of XML Schema annotations is used to describe any *physical formats* that you define for the messages.

Large message sets can contain several message definition files. This keeps the individual files to a manageable size, making them faster and easier to work with.

Message definition files can be created by using the Message Definition editor, or can be imported from a range of different file formats as described in "Importing from other model representations to create message definitions" on page 1254.

A message definition file can be associated with a namespace, so that all message model objects that are declared within the file belong to that namespace. Namespaces provide a means of avoiding name clashes among similarly named global objects. They are described in detail in "Namespaces in the message model" on page 1201.

One message definition file can reuse message model objects that are defined in another message definition file. XML Schema provides two mechanisms to do this: **`import`** and **`include`**. For more information, see "Reusing message model files" on page 1209.

Related concepts:

"Message sets overview" on page 1162
A message set is a container for grouping messages and associated message resources (elements, types, groups).

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

"Namespaces in the message model" on page 1201
Use namespaces to qualify message model object names.

"XML Schema" on page 1172
XML Schema is an international standard from the World Wide Web Consortium (W3C) that defines a language for describing the structure of XML documents.

"Ways to create message definitions" on page 1253
When you have created a message set, you must populate the message set with message definitions.

"Reusing message model files" on page 1209
One message definition file can reuse message model objects defined in another file.

Related tasks:

"Creating a message definition file" on page 2865
Creating an empty message definition file to contain your message model objects.

XML Schema:

XML Schema is an international standard from the World Wide Web Consortium (W3C) that defines a language for describing the structure of XML documents.

The XML Schema language is ideally suited to describing the messages that flow between business applications, and it is widely used in the business community for this purpose.

WebSphere Message Broker uses XML Schema 1.0 to describe the logical structure of messages. At a simple level, the types and elements in the message are modeled by using XML Schema types and elements. However, when the need arises, all of the advanced modeling features of XML Schema are available for modeling messages.

Some important restrictions and extensions of XML Schema exist. These are documented in "XML Schema restrictions in message sets" and "XML Schema extensions in message sets" on page 1173.

Further information about XML Schema: For details about XML Schema, see XML Schema Part 0: Primer on the World Wide Web Consortium (W3C) website.

Related concepts:

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

"XML Schema restrictions in message sets"
Some XML Schema 1.0 features are not supported when message models are contained in message sets.

"XML Schema extensions in message sets" on page 1173
WebSphere Message Broker provides some additional facilities that are not specified in the XML Schema 1.0 specification. When using a message set, further extensions are provided.

XML Schema restrictions in message sets:

Some XML Schema 1.0 features are not supported when message models are contained in message sets.

Unsupported XML Schema features: The following feature is accepted, but not supported, and causes validation errors if it is used in your message model:

- Redefines

A quick fix is provided to convert Redefines into Include.

Further information about XML Schema: For details about XML Schema, see XML Schema Part 0: Primer on the World Wide Web Consortium (W3C) website.

Related concepts:

"XML Schema extensions in message sets"
WebSphere Message Broker provides some additional facilities that are not specified in the XML Schema 1.0 specification. When using a message set, further extensions are provided.

Related reference:

"Importing from XML Schema: unsupported features" on page 6359
A number of features in XML Schema are not supported, or their support is restricted in some way.

XML Schema extensions in message sets:

WebSphere Message Broker provides some additional facilities that are not specified in the XML Schema 1.0 specification. When using a message set, further extensions are provided.

Messages: A message is a global element that represents an entire message (rather than a structure within a message). Within a message definition file, a message is represented by a special global element that carries the extra information required by WebSphere Message Broker.

Composition: The message model in a message set adds the following compositions that are beyond the XML Schema 1.0 specification:

message
: A refinement of *choice* that can contain only a set of references to messages within the same message set. Groups and complex types with composition of *message* are used when modeling multipart messages.

orderedSet
: A set of elements that must be present in the order that they are listed. Groups cannot be used within an *orderedSet*. Elements can repeat, but duplicate elements cannot be used.

unorderedSet
: A set of elements that can be present in any order. Groups cannot be used within an *unorderedSet*. Unlike an *all* group, elements within an *unorderedSet* can repeat. However, duplicate elements cannot be used.

Compositions *orderedSet* and *unorderedSet* enable message models that were produced in earlier versions of the product to be supported.

Content validation: The message model in a message set adds a validation control called 'Content validation' that is used only if the domain is MRM or IDOC, and if validation is selected. It determines how strictly the content of the group is validated. See MRM content validation for more details.

The Content validation property does not affect validation in the XMLNSC or SOAP domains. Validation in these domains follows the rules of XML Schema 1.0.

Physical format information: If one or more physical formats are defined for a message set, the XML Schema objects within the message set can hold extra information about how they must be parsed and serialized.

Further information about XML Schema: For details about XML Schema, see XML Schema Part 0: Primer on the World Wide Web Consortium (W3C) website.

Related concepts:

"XML Schema restrictions in message sets" on page 1172
Some XML Schema 1.0 features are not supported when message models are contained in message sets.

Message model objects:

An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

Message
A *message* describes the structure and content of a set of data that is exchanged between applications that send and receive the data. A *message* is a special complex element.

Simple element
A *simple element* describes one or more named data fields in a message. It is based on a *simple type* (for example, string, integer or float). A simple element can repeat, and it can define a default or a fixed value.

Simple type
A *simple type* describes a class of data within a message. It describes the type of data (for example, string, integer or float) and it can have value constraints which place limits on the values of any simple elements based on that simple type.

Complex element
A *complex element* describes a named complex structure within the message. The content of a complex element is defined by a *complex type*. A complex element can repeat.

Complex type
A *complex type* describes a complex structure within a message. It contains *elements* (simple or complex), *attributes* (if the data is XML), and *groups* that are organized into a tree-like hierarchy.

Group A *group* describes a list of elements with information about how those elements can appear in a message. Groups can be ordered (sequence), unordered (all), or selective (choice). A group can repeat.

Attribute
An `attribute` describes an XML attribute. Attributes are similar to simple elements, but they require special treatment when used with XML messages. In messages that are not XML messages, attributes are not used.

Global and local objects: Most objects in the message model can be either global or local. A global object must have a unique name, which is used to refer to the object from one or more places in the message model. Local objects are defined and used in only one place in the message model.

Make objects local unless they must be used in more than one place. This reduces the probability of name clashes among the global objects in the message model, and makes the message set easier to work with.

Properties of message model objects: All message model objects have properties. The properties fall into three categories:

Logical
The logical properties of an object are defined by XML Schema. They relate to the format-independent description of the object called the 'logical model'. Logical properties describe *what* data the object contains without saying anything about *how* it is written down.

Physical
If the message model is for a data format that is not XML, additional physical properties are provided for an object that describe how the object is written down. These properties control the parsing and writing of the object. If the message model is in a message set, then the properties are an IBM proprietary set that is understood by the MRM domain and parser.

Documentation
This field is present for all message model objects. It provides a standard place for any description of the object that you might require. Text entered here does not affect the processing of messages in any way.

Related concepts:

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

Message model objects: messages:

A *message* describes the structure and content of a set of data that is passed from one application to another.

A message consists of elements that are organized into a logical structure agreed by the sending and receiving applications. This logical structure can be modeled so that message data can be parsed into a logical tree and manipulated easily by the broker.

In the message model, a message is always based on a global element. The complex type of a global element describes the contents of the global element, and therefore describes all of the content of the message.

Multipart messages: If necessary, a message can contain other messages, and is necessary for modeling certain large and complex messaging standards such as SWIFT and EDIFACT. Such a message is known as a *multipart message*. The contained messages are known as *embedded messages*.

Message identification: Messages are identified by their name. In a message set only, a message can also be identified by an alias. The alias is an optional user-specified string that identifies the (multipart) message. The name and alias of a message must be unique within a message set.

XML Schema model: In the message definition file, a message is modeled as an XML Schema global element declaration. Extra information is provided by XML Schema annotations on the element declaration.

Related concepts:

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere

Message Broker extension to XML Schema.

"Multipart messages" on page 1191
A *multipart message* contains one or more other messages within its structure. The contained message is sometimes referred to as an *embedded message*.

"Message model objects: elements"
An *element* is a named piece of information (a field) within a message, with a meaning that is agreed by the applications that create and process the message.

Related reference:

"Message Definition editor" on page 6804
The Message Definition editor is the default editor provided by the Broker Application Development perspective for editing message definition (`.mxsd`) files.

Message model objects: elements:

An *element* is a named piece of information (a field) within a message, with a meaning that is agreed by the applications that create and process the message.

An element has a specific meaning that is agreed by the applications that create and process the message. For example, a message might include a string that your applications have agreed is a 'Customer Name'. An element is always based on a type, either simple or complex.

An element:

- Has a business meaning.
- Is an instantiation of a simple or complex type.
- Can be accessed by name from ESQL, Java, or PHP.
- Is further defined by its type; for example, the type defines the range of values that an element can have.
- Can be defined globally or locally.

Simple and complex elements: Elements can be simple or complex. A simple element is a single, named piece of information such as 'Age' or 'Customer Name'. A simple element is based on a *simple type* that defines its content.

A complex element is a named structure that contains other elements. A complex element named 'Customer Details' might contain the simple elements 'Age' and 'Customer Name'. A complex element can also contain other complex elements. A complex element is based on a *complex type* that defines its content and structure.

Global and local elements: Elements can be global or local. A *global element* can be used in several different messages, or even in several places within the same message. It must be given a unique name by which it can be referenced by an *element reference*. A *local element* is defined in one position within one complex type or group, and is not available for reuse elsewhere in the message model.

Optional and repeating elements: Elements can be defined as optional, mandatory, and repeating, by using the properties `Min Occurs` and `Max Occurs`. For further information, see "Cardinality: optional, required, and repeating elements" on page 1197.

Default and fixed values: An element can be given a default value, so that if no value is supplied by the message, the default value is used. Alternatively, a fixed value can be defined, and the element always takes that value. The precise use of default and fixed values is dependent on the message domain.

Value constraints: The value of an element can be constrained by using *value constraints* which define the range of legal values for the element. The value constraints are associated with the simple type on which the element is based. For further information, see "Message model objects: simple types" on page 1180. The XML Schema term for a value constraint is a *facet*.

Defining substitution groups: If you are modeling XML messages, an element can be marked as a valid substitute for another element by using the substitution group property on the element. In this way, groups of elements can be assembled where any of the elements in the group can substitute for one element, the *head* element. For further information, see "Substitution groups in the message model" on page 1199.

Related concepts:

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

"XML Schema" on page 1172
XML Schema is an international standard from the World Wide Web Consortium (W3C) that defines a language for describing the structure of XML documents.

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

"Message model objects: simple types" on page 1180
A *simple type* is an abstract definition of an item of data such as a number, a string, or a date.

"Message model objects: complex types" on page 1178
A *complex type* describes the structure of one or more complex elements.

"Message model objects: simple type value constraints" on page 1188
Value constraints, also known as *facets* in XML Schema, refine a simple type by defining limits on the values that it can represent.

"Cardinality: optional, required, and repeating elements" on page 1197
The number of occurrences of an element can be controlled using the properties `Min Occurs` and `Max Occurs`. Using these properties, an element can be defined as mandatory, optional or repeating.

Related tasks:

"Adding MRM message model objects" on page 2870
Various tasks are involved in adding message model objects to a message definition file.

Message model objects: types:

Types describe the data content of elements.

Simple types describe simple elements with data types such as string, integer, or dateTime.

Complex types describe complex elements - elements that contain a hierarchy of other elements.

For more information, see:

- "Message model objects: simple types" on page 1180
- "Message model objects: complex types" on page 1178

- "Message model objects: type inheritance" on page 1182

Related concepts:

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Message model objects: complex types:

A *complex type* describes the structure of one or more complex elements.

Complex types are an essential part of every message model because they define the logical structure of the messages and elements in the model.

What is a complex type for?: Complex types define the structure of the messages and elements in the message model. By combining elements, attributes, groups, and wildcards, almost any message structure can be modeled.

Contents of a complex type:

Elements

Most complex types contain some elements, and some contain a large hierarchy of complex elements. The elements within a complex type are always contained within a group. This group can be local to the complex type, in which case the Message Definition Editor hides it from view.

Alternatively, the group that contains the elements can be a global group, and this group defines the element content, the composition, and the content validation for the complex type.

If a complex type is derived from a simple type, it cannot contain any element content.

Attributes

If you are modeling XML messages, your complex types can contain attributes. The attributes for a complex type can be local or global, and they can be contained within an attribute group.

Groups

Groups enable sets of elements to be included in a complex type. The members of the group are included as peers of the other elements. For more information about their use, see "Message model objects: groups" on page 1183.

Wildcards

If you are modeling XML messages, your complex types can contain wildcard elements. Wildcard elements enable unmodeled elements to be present in the complex type. Any such, elements must be present within the message at the same position as the wildcard. Complex types can also contain wildcard attributes. Wildcard attribute enable unmodeled attributes to be present within any elements that are based on the complex type.

Global and local complex types: Complex types can be global or local. A global complex type can be used as the basis for more than one complex element. It must be given a unique name by which it can be referenced. A local complex type is associated with a single complex element, and is not available for reuse elsewhere in the message model. Local types do not have a name, and are sometimes referred to as anonymous types.

Composition: The composition of a complex type describes how the members of the type are organized. For more information, see "Message model objects: groups" on page 1183.

Controlling validation of type content: The *Content validation* parameter on a complex type specifies how strictly the contents of the type is validated. For more information, see "Message model objects: groups" on page 1183.

Substitution settings: A complex type has parameters that control whether other types can be derived from it (final) and whether other types can substitute for it (block). For more information, see "Substitution groups in the message model" on page 1199 and "Message model objects: type inheritance" on page 1182.

Related concepts:

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

"Message model objects: Wildcard elements" on page 1186
For XML messages, a wildcard element represents an element that is not present in the message model, but which might be present at the position of the wildcard element in a message.

"Message model objects: simple types" on page 1180
A *simple type* is an abstract definition of an item of data such as a number, a string, or a date.

"Message model objects: groups" on page 1183
A *group* is a list of elements that defines how those elements appear in a message. Groups define the composition and content validation of a complex type.

"Message model objects: Wildcard attributes" on page 1187
For XML messages, a *wildcard attribute* enables unmodeled attributes to be present in a message.

"Message model objects: type inheritance" on page 1182
The XML Schema language allows a type definition to be based on another type definition. In this way, a hierarchy of types can be constructed.

"Substitution groups in the message model" on page 1199
Substitution groups are an XML Schema feature that provides a way of substituting one element for another in an XML message.

Related reference:

"Message Definition editor" on page 6804
The Message Definition editor is the default editor provided by the Broker Application Development perspective for editing message definition (`.mxsd`) files.

Message model objects: simple types:

A *simple type* is an abstract definition of an item of data such as a number, a string, or a date.

The purpose of a simple type is to define the content of one or more simple elements. Simple types (and any elements that are based on simple types) cannot contain attributes or child elements. Simple types stand in contrast to complex types, which define the structure of an element, but typically do not define any simple data.

Global and local simple types

Simple types can be global or local. A global simple type can be used as the basis for more than one element. It must be given a unique name by which it can be referenced. A local simple type is associated with a single element, and is not available for reuse elsewhere in the message model. Local types do not have a name and are sometimes referred to as anonymous types.

Variations of simple types

Built-in

XML Schema defines many simple types for you to use, covering all the standard data types such as strings, integers, decimals, and floats.

Restriction

You can define your own simple types by deriving from another simple type (the base type) by restriction. A restriction type can have value constraints applied to it.

A restriction type can derive from a built-in simple type or a restriction simple type.

List

For XML messages only, a list type is a way of rendering a repeating simple value in XML. The notation is more compact than the notation for a repeating element, and offers a way to have multi-valued attributes.

A list type can be based on a union type (introduced later in this section). This can describe a space-separated list of items in which each item can be based on any of the simple types in the union.

A list of lists is not legal. The item type of a list cannot be a list itself, or derived at any level from another list type.

A list type can have the facets of minLength, maxLength, and length applied to it. These facets restrict the number of items in the list. To restrict the values of each item in the list, facets must be applied to the item type and not to the list itself. The message definition editor provides additional support for enumeration and pattern facets directly on a List type, to enable the import of any schema that uses them, but issues a warning that enumeration and pattern facets are ignored by the broker.

Union

A union type is a union of two or more other simple types.

A union type enables a value to conform to any one of several different simple types. The simple types that comprise a union type are known as

its member types. There is no upper limit on how many member types can exist, but there must be at least one. A member type can be defined as a built-in simple type, a user defined simple type, or a local simple type defined anonymously within the union type.

A union type can also include list, union, and restricted simple types, among its members.

MRM domain

The MRM parser does not apply value constraints until the data is in the logical tree. This means that it is not possible to choose between two simple types that are derived from the same fundamental type, but have different value constraints (for example, an integer in the range 1-10 and an integer in the range 11-20). If you attempt to choose between two such types, a warning is displayed in the task list, and the parser ignores the value constraints when it resolves the union. The message definition editor provides additional support for enumeration and pattern facets directly on a Union type, to enable the import of any schema that uses them, but the editor issues a warning that enumeration and pattern facets are ignored by the MRM parser.

Value constraints

Value constraints are known as facets by XML Schema. Any value constraints that are applied to a derived type must further restrict the base type. It is not valid for a derived type to weaken or remove a value constraint that its base type has defined. If no value constraints are applied to the derived type, the derived type is almost identical to its base type, but it is treated as a restriction of the base type in situations where that is relevant (type inheritance and element substitution).

Related concepts:

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

"Message model objects: type inheritance" on page 1182
The XML Schema language allows a type definition to be based on another type definition. In this way, a hierarchy of types can be constructed.

"Message model objects: simple type value constraints" on page 1188
Value constraints, also known as *facets* in XML Schema, refine a simple type by defining limits on the values that it can represent.

"Substitution groups in the message model" on page 1199
Substitution groups are an XML Schema feature that provides a way of substituting one element for another in an XML message.

Related tasks:

"Adding a simple type" on page 2886
Add a simple type to your message model.

Related reference:

"Message Definition editor" on page 6804
The Message Definition editor is the default editor provided by the Broker Application Development perspective for editing message definition (`.mxsd`) files.

"Simple type logical properties" on page 5450
The logical properties of a simple type.

Message model objects: type inheritance:

The XML Schema language allows a type definition to be based on another type definition. In this way, a hierarchy of types can be constructed.

This topic outlines the concepts of *type inheritance*, and highlights some important issues relating to substitution.

A full discussion of XML Schema type inheritance can be found on the World Wide Web Consortium (W3C) website, or in numerous books about XML Schema.

Restriction and extension: A type is a *restriction* of its base type, if elements of the derived type have a smaller range of valid values (or valid type members) than elements of the base type.

- For example, a restriction of a complex type might reduce the number of occurrences of one of its type members, or might omit that type member completely.
- Similarly, a restriction of a simple type might lower the `Max Inclusive` facet value, or raise the `Min Inclusive` facet value.

A type is an *extension* of its base type if elements of the derived type have a wider range of valid values (or valid type members) than elements of the base type.

- For example, an extension of a complex type might add type members that were not present in the base type, or might allow a type member to repeat.
- Similarly, an extension of a simple type must *always* be a complex type that is based on the simple type; you cannot extend a simple type by widening its range of valid values.

Special rules apply to the derivation of simple types. A simple type cannot extend another simple type. This ensures that restrictions that are imposed by a simple type cannot be removed by deriving another simple type from it.

However, a complex type can extend a simple type. This does not affect the range of valid values of the simple type, but it does allow attributes to be added. The result of extending a simple type is always a complex type that contains zero or more attributes.

Controlling type inheritance: The `final` attribute on a complex type can take three values, with the following effects:

- restriction: It is not valid to derive another complex type from this type by restriction.
- extension: It is not valid to derive another complex type from this type by extension.
- all: It is not valid to derive another complex type from this type by either extension or restriction

Type inheritance and substitution: XML Schema provides two different substitution mechanisms, both of which use type inheritance information to allow or disallow substitutions.

Element substitution is controlled by substitution groups, and element substitution can be blocked or allowed for extension and restriction by settings on either the element itself or the type of the element.

Type substitution allows the type of the element to be defined within the instance document, using the xsi:type attribute on the element, so that the real type of the element is not known until the element has been partly parsed. This mechanism can also be blocked or allowed based on the derivation method of the types involved.

Related concepts:

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

Message model objects: groups:

A *group* is a list of elements that defines how those elements appear in a message. Groups define the composition and content validation of a complex type.

Groups can be ordered (sequence or orderedSet) unordered (all or unorderedSet), or selective (choice or message). Groups define the composition and content validation of a set of type members.

What are groups for?: Groups can be used for any of the following purposes:

- To define the entire element content of a complex type.

  A complex type can refer to a global group that completely defines its content. (If it does not, the content of the complex type is defined by an *anonymous local group*, which is hidden within the Message Definition Editor.)
- To represent a common substructure within more than one type.

  Two or more complex types can refer to the same global group, if they both contain the same subset of elements.
- To change the composition midway through a complex type.

  You might have a complex type that is a sequence of three members, but the second member is a choice of two elements. To model this circumstance, a group with `composition` set to choice can be inserted as the second member of the sequence.

Contents of a group: Groups can contain complex elements, simple elements, wildcard elements, and groups.

By combining these components, the structure of any message can be modeled. Wildcard elements can be included to enable the presence of unmodeled elements, thus making the message model robust and flexible.

Global and local groups: Groups can be global or local.

A *global group* can be used in more than one place in the message model. It represents a structure that is present in more than one place in the message model. A global group must be given a unique name by which it can be referenced.

A *local group* is defined in one position within one group, and is not available for reuse elsewhere in the message model. Local groups do not have a name, and are displayed by using the composition of the group.

Composition: In XML Schema, a group can have its `composition` set to sequence, all, or choice.

- A sequence is a set of elements that must be present in the same order as they are listed.
- An all group is a set of elements that can be present in any order, and cannot repeat.
- A choice is a set of elements, only one of which can be present in any given message.

When using a message set, other compositions are also possible. For more information, see "XML Schema extensions in message sets" on page 1173.

Content validation: The `Content validation` property is applied only if the domain is MRM or IDOC, and if validation is selected.

Content validation determines how strictly the content of the group is validated. See "MRM content validation" on page 5422 for more details.

Allowable values of the `Content validation` property are:

Closed
: The contents of the group are validated strictly against the model. Only elements that are defined as children of the group can be present as children.

Open Defined
: Elements that are declared within the same message set can be present as children of the group, even if they are not defined as children.

Open Any elements can be present as children of the group.

The `Content validation` property does not affect validation in the XMLNSC or SOAP domains. Validation in these domains follows the rules of XML Schema 1.0.

Related concepts:

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

"Message model objects: complex types" on page 1178
A *complex type* describes the structure of one or more complex elements.

"XML Schema extensions in message sets" on page 1173
WebSphere Message Broker provides some additional facilities that are not specified in the XML Schema 1.0 specification. When using a message set, further extensions are provided.

Related tasks:

"Adding a global group" on page 2890
Add a global group to your message model.

Related reference:

"Message Definition editor" on page 6804
The Message Definition editor is the default editor provided by the Broker Application Development perspective for editing message definition (`.mxsd`) files.

Message model objects: attributes:

An *attribute* describes an XML attribute. They are used only when the data is XML.

Attributes are provided to simplify the modeling of XML messages; if none of your messages use the XML physical format, use simple elements instead.

Attributes and XML: The most common use for an attribute is to model an XML attribute within an XML message. In this scenario, each attribute that can be present in the XML message has a corresponding attribute in the logical message definition.

Attributes in other physical formats: Sometimes a message must be parsed as XML, but written in another physical format (Custom Wire Format or Tagged Delimited String Format). In this case, any attributes in the message are treated in the same way as simple elements with the same properties.

Global and local attributes: Attributes can be global or local.

A *global attribute* can be used in more than one place in the message model. It must be given a unique name by which it can be referenced by an *attribute reference*.

A *local attribute* is defined in one position within one complex type, and cannot be used elsewhere in the message model.

Optional attributes: Attributes can be defined in XML Schema as optional, required, or prohibited. Attributes cannot repeat. For further information, see "Cardinality: optional, required, and repeating elements" on page 1197.

Default and fixed values: An attribute can be given a default value so that, if the attribute is missing from the message, the default is used. Alternatively, a fixed value can be defined, and the attribute always takes that value. The precise use of default and fixed values is dependent on the message domain.

Value constraints: The value of an attribute can be constrained by using *value constraints*, which define the range of legal values for the attribute. Value constraints are associated with the simple type on which the attribute is based. For more details, see "Message model objects: simple types" on page 1180. In XML Schema, the term for *value constraint* is *facet*.

Related concepts:

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

Related tasks:

"Adding a global attribute" on page 2881
Add a global attribute to your message model.

"Adding a local attribute" on page 2882
Add a local attribute to a message, complex type, or complex element.

Message model objects: Wildcard elements:

For XML messages, a wildcard element represents an element that is not present in the message model, but which might be present at the position of the wildcard element in a message.

Wildcard elements provide a means of adding flexibility to the message model, so that messages can be parsed even if they do not exactly match the message model.

Wildcard elements can be present only within a complex type or group with `Composition` of sequence and `Content Validation` of closed. Wildcard elements provide a similar capability to setting the `Content Validation` property of a complex type or group to Open or Open Defined.

The `Process Content` and `Namespace` properties control the namespace to which elements that are present at the position of the wildcard element must belong.

MRM domain

If you have enabled validation in your message flow, and your message is in the MRM domain, wildcard elements are validated against the model according to the following rules:

- If `Process Content` is set to strict, only elements that are declared in the same message set are able to be present in the position of the wildcard element.
- If `Process Content` is set to lax or skip, any element is able to be present in place of the wildcard element.

If you are working with WebSphere Message Broker Version 6.0 or earlier, the number of elements that can match the wildcard element is unpredictable (`Min Occurs` and `Max Occurs` are ignored).

Related concepts:

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

"Message model objects: complex types" on page 1178
A *complex type* describes the structure of one or more complex elements.

"Message model objects: groups" on page 1183
A *group* is a list of elements that defines how those elements appear in a message. Groups define the composition and content validation of a complex type.

"Message model objects: Wildcard attributes" on page 1187
For XML messages, a *wildcard attribute* enables unmodeled attributes to be present in a message.

Related tasks:

"Adding a wildcard element" on page 2880
Add a wildcard element to a message, type, group, or complex element in a message model.

"Adding a wildcard attribute" on page 2885
Add a wildcard attribute to a message, complex type, or complex element.

Related reference:

"MRM content validation" on page 5422
`Content Validation` is applied when the domain is MRM and validation is enabled. The `Content Validation` property specifies how strictly the MRM parser validates the members of a complex type or group.

Message model objects: Wildcard attributes:

For XML messages, a *wildcard attribute* enables unmodeled attributes to be present in a message.

The `Process Content` and `Namespace` properties control the namespace to which attributes that are present in the position of the wildcard must belong.

MRM domain

If you have enabled validation in your message flow, and your message is in the MRM domain, wildcard attributes are validated against the model according to the following rules:

- If `Process Content` is set to strict, only attributes which are declared in the same message set can be present in the position of the wildcard attribute.
- If `Process Content` is set to lax or skip, any attribute can be present in the position of the wildcard attribute.

Tip: If the namespace property is set to the namespace of the message set, these rules are then similar to the behavior of the XMLNSC domain in validating mode.

Related concepts:

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

"Message model objects: complex types" on page 1178
A *complex type* describes the structure of one or more complex elements.

"Message model objects: groups" on page 1183
A *group* is a list of elements that defines how those elements appear in a message. Groups define the composition and content validation of a complex type.

Related tasks:

"Adding a wildcard element" on page 2880
Add a wildcard element to a message, type, group, or complex element in a message model.

"Adding a wildcard attribute" on page 2885
Add a wildcard attribute to a message, complex type, or complex element.

Related reference:

"MRM content validation" on page 5422
`Content Validation` is applied when the domain is MRM and validation is enabled. The `Content Validation` property specifies how strictly the MRM parser validates the members of a complex type or group.

Message model objects: attribute groups:

For XML messages, an *attribute group* defines a set of attributes that can be present in a complex type.

An attribute group provides a way to include the same set of attributes in more than one complex type, without duplicating the definitions.

For example, if most of the elements in your message model have the attributes 'amount', 'currency' and 'date', these elements can be contained in a single attribute group, which is referenced by all the complex types that use them.

Related concepts:

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

"Message model objects: complex types" on page 1178
A *complex type* describes the structure of one or more complex elements.

"Message model objects: attributes" on page 1185
An *attribute* describes an XML attribute. They are used only when the data is XML.

Message model objects: simple type value constraints:

Value constraints, also known as *facets* in XML Schema, refine a simple type by defining limits on the values that it can represent.

It is often useful to be able to constrain the values that an element or attribute can take, perhaps to ensure that messages conform to business rules. This topic describes how to add value constraints to a simple type, in order to constrain the values of all elements or attributes that are based on that simple type.

If the model is deployed to WebSphere Message Broker, elements and attributes can be validated against value constraints, so that violations are reported as errors or warnings. The XMLNSC domain uses all the different types of value constraint when validating. The MRM domain uses a subset; the restrictions are noted later in this section.

Types of value constraint:

Length Constraints: `Length`, `Min Length`, `Max Length`
Using length constraints, the length of all elements based on the simple type can be constrained, or even limited to a single value.

Length constraints can be applied to simple types that are derived from xsd:hexBinary, xsd:base64Binary or xsd:string (including built-in schema types such as xsd:normalisedString).

Length constraints are inherited from ancestor types, and any length constraints that are defined for a simple type must not relax the constraints that are imposed by any of its ancestor types. For example, a type 'longString' (`Max Length`=100) cannot be derived from a type 'shortString' (`Max Length`=10).

Note: For the MRM domain, by default, `Length` value constraints are converted to `Max Length` constraints when a message set is added to a BAR file. This default avoids WebSphere Message Broker raising spurious validation errors for fixed-length data structures, where the strings tend to be padded to fit a fixed-width field. If strict length validation is required, this default can be changed in the message set properties by changing the flag `Broker treats Length facet as MaxLength`.

Range constraints: `Min Inclusive`, `Max Inclusive`, `Min Exclusive`, `Max Exclusive`
Range constraints specify the allowable range of values for all elements that are based on the simple type. Inclusive constraints include the specified endpoints in the permitted range, whereas exclusive constraints do not. Range constraints can be applied to simple types that are numeric, or that relate to calendar and time values. They cannot be applied to strings, because the ordering of string values depends on the character set that is used.

Range constraints are inherited from ancestor types, and any range constraints that are defined for a simple type must not relax the constraints that are imposed by any of its ancestor types. For example, a type 'largeNumber' (`Max Inclusive`=100) cannot be derived from a type 'smallNumber' (`Max Inclusive`=10).

Note: For the MRM domain, exclusive constraints cannot be applied to non-integral types (float, decimal, double, dateTime, and other non-integral types).

Enumeration constraints
An enumeration constraint specifies a single permitted value for all elements that are based on the simple type. A list of permitted values can be specified by defining more than one enumeration constraint for the same simple type. Enumeration constraints can be applied to all simple types.

Enumeration constraints are inherited from ancestor types, and any set of enumeration constraints that are defined for a simple type must not increase the range of permitted values. For example, a type 'AllColors' (with enumerations for all colors of the rainbow) cannot be derived from a type 'MonoColors' (with enumerations for 'black' and 'white' only).

Precision constraints: Total Digits and Fraction Digits
Precision constraints relate only to decimal and integer values. They limit the number of significant digits (total digits) and, for decimals, the number of decimal places (fraction digits) for all elements that are based on the simple type. Precision constraints can be applied to simple types that are derived from xsd:decimal and xsd:integer.

Precision constraints are inherited from ancestor types, and any precision constraints that are defined for a simple type must not relax the constraints that are imposed by any of its ancestor types. For example, a type 'veryPrecise' (`Fraction Digits`=10) cannot be derived from a type 'notVeryPrecise' (`Fraction Digits`=1).

Note: For the MRM domain, the broker applies these constraints only to xsd:decimal and user types that are derived from it; precision constraints that are applied to an integer simple type are ignored.

Pattern constraints

A pattern constraint is a regular expression that specifies a set of permitted values for all elements that are based on the simple type. Multiple patterns can be defined for the same simple type, permitting complex validation rules to be expressed in logically separate parts. Each pattern constraint on a simple type contributes to the set of permitted values for elements that are based on the simple type; that is, all the patterns are combined by using Boolean OR.

As with all value constraints, a simple type can inherit pattern constraints from the simple type on which it is based. In this case, the set of pattern constraints that are contributed by each ancestor type must be satisfied, in addition to the set that is contributed by the simple type itself; that is, the sets of pattern constraints from each level in the type hierarchy are combined by using Boolean AND.

Note: For the MRM domain, pattern constraints can be applied only to simple types that are derived from xsd:string.

White space constraints

A white space constraint specifies how a parser treats white space for all elements that are based on the simple type.

Note: For the MRM domain, white space constraints are not applied. The MRM physical formats enable white space to be precisely controlled for each physical format that is defined for the message, but these physical properties are separate from the white space constraint in the logical model, and are not used for validation.

Related concepts:

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

"Message model objects: complex types" on page 1178
A *complex type* describes the structure of one or more complex elements.

"Message model objects: type inheritance" on page 1182
The XML Schema language allows a type definition to be based on another type definition. In this way, a hierarchy of types can be constructed.

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Message model object identification:

Objects in a message model (elements, attributes, types, groups) are identified by their name only.

This means that no two objects in the same scope can have the same name. Name clashes can be avoided more easily if global objects are used only when necessary. Local objects are not visible outside of the scope of their parent object, so their names can be reused without causing a name clash.

Namespaces
: If namespaces are enabled for a message set, each message definition file within the message set can specify a namespace. Namespaces are an XML Schema mechanism for organizing groups of related objects into a named 'module'.

  Global objects in different namespaces can share the same name. Therefore, namespaces offer another means of avoiding name clashes among global objects.

Valid names
: Since the message model is based on the XML Schema language, the name of every message model object must be a valid XML Schema identifier. For information about what constitutes a valid XML Schema identifier, see XML Schema Part 0: Primer.

For details about XML Schema, see XML Schema Part 0: Primer on the World Wide Web Consortium (W3C) website.

Related concepts:

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

Multipart messages:

A *multipart message* contains one or more other messages within its structure. The contained message is sometimes referred to as an *embedded message*.

Tip: Multipart messages are used when working with messages that are modeled by using message sets.

A multipart message must contain a group, or a complex type, with its `Composition` property set to Message. This group or complex type can contain a list of references to messages that can be present at that location in the message structure. If the group or complex type is empty, any message can be present. When a message is parsed, only one embedded message can be present in that location.

Message envelopes: A common use of multipart messages is to define an outer message with a fixed structure. This outer message is called the *message envelope*. Within the message envelope a group or complex type is included, as described earlier in this topic. Examples of message standards that can be modeled by using this technique are EDIFACT, X12, SWIFT, SOAP XML, SAP ALE IDoc, multipart MIME, and RosettaNet.

Identifying the embedded message: When a multipart message is parsed, the parser must be able to identify the embedded message; it might be any of the messages that are referenced by the group or complex type, or it might be a message that is not referenced by the group or complex type, perhaps from a different message set. This is achieved by using one of four techniques, *Automatic*, *Message Identity*, *Message Path*, or *Manual*.

Automatic
: Used when parsing XML messages, such as SOAP. The parser automatically identifies and parses embedded messages by using the tag in the XML document.

Message Identity
: Used by the MRM parser. See "Identifying an embedded message by using a Message Identity" on page 1193.

Message Path
: Used by the MRM parser. See "Identifying an embedded message by using a Message Path" on page 1196.

Manual
: Used by the MIME parser. The parser treats embedded messages as BLOBs. If you want to parse the BLOB by using another parser, you must do so manually by using ESQL, or Java, or a ResetContentDescriptor node.

Restrictions: Unless using the *Manual* identification technique, all embedded messages must be of the same physical format as the outermost message, and have the same character set and encoding.

When using the *Automatic* or *Message Path* identification techniques, all embedded messages must be from the same message set as the multipart message.

Related concepts:

"Identifying an embedded message by using a Message Identity" on page 1193
You can identify an embedded message by using the Message Identity. This technique is used by the MRM domain, and replaces the use of the Message Key.

"Identifying an embedded message by using a Message Path" on page 1196
The Message Path technique for identifying embedded messages is useful when the multipart message contains no information about the identity of an embedded message.

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

"MRM Custom wire format: Multipart messages" on page 1217
The Custom Wire Format (CWF) supports both the Message Identity technique and the Message Path technique of identifying embedded messages within a multipart message.

"MRM XML physical format: Multipart messages" on page 1250
Identify embedded messages by using either a Message Identity or a Message Path.

"MRM TDS format: Multipart messages" on page 1241
The Tagged/Delimited String Format (TDS) supports both the Message Identity technique and the Message Path technique of identifying embedded messages within a multipart message.

"MIME parser and domain" on page 1117
Use the MIME domain if your messages use the MIME standard for multipart messages.

Related tasks:

"Creating a multipart message" on page 2919
A multipart message occurs when you embed a message in another message.

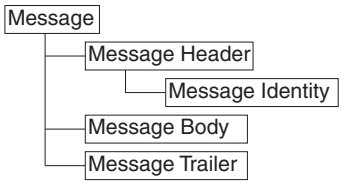
"Accessing embedded messages in the MRM domain" on page 2594
If you have defined a multipart message, you have at least one message embedded within another. Within the overall complex type that represents the outer messages, you can model the inner message in two ways.

Identifying an embedded message by using a Message Identity:

You can identify an embedded message by using the Message Identity. This technique is used by the MRM domain, and replaces the use of the Message Key.

Tip: Multipart messages are used when working with messages that are modeled by using message sets.

The Message Identity technique for identifying embedded messages is useful when a multipart message has a format such as that shown in the diagram:



In this example, the Message Header and Message Trailer act as an envelope for the message body. They typically have a fixed structure, although the Message Body can be defined with many different structures.

A place holder for an embedded message is created by setting the `Composition` property of the complex type or group of the Message Body element to Message. This enables an embedded message to be added within the outer Message, creating a multipart message.

When using the Message Identity technique to parse such a multipart message, the embedded message must be identified earlier in the Message Header by using a Message Identity element. A Message Identity Element is a string element (or attribute) that precedes the embedded message in the model and whose `Interpret Value As` property is set to Message Identity.

When a multipart message is input to a message flow, the Message Identity element must have a value that corresponds to either the `Name` or the `Message`

Alias of the next embedded message in the bit stream. This enables the MRM parser to correctly identify the embedded message in the model.

For cases where the Message Identity element value does not match the Name of the message, use the Message Alias property to specify this value. The MRM parser tries to match on Name first, and if that fails, it tries to match on Message Alias.

When the MRM parser has encountered a Message Identity element, its value applies to all embedded messages that are contained immediately within the current message. This does not apply to embedded messages within embedded messages; any embedded message must have its identity provided by a Message Identity element within its immediate parent message.

If a second Message Identity element is encountered within the current message, its value overrides any previously held. This enables different peer embedded messages to exist within a given message.

Message Identity takes priority over Message Path. If both are specified, Message Identity is used. Use only one of these techniques for a given multipart message.

Embedded messages defined in different message sets

By default, an embedded message is assumed to be defined within the same message set as the current message. This can be overridden by using a Message Set Identity, which works in a similar manner to a Message Identity.

An embedded message that is defined within a different message set must have its message set identified earlier in the message, by using a Message Set Identity element. A Message Set Identity Element is a string element (or attribute) that precedes the embedded message in the model and whose Interpret Value As property is set to Message Set Identity.

When a multipart message is input to a message flow, the Message Set Identity element must have a value that corresponds to either the Identifier, Name, or Message Set Alias of the message set that defines the next embedded message in the bit stream. This enables the MRM parser to correctly identify the message set to use.

If the Message Set Identity element value does not match the Identifier or Name of the message set, use the Message Set Alias property to specify this value. The MRM parser tries to match on Identifier first, then on Name, and finally on Message Set Alias.

After the MRM parser has encountered a Message Set Identity element, its value applies to all embedded messages that are contained within the current message. It also applies to embedded messages within embedded messages, unless an embedded message also contains a Message Set Identity element.

If a second Message Set Identity element is encountered within the current message, its value overrides any previously held. This enables peer embedded messages to be contained within different message sets.

The following example of an X12 message shows the use of both Message Identity and Message Set Identity. The field that contains 004010X092 within the GS segment on line 0002 holds the Message Set Identity as a Message Set Alias. The

207 on line 0003 in the ST segment is the Message Identity held as a `Message Alias`. The embedded message is from line 0004 - 0015 inclusive.

Note: The line numbers and spaces at the beginning of each line are for illustrative purposes only and do not exist in the actual message.

```
0001  ISA*00*          *00*          *30*12-3456789     *ZZ
      *9876543-21     *000104*1820*U*00401*000000001*0*T*:!
0002  GS*HS*HOSP CLAIM*PAYER ADJDEPT*20000104*1820*1*X*004010X092!
0003  ST*270*1234!
0004  BHT*0022*13*10001234*19990501*1319!
0005  HL*1**20*1!
0006  NM1*PR*2*ABCCOMPANY*****PI*842610001!
0007  HL*2*1*21*1!
0008  NM1*1P*2*BONE AND JOINT CINIC*****SV*2000035!REF*N7*234899!
0009  N3*55*HIGH STREET!
0010  N4*SEATTLE*WA*98123!
0011  HL*3*2*22*0!TRN*1*93175-12547*9877281234!
0012  NM1*IL*1*SMITH*ROBERT*B***MI*11122333301!
0013  REF*1L*599119!
0014  DMG*D8*19430519*M!
0015  DTP*472*RD8*19990501-19990515!EQ*30**FAM!SE*17*1234!
0016  GE*1*1!IEA*1*000000001!
```

Physical format considerations

Both Message Identity and Message Set Identity are applicable to all physical formats. Versions of the TDS physical format before Version 6.0 included embedded message identification by Message Key, which worked in a similar manner to Message Identity. Message Key has been deprecated and is superseded by Message Identity.

Related concepts:

"Multipart messages" on page 1191
A *multipart message* contains one or more other messages within its structure. The contained message is sometimes referred to as an *embedded message*.

"Identifying an embedded message by using a Message Path" on page 1196
The Message Path technique for identifying embedded messages is useful when the multipart message contains no information about the identity of an embedded message.

"MRM Custom wire format: Multipart messages" on page 1217
The Custom Wire Format (CWF) supports both the Message Identity technique and the Message Path technique of identifying embedded messages within a multipart message.

"MRM XML physical format: Multipart messages" on page 1250
Identify embedded messages by using either a Message Identity or a Message Path.

"MRM TDS format: Multipart messages" on page 1241
The Tagged/Delimited String Format (TDS) supports both the Message Identity technique and the Message Path technique of identifying embedded messages within a multipart message.

Related tasks:

"Creating a multipart message" on page 2919
A multipart message occurs when you embed a message in another message.

"Accessing embedded messages in the MRM domain" on page 2594
If you have defined a multipart message, you have at least one message embedded within another. Within the overall complex type that represents the outer messages, you can model the inner message in two ways.

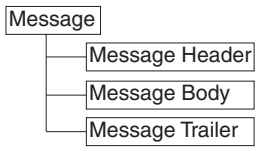
Identifying an embedded message by using a Message Path:

The Message Path technique for identifying embedded messages is useful when the multipart message contains no information about the identity of an embedded message.

Tip: Multipart messages are used when working with messages that are modeled by using message sets.

This technique is used by the MRM domain.

In the diagram, the Message Header and Message Trailer act as an envelope for the message body. Typically, they have a fixed structure, but the Message Body can be defined with many different structures.



A place holder for an embedded message is created by setting the `Composition` property of the complex type or group of the Message Body element to Message. This enables an embedded message to be added within the outer message, creating a multipart message.

When using the Message Path technique to parse such a multipart message, the embedded message must be identified by a fixed path to the innermost message from the outermost message. For this example, this would be:

```
Message/Message Body
```

If the path to the innermost message contains intermediate elements, these intermediate elements must also be included in the path. In the following example, these elements are shown in bold:

```
Message/Data1/Data12/Message Body
```

This technique can be used to identify nested embedded messages as well, by extending the path. For example:

```
Message/Data1/Data12/Message Body/Data2/Inner Message
```

The path is specified by using one or both of two properties, the `Message Type` property of a WebSphere Message Broker input node (or MQRFH2 header) and the `Message Type Prefix` property of the containing message set. These two properties are combined to produce a final path that is used to locate embedded messages.

Message Identity takes priority over Message Path. If both are specified, Message Identity is used. Use only one of these techniques for a given multipart message.

You cannot use the Message Path technique to identify multiple peer embedded messages.

Embedded messages defined in different message sets

This option is not supported by the Message Path technique.

Physical format considerations

The Message Path technique is applicable to all physical formats.

Related concepts:

"Specifying namespaces in the Message Type property" on page 1208
When using the MRM domain, the *Message Type* property is used to specify the name of the message.

"Multipart messages" on page 1191
A *multipart message* contains one or more other messages within its structure. The contained message is sometimes referred to as an *embedded message*.

"Identifying an embedded message by using a Message Identity" on page 1193
You can identify an embedded message by using the Message Identity. This technique is used by the MRM domain, and replaces the use of the Message Key.

"MRM Custom wire format: Multipart messages" on page 1217
The Custom Wire Format (CWF) supports both the Message Identity technique and the Message Path technique of identifying embedded messages within a multipart message.

"MRM XML physical format: Multipart messages" on page 1250
Identify embedded messages by using either a Message Identity or a Message Path.

"MRM TDS format: Multipart messages" on page 1241
The Tagged/Delimited String Format (TDS) supports both the Message Identity technique and the Message Path technique of identifying embedded messages within a multipart message.

Related tasks:

"Creating a multipart message" on page 2919
A multipart message occurs when you embed a message in another message.

"Accessing embedded messages in the MRM domain" on page 2594
If you have defined a multipart message, you have at least one message embedded within another. Within the overall complex type that represents the outer messages, you can model the inner message in two ways.

Related reference:

"Message set properties" on page 5371
Message sets have properties that you can set to define their characteristics and the way in which they are processed.

Cardinality: optional, required, and repeating elements:

The number of occurrences of an element can be controlled using the properties `Min Occurs` and `Max Occurs`. Using these properties, an element can be defined as mandatory, optional or repeating.

Elements: A *mandatory*, or required, element has `Min Occurs`>= 1. A mandatory element must occur at least once in the message.

An *optional* element has `Min Occurs` = 0. An optional element can be omitted from the message.

A *repeating* element has `Max Occurs`> 1 to indicate a bounded number of repeats, or `Max Occurs`=unbounded (sometimes displayed as -1), to indicate an unlimited number of repeats. A repeating element occurs more than once in the message, and all the occurrences must appear together without any other elements between them.

If a complex type or a group contains two, or more, members that refer to the same element, the second reference is a *duplicate*. This is different from a repeating element, because the two references are typically separated by other members of the type or group. In the message, the second occurrence typically does not appear immediately after the first occurrence.

Attributes: The number of occurrences of an attribute can be controlled by setting it to *required*, *optional* or *prohibited*.

A *required* attribute is similar to a mandatory element - it must occur in the message.

An *optional* attribute is similar to an optional element - it can be omitted from the message.

A *prohibited* attribute must not appear in the message.

An attribute is not allowed to repeat, and duplicate attribute references are not allowed within an attribute group.

Related concepts:

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

"Message model objects: elements" on page 1176
An *element* is a named piece of information (a field) within a message, with a meaning that is agreed by the applications that create and process the message.

"Message model objects: types" on page 1177
Types describe the data content of elements.

"Message model objects: attributes" on page 1185
An *attribute* describes an XML attribute. They are used only when the data is XML.

Self-defining elements and messages:

An instance element is *predefined* if it is possible for the parser to find a matching element definition in the message model with an appropriate set of properties and in the correct context. Otherwise, it is *self-defining*. Similarly, an entire message is self-defining if no corresponding message is present in the message model.

Self-defining elements can be used only when the format of the message is a self-describing one, such as XML or JSON. For general text or binary formats (for example, comma separated), you must ensure that your message model defines every message and every element that must be parsed.

If you have chosen not to model your messages, or you have a model but have chosen not to deploy it to the broker, all messages and elements are self-defining. In this situation, you cannot use message definitions to influence the parsing and writing of elements; the self-defining elements are parsed and written according to the default behavior of the parser and writer.

Self-defining elements, and all elements within a self-defining message, are not validated against value constraints, and any missing fields are not assigned default or fixed values, and all data is assumed to be string type unless the parser is able to deduce the type in a reliable manner.

Related concepts:

"Why model messages?" on page 1158
WebSphere Message Broker supplies a range of parsers to parse and write message formats. Some message formats are self-defining and can be parsed without reference to a model. Most message formats, however, are not self-defining, and the parser must have access to a predefined model that describes the message, if it is to parse it correctly.

"Message model objects: simple type value constraints" on page 1188
Value constraints, also known as *facets* in XML Schema, refine a simple type by defining limits on the values that it can represent.

Substitution groups in the message model:

Substitution groups are an XML Schema feature that provides a way of substituting one element for another in an XML message.

A substitution group is a list of global elements that can be present in place of another global element, called the *head element*.

A substitution group is defined by setting the `substitution group` property on one global element (the *member* element) to point at another global element (the *head* element). This adds the member element to the substitution group of the head element.

Head elements

A head element is an element that can be substituted. When a message is parsed, one of its member elements can be present in place of the head element without causing a validation error.

Abstract elements

An abstract element is a head element which must be substituted, and is indicated by the 'abstract' attribute on the element. Typically, abstract elements have other elements in their substitution group, otherwise they are of little use. Wherever an abstract element is present in a message definition, a member of its substitution group must be present instead.

The `block` attribute on elements

The `block` attribute on an element limits the set of global elements that can substitute for the element. The `block` attribute can take any subset of the values restriction, extension, substitution, or all.

- If the `block` attribute contains restriction, an element that is based on a restriction of the type of the element cannot be substituted for the element.
- If the `block` attribute contains extension, an element that is based on an extension of the type of the element cannot be substituted for the element.
- If the `block` attribute contains substitution, an element that is a member of the substitution group of the element cannot be substituted for the element.
- If the `block` attribute contains all, all of the above limits apply.

The `final` attribute on elements

The `final` attribute on an element limits the set of global elements that can be a member of the substitution group of the element. The `final` attribute can take any subset of the values restriction, extension, or all.

- If the `final` attribute contains restriction, an element that is based on a restriction of the type of the element cannot be in the substitution group of the element.
- If the `final` attribute contains extension, an element that is based on an extension of the type of the element cannot be in the substitution group of the element.
- If the `final` attribute contains all, both of the above limits apply.

The `block` attribute on complex types

The `block` attribute on a complex type limits the set of other types that can substitute for that type. The `block` attribute can take values restriction, extension, or all. The meanings for these values are the same as the values that are shown for the block attribute on an element. An element that is a member of a substitution group can substitute only for the head element if its type is compatible with the block attribute on the type of the head element.

Default `block` and `final` attributes

A default for the `block` and `final` attributes can be set at the message definition file level. If a default for one or both of these attributes has been set and the relevant `block` or `final` attribute has not been set at the object level, the default setting is used for that object. You can override the default setting at the object level.

Related concepts:

"XML Schema" on page 1172
XML Schema is an international standard from the World Wide Web Consortium (W3C) that defines a language for describing the structure of XML documents.

"Message model objects: type inheritance" on page 1182
The XML Schema language allows a type definition to be based on another type definition. In this way, a hierarchy of types can be constructed.

Message categories:

Message category files have the suffix `.category`. These files are optional. You can have many message category files in a message set.

A message category provides another way of grouping your messages, perhaps for documentation or convenience purposes. A message category can also be used for assisting in the generation of Web Services Description Language (WSDL) files, but this use is deprecated. For more information, see "Generate WSDL" on page 1274.

A message set category file is created by using the New Message Category File wizard.

When you have created your message category file, you must specify one key property.

`Message Category Kind`

The Message Category Kind indicates whether the message category is to participate in the generation of WSDL files.

You can then add messages to the message category file. If the message category is to participate in WSDL generation, assign appropriate values to the `Role Name` and `Role Type` properties of each member message.

Message category identification: The name of a message category is provided by the name of the `.category` file. You can change the message category name by renaming the file.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

"Message sets overview" on page 1162
A message set is a container for grouping messages and associated message resources (elements, types, groups).

"Message definition files" on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Creating a message category file" on page 2924
Create a message category file to add categories that you can use to group different message sets.

"Deleting a message category file" on page 2930
You can delete a message category file from your message model.

Related reference:

"Message Category editor" on page 6802
The Message Category editor is the default editor provided by the Broker Application Development perspective for working with message category (`.category`) files in a message set.

"Message category properties" on page 5413
A message category provides a way of grouping your messages.

Namespaces in the message model:

Use namespaces to qualify message model object names.

A single XML instance document can contain elements and attributes that are defined for, and possibly used by, multiple applications. Two different elements or attributes within the same document might require the same name. Individual applications must be able to recognize the elements and attributes that they are designed to process. In circumstances such as this, the definitions can be distinguished from each other by qualifying each element with a different namespace. This avoids problems of name collision and mistaken recognition.

XML Schemas can define a target namespace. Global elements, attributes, groups, and types that are defined within an XML Schema are qualified by the target namespace, if it has been defined. Optionally, local elements and attributes can also be qualified by the target namespace. Therefore, namespaces assist in the development of a library of XML Schemas that can be developed independently. If the namespace name that is used for an XML Schema is unique, you do not have to be concerned about name clashes with objects that are defined within other XML Schemas.

The scope of a namespace extends beyond the scope of its containing document and is identified by a Uniform Resource Identifier (URI). In order to serve its purpose, a URI must be unique. You might be more familiar with the concept of a Universal Resource Locator (URL). URIs often use the same syntax as URLs, but the URI definition is broader than the specification of a URL. This is an example of a URI: `http://mycompany.com/xml_schema`

A namespace prefix is declared as a shorthand for the full URI name and this is used to qualify all elements that belong to that namespace. The prefix to be substituted for a namespace in an XML instance document or XML Schema is specified by using an `xmlns` attribute. A default namespace can also be defined by using an `xmlns` attribute. If a default namespace is defined, any element or attribute with no prefix is qualified with the default namespace. If no default namespace is defined, any element or attribute with no prefix is not qualified by a namespace.

Namespaces and the message model

The message model supports namespaces within message sets. However, you can choose whether to enable or disable namespaces for your message set. If you disable namespaces when you create your message set, you can enable namespaces later. However, when you have enabled namespaces for a message set you cannot disable namespaces.

A single message set which has namespaces enabled can contain a number of different namespaces. Each namespace is represented by a different Message Definition File. When you create a Message Definition File, you can choose whether it has an associated namespace, or whether it is in the notarget namespace. If you associate a namespace with a Message Definition File, you must also choose a prefix.

If the Message Definition File has an associated namespace, the following global objects are qualified with the namespace:

- Elements
- Attributes
- Simple Types
- Complex Types
- Groups
- Attribute Groups

Optionally, local elements and attributes can be qualified with the namespace.

Objects that are defined within a Message Definition File can reference objects in other Message Definition Files within the same message set. To do this, import or include one Message Definition file within another Message Definition File.

Message parsing and message flows

WebSphere Message Broker parsers for XML data recognize prefixed names in the XML messages that they parse, and internally map these to the correct namespace. The message tree stores the name and the namespace of the element or attribute.

Namespaces can be used even when the data is not XML. Adapter schema, and message definition files can be created with an associated target namespace. Though the data itself does not contain prefixed names, the namespace is obtained from the corresponding element in the file. Again, the message tree stores the name and the namespace of the element.

If you are using XML format in the MRM domain, elements or attributes are matched, based on the namespace in the dictionary when the parsed message is matched against the dictionary that is generated from the message model. Therefore, for an element or attribute in a message to match with the dictionary, both its name and its namespace must match.

If you are using the DataObject domain, the SOAP domain, or the XMLNSC domain (in validating mode), elements or attributes are matched, based on the namespace in the XML Schema when the parsed message is matched against the XML Schema that is generated from the message model. Therefore, for an element or attribute in a message to match the XML Schema, both the name and the namespace of the element or attribute must match.

You can specify namespaces when writing ESQL or Java. It is not necessary to write ESQL or Java that is namespace aware if you are not using namespaces. However, if you decide to use namespaces it is necessary to write namespace-aware ESQL or Java. The namespace in which an element is contained is stored in the message tree when parsed. This is a logical property and it is held regardless of the physical wire format in which messages are parsed and written. Syntax has been added to ESQL to make it easy to reference namespaces of other elements by using defined prefixes. In Java, XPath expressions are used to reference elements.

Importing from other formats

The message model enables you to create Message Definition files from other formats by importing them into the WebSphere Message Broker Toolkit.

- If you import an XML DTD file, the Message Definition File that is created is in the notarget namespace.
- If you import an XML Schema file, the target namespace of the created Message Definition File depends on whether namespaces have been enabled for the message set.
  - If namespaces are enabled, the target namespace of the Message Definition File that is created is the target namespace of the XML Schema that is being imported.
  - If namespaces are disabled for the message set, the created Message Definition File is in the notarget namespace. This type of import does not provide full namespace support. If you are using WebSphere Message Broker, you do not have to write namespace-aware ESQL or Java to process an XML message that is parsed against the dictionary that is generated from this message model. For reasons why you might want to do this, see “Importing XML Schema into message sets with namespaces disabled” on page 1259
- If you import a COBOL Copybook or a C Header file, the target namespace of the created Message Definition File depends on whether namespaces have been enabled for the message set.
  - If namespaces are enabled, the target namespace of the Message Definition File that is created is the notarget namespace. This default namespace can be overridden by specifying a target namespace in the New Message Definition File wizard. For reasons why you might want to do this, see “Namespaces with non-XML messages” on page 1206.
  - If namespaces are disabled for the message set, the Message Definition File that is created is in the notarget namespace

Further information about XML: On the World Wide Web Consortium (W3C) website, see:

- Extensible Markup Language (XML)
- XML Schema Part 0: Primer
- Namespaces in XML

Related concepts:

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"Namespaces with non-XML messages" on page 1206
The use of namespaces by WebSphere Message Broker is not necessarily limited to XML message models.

"Specifying namespaces in the Message Type property" on page 1208
When using the MRM domain, the *Message Type* property is used to specify the name of the message.

"Reusing message model files" on page 1209
One message definition file can reuse message model objects defined in another file.

"Importing XML Schema into message sets with namespaces disabled" on page 1259
You can import an XML Schema file with a target namespace even if the message set does not have namespaces enabled.

"Importing from XML Schemas to create message definitions" on page 1256
You can populate a message set with message definitions by importing XML Schema files, by using the **New Message Definition File From XML Schema file** wizard, the **Start from WSDL and/or XSD files** quick start wizard, or the **mqsicreatemsgdefs** command-line utility.

Related tasks:

"Importing from XML Schema" on page 2957
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to create a new message definition from an XML Schema

"Creating a message set" on page 2842
Use the New Message Set wizard to create a message set.

"Adding an XML wire format" on page 2853
You can add an XML wire format physical format layer to a message set by using the Message Set editor.

"Creating a message definition file" on page 2865
Creating an empty message definition file to contain your message model objects.

"Configuring XML Wire Format properties: Message model objects" on page 2916
You can configure the XML Wire Format properties of a message model object.

"Importing from XML DTD" on page 2954
You can create a new message definition from an XML DTD by using the New Message Definition File wizard in the WebSphere Message Broker Toolkit.

Related reference:

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"Import formats" on page 6346
Supported features of formats that have been imported from C, COBOL, SCA Export, SCA Import, WSDL, or XML Schema.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

Namespaces with MRM XML messages:

The namespace that is associated with a message definition file is part of the logical layer of the message model.

Therefore, it is not dependent on an XML Wire Format being present. However, if you have an XML Wire Format, the namespace information from the logical layer is used to populate some of the properties of the XML Wire Format. If namespaces are enabled for a Message Set, in the XML Wire Format, a table of namespace URI/prefix pairs is maintained. This table is initially populated with the namespaces of all of the Message Definition Files with their prefixes when they were created.

If your message set has namespaces enabled, the broker does not store the values of any `xmlns` attributes in the tree when it parses an XML instance document. It also does not store the values of any `Schema Location` and `No Namespace Schema Location` attributes. When an XML document is written out, the broker regenerates this information from the properties of the XML Wire Format of the message set.

The table of namespace URI/prefix pairs is used by the MRM Domain when it produces an XML message. Elements and attributes that are qualified by a namespace are prefixed with the corresponding prefix from the table. The broker also manages the output of the corresponding `xmlns` attributes that map the prefixes to namespaces. You can choose whether `xmlns` attributes for all of the entries in the namespace URI/prefix table are written at the start of the document, or whether they are only written in the document when required.

If namespaces are enabled for a Message Set, in the XML Wire Format there is a table of schema locations that map namespace URIs to file names. You can add entries to this table and you can map a file name to the notarget namespace. If you are using WebSphere Message Broker, this table is used to produce `schemaLocation` and `No Namespace Schema Location` attributes at the start of the XML document.

Related concepts:

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"Namespaces in the message model" on page 1201
Use namespaces to qualify message model object names.

"Namespaces with non-XML messages" on page 1206
The use of namespaces by WebSphere Message Broker is not necessarily limited to XML message models.

"Specifying namespaces in the Message Type property" on page 1208
When using the MRM domain, the *Message Type* property is used to specify the name of the message.

"Reusing message model files" on page 1209
One message definition file can reuse message model objects defined in another file.

"Importing XML Schema into message sets with namespaces disabled" on page 1259
You can import an XML Schema file with a target namespace even if the message set does not have namespaces enabled.

"Importing from XML Schemas to create message definitions" on page 1256
You can populate a message set with message definitions by importing XML Schema files, by using the **New Message Definition File From XML Schema file** wizard, the **Start from WSDL and/or XSD files** quick start wizard, or the **mqsicreatemsgdefs** command-line utility.

Related tasks:

"Importing from XML Schema" on page 2957
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to create a new message definition from an XML Schema

"Creating a message set" on page 2842
Use the New Message Set wizard to create a message set.

"Adding an XML wire format" on page 2853
You can add an XML wire format physical format layer to a message set by using the Message Set editor.

"Creating a message definition file" on page 2865
Creating an empty message definition file to contain your message model objects.

"Configuring XML Wire Format properties: Message model objects" on page 2916
You can configure the XML Wire Format properties of a message model object.

"Importing from XML DTD" on page 2954
You can create a new message definition from an XML DTD by using the New Message Definition File wizard in the WebSphere Message Broker Toolkit.

Related reference:

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"Import formats" on page 6346
Supported features of formats that have been imported from C, COBOL, SCA Export, SCA Import, WSDL, or XML Schema.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

Namespaces with non-XML messages:

The use of namespaces by WebSphere Message Broker is not necessarily limited to XML message models.

There is one scenario where the use of namespaces by non-XML message models can simplify the ESQL or Java code that you write. Before describing this scenario, it is important to understand that the MRM parser, when parsing messages that are defined in a file that has a target namespace, produces a logical message tree that contains both name and namespace information. For non-XML messages, the namespace is obtained from the element declaration in the file, and not from the data.

Consider a transformation scenario where a message from a COBOL application requires to be transformed into namespace-aware XML; for example, a SOAP XML message. The transform must map the logical message tree that was created for the COBOL message to a logical message tree that matches the XML message. If the COBOL message tree does not contain namespace information, each mapping from a COBOL field to an XML element must set the namespace for the XML element. However, if the COBOL message tree already contains the required namespace information, this mapping is much simpler.

To enable the MRM parser to create namespace information in a message tree that was created from a non-XML message, you must specify a target namespace for the message definition file. For MRM, this must be done as part of the file creation process. Make the target namespace of the file the same as the target namespace of the XML message into which the non-XML message is being transformed.

- If you are creating your non-XML message model by hand in the message editor, use the New Message Definition File wizard to specify a target namespace.
- If you are importing from COBOL or C, use the New Message Definition File wizard, or the **mqsicreatemsgdefs** command options file, to specify a target namespace.

When dealing with both the message tree for the non-XML message and the message tree for the XML message, the ESQL or Java code that you write to perform the transformation must be namespace aware.

Related concepts:

"Namespaces in the message model" on page 1201
Use namespaces to qualify message model object names.

"Namespaces with MRM XML messages" on page 1205
The namespace that is associated with a message definition file is part of the logical layer of the message model.

"Specifying namespaces in the Message Type property" on page 1208
When using the MRM domain, the *Message Type* property is used to specify the name of the message.

"Reusing message model files" on page 1209
One message definition file can reuse message model objects defined in another file.

Related tasks:

"Creating a message set" on page 2842
Use the New Message Set wizard to create a message set.

"Creating a message definition file" on page 2865
Creating an empty message definition file to contain your message model objects.

"Importing from XML DTD" on page 2954
You can create a new message definition from an XML DTD by using the New Message Definition File wizard in the WebSphere Message Broker Toolkit.

"Importing from XML Schema" on page 2957
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to create a new message definition from an XML Schema

"Importing from C" on page 2934
Create a message definition file from a C header file for use in the MRM and IDOC domains.

"Importing from COBOL copybooks" on page 2937
This topic describes how to create a new message definition from a COBOL data structure using the New Message Definition File wizard in the WebSphere Message Broker Toolkit.

Related reference:

"Import formats" on page 6346
Supported features of formats that have been imported from C, COBOL, SCA Export, SCA Import, WSDL, or XML Schema.

Specifying namespaces in the Message Type property:

When using the MRM domain, the *Message Type* property is used to specify the name of the message.

The format of a simple message type is `{namespace-uri}:name` where *name* is the name of the message, and *namespace-uri.* identifies the namespace. The namespace must be the full URI specification and must be enclosed in braces.

The format `{namespace-uri}name` (that is, with no colon) is also valid. This maintains compatibility with previous versions of the broker product.

If you omit `{namespace-uri}`, the first match for the name that is found in the model is used. You can do this if namespaces are not enabled for the message set, or if a name is unique within a message set. However, if a name is not unique, you must specify the namespace to be sure that the correct match is made in the model.

The following are examples of message types:

- A simple message type for a message in a real target namespace: `{http://www.ibm.com/space}:name`
- A simple message type for a message in the notarget namespace: `{}:name`
- A simple message type for a message in a message set that does not support namespaces: `name`

When identifying an embedded message using a message path, a message type path would be entered as A simple message type for a message in a real target namespace: `{http://www.ibm.com/space}:name`

The same name can occur in more than one namespace. To specify that a name is to be qualified with a specific namespace, the name must be prefixed with the namespace within the `Message Type`.

For example a `Message Type` with a single name would be entered as:

`{http://www.ibm.com/space}:id/.../{http://www.ibm.com/space}:name`

Related concepts:

"Namespaces in the message model" on page 1201
Use namespaces to qualify message model object names.

"Identifying an embedded message by using a Message Path" on page 1196
The Message Path technique for identifying embedded messages is useful when the multipart message contains no information about the identity of an embedded message.

"Namespaces with MRM XML messages" on page 1205
The namespace that is associated with a message definition file is part of the logical layer of the message model.

"Namespaces with non-XML messages" on page 1206
The use of namespaces by WebSphere Message Broker is not necessarily limited to XML message models.

"Reusing message model files" on page 1209
One message definition file can reuse message model objects defined in another file.

Related tasks:

"Applying a Quick Fix to a task list error" on page 2862
During the creation, migration and manipulation of message models, warnings or errors might occur; these are listed in the Problems view of the Broker Application Development perspective. Some of these warnings or errors can be cleared by applying a Quick Fix.

"Handling large MRM messages" on page 2605
When an input bit stream is parsed, and a logical tree created, the tree representation of an MRM message is typically larger, and in some cases much larger, than the corresponding bit stream.

"Creating a message set" on page 2842
Use the New Message Set wizard to create a message set.

"Creating a message definition file" on page 2865
Creating an empty message definition file to contain your message model objects.

Related reference:

"Message set properties" on page 5371
Message sets have properties that you can set to define their characteristics and the way in which they are processed.

Reusing message model files:

One message definition file can reuse message model objects defined in another file.

There are two mechanisms that XML Schema provides to reuse message definition files: **import** and **include**. The namespaces of the two files determine which of the **import** or **include** commands is be used:

	Target file has a target namespace	**Target file has notarget namespace**
Parent file has a target namespace	xsd:import	xsd:include[1]
Parent file has notarget namespace	xsd:import	xsd:include

Note: When a target namespace file includes a notarget namespace file, referencing an object in the target file from the parent file causes the object to be present in the namespace of the parent file.

When **import** or **include** are used, global objects from the target file can be used in the parent file. For example, an element in the parent file can be given a complex type defined in the target file.

The namespace of objects in the target file is preserved in the parent file, with the exception noted in the previous table of a target namespace file that includes a notarget namespace file. This exception is sometimes called the chameleon namespace effect.

Chameleon namespaces have limited support when used with the MRM domain. When referenced in the parent file, the objects in the target file are present in the namespace of the parent file, but they are assigned default physical format information. Therefore, physical format information defined in the target file is not available for use in the parent file. Use Chameleon Namespaces in the MRM domain to model XML messages only if physical format information has not changed from the default.

XML Schema provides a variation of xsd:include called xsd:redefine, which is not supported by WebSphere Message Broker. Using xsd:redefine gives a task list error. A Quick Fix is offered to convert occurrences of xsd:redefine into xsd:include when using message definition files.

Related concepts:

"Namespaces in the message model" on page 1201
Use namespaces to qualify message model object names.

"Namespaces with non-XML messages" on page 1206
The use of namespaces by WebSphere Message Broker is not necessarily limited to XML message models.

"Specifying namespaces in the Message Type property" on page 1208
When using the MRM domain, the *Message Type* property is used to specify the name of the message.

Related tasks:

"Applying a Quick Fix to a task list error" on page 2862
During the creation, migration and manipulation of message models, warnings or errors might occur; these are listed in the Problems view of the Broker Application Development perspective. Some of these warnings or errors can be cleared by applying a Quick Fix.

"Creating a message set" on page 2842
Use the New Message Set wizard to create a message set.

"Creating a message definition file" on page 2865
Creating an empty message definition file to contain your message model objects.

Related reference:

"Message definition file properties" on page 5409
The properties of a message definition file.

Message model integrity:

When you create your message model, it is important that it is internally consistent.

To assist with this, whenever you save a message definition file, it is validated as follows:

Logical validation
: This validation ensures that the logical model is correct. For message definition files, this involves ensuring that the rules of XML Schema have been correctly followed.

Physical validation
: This validation ensures that any physical properties that you have specified for your model have been correctly populated and are consistent. There is a set of checks for each of the MRM domain physical formats - CWF, XML, and TDS.

  After model validation has taken place, any errors or warnings are shown in the task list. Double clicking a task list entry opens the file and positions the editor at the object in error. Organize the task list so that errors are shown before warnings. In this way, errors are not hidden. The task list provides a comprehensive filtering capability if you want to hide low priority warnings, or warnings that you are know about and are comfortable with.

  The deployment of a message model in a broker archive (BAR) file is prevented if any errors are present. The presence of warnings alone does

not prevent deployment, but high priority warnings must be reviewed because a model that generates such warnings might be incomplete.

Where task list warnings or errors occur, these are listed in the Problems view of the Broker Application Development perspective. While a majority of these require you to manually investigate and resolve them, a number of warnings and errors that meet specific criteria can be repaired by using a quick fix process.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

"XML Schema" on page 1172
XML Schema is an international standard from the World Wide Web Consortium (W3C) that defines a language for describing the structure of XML documents.

"Message sets overview" on page 1162
A message set is a container for grouping messages and associated message resources (elements, types, groups).

"Message definition files" on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

"Physical formats in the MRM domain"
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM Custom Wire Format" on page 1214
Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not separated by delimiters.

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"MRM TDS format" on page 1221
The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

Related tasks:

"Applying a Quick Fix to a task list error" on page 2862
During the creation, migration and manipulation of message models, warnings or errors might occur; these are listed in the Problems view of the Broker Application Development perspective. Some of these warnings or errors can be cleared by applying a Quick Fix.

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

Related reference:

"Message model task list errors that have a quick fix" on page 6336
You can apply a quick fix to some message modeling task list warnings or errors to correct them.

Physical formats in the MRM domain

Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

If you are using the MRM domain or the IDOC domain, physical format information *must* be provided, as it tells the parser exactly how to parse the message bit stream.

You can think of a message as a packet of data that is sent from one place to another. The sender and receiver of the message will have agreed the structure of the message and what each field in the message means. This is the platform and protocol independent logical structure.

The sender and receiver will have also agreed on the physical representation of the message, how the data is physically laid out on the wire. For example, if you define a message that conveys information about a debit of an individual's bank account, it can be represented in different physical forms (examples are XML, or a fixed structure such as a COBOL copybook). The meaning and data are the same in both cases: only the physical layout has changed.

If you are using the MRM domain, you can model various different physical representations by using named physical formats.

- Use the Custom Wire Format (CWF) physical format to model fixed-format messages from applications that are written in C, COBOL, PL/I and other languages. This support includes the ability to create a message model directly from a C header file or COBOL copybook.
- Use the Tagged Delimited String Format (TDS) physical format to model formatted text messages, perhaps with field content identified by tags or separated by specific delimiters or both. This support is rich enough to model industry standards such as SWIFT, EDIFACT, and X12.
- Use the XML physical format to model XML messages, including those that use XML namespaces. This support includes the ability to create a message model directly from an XML DTD or XML Schema file.

Different physical representations: The following example shows how a simple logical message can have different physical representations.

The logical model defines the structure and order of the message. In the following example, the three fields are simple integers, and C follows B, which follows A:

```
int   A;
int   B;
int   C;
```

- A typical Custom Wire Format representation for this logical message would be 12 bytes of data, with each of `A`, `B` and `C` occupying 4 bytes. Alternatively, perhaps `A` is 4 bytes long, but `B` and `C` are only 2 bytes long. You supply the precise physical information for each field in the message as CWF properties.
- TDS enables several different representations to be modeled. Each integer can be preceded by a tag to identify it and a delimiter to terminate it, as follows:

  ```
  {A_tag:xxxxxxxx;B_tag:xxxxxxxx;C_tag:xxxxxxxx}
  ```

  An alternative might rely on the data being ordered so only the terminating delimiter must be specified, as follows:

  ```
  [xxxxxxxx;xxxxxxxx;xxxxxxxx]
  ```

  You supply the precise identification regime as TDS properties.
- A typical XML representation of this model is as follows:

  ```
  <Msg><A>xxxxxxxx</A><B>xxxxxxxx</B><C>xxxxxxxx</C></Msg>
  ```

where xxxxxxxx is the value of the integer represented as a string (XML deals only with strings). An alternate representation might be:

```
<Msg A="xxxxxxxx" B="xxxxxxxx" C="xxxxxxxx"/>
```

where the values of the integers are stored as XML attributes rather than XML elements. You supply the precise XML rendering for each field in the message as XML properties.

This shows that the logical model is unchanged. It is constant, regardless of the physical representation that you choose to model on top of it, using the physical format support provided by the MRM domain. The MRM parser is able to transform the message from the input physical representation to any number of output representations, based on the physical format layers that you have defined.

Creating physical formats: When you have created your message set, you can create physical formats. You do this using the Message Set Editor. When you next save the `messageSet.mset` file, any new physical formats are added to all the objects in all the message definition files in that message set.

The next time you edit an object in a message definition file, you see the physical formats in the properties hierarchy pane of the Properties tab. If you click a physical format for an object, you are presented with a property sheet where you can enter the information for that physical format for that object.

Not all objects have properties in all physical formats. For example:

- CWF properties apply only to local elements and attributes, and element and attribute references.
- Complex types and groups have only TDS properties.
- Messages have only XML properties.

These differences occur because of the different nature of each physical format, and are explained in more detail in the section for each physical format.

There is no limit to the number of physical formats you can create in a given message set. However there are some recommendations that apply if you want to mix physical formats of different kinds in the same message set.

Physical formats can be deleted if no longer required.

Related concepts:

"Message sets overview" on page 1162
A message set is a container for grouping messages and associated message resources (elements, types, groups).

"Message definition files" on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

"MRM parser and domain" on page 1111
You can use the MRM domain to parse and write a wide range of message formats.

"Which XML parser should you use?" on page 1080
If your messages are general-purpose XML documents, you can use one of the dedicated XML domains (XMLNSC or XMLNS) to parse the message, or you can use the MRM domain to parse the message.

"MRM Custom Wire Format"
Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not separated by delimiters.

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"MRM TDS format" on page 1221
The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

Related tasks:

"Working with a message set" on page 2840
Complete a variety of tasks that are involved in working with a message set.

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

Related reference:

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

MRM Custom Wire Format:

Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not separated by delimiters.

Within a CWF messaging environment, it is not possible to distinguish one element from the next without knowledge of the message structure. To correctly determine the values of individual elements, the following information must be made available to the message parser:

- The order (this is defined in the logical properties)
- The length (can be specified in bytes, characters, or character units)
- The cardinality (that is, the number of repeats)
- The type of data contained in each element (this is partly defined in the logical properties but can be further qualified in the CWF physical format)
- A number of characteristics based upon the logical type of the data

A CWF physical format is typically used to describe messages which are mapped to a C structure, a COBOL copybook, or other programming language data structure definition.

You can add more than one CWF physical format to a message set, but within that message set, each physical format must have a unique name. When parsing a CWF message by using the MRM parser, the physical format name specifies the physical properties that are to be used by the parser.

Adding a CWF physical format to a message set enables you to process input messages and construct output messages in this format. Messages can be transformed between CWF and the other physical representations (for example TDS or XML). While the other physical representations support self-defining elements (that is elements which do not have a definition in the logical model) within the MRM domain, the parsing of a CWF message does not. Consequently, any such self-defining elements are discarded during the output of messages in CWF format.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

Related tasks:

"Working with a message set" on page 2840
Complete a variety of tasks that are involved in working with a message set.

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Adding a Custom Wire Format (CWF)" on page 2848
You can add a Custom Wire Format (CWF) physical format layer to a message set by using the Message Set editor.

"Configuring Custom Wire Format (CWF) properties: Message sets" on page 2849
Configure the Custom Wire Format (CWF) properties of a message set using the Message Set editor.

"Removing a physical format" on page 2858
You can remove a physical format layer from your message set.

Related reference:

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

MRM Custom wire format: Message model integrity:

When you save a message definition file, the definitions that it contains are checked to ensure that they make sense and provide sufficient information about the message. This action is called *model validation*.

The CWF physical format depends on fixed-format data structures. Therefore, most tests that are applied to a CWF message confirm that each fragment of a message - and therefore, the message as a whole - has a well-defined length. Therefore, these tests examine properties such as `Length`, `Length Reference` and `Length Units`.

Typically, one or other of `Length` and `Length Reference` must be set. If `Length Reference` is set, it must refer to an element that is of simple type integer and that appears earlier in the message than the current item.

Tests other than these tend to be both simple and obvious so that, for example, the message set property `First Day of Week` must be the name of a day in the week.

The fact that CWF relies on fixed-format data structures also imposes some limitations on the messages that can be represented:

- CWF cannot represent a message that includes the use of XML Schema wild cards; this is a consequence of its inability to handle undefined content.
- CWF cannot represent a message that includes recursive definitions.
- CWF cannot represent a message that includes the use of substitution groups, because there is no way to recognize the substituted element.

Related concepts:

"Message model integrity" on page 1210
When you create your message model, it is important that it is internally consistent.

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

Related tasks:

"Working with a message set" on page 2840
Complete a variety of tasks that are involved in working with a message set.

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Adding a Custom Wire Format (CWF)" on page 2848
You can add a Custom Wire Format (CWF) physical format layer to a message set by using the Message Set editor.

"Configuring Custom Wire Format (CWF) properties: Message sets" on page 2849
Configure the Custom Wire Format (CWF) properties of a message set using the Message Set editor.

"Removing a physical format" on page 2858
You can remove a physical format layer from your message set.

Related reference:

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

MRM Custom wire format: NULL handling:

CWF supports the handling of explicit NULL values within messages, if the logical nillable property of the element is set.

An explicit null is identified by a specific value that identifies an element as being null.

The `Boolean Null Value` can be specified at the message set level, and applies to the Boolean elements of all messages that are defined in that message set. The null value of all other element types can be specified individually for each element.

CWF supports the representation of null values by using the `Encoding Null` and `Encoding Null Value` element properties. Together, this information controls how null values are handled by the MRM parser.

The `Encoding Null` property can be set to one of four values:

NullLogicalValue

The `Encoding Null Value` property is interpreted as a logical value. Therefore, if its value is set to 0, for example, both `0` and `0.00` are interpreted as null values.

NullLiteralValue

The `Encoding Null Value` property is interpreted as a string value. Therefore, the value of the element in the bit stream must match exactly the value that is specified to be interpreted as a null value.

NullPadFill

Used for fixed-length elements. On output, any element with a null value is padded to the appropriate length with the specified padding character.

NullLiteralFill

The `Encoding Null Value` property is interpreted as a single character string value. Therefore, each character of the value of the element in the bit stream must match exactly the character value specified to be interpreted as a null value.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

Related tasks:

"Working with a message set" on page 2840
Complete a variety of tasks that are involved in working with a message set.

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Adding a Custom Wire Format (CWF)" on page 2848
You can add a Custom Wire Format (CWF) physical format layer to a message set by using the Message Set editor.

"Configuring Custom Wire Format (CWF) properties: Message sets" on page 2849
Configure the Custom Wire Format (CWF) properties of a message set using the Message Set editor.

"Removing a physical format" on page 2858
You can remove a physical format layer from your message set.

Related reference:

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

MRM Custom wire format: Multipart messages:

The Custom Wire Format (CWF) supports both the Message Identity technique and the Message Path technique of identifying embedded messages within a multipart message.

Alternatively, you can resolve an embedded message by using ESQL or Java to identify the message. The first message that you reference in this way is assumed to be the selected message. This technique works in an identical manner to `unresolved choice` handling.

Related concepts:

"Multipart messages" on page 1191
A *multipart message* contains one or more other messages within its structure. The contained message is sometimes referred to as an *embedded message*.

"Identifying an embedded message by using a Message Identity" on page 1193
You can identify an embedded message by using the Message Identity. This technique is used by the MRM domain, and replaces the use of the Message Key.

"Identifying an embedded message by using a Message Path" on page 1196
The Message Path technique for identifying embedded messages is useful when the multipart message contains no information about the identity of an embedded message.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

"Specifying namespaces in the Message Type property" on page 1208
When using the MRM domain, the *Message Type* property is used to specify the name of the message.

Related tasks:

"Creating a multipart message" on page 2919
A multipart message occurs when you embed a message in another message.

"Accessing embedded messages in the MRM domain" on page 2594
If you have defined a multipart message, you have at least one message embedded within another. Within the overall complex type that represents the outer messages, you can model the inner message in two ways.

Related reference:

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

MRM Custom wire format: Data Conversion:

The Custom Wire Format supports the conversion of data to a different code page (for string simple types) or encoding (for numeric simple types), or both.

A message set contains properties to enable the character (CCSID) and numeric encoding (`Byte Order / Float Format`) information to be specified. If you generate a message dictionary for deployment to a WebSphere Message Broker, this information can be overridden by using the appropriate fields of the WebSphere MQ message header, or other transport header.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"Importing from other model representations to create message definitions" on page 1254
You can add message definitions to your message set by importing application message formats that already exist.

Related tasks:

"Removing a physical format" on page 2858
You can remove a physical format layer from your message set.

"Working with a message set" on page 2840
Complete a variety of tasks that are involved in working with a message set.

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Adding a Custom Wire Format (CWF)" on page 2848
You can add a Custom Wire Format (CWF) physical format layer to a message set by using the Message Set editor.

"Configuring Custom Wire Format (CWF) properties: Message sets" on page 2849
Configure the Custom Wire Format (CWF) properties of a message set using the Message Set editor.

Related reference:

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

MRM Custom wire format: Relationship to the logical model:

Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Composition: A CWF message is always written with the elements in the sequence that is specified in the logical message model definition. However, you do not always have to specify the ESQL or Java that builds the elements in that sequence. The following rules for coding ESQL are given for each value of the type `Composition` property.

Sequence

You must build the output message to match the sequence of the elements or groups in the message. You can normally do this using ESQL SET statements to assign a value to each element or type. The first SET statement sets the value of the first element or type in the message, the second SET statement sets the value for the second element or type, and so on. You can vary this sequence of statements by using ESQL ATTACH, CREATE, and MOVE statements.

If the elements or types have default values, and you do not build the message in the correct sequence, those elements that are built out of sequence contain their default values, not the values that you set. This is because elements that are built out of sequence are assumed to be self-defining and, for CWF, these are discarded when the message is written to the bit stream.

Ordered Set

You must build the output message to match the sequence of the elements in the message. You can normally do this using ESQL SET statements to assign a value to each element. The first SET statement sets the value of the first element in the message, the next SET statement sets the value for the second element, and so on. You can vary this sequence of statements by using ESQL ATTACH, CREATE, and MOVE statements.

If the elements have default values, and you do not build the message in the correct sequence, those elements that are built out of sequence contain their default values, not the values that you set. This is because elements

that are built out of sequence are assumed to be self-defining and, for CWF, these are discarded when the message is written to the bit stream.

Unordered Set
You can build elements of the output message in any sequence. On output, the elements are written in the order that is specified in the logical message model definition.

All You can build elements of the output message in any sequence. Each element must be present only once (that is, it must not repeat). On output, the elements are written in the order that is specified in the logical message model definition.

Choice
A choice cannot be resolved purely from the data. The receiving program must interpret the data and decide which option of the choice the message instance contains. This process is known as *unresolved choice* handling. The first reference in the application to any one of the choice elements resolves the choice to the option that contains that element.

Message
Mechanisms for the resolution of embedded messages are discussed in the "MRM Custom wire format: Multipart messages" on page 1217 topic.

Content validation: CWF is a fixed format, and all elements must be present in a message. Therefore, content validation is ignored. On output, all elements must be set explicitly (for example, by using ESQL SET), set implicitly (by using a tree copy function), or must have a default value defined.

Default values: On output of a CWF message in the MRM domain, any element, or occurrence of an element for which a value has not been set (either explicitly or implicitly), inherits the specified default value of the element. If no default value has been specified then an exception is thrown.

Min Occurs and Max Occurs: The logical properties `Min Occurs` and `Max Occurs` specify the permitted number of occurrences of an element, or group, in a message. These properties are used when parsing and writing messages, and when validating the content of a message.

In CWF, `Max Occurs` occurrences are expected when parsing, and `Max Occurs` occurrences are produced when writing. Default values are used for missing elements, and any excess elements are discarded.

- A varying number of occurrences (`Min Occurs` <> `Max Occurs`) is ignored, `Max Occurs` is assumed.
- Optional occurrence (`Min Occurs` = 0) is ignored, `Max Occurs` is assumed.
- Always absent (`Max Occurs` = 0) is allowed.
- An unbounded number of occurrences (`Max Occurs` = -1) is allowed if the element or group is the last child in its parent group, and the group is terminated by the end of the message bit stream. On writing, the writer writes all occurrences in the message tree, if this number is less than `Min Occurs`, additional default values are written.

These rules arise because, in a CWF message format, there are no tags or other markup that can be used to determine the end of a variable number of repeats.

However this behavior is overridden if the CWF property `Repeat Reference` is set, which indicates that the number of occurrences is given instead by an integer element that occurs earlier in the message. In this case `Max Occurs` is ignored.

When validating, `Min Occurs` and `Max Occurs` are both used to check that the content of the message tree matches the model.

Simple types – lists and unions: Lists and unions are XML-specific concepts. An element or attribute of a simple type that is a list or a union causes a task list warning if a CWF physical format is present in the message set. The user can choose whether to make this an error, warning, or information by editing the Validation preferences. The dictionary generator omits messages defined to contain such elements or attributes from the CWF section of the dictionary.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Multipart messages" on page 1217
The Custom Wire Format (CWF) supports both the Message Identity technique and the Message Path technique of identifying embedded messages within a multipart message.

Related tasks:

"Working with a message set" on page 2840
Complete a variety of tasks that are involved in working with a message set.

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Adding a Custom Wire Format (CWF)" on page 2848
You can add a Custom Wire Format (CWF) physical format layer to a message set by using the Message Set editor.

"Configuring Custom Wire Format (CWF) properties: Message sets" on page 2849
Configure the Custom Wire Format (CWF) properties of a message set using the Message Set editor.

"Removing a physical format" on page 2858
You can remove a physical format layer from your message set.

Related reference:

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

MRM TDS format:

The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

The TDS physical format is designed to model messages that consist of text strings, but it can also handle binary data. Examples of TDS messages are those that conform to the ACORD AL3, EDIFACT, HL7, SWIFT, or X12 standards. The TDS physical format allows a high degree of flexibility when defining message formats, and is not restricted to modeling specific industry standards; therefore, you can use the TDS format to model your own messages.

TDS message characteristics: There are a number of features of text string messages that are common to many formats. This is an overview of the main features that are supported by the TDS physical format:

Tags The text strings in the message can have a *tag* or a label preceding the data value. The tag is a string that uniquely identifies the data value. The TDS format allows you to associate a tag with each element when you define the element.

Delimiters and tagged data separators
The message can contain various special characters or strings in addition to the tags and text string data values. The TDS format supports a number of different types of special characters or strings.

Some messages have a special character or string that separates each data value from the next. In the TDS format this is a known as a *delimiter*.

In formats that have a tag before each data value, the tag can be separated from its data value by a special character or string. In the TDS format this is known as a *tag data separator*.

Group indicators and terminators
A message can be split into a number of substructures in a similar manner to a COBOL or C structure. You can model each of these substructures separately by defining groups, complex types, or elements for each one.

A substructure can have a special character or string that indicates its start within the data. This is known in the TDS format as a *group indicator*.

A substructure can also have a special character or string that indicates its end in the data. In the TDS format, this is known as a *group terminator*.

A group indicator and group terminator can also be defined for the whole message. Group indicators and group terminators are optional for the message and each substructure.

Fixed-length strings
Some text strings within a message can be of fixed length; therefore, a delimiter between each data value is not necessary. This is supported by the TDS format.

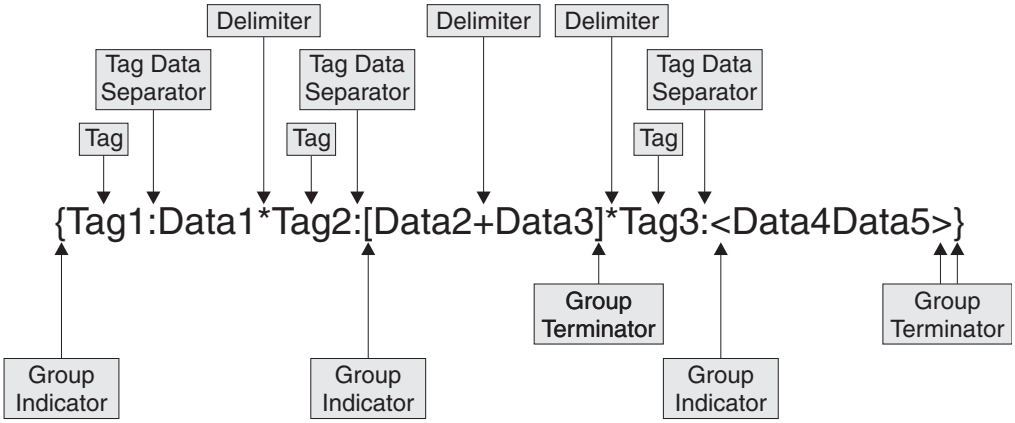
Fixed-length tags
Some tags can be defined as fixed length; therefore, a tag data separator is not necessary.

Separation types
The TDS property that controls the way text strings are separated is `Data Element Separation`. It has several options, for example, whether tags are used, whether strings lengths are fixed or variable, and what types of text strings are permitted.

The substructures within a message can use different types of data element separation and use different special characters. Therefore the TDS format allows you to define different types of data element separation and special characters for each complex type within the message.

Regular Expressions
If you choose the Use data pattern option for `Data Element Separation`, you can use regular expressions to identify parts of the message data to be assigned to sub fields. This is done by setting the regular expression in the `Data Pattern` property.

The following diagram shows an example data message with each of its components labeled.



- At the top level, each data value has a *tag* associated with it, each tag is separated from its data value by using a *tag data separator* of colon (:), and the data values are separated from each other using the asterisk *delimiter* (*).
- The *group indicator* for the message is the left brace ({) and the *group terminator* is the right brace (}).
- The data values `Data2` and `Data3` are in a substructure in which there are no tags, and each data element is separated from the next using the plus delimiter (+). The group indicator for this substructure is the left bracket ([) and the group terminator is the right bracket (]).
- The data values `Data4` and `Data5` are in a substructure in which the values are fixed length, and are therefore not separated by a delimiter. The group indicator for this substructure is the less than symbol (<) and the group terminator is the greater than symbol (>).

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM TDS format: Data element separation" on page 1225
Data element separation defines how a TDS message is to be parsed.

"Message model objects: elements" on page 1176
An *element* is a named piece of information (a field) within a message, with a meaning that is agreed by the applications that create and process the message.

"MRM TDS format: Determining the length of simple data values" on page 1224
The TDS format supports two categories of simple data types: textual and non-textual.

"Message model objects: types" on page 1177
Types describe the data content of elements.

Related tasks:

"Adding a TDS physical format" on page 2851
Add a Tagged/Delimited String (TDS) physical format layer to a message set using the Message Set editor.

"Configuring TDS properties: Message model objects" on page 2914
You can configure the Tagged/Delimited String (TDS) properties of a message model object.

"Removing a physical format" on page 2858
You can remove a physical format layer from your message set.

Related reference:

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

MRM TDS format: Determining the length of simple data values:

The TDS format supports two categories of simple data types: textual and non-textual.

The Physical Type of an element determines whether it is categorized as textual or non-textual.

Textual data

Physical Type is either Text or TLOG Specific. For textual data, the Data Element Separation of the parent complex type or group determines how the length of the data is determined. See "MRM TDS format: Data element separation" on page 1225 and its subtopics.

Non-textual data

Elements of all other Physical Types are non-textual. The length of non-textual data is determined by the Physical Type of the element. For non-textual data, the Data Element Separation property of the parent complex type or group does not determine the length, unless Data Element Separation is Use Data Pattern. See "MRM TDS format: Data pattern separation types" on page 1237 for more information.

The following table describes how the length of data is determined for each Physical Type.

Physical Type	Determination of Length
Text TLOG Specific	The Data Element Separation of the parent complex type or group determines how the length of the data is determined.
External Decimal Integer Packed Decimal Float Time Seconds Time Milliseconds	Uses the value of the Length property of the element. If Physical Type is Time Seconds, the Length property is set to 4. If Physical Type is Time Milliseconds, the Length property is set to 8. In neither case can this value be changed.
Length Encoded String 1 Length Encoded String 2	Uses the encoded length value in the data.
Null Terminated String	Uses the null terminator at the end of the data.
Binary	Uses the value of the Length Reference or Length property of the element.

Related concepts:

"Multipart messages" on page 1191
A *multipart message* contains one or more other messages within its structure. The contained message is sometimes referred to as an *embedded message*.

"MRM TDS format" on page 1221
The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

"MRM TDS format: Fixed-length separation types" on page 1227
For fixed-length separation types, each data value is a fixed length.

"MRM TDS format: Tagged separation types" on page 1228
For tagged separation types, each data value is preceded by a tag that is specified as an element property.

"MRM TDS format: Delimited separation types" on page 1232
For delimited separation types, a delimiter is used to separate data fields, but there are no tags present. The data fields must be given in the correct order in the bit stream and elements cannot be omitted from the middle of the bit stream.

"MRM TDS format: Data pattern separation types" on page 1237
For a data pattern separation type, each data value is matched with a regular expression that is specified as a property of each element.

"MRM TDS format: Message model integrity" on page 1239
When you use the TDS wire format, you must conform to a number of rules that apply to the setting of values of properties. This is necessary to avoid any discrepancies when processing a message within the specified model.

Related tasks:

"Adding a TDS physical format" on page 2851
Add a Tagged/Delimited String (TDS) physical format layer to a message set using the Message Set editor.

"Configuring TDS properties: Message sets" on page 2852
Configure the Tagged/Delimited String (TDS) format properties of a message set using the Message Set editor.

"Configuring TDS properties: Message model objects" on page 2914
You can configure the Tagged/Delimited String (TDS) properties of a message model object.

Related reference:

"TDS message model integrity" on page 6295
When you use the TDS wire format, you must conform to a number of rules that apply to the setting of values of properties. These rules are checked whenever the project is saved.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

MRM TDS format: Data element separation:

Data element separation defines how a TDS message is to be parsed.

Data element separation defines which method of identifying data elements is to be used and how the data elements are constructed. The different methods vary from full flexibility to fixed format, depending on how they are defined.

The four main types of data element separation are:

Fixed-length types
Fixed-length types are dependent on each element having a length. See "MRM TDS format: Fixed-length separation types" on page 1227.

Tagged separation types
Tagged separation types are dependent on each element having tag prefix to identify it. See "MRM TDS format: Tagged separation types" on page 1228.

Delimited separation types

Delimited separation types use delimiters to identify the end of one data elements and the beginning of the next. See "MRM TDS format: Delimited separation types" on page 1232.

Data pattern types

Data pattern types use a regular expression to identify each element. See "MRM TDS format: Data pattern separation types" on page 1237.

There is a fifth category, which is different from the four described earlier in this topic:

Undefined separation types

Undefined separation types contain no data elements. They are applicable to embedded messages only. They use none of the TDS type-specific parameters other than `Data Element Separation`. See "Multipart messages" on page 1191.

Related concepts:

"Multipart messages" on page 1191
A *multipart message* contains one or more other messages within its structure. The contained message is sometimes referred to as an *embedded message*.

"MRM TDS format: Fixed-length separation types" on page 1227
For fixed-length separation types, each data value is a fixed length.

"MRM TDS format: Tagged separation types" on page 1228
For tagged separation types, each data value is preceded by a tag that is specified as an element property.

"MRM TDS format: Delimited separation types" on page 1232
For delimited separation types, a delimiter is used to separate data fields, but there are no tags present. The data fields must be given in the correct order in the bit stream and elements cannot be omitted from the middle of the bit stream.

"MRM TDS format: Data pattern separation types" on page 1237
For a data pattern separation type, each data value is matched with a regular expression that is specified as a property of each element.

"MRM TDS format: Message model integrity" on page 1239
When you use the TDS wire format, you must conform to a number of rules that apply to the setting of values of properties. This is necessary to avoid any discrepancies when processing a message within the specified model.

Related tasks:

"Adding a TDS physical format" on page 2851
Add a Tagged/Delimited String (TDS) physical format layer to a message set using the Message Set editor.

"Configuring TDS properties: Message sets" on page 2852
Configure the Tagged/Delimited String (TDS) format properties of a message set using the Message Set editor.

"Configuring TDS properties: Message model objects" on page 2914
You can configure the Tagged/Delimited String (TDS) properties of a message model object.

Related reference:

"TDS message model integrity" on page 6295
When you use the TDS wire format, you must conform to a number of rules that apply to the setting of values of properties. These rules are checked whenever the project is saved.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

MRM TDS format: Fixed-length separation types:

For fixed-length separation types, each data value is a fixed length.

For fixed-length data element separation types, all textual elements have a length or length reference, and are padded out to their full length in the bit stream. No tags or delimiters are used, and each data value directly follows the preceding data value.

For example:

```
data1data200data30
```

The first element is length 5, the second is length 7, and the third is length 6. The padding character is "0".

For non-textual elements, the length is determined by the `Physical Type` of the element. See "MRM TDS format: Determining the length of simple data values" on page 1224.

Fixed-length type: In fixed-length type, all textual elements must have a length or length reference, and must be written out to that full length. The elements must be presented in the correct order, and all elements must be written in the bit stream. This includes all repeats of any repeating element (that is, the `Maximum Occurrences` must be written out for each element).

For non-textual elements, the length is determined by the `Physical Type` of the element. See "MRM TDS format: Determining the length of simple data values" on page 1224.

For example:

```
data10data2data2data2data300
```

The first element is length 6, the second is length 5, and repeats three times, and the third element is length 7. The padding character is "0".

Applicable parameters: The main parameters for this format are the `Length` or `Length Reference` of the element. All fields must be padded out to their full length for the bit stream to be correctly specified to the parser.

Tag and delimiter parameters are ignored. Group indicators and terminators are observed, because they are of fixed length.

Default values are required for each field that might not be set, because then every field can be produced as output, even if it is not filled with data from the message.

Fixed-length AL3 type (Deprecated): This separation type has been deprecated. ACORD AL3 support will be provided by a different method in a future release, at which time this separation type will be removed from service.

Fixed-length AL3 types are similar to fixed-length types, but follow extra rules that are specified by the ACORD AL3 format regarding truncation and missing elements. If elements are missing from the end of an AL3 type, they can be truncated. They cannot be omitted from the middle of a bit stream. If a field is

missing from the middle of the bit stream, that field is produced for output as the appropriate length string of the "?" character.

Applicable parameters: The main parameters for this format are the `Length` or `Length Reference` of the element. All fields must be padded out to their full length for the bit stream to be correctly specified to a parser.

Tag and delimiter parameters are ignored. `Group Indicators` and `Terminators` are observed, because they are of fixed length.

Related concepts:

"MRM TDS format: Data element separation" on page 1225
Data element separation defines how a TDS message is to be parsed.

"MRM TDS format: Tagged separation types"
For tagged separation types, each data value is preceded by a tag that is specified as an element property.

"MRM TDS format: Delimited separation types" on page 1232
For delimited separation types, a delimiter is used to separate data fields, but there are no tags present. The data fields must be given in the correct order in the bit stream and elements cannot be omitted from the middle of the bit stream.

"MRM TDS format: Data pattern separation types" on page 1237
For a data pattern separation type, each data value is matched with a regular expression that is specified as a property of each element.

"MRM TDS format: Message model integrity" on page 1239
When you use the TDS wire format, you must conform to a number of rules that apply to the setting of values of properties. This is necessary to avoid any discrepancies when processing a message within the specified model.

Related tasks:

"Adding a TDS physical format" on page 2851
Add a Tagged/Delimited String (TDS) physical format layer to a message set using the Message Set editor.

"Configuring TDS properties: Message sets" on page 2852
Configure the Tagged/Delimited String (TDS) format properties of a message set using the Message Set editor.

"Configuring TDS properties: Message model objects" on page 2914
You can configure the Tagged/Delimited String (TDS) properties of a message model object.

Related reference:

"TDS message model integrity" on page 6295
When you use the TDS wire format, you must conform to a number of rules that apply to the setting of values of properties. These rules are checked whenever the project is saved.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

MRM TDS format: Tagged separation types:

For tagged separation types, each data value is preceded by a tag that is specified as an element property.

The `Tag Data Separator`, or specific `Length of Tag` parameter is used to determine where the tag ends and the data starts. Different methods are used by each separation type to determine the end of the data.

After considering these two parameters, this topic describes the following supported tagged separation types:

- "Tagged Delimited separation"
- "Tagged Fixed Length separation" on page 1230
- "Tagged Encoded Length separation" on page 1230

Tagged separation is a flexible format. The elements do not have to occur in a specific order. They do not all need to be present, and can be absent from any point in the message.

Tag Data Separator and Tag Lengths: Either `Tag Data Separator` and `Length of Tag` are used by all tagged separation types. But only one of these parameters can be set at the same time.

The point at which a tag ends and data starts can be determined by one of two methods. If the `Tag Data Separator` is set, then this character indicates where the data ends. For example, the string might be:

```
tag1:data1
```

where `Tag Data Separator` is :

However if the `Tag Data Separator` is not set and the `Length of Tag` field *is* set, then the tag is the specified length, and is immediately followed by the data. No separating character is required. For example, the string might be:

```
tag1data1
```

where `Length of Tag` is 4

Tagged Delimited separation: Tagged Delimited separation is a flexible format. Elements are separated by a predefined delimiter. The textual elements are not of specific lengths. For non-textual elements, the length is determined by the `Physical Type` of the element. See "MRM TDS format: Determining the length of simple data values" on page 1224.

Applicable parameters: These parameters are used:

- `Group Indicator` indicates the start of a group or complex type.
- `Group Terminator` indicates the end of a group or complex type.
- `Delimiter` separates the data elements within a group or complex type.
- `Tag` for each element, indicates the tag needed to precede the data in that field.
- Either `Tag Data Separator` or `Tag Length` as described earlier in this topic.

Examples: If `Tag Data Separator` is set to :

```
{tag1:data1*tag2222222:data2*tag333:data3}
```

where:

- `Group Indicator` is {.
- `Group Terminator` is }.
- `Delimiter` is *.
- `Tag` defined for each element, is tag1 (for data1), tag2222222 (for data2), and tag333 (for data3).

or, for example, if `Length of Tag` is set to 5

```
{tag11data1*tag22data2*tag33data3}
```

where parameters are as above, except:

- `Tag`, defined for each element (fixed at five characters), is tag11 (for data1), tag22 (for data2), and tag33 (for data3).

Tagged Fixed Length separation: Although Tagged Fixed Length separation is a flexible format, the data must be a specific length. This means that a delimiter is not needed to determine the end of each element.

Applicable parameters: These parameters are used:

- `Group Indicator` indicates the start of a group or complex type.
- `Group Terminator` indicates the end of a group or complex type.
- `Tag` for each element, indicates the tag needed to precede the data in that field.
- For each textual element, `Length` or `Length Reference` indicates the length of the data (this value does *not* include the length of the tag). For non-textual elements, the length is determined by the `Physical Type` of the element. See "MRM TDS format: Determining the length of simple data values" on page 1224.
- Either `Tag Data Separator` or `Tag Length` as described earlier in this topic.

Examples: If `Tag Data Separator` is set to :

```
{tag1:data1tag22222222:data2000tag333:data300}
```

where:

- `Group Indicator` is {.
- `Group Terminator` is }.
- `Delimiter` is *.
- `Tag`, defined for each element, is tag1 (for data1), tag22222222 (for data2000), and tag333 (for data300).
- `Length`, defined for each element, is 5 (data1), 8 (data2000), and 7 (data300).

or, for example, if `Length of Tag` is set to 5

```
{tag11data1tag22data2000tag33data300}
```

where parameters are as above, except:

- `Tag`, defined for each element (fixed at five characters), is tag11 (data1), tag22 (data2000), and tag33 (data300).

Tagged Encoded Length separation: This method has both a tag and a length field before the data. The length field indicates to the parser the length of the data following it.

The length of this length field is itself defined in the `Length of Encoded Length` parameter. Extra lengths to be added in this, such as the length of the field itself, is set in the `Extra Chars in Encoded Length` parameter.

Only textual elements and elements with a Binary logical and physical type are supported within a Tagged Encoded Length separation.

These examples show how the values set in these parameters are applied:

- `tagA007dataAAAtagB006dataBBtagC009dataCCCCC`

  If `Length of Tag` is 4, `Length of Encoded Length` is 3, `Extra Chars in Encoded Length` is 0, then in this bit stream, `TagA` is followed by the three character long length field. This indicates that the following data (`dataAAA`) is seven characters long. The next field, `tagB` is then considered, and so on.

- `tagA012dataAAAAAtagB010dataBBBtagC016dataCCCCCCCCC`

  If `Length of Tag` is 4, `Length of Encoded Length` is 3, `Extra Chars in Encoded Length` is 3, then in this bit stream, `TagA` is followed by the three-character length field. This indicates that the following data, plus extra characters, is 12 characters long: length of the length field (3) + length of data (9) = 12. Therefore the length of the actual data is only 12-3 = 9. The next field, `tagB` is then considered, and so on. In each case the length given in the bit stream is 3 greater than the actual length of the data.

Applicable parameters: These parameters are used:

- `Group Indicator` indicates the start of a group or complex type.
- `Group Terminator` indicates the end of a group or complex type.
- `Tag` for each element, indicates the tag needed to precede the data in that field.
- `Length of Encoded Length` indicates the length of the length field in the bit stream.
- `Extra Chars in Encoded Length` indicates how many extra characters should be included in calculating the value for the length field in the bit stream.
- Either `Tag Data Separator` or `Tag Length` as described earlier in this topic.

Examples: If `Tag Data Separator` is set to :

```
{tag1111:008data1tag222222222:010data2AAtag3333:009data3A}
```

where:

- `Group Indicator` is {
- `Group Terminator` is }
- `Length of Encoded Length` is 3
- `Extra Chars in Encoded Length` is 3
- `Tag`, defined for each element, is tag1111, tag222222222, tag3333, and so on

or, for example, if `Length of Tag` is set to 5

```
{tag11008data1tag22010data2AAtag33009data3A}
```

where parameters are as above, except:

- `Tag`, defined for each element (fixed at five characters), is tag11, tag22, tag33, and so on

Related concepts:

"MRM TDS format: Data element separation" on page 1225
Data element separation defines how a TDS message is to be parsed.

"MRM TDS format: Fixed-length separation types" on page 1227
For fixed-length separation types, each data value is a fixed length.

"MRM TDS format: Delimited separation types" on page 1232
For delimited separation types, a delimiter is used to separate data fields, but there are no tags present. The data fields must be given in the correct order in the bit stream and elements cannot be omitted from the middle of the bit stream.

"MRM TDS format: Data pattern separation types" on page 1237
For a data pattern separation type, each data value is matched with a regular expression that is specified as a property of each element.

"MRM TDS format: Message model integrity" on page 1239
When you use the TDS wire format, you must conform to a number of rules that apply to the setting of values of properties. This is necessary to avoid any discrepancies when processing a message within the specified model.

Related tasks:

"Adding a TDS physical format" on page 2851
Add a Tagged/Delimited String (TDS) physical format layer to a message set using the Message Set editor.

"Configuring TDS properties: Message sets" on page 2852
Configure the Tagged/Delimited String (TDS) format properties of a message set using the Message Set editor.

"Configuring TDS properties: Message model objects" on page 2914
You can configure the Tagged/Delimited String (TDS) properties of a message model object.

Related reference:

"TDS message model integrity" on page 6295
When you use the TDS wire format, you must conform to a number of rules that apply to the setting of values of properties. These rules are checked whenever the project is saved.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

MRM TDS format: Delimited separation types:

For delimited separation types, a delimiter is used to separate data fields, but there are no tags present. The data fields must be given in the correct order in the bit stream and elements cannot be omitted from the middle of the bit stream.

The All Elements Delimited separation type means that data fields are delimited by a pre-specified character or string. In this example, four data fields are separated by an asterisk (*) delimiter:

```
data1*data2*data3*data4
```

Delimited separation types are restrictive in the ordering and presence of elements:

- The elements must be given in the order specified.
- No element can be omitted in the middle of a group or complex type, because the parser cannot determine this from the resulting bit stream.
- Elements can sometimes be absent from the end of a complex type or group.

After considering "Delimiter suppression and truncation rules," this topic describes the following delimited separation types:

- "All Elements Delimited" on page 1233
- "Variable Length Elements Delimited" on page 1234

Delimiter suppression and truncation rules:

- Elements cannot be omitted from the middle of a group or complex type. An absent element results in the inclusion of a zero-length string.

  For example, with all elements present, the string might be:

  ```
  data1*data2*data3*data4
  ```

  where `Delimiter` is `*`

  If `data2` is missing, the string would read:

  ```
  data1**data3*data4
  ```

- It is possible to suppress the delimiters at the end of a string for absent elements. The Suppress Absent Element Delimiter property determines whether this is done. If this property is set to End of Type, this can be done (with one exception, shown later in this section).

  In this case, for the example with data3 and data4 missing, the string would read:

  ```
  data1*data2
  ```

  That is, the delimiters have been suppressed from the end of this group or complex type.
- If the Suppress Absent Element Delimiter property is set to Never, delimiter suppression never takes place. The string would read:

  ```
  data1*data2**
  ```

  That is, the delimiters must be present to indicate absent (zero-length) elements.

  An exception to the above rule occurs in the case where the same delimiters are used at multiple levels in the model.

  For example, you have a complex type or group with delimiter * and this contains an element of another complex type (indicated by the element3 prefix on data fields in the following example), which also has delimiter *. If both types use a delimited separation type, with all elements present, you might have:

  ```
  data1*data2*element3Data1*element3Data2*element3Data3*data4
  ```

  If element3Data2 and element3Data3 are missing, and the delimiters are suppressed, it is not possible for the parser to determine which elements are missing.

  Therefore, in this case, you must override the Suppress Absent Element Delimiter property, and write out all the delimiters to clearly define the message to the parser. Therefore, the string must be:

  ```
  data1*data2*element3Data1***data4
  ```

  This restriction also applies where Group Indicators and Group Terminators use the same character strings as delimiters; otherwise, the bit stream is not clear to the parser.

All Elements Delimited: In an All Elements Delimited separation type, all elements are separated by a delimiter; for example:

```
data1*data2*data3*data4*data5
```

where Delimiter is *.

An All Elements Delimited separation type does not use tags or their associated parameters.

For textual elements, the length is determined by the delimiter, and the Length property is ignored unless the Observe Element Length property is set.

For non-textual elements, the length is determined by the Physical Type of the element. See "MRM TDS format: Determining the length of simple data values" on page 1224.

Applicable properties: These properties are used:

- Group Indicator indicates the start of a group or complex type.
- Group Terminator indicates the end of a group or complex type.

- `Delimiter` indicates the separator between the data elements within a group or complex type.
- `Suppress Absent Element Delimiters` indicates whether delimiter suppression is permitted.

For example:

```
{data1*data22222*data3}
```

where:

- `Group Indicator` is {
- `Group Terminator` is }
- `Delimiter` is *

Repeating element rules: If an element must be repeated when the separation type is All Elements Delimited, the `Repeating Element Delimiter` (`RED`), is used to separate the repeated elements.

For example if `data2` repeats five times:

```
data1*data2:data2:data2:data2:data2*data3*data4
```

where:

- `Delimiter` is *
- `Repeating Element Delimiter` is:

If the `Suppress Absent Element Delimiters` property is set to End of Type, you can use delimiter suppression. Therefore, if only the first `data2` element was present in the previous example, the bit stream reads:

```
data1*data2*data3*data4
```

However, if the `Suppress Absent Element Delimiters` property is set to Never, the bit stream reads:

```
data1*data2::::*data3*data4
```

If `Delimiter` and `RED` match, two delimiters are output to indicate that the repeat is ending. Therefore, if the delimiter and `RED` are *, the bit stream reads:

```
data1*data2**data3*data4
```

Variable Length Elements Delimited: In a complex type with Variable Length Elements Delimited separation, some elements are determined by their length, and other elements are delimited. This combination of a delimited and a fixed-length format follows rules that are associated with both formats. Lengths can be given and used, but they are not mandatory.

- If a length is present for a textual element, it is used, and a delimiter is not needed to terminate that element. The element must be padded to the correct length, and cannot exceed that length.
- If no length is given for a textual element, the delimiter is required.
- For non-textual elements, the length is determined by the `Physical Type` of the element. See "MRM TDS format: Determining the length of simple data values" on page 1224.

A complex type with Variable Length Elements Delimited separation that contains only variable length elements resembles a complex type with All Elements Delimited separation. If it contains only fixed-length elements, it resembles a Fixed Length type.

For example:

```
data1*data2*data3*data4000data5
```

where:

- `Delimiter` is *
- `data4` has a length of 8

Applicable properties: The following properties are used:

- `Group Indicator` indicates the start of a group or complex type.
- `Group Terminator` indicates the end of a group or complex type.
- `Delimiter` indicates the separator between the data elements within a group or complex type.
- `Suppress Absent Element Delimiters` indicates whether delimiter suppression is permitted.
- (Optionally) `Length` or `Length Reference` indicates the length of a textual element. If a textual element has a length, this length is used. Because the length of this element is known, it is not necessary to output a delimiter after it. If the length is not known, a delimiter is required. A delimiter is never required for a non-textual element.

In this example, the fourth field (containing `data4`) is of fixed length 8 and its padding character is `0`:

```
{data1*data22222*data3*data4000data5}
```

where:

- `Group Indicator` is {
- `Group Terminator` is }
- `Delimiter` is *

Repeating element rules: The action of a repeating element in a Variable Length Elements Delimited environment is dependent on the minimum and maximum number of repeats and whether the element has a length.

Delimited Element Repeating: If a delimited element (that is, an element with no length) is repeated, a `Repeating Element Delimiter` (`RED`) is required and the rules for All Elements Delimited are followed. A delimiter is therefore required after the last repeat. Delimiter suppression of this repeat can also occur.

For example, if `data2` is repeating:

```
data1*data2:data2:data2:data2:data2:data2*data3*data4000data5
```

where:

- `Delimiter` is *
- `Repeating Element Delimiter` is :
- `data4` has a fixed length of 8

If the Suppress Absent Element Delimiters field is set to End of Type, you can use delimiter suppression.

If in the above example only the first data2 is present:

```
data1*data2*data3*data4000data5
```

However, if Suppress Absent Element Delimiters is set to Never, the bit stream reads:

```
data1*data2:::::*data3*data4000data5
```

If the delimiter and RED match, two delimiters are output to indicate that the repeat is ending. So if the delimiter and RED are both *, the bit stream reads:

```
data1*data2**data3*data4
```

This also applies for a non-fixed length complex type or group inside a Variable Length Elements Delimited environment.

Fixed Length Element Repeating: If an element with a defined length (a fixed-length element) is repeating and the minimum occurrences is not the same as maximum occurrences, an RED is not required, but a delimiter *is* required after the last repeat. Delimiter suppression of this repeat can occur.

For example, if data4 (with a fixed length of 8) is repeating, and its minimum occurrences is 2, maximum occurrences is 4:

```
data1*data2*data3*data400data400data400data400*data5
```

where Delimiter is *

Or, if there are only two occurrences of data4:

```
data1*data2*data3*data4000data4000*data5
```

If an element with a defined length (a fixed-length element) is repeated, and the minimum occurrences is the same as maximum occurrences, an RED is not required. A delimiter is also not required after the last repeat. Truncation of this repeat cannot occur and all elements need to be present.

For example, if data4 (with a fixed length of 8) repeats four times:

```
data1*data2*data3*data4000data4000data4000data4000data5
```

where Delimiter is *

Or, if there are only two occurrences of data4:

```
data1*data2*data3*data4000data400000000000000000000data5
```

This also applies for a non-fixed length complex type or group inside a Variable Length Elements Delimited environment.

If a complex type has Variable Length Elements Delimited separation, a delimiter is always output between an included ('child') complex element and the next element even if the separation of the 'child' complex element is Fixed Length. On input, the parser accepts the bit stream with or without such a delimiter.

Related concepts:

"MRM TDS format: Data element separation" on page 1225
Data element separation defines how a TDS message is to be parsed.

"MRM TDS format: Tagged separation types" on page 1228
For tagged separation types, each data value is preceded by a tag that is specified as an element property.

"MRM TDS format: Delimited separation types" on page 1232
For delimited separation types, a delimiter is used to separate data fields, but there are no tags present. The data fields must be given in the correct order in the bit stream and elements cannot be omitted from the middle of the bit stream.

"MRM TDS format: Data pattern separation types"
For a data pattern separation type, each data value is matched with a regular expression that is specified as a property of each element.

"MRM TDS format: Message model integrity" on page 1239
When you use the TDS wire format, you must conform to a number of rules that apply to the setting of values of properties. This is necessary to avoid any discrepancies when processing a message within the specified model.

Related tasks:

"Adding a TDS physical format" on page 2851
Add a Tagged/Delimited String (TDS) physical format layer to a message set using the Message Set editor.

"Configuring TDS properties: Message sets" on page 2852
Configure the Tagged/Delimited String (TDS) format properties of a message set using the Message Set editor.

"Configuring TDS properties: Message model objects" on page 2914
You can configure the Tagged/Delimited String (TDS) properties of a message model object.

Related reference:

"TDS message model integrity" on page 6295
When you use the TDS wire format, you must conform to a number of rules that apply to the setting of values of properties. These rules are checked whenever the project is saved.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

MRM TDS format: Data pattern separation types:

For a data pattern separation type, each data value is matched with a regular expression that is specified as a property of each element.

The length of both textual and non-textual data is determined by the `Data Pattern` property of the element. If the `Physical Type` of the element is Length Encoded String 1 or Length Encoded String 2, the regular expression must match both the encoded length and the following data. The length in the encoded length must be consistent with the length matched by the regular expression. If the `Physical Type` of the element is Null Terminated String, the regular expression must match both the data and the following null terminator.

The Data Pattern separation type uses a regular expression that is specified for each element to match the data. The parser matches the data with the regular expression in the `Data Pattern` property for that element. TDS parsing in the MRM parser uses the regular expression in `Data Pattern` to determine the length of the element, whether it is repeating, and whether it is present in the bit stream.

No delimiters or tags, other than those coded as part of the regular expression pattern, are used in the bit stream. See "Using regular expressions to parse data elements" on page 6301 for an explanation of pattern matching.

For example, if the first three `Data Pattern` properties are, respectively:

- [A-Z]{1,3}
- [0-9]+
- [a-z]*

and the message data is:

```
DT31758934information for you
```

Then, in this example:

- First data element = `DT`
- Second data element = `31758934`
- Third data element = `information`

The first data pattern means "from one to three characters in the range A to Z", the second means "one or more characters in the range 0 to 9", and the third means "zero or more characters in the range a to z". Note how each element's data was terminated by the first character that did not match the element's `Data Pattern`.

If the TDS message that is being parsed is encoded in a single-byte code page, the `Data Pattern` property can include hexadecimal values. A hexidecimal value is specified as \xNN, where N is a hexadecimal digit in the range 0 to F. Note, however, that the value \x00 is not valid.

Performance issues

The parsing required in Data Pattern separation type is the slowest of all the different separation types because of its complexity.

Therefore, use Data Pattern separation type only when no other separation type models the message. Do not use it, for example, when you can use Fixed Length separation type.

Applicable parameters: Only one parameter is used:

`Data Pattern` for each element, indicates the regular expression that is used for string matching.

Related concepts:

"MRM TDS format: Data element separation" on page 1225
Data element separation defines how a TDS message is to be parsed.

"MRM TDS format: Fixed-length separation types" on page 1227
For fixed-length separation types, each data value is a fixed length.

"MRM TDS format: Tagged separation types" on page 1228
For tagged separation types, each data value is preceded by a tag that is specified as an element property.

"MRM TDS format: Delimited separation types" on page 1232
For delimited separation types, a delimiter is used to separate data fields, but there are no tags present. The data fields must be given in the correct order in the bit stream and elements cannot be omitted from the middle of the bit stream.

"MRM TDS format: Message model integrity" on page 1239
When you use the TDS wire format, you must conform to a number of rules that apply to the setting of values of properties. This is necessary to avoid any discrepancies when processing a message within the specified model.

Related tasks:

"Adding a TDS physical format" on page 2851
Add a Tagged/Delimited String (TDS) physical format layer to a message set using the Message Set editor.

"Configuring TDS properties: Message sets" on page 2852
Configure the Tagged/Delimited String (TDS) format properties of a message set using the Message Set editor.

"Configuring TDS properties: Message model objects" on page 2914
You can configure the Tagged/Delimited String (TDS) properties of a message model object.

Related reference:

"TDS message model integrity" on page 6295
When you use the TDS wire format, you must conform to a number of rules that apply to the setting of values of properties. These rules are checked whenever the project is saved.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

MRM TDS format: Message model integrity:

When you use the TDS wire format, you must conform to a number of rules that apply to the setting of values of properties. This is necessary to avoid any discrepancies when processing a message within the specified model.

Rules of TDS physical format properties: Restrictions to message formats are checked. These restrictions follow the rules specified in "TDS message model integrity" on page 6295. Most rules are applied for at least one of these reasons:

Rules for message definition
Some rules are necessary for the message to be defined.

For example, in a Fixed Length separation type all elements must have some length defined, either directly or by using a Length Reference. Without this information, it is impossible to tell in the message bit stream where one data element ends and the next starts.

Rules for nesting
Nesting rules relate to which separation types can be nested inside each other.

Such rules are applied when an element of a complex type is present inside another complex type. An example is that it is not possible to have a Tagged Delimited separation type inside a Fixed Length type. Because a Tagged Delimited separation type is of variable length, the parent Fixed Length type would be unable to tell where that particular element ended, as there would be no length provided. Therefore the message could not be processed.

Rules linking to the logical model
There are also rules linking TDS to the logical model.

These rules occur where a group composition or group content validation cannot be used with a particular separation type. Again this is for message integrity. For example, a separation type of All Elements Delimited cannot have a group composition of Open, as there is no information as to what the extra elements represent and where they are in the bit stream.

Related Concepts

"MRM TDS format: Data element separation" on page 1225
Data element separation defines how a TDS message is to be parsed.

"MRM TDS format: NULL handling"
NULL handling dictates the way in which the MRM parser for TDS messages handles elements that have been set to Null.

"MRM TDS format: Multipart messages" on page 1241
The Tagged/Delimited String Format (TDS) supports both the Message Identity technique and the Message Path technique of identifying embedded messages within a multipart message.

"MRM TDS format: Data conversion" on page 1242
TDS string data is subject only to CCSID conversion.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related Tasks

"Configuring logical properties: Message sets" on page 2846
Configure the logical properties of a message set using the Message Set editor.

"Configuring TDS properties: Message sets" on page 2852
Configure the Tagged/Delimited String (TDS) format properties of a message set using the Message Set editor.

"Configuring TDS properties: Message model objects" on page 2914
You can configure the Tagged/Delimited String (TDS) properties of a message model object.

Related Reference

"TDS message model integrity" on page 6295
When you use the TDS wire format, you must conform to a number of rules that apply to the setting of values of properties. These rules are checked whenever the project is saved.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

MRM TDS format: NULL handling:

NULL handling dictates the way in which the MRM parser for TDS messages handles elements that have been set to Null.

Null handling takes place only if the logical Nillable property of the element is set. The rules for whether nulls are permitted are described in "TDS Null handling options" on page 6293.

Null properties: The element properties `Encoding Null` and `Encoding Null Value` control how null handling is represented for individual elements.

You can select the `Encoding Null` property from the enumerated values NULLPadFill, NULLLogicalValue, NULLLiteralValue, and NULLLiteralFill. The use of the `Encoding Null Value` property is dependent on the value that you select for the `Encoding Null` property.

NULL values are not defined for schema xsd:hexBinary simple types. The properties `Encoding Null` and `Encoding Null Value` are therefore not set for xsd:hexBinary types.

NULL values for schema `Boolean` simple types are defined at the message set level. The message set property `Boolean Null Representation` specifies the value to be used for Boolean Null representation.

Related concepts:

"MRM TDS format" on page 1221
The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

"MRM TDS format: Data element separation" on page 1225
Data element separation defines how a TDS message is to be parsed.

"MRM TDS format: Message model integrity" on page 1239
When you use the TDS wire format, you must conform to a number of rules that apply to the setting of values of properties. This is necessary to avoid any discrepancies when processing a message within the specified model.

"MRM TDS format: Multipart messages"
The Tagged/Delimited String Format (TDS) supports both the Message Identity technique and the Message Path technique of identifying embedded messages within a multipart message.

"MRM TDS format: Data conversion" on page 1242
TDS string data is subject only to CCSID conversion.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Adding a TDS physical format" on page 2851
Add a Tagged/Delimited String (TDS) physical format layer to a message set using the Message Set editor.

"Configuring TDS properties: Message sets" on page 2852
Configure the Tagged/Delimited String (TDS) format properties of a message set using the Message Set editor.

"Configuring TDS properties: Message model objects" on page 2914
You can configure the Tagged/Delimited String (TDS) properties of a message model object.

Related reference:

"TDS Null handling options" on page 6293
TDS supports the handling of null values within messages, provided that the logical Nillable property of the element is set.

"TDS message model integrity" on page 6295
When you use the TDS wire format, you must conform to a number of rules that apply to the setting of values of properties. These rules are checked whenever the project is saved.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

MRM TDS format: Multipart messages:

The Tagged/Delimited String Format (TDS) supports both the Message Identity technique and the Message Path technique of identifying embedded messages within a multipart message.

The SWIFT, X12, and EDIFACT messaging standards can all be modeled by using the Message Identity technique.

Related concepts:

"Multipart messages" on page 1191
A *multipart message* contains one or more other messages within its structure. The contained message is sometimes referred to as an *embedded message*.

"Identifying an embedded message by using a Message Identity" on page 1193
You can identify an embedded message by using the Message Identity. This technique is used by the MRM domain, and replaces the use of the Message Key.

"Identifying an embedded message by using a Message Path" on page 1196
The Message Path technique for identifying embedded messages is useful when the multipart message contains no information about the identity of an embedded message.

Related tasks:

"Creating a multipart message" on page 2919
A multipart message occurs when you embed a message in another message.

"Accessing embedded messages in the MRM domain" on page 2594
If you have defined a multipart message, you have at least one message embedded within another. Within the overall complex type that represents the outer messages, you can model the inner message in two ways.

Related reference:

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

MRM TDS format: Data conversion:

TDS string data is subject only to CCSID conversion.

All TDS message data apart from binary types are handled as strings. All string data is therefore subject to CCSID conversion only. This includes the special characters used as delimiters, data separators, and so on\.

Related concepts:

"MRM TDS format" on page 1221
The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

"MRM TDS format: Data element separation" on page 1225
Data element separation defines how a TDS message is to be parsed.

"MRM TDS format: Message model integrity" on page 1239
When you use the TDS wire format, you must conform to a number of rules that apply to the setting of values of properties. This is necessary to avoid any discrepancies when processing a message within the specified model.

"MRM TDS format: NULL handling" on page 1240
NULL handling dictates the way in which the MRM parser for TDS messages handles elements that have been set to Null.

"MRM TDS format: Multipart messages" on page 1241
The Tagged/Delimited String Format (TDS) supports both the Message Identity technique and the Message Path technique of identifying embedded messages within a multipart message.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Adding a TDS physical format" on page 2851
Add a Tagged/Delimited String (TDS) physical format layer to a message set using the Message Set editor.

"Configuring TDS properties: Message sets" on page 2852
Configure the Tagged/Delimited String (TDS) format properties of a message set using the Message Set editor.

"Configuring TDS properties: Message model objects" on page 2914
You can configure the Tagged/Delimited String (TDS) properties of a message model object.

Related reference:

"CWF data conversion" on page 6255
You can convert an MRM message to a different code page or encoding, or both.

MRM TDS format: Relationship to the logical model:

TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

The rules that govern these options are explained in "Restrictions for nesting complex types" on page 6298.

These rules exist to ensure the integrity of the message. A combination of separation type and group composition or group content validation must not lead to a message that is unclear to a TDS parser.

Default values

In TDS, *Default* values are only observed by fixed-length elements:

Separation Type	**Use of `Default` values**
Tagged Delimited Tagged Fixed Length Tagged Encoded Length All Elements Delimited Data Pattern	`Default` values are never observed.
Fixed Length Fixed Length AL3	`Default` values are observed on output by all elements. An absent element that has no `Default` value defined, causes an error on writing.
Variable Length Elements Delimited	`Default` values are only observed by fixed-length elements on output. Absent fixed-length values must have a `Default` value available to them. An absent element that has no `Default` value defined, causes an error on writing.

Simple types – lists and unions

Lists and unions are XML-specific concepts. An element or attribute of a simple type that is a list or a union causes a task list warning if a TDS physical format is present in the message set. The user can choose whether to make this an error, warning, or information by editing the Validation preferences. If a dictionary is generated from the message set, and an attempt is made to parse a TDS message defined to contain such elements or attributes, a runtime error occurs.

Min Occurs and Max Occurs

The logical properties Min Occurs and Max Occurs specify the permitted number of occurrences of an element or group in a message. They are used when parsing and writing messages, and when validating the content of a message.

When parsing and writing, the exact interpretation of these properties depends on the Data Element Separation property of the parent complex type or group as shown in the following table.

However, this behavior is overridden if the TDS Repeat Reference property is set, which indicates that the number of occurrences is given instead by an integer element that occurs earlier in the message. See "Repeat reference" on page 1245 for more information.

When validating, Min Occurs and Max Occurs are both used to check that the content of the message tree matches the model.

Separation type	Interpretation of Min Occurs and Max Occurs
Tagged Delimited Tagged Fixed Length Tagged Encoded Length	Min Occurs and Max Occurs are effectively ignored when parsing and writing. When parsing, the number of occurrences is identified by the tags in the message. When writing, the writer outputs all occurrences in the message tree. • A varying number of occurrences (Min Occurs <> Max Occurs) is allowed. • Optional occurrence (Min Occurs = 0) is allowed. • Always absent (Max Occurs = 0) is allowed. • An unbounded number of occurrences (Max Occurs = -1) is allowed.
All Elements Delimited	Max Occurs is used only when parsing and writing, with the element's Repeating Element Delimiter property, and the parent type's Suppress Absent Element Delimiters property. A varying number of occurrences (Min Occurs <> Max Occurs) is allowed if Suppress Absent Element Delimiters is set to End of Type. • If the Delimiter is different from the Repeating Element Delimiter, the Delimiter signifies the end of the occurrences. • If the Delimiter is the same as the Repeating Element Delimiter, an empty repeat signifies the end of the occurrences. • In both these cases, Max Occurs is the maximum number of repeats that are expected. If Suppress Absent Element Delimiters is Never, all occurrences are expected when parsing, and produced when writing, although parsing accepts elements being absent. Optional occurrence (Min Occurs = 0) is ignored and a delimiter is still expected when parsing, and produced when writing. Always absent (Max Occurs = 0) is allowed. No delimiter is expected when parsing, nor output when writing. An unbounded number of occurrences (Max Occurs = -1) is only allowed if the Repeating Element Delimiter is different from the Delimiter. The repeats must be terminated by the delimiter, or a containing group's Group Terminator or Delimiter, or by the end of the message bit stream. On writing, the writer outputs all occurrences in the message tree.

Separation type	Interpretation of Min Occurs and Max Occurs
Fixed Length Fixed Length AL3	Max Occurs is used only when parsing and writing. In general, Max Occurs occurrences are expected when parsing, and Max Occurs occurrences are produced when writing; default values are used for missing elements, and any excess elements are discarded. A varying number of occurrences (Min Occurs <> Max Occurs) is ignored, Max Occurs is assumed. Optional occurrence (Min Occurs = 0) is ignored, Max Occurs is assumed. Always absent (Max Occurs = 0) is allowed. Fixed Length only. An unbounded number of occurrences (Max Occurs = -1) is allowed if the element or group is the last child in its parent group, and the group is terminated by a Group Terminator or a containing group's Group Terminator or Delimiter or by the end of the message bit stream. On writing, the writer outputs all occurrences in the message tree, if this number is less than Min Occurs, additional default values are written.
Variable Length Elements Delimited	For fixed length simple elements, the rules for Fixed Length separation above are followed with two differences. 1. A varying number of occurrences (Min Occurs <> Max Occurs) is allowed, the end of the occurrences being signified by an extra delimiter. 2. An unbounded number of occurrences (Max Occurs = -1) is allowed, the end of the occurrences being signified by an extra delimiter. On writing, the writer outputs all occurrences in the message tree, followed by an extra delimiter. For variable length simple elements, all complex elements and groups, the rules for All Elements Delimited above are followed.
Data Pattern	Min Occurs and Max Occurs are effectively ignored when parsing and writing. When parsing, the pattern is matched as many times as possible. When writing, the writer outputs all occurrences in the message tree. Note that on parsing, if the data pattern permits a zero length match, and a zero length match occurs, an element is added to the message tree, and the matching terminates to prevent an infinite loop. A varying number of occurrences (Min Occurs <> Max Occurs) is allowed. Optional occurrence (Min Occurs = 0) is allowed. Always absent (Max Occurs = 0) is allowed. An unbounded number of occurrences (Max Occurs = -1) is allowed.

Repeat reference

The TDS property Repeat reference specifies a field that holds the number of repeats of an object (Element or Group) within a message. The field that holds the number of repeats must be within the message before the object that it refers to.

From a parsing perspective, the Repeat reference property replaces the role of the minOccurs and maxOccurs properties.

If a value for the Repeat reference property is specified for an object, values that are specified for minOccurs and maxOccurs are ignored when parsing and writing. However, values that are specified for minOccurs and maxOccurs are used by logical validation.

When parsing and writing, the exact interpretation of the Repeat reference property depends on the Data Element Separation property of the parent complex type or group as shown in the following table.

Separation type	Interpretation of Repeat reference
Tagged Delimited Tagged Fixed Length Tagged Encoded Length	Repeat reference is effectively ignored when parsing and writing. When parsing, the number of occurrences is identified by the tags in the message. When writing, the writer outputs all occurrences in the message tree.
All Elements Delimited	Repeat reference is used when parsing and writing, with the element's Repeating Element Delimiter property, and the parent type's Suppress Absent Element Delimiters property. A Repeat reference is allowed only if the parent complex type or group has Suppress Absent Element Delimiters set to Never. All Repeat reference occurrences are expected when parsing, and produced when writing. However, parsing accepts elements being absent. Repeat reference = 0 is allowed. No delimiter is expected when parsing, nor produced when writing.
Fixed Length Fixed Length AL3	Repeat reference is used when parsing and writing. Repeat reference occurrences are expected when parsing, and are produced when writing, with default values used for missing elements. Repeat reference = 0 is allowed.
Variable Length Elements Delimited	For fixed length simple elements, the rules for Fixed Length separation above are followed. For variable length simple elements, all complex elements and groups, the rules for All Elements Delimited that are listed above are followed.
Data Pattern	Repeat reference is effectively ignored when parsing and writing. When parsing, the pattern is matched as many times as possible. When writing, the writer outputs all occurrences in the message tree. Note that, on parsing, if the data pattern permits a zero length match, and a zero length match occurs, an element is added to the message tree, and the matching terminates to prevent an infinite loop.

Related concepts:

"MRM TDS format" on page 1221
The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

"MRM TDS format: Data element separation" on page 1225
Data element separation defines how a TDS message is to be parsed.

"MRM TDS format: Message model integrity" on page 1239
When you use the TDS wire format, you must conform to a number of rules that apply to the setting of values of properties. This is necessary to avoid any discrepancies when processing a message within the specified model.

"MRM TDS format: NULL handling" on page 1240
NULL handling dictates the way in which the MRM parser for TDS messages handles elements that have been set to Null.

"MRM TDS format: Multipart messages" on page 1241
The Tagged/Delimited String Format (TDS) supports both the Message Identity technique and the Message Path technique of identifying embedded messages within a multipart message.

"MRM TDS format: Data conversion" on page 1242
TDS string data is subject only to CCSID conversion.

Related reference:

"Restrictions for nesting complex types" on page 6298
If you include a group within another group or complex type, the *Data Element Separation* property for the nested group must be compatible with the *Data Element Separation* property of the parent group or complex type.

MRM XML physical format:

The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

An XML wire format describes the physical representation of a message that is written according to the standards given in the W3C Extensible Markup Language (XML) specification. The wire format defines information that is used to parse or write XML messages in a runtime environment such as a broker. XML versions 1.0 and 1.1 are both supported.

You can add more than one XML physical format to a message set, but within that message set, each physical format must have a unique name. The default name for an XML wire format is XML1. The physical format name identifies the definitions that are to be used by the broker at run time.

After adding an XML physical format, all XML properties for all existing objects in the message set are set to default values. Therefore, immediately after adding the format and deploying the message set to a runtime environment, you can process XML messages by using MRM features.

You can configure XML properties for the message set, and for objects within the message set. Objects that can have XML properties are messages, elements, and attributes. For example, a message object can be customized to define a specific DTD declaration on output; an element can have a tag name assigned to it which is different from its element name in the model.

Adding an XML wire format to a message set allows you to both process input messages, and to construct output messages in this format. You can also transform messages between XML and either CWF or TDS.

XML messages are, by their nature, self-describing: each piece of data is prefixed by a tag name or an attribute name. Therefore, it is possible for an XML message instance to contain elements that are not in the definition for that message.

- If such an element exists in the message set, the model objects for that element are used in parsing or writing the message.
- If the element does not exist in the message set, it is treated as a self-defining element, and its data type is set to string.

Although it is possible to define an XML message 'by hand', using the Message Definition Editor, WebSphere Message Broker also provides importers for both XML Schema and DTD, and these are often quicker and easier than manual definition.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"Which XML parser should you use?" on page 1080
If your messages are general-purpose XML documents, you can use one of the dedicated XML domains (XMLNSC or XMLNS) to parse the message, or you can use the MRM domain to parse the message.

"Importing from other model representations to create message definitions" on page 1254
You can add message definitions to your message set by importing application message formats that already exist.

Related tasks:

"Adding an XML wire format" on page 2853
You can add an XML wire format physical format layer to a message set by using the Message Set editor.

"Configuring XML Wire Format properties: Message model objects" on page 2916
You can configure the XML Wire Format properties of a message model object.

"Importing from XML DTD" on page 2954
You can create a new message definition from an XML DTD by using the New Message Definition File wizard in the WebSphere Message Broker Toolkit.

"Adding files to a broker archive" on page 3223
To deploy files to an execution group, include them in a broker archive (BAR) file.

"Removing a physical format" on page 2858
You can remove a physical format layer from your message set.

"Configuring XML Wire Format properties: Message sets" on page 2855
Configure the XML Wire Format properties of a message set using the Message Set editor.

Related reference:

"Import formats" on page 6346
Supported features of formats that have been imported from C, COBOL, SCA Export, SCA Import, WSDL, or XML Schema.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

MRM XML physical format: Message model integrity:

When you save a message definition file, the definitions that it contains are checked to ensure that they make sense and provide sufficient information about the message. This action is called *model validation*.

For XML, these checks mostly concern the uniqueness and validity of XML names in global elements and attributes, and also for elements and attributes within complex types or groups.

Tests other than these tend to be both simple and obvious so that, for example, the message set property `First Day of Week` must be the name of a day in the week.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"Importing from other model representations to create message definitions" on page 1254
You can add message definitions to your message set by importing application message formats that already exist.

Related tasks:

"Adding an XML wire format" on page 2853
You can add an XML wire format physical format layer to a message set by using the Message Set editor.

"Configuring XML Wire Format properties: Message model objects" on page 2916
You can configure the XML Wire Format properties of a message model object.

"Importing from XML DTD" on page 2954
You can create a new message definition from an XML DTD by using the New Message Definition File wizard in the WebSphere Message Broker Toolkit.

Related reference:

"Import formats" on page 6346
Supported features of formats that have been imported from C, COBOL, SCA Export, SCA Import, WSDL, or XML Schema.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

MRM XML physical format: NULL handling:

The purpose of null handling is to specify how messages deal with null values; that is, the absence of a meaningful value for an element.

Null properties for the MRM XML physical format are set for the message set only, and apply to all the defined objects within the message set, by using the four properties `Encoding Null Num`, `Encoding Null Non-Num`, `Encoding Null Num Val` and `Encoding Null Non-Num Val`.

Null handling takes place only if the logical Nillable property of the element is set.

The purpose of these parameters is to specify how messages deal with null values. In an XML message there are several options. Most obviously an element could omit a value, for example:

```
<element1></element1>
```

Or, the element could include a distinctive value that means that no real value is present, for example.

```
<element1>null</element1>
```

Or, the element could follow XML Schema instance rules:

```
<element1 xsi:nil="true"/>
```

The properties `Encoding Null Num` and `Encoding Null Non-Num` specify the style of null handling, for example, that null is represented by an empty element.

The properties `Encoding Null Num Val` and `Encoding Null Non-Num Val` provide a value (if needed) to represent a null value. For an element of type string, this might be `null` or `unspecified` while for a number it might be `0` or `0.0`.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"Importing from other model representations to create message definitions" on page 1254
You can add message definitions to your message set by importing application message formats that already exist.

Related tasks:

"Adding an XML wire format" on page 2853
You can add an XML wire format physical format layer to a message set by using the Message Set editor.

"Configuring XML Wire Format properties: Message model objects" on page 2916
You can configure the XML Wire Format properties of a message model object.

"Importing from XML DTD" on page 2954
You can create a new message definition from an XML DTD by using the New Message Definition File wizard in the WebSphere Message Broker Toolkit.

Related reference:

"Import formats" on page 6346
Supported features of formats that have been imported from C, COBOL, SCA Export, SCA Import, WSDL, or XML Schema.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

MRM XML physical format: Multipart messages:

Identify embedded messages by using either a Message Identity or a Message Path.

If you are using the MRM XML physical format, an embedded message can be identified in any of the following ways:

- Message Identity

  See "Identifying an embedded message by using a Message Identity" on page 1193.
- Message Path

  See "Identifying an embedded message by using a Message Path" on page 1196.
- Automatic

  The MRM parser identifies the message by matching the next XML tag in the bit stream against the XML Name of a message definition.

If you choose the Message Identity or Message Path technique, the MRM parser still checks that the next XML tag name matches the XML Name of the message that was identified. If the XML Name does not match, an exception is thrown.

Where you have defined the embedded message in a different message set, you must use a Message Set Identity element or attribute value to specify the target message set. Note that the message sets within which the root and subsequent embedded messages are defined must be consistent in their use of the 'Use Namespace' property of the message set. That is, embedded messages that are defined in a namespace-aware message set and that are contained within a parent message that is defined in a message set that is not namespace-aware, are not supported. Similarly, embedded messages that are defined in a message set that is not namespace-aware and that are contained within a parent message that is defined in a namespace-aware message set, are not supported.

If the embedded message definition is a complex type, the message definition contains a complex element based on that complex type. This complex element has

its own tag, which appears in the bit stream before the tag for the embedded message. If you want to avoid this extra tag, you can create the embedded message definition from a group, and insert the group at the appropriate position in the message model.

Tip: Note that the root tag property of an embedded message is not applicable.

Related concepts:

"Multipart messages" on page 1191
A *multipart message* contains one or more other messages within its structure. The contained message is sometimes referred to as an *embedded message*.

"Identifying an embedded message by using a Message Identity" on page 1193
You can identify an embedded message by using the Message Identity. This technique is used by the MRM domain, and replaces the use of the Message Key.

"Identifying an embedded message by using a Message Path" on page 1196
The Message Path technique for identifying embedded messages is useful when the multipart message contains no information about the identity of an embedded message.

Related tasks:

"Creating a multipart message" on page 2919
A multipart message occurs when you embed a message in another message.

"Accessing embedded messages in the MRM domain" on page 2594
If you have defined a multipart message, you have at least one message embedded within another. Within the overall complex type that represents the outer messages, you can model the inner message in two ways.

MRM XML physical format: Relationship to the logical model:

The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

These restrictions are documented in "MRM restrictions" on page 6252.

Default values

The MRM XML physical format ignores default and fixed values on elements and attributes. If validation is enabled in WebSphere Message Broker, this can lead to unexpected validation errors for missing elements, even though they have default or fixed values.

Simple types – unions and lists

The XML properties of an element or attribute of a simple type that is a union or list vary depending on the members of the union or the itemType of the list. If the union or list includes a dateTime type (or other date/time related type) the `Date Format` field is displayed. If the union includes a binary type, the `Encoding` field is displayed.

Min Occurs and Max Occurs

The logical properties `Min Occurs` and `Max Occurs` specify the permitted number of occurrences of an element or group in a message. They are used when validating the content of a message.

When parsing and writing, using the MRM XML physical format, `Min Occurs` and `Max Occurs` are effectively ignored. When parsing, the number of occurrences is identified by the tags in the message. When writing, the writer outputs all occurrences in the message tree.

- A varying number of occurrences (`Min Occurs` <> `Max Occurs`) is allowed.
- Optional occurrence (`Min Occurs` = 0) is allowed.
- Always absent (`Max Occurs` = 0) is allowed
- An unbounded number of occurrences (`Max Occurs` = -1) is allowed.

When validating, `Min Occurs` and `Max Occurs` are both used to check that the content of the message tree matches the model.

Related concepts:

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

Related reference:

"MRM restrictions" on page 6252
The MRM parser does not exactly follow the XML Schema 1.0 specification.

MRM XML physical format: Handling xsi:type attributes:

The prefix "xsi" is the namespace prefix used by convention for the XML Schema instance namespace. XML documents can contain elements that have an xsi:type attribute. This behavior provides an explicit data type for the element.

The MRM XML parser in sensitive to xsi:type attributes in the XML document. It modifies the data type of the element accordingly and adds the xsi:type attribute into the message tree.

The MRM XML writer is sensitive to xsi:type attributes in the message tree. It produces xsi:type attributes according to XML Wire Format message set property **Output policy for xsi:type attributes**. For example, xsi:type attributes can be removed, output on all elements or output according to rules specified in the SOAP standard.

If validation is enabled for a WebSphere Message Broker message flow, the validation logic is sensitive to xsi:type attributes and uses them to modify the validation of the element. It also validates the values of xsi:type attributes by using the rules described in XML Schema Part 1: Structures on the World Wide Web Consortium (W3C) website.

There are several important points to remember when parsing and writing XML documents that contain xsi:type attributes.

- In order to detect and use xsi:type attributes, the message set must be namespace-enabled. To make a message set namespace-enabled, check the message set property **Use namespaces**.
- If the value of the xsi:type attribute contains a namespace prefix, the prefix is expanded into a fully qualified URI by the MRM XML parser. If the same xsi:type attribute is produced later by the MRM XML writer, the same prefix is

not automatically used for the value. You can control the prefixes used on output by using the **Namespace settings** list in the XML Wire Format message set properties. If no prefix is supplied, the XML writer assigns a default prefix.

- If the xsi:type attribute of an element does not resolve to a type in the model, the behavior depends on whether MRM validation is enabled. If not validating, the MRM assumes that the type of the element is that declared in the model, and continue. If validating, a validation exception occurs.
- If MRM validation is enabled, any required xsi:type attributes must be present in the message tree at the point when validation is performed. An xsi:type attribute is required when its value is different from the data type of the element as defined in the message model (this most commonly occurs when using XML Schema type derivation).
  - If validation is being performed on an input message, the MRM XML parser ensures that xsi:type attributes appear in the message tree, as described above.
  - If validation is being performed on an output message, you must ensure that the correct xsi:type attributes appear in the message tree. Ensure that any required xsi:type attributes are copied from input message tree to output message tree, or are explicitly created in the output message tree.
- If you are using simple types that are xsd:unions, an xsi:type attribute can be used to direct the MRM XML parser when resolving the union.
- If you have migrated from an earlier version of WebSphere Message Broker that was not sensitive to xsi:type attributes, you might notice some changes of behavior. For example, xsi:type attributes are no longer treated as self-defining attributes, so they appear in the message tree with the name 'type' instead of '@type'. If your message flow logic is sensitive to xsi:type attributes in the message tree, change your message flow to comply with the current behavior.

For more information about xsi:type attributes, see XML Schema Part 0: Primer on the World Wide Web Consortium (W3C) website.

Related concepts:

"XML Schema" on page 1172
XML Schema is an international standard from the World Wide Web Consortium (W3C) that defines a language for describing the structure of XML documents.

Related reference:

"XML Wire Format message set properties" on page 5400
The following tables define the properties for the XML Wire Format for the message set.

Ways to create message definitions

When you have created a message set, you must populate the message set with message definitions.

You can populate the message set in one of the following ways:

- By importing application message formats that are described by XML Schemas, IBM supplied messages, XML DTD, C structures, COBOL structures, SCA imports or exports, or WSDL definitions; use the wizards that are started from the New Message Definition File From wizard.
- By creating empty message definition files, then creating your messages by using the Message Definition Editor; use the New Message Definition File wizard.
- By using the Adapter Connection wizard to import EIS metadata.

Additionally, you can import application message formats by using the **mqsicreatemsgdefs** or **mqsicreatemsgdefsfromwsdl** command line utilities.

The **mqsicreatemsgdefs** command has a bulk import capability, but **mqsicreatemsgdefsfromwsdl** imports only one WSDL definition at a time.

Related concepts:

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

"Message definition files" on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

"Importing from other model representations to create message definitions"
You can add message definitions to your message set by importing application message formats that already exist.

Related tasks:

"Working with a message set" on page 2840
Complete a variety of tasks that are involved in working with a message set.

"Creating a message definition file" on page 2865
Creating an empty message definition file to contain your message model objects.

"Working with data structures" on page 2930
You can create a message definition file in a message set by importing from XML Schema, XML DTD, SCA import or export components, IBM supplied messages, WSDL definitions, IDL files, C header files, and COBOL copybooks. This topic area describes how to import from these data structures using the command line or the WebSphere Message Broker Toolkit.

Related reference:

"Message Definition editor" on page 6804
The Message Definition editor is the default editor provided by the Broker Application Development perspective for editing message definition (`.mxsd`) files.

"**mqsicreatemsgdefs** command" on page 3702
Use the **mqsicreatemsgdefs** command to create message definition files.

"**mqsicreatemsgdefsfromwsdl** command" on page 3712
Use the **mqsicreatemsgdefsfromwsdl** command to import a single WSDL definition.

Importing from other model representations to create message definitions:

You can add message definitions to your message set by importing application message formats that already exist.

You can import the following message formats to create message sets:

- XML Schema files
- IBM supplied messages
- XML DTD files
- C header files
- COBOL copybooks
- WSDL definitions
- SCA imports or exports
- EIS metadata

When you import one of these formats, a new message model is created that consists of the elements, attributes, groups, and types that are required to represent your message format. You can choose the name of the message model; if it already exists, the content is deleted and re-created as part of the import operation.

The new message model that is created can consist of both logical and physical information.

To find out which wizards to use to import message formats, see "Ways to create message definitions" on page 1253.

You can also import C header files, COBOL copybooks, XML DTD files, or XML Schema files by using the **mqsicreatemsgdefs** command-line utility. The **mqsicreatemsgdefs** command allows you to import several middle-format files in a single operation, and allows you to create a message set (based on an existing message set) into which the message definition files are placed.

WSDL definitions can be imported by using the **mqsicreatemsgdefsfromwsdl** command-line utility. This utility imports only one WSDL definition at a time.

Client application access to messages

Client applications must have access to message definitions to be able to construct the messages that they send, and to interpret the messages that they receive.

- If the message formats have been imported from C or COBOL structures by using the WebSphere Message Broker Toolkit, your applications can continue to use the same C and COBOL data structures that were imported to create the message dictionary that is used by the brokers.
- If the messages are self-defining XML, the client applications must construct valid messages by using the structures that can be understood by the recipients of the message.

Related concepts:

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

"XML Schema" on page 1172
XML Schema is an international standard from the World Wide Web Consortium (W3C) that defines a language for describing the structure of XML documents.

"Message definition files" on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

"Importing from C header files to create message definitions" on page 1263
Create a message definition file from a C header file for use in the MRM and IDOC domains.

"Importing from COBOL copybooks to create message definitions" on page 1265
You can populate your message set with message definitions by importing COBOL copybook files, by using either the "New Message Definition File From COBOL file" wizard or the **mqsicreatemsgdefs** command.

"Importing WSDL files to create message definitions" on page 1267
Import WSDL files by using the **New Message Definition File From WSDL file** wizard, the **Start from WSDL and/or XSD files** Quick Start wizard, or the **mqsicreatemsgdefsfromwsdl** command.

"Importing from IBM supplied messages to create message definitions" on page 1261
You can add messages to a message set by using the New Message Definition File From IBM supplied messages wizard to import IBM supplied messages.

"Message sets: Importing from DTDs to create message definitions" on page 1261
You can populate a message set with message definitions by importing DTD files, by using either the New Message Definition File From XML DTD file wizard or the **mqsicreatemsgdefs** command.

"Importing from XML Schemas to create message definitions"
You can populate a message set with message definitions by importing XML Schema files, by using the **New Message Definition File From XML Schema file** wizard, the **Start from WSDL and/or XSD files** quick start wizard, or the **mqsicreatemsgdefs** command-line utility.

Related tasks:

"Working with a message set" on page 2840
Complete a variety of tasks that are involved in working with a message set.

"Creating a message definition file" on page 2865
Creating an empty message definition file to contain your message model objects.

"Importing from C" on page 2934
Create a message definition file from a C header file for use in the MRM and IDOC domains.

"Importing from COBOL copybooks" on page 2937
This topic describes how to create a new message definition from a COBOL data structure using the New Message Definition File wizard in the WebSphere Message Broker Toolkit.

"Importing from WSDL" on page 2946
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to create a new message definition from WSDL.

"Importing from IBM supplied messages" on page 2942
You can create a new message definition file from an IBM supplied message.

"Importing from XML Schema" on page 2957
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to create a new message definition from an XML Schema

"Importing from XML DTD" on page 2954
You can create a new message definition from an XML DTD by using the New Message Definition File wizard in the WebSphere Message Broker Toolkit.

Importing from XML Schemas to create message definitions:

You can populate a message set with message definitions by importing XML Schema files, by using the **New Message Definition File From XML Schema file** wizard, the **Start from WSDL and/or XSD files** quick start wizard, or the **mqsicreatemsgdefs** command-line utility.

Each XML Schema file that you import results in a new message definition file within the message set. The root name of the message definition file defaults to the root name of the XML Schema file, but the **New Message Definition File From XML Schema file** wizard allows you to choose a different root file name.

If the message definition file already exists, you must have enabled overwriting to occur for the import to proceed, in which case the existing content is deleted and re-created.

The namespace to which the message definition file created belongs depends on whether namespaces have been enabled for the message set.

- If namespaces have been enabled, the message definition file belongs to the target namespace of the XML Schema file that is imported.
- If namespaces have not been enabled for the message set, the message definition file belongs to the `noTarget` XML namespace irrespective of the target namespace of the imported XML Schema file and therefore is contained in the `(default)` location in your workspace. The implications of importing into a message set with namespaces disabled are described in "Importing XML Schema into message sets with namespaces disabled" on page 1259.

A report file is created during the import operation. This is located by default in the `log` folder of the message set. By default it takes the name of the message definition file, with `.report.txt` appended.

Import by using the New Message Definition File From XML Schema file wizard: When you import by using the **New Message Definition File From XML Schema file** wizard, you can specify which of the global elements or global complex types within the imported XML Schema file are to be messages within the message definition file.

You can import only one XML Schema file with each import operation. If your XML Schema file references other XML Schema files, with import or include elements, these XML Schema files must be imported into the same message set by using a separate import operation.

Import by using the command-line: When you import by using the command line you have the option of either creating no messages or creating a message for each global element and global complex type within the imported XML Schema file. The import operation creates a message and corresponding global element in the message definition file for each global element you specify. If you do not specify that messages are to be created, you must create them manually using the message definition editor after the import has completed.

You can import several XML Schema files in each import operation.

Physical information: In addition to creating logical information, the import can also create physical information. If the message set contains any XML wire format physical formats, the physical format properties for *all* XML Wire Format layers is populated. If the message set does not contain any XML physical formats, only logical information is created. Also, if you import from the command line, only logical information is created in the new message set by default. If you want physical information created as well, see "Importing from the command line" on page 2936 for details.

MRM CWF and TDS physical format properties are *not* populated and so take default values.

If you have one or more CWF or TDS layers, the import can cause entries in the task list, warning you that certain CWF or TDS properties must be set if the XML structures you have imported are to appear in a CWF or TDS message.

If the CWF or TDS physical formats are not applicable to your XML structures, you can ignore these task list entries because they are just warnings, they do not prevent your model being generated in another form; for example, as a message dictionary.

Command-line invocation: The **mqsicreatemsgdefs** command-line utility allows you to import several XML Schema files in a single operation. All the XML Schema files must be in single directory, and the directory location must be passed as a parameter to the utility.

When you import into a message set for which namespaces are not enabled, the action to take for unsupported constructs can be specified by using an XML options file. This must contain an XML element called `<XSD_NO_NS>` that holds a set of information that applies to all XML Schema files that are imported during an invocation of the utility. A default XML options file, called `mqsicreatemsgdefs.xml`, is supplied. If you want to apply different sets of information to different XML Schema files, you must create multiple XML files and run the utility multiple times.

When you are importing into a message set for which namespaces are not enabled, there are two other options that you can specify in the `<XSD_NO_NS>` element in the XML options file:

- The prefix to use for the `http://www.w3.org/2001/XMLSchema-instance` namespace; the default is `xsi`.
- Additional namespace URI and prefix pairs.

The **mqsicreatemsgdefs** utility also allows you to create a message set into which the message definition files are placed, as part of the import operation. You can also choose to base the message set created on an existing message set. This facility enables you to prepare an empty message set that contains an XML physical format and pre-populated message set level XML properties, which are then copied into the message set that is created by the import.

Further information about XML Schema: For details about XML Schema, see XML Schema Part 0: Primer on the World Wide Web Consortium (W3C) website.

Related concepts:

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

"Message definition files" on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

"Namespaces in the message model" on page 1201
Use namespaces to qualify message model object names.

"Importing XML Schema into message sets with namespaces disabled" on page 1259
You can import an XML Schema file with a target namespace even if the message set does not have namespaces enabled.

"Quick Start wizards overview" on page 1409
You can use a Quick Start wizard to set up the basic resources that are required to develop a message flow application. The wizard sets up and gives names to containers for the resources that you need when you develop a message flow.

Related tasks:

"Working with a message set" on page 2840
Complete a variety of tasks that are involved in working with a message set.

"Creating a message definition file" on page 2865
Creating an empty message definition file to contain your message model objects.

"Importing from XML Schema" on page 2957
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to create a new message definition from an XML Schema

"Importing from XML DTD" on page 2954
You can create a new message definition from an XML DTD by using the New Message Definition File wizard in the WebSphere Message Broker Toolkit.

"Importing from the command line" on page 2936
This describes how to use the command line importer **mqsicreatemsgdefs** to import C, COBOL copybooks, XML DTD or XML Schema in order to populate a message set with message definitions.

"Creating an application based on WSDL or XSD files" on page 1413
You can use existing WSDL or XSD files as the basis for your solution.

Related reference:

"XML Schema Importer" on page 5370
Preferences for the message set XML Schema Importer.

"Importing from XML Schema: unsupported features" on page 6359
A number of features in XML Schema are not supported, or their support is restricted in some way.

"XSD options file for the **mqsicreatemsgdefs** command" on page 3709
Specify the options for the **mqsicreatemsgdefs** command when you import an XML Schema file.

Importing XML Schema into message sets with namespaces disabled:

You can import an XML Schema file with a target namespace even if the message set does not have namespaces enabled.

When you import an XML Schema file with a target namespace into a message set for which namespaces have not been enabled, the created message definition file is placed in the XML `no target` namespace. In some cases, this action can lead to name conflicts if global constructs have the same name in different namespaces in the XML Schema files imported into the same message set. These conflicts cause error entries in the task list that you must resolve before generating the model in another format, such as a message dictionary.

Because all the message definition files are in the XML `no target` namespace, the namespace information associated with the XML Schema file is lost. However, the importer provides a limited form of namespace support by prefixing the XML names in the XML Wire Format layers with a namespace prefix. To allow this namespace support to work, an imported XML Schema file must specify an `xmlns` attribute with a non-empty prefix for the target namespace of the XML Schema file. This prefix is used in the XML names in the XML Wire Format layers.

Therefore you cannot specify the target namespace of the XML file as the default namespace. Each namespace in the XML Schema files must use a unique prefix and the same namespace must always use the same prefix. Any XML instance documents against which you match any of the forms generated from the model, must also use the same prefixes for the namespaces.

The XML Schema importer creates a number of optional attributes in an attribute group to represent namespace information. This attribute group is referenced by the type of any message. An attribute is created to represent the location of the XML Schema file, and an attribute is created to represent the mapping of the prefix

to the `http://www.w3.org/2001/XMLSchema-instance` namespace. An attribute is also created for each xmlns attribute in the XML Schema document.

When importing using the Message Definition File wizard the prefix `http://www.w3.org/2001/XMLSchema-instance` namespace can be changed and additional namespace URI/prefix pairs added using the last panel of the Message Definition File wizard. When you use the **mqsicreatemsgdefs** command line utility, the same modifications can be made using the XML options file.

Further information about XML Schema: For details about XML Schema, see XML Schema Part 0: Primer on the World Wide Web Consortium (W3C) Web site.

Related Concepts

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

"Message definition files" on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

"Namespaces in the message model" on page 1201
Use namespaces to qualify message model object names.

"Importing XML Schema into message sets with namespaces disabled" on page 1259
You can import an XML Schema file with a target namespace even if the message set does not have namespaces enabled.

Related Tasks

"Working with a message set" on page 2840
Complete a variety of tasks that are involved in working with a message set.

"Creating a message definition file" on page 2865
Creating an empty message definition file to contain your message model objects.

"Importing from XML Schema" on page 2957
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to create a new message definition from an XML Schema

"Importing from XML DTD" on page 2954
You can create a new message definition from an XML DTD by using the New Message Definition File wizard in the WebSphere Message Broker Toolkit.

"Importing from the command line" on page 2936
This describes how to use the command line importer **mqsicreatemsgdefs** to import C, COBOL copybooks, XML DTD or XML Schema in order to populate a message set with message definitions.

Related References

"XML Schema Importer" on page 5370
Preferences for the message set XML Schema Importer.

"Importing from XML Schema: unsupported features" on page 6359
A number of features in XML Schema are not supported, or their support is restricted in some way.

"XSD options file for the **mqsicreatemsgdefs** command" on page 3709
Specify the options for the **mqsicreatemsgdefs** command when you import an XML Schema file.

Importing from IBM supplied messages to create message definitions:

You can add messages to a message set by using the New Message Definition File From IBM supplied messages wizard to import IBM supplied messages.

Each IBM supplied message that you import results in a new message definition file in the message set. The name of the message definition file defaults to the name of the IBM supplied message, but you can use the New Message Definition File From IBM supplied messages wizard to choose a different file name.

For information about what IBM supplied messages can be imported, see "Importing from the command line" on page 2936.

When you import message by using the New Message Definition File From IBM supplied messages wizard, you can specify only one IBM supplied message definition for each import operation.

If the message definition file already exists, you must have permitted overwriting to occur for the import to proceed, in which case the existing content is deleted and re-created.

A report file is generated during the import operation that allows you to examine what occurred during the import process and check any errors that resulted.

Related concepts:

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

"Message definition files" on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

Related tasks:

"Working with a message set" on page 2840
Complete a variety of tasks that are involved in working with a message set.

"Creating a message definition file" on page 2865
Creating an empty message definition file to contain your message model objects.

Message sets: Importing from DTDs to create message definitions:

You can populate a message set with message definitions by importing DTD files, by using either the New Message Definition File From XML DTD file wizard or the **mqsicreatemsgdefs** command.

Each XML DTD file that you import results in a new message definition file within the message set. The root name of the message definition file defaults to the root name of the XML DTD file, but the New Message Definition File From XML DTD file wizard allows you to choose a different root file name.

If the message definition file exists, you must have permitted overwriting to occur for the import to proceed, in which case the existing content is deleted and re-created.

All message definition files that are created as a result of DTD file import belong to the `noTarget` XML namespace and so are contained in the `(default)` location in your workspace.

A report file is created during the import operation, by default in the `log` folder of the message set. By default, it takes the name of the message definition file, with `.report.txt` appended.

Import by using the New Message Definition File From XML DTD file wizard: When you import by using the "New Message Definition File From XML DTD file" wizard, you can specify which of the elements within the imported XML DTD file are to be messages within the message definition file.

You can import only one XML DTD file with each import operation.

Import by using the command line: When you import by using the command line you have the option of either creating no messages or creating a message for each element within the imported XML DTD file. The import operation creates a message and a corresponding element in the message definition file for each element that you specify. If you do not specify that messages are to be created, you must create them manually using the message definition editor after the import has completed.

You can import several XML DTD files in each import operation.

Physical information: In addition to creating logical information, the import can also create physical information. If the message set contains any XML wire format physical formats, the physical format properties for *all* XML Wire Format layers is populated. If the message set does not contain any XML physical formats, only logical information is created. Also, if you import from the command line, only logical information is created in the new message set by default. If you want physical information created as well, see “Importing from the command line” on page 2936 for details.

MRM CWF and TDS physical format properties are *not* populated and therefore take default values.

If you have one or more CWF or TDS layers, the import can cause entries in the task list, warning you that certain CWF or TDS properties must be set if the XML structures that you have imported are to appear in a CWF or TDS message.

If the CWF or TDS physical formats are not applicable to your XML structures, you can ignore these task list entries because they are just warnings; they do not prevent your model being generated in another form, such as a message dictionary.

Command-line invocation: The **mqsicreatemsgdefs** command-line utility allows you to import several XML DTD files in a single operation. All the XML DTD files must be in single directory, and the directory location must be passed as a parameter to the utility.

The **mqsicreatemsgdefs** utility also allows you to create a message set into which the message definition files are placed, as part of the import operation. You can also choose to base the message set created on an existing message set. This facility enables you to prepare an empty message set that contains an XML physical format and pre-populated message set level XML properties, which are then copied into the message set that is created by the import.

Further information about XML DTDs: For details about XML DTDs, see the World Wide Web Consortium (W3C) website.

Related concepts:

“Message modeling concepts” on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

“Message definition files” on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

“Namespaces in the message model” on page 1201
Use namespaces to qualify message model object names.

Related tasks:

“Working with a message set” on page 2840
Complete a variety of tasks that are involved in working with a message set.

“Creating a message definition file” on page 2865
Creating an empty message definition file to contain your message model objects.

“Importing from XML Schema” on page 2957
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to create a new message definition from an XML Schema

“Importing from XML DTD” on page 2954
You can create a new message definition from an XML DTD by using the New Message Definition File wizard in the WebSphere Message Broker Toolkit.

“Importing from the command line” on page 2936
This describes how to use the command line importer **mqsicreatemsgdefs** to import C, COBOL copybooks, XML DTD or XML Schema in order to populate a message set with message definitions.

Importing from C header files to create message definitions:

Create a message definition file from a C header file for use in the MRM and IDOC domains.

You can populate your message set with message definitions by importing C header files, by using either the New Message Definition File From C header file wizard or the **mqsicreatemsgdefs** command.

Each C header file that you import results in a new message definition file. The root name of the message definition file defaults to the root name of the C header file, but the "New Message Definition File From C header file" wizard allows you to choose a different root file name.

If the message definition file already exists, you must have permitted overwriting to occur for the import to proceed, in which case the existing content is deleted and re-created.

By default, all message definition files that are created as a result of an import from a C header file belong to the `noTarget` XML namespace and therefore reside in the `(default)` location in your workspace. This default namespace can be overridden by specifying a target namespace. See “Namespaces with non-XML messages” on page 1206 for reasons why you might want to do this.

In your C header file there are typically one or more C structures. You can select which of these structures to import. The import operation then imports those structures, plus any others that they require. All imported structures are converted into the equivalent elements, groups, and types in the message definition file.

You can also specify which of the selected structures are to be messages in the message definition file. The import operation creates a message and a corresponding global element in the message definition file for each structure that you specify. If you do not specify that messages are to be created, you must create them manually using the Message Definition editor after the import has completed.

If you import by using the "New Message Definition File From C header file" wizard you can import only one C header file with each import operation. But, if you import by using the command-line utility, you can import several C header files in each import operation.

If your C header file needs any other header files for a successful compilation, you must provide these and specify their location, because a compilation of your header file is performed as part of the import operation.

A report file is created during the import operation. This is located by default in the `log` folder of the message set. By default, it takes the name of the message definition file, with `.report.txt` appended.

Physical information: In addition to creating logical information, the import can also create physical information.

If the message set contains any Custom Wire Format (CWF) physical formats, the physical format properties for all CWF layers are populated.

If the message set does not contain any CWF physical formats, only logical information is created. Also, if you import from the command line, only logical information is created in the new message set by default.

XML and TDS physical format properties are *not* populated and so take default values.

If you have one or more TDS layers, the import can cause entries in the task list, warning you that certain TDS properties must be set if the C structures you have imported were to appear in a TDS message.

If the TDS physical format is not applicable to your C structures, you can ignore these task list entries because they are just warnings; they do not prevent your model being generated in another form, such as a message dictionary.

Because physical information is created, the application target environment (platform and compiler) is important because it governs the way that, for example, integers appear in the message. You can specify environment-specific information as part of the import operation, and the necessary properties are set accordingly. There is a range of environments supported; if your environment is not shown, choose the closest match and review the created physical information by using the Message Definition Editor after the import has completed.

Command-line invocation: The **`mqsicreatemsgdefs`** command-line utility allows you to import several C header files in a single operation. All the C header files must be placed in the same directory and the directory location passed as a parameter to the utility.

You provide the necessary environment-specific information, and include file location information by using an XML file. This must contain an XML element

called <C> which holds one set of information that applies to all C header files imported during an invocation of the utility. A default XML file called mqsicreatemsgdefs.xml is supplied. If you want to apply different sets of information to different header files, you must create multiple XML files and run the utility multiple times.

The **mqsicreatemsgdefs** utility also allows you to create message set into which the message definition files are placed, as part of the import operation. You can also choose to base this new message set on an existing message set. This facility enables you to prepare an empty message set containing a CWF physical format and message set level CWF properties already populated, which then gets copied into the message set created by the import.

Related concepts:

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

"Message definition files" on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

"Importing from COBOL copybooks to create message definitions"
You can populate your message set with message definitions by importing COBOL copybook files, by using either the "New Message Definition File From COBOL file" wizard or the **mqsicreatemsgdefs** command.

Related tasks:

"Working with a message set" on page 2840
Complete a variety of tasks that are involved in working with a message set.

"Creating a message definition file" on page 2865
Creating an empty message definition file to contain your message model objects.

"Importing from C" on page 2934
Create a message definition file from a C header file for use in the MRM and IDOC domains.

"Importing from the command line" on page 2936
This describes how to use the command line importer **mqsicreatemsgdefs** to import C, COBOL copybooks, XML DTD or XML Schema in order to populate a message set with message definitions.

Importing from COBOL copybooks to create message definitions:

You can populate your message set with message definitions by importing COBOL copybook files, by using either the "New Message Definition File From COBOL file" wizard or the **mqsicreatemsgdefs** command.

Each COBOL copybook that you import results in a new message definition file. The root name of the message definition file defaults to the root name of the COBOL copybook file, but the "New Message Definition File From COBOL file" wizard allows you to choose a different root file name.

If the message definition file already exists, you must have permitted overwriting to occur for the import to proceed, in which case the existing content is deleted and re-created.

By default, all message definition files that are created as a result of COBOL copybook file import belong to the noTarget XML namespace and therefore reside in the (default) location in your workspace. This default namespace can be

overridden by specifying a target namespace. See “Namespaces with non-XML messages” on page 1206 for reasons why you might want to do this.

In your COBOL copybook file there are typically one or more level 01 structures. You can select which of these structures to import. The import operation then imports those structures, plus any others that they require. All imported structures are converted into the equivalent elements, groups, and types in the message definition file.

You can also specify which of the selected level 01 structures are to be messages in the message definition file. The import operation creates a message and corresponding global element in the message definition file for each structure that you specify. If you do not specify that messages are to be created, you must create them manually using the Message Definition Editor after the import has completed.

If you import by using the "New Message Definition File From COBOL file" wizard, you can import only one COBOL copybook file with each import operation. If you use the command-line utility, you can import several COBOL copybook files in each import operation.

If your COBOL copybook file needs any other copybooks in order to compile successfully, you must provide these in the same directory, because a compilation of your copybook is performed as part of the import operation.

A report file is created during the import operation. This is located by default in the `log` folder of the message set. By default it takes the name of the message definition file, with `.report.txt` appended.

The copybook must not contain field names that are COBOL reserved keywords.

Physical information: In addition to creating logical information, the import can also create physical information. If the message set contains any Custom Wire Format (CWF) physical formats, the physical format properties for all CWF layers are populated. If the message set does not contain any CWF physical formats, only logical information is created. If you import from the command line, only logical information is created in the new message set by default. If you want physical information created as well, see “Importing from the command line” on page 2936 for details.

XML and TDS physical format properties are *not* populated and therefore take default values.

If you have one or more TDS layers, the import can cause entries in the task list, warning you that certain TDS properties must be set if the COBOL structures that you have imported were to appear in a TDS message.

If the TDS physical format is not applicable to your COBOL structures, you can ignore these task list entries because they are just warnings, they do not prevent your model being generated in another form, such as a message dictionary.

Because physical information is created, the application target environment (platform and compiler) is important because it governs the way that, for example, integers appear in the message. You can specify environment-specific information as part of the import operation, and the necessary properties are set accordingly. There is a range of environments supported; if your environment is not shown,

choose the closest match and review the created physical information by using the Message Definition Editor after the import has completed.

Command-line invocation: The **mqsicreatemsgdefs** command-line utility allows you to import several COBOL files in a single operation. All the COBOL copybook files must be in single directory, and the directory location passed as a parameter to the utility.

You provide the necessary environment-specific information by using an XML file. This must contain an XML element called `<COBOL>` that holds one set of environment-specific information that applies to all COBOL copybook files that are imported during an invocation of the utility. A default XML file called `mqsicreatemsgdefs.xml` is supplied. If you want to apply different sets of information to different copybooks, you must create multiple XML files and run the utility multiple times.

The **mqsicreatemsgdefs** utility also allows you to create message set into which the message definition files are placed, as part of the import operation. You can also choose to base the message set created on an existing message set. This facility enables you to prepare an empty message set that contains a CWF physical format and pre-populated message set level CWF properties, which are then copied into the message set that is created by the import.

Related concepts:

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

"Message definition files" on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

"Importing from C header files to create message definitions" on page 1263
Create a message definition file from a C header file for use in the MRM and IDOC domains.

Related tasks:

"Working with a message set" on page 2840
Complete a variety of tasks that are involved in working with a message set.

"Creating a message definition file" on page 2865
Creating an empty message definition file to contain your message model objects.

"Importing from COBOL copybooks" on page 2937
This topic describes how to create a new message definition from a COBOL data structure using the New Message Definition File wizard in the WebSphere Message Broker Toolkit.

"Importing from the command line" on page 2936
This describes how to use the command line importer **mqsicreatemsgdefs** to import C, COBOL copybooks, XML DTD or XML Schema in order to populate a message set with message definitions.

Importing WSDL files to create message definitions:

Import WSDL files by using the **New Message Definition File From WSDL file** wizard, the **Start from WSDL and/or XSD files** Quick Start wizard, or the **mqsicreatemsgdefsfromwsdl** command.

Each WSDL file that you import results in one or more new message definition files within the message set. A new message definition file is created for each

namespace that is defined for the message set. The name of the message definition file defaults to the name of the WSDL file, but the **New Message Definition File From WSDL file** wizard allows you to choose a different file name.

If the message definition file exists, you must have permitted overwriting to occur for the import to proceed. The existing content is deleted and re-created.

The message set that you are importing the WSDL file into must be namespace-enabled. If it uses the MRM domain, it must have an XML physical format so that the message set is suitable for the runtime parsing of XML messages such as SOAP.

Use the report generated during the import operation to see what happened and to check any errors.

You specify a single WSDL definition for each import operation. If the WSDL definition consists of a hierarchy of files, you must supply the name of the file that contains the WSDL service or binding definitions. The WSDL definition that is imported must contain one or more WSDL bindings for the import to proceed. You can then select multiple bindings from the WSDL definition to define messages to model in the message set.

Importing by using the New Message Definition File wizard

When you import by using the New Message Definition File wizard, you can specify only one WSDL definition for each import operation. A WSDL definition can be held as one or more WSDL files, which are all imported as a result of importing the definition. The WSDL definition that is imported must contain one or more WSDL bindings for the import to proceed.

Importing by using the command line

The WSDL command-line importer (**mqsicreatemsgdefsfromwsdl**) can create a message set or update an existing one. If the message set project exists, it must be namespace-enabled and have an XML physical format layer. If the project does not exist, a new namespace-enabled project is created. If the import succeeds, new message definition files are added to the message set.

The **mqsicreatemsgdefsfromwsdl** command allows you to import one WSDL definition in a single operation.

The **mqsicreatemsgdefsfromwsdl** command copies the WSDL files it needs into the workspace before the import runs. These files are the top-level WSDL files and any imports are resolved by using an absolute or relative location. The files are copied under the specified message set in a folder called `importFiles`. They are not removed after the import; the user can later update or run validation on them in the WebSphere Message Broker Toolkit.

Physical information

An XML physical format layer is required for the MRM domain, and must be added to an existing message set before importing the WSDL definition.

Related concepts:

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by

your applications.

"Message definition files" on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

"Namespaces in the message model" on page 1201
Use namespaces to qualify message model object names.

Related tasks:

"Working with a message set" on page 2840
Complete a variety of tasks that are involved in working with a message set.

"Creating a message definition file" on page 2865
Creating an empty message definition file to contain your message model objects.

"Importing from WSDL" on page 2946
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to create a new message definition from WSDL.

"Importing WSDL definitions from the command line" on page 2948
WSDL definitions can be imported using the (**mqsicreatemsgdefsfromwsdl**) command.

Related reference:

"Importing from WSDL: generated objects and restrictions" on page 6355
Several objects are generated when you import from WSDL but restrictions might apply.

Relationship of WSDL to Message Model:

If a broker is to communicate with an existing Web service, it typically needs to send and receive SOAP messages. To take this approach, use the MRM domain. You must ensure that the broker message model and the WSDL definition used by the Web service describe the same messages. In general, you can achieve this result by importing the WSDL for the existing Web service by using the broker tooling. Currently only WSDL version 1.1 is supported.

Only the WSDL operation, message, and part definitions are represented in the resulting broker model. Starting with the lowest level, a WSDL definition describes the following resources:

- A number of logical messages and their constituent parts, which are defined in terms of XML Schema. The part definitions are imported directly into the message model to create the corresponding element and type definitions. The definitions are allocated to message definition files according to the target namespace of their schema definition. If no targetNamespace is defined on the <xsd:schema> element, the resulting elements and types have no namespace.
- A number of operations that form the WSDL portType or interface. The operations define the business payload for the SOAP messages at run time. Broker messages are created for each possible payload. In the case of rpc-style WSDL, the message model needs message definitions that correspond to the WSDL operations themselves. The names of these message definitions are derived from the WSDL operation name, and their namespace is given by the namespace attribute on the WSDL <soap:body> definition.
- A SOAP version and encoding style. Message definitions for the SOAP envelope and, if necessary, the SOAP encoding scheme are imported into the message set. Currently the WSDL importer assumes the use of SOAP version 1.1. WSDL version 1.1 can define a Web service that uses SOAP version 1.2, but no standard method exists to achieve this definition. If your Web service does use SOAP

version 1.2, you might have to remove the SOAP version 1.1 definitions manually and import the SOAP version 1.2 definitions.

Resulting message model

The resulting model allows the user to parse incoming SOAP messages by using the MRM XML parser where the message type is Envelope. The message model for the SOAP envelope defines the outer SOAP wrapper with its constituent Header and Body sections and a number of attachment points where the various business payloads can appear. These attachment points are defined with composition message, allowing the broker messages that are created by the WSDL importer to appear at these points.

The allowed attachment points are Envelope.Body, Envelope.Header, and Envelope.Body.Fault.detail. A message from the user's message model might appear at each point (in the case of the Envelope.Header, multiple messages might appear). In the case of rpc-style WSDL, the message expected at Envelope.Body is the automatically-generated message that corresponds to the WSDL operation. In all other cases, the messages expected are those defined by the WSDL message parts for each operation.

Related concepts:

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

"Message definition files" on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

"Namespaces in the message model" on page 1201
Use namespaces to qualify message model object names.

Related tasks:

"Working with a message set" on page 2840
Complete a variety of tasks that are involved in working with a message set.

"Creating a message definition file" on page 2865
Creating an empty message definition file to contain your message model objects.

"Importing from WSDL" on page 2946
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to create a new message definition from WSDL.

"Importing WSDL definitions from the command line" on page 2948
WSDL definitions can be imported using the (**mqsicreatemsgdefsfromwsdl**) command.

Generate model representations

After you have created and populated a message set, you can generate a message model in several different representations for use by a broker, a parser, or your applications.

The following representations are supported:

- A message dictionary, for deployment to a broker.
- A W3C XML Schema, for use by an application building or processing XML messages, or for deployment to a broker.
- Web Services Description Language (WSDL), for a Web services client application, or for deployment to a broker.
- Documentation, to give to programmers or business analysts.

Generation for deployment to a broker takes place automatically when you add your message set to a Broker Archive (BAR) file.

Generation for other purposes is achieved using the **Generate** menu actions.

Related concepts:

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

"Message definition files" on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

"Generate message dictionaries"
A *message dictionary* is data structure that describes all of the messages in a message set in a form suitable for deployment to the MRM parser.

"Generate XML schema" on page 1272
You can generate a schema file from a message model.

"Generate WSDL" on page 1274
A *Web Services Description Language* (WSDL) document specifies the interface to a web service, and enables a web service client to start it. A WSDL document that is generated from a message model defines web service requests and responses in terms of the messages that you have defined in that message model.

"Generating message model documentation" on page 1277
When you have created one or more message models, it can be useful to generate documentation for business analysis and for developers who are involved with the messages.

Related tasks:

"Working with a message set" on page 2840
Complete a variety of tasks that are involved in working with a message set.

"Generating documentation from message sets and message flows" on page 2962
You can generate documentation from your message sets, message flows, message definition files, message maps, Java files, ESQL files, and deployable WSDL files.

"Adding files to a broker archive" on page 3223
To deploy files to an execution group, include them in a broker archive (BAR) file.

Generate message dictionaries:

A *message dictionary* is data structure that describes all of the messages in a message set in a form suitable for deployment to the MRM parser.

Purpose of a message dictionary: A dictionary describes the logical structure and content of a set of messages, and typically contains one or more physical formats that describe how those messages are serialized in a bit stream. The MRM parser within WebSphere Message Broker uses this information to parse an incoming message bit stream into a message tree, or to write a message tree into a physical bit stream.

Contents of a message dictionary: A message dictionary contains the same information as the message set from which it was created, but in a compressed form that the MRM parser within WebSphere Message Broker can understand and use. A message dictionary contains all the elements in the message set, the structure of the messages, and all the value constraints. A message dictionary also contains any physical formats that were defined in the message set.

Generating a message dictionary: Before a message dictionary can be used, it must be deployed to WebSphere Message Broker. To do this, add the message set to a BAR file, then deploy the BAR file to a broker. The generation of the message dictionary is performed automatically when a message set is added to a BAR file, if the message set supports the MRM domain.

Before adding a message set to a BAR file, the WebSphere Message Broker Toolkit performs a full validation of the message set. If this validation finds any errors, the message set is not added to the BAR file, and therefore no message dictionary is generated.

Dictionary generation report files:

Whenever a message dictionary is generated, a user log file is also generated and added to the same BAR file. This file contains informational messages and warnings that relate to the use of the generated dictionary. Always check this file after generating a message dictionary.

Related concepts:

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

"Message definition files" on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

"MRM parser and domain" on page 1111
You can use the MRM domain to parse and write a wide range of message formats.

Related tasks:

"Adding files to a broker archive" on page 3223
To deploy files to an execution group, include them in a broker archive (BAR) file.

Generate XML schema:

You can generate a schema file from a message model.

XML schema is a standard way of describing complex message models.

You can generate a schema file for an individual message definition file, or for all message definition files in a message set. If any XML physical formats have been defined for the message set, you can select which of these XML wire formats are to be applied.

Generating from message sets: If any XML physical formats have been defined for the message set, you can select which of these XML wire formats are to be applied.

- If an XML format has been selected, the physical format information is also included.
- If no XML format is selected, the generated schema file contains information about only the logical message model.

You can choose whether 'strict' or 'lax' schema generation is to be performed. This is necessary because the logical extensions to the XML schema model provided by the message definition file cannot be represented in XML schema. So you can choose either to generate a schema with more strict or more lax validation than the equivalent validation performed by the message model parser. The affected model extensions are:

- `Content Validation` = open
- `Content Validation` = open defined
- `Composition` = unordered set

Further information about XML schema: For details about XML schema, see XML Schema Part 0: Primer on the World Wide Web Consortium (W3C) Web site.

Related concepts:

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

"Message definition files" on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

Related tasks:

"Importing from XML Schema" on page 2957
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to create a new message definition from an XML Schema

"Generating XML Schemas" on page 2963
You can generate either a single XML Schema from a message definition file, or multiple XML Schemas from a message set.

"Generating multiple XML Schemas" on page 2963
You can generate an XML Schema for each message definition file in a message set.

Validating an XML message against a schema:

The XMLNSC parser can validate an XML message against any deployed XML schema.

Before you begin

You can validate an XML message against an XML schema when the message is parsed, when the message is serialized, or at any point within a message flow.

About this task

To construct a message flow for schema validation, you must perform the following tasks:

Procedure

1. Enable validation at the appropriate point within the message flow. See "Validating messages" on page 1478.
2. Ensure that you have deployed all XML Schema files that are required. See "Deploying an XML Schema" on page 1274.
3. Specify the message set in which the schema was deployed; this is done using the MessageSet property of the message. See "Accessing the Properties tree" on page 2460.

What to do next

Schemas are deployed within a message set, and are identified by supplying the message set name in the message properties.

Related concepts:

"XML parsers and domains" on page 1084
You can use XML domains to parse and write messages that conform to the W3C XML standard.

"Generate XML schema" on page 1272
You can generate a schema file from a message model.

Deploying an XML Schema:

XML Schemas are created as Message Definition Files within a message set that is then deployed.

Before you begin

To create and deploy a message set for schema validation you must:

Procedure

1. Create or locate a message set that will contain the schemas.
2. Set the Message Domain property of the message set to XMLNSC.
3. Use the New Message Definition File wizard to create a message definition file (mxsd) from the XML Schema file (.xsd).
4. Add the message set to a BAR file and deploy the BAR file.

What to do next

Repeat step 3 for each XML Schema file that you want to deploy.

If your XML Schema imports or includes other XML Schema files, you can use the **mqsicreatemsgdefs** command to create all the message definition files in a single operation.

Related concepts:

"Generate XML schema" on page 1272
You can generate a schema file from a message model.

Related reference:

"New message definition file wizards" on page 6360
Use the New message definition file wizards to create message definition files.

"**mqsicreatemsgdefs** command" on page 3702
Use the **mqsicreatemsgdefs** command to create message definition files.

Generate WSDL:

A *Web Services Description Language* (WSDL) document specifies the interface to a web service, and enables a web service client to start it. A WSDL document that is generated from a message model defines web service requests and responses in terms of the messages that you have defined in that message model.

Use message model files with target namespaces when you generate WSDL. If you do not, WebSphere Message Broker uses the WSDL target namespace as the default target namespace.

In the main WSDL document, operations are defined in terms of logical messages, which are themselves defined in terms of the elements and types that are defined in these message model files.

WSDL operations are grouped into a logical interface or portType, and are then associated with a binding which defines the physical format of the messages. You can select only one of the following bindings when you generate WSDL:
- SOAP (over JMS)
- SOAP (over HTTP)

A WSDL service definition specifies the endpoint where the service is available. You can elect to have the service, binding, and portType definitions generated as a single file or as separate files. Tools that use WSDL are typically more tolerant of the single-file format. If you select single-file, you can also choose whether the associated XML Schema is generated from the message model as a separate file or in-line.

Related concepts:

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

"Message definition files" on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

"XML Schema" on page 1172
XML Schema is an international standard from the World Wide Web Consortium (W3C) that defines a language for describing the structure of XML documents.

"Generate XML schema" on page 1272
You can generate a schema file from a message model.

Related tasks:

"Generating a WSDL definition from a message set" on page 2968
To ensure the highest interoperability of your Web services, use the document style of WSDL whenever possible. If rpc style WSDL is necessary, use literal encoding.

"Importing from WSDL" on page 2946
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to create a new message definition from WSDL.

Relationship to the message model when generating WSDL:

If a broker is to communicate with a Web service client, it must typically accept SOAP messages. You can model your messages in the MRM domain, and the message model you deploy to the broker and the WSDL definition used by the Web service client must describe the same messages.

If the broker has an existing message model (possibly created by importing a C header file or COBOL copybook), you can export the model to create a corresponding WSDL definition for use by the client. At the same time, you must enhance your message model with appropriate definitions for the SOAP envelope and (for rpc-style) the WSDL operations. Currently only WSDL version 1.1 is supported.

To generate WSDL, you need:

1. A way of representing the WSDL operations. The message category performs this service.

2. Away of representing the data for these operations. The message model performs this service
3. A way of soliciting the Web service end-point and binding details. The WSDL Generator wizard performs this service.

A message category is required for each WSDL operation. The category specifies a set of messages from the broker model and associates them with the required WSDL qualifiers for the specified WSDL operation type.

At run time, the format of the SOAP messages depends on the WSDL style specified in the wizard. If you select rpc-style, the SOAP Envelope contains a message that corresponds to a WSDL operation. The WSDL generator adds an appropriate message definition that corresponds to the WSDL operation to your message set. If you select document-style, the SOAP envelope contains messages specified in the category, therefore you do not have to add additional message definitions to your message set.

Message definitions for the SOAP envelope and, if necessary, the SOAP encoding scheme, are imported into the message set.

Resulting message model

The resulting model allows incoming SOAP messages to be parsed by the MRM XML parser, where the message type would be Envelope. The message model for the SOAP envelope defines the outer SOAP wrapper with its constituent header and body sections and a number of attachment points where the various business payloads can appear. These attachment points are defined with composition of type message, allowing broker messages to appear at these points.

The supported attachment points are Envelope.Body, Envelope.Header and Envelope.Body.Fault.detail. A message from your message model might appear at each point (in the case of the Envelope.Header, multiple messages can appear). For rpc-style WSDL, the message expected at Envelope.Body is the automatically generated message that corresponds to the WSDL operation; for example, the message category. In all other cases, the messages expected are those referenced by the message categories.

Related concepts:

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

"Message definition files" on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

"Namespaces in the message model" on page 1201
Use namespaces to qualify message model object names.

"Generate WSDL" on page 1274
A *Web Services Description Language* (WSDL) document specifies the interface to a web service, and enables a web service client to start it. A WSDL document that is generated from a message model defines web service requests and responses in terms of the messages that you have defined in that message model.

Related tasks:

"Working with a message set" on page 2840
Complete a variety of tasks that are involved in working with a message set.

"Creating a message definition file" on page 2865
Creating an empty message definition file to contain your message model objects.

"Generating a WSDL definition from a message set" on page 2968
To ensure the highest interoperability of your Web services, use the document style of WSDL whenever possible. If rpc style WSDL is necessary, use literal encoding.

Generating message model documentation:

When you have created one or more message models, it can be useful to generate documentation for business analysis and for developers who are involved with the messages.

Message model schema files contain both logical and physical definitions for the message model. The generated documentation describes the logical information only.

The documentation exists as a PDF file.

Related concepts:

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

"Generate model representations" on page 1270
After you have created and populated a message set, you can generate a message model in several different representations for use by a broker, a parser, or your applications.

Related tasks:

"Generating documentation from message sets and message flows" on page 2962
You can generate documentation from your message sets, message flows, message definition files, message maps, Java files, ESQL files, and deployable WSDL files.

Message flow behavior

Message flow behavior is initially defined by the broker, but you can change or add to that behavior in some situations.

When a message is received and processed by a message flow, the broker handles both successful and failure processing. The default actions taken by the broker, which are independent of the protocol you are using to exchange messages with your message flows, are described in "Default message flow behavior" on page 1278.

If the default behavior does not meet your processing requirements in some way, you can change some aspects of the behavior by tuning or reconfiguring your message flows.

- Learn about the default behavior supported by the broker.
- Review the options that are available to you if you want to change the behavior.

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

"Execution groups" on page 53
An execution group is a named grouping of message flows that have been assigned to a broker. The broker enforces a degree of isolation between message flows in distinct execution groups by ensuring that they run in separate address

spaces, or as unique processes.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Default message flow behavior

The broker controls the behavior of your message flows, and defines what actions are taken if you do not change that behavior.

The broker provides the following default actions for all messages that are processed by your message flows:

The execution and threading model
When you deploy a message flow to an execution group, the broker allocates a certain number of threads to that message flow. For further information about how this processing works, see "Execution and threading models in a message flow" on page 1279.

Error handling
When you configure your message flows to process messages, you can include error processing of your own. The broker runs default processing if an error occurs that you have not provided additional processing for; this processing is independent of the protocol that you are using to communicate with the broker. It is described in "Default error handling" on page 1280.

Transactional support
The broker handles all messages individually, and does not relate an action taken by the message flow to other actions, processes, or messages. A message flow that is not configured to implement transactional support succeeds or fails regardless of the outcome of interactions with or updates to other resources such as WebSphere MQ queues and databases. Learn more about transactional support in "Message flow transactions" on page 1281.

Data conversion
Different hardware platforms and operating systems operate in different code pages. Operating system code pages are also affected by the locale in which you operate. The broker does not allow for these differences unless you change some aspects of its configuration. If you are exchanging messages between unlike systems, you might have to update broker or WebSphere MQ configuration, or design and supply your own conversion procedures.

For information about how you can change the configuration of your broker, its WebSphere MQ queue manager, and your message flows to influence these behaviors, see "Changing message flow behavior" on page 1288.

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

"Execution groups" on page 53
An execution group is a named grouping of message flows that have been assigned to a broker. The broker enforces a degree of isolation between message flows in distinct execution groups by ensuring that they run in separate address spaces, or as unique processes.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an

input message is received.

Related tasks:

"Changing message flow behavior" on page 1288
You can change the behavior taken by your message flows to process messages in different ways and at different times.

Execution and threading models in a message flow:

The execution model is the system used to start message flows which process messages through a series of nodes.

When an execution group is initialized, the appropriate loadable implementation library (LIL) files and Plug-in Archive (PAR) files are made available to the runtime environment. The execution group runtime process starts, and creates a dedicated configuration thread.

The message flow execution environment is conceptually like procedural programming. Nodes that you insert into a message flow are like subroutines that are called by using a function call interface. However, rather than a call-return interface, in which parameters are passed in the form of input message data, the execution model is referred to as a propagation-and-return model.

In the message flow execution environment, the message flow is thread-safe. You can run message flows concurrently on many operating system threads, without having to consider serialization issues.

Each input message that passes through a message flow for processing by a series of nodes executes on a single thread; it is processed only by the thread that received it. If you want to increase the throughput of a message flow, you can increase the number of threads that are assigned to that message flow. The memory requirements of an execution group are not unduly affected by running message flows on more operating system threads.

With a larger number of threads, the message flow can handle peak message loads. At other times, the additional threads are idle.

You can increase or decrease the number of threads servicing a flow by using the `Additional instances` property on the input node of the message flow.

Each instance of a message flow processing node is shared, and used by all the threads that service the message flow in which the node is defined.

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

"Execution groups" on page 53
An execution group is a named grouping of message flows that have been assigned to a broker. The broker enforces a degree of isolation between message flows in distinct execution groups by ensuring that they run in separate address spaces, or as unique processes.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message flow behavior" on page 1277
Message flow behavior is initially defined by the broker, but you can change or add to that behavior in some situations.

"Message flow transactions" on page 1281
A transaction describes a set of updates that are made by an application program, which must be managed together. The updates might be made to one or more systems. The updates made by the program are controlled by the environment in which the program executes, and either all are completed, or none. This property of a transaction is known as *consistency*: transactions might have other properties of *atomicity*, *isolation*, and *durability*.

"Threading considerations for user-defined extensions" on page 2978
Message processing nodes and parsers must work in a multi-instance, multithreaded environment. Many node objects or parser objects are available, each with several syntax elements, and many threads can be executing methods on these objects.

"User-defined extensions execution model" on page 2981
The execution model is the system used to start message flows through a series of nodes.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

Default error handling:

The broker provides basic error handling for all your message flows.

When an exception is detected within a node, the message and the exception information are propagated to the Failure terminal on the node. If the node does not have a Failure terminal, or it is not connected, the broker throws an exception and returns control to the closest upstream node that can process it. The default behavior is that the message is returned to the input node.

The actions taken by the input node are protocol-dependent; if your message flow starts with an MQInput, its error handling is different from the error handling provided by a FileInput node.

The actions also depend on whether you have configured the message flow to be transactional. Some nodes support transactions; others are non-transactional. If the message is not being processed in a transaction, the message is discarded by the input node. If the flow is transactional, the message is returned to its source; for example, it is restored on the WebSphere MQ queue.

If basic error processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling. For example, you can add in a sequence of nodes to deal with one or more errors that you might expect to occur in your flow. You can also configure your flow to handle unexpected errors (exceptions). For more details about the actions that you can take, see "Handling errors in message flows" on page 2823.

Related tasks:

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Message flow transactions:

A transaction describes a set of updates that are made by an application program, which must be managed together. The updates might be made to one or more systems. The updates made by the program are controlled by the environment in which the program executes, and either all are completed, or none. This property of a transaction is known as *consistency*: transactions might have other properties of *atomicity*, *isolation*, and *durability*.

WebSphere Message Broker supports transactions, and every piece of data processed by a message flow has an associated transaction. A message flow transaction is started by the broker when input data is received at an input node in the flow; it is committed when the flow has finished with that message, or rolled back if an error occurs.

If the flow contains more than one input node, one transaction is started for each input node when it receives input data. A transaction is started for every type of input node, including user-defined input nodes.

The nodes that you include in your message flow provide specific processing of a message, according to the defined function of each node. The processing that they do includes internal work, some of which you can influence by configuring the node properties. Some nodes perform additional tasks that might affect systems that are external to the message flow or the broker.

If an external system, such as a database, supports that concept of commit and rollback, and it can take part in a broker transaction, you can configure the node so that the work it does is included in the flow transaction. Depending on the node, you can also specify if the work done in an external system that supports transactions is committed immediately, or when the message flow transaction completes.

Many of the resources with which your message flows can interact are controlled by resource managers that can participate in coordinated transactions; for example, databases, WebSphere MQ messages and queues, and JMS messages. Other resource managers do not provide transactional support; for example, the HTTP protocol and file systems.

Commit or rollback

If the resource can participate in a transaction, you can configure it so that the work it does is committed or rolled back only when the message flow completes, or when the node completes. Databases and WebSphere MQ queues are examples of resources that you can use in this way. If the resource does not have transactional behavior, all the work that it does is committed immediately. For example, files and HTTP connections do not support transactions.

Updates that are made by a message flow are committed when the flow processes the input message successfully. The updates are rolled back if both of the following conditions are met:

- A node in the flow throws an exception that is not caught by a node other than the input node (for example, the node itself, or a TryCatch node)
- The Catch terminal of the input node is not connected

On distributed systems, message flow transactions are managed by the broker, by default. These transactions are known as *broker-coordinated transactions* or *partially coordinated transactions*. When control returns to the input node when the flow finishes processing, the node either commits or rolls back the operations that have been taken, excluding the individual nodes that have been configured to perform their own commits and rollbacks, or that have no support for this option.

If more than one resource is accessed by the message flow, an error might occur that prevents all the resources committing all the work that has been done. The broker raises an exception, and handles exception processing in a way that is determined by the transport that is involved. For example, messages that were read from WebSphere MQ queues are restored to those queues, and fault messages are sent to applications that submitted a message across HTTP (because HTTP has no concept of rollback). Because of these actions, the status of the resources might become inconsistent.

If it is important that your data and operations remain consistent, and that all operations are committed, or rolled back if one or more operations fail, you can coordinate the activity of the message flow. Coordination is provided by an external transaction manager which uses XA protocols to interact with resource managers. The transaction manager is called by the input node when the message flow has concluded (successfully or with errors). The transaction manager, rather than the input node and the broker, interacts with the relevant resource managers to initiate the correct actions for each resource. Transactions that are controlled by a transaction manager in this way are known as *globally coordinated transactions*.

On z/OS, transactions are always globally coordinated; you do not have to choose or configure for this option.

For a more detailed description of the transaction model in the broker, see “The transactional model” on page 1285.

The role of the transaction manager

The result of the actions taken by message flows is the same for both partially and globally coordinated transactions if the message flow is successful in all its actions. The advantage of a globally coordinated transaction is the ability to ensure that either all actions are committed, or none.

The external transaction manager, which operates a two-phase commit strategy, supports cases where one or more external resource managers are temporarily unavailable during commit processing. This potentially small window for failure might be costly for your business environment; the external transaction manager helps to eliminate the occurrence of a failure window. Therefore, the decision to include an external transaction manager, which involves a performance overhead, is an administrative decision, not one to be taken at message flow design time.

An external transaction manager does not prevent message loss; even if you use transaction coordination, you must configure and code your message flows to handle potential errors as much as you can.

To configure a message flow to be globally coordinated, you must also set up your environment so that your resource managers are defined to the supported transaction manager:

- On distributed systems, transactions can be coordinated by WebSphere MQ
- On z/OS, all transactions are globally coordinated by Resource Recovery Service (RRS)

This configuration might require you to change settings in the transaction manager as well as the participating resource managers.

Database access modes and locks

You must use separate ODBC connections if you want to include nodes with Automatic transaction status and nodes with Commit transaction status in the same message flow, where the nodes operate on the same external database. Set up one connection for the nodes that are not to commit until the completion of the message flow, and a second connection for the nodes that are to commit immediately.

- If nodes with Commit transaction status are followed by a node with Automatic transaction status, the nodes with Commit transaction status commit independently of the flow transaction, and the nodes with Automatic transaction status commit at the end of the flow.
- However, if nodes with Automatic transaction status are followed by a node with Commit transaction status, and you do not use separate ODBC connections, error message BIP4001 is issued, because otherwise the node with Commit transaction status commits the work of the Automatic nodes prematurely.

Linux UNIX Windows On systems other than z/OS, individual relational databases might not support this mode of operation.

If you define more than one ODBC connection to the same data source, you might get database locking problems. In particular, if a node with Automatic transaction status carries out an operation, such as an INSERT or an UPDATE, that causes a database object (such as a table) to be locked, and a subsequent node tries to access that database object by using a different ODBC connection, an infinite lock (deadlock) occurs.

The second node waits for the lock acquired by the first to be released, but the first node does commit its operations and release its lock until the message flow completes. However, the flow cannot complete because the second node is waiting for the database lock held by the first node to be released.

Such a situation cannot be detected by a DBMS automatic deadlock-avoidance routine because the two operations are interfering with each other indirectly by using the broker.

You can use either of two options to avoid this type of locking problem:

- Design your message flow so that uncommitted (automatic) operations do not lock database objects that subsequent operations access across a different ODBC connection.

- Configure the lock timeout parameter of your database so that an attempt to acquire a lock fails after a specified length of time. If a database operation fails because of a lock timeout, an exception is thrown that the broker handles in the typical way.

For information concerning which database objects are locked by particular operations, and how to configure the lock timeout parameter of your database, consult your database product documentation.

Example of transactions in message flows

The following sample demonstrates the use of globally coordinated transactions, and the differences in the message flow when database updates are coordinated (the main flow), and when they are not (the error flow).

- Error Handler

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Configuring transactionality for message flows" on page 1290
A message flow runs in a single transaction, which is started when data is received by an input node, and can be committed or rolled back when all processing has completed.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"Database connections for coordinated message flows" on page 4235
When you configure a message flow to access a database, the broker establishes a connection to that database based on the ODBC DSN.

"Database support for coordinated message flows" on page 4236
If the message flow processing includes interaction with an external database, you can coordinate the transaction by using XA technology.

The transactional model:

The transactional model describes the way in which you can use transactions in message flows to accomplish certain tasks and results.

A message flow consists of the following constituent parts:

- An input source
- The message flow or logic, which is defined by a sequence of nodes
- Zero or more external resources that are accessed during the flow
- Zero or more output targets

The following steps represent a typical sequence of events in the message flow transaction:

1. A message is taken from the input source; for example, a queue.
2. Data is read from or written to one or more external resources; for example, a database.
3. A message is sent to an output target; for example, a queue.
4. The system quiesces and waits for the next input message.

During this sequence of events, the state of the data in the system changes, regardless of the number of external resources that the message flow accesses, and whether it generates an output message.

Consider the following diagram:

```
-----x---------x---x-------x-------------x----x-----
     1         2   3       4             5    6
```

The line represents the data in the system as time passes. At time 1, a message arrives and is taken from the input source. At times 2, 3, 4, and 5, data is used to update external resources; for example, a database table or an output queue. Changes in the state of the data are indicated in the diagram by the *x* symbol. At time 6, the output messages are sent and the system is inactive. Between these events, the state of the data does not change; this state is indicated in the diagram by the = symbol.

If a failure occurs on the system (for example, a loss of power to the computer on which the broker is running), the changes to the state of external resources that were made before the failure have been implemented, but no more changes take place after the failure. This situation is unacceptable in certain circumstances; for example, if a system failure occurs when making a payment from a current account to a mortgage account, the payment might be taken from the current account, but it is not added to the mortgage account.

Transactions: To avoid the problem that is described previously, the broker and the external resource managers with which it works, have a *transactional model*. The broker starts a transaction when data is received by an input node in the message flow, and completes when the processing on that data is finished. For more details about message flow transactions, see “Message flow transactions” on page 1281.

As processing proceeds in a transaction, additional data is recorded that allows the original state to be restored in the event of a failure. The following diagram illustrates the state of this extra data:

```
-----x=========x===x=======x=============x====x-----
     1         2   3       4             5    6
```

The line in the diagram represents the extra data in the system as time passes. At time 1, input data arrives from the input source; for example, a queue. Before time 1, no extra data exists in the system; this state is indicated in the diagram by the - symbol. After time 1, the state represents the fact that data has been received from the input source, so that it can be restored, if necessary. At times 2, 3, 4, and 5, data is used to update external resources such as databases or files. Again, the state of the extra data changes so that the changes to those external resources can be undone, if necessary. At time 6, the output messages are sent, the system is inactive, and extra data in the system no longer exists.

Between these events, the state of the extra data does not change; this state is indicated by the = symbol. If a failure occurs between time 1 and time 6, the extra data is used to restore the original state of the data held by the external resources. Therefore, effectively, no output data has been written to the output target, none of the external resources have been updated, and the input data has not been received from the input source. If no failure occurs, the changes become permanent at time 6 (an undo operation that follows a subsequent failure will not undo the changes).

This mode of operation is known as *coordinated transaction mode*. The successful completion of a transaction is known as its *commit*. Unsuccessful completion is known as *rollback*.

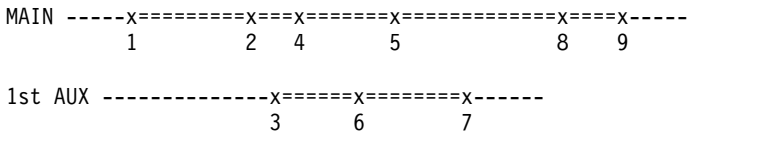
Uncoordinated auxiliary transactions: The key feature of the coordinated transaction mode of operation is that, regardless of where or when the failure appears, either all of the changes to external resources that are associated with one input message are made, or none of the changes are made. However, this behavior is not always suitable, as the following examples illustrate:

- You want to create an audit log of all attempts at processing. The log entries must be committed, even when updates to other resources are rolled back.
- You want to send an acknowledgment or non-acknowledgment message back to the originator of the messages that you are processing, according to whether the message processing succeeds or fails. These messages must be sent even when the updates to other resources are rolled back.

If your message flows have requirements like these, you can configure message flows to change one or more resources in a separate, or auxiliary, transaction. Not all resource managers support this type of transaction.

For some resources, an auxiliary transaction is automatically started; for example, each database connection starts a transaction that is specific to that database, and all updates made in that transaction can be committed or rolled back.

The behavior of an auxiliary transaction is shown in the following diagram:



The *MAIN* line represents the main transaction, which includes the extra data that is recorded to restore the original state if necessary. The *1st AUX* line represents an auxiliary transaction. At time 3, an external resource is updated, and another update is made at time 6. At time 7, the message flow determines that all the changes that must be made under the auxiliary transaction are complete, and it commits the changes.

If the message flow fails before time 7, the state of the system would be unchanged because both transactions would be rolled back. If failure occurs after time 7 but before time 9, the auxiliary transaction would already have been committed. However, the main transaction would be rolled back. If a failure has not occurred by time 9, both transactions are committed.

Database auxiliary transactions: You can use more than one auxiliary transaction, and make a number of updates to database tables that can be committed or rolled back. You can then make additional changes to the same database tables, or to different tables, then commit or rollback these changes.

Each database that you use has its own auxiliary transaction; therefore, if the message flow updates tables that belong to different database instances (different data source names), an auxiliary transaction exists for each database. You can optionally commit or roll back these auxiliary transactions individually. Updates that have not been committed or rolled back when the message flow completes (at time 9 in the example shown previously) are committed or rolled back automatically by the broker, according to whether the processing succeeded or failed.

Use the ESQL COMMIT and ROLLBACK statements to commit and roll back auxiliary database transactions. Obtain operations outside the main transaction by specifying the UNCOORDINATED keyword on the individual database statements (for example, the INSERT and UPDATE statements).

Queue auxiliary transactions: Not all queuing systems have the database capability that is described in the previous section. With WebSphere MQ, each individual uncoordinated read or write operation to a queue has an implied commit action. Therefore, you cannot put two messages, then decide to commit both or roll back both. The COMMIT and ROLLBACK statements therefore operate only on databases.

Nodes: The previous sections refer to message flows, but not to nodes. The way in which a message flow is divided into nodes has no effect on transactions. For operations on databases, an unlimited number of nodes can make updates to the main transaction, and to an unlimited number of auxiliary transactions, without restriction.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message flow transactions" on page 1281
A transaction describes a set of updates that are made by an application program, which must be managed together. The updates might be made to one or more systems. The updates made by the program are controlled by the environment in which the program executes, and either all are completed, or none. This property of a transaction is known as *consistency*: transactions might have other properties of *atomicity*, *isolation*, and *durability*.

Related tasks:

"Configuring databases for global coordination of transactions" on page 665
If your message flow interacts with a user database, and you want to globally coordinate the updates made to the database with other actions within the message flow, configure your databases for global coordination.

"Configuring transactionality for message flows" on page 1290
A message flow runs in a single transaction, which is started when data is received by an input node, and can be committed or rolled back when all processing has completed.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"Database connections for coordinated message flows" on page 4235
When you configure a message flow to access a database, the broker establishes a connection to that database based on the ODBC DSN.

"Database support for coordinated message flows" on page 4236
If the message flow processing includes interaction with an external database, you can coordinate the transaction by using XA technology.

Changing message flow behavior

You can change the behavior taken by your message flows to process messages in different ways and at different times.

About this task

If one or more of the default behaviors described in "Default message flow behavior" on page 1278 is not sufficient or appropriate for your message processing, you can change some characteristics of that behavior at different times during the design, development, and production cycles:

- "Behavior you can change when you design your message flows"
- "Behavior you can change when you deploy your message flows" on page 1289
- "Behavior you can change during message flow processing" on page 1289

Behavior you can change when you design your message flows:
About this task

You can influence the behavior of your message flows when you set message node properties, and when you connect together the nodes that you select to run in that flow.

Transactional support

You can configure your message flows to handle your messages, and other data, in transactions. These options are described in "Configuring transactionality for message flows" on page 1290.

Error handling

The broker provides an initial level of error processing for all message flows. If you require further support in your message flows, you can add your own processing. You can learn more about these options in "Handling errors in message flows" on page 2823.

The nodes that support some protocols provide additional default error handling; this support is described, where relevant, in the sections in "Connecting client applications" on page 1537.

Data conversion

If you are exchanging messages between unlike systems, you can update broker or WebSphere MQ configuration, or design and supply your own conversion procedures. Options are described in "Configuring message flows for data conversion" on page 1293.

User-defined properties

You can create user-defined properties for your message flows to associate values with those message flows. You can then configure the nodes in your message flows to access those properties and their values by coding ESQL or Java programs. Read more about user-defined properties, and see how to create them in the "Message Flow editor" on page 6810.

Promoted properties

You can promote some message node properties to the level of the message flow in which the node is included. The advantages of this technique are described in "Promoted properties" on page 1145; see "Defining a promoted property" on page 1297 for details of how to use these properties.

Behavior you can change when you deploy your message flows: About this task

After you have decided on the message flow content, you can change some aspects of its operation before or after deployment.

Configurable properties and BAR overrides

Some of the properties on message flow nodes are configurable; that is, you can change their values when you deploy the message flow. By using this option, you can change some characteristics of a deployed message flow without changing the message flow definitions. For example, you can update queue manager and data source information in the BAR file when you deploy it.

You can change these values by using the WebSphere Message Broker Explorer, the WebSphere Message Broker Toolkit, or the **mqsiapplybaroverride** command.

The execution and threading model

You can increase the number of threads that your message flows use to reduce the time in which your messages are handled. You can also deploy multiple copies of a message flow to one or more brokers. For further information, see "Optimizing message flow throughput" on page 587.

Behavior you can change during message flow processing: About this task

If you want to change the behavior of your message flows in a more dynamic way, you can use the techniques described here. However, you must design your message flows so that they can take advantage of these additional services.

Configurable services

By using a configurable service, you can create and configure properties that relate to external services called by the broker from your message flows. A number of configurable services are supplied with the broker; you can modify or add to these default services, and you can create your own services.

Learn more about this option in “Configurable services” on page 1296. You can use the WebSphere Message Broker Explorer to work with configurable services; several commands are also supplied to create, view, and modify these services.

Local environment overrides

You can configure some nodes to include your own processing; by coding ESQL, Java, PHP, or maps, you can modify the contents of the local environment tree within a message. Some fields in the local environment are used by nodes to determine how the message is processed, therefore by changing the tree contents, you can influence the behavior of subsequent nodes in the message flow. For more details about this option, see “Transforming and enriching messages” on page 2227.

Related concepts:

“The broker environment” on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

“Execution groups” on page 53
An execution group is a named grouping of message flows that have been assigned to a broker. The broker enforces a degree of isolation between message flows in distinct execution groups by ensuring that they run in separate address spaces, or as unique processes.

“Message flows overview” on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

“Default message flow behavior” on page 1278
The broker controls the behavior of your message flows, and defines what actions are taken if you do not change that behavior.

Configuring transactionality for message flows:

A message flow runs in a single transaction, which is started when data is received by an input node, and can be committed or rolled back when all processing has completed.

Before you begin

Before you start:

Ensure that you have completed the following tasks.

- To ensure that you understand how the broker handles transactions, read “Message flow transactions” on page 1281.
- Create a message flow by following the instructions in “Creating a message flow” on page 1431.

About this task

How individual nodes, and the message flow itself, participate in transactions depends on the way you design and develop the message flow, and the level of additional configuration you perform:

1. Configure the node properties in your message flow to set the required level of participation in transactions.
2. If you want the updates made by the message flow to be globally coordinated by an external transaction manager, configure the message flow properties.

What to do next

When you have finished message flow design and development, you can deploy the BAR file to the broker or brokers on which you want the message flow to run. However, if you have configured your message flows for globally coordinated transactions, additional configuration might be required. You, or your system administrator, must ensure that your broker environment, the transaction manager, and the participating resource managers are all correctly configured to support coordinated transactions, before you run the message flow. For details of what might be required, see "Configuring global coordination of transactions (two-phase commit)" on page 697.

If the broker environment, the transaction manager, and the external resource managers are not correctly configured for global coordination, the message flow transactions will not be globally coordinated.

Configuring node properties:
About this task

You can configure the nodes in your message flow to determine how the work taken by each node participates in the message flow transaction. Most nodes for which transactionality is relevant have one or more properties that you can configure to dictate behavior. Therefore, you can decide for each individual node whether it participates in the message flow transaction, or operates independently. Typically, these properties include an option of Automatic, so that subsequent nodes in the flow assume the characteristics set by the input node.

Nodes that support transports that cannot participate in transactions might have other properties to determine what the broker does when a message flow failure occurs. For example, the FileInput node has a set of Retry properties that you can set to determine failure behavior.

A few nodes that interact with external resources do not provide properties; typically, these nodes are included in the message flow transactions, but some exceptions exist; you must check the section that describes properties and how to set them, for each node that you include in your flow to ensure that you understand what action is taken.

If you configure a node not to participate in the message flow transaction, the actions that it takes are committed, or rolled back, when the node exits. No further action is taken when the flow itself completes.

To configure message flow behavior by setting node properties:

Procedure

1. Open the message flow that you want to configure.
2. Set the `Transaction mode` property for the input nodes in this message flow. The value that you set determines the behavior of the input node, and sets the default behavior for the rest of the message flow. Typically, you can choose the value Yes or No;
   - Yes means that the input node completes its own operation under sync point, and the default behavior in the message flow is for actions to be taken under sync point.

- No means that the input node completes its own operation out of sync point, and the default behavior in the message flow is for actions to be taken out of sync point.

Some nodes have additional or alternative values; for example, you can set the property on the MQInput node to Automatic, which means that the node gets the message under sync point if the message is persistent, and out of sync point if it is non-persistent.

For details of the specific options for and actions taken by each node, see the relevant node description; the properties, the tabs they are defined on, and the resulting behavior are not identical across all input nodes.

3. If your message flow includes nodes that interact with external resources, including output, request, and reply nodes, you can set a transaction property on most of these nodes.

   Set the property only if you want to change the behavior of the individual node from the default behavior for the message flow, which you set on the input node. The value that you set on this node has no effect on subsequent nodes in the message flow. If the node does not have a transactional property, its behavior is governed by the default behavior for the message flow, which you set in the input node.

   If your message flow is updating a database from multiple nodes in a single message flow, read the conceptual information about message flow transactions to understand the possible interactions.

   a. Set the `Transaction` property for each node, if supported.
   b. Set the properties that define how errors are handled, if supported. For example, for nodes like the Compute node that can access databases, set the `Treat warnings as errors` and `Throw exception on database error` properties to define how that node handles database warnings and errors. Whether you select these properties, and how you connect the failure terminals of the nodes, also affect the way in which database updates are committed or rolled back.

Configuring message flow properties:
About this task

When you have configured your message flow, you must add it to a BAR file before you can deploy it. When you add it to a BAR file, the message flow is compiled, and additional properties are available for configuration.

The most important property concerned with transactions on distributed systems is Coordinated Transaction. By default, this property is cleared (not selected), which means the message flow is partially coordinated and the broker commits or rolls back the message flow transaction. If you select this property, the input node calls the external transaction manager WebSphere MQ for commit and rollback processing.

This property is ignored when the message flow is deployed to a broker that is running on a z/OS system.

To configure message flow properties:

Procedure

1. Add the message flow to a broker archive.

2. Select the **Manage and Configure** tab below the broker archive editor view, and select the message flow. The configurable properties for the message flow in the broker archive are displayed in the **Properties** view.

   Select `coordinatedTransaction` to configure the message flow as globally coordinated; when you set this property, the external transaction manager (WebSphere MQ) coordinates the transaction with all the resource managers that you have defined to the queue manager.

   z/OS On z/OS, transactions are always globally coordinated. The setting of the `coordinatedTransaction` property for a message flow is ignored. Coordination is provided by the transaction manager RRS.

Related concepts:

"Message flow transactions" on page 1281
A transaction describes a set of updates that are made by an application program, which must be managed together. The updates might be made to one or more systems. The updates made by the program are controlled by the environment in which the program executes, and either all are completed, or none. This property of a transaction is known as *consistency*: transactions might have other properties of *atomicity*, *isolation*, and *durability*.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

"Configuring global coordination of transactions (two-phase commit)" on page 697
Globally coordinate message flow transactions with a transaction manager to ensure the data integrity of transactions. On distributed systems, the WebSphere MQ queue manager that is associated with the broker performs the transaction manager role.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Adding files to a broker archive" on page 3223
To deploy files to an execution group, include them in a broker archive (BAR) file.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Configuring message flows for data conversion:

If you exchange messages between applications that run on systems that are incompatible in some way, you can configure your system to provide data conversion as the message passes through the broker.

About this task

Data conversion might be necessary if either of the following two values are different on the sending and receiving systems:

1. CCSID. The Coded Character Set Identifier refers to a set of coded characters and their code point assignments. WebSphere Message Broker can process and construct application messages in any code page for which WebSphere MQ provides conversion to and from Unicode, on all operating systems. For more information about code page support, see the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online.

   This behavior might be affected by the use of other products in conjunction with WebSphere Message Broker. Check the documentation for other products, including any databases that you use, for further code page support information.
2. Encoding. This setting defines the way in which a machine encodes numbers; that is, binary integers, packed-decimal integers, and floating point numbers. Numbers that are represented as characters are handled in the same way as all other string data.

If the native CCSID and encoding on the sending and receiving systems are the same, you do not need to call data conversion processes.

WebSphere Message Broker and WebSphere MQ provide data conversion facilities to support message exchange between unlike systems. Your choice of which facilities to use depends on the characteristics of the messages that are processed by your message flows:

- Messages that contain text only
- Message that include numerics
- Messages that are self-defining

Messages that contain text only

Read this section if your messages are WebSphere MQ messages that contain all text (character data or string).

If WebSphere MQ supports the systems on which both sending and receiving applications are running for data conversion, use WebSphere MQ facilities which provide the most efficient data conversion option.

The default behavior of WebSphere MQ is to put messages to queues specifying the local system CCSID and encoding. Applications issuing MQGET can request that the queue manager provides conversion to their local CCSID and encoding as part of get processing.

To use this option:

1. Design messages to be text-only. If you are using COBOL, move numeric fields to USAGE DISPLAY to put them into string form.
2. Set the Format field in the MQMD to MQFMT_STRING (value MQSTR).
3. Call MQGET with MQGMO_CONVERT in the receiving application. If you prefer, you can convert when the message is received by the broker, by setting the Convert property of the MQInput node to yes (by selecting the check box).

If you require more sophisticated data conversion than WebSphere MQ provides in this way (for example, to an unsupported code page), use WebSphere MQ data conversion exits. For more information about these, see the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online.

Messages that include numerics

Read this section if your messages include numeric data, or are text only but are not WebSphere MQ messages.

If these messages can be predefined (that is, their content and structure is known and predictable), use the facilities provided by WebSphere Message Broker and the MRM.

All application messages are handled by the broker in Unicode, to which they are converted on input, and from which they are converted on output. You can configure message flows to influence the way in which output messages are constructed.

To use this option:

1. Define the output message in the MRM domain. You can create this definition in one of the following ways:
   - Import an external message definition (for example a C header or COBOL copybook).
   - Create the message model in the message definition editor.
2. Configure a message flow to receive and process this message:
   a. If you include an MQInput node, do not request conversion by this node.
   b. Include a Compute node in the message flow to create the output message with the required content:
      - If the output message is a WebSphere MQ message, code ESQL in the Compute node to set the CCSID and encoding for the target system in the MQMD.

        For example, to set values for a target z/OS system running with CCSID of 37 and encoding of 785:

        ```
        SET OutputRoot.MQMD.CodedCharSetId = 37;
        SET OutputRoot.MQMD.Encoding = 785;
        ```

      - If the output message is not a WebSphere MQ message, code ESQL in the Compute node to set the CCSID and encoding for the target system in the Properties folder.

Messages that are self-defining

Read this section if your messages are self-defining.

Self-defining messages are supported in the XML and JMS domains. These messages are all text and can be handled by WebSphere MQ, if they originate from, or are destined for, WebSphere MQ applications. If not, use WebSphere Message Broker facilities by setting the CCSID and Encoding fields in the Properties folder in the message when it passes through a Compute node.

Related concepts:

"Predefined and self-defining messages" on page 1076
Both predefined and self-defining messages are supported.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Nodes for connectivity" on page 1028
WebSphere Message Broker supports direct connections from applications, and can send direct requests to other application endpoints. WebSphere Message Broker can also connect to various subsystems including WebSphere MQ, files, and databases, to read and write existing application data.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Converting code page and message encoding" on page 2476
You can use ESQL within a Compute node to convert data for code page and message encoding.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Connecting client applications" on page 1537
Connect your client applications to the broker by using one or more of the supported protocols from other resources and software servers in your network.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"Supported code pages" on page 4176
Application messages must conform to supported code pages.

Related information:

WebSphere MQ Version 7 Information Center online

Configurable services:

Configurable services are typically runtime properties. You can use them to define properties that are related to external services on which the broker relies; for example, an SMTP server or a JMS provider.

Instead of defining properties on the node or message flow, you can create configurable services so that nodes and message flows can refer to them to find properties at run time. If you use this method, you can change the values of attributes for a configurable service on the broker, which then affects the behavior of a node or message flow without the need for redeployment.

Unless it is explicitly stated by the function that is using the configurable service, you need to stop and start the execution group for the change of property value to take effect.

You can use the WebSphere Message Broker Explorer to view, add, modify and delete configurable services, see "Using the WebSphere Message Broker Explorer to work with configurable services" on page 644.

Alternatively, use the following commands to work with configurable services:

- Use the **mqsicreateconfigurableservice** command to create configurable services.
- Use the **mqsideleteconfigurableservice** command to delete configurable services.

- Use the **mqsichangeproperties** command to set attributes after you have created the configurable services.
- Use the **mqsireportproperties** command to report attributes.

For a full list of configurable services and their properties, see "Configurable services properties" on page 3766.

Related concepts:

"Using the WebSphere Message Broker Explorer to work with configurable services" on page 644
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view, add, modify and delete configurable services.

Related reference:

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

Defining a promoted property:

When you create a message flow, you can promote properties from individual nodes in that message flow to the message flow level. Properties promoted in this way override the property values that you have set for the individual nodes.

Before you begin

Before you start:

Read the concept topic about promoted properties.

About this task

You can perform the following tasks related to promoting properties:
- "Promoting a property" on page 1298
- "Renaming a promoted property" on page 1302
- "Removing a promoted property" on page 1304
- "Converging multiple properties" on page 1306

Some of the properties that you can promote to the message flow are also configurable; you can modify them when you deploy the broker archive file in which you have stored the message flow to each broker. If you set values for configurable properties when you deploy a broker archive file, the values that you

set override values set in the individual nodes, and those that you have promoted.

Related concepts:

"Promoted properties" on page 1145
A promoted property is a message flow node property that has been promoted to the level of the message flow in which it is included.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Promoting a property:

You can promote a node property to the message flow level to simplify the maintenance of the message flow and its nodes, and to provide common values for multiple nodes in the flow by converging promoted properties.

Before you begin

Before you start:
- Create a message flow, as described by "Creating a message flow" on page 1431
- Read the concept topic about promoted properties

About this task

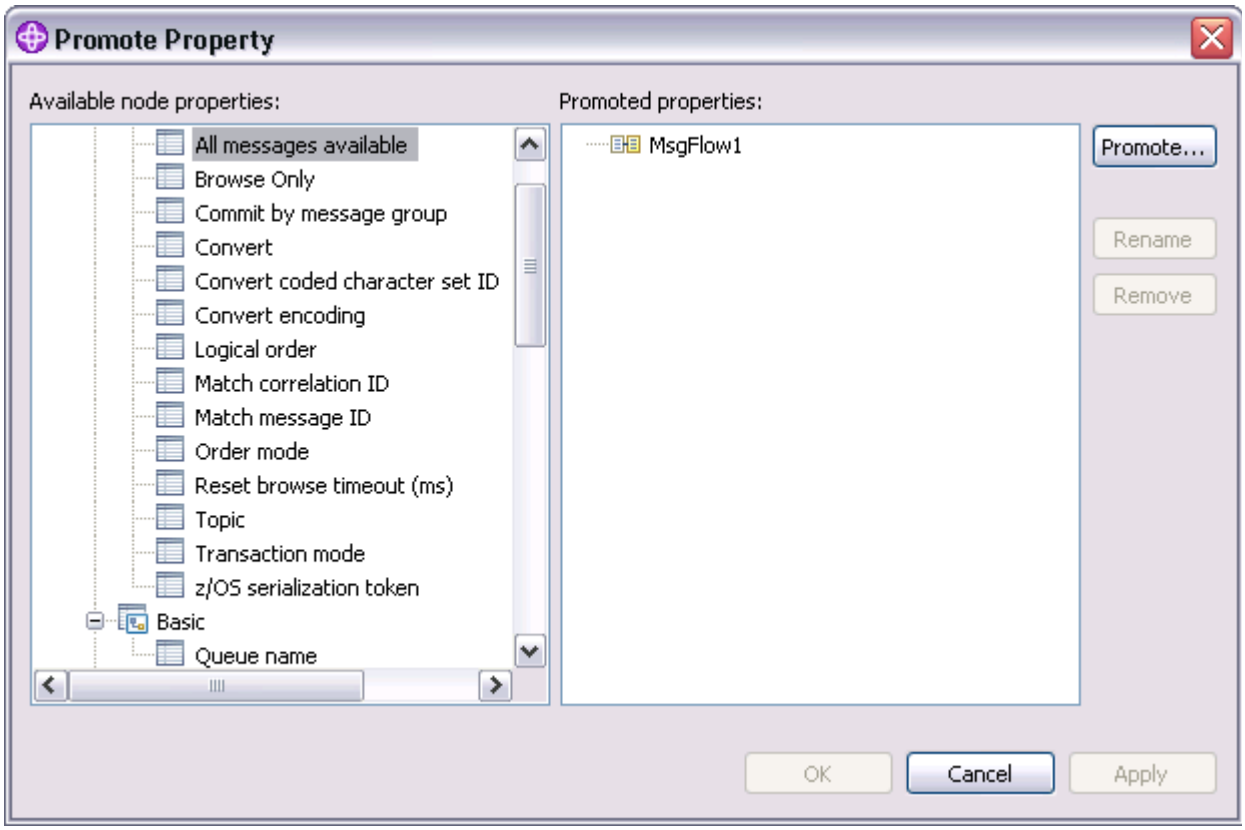
The majority of message flow node properties are available for promotion, but you cannot promote the following properties:
- Properties that name mapping modules
- A property group (but you can promote an individual property)
- A property that you cannot edit (for example, the `Fix` property of the MQInput node)
- The description properties (`Short Description` and `Long Description`)
- Complex properties (for example, the `Query elements` property of the DatabaseRoute node, or the `Opaque elements` property of the MQInput and several other nodes)

To promote message flow node properties to the message flow level, complete the following steps.

Procedure

1. Switch to the Broker Application Development perspective.
2. Open the message flow for which you want to promote properties.
3. Right-click the appropriate node and click **Promote Property**.

The Promote Property dialog box is displayed.



The Available node properties pane lists all available properties for all the nodes in the message flow. The properties for the node that you clicked are expanded. You can expand the properties for all the nodes in the open message flow, regardless of the node that you clicked initially.

The Promoted properties pane displays the name of the open message flow and all the properties that are currently promoted to the message flow. If you have not yet promoted any properties, only the message flow name is displayed as the root of the promoted property tree, as shown in the previous example. If you have already promoted properties from this node, the properties appear in the Promoted properties pane, but not in the Available node properties pane.

4. Select the property or properties that you want to promote to the message flow. You can select multiple properties by holding down Ctrl and selecting the properties.
5. Click **Promote**. The Target Selection dialog box opens and displays valid targets for the promotion.
6. Select the destination group or property for the properties that you want to promote. You can group together related properties from the same or different nodes in the message flow by dropping the selected properties onto a group or property that already exists, or you can create a new target for the promotion by clicking **New Group** or **New Property**. You can rename groups and properties by selecting them and clicking **Rename**.
7. Click **OK** to confirm your selections and close the Target Selection dialog box.

   If you create a new group or property by using the Target Selection dialog box, the changes persist even if you select **Cancel** in the dialog box. When the dialog box closes, groups or properties that you have created by using the

Target Selection dialog box appear in the Promote Property dialog box. You can remove any of these properties from the Promote Property dialog box by selecting them and clicking **Remove**.

8. Click **OK** to commit your changes and close the Promoted Property dialog box. If you click **Apply**, the changes are committed but the dialog box remains open.

Results

The message flow node properties are promoted to the message flow. When you have promoted a property, you can no longer make any changes to that property at the node level; you can update its value only at the message flow level. To view the message flow's properties, click the message flow (not the individual nodes) in the Message Flow editor to display the properties in the Properties view. The properties that you have promoted are organized in the groups that you created. If you now set a value for one of these properties, that value appears as the default value for the property whenever the message flow is included in other message flows.

When you select an embedded message flow in another message flow (a subflow) and view its properties, you see the promoted property values. If you look inside the embedded flow (by selecting **Open Subflow**), you see the original values for the properties. The value of a promoted property does not replace the original property, but it takes precedence when you deploy the message flow.

Promoting properties by dragging:
About this task

You can also promote properties from the Promote Property dialog box by dragging the selected property or properties from the Available node properties pane of the Promote Property dialog box to the Promoted properties pane, as described in the following steps.

Procedure

1. Select the property that you want to promote. You can select multiple properties by holding down Ctrl, and selecting the properties.
2. You can promote the selected properties using the following methods:
   - Drop the selected property or properties in an empty space.

     A new group is created automatically for the message flow, and the property is placed in it, with the original name of the property and the name of the message flow node from which it came displayed beneath the property entry.

     The name of the first group that is created is Group1 by default. If a group called Group1 already exists, the group is given the name Group2, and so on. You can rename the group by double-clicking it and entering new text, or by selecting the group in the Promoted properties pane and clicking **Rename**.

     When you create a new promoted property, the name that you enter is the name by which the property is known within the system, and must meet certain Java and XML naming restrictions. These restrictions are enforced by the dialog box, and a message is displayed if you enter a name that includes a non-valid character. For example, you cannot include a space or quotation marks (").

     If you are developing a message flow in a user-defined project that will be delivered as an Eclipse plug-in, you can add translations for the promoted properties that you have added. Translated names can contain characters,

such as space, that are restricted for system names. The option to provide translated strings for promoted properties is not available if you are working with a message flow in a message flow project.

- Drop the selected property or properties onto a group that already exists, to group together related properties from the same or different nodes in the message flow.

  For example, you might want to group all promoted properties that relate to database interactions. You can change the groups to which promoted properties belong at any time by selecting a property in the Promoted properties pane and dragging it onto a different group.
- Drop the selected property or properties onto a property that already exists, to converge related properties from the same or different nodes in the message flow.

  For example, you might want to create a single promoted property that overrides the property on each node that defines a data source.

  For more information on converging properties, see "Converging multiple properties" on page 1306.

Promoting mandatory properties:
About this task

If you promote a property that is mandatory (that is, an asterisk appears beside the name in the Properties view), the mandatory characteristic of the property is preserved. When a mandatory property is promoted, its value does not need to be set at the node level. If the flow that contains the mandatory promoted property is included as a subflow in another flow, the property must be populated for the subflow node.

Promoting properties through a hierarchy of message flows:
About this task

You can repeat the process of promoting message flow node properties through several levels of message flow. You can promote properties from any level in the hierarchy to the next level above, and so on through the hierarchy to the top level. The value of a property is propagated from the highest point in the hierarchy at which it is set down to the original message flow node when the message flow is deployed to a broker. The value of that property on the original message flow node is overridden.

Related concepts:

"Promoted properties" on page 1145
A promoted property is a message flow node property that has been promoted to the level of the message flow in which it is included.

"Broker properties" on page 1144
For each broker, WebSphere Message Broker maintains a set of properties. You can access some of these properties from your ESQL programs. A subset of the properties is also accessible from Java programs. It can be useful, at run time, to have real-time access to details of a specific node, flow, or broker.

"Packaging and deployment overview" on page 3210
Deployment is the process of transferring data to an execution group on a broker so that it can take effect in the broker. Message flows and associated resources are packaged in broker archive (BAR) files for deployment.

Related tasks:

"Defining a promoted property" on page 1297
When you create a message flow, you can promote properties from individual nodes in that message flow to the message flow level. Properties promoted in this way override the property values that you have set for the individual nodes.

"Renaming a promoted property"
If you have promoted a property from the node to the message flow level, it is initially assigned the same name that it has at the node level. You can rename the property to have a more meaningful name in the context of the message flow.

"Converging multiple properties" on page 1306
You can promote properties from several nodes in a message flow to define a single promoted property, which applies to all those nodes.

"Removing a promoted property" on page 1304
If you have promoted a property from the node to the message flow level, you can remove (delete) it if you no longer want to specify its value at the message flow level. The property reverts to the value that you specified at the node level. If you remove a promoted property that is a mandatory property, ensure that you have set a value at the node level. If you have not, you cannot successfully deploy a broker archive file that includes this message flow.

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Renaming a promoted property:

If you have promoted a property from the node to the message flow level, it is initially assigned the same name that it has at the node level. You can rename the property to have a more meaningful name in the context of the message flow.

Before you begin

Before you start:
- Promote a property
- Read the concept topic about promoted properties

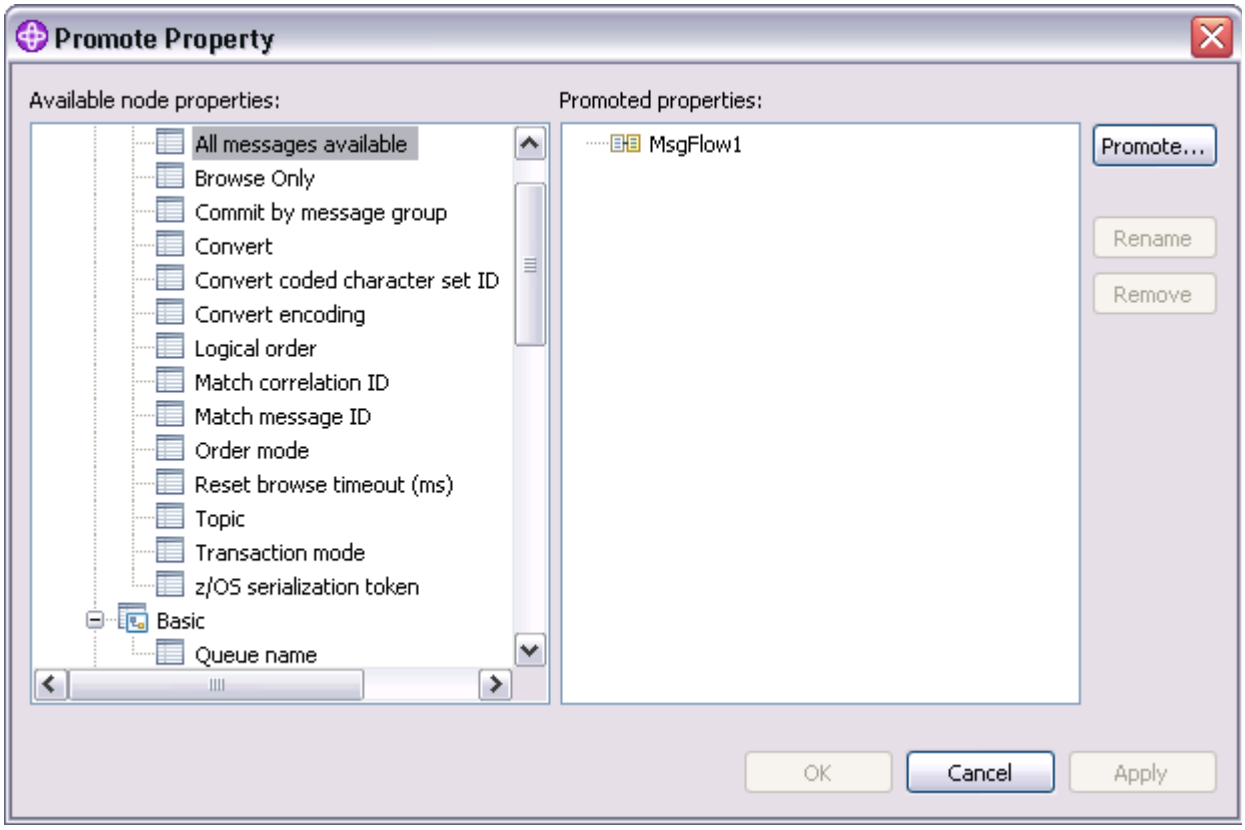
About this task

To rename a promoted property :

Procedure

1. Switch to the Broker Application Development perspective.
2. Open the message flow for which you want to promote properties by double-clicking the message flow in the Broker Development view. You can also open the message flow by right-clicking it in the Broker Development view and clicking **Open** The message flow contents are displayed in the editor view.
3. In the editor view, right-click the symbol of the message flow node for which you want to promote properties.
4. Select **Promote Property**.

   The Promote Property dialog box is displayed.



5. Promoted properties are shown in the Promoted properties pane on the right of the Promote Property dialog box. Double-click the promoted property in the list of properties that are currently promoted to the message flow level, or select the property that you want to rename and click **Rename**. The name is highlighted, and you can edit it. Modify the existing text or enter new text to give the property a new name, and press Enter.
6. Click **Apply** to commit this change without closing the Promote Property dialog box. Click **OK** to complete your updates and close the dialog box.

Related concepts:

"Promoted properties" on page 1145
A promoted property is a message flow node property that has been promoted to the level of the message flow in which it is included.

Related tasks:

"Defining a promoted property" on page 1297
When you create a message flow, you can promote properties from individual nodes in that message flow to the message flow level. Properties promoted in this way override the property values that you have set for the individual nodes.

"Promoting a property" on page 1298
You can promote a node property to the message flow level to simplify the maintenance of the message flow and its nodes, and to provide common values for multiple nodes in the flow by converging promoted properties.

"Removing a promoted property" on page 1304
If you have promoted a property from the node to the message flow level, you can remove (delete) it if you no longer want to specify its value at the message flow level. The property reverts to the value that you specified at the node level. If you remove a promoted property that is a mandatory property, ensure that you have set a value at the node level. If you have not, you cannot successfully deploy a

broker archive file that includes this message flow.

"Converging multiple properties" on page 1306
You can promote properties from several nodes in a message flow to define a single promoted property, which applies to all those nodes.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Removing a promoted property:

If you have promoted a property from the node to the message flow level, you can remove (delete) it if you no longer want to specify its value at the message flow level. The property reverts to the value that you specified at the node level. If you remove a promoted property that is a mandatory property, ensure that you have set a value at the node level. If you have not, you cannot successfully deploy a broker archive file that includes this message flow.

Before you begin

Before you start:
- Promote a property
- Read the concept topic about promoted properties

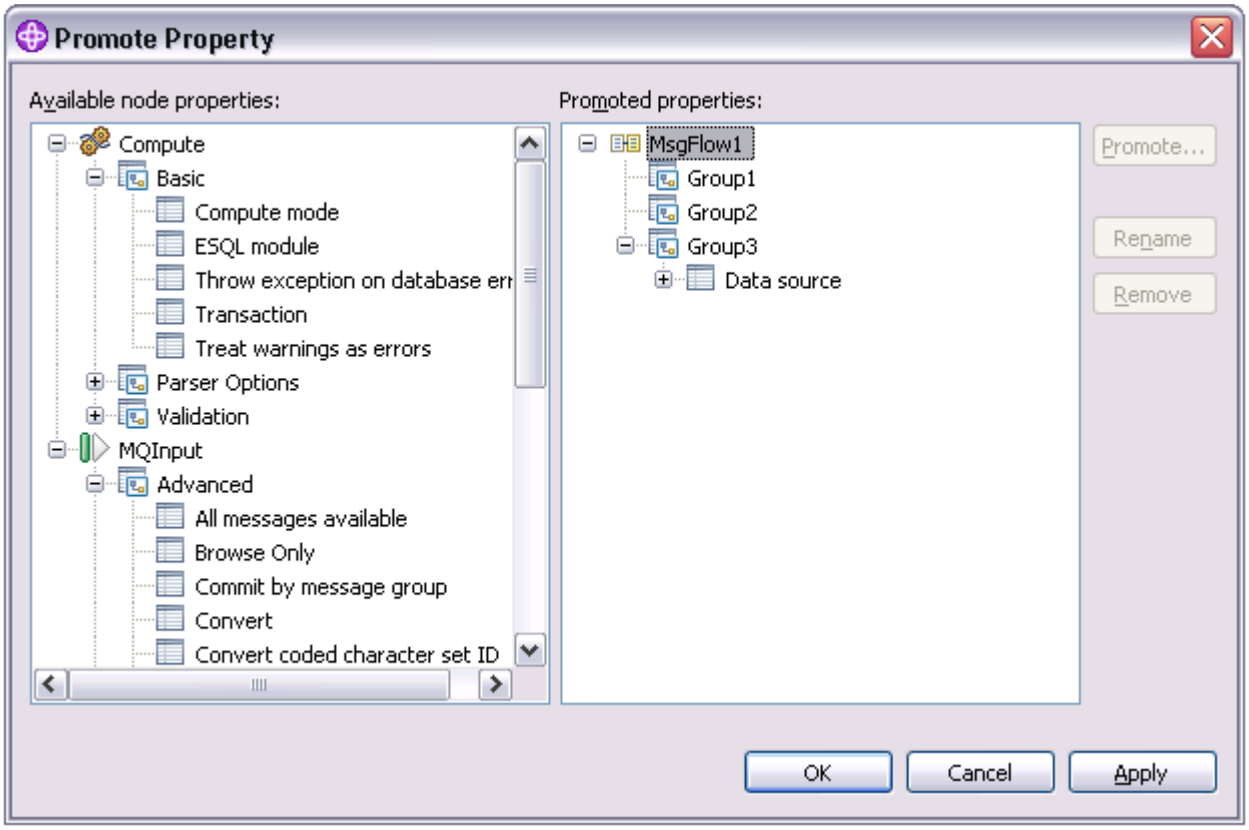
About this task

If you have promoted one or more message flow node properties, and want to delete them:

Procedure

1. Switch to the Broker Application Development perspective.
2. Open the message flow for which you want to promote properties by double-clicking the message flow in the Broker Development view. You can also open the message flow by right-clicking it in the Broker Development view and clicking **Open** The message flow contents are displayed in the editor view.
3. In the Editor view, right-click the symbol of the message flow node whose properties you want to promote.
4. Select **Promote Property**.

   The Promote Property dialog is displayed.



5. Select the promoted property that you want to remove from the list of properties in the Promoted properties pane, and click **Remove**. The property is removed from the Promoted properties pane and is restored to the list in the Available node properties pane, in the appropriate place in the tree of properties for the node from which you promoted it. You can promote this property again.
6. To delete all the promoted properties in a single group, select the group in the Promoted properties pane and click **Remove**. The group and all the properties it contains are deleted from this list: the individual properties that you promoted are restored to the nodes from which you promoted them.
7. Click **Apply** to commit this change without closing the Promote Property dialog box. Click **OK** to complete your updates and close the dialog box.

Results

If you have included this message flow in a higher-level message flow, and have set a value for a promoted property that you have now deleted, the embedding flow is not automatically updated to reflect the deletion. However, when you deploy that embedding message flow in the broker domain, the deleted property is ignored.

Related concepts:

"Promoted properties" on page 1145
A promoted property is a message flow node property that has been promoted to the level of the message flow in which it is included.

Related tasks:

"Defining a promoted property" on page 1297
When you create a message flow, you can promote properties from individual nodes in that message flow to the message flow level. Properties promoted in this way override the property values that you have set for the individual nodes.

"Promoting a property" on page 1298
You can promote a node property to the message flow level to simplify the maintenance of the message flow and its nodes, and to provide common values for multiple nodes in the flow by converging promoted properties.

"Renaming a promoted property" on page 1302
If you have promoted a property from the node to the message flow level, it is initially assigned the same name that it has at the node level. You can rename the property to have a more meaningful name in the context of the message flow.

"Converging multiple properties"
You can promote properties from several nodes in a message flow to define a single promoted property, which applies to all those nodes.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Converging multiple properties:

You can promote properties from several nodes in a message flow to define a single promoted property, which applies to all those nodes.

Before you begin

Before you start:

- Create a message flow, as described in "Creating a message flow" on page 1431
- Read the concept topic about promoted properties

About this task

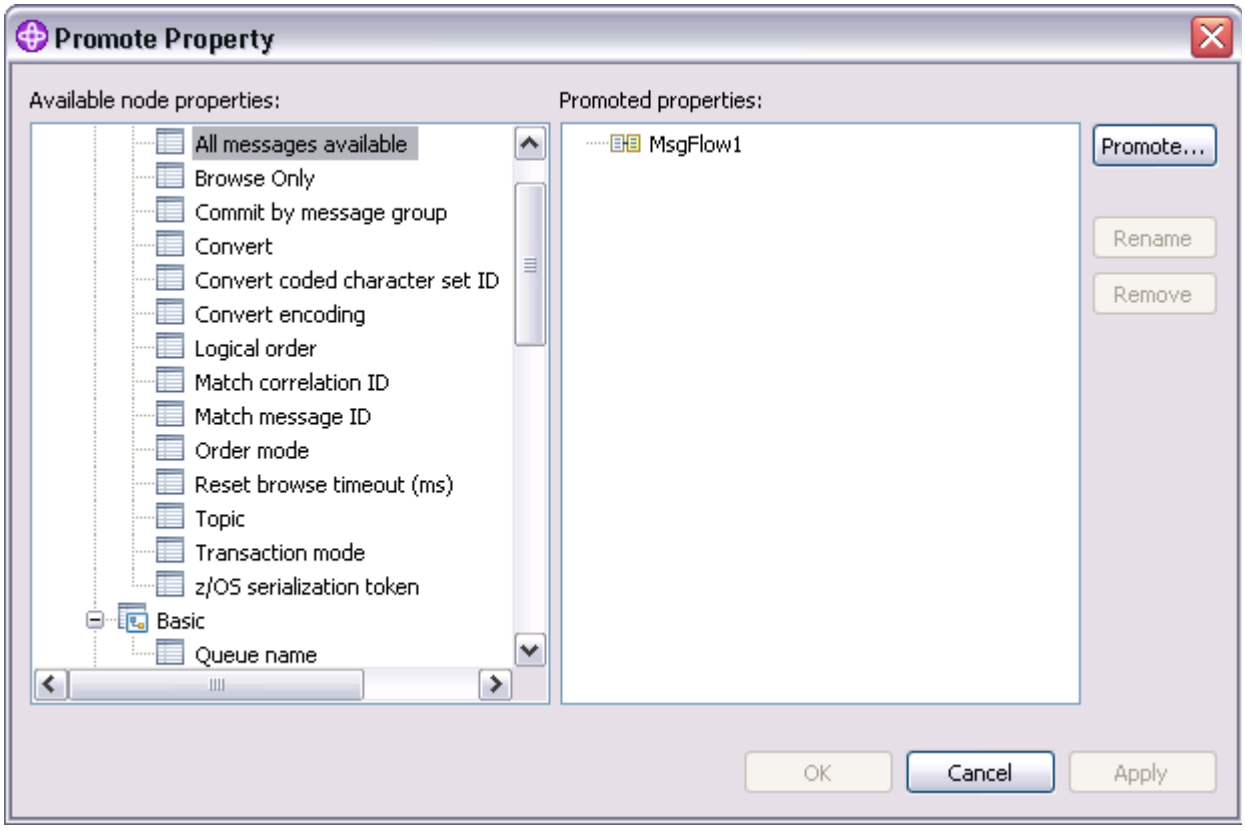
One example for the use of promoting properties is for database access. If a message flow contains two Database nodes that each refer to the same physical database, you can define the physical database just once on the message flow by promoting the `Data Source` property of each Database node to the message flow, and setting the property at the message flow (promoted) level.

To converge multiple node properties to a single promoted property:

Procedure

1. Switch to the Broker Application Development perspective.
2. Open the message flow in the Message Flow editor.
3. Right-click the node for which you want to promote the properties, then click **Promote Property**.

   The Promote Property dialog box is displayed.



4. Select the property that you want to converge. The list in the Available node properties pane initially shows the expanded list of all available properties for the selected node. If you have already promoted properties from this node, they do not appear in this pane, but they appear in the Promoted properties pane.

   The Available node properties pane also includes the other nodes in the open message flow. You can expand the properties that are listed under each node and work with all these properties at the same time. You do not have to close the dialog box and select another node from the Message Flow editor to continue promoting properties.

   You can select multiple properties to promote by selecting a property, holding down Ctrl, and selecting one or more other properties.

   If you have you selected multiple properties to converge, all the properties that you have selected must be available for promotion. If one or more of the selected properties is not available for promotion, the entire selection becomes unavailable for promotion, and the **Promote** button is disabled.

5. Click **Promote** to promote the property or properties

   The Target Selection dialog box opens:

   The Target Selection dialog box displays only the valid targets for the promotion of the previously selected property or properties and allows you to create a new target for the promotion, such as to a new group or to a new property.

6. To converge properties from the same or different nodes in the message flow, expand the tree and click a property that already exists. You can rename the properties by selecting them and clicking **Rename**, or by double-clicking the group or property.

7. Click **OK** to confirm your selections.

Note: If you create a new group or property by using the Target Selection dialog box, the changes persist even if you click **Cancel**. When the dialog box closes, groups or properties that you have created by using the Target Selection dialog box appear in the Promote Property dialog box.

8. Expand the property trees for all the nodes for which you want to promote properties.
9. Drag the first instance of the property that you want to converge from the Available node properties pane, and drop it onto the appropriate group in the Promoted properties pane.
   - If the group already contains one or more promoted properties, the new property is added at the end of the group. You can rename the new property by double-clicking the property, or by selecting the property and clicking **Rename**.
   - If you want the promoted property to appear in a new group, drag the property into an empty space below the existing groups to create a new group. Alternatively:
     a. Select the property that you want to promote, and click **Promote**. The Target Selection dialog box opens.
     b. Click **New Group**, and enter the name of the new group.
     c. Click **OK** to confirm your changes.
   - If you drag the property onto an existing promoted property of a different type, a no-entry icon is displayed and you cannot drop the property. You must create this property as a new promoted property, or drop it onto a compatible existing promoted property. Properties must be associated with the same property editor to be compatible. For example, if you are using built-in nodes, you can converge only properties of the same type (string with string, Boolean with Boolean).

   If you are using user-defined nodes, you must check the compatibility of the property editors for the properties that you want to converge. If you have written compiler classes for a node, you must also ensure that converged properties have the same compiler class.
10. Drag all remaining instances of the property from each of the nodes in the Available node properties pane onto the existing promoted property. The new property is added under the existing promoted property, and is not created as a new promoted property.
11. Click **Apply** to commit this change without closing the Promote Property dialog box. Click **OK** to complete your updates and close the dialog box.

    You can also converge properties from the Promote Property dialog box by dragging the selected property or properties from the Available node properties pane to the Promoted properties pane:
    a. Select the property that you want to promote. You can select multiple properties to promote by selecting a property, holding down Ctrl, and selecting one or more other properties.
    b. Drop the selected property or properties onto a property in the Promoted properties pane to converge related properties from the same or different nodes in the message flow.

       For example, you might want to create a single promoted property that overrides the property on each node that defines a data source.

Results

You have promoted properties from several nodes to define a single promoted property, which is used for all those nodes.

Related concepts:

"Promoted properties" on page 1145
A promoted property is a message flow node property that has been promoted to the level of the message flow in which it is included.

Related tasks:

"Defining a promoted property" on page 1297
When you create a message flow, you can promote properties from individual nodes in that message flow to the message flow level. Properties promoted in this way override the property values that you have set for the individual nodes.

"Promoting a property" on page 1298
You can promote a node property to the message flow level to simplify the maintenance of the message flow and its nodes, and to provide common values for multiple nodes in the flow by converging promoted properties.

"Renaming a promoted property" on page 1302
If you have promoted a property from the node to the message flow level, it is initially assigned the same name that it has at the node level. You can rename the property to have a more meaningful name in the context of the message flow.

"Removing a promoted property" on page 1304
If you have promoted a property from the node to the message flow level, you can remove (delete) it if you no longer want to specify its value at the message flow level. The property reverts to the value that you specified at the node level. If you remove a promoted property that is a mandatory property, ensure that you have set a value at the node level. If you have not, you cannot successfully deploy a broker archive file that includes this message flow.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Developing message flow applications by using patterns

Create resources that are used to solve a specific business problem by using patterns.

Before you begin

Before you start: If you are not familiar with message flow concepts, message model concepts, and common tasks to manage message flow resources, see "Processing messages" on page 1021.

This section describes one of the four methods that you can use to create message flow applications. The other three methods are described in the following sections:

- "Developing message flow applications by using samples" on page 1406
- "Developing message flow applications from a wizard" on page 1408
- "Developing message flow applications from scratch" on page 1423

If you are unsure which method to use, see Chapter 9, "Developing message flow applications," on page 1019 for a discussion of the advantages of each one.

About this task

The following areas are included:
- "Patterns"
- "Using patterns" on page 1312
- "Pattern categories" on page 1331
- "Built-in patterns" on page 1332
- "User-defined patterns" on page 1334

Related tasks:

"Developing message flow applications by using samples" on page 1406
Use the samples to learn more about the features that are available in WebSphere Message Broker, and how to use them.

"Developing message flow applications from a wizard" on page 1408
A Quick Start wizard sets up the basic resources that are required to develop a message flow application. The wizard sets up and gives names to containers for the resources in which you then develop your message flow.

"Developing message flow applications from scratch" on page 1423
Design, create, and configure message flows by using the WebSphere Message Broker Toolkit.

"Connecting client applications" on page 1537
Connect your client applications to the broker by using one or more of the supported protocols from other resources and software servers in your network.

"Routing messages" on page 2209
Route messages through your message flow or the broker by using one or more of the techniques described in this section.

"Transforming and enriching messages" on page 2227
Transform and enrich messages by using one or more of the techniques described in this section.

Patterns

A *pattern* is a reusable solution that encapsulates a tested approach to solving a common architecture, design, or deployment task in a particular context.

A pattern captures a tested solution to a commonly recurring problem, addressing the objectives that you want to achieve. The specification of a pattern describes the problem that is being addressed, why the problem is important (the value statement), and any constraints for the solution. Patterns typically emerge from common usage and the application of a particular product or technology.

A WebSphere Message Broker pattern can be used to generate customized solutions to a recurring problem in an efficient way. WebSphere Message Broker patterns are provided to encourage the adoption of preferred techniques in message flow design, to produce efficient and reliable flows. Patterns provide the following benefits:

- Give you guidance for the implementation of solutions
- Increase development efficiency, because resources are generated from a set of predefined templates
- Result in higher quality solutions, through reuse of assets and common implementation of programming approaches, such as error handling and logging

A catalog of WebSphere Message Broker patterns is provided in the WebSphere Message Broker Toolkit in the Patterns Explorer view. The WebSphere Message Broker patterns are divided into pattern categories. *Pattern categories* are categories

that are based on the pattern classification and structure the display in the Patterns Explorer. The catalog provides detailed help that guides you toward a suitable WebSphere Message Broker pattern to create resources that are used to solve a specific business problem.

The catalog of WebSphere Message Broker patterns contains built-in patterns and might also contain user-defined patterns. *Built-in patterns* cover a set of commonly encountered message flow scenarios and are packaged and released with WebSphere Message Broker. You can also create your own *user-defined patterns*.

Each pattern has values that are known as pattern parameters. *Pattern parameters* are parameters that customize and configure a WebSphere Message Broker pattern. The pattern parameters that you configure depend on the particular pattern, and on the options that you enable for that pattern. An example of a pattern parameter is a queue name from where messages are read.

The WebSphere Message Broker patterns provide defaults for most pattern parameters, and help is provided to explain them. After you have configured the pattern parameters, you generate a *pattern instance project*, which contains references to all other projects in the workspace that relate to your pattern instance. A pattern instance project also contains a pattern instance configuration file that stores the pattern parameter values. This configuration file stores the pattern parameters that you configured. Generating a pattern instance project also creates one or more additional WebSphere Message Broker projects that typically contain message flows and other WebSphere Message Broker resources that implement the pattern.

You can open the pattern instance configuration file at any time to see the values of the pattern parameters. When a pattern configuration file has been reopened, you can regenerate the WebSphere Message Broker projects. Regeneration deletes the generated WebSphere Message Broker projects and re-creates them from scratch. The pattern instance project contains an HTML summary file and a pattern instance configuration file. The summary file has a section that explains additional tasks that might be required, such as creating queues.

You can create resources from each WebSphere Message Broker pattern more than once to give you unique pattern instances, each with a different configuration. The pattern parameters that you can configure depend on the particular pattern, and also the options that you enable for that pattern.

Some of the pattern parameter settings can affect the resources that are generated. For example, if you enable logging and error handling, the generated projects contain additional message flows and ESQL scripts.

In all cases, the specification of both the problem and the solution are indispensable parts of the pattern definition.

Related concepts:

"Using patterns" on page 1312
Use patterns that are supplied with the WebSphere Message Broker Toolkit to create resources that are used to solve a specific business problem.

"Built-in patterns" on page 1332
A *built-in pattern* is a pattern that covers a set of commonly encountered message flow scenarios and that is packaged and released with WebSphere Message Broker.

"User-defined patterns" on page 1334
A *user-defined pattern* extends the function of WebSphere Message Broker so that you are able to create patterns that you can reuse within your organization.

Related tasks:

"Choosing a pattern" on page 1313
Select a pattern in the Patterns Explorer view to create resources to solve a specific business problem.

Using patterns

Use patterns that are supplied with the WebSphere Message Broker Toolkit to create resources that are used to solve a specific business problem.

Each *WebSphere Message Broker pattern* is designed to address a specific business problem. To find out about the patterns that are supplied in the WebSphere Message Broker Toolkit, see "Built-in patterns" on page 1332. You might also have *user-defined patterns* available to use that are created by a *pattern author*, see "User-defined patterns" on page 1334.

Each pattern has values known as *pattern parameters* that you use to create the pattern resources for use in your environment. The pattern parameters that you can configure depend on the specific pattern, and also the options that you enable for that pattern; for example, logging.

When you select a pattern from the Patterns Explorer, the **Pattern Specification** tab gives details about the purpose and configuration of each pattern parameter associated with a pattern, the effects and consequences of changing them, and any postrequisite tasks. All patterns have pattern parameter properties that distinguish one application of a pattern template from another.

All patterns are either abstract or implementations and as you move down the tree the patterns become more specific. At any level in a tree a pattern can be an implementation.

You can apply pattern implementations only. Pattern implementations map to a complete specification with prerequisite and postrequisite tasks, and pattern parameter details. Pattern implementations have a **Create New Instance** button in the **Pattern Specification** tab. *Abstract* patterns, which cannot be applied, do not have a **Create New Instance** button in the **Pattern Specification** tab.

You can create resources from each pattern more than once to give unique *pattern instances* with different configurations. The configuration for each pattern instance is contained within a single *pattern instance project*. The pattern instance project contains links to all projects containing the resources that are created as a result of generating a pattern instance from your configuration, such as message flows, Java classes for JavaCompute nodes, ESQL modules, message maps, test client, XML files, and style sheet files.

Related concepts:

"Patterns" on page 1310
A *pattern* is a reusable solution that encapsulates a tested approach to solving a common architecture, design, or deployment task in a particular context.

"Built-in patterns" on page 1332
A *built-in pattern* is a pattern that covers a set of commonly encountered message flow scenarios and that is packaged and released with WebSphere Message Broker.

Related tasks:

"Choosing a pattern"
Select a pattern in the Patterns Explorer view to create resources to solve a specific business problem.

"Generating a pattern instance" on page 1326
Generate resources from the pattern.

Related reference:

"Pattern instance projects" on page 6827
A pattern instance project contains project references to all the other projects in the workspace that relate to a specific pattern instance.

Choosing a pattern

Select a pattern in the Patterns Explorer view to create resources to solve a specific business problem.

Before you begin

Before you start:

Before completing this task, read the following overview topics about patterns:
- "Patterns" on page 1310
- "Using patterns" on page 1312

About this task

To create resources by using a patter, complete the following steps.

Procedure

1. Open the Patterns Explorer view by using one of the following methods.
   - In the Broker Development view, click **New pattern instance** in the Pattern Instances section.
   - Click the **Patterns Explorer** tab.
   - Click the **Open Patterns Explorer** icon in the task bar.

   The Patterns Explorer view is displayed.
2. Click **Patterns** in the Patterns Explorer view. The **Pattern Specification** tab displays information about the patterns. For example, to find out about message-based integration patterns, complete the following steps:
   a. Click **Message-based Integration**. The **Pattern Specification** tab displays information about the selected pattern, and because it is an abstract pattern the **Create New Instance** button is not displayed.
   b. Under **Message-based Integration** click another pattern, for example **Message Correlator**. The **Pattern Specification** tab displays information about the selected pattern, and because it is an abstract pattern the **Create New Instance** button is not displayed.
   c. Under **Message Correlator** click another pattern, for example **MQ request-response with persistence**. The **Pattern Specification** tab displays information about the selected pattern, and because it is a pattern implementation the **Create New Instance** button is displayed.
3. Ensure that you look at "Tasks to complete before applying the pattern" and "Constraints on the use of the pattern" for your chosen pattern before you use it.

4. To use a pattern implementation from the **Pattern Specification** tab, click **Create New Instance**. The **Create New Instance** window opens.
5. Enter a new pattern instance name.

   You cannot use the following names:

   - The name of an existing project in the current workspace.
   - On Windows only: MS-DOS device driver names. For more information, see the article on the Microsoft support site: MS-DOS device driver names cannot be used as file names.
6. Click **OK**. If you click **Cancel**, no changes are made to the workspace. The Configure Pattern Parameters page is displayed.
7. On the Configure Pattern Parameters page, complete all the pattern parameter fields as required. Configure pattern parameters by entering the appropriate values. If you do not want to generate the pattern immediately, you can save the parameters to work with later. Before you apply a pattern, you must enter a value for each pattern parameter. You can accept a default value, if one is offered. Empty strings are permitted for some fields.

   If you have already saved a configuration, you can reuse that configuration by editing the parameters, see "Editing and regenerating a pattern" on page 1327.

   Parameter groups might be marked with error markers. You can click the error marker to move to the first field that has an error.

What to do next

Next: Now complete the following task:

- "Generating a pattern instance" on page 1326

Related concepts:

"Patterns" on page 1310
A *pattern* is a reusable solution that encapsulates a tested approach to solving a common architecture, design, or deployment task in a particular context.

"Using patterns" on page 1312
Use patterns that are supplied with the WebSphere Message Broker Toolkit to create resources that are used to solve a specific business problem.

Related tasks:

"Generating a pattern instance" on page 1326
Generate resources from the pattern.

"Editing and regenerating a pattern" on page 1327
You can regenerate a pattern instance after you have edited a saved configuration in a pattern instance project.

Related reference:

"Resource types in the WebSphere Message Broker Toolkit" on page 6821
Links to information on projects and resource files.

Working with patterns in the Broker Development view

Using the Broker Development view to create patterns.

Before you begin

Before you start:

Before completing this task, read the following overview topics about patterns:

- "Patterns" on page 1310
- "Using patterns" on page 1312

About this task

The Broker Development view has three sections: the working set selection list (<all resources>), Pattern Instances (initially collapsed), and Projects (initially expanded). Pattern instance projects are shown only in the Pattern Instances section, and all other project types are shown in the Projects section. If no projects exist, the Projects section contains a list of Quick Starts links. Use the following instructions to alter the layout of these three sections.

When the workspace contains both pattern instance projects and projects, the Pattern Instances section and the Projects section each have their own resource tree. Each section also has its own scroll bar.

- To open the Pattern Instances or the Projects section to its default or last known size, click the **Restore** icon in the relevant section. If no pattern instance projects exist in the workspace, the Pattern Instances section contains a hyperlink sentence, **New pattern instance**, click this hyperlink to start working with patterns:
  - To maximize the Pattern Instances section, click its **Maximize** icon. This action automatically collapses the Projects section.
  - To maximize the Projects section, click its **Maximize** icon. This action automatically collapses the Pattern Instances section.
- To use a Quick Starts link, use either of the following options:
  - In the Quick Starts list in the Projects section, click the link that you want to use. However, when the Projects section contains projects, the Quick Starts list is not shown in the Projects section, you must, therefore, use the other option.
  - In the toolbar of the Projects section, click the **Quick Starts** link.

What to do next

Next: Now complete the following task:

- "Choosing a pattern" on page 1313

Related concepts:

"Patterns" on page 1310
A *pattern* is a reusable solution that encapsulates a tested approach to solving a common architecture, design, or deployment task in a particular context.

"Using patterns" on page 1312
Use patterns that are supplied with the WebSphere Message Broker Toolkit to create resources that are used to solve a specific business problem.

Related tasks:

"Creating an application from scratch" on page 1411
Use the "Start from scratch" wizard to create the basic resources that are required to develop a broker application.

"Generating a pattern instance" on page 1326
Generate resources from the pattern.

Related reference:

"Resource types in the WebSphere Message Broker Toolkit" on page 6821
Links to information on projects and resource files.

Adding or removing project references in a pattern instance project:

A pattern instance project contains references to regular projects and they are listed in the Project References category of a pattern instance project.

About this task

The Project References category under a pattern instance project in the Broker Development view shows the same project references as the Project References page in the Properties window.

To add another project to the list of project references for your pattern instance project, complete the following steps:

Procedure

1. Open the Pattern Instances section of the Broker Development view.
2. Right-click the pattern instance project name, select **Add or Remove Project References** from the menu. The Add or Remove Project References window for the pattern instance project opens, which shows a list of all regular projects, but not pattern instance projects, in the workspace.
3. To add another project to the list of project references, select the project name and click **OK**. The Broker Development view shows the selected project name under the pattern instance Project References category.
4. To remove a project from the list of project references, right-click the project name, select **Add or Remove Project References**, clear the project name check box, and click **OK**. The project name is removed from the Project References category.

Related concepts:

"Patterns" on page 1310
A *pattern* is a reusable solution that encapsulates a tested approach to solving a common architecture, design, or deployment task in a particular context.

"Using patterns" on page 1312
Use patterns that are supplied with the WebSphere Message Broker Toolkit to create resources that are used to solve a specific business problem.

Related tasks:

"Generating a pattern instance" on page 1326
Generate resources from the pattern.

"Going to a referenced project"
How to display your pattern instance project in the Broker Development view.

Related reference:

"Resource types in the WebSphere Message Broker Toolkit" on page 6821
Links to information on projects and resource files.

"Pattern instance projects" on page 6827
A pattern instance project contains project references to all the other projects in the workspace that relate to a specific pattern instance.

Going to a referenced project:

How to display your pattern instance project in the Broker Development view.

About this task

To display your pattern instance project complete the following steps.

Procedure

1. Open the Pattern Instances section of the Broker Development view.
2. Expand your pattern instance project, and expand **Project References**.

3. Right-click your pattern instance project reference.
4. Select **Go to Referenced Project**. The referenced flows project is selected in the Broker Development view.

Related concepts:

"Patterns" on page 1310
A *pattern* is a reusable solution that encapsulates a tested approach to solving a common architecture, design, or deployment task in a particular context.

"Using patterns" on page 1312
Use patterns that are supplied with the WebSphere Message Broker Toolkit to create resources that are used to solve a specific business problem.

Related tasks:

"Generating a pattern instance" on page 1326
Generate resources from the pattern.

"Adding or removing project references in a pattern instance project" on page 1315
A pattern instance project contains references to regular projects and they are listed in the Project References category of a pattern instance project.

Related reference:

"Resource types in the WebSphere Message Broker Toolkit" on page 6821
Links to information on projects and resource files.

"Pattern instance projects" on page 6827
A pattern instance project contains project references to all the other projects in the workspace that relate to a specific pattern instance.

Showing a working set for a pattern instance project:

When a pattern instance project is created by the Patterns Explorer, a working set of the same name is generated, and all projects that are part of that pattern instance are placed inside it.

About this task

To view all of the projects in your pattern instance:

Procedure

1. In the Broker Development view, select your pattern instance project in the Working Sets section. The Pattern Instances section displays all of the projects, and the Projects section displays all of the message flow details that are associated with that pattern instance.
2. Select **<all resources>** in the Working Set section, the navigator shows all of the projects in the workspace.

Related concepts:

"Patterns" on page 1310
A *pattern* is a reusable solution that encapsulates a tested approach to solving a common architecture, design, or deployment task in a particular context.

"Using patterns" on page 1312
Use patterns that are supplied with the WebSphere Message Broker Toolkit to create resources that are used to solve a specific business problem.

"Resources" on page 36
The projects, folders, and files that you work with in the WebSphere Message Broker Toolkit workspace are called *resources*. By default, these resources are stored with their metadata in the workspace directory in your local file system. The workspace directory is created the first time that you start the WebSphere Message

Broker Toolkit.

Related tasks:

"Generating a pattern instance" on page 1326
Generate resources from the pattern.

"Focusing on a pattern instance"
Several ways are available for you to focus on a particular pattern instance.

"Creating a working set and focus on a pattern instance" on page 1319
Creating a working set for an imported project interchange file that contains a pattern instance project and its associated projects.

Related reference:

"Resource types in the WebSphere Message Broker Toolkit" on page 6821
Links to information on projects and resource files.

"Pattern instance projects" on page 6827
A pattern instance project contains project references to all the other projects in the workspace that relate to a specific pattern instance.

Focusing on a pattern instance:

Several ways are available for you to focus on a particular pattern instance.

Procedure

To focus on your pattern you can use one of the following methods:

- In the **<all resources>** section, click the down arrow icon to open the menu, and select the pattern instance name.
- In the Pattern Instances section, right-click your pattern instance project, select **Focus on Pattern Instance**.
- In the toolbar of the Broker Development view, click the down arrow and select your pattern instance name from the menu.

The navigator refreshes to show the currently selected working set and its associated projects.
To confirm that your pattern instance project is currently in focus, right-click your pattern instance project, ensure that the check box to the left of the **Focus on Pattern Instance** menu item is selected. If you clear this check box, the navigator reverts to the **<all resources>** working set, and shows all of the projects.
If the navigator has focus on your working set, if you click the down arrow in the WebSphere Message Broker Development toolbar, your working set menu item is selected in the menu. You cannot clear a working set from this menu to return to the **<all resources>** section, but you can select a different working set, if one is available.

Related concepts:

"Patterns" on page 1310
A *pattern* is a reusable solution that encapsulates a tested approach to solving a common architecture, design, or deployment task in a particular context.

"Using patterns" on page 1312
Use patterns that are supplied with the WebSphere Message Broker Toolkit to create resources that are used to solve a specific business problem.

"Resources" on page 36
The projects, folders, and files that you work with in the WebSphere Message Broker Toolkit workspace are called *resources*. By default, these resources are stored with their metadata in the workspace directory in your local file system. The workspace directory is created the first time that you start the WebSphere Message

Broker Toolkit.

Related tasks:

"Generating a pattern instance" on page 1326
Generate resources from the pattern.

"Showing a working set for a pattern instance project" on page 1317
When a pattern instance project is created by the Patterns Explorer, a working set of the same name is generated, and all projects that are part of that pattern instance are placed inside it.

"Creating a working set and focus on a pattern instance"
Creating a working set for an imported project interchange file that contains a pattern instance project and its associated projects.

Related reference:

"Resource types in the WebSphere Message Broker Toolkit" on page 6821
Links to information on projects and resource files.

"Pattern instance projects" on page 6827
A pattern instance project contains project references to all the other projects in the workspace that relate to a specific pattern instance.

Creating a working set and focus on a pattern instance:

Creating a working set for an imported project interchange file that contains a pattern instance project and its associated projects.

About this task

If you import a project interchange file that contains a pattern instance project and its associated projects, a working set is not automatically created for the pattern instance project. Therefore, you must create a working set.

Procedure

In the Pattern Instances section of the Broker Development view, right-click the pattern instance project, and select **Create Working Set and Focus on Pattern Instance** in the menu. When this item is selected, a working set is created with the same name as the pattern instance project, the pattern instance project and its associated projects are added to that working set, and the working set is given focus in the navigator.

Related concepts:

"Patterns" on page 1310
A *pattern* is a reusable solution that encapsulates a tested approach to solving a common architecture, design, or deployment task in a particular context.

"Using patterns" on page 1312
Use patterns that are supplied with the WebSphere Message Broker Toolkit to create resources that are used to solve a specific business problem.

"Resources" on page 36
The projects, folders, and files that you work with in the WebSphere Message Broker Toolkit workspace are called *resources*. By default, these resources are stored with their metadata in the workspace directory in your local file system. The workspace directory is created the first time that you start the WebSphere Message Broker Toolkit.

Related tasks:

"Generating a pattern instance" on page 1326
Generate resources from the pattern.

"Showing a working set for a pattern instance project" on page 1317
When a pattern instance project is created by the Patterns Explorer, a working set of the same name is generated, and all projects that are part of that pattern instance are placed inside it.

"Creating a working set and focus on a pattern instance" on page 1319
Creating a working set for an imported project interchange file that contains a pattern instance project and its associated projects.

"Importing and exporting resources in a Project Interchange file" on page 1452
You can import resources to, or export resources from, WebSphere Message Broker by using a Project Interchange file.

Related reference:

"Resource types in the WebSphere Message Broker Toolkit" on page 6821
Links to information on projects and resource files.

"Pattern instance projects" on page 6827
A pattern instance project contains project references to all the other projects in the workspace that relate to a specific pattern instance.

Deleting a pattern instance project:

Pattern instance projects typically contain references to regular projects. Therefore, when you select a pattern instance project for deletion you must decide whether to delete all of the referenced projects from the file system. Some pattern instance projects have no project references.

About this task

Use the following instructions depending on the type of pattern instance that you want to delete:

- "Deleting a pattern instance project with project references" on page 1321
- "Deleting a pattern instance project with no project references" on page 1322

Related concepts:

"Patterns" on page 1310
A *pattern* is a reusable solution that encapsulates a tested approach to solving a common architecture, design, or deployment task in a particular context.

"Using patterns" on page 1312
Use patterns that are supplied with the WebSphere Message Broker Toolkit to create resources that are used to solve a specific business problem.

"Resources" on page 36
The projects, folders, and files that you work with in the WebSphere Message Broker Toolkit workspace are called *resources*. By default, these resources are stored with their metadata in the workspace directory in your local file system. The workspace directory is created the first time that you start the WebSphere Message Broker Toolkit.

Related tasks:

"Generating a pattern instance" on page 1326
Generate resources from the pattern.

"Showing a working set for a pattern instance project" on page 1317
When a pattern instance project is created by the Patterns Explorer, a working set of the same name is generated, and all projects that are part of that pattern instance are placed inside it.

"Deleting projects that are referenced by a pattern instance project" on page 1324
If you want to delete a project, you must take into account whether it is referenced

by a pattern instance project.

Related reference:

"Resource types in the WebSphere Message Broker Toolkit" on page 6821
Links to information on projects and resource files.

"Pattern instance projects" on page 6827
A pattern instance project contains project references to all the other projects in the workspace that relate to a specific pattern instance.

Deleting a pattern instance project with project references:

Pattern instance projects typically contain references to regular projects. Therefore, when you select a pattern instance project for deletion you must decide whether to delete all of the referenced projects from the file system.

About this task

Use the following steps to delete a pattern instance project that has project references:

Procedure

1. In the Broker Development view, open the Pattern Instances section.
2. Right-click the pattern instance project, or projects, that you want to delete, click **Delete**. The Confirm Project Delete window opens, displaying any projects that are referenced by the pattern instance project. The **Also delete the projects referenced by this pattern instance project** check box is selected and the **Also delete contents in the file system** check box is cleared by default.
3. Choose one of the following options:
   - To delete the pattern instance project, or projects, and the project references, but leave the contents in the file system, keep the default settings, click **Yes**. The working set section of the Broker Development view switches to **<all resources>**. The pattern instance project, or projects, and the project references still exist in the file system.
   - To delete the pattern instance project, or projects, and the project references from the workspace and the file system, ensure that **Also delete the projects referenced by this pattern instance project** and **Also delete contents in the file system** are both selected, click **Yes**. The working set is deleted. The working set section of the Broker Development view switches to **<all resources>**.
   - To delete your pattern instance project, or projects, from the workspace but not from the file system and to keep the project references in the workspace and file system, ensure that **Also delete the projects referenced by this pattern instance project** and **Also delete contents in the file system** are both cleared. The working set is deleted. The working set section of the Broker Development view switches to **<all resources>**.
   - If you decide not to delete the pattern instance project, or projects, click **No**.

Related concepts:

"Patterns" on page 1310
A *pattern* is a reusable solution that encapsulates a tested approach to solving a common architecture, design, or deployment task in a particular context.

"Using patterns" on page 1312
Use patterns that are supplied with the WebSphere Message Broker Toolkit to create resources that are used to solve a specific business problem.

"Resources" on page 36
The projects, folders, and files that you work with in the WebSphere Message Broker Toolkit workspace are called *resources*. By default, these resources are stored with their metadata in the workspace directory in your local file system. The workspace directory is created the first time that you start the WebSphere Message Broker Toolkit.

Related tasks:

"Generating a pattern instance" on page 1326
Generate resources from the pattern.

"Showing a working set for a pattern instance project" on page 1317
When a pattern instance project is created by the Patterns Explorer, a working set of the same name is generated, and all projects that are part of that pattern instance are placed inside it.

"Deleting a pattern instance project with no project references"
Decide whether to delete your pattern instance project, which has no project references, from both the workspace and the file system.

"Deleting projects that are referenced by a pattern instance project" on page 1324
If you want to delete a project, you must take into account whether it is referenced by a pattern instance project.

Related reference:

"Resource types in the WebSphere Message Broker Toolkit" on page 6821
Links to information on projects and resource files.

"Pattern instance projects" on page 6827
A pattern instance project contains project references to all the other projects in the workspace that relate to a specific pattern instance.

Deleting a pattern instance project with no project references:

Decide whether to delete your pattern instance project, which has no project references, from both the workspace and the file system.

About this task

Use the following steps to delete a pattern instance project that has no project references:

Procedure

1. In the Broker Development view, open the Pattern Instances section.
2. Right-click the pattern instance project, or projects, that you want to delete, click **Delete**. The Confirm Project Delete window opens, displaying the pattern instance project. The **Also delete contents under *pattern _instance_name*** check box is cleared by default. The default is to delete the pattern instance project from the workspace, but not to delete it from the file system.
3. Choose one of the following options:
   - To delete the pattern instance project, or projects, from the workspace, but not from the file system, click **Yes**. The working set is also deleted. The working set section of the Broker Development view switches to **<all resources>**.
   - To delete the pattern instance project, or projects, from the workspace and from the file system, select the **Also delete contents under *pattern_instance_name*** check box, click **Yes**. The working set is also deleted. The working set section of the Broker Development view switches to **<all resources>**.

- If you decide not to delete the pattern instance project, or projects, click **No**.

Related concepts:

"Patterns" on page 1310
A *pattern* is a reusable solution that encapsulates a tested approach to solving a common architecture, design, or deployment task in a particular context.

"Using patterns" on page 1312
Use patterns that are supplied with the WebSphere Message Broker Toolkit to create resources that are used to solve a specific business problem.

"Resources" on page 36
The projects, folders, and files that you work with in the WebSphere Message Broker Toolkit workspace are called *resources*. By default, these resources are stored with their metadata in the workspace directory in your local file system. The workspace directory is created the first time that you start the WebSphere Message Broker Toolkit.

Related tasks:

"Generating a pattern instance" on page 1326
Generate resources from the pattern.

"Showing a working set for a pattern instance project" on page 1317
When a pattern instance project is created by the Patterns Explorer, a working set of the same name is generated, and all projects that are part of that pattern instance are placed inside it.

"Deleting a pattern instance project with project references" on page 1321
Pattern instance projects typically contain references to regular projects. Therefore, when you select a pattern instance project for deletion you must decide whether to delete all of the referenced projects from the file system.

"Deleting projects that are referenced by a pattern instance project" on page 1324
If you want to delete a project, you must take into account whether it is referenced by a pattern instance project.

Related reference:

"Resource types in the WebSphere Message Broker Toolkit" on page 6821
Links to information on projects and resource files.

"Pattern instance projects" on page 6827
A pattern instance project contains project references to all the other projects in the workspace that relate to a specific pattern instance.

Deleting projects:

If you want to delete a project, you must take into account whether it is referenced by a pattern instance project.

About this task

Use the following instructions depending on whether the project that you want to delete is referenced by a pattern instance project:

- "Deleting projects that are referenced by a pattern instance project" on page 1324
- "Deleting projects that are not referenced by a pattern instance project" on page 1325

Related concepts:

"Patterns" on page 1310
A *pattern* is a reusable solution that encapsulates a tested approach to solving a common architecture, design, or deployment task in a particular context.

"Using patterns" on page 1312
Use patterns that are supplied with the WebSphere Message Broker Toolkit to

create resources that are used to solve a specific business problem.

"Resources" on page 36
The projects, folders, and files that you work with in the WebSphere Message Broker Toolkit workspace are called *resources*. By default, these resources are stored with their metadata in the workspace directory in your local file system. The workspace directory is created the first time that you start the WebSphere Message Broker Toolkit.

Related tasks:

"Generating a pattern instance" on page 1326
Generate resources from the pattern.

Related reference:

"Resource types in the WebSphere Message Broker Toolkit" on page 6821
Links to information on projects and resource files.

"Pattern instance projects" on page 6827
A pattern instance project contains project references to all the other projects in the workspace that relate to a specific pattern instance.

Deleting projects that are referenced by a pattern instance project:

If you want to delete a project, you must take into account whether it is referenced by a pattern instance project.

About this task

Complete the following steps to delete your project if it is referenced by a pattern instance project.

Procedure

1. In the Broker Development view, select the project that you want to delete, then click **Delete**. The Confirm Project Delete window opens with a warning that the project is referenced by a pattern instance project. The default action is to delete the selected project from the workspace, not to delete the project from the file system, and to update the project references of all pattern instance projects that reference it. Therefore, by default **Also remove references from all pattern instance projects** is selected, and **Also delete content under *project_name*** is cleared.
2. Choose one of the following options:
   - To delete the projects from the workspace but leave the contents in the file system, and delete the project reference from the pattern instance projects list of project references, keep the default settings and click **Yes**.
   - To delete the projects from the workspace and remove the contents in the file system, and delete the project reference from the pattern instance projects list of project references, ensure that both **Also remove references from all pattern instance projects** and **Also delete content under *project_name*** are selected, and click **Yes**.
   - To delete the projects from the workspace and remove the contents in the file system, and leave the project reference in the pattern instance projects list of project references, ensure that **Also remove references from all pattern instance projects** is cleared and that **Also delete content under *project_name*** is selected, and click **Yes**. The icon for this project reference changes to reflect that the project no longer exists.
   - To delete the projects from the workspace but leave the contents in the file system, and leave the project reference in the pattern instance projects list of

project references, ensure that both **Also remove references from all pattern instance projects** and **Also delete content under *project_name*** are cleared, and click **Yes**. The icon for this project reference changes to reflect that the project no longer exists.

- If you decide not to delete the projects, click **No**.

Related concepts:

"Patterns" on page 1310
A *pattern* is a reusable solution that encapsulates a tested approach to solving a common architecture, design, or deployment task in a particular context.

"Using patterns" on page 1312
Use patterns that are supplied with the WebSphere Message Broker Toolkit to create resources that are used to solve a specific business problem.

"Resources" on page 36
The projects, folders, and files that you work with in the WebSphere Message Broker Toolkit workspace are called *resources*. By default, these resources are stored with their metadata in the workspace directory in your local file system. The workspace directory is created the first time that you start the WebSphere Message Broker Toolkit.

Related tasks:

"Generating a pattern instance" on page 1326
Generate resources from the pattern.

"Deleting projects that are not referenced by a pattern instance project"
If you want to delete a project, you must take into account whether it is referenced by a pattern instance project.

Related reference:

"Resource types in the WebSphere Message Broker Toolkit" on page 6821
Links to information on projects and resource files.

"Pattern instance projects" on page 6827
A pattern instance project contains project references to all the other projects in the workspace that relate to a specific pattern instance.

Deleting projects that are not referenced by a pattern instance project:

If you want to delete a project, you must take into account whether it is referenced by a pattern instance project.

About this task

Complete the following steps to delete your project if it is not referenced by a pattern instance project:

Procedure

1. In the Broker Development view, select the project that you want to delete, then click **Delete**. The Confirm Project Delete window opens asking whether you are sure that you want to delete the project. The default action is to delete the project from the workspace, but not to delete it from the file system. Therefore, by default, the **Also delete content under *project_name*** check box is cleared.
2. Choose one of the following options:
   - To delete the projects from the workspace, but not to delete it from the file system, ensure that the **Also delete content under *project_name*** check box is cleared.

- To delete the projects from the workspace and delete it from the file system, ensure that the **Also delete content under *project_name*** check box is selected.
- If you decide not to delete the projects, click **No**.

Related concepts:

"Patterns" on page 1310
A *pattern* is a reusable solution that encapsulates a tested approach to solving a common architecture, design, or deployment task in a particular context.

"Using patterns" on page 1312
Use patterns that are supplied with the WebSphere Message Broker Toolkit to create resources that are used to solve a specific business problem.

"Resources" on page 36
The projects, folders, and files that you work with in the WebSphere Message Broker Toolkit workspace are called *resources*. By default, these resources are stored with their metadata in the workspace directory in your local file system. The workspace directory is created the first time that you start the WebSphere Message Broker Toolkit.

Related tasks:

"Generating a pattern instance"
Generate resources from the pattern.

"Deleting projects that are referenced by a pattern instance project" on page 1324
If you want to delete a project, you must take into account whether it is referenced by a pattern instance project.

Related reference:

"Resource types in the WebSphere Message Broker Toolkit" on page 6821
Links to information on projects and resource files.

"Pattern instance projects" on page 6827
A pattern instance project contains project references to all the other projects in the workspace that relate to a specific pattern instance.

Generating a pattern instance

Generate resources from the pattern.

Before you begin

Before you start:

You must have chosen pattern. For more details, see "Choosing a pattern" on page 1313.

About this task

The **Generate** button is unavailable until all required parameters are complete.

To generate the WebSphere Message Broker Toolkit resources for the pattern, complete the following steps.

Procedure

On the Configuration page, click **Generate**. Focus is automatically moved to the Broker Development view, which lists all the new resources that have been generated.
All of the other resources are filtered out in this view.

Results

Following a successful **Generate** action:

- A pattern instance project is created in your workspace and is displayed in the Broker Development view. The pattern instance project contains a pattern instance configuration file and a summary file. The summary file describes the tasks that have been generated, and the tasks that are still required.
- All the resources generated for a pattern instance are packaged into their own working set and the Broker Development view displays only the resources in this working set. The name of the working set is always the same as the pattern instance name provided. This name must always be unique within the workspace.
- You can regenerate the pattern by using the same instance name, but, if you confirm the regeneration, existing projects are deleted, see "Editing and regenerating a pattern."

What to do next

Next: Now complete the following task:

- "Reviewing the pattern instance summary and tasks" on page 1328

Related concepts:

"Using patterns" on page 1312
Use patterns that are supplied with the WebSphere Message Broker Toolkit to create resources that are used to solve a specific business problem.

Related tasks:

"Editing and regenerating a pattern"
You can regenerate a pattern instance after you have edited a saved configuration in a pattern instance project.

"Creating a working set" on page 575
Create a working set to limit the number of resources that are displayed in the Broker Development view.

Related reference:

"Pattern instance projects" on page 6827
A pattern instance project contains project references to all the other projects in the workspace that relate to a specific pattern instance.

Editing and regenerating a pattern:

You can regenerate a pattern instance after you have edited a saved configuration in a pattern instance project.

About this task

Before you begin:

You must be in the WebSphere Message Broker Toolkit and have an existing pattern configuration file in a pattern instance project.

Procedure

1. Open an existing pattern configuration by double-clicking the configuration XML file in a pattern instance project. The Configure Pattern Parameters page of the editor displays parameter values for that configuration instance. The

Broker navigation pane is in focus. The title of the Configure Pattern Parameters page for the parameter reflects the pattern instance name (not the pattern name).

2. View and edit the pattern parameters. You can now choose either of the following options:
   - Close the editor without saving the changes.
   - Select **Generate**. Resources, such as projects, already exist in the workspace. You are warned that these resources will be deleted and you can choose whether to continue. If you continue, the resources are regenerated; the existing resources are deleted and re-created. If you decide not to continue, the pattern configuration remains open and no changes are made to the pattern resources.

     When you make changes in the Pattern Configuration editor, the summary file and configuration XML file might be out of date because the *instance_name*_configuration.xml and the *instance_name*_summary.html files do not match the generated resources. The Problems view displays any warning messages. Double-click the warning message to open the relevant editor for the resource.

What to do next

Next: Now complete the following task:

- "Reviewing the pattern instance summary and tasks"

Related concepts:

"Patterns" on page 1310
A *pattern* is a reusable solution that encapsulates a tested approach to solving a common architecture, design, or deployment task in a particular context.

"Using patterns" on page 1312
Use patterns that are supplied with the WebSphere Message Broker Toolkit to create resources that are used to solve a specific business problem.

Related tasks:

"Generating a pattern instance" on page 1326
Generate resources from the pattern.

"Choosing a pattern" on page 1313
Select a pattern in the Patterns Explorer view to create resources to solve a specific business problem.

"Reviewing the pattern instance summary and tasks"
Showing the summary page for your pattern instance project.

Related reference:

"Resource types in the WebSphere Message Broker Toolkit" on page 6821
Links to information on projects and resource files.

Reviewing the pattern instance summary and tasks

Showing the summary page for your pattern instance project.

Before you begin

Before you start:

You must have completed the following tasks:

- "Generating a pattern instance" on page 1326

About this task

After generating the pattern instance, you might have to do additional tasks to complete your solution and to support its operation following instance. These tasks are located in a summary file that is created in the pattern reference project, which has the same name as the instance name.

The summary file lists the resources that were created for the pattern instance, based on the pattern parameter values that you entered in the Configure Pattern Parameters page. The summary also lists any outstanding tasks that you must complete to make the solution operational. For example, you might have to configure the message set details that are required by one or more input nodes that form part of the overall solution, or create WebSphere MQ queues for the nodes.

When a pattern has been configured and generated once, it contains both a configuration file and a summary file. If you edit and save the configuration file, both the configuration and the summary files display a warning symbol and a warning also shows in the Problems window indicating that the two files are out of sync. To get the files back in sync, regenerate the pattern.

To view the summary and tasks:

Procedure

1. Click **Windows** > **Show view** > **Tasks** to open the Task view.
2. In the **Path** column, find your pattern instance project name, which includes the pattern instance name that you used when you created the pattern instance.
3. To open the Summary page, double-click anywhere on the task, or right-click and select **Go To**.
4. Complete the tasks that are shown in the summary file.

What to do next

Next: Now complete the following task:

- "Adding files to a broker archive" on page 3223

Related concepts:

"Using patterns" on page 1312
Use patterns that are supplied with the WebSphere Message Broker Toolkit to create resources that are used to solve a specific business problem.

Related tasks:

"Choosing a pattern" on page 1313
Select a pattern in the Patterns Explorer view to create resources to solve a specific business problem.

"Generating a pattern instance" on page 1326
Generate resources from the pattern.

Related reference:

"Pattern instance projects" on page 6827
A pattern instance project contains project references to all the other projects in the workspace that relate to a specific pattern instance.

Importing an existing configuration

You can import a pattern configuration XML file from the workspace or file system and populate your current pattern instance with its values.

Before you begin

Before you start:

Before completing this task, you must have a compatible pattern configuration.

You must be in the WebSphere Message Broker Toolkit and have a pattern instance open.

Procedure

1. Open your current configuration. On the Configure Pattern Parameters page, click the **Open existing configuration** icon. The Open Pattern Configuration window opens.
2. You can select an existing pattern XML configuration file either from a workspace or file system location. Only configuration files created by the same pattern can be imported.
   - To select from a list of compatible XML configuration files that are already in the workspace, click **Workspace**. If you select a valid XML configuration file, the editor is configured with the values from the existing configuration file. You can then choose one of the following options:
     – Save the changes by clicking **OK**.
     – Return to the Configure Pattern Parameters page and discard the changes by clicking **Cancel**.
   - To select an XML configuration file from the file system by using a standard file system window, click **File System**. Select the file that you want to use, click **Open**.

   The configuration XML is validated against the corresponding pattern schema, if you select a configuration that is not valid, for example, from a different pattern, you receive notification. You can then either select the correct file, or cancel the action.
3. Share the pattern instance projects. To share pattern instance projects between WebSphere Message Broker Toolkit workspaces, you can export the pattern instance project into a directory in the file system or into a Project Interchange File (PIF). You can then import the pattern instance project into another workspace. For more information, see “Importing and exporting resources in a Project Interchange file” on page 1452.

What to do next

Next: Now complete the following task:

- “Generating a pattern instance” on page 1326

Related concepts:

“Patterns” on page 1310
A *pattern* is a reusable solution that encapsulates a tested approach to solving a common architecture, design, or deployment task in a particular context.

“Using patterns” on page 1312
Use patterns that are supplied with the WebSphere Message Broker Toolkit to create resources that are used to solve a specific business problem.

Related tasks:

“Generating a pattern instance” on page 1326
Generate resources from the pattern.

"Editing and regenerating a pattern" on page 1327
You can regenerate a pattern instance after you have edited a saved configuration in a pattern instance project.

"Creating a working set and focus on a pattern instance" on page 1319
Creating a working set for an imported project interchange file that contains a pattern instance project and its associated projects.

Related reference:

"Resource types in the WebSphere Message Broker Toolkit" on page 6821
Links to information on projects and resource files.

Pattern categories

Pattern categories are used to structure the display in the Patterns Explorer. A number of categories are provided, and you can also create your own.

A number of pattern categories are available in WebSphere Message Broker, from which you can choose the WebSphere Message Broker pattern that you want to use.

WebSphere Message Broker includes the following categories for built-in patterns. You can also use these pattern categories for your user-defined patterns, or you can create your own categories.

Message-based Integration
An Enterprise Service Bus can extend an existing messaging infrastructure by providing an environment for building and deploying infrastructure level message-based applications. Examples of these applications include routing and transformation services, and logging services. This environment can extend a single underlying messaging infrastructure or provide a bridge between different products and technologies.

Service Enablement
These patterns encapsulate functionality that does not have a service interface, and present this functionality through a service-oriented interface. These patterns represent the move from traditional enterprise application integration into service-oriented architectures, allowing existing assets to be reused in the new style without requiring radical change.

Service Virtualization
These patterns provide loose coupling between services by providing additional levels of direction through an Enterprise Service Bus. These patterns also address the requirements of mediation (for example, routing, protocol conversion, data transformation, and logging) between services, when addressing connectivity requirements in a service-oriented architecture.

Gateway
A Gateway is a part of a message or service bus that provides boundary functions that apply to all incoming messages, and are not format-dependent. Boundary functions typically use data from standard headers (at transport, SOAP, or data level) to determine what action to take, but are not required to understand the complete format of the message data or body. A Gateway pattern can then call a service directly, or call another pattern.

File Processing
An Enterprise Service Bus can provide a managed runtime environment for processing files locally or by using an FTP protocol. Typically this

processing involves activities including the transformation or translation of data held in the files, the shredding of files into multiple individual transaction records, the routing of records, the accumulation of records into target files, and the routing of files or records to specified locations.

Event-driven Integration
Event-driven architecture covers different application scenarios in which an Enterprise Service Bus plays a key role. These scenarios include integration with complex event processing engines that include the ability to filter information or event streams, distribute events in real time, and process events from physical devices, for example, detectors and sensors.

Application Integration
Application Integration is a collection of technologies and services that form a middleware to enable the integration of systems and applications across the enterprise.

For more information about individual built-in patterns, see "Built-in patterns."

Related concepts:

"Patterns" on page 1310
A *pattern* is a reusable solution that encapsulates a tested approach to solving a common architecture, design, or deployment task in a particular context.

"Using patterns" on page 1312
Use patterns that are supplied with the WebSphere Message Broker Toolkit to create resources that are used to solve a specific business problem.

"Built-in patterns"
A *built-in pattern* is a pattern that covers a set of commonly encountered message flow scenarios and that is packaged and released with WebSphere Message Broker.

"User-defined patterns" on page 1334
A *user-defined pattern* extends the function of WebSphere Message Broker so that you are able to create patterns that you can reuse within your organization.

Built-in patterns

A *built-in pattern* is a pattern that covers a set of commonly encountered message flow scenarios and that is packaged and released with WebSphere Message Broker.

The built-in patterns that are supplied in WebSphere Message Broker are shown in the following tables.

You can view patterns in the information center by using the links only when you use the information center that is integrated with the WebSphere Message Broker Toolkit, or when you use the online information center.

Message-based Integration patterns

Pattern name	Description
Message Correlator for WebSphere MQ: request-response with persistence	Use this pattern to accept requests from many client applications on a single queue, and to return responses to the correct client by using transactional flows and persistent WebSphere MQ messages.

Pattern name	Description
Message Correlator for WebSphere MQ: request-response without persistence	Use this pattern to accept requests from many client applications on a single queue, and to return responses to the correct client by using non-transactional flows and non-persistent WebSphere MQ messages.
Message Splitter for WebSphere MQ: one-way (for XML)	Use this pattern to split a large XML message into smaller elements for processing by one or more targets by using transactional flows and persistent WebSphere MQ messages.

Service Enablement patterns

Pattern name	Description
Service Facade to WebSphere MQ: one-way with acknowledgment	Use this pattern to present a Web service interface to clients and to fulfill the service requests by using a WebSphere MQ enabled application.
Service Facade to WebSphere MQ: request-response	Use this pattern to provide a Web service facade to functions that are accessible only through WebSphere MQ. This pattern creates a bridge between the synchronous HTTP protocol, which is typically used with Web services, and existing applications with WebSphere MQ interfaces that cannot be easily upgraded.
Service Access from WebSphere MQ: one-way	Use this pattern to process WebSphere MQ XML messages by using the data that the pattern contains to call a Web service. Use this pattern to bridge from the reliable WebSphere MQ messaging protocols of client application with the synchronous requests, to services to handle updates with an assurance that service failures, including timeouts, are reliably reported. This pattern provides loose coupling between client applications and service providers in timing, protocols, and transport. It is appropriate for service interfaces to existing systems.

Service Virtualization patterns

Pattern name	Description
Service Proxy: static endpoint	Use this pattern to provide decoupling between Web service requesters and Web service providers by routing through a virtual service that is bound directly to the target service provider.

File Processing patterns

Pattern name	Description
Record Distribution to WebSphere MQ: one-way	Use this pattern to bridge between two styles of integration, file based and transaction based.

Application Integration patterns

Pattern name	Description
Data distribution SAP to WebSphere MQ: one-way (for IDoc)	Use this pattern to process various types of IDocs that have a single program identifier without you being required to redeploy or rediscover existing message sets and adapters, even when you are adding different types of IDocs.

Related concepts:

"Patterns" on page 1310
A *pattern* is a reusable solution that encapsulates a tested approach to solving a common architecture, design, or deployment task in a particular context.

"Pattern categories" on page 1331
Pattern categories are used to structure the display in the Patterns Explorer. A number of categories are provided, and you can also create your own.

"Using patterns" on page 1312
Use patterns that are supplied with the WebSphere Message Broker Toolkit to create resources that are used to solve a specific business problem.

"User-defined patterns"
A *user-defined pattern* extends the function of WebSphere Message Broker so that you are able to create patterns that you can reuse within your organization.

User-defined patterns

A *user-defined pattern* extends the function of WebSphere Message Broker so that you are able to create patterns that you can reuse within your organization.

The *pattern author* creates a user-defined pattern to meet a business or technical requirement, and the *pattern user* configures a user-defined pattern that the pattern author has created. The user-defined pattern is available to a pattern user in the Patterns Explorer view in the Broker Application Development perspective of the WebSphere Message Broker Toolkit.

To create user-defined patterns, you must provide the implementation of the solution that reflects good practice in your organization. This implementation is known as an *exemplar*. An exemplar is a project that holds content for a pattern. An exemplar contains message flows and other resources, such as source code, Java classes for JavaCompute nodes, ESQL modules, message maps, test client, XML files, and style sheet files. Exemplars are used to create pattern plug-ins by configuring a *pattern authoring project*.

You can use the Pattern Authoring editor to configure a pattern and create the pattern plug-ins that implement the pattern. You can distribute the pattern plug-ins so that other people can use your pattern. Your pattern is displayed in the Patterns Explorer view and can be used in the same way as the built-in patterns.

To create a user-defined pattern, see "Creating a user-defined pattern" on page 1336.

Related concepts:

"Patterns" on page 1310
A *pattern* is a reusable solution that encapsulates a tested approach to solving a common architecture, design, or deployment task in a particular context.

"Pattern categories" on page 1331
Pattern categories are used to structure the display in the Patterns Explorer. A number of categories are provided, and you can also create your own.

"Using patterns" on page 1312
Use patterns that are supplied with the WebSphere Message Broker Toolkit to create resources that are used to solve a specific business problem.

"Built-in patterns" on page 1332
A *built-in pattern* is a pattern that covers a set of commonly encountered message flow scenarios and that is packaged and released with WebSphere Message Broker.

Related tasks:

"Creating a pattern authoring project" on page 1337
Create a pattern authoring project and choose an exemplar for the project.

"Extending a user-defined pattern" on page 1340
Extend a user-defined pattern to customize its behavior so that it is easier to use or to provide guidance to pattern users.

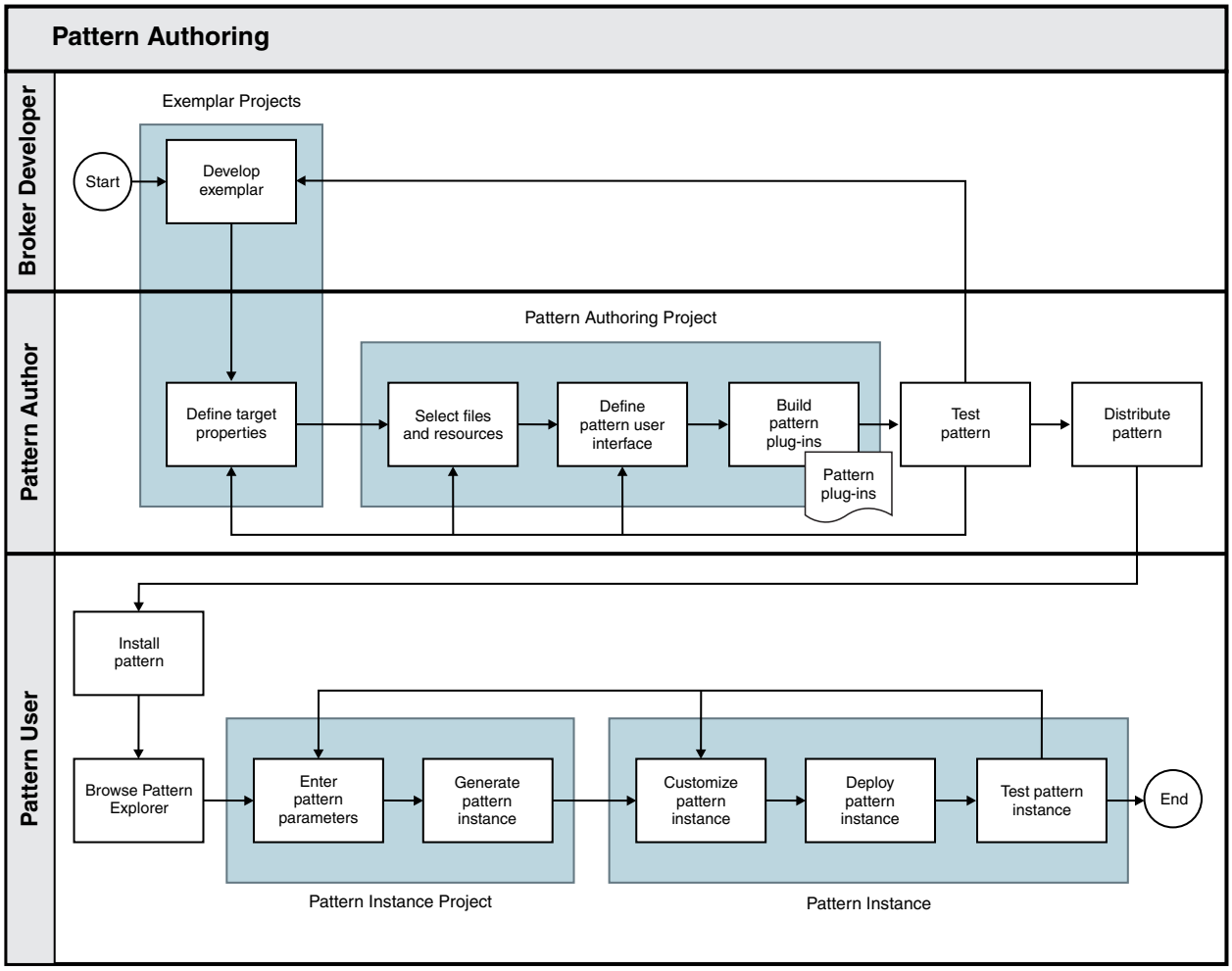
"Building pattern plug-ins" on page 1395
You must build the pattern plug-ins before you can distribute user-defined patterns to pattern users.

"Testing a user-defined pattern" on page 1396
Ensure that you test your user-defined pattern before you share it with your pattern users.

"Packaging and distributing pattern plug-ins" on page 1397
Package the plug-ins for your user-defined pattern into a pattern archive so that pattern users can download and install the pattern on their own system.

Creating a user-defined pattern

The workflow showing the actions required for pattern authoring.



The three stages of creating a user-defined pattern are performed by the WebSphere Message Broker developer, the pattern author, and the pattern user.

1. The WebSphere Message Broker developer develops the exemplar. The exemplar is fundamental to the pattern authoring process. It assumes that the exemplar is the starting point for a pattern. Some modifications might be required to the exemplar to prepare it for pattern authoring.
2. The pattern author creates a pattern plug-in from the exemplar. Authoring a pattern is a design activity. Some of the pattern authoring focuses on the resources in WebSphere Message Broker, for example, defining the target properties for the pattern and configuring the user interface that is presented to the pattern user.

   The pattern author then shares the user-defined pattern with the pattern user. Patterns are most useful when they are shared with a user base. This user base can be within an organization, or a broader community, for example, open source licensing.
3. The pattern user receives a user-defined pattern, and uses it in accordance with the requirements of the organization.

Pattern authoring

The following sample shows how to build a WebSphere Message Broker pattern. The sample provides an example message flow project that calculates the sunrise and sunset times in a PHPCompute node. The sample also provides a pattern authoring project that configures a pattern.

- Solar Pattern Authoring

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

To create a user-defined pattern, complete the following tasks. "Extending a user-defined pattern" is an optional set of tasks that you can use to add features to your user-defined pattern. For example, by extending a user-defined pattern you can create user-defined patterns that include parameters that pattern users can modify.

1. "Creating a pattern authoring project"
2. "Selecting the source files to use for a user-defined pattern" on page 1338
3. Optional: "Adding and removing the project references for a user-defined pattern" on page 1339
4. Optional: "Extending a user-defined pattern" on page 1340
5. "Building pattern plug-ins" on page 1395
6. "Testing a user-defined pattern" on page 1396
7. "Packaging and distributing pattern plug-ins" on page 1397

When you have created pattern plug-ins for your user-defined pattern, pattern users can install them to use in WebSphere Message Broker projects, see "Downloading and installing a pattern archive" on page 1400.

Related concepts:

"Patterns" on page 1310
A *pattern* is a reusable solution that encapsulates a tested approach to solving a common architecture, design, or deployment task in a particular context.

"Pattern categories" on page 1331
Pattern categories are used to structure the display in the Patterns Explorer. A number of categories are provided, and you can also create your own.

"Using patterns" on page 1312
Use patterns that are supplied with the WebSphere Message Broker Toolkit to create resources that are used to solve a specific business problem.

"Built-in patterns" on page 1332
A *built-in pattern* is a pattern that covers a set of commonly encountered message flow scenarios and that is packaged and released with WebSphere Message Broker.

"User-defined patterns" on page 1334
A *user-defined pattern* extends the function of WebSphere Message Broker so that you are able to create patterns that you can reuse within your organization.

Creating a pattern authoring project:

Create a pattern authoring project and choose an exemplar for the project.

About this task

Procedure

To create a pattern authoring project:

1. Click **File** > **New** > **Other**. A window opens in which you can select a wizard.
2. Expand **Message Broker - Flow Development**, select **Pattern Authoring Project**, click **Next**. The New Pattern Authoring Project wizard opens.
3. Enter a pattern name and a project name for your pattern, then click **Next**.
4. Select the project references that you require. These references are the exemplars for the user-defined pattern.
5. Click **Finish**.

Results

Your pattern folder is displayed in the Projects section of the **Broker Development** view and in the Pattern Authoring editor.

What to do next

Next:

Select the source files. For more information, see "Selecting the source files to use for a user-defined pattern."

Related concepts:

"User-defined patterns" on page 1334
A *user-defined pattern* extends the function of WebSphere Message Broker so that you are able to create patterns that you can reuse within your organization.

"Creating a user-defined pattern" on page 1336
The workflow showing the actions required for pattern authoring.

Selecting the source files to use for a user-defined pattern:

Select the source files to include in your pattern.

Before you begin

Before you start:

Complete the following task:

- "Creating a pattern authoring project" on page 1337

Procedure

To select the source files to use in the user-defined pattern:

1. In the Pattern Authoring editor, click the **Source Files** tab.
2. In the **Select Source Files** section, select the source files in the referenced projects that you want to include in your pattern.

What to do next

Next:

- Extend the pattern, see "Extending a user-defined pattern" on page 1340.

- Change the project references, see “Adding and removing the project references for a user-defined pattern.”
- Build the pattern, see “Building pattern plug-ins” on page 1395.

Related concepts:

“User-defined patterns” on page 1334
A *user-defined pattern* extends the function of WebSphere Message Broker so that you are able to create patterns that you can reuse within your organization.

“Creating a user-defined pattern” on page 1336
The workflow showing the actions required for pattern authoring.

Related tasks:

“Creating a pattern authoring project” on page 1337
Create a pattern authoring project and choose an exemplar for the project.

Adding and removing the project references for a user-defined pattern:

Change the project references for a user-defined pattern so that you can change the source files in the user-defined pattern.

Before you begin

Before you start:

Complete the following task:

- “Creating a pattern authoring project” on page 1337

Procedure

To add or remove project references:

1. In the Pattern Authoring editor, click the **Source Files** tab. Click **Change Project References**.
2. To add a new project reference, select the project reference that you want to add and click **OK**. If you add a new project reference, none of the source files within the project reference are selected. To select source files for the new project reference, see “Selecting the source files to use for a user-defined pattern” on page 1338. If your user-defined pattern requires referenced projects but they are not included as source files in the pattern, you must distribute the referenced projects to pattern users to ensure the user-defined pattern works.
3. To remove a project reference, clear the project reference that you want to remove and click **OK**. If you remove a referenced project from the workspace, the source files are removed from the pattern authoring project but the project reference remains. If you later add the project back into the workspace, the source files are added to the pattern authoring project.

What to do next

Next:

- You can extend the pattern; see “Extending a user-defined pattern” on page 1340.
- If the pattern is complete, you can now build the pattern; see “Building pattern plug-ins” on page 1395.

Related concepts:

“User-defined patterns” on page 1334
A *user-defined pattern* extends the function of WebSphere Message Broker so that

you are able to create patterns that you can reuse within your organization.

"Creating a user-defined pattern" on page 1336
The workflow showing the actions required for pattern authoring.

Related tasks:

"Creating a pattern authoring project" on page 1337
Create a pattern authoring project and choose an exemplar for the project.

"Selecting the source files to use for a user-defined pattern" on page 1338
Select the source files to include in your pattern.

Extending a user-defined pattern

Extend a user-defined pattern to customize its behavior so that it is easier to use or to provide guidance to pattern users.

Before you begin

Before you start:

Complete the following tasks:

- "Creating a pattern authoring project" on page 1337
- "Selecting the source files to use for a user-defined pattern" on page 1338

About this task

A basic user-defined pattern is a copy of an existing message flow, but by extending a user-defined pattern you can allow the pattern user to customize the pattern.

Procedure

You can extend a user-defined pattern in the following ways:

- Create documentation for the pattern. Creating documentation provides guidance to your pattern users. To update the pattern documentation, see "Creating documentation for a pattern" on page 1342.
- Define the target properties. A user-defined pattern can change the value of user-defined properties, promoted node properties, and node properties in a message flow. A property that is changed by a pattern is called a target property. Pattern users create a pattern instance by configuring pattern parameters. The values of the pattern parameters that are configured by a pattern user can be used to configure the target properties in the pattern instance. To define target properties, see "Defining the target properties" on page 1342.
- Define the user interface. You can customize how the pattern parameters are displayed to pattern users by defining the user interface. For example, you can rename or group pattern parameters, and you can define default values for pattern parameters. To define the user interface, see "Defining the user interface" on page 1343, "Adding and editing parameter groups" on page 1345, and "Enabling parameter groups" on page 1347.
- Configure the categories in the Patterns Explorer view. When a pattern user imports a user-defined pattern, it is shown in the Patterns Explorer view. You can select which category the user-defined pattern is assigned to in the Patterns Explorer view and create new categories. To configure the categories, see "Creating and configuring categories" on page 1350.
- Configure the SOAP nodes. To use SOAP nodes in user-defined patterns, see "Configuring SOAP nodes for user-defined patterns" on page 1351.

- Change pattern parameter IDs. Pattern parameter IDs are used to refer to pattern parameters in XPath, Java, and PHP code. You can write the code to modify pattern parameters or pattern instances. Default parameter IDs are assigned, but you can change them to custom IDs. To change parameter IDs, see "Changing pattern parameter IDs" on page 1352.
- Transform the pattern parameters. You can calculate values for pattern parameters from other pattern parameters by using XPath expressions. To set up transformations for pattern parameters and then test the transformations, see "Transforming pattern parameters" on page 1353 and "Testing a transformation expression" on page 1355.
- Enable or disable the pattern parameters. You can use XPath expressions to control whether a pattern parameter can be modified by a pattern user based on the values of other pattern parameters. To set up enabling expressions for pattern parameters and then test the expressions, see "Enabling pattern parameters" on page 1357 and "Testing an enabling expression" on page 1358.
- Use enumerated values for the pattern parameters. You can set up enumerated types for pattern parameters so that pattern users have a predefined list of values for the pattern parameters. If a target property has a list of possible values, the pattern authoring tool generates an enumerated type for that target property. You can modify the enumerated type that is automatically created. To use enumerated types see "Using enumerated values for pattern parameters" on page 1360 and "Creating enumerated types for pattern parameters" on page 1361.
- Modify pattern instances by using Java or PHP. You can write code in Java or PHP that modifies pattern instances when a pattern user generates an instance of a user-defined pattern. For example, you can write code to modify the structure of a message flow based on the values of pattern parameters. To use Java or PHP in user-defined patterns, see "Modifying pattern instances by using Java or PHP" on page 1364.

What to do next

Next:

After extending your user-defined pattern, you must build the pattern plug-in, see "Building pattern plug-ins" on page 1395.

Related concepts:

"User-defined patterns" on page 1334
A *user-defined pattern* extends the function of WebSphere Message Broker so that you are able to create patterns that you can reuse within your organization.

"Creating a user-defined pattern" on page 1336
The workflow showing the actions required for pattern authoring.

Related tasks:

"Creating a pattern authoring project" on page 1337
Create a pattern authoring project and choose an exemplar for the project.

"Selecting the source files to use for a user-defined pattern" on page 1338
Select the source files to include in your pattern.

"Generating a pattern instance" on page 1326
Generate resources from the pattern.

Creating documentation for a pattern:

When you create a pattern authoring project, a default HTML pattern specification is created as part of the project. You can edit this pattern specification to document your user-defined pattern.

Before you begin

Before you start:

Complete the following task:

- "Creating a pattern authoring project" on page 1337

About this task

Procedure

To edit the pattern specification, complete the following steps.

1. Open the Projects section of the Broker Development view.
2. Expand your pattern authoring project, and expand **Pattern Specification**.
3. Right-click `overview.htm` and select **Open With** > **HTML Editor**. The default user-defined pattern specification is shown in the HTML editor.
4. Edit the content of the pattern specification in the HTML editor. The default specification contains example links to suggested HTML files. To ensure that these links work correctly, you must create the required files and edit the links in the default specification. If you do not require these additional pages, delete the example links from the default specification.
5. To save the changes, click **File** > **Save**.
6. To view the updated pattern specification, click the **Categories** tab of the Pattern Authoring editor.

Related concepts:

"User-defined patterns" on page 1334
A *user-defined pattern* extends the function of WebSphere Message Broker so that you are able to create patterns that you can reuse within your organization.

"Creating a user-defined pattern" on page 1336
The workflow showing the actions required for pattern authoring.

Defining the target properties:

Identify the target properties in the workspace project so that you can define the pattern interface and pattern users can modify the pattern parameters.

Before you begin

Before you start:

Create the pattern authoring project and select the source files. For information about creating a pattern authoring project, see "Creating a pattern authoring project" on page 1337. For information about selecting the source files, see "Selecting the source files to use for a user-defined pattern" on page 1338.

Procedure

To identify the target properties that you want to use in the workspace project:

1. In the Broker Development view, double-click the message flow that you want to use.
2. Open the Select Target Properties window by completing one of the following steps:
   - To select a node property as a target property, in the Message Flow editor, right-click the node that you want to use and click **Pattern** > **Select Target Properties**.
   - To select a user-defined property as a target property, in the Message Flow editor, right-click the canvas and click **Pattern** > **Select Target Properties**.
3. In the Select Target Properties window, select the target properties that you want to use in the pattern.
4. Close the Select Target Properties window.
5. Save the message flow.

Results

The selected target properties are displayed in the Target Properties panel of the **Source Files** tab in the Pattern Authoring editor.

What to do next

Next:

Define the user interface; see "Defining the user interface."

Related concepts:

"User-defined patterns" on page 1334
A *user-defined pattern* extends the function of WebSphere Message Broker so that you are able to create patterns that you can reuse within your organization.

"Creating a user-defined pattern" on page 1336
The workflow showing the actions required for pattern authoring.

Related tasks:

"Extending a user-defined pattern" on page 1340
Extend a user-defined pattern to customize its behavior so that it is easier to use or to provide guidance to pattern users.

Defining the user interface:

To control how pattern users view and edit pattern parameters in the Pattern Instance editor after the pattern user has created an instance of a user-defined pattern, you must define the user interface.

Before you begin

Before you start:

Complete the following tasks:

- "Creating a pattern authoring project" on page 1337
- "Selecting the source files to use for a user-defined pattern" on page 1338
- "Defining the target properties" on page 1342

About this task

When a pattern user creates an instance of a user-defined pattern, the parameter groups and pattern parameters that are defined in the **Pattern Configuration** tab of the Pattern Authoring editor are displayed in the Pattern Instance editor. If the **Pattern Configuration** tab displays groups or parameters that you do not want in the final pattern, you must delete them from the **Pattern Configuration** tab before building the pattern plug-ins.

To define the user interface, open the **Pattern Configuration** tab of the Pattern Authoring editor. In the **Groups and Parameters** list, entries are shown in a hierarchy in which groups contain pattern parameters, and pattern parameters contain target properties.

Procedure

To define the user interface, choose one or more of the following options:

1. You can define the groups in which pattern parameters are displayed and assign target properties to pattern parameters:
   a. To define the groups in which pattern parameters are displayed in the Pattern Instance editor, drag pattern parameters to groups. To add new groups or edit existing groups, see "Adding and editing parameter groups" on page 1345. To control whether parameter groups are enabled by using XPath expressions, see "Enabling parameter groups" on page 1347.
   b. To assign target properties to pattern parameters, drag target properties to pattern parameters. If the target property and pattern parameter do not use a compatible editor, an error message is displayed; see step 5 on page 1345 to change the parameter editor. If you drag multiple target properties to one pattern parameter, all target properties in that pattern parameter are populated with the same value that is entered for the pattern parameter by the pattern user.
2. To add a new pattern parameter, click **Add Parameter**, and click **OK**. To configure the new parameter, see steps 4 and 5 on page 1345.
3. To remove a pattern parameter or parameter group, select the pattern parameter or parameter group to remove and click **Delete**.
   - To remove a parameter group it must have no pattern parameters assigned to it.
   - To remove a pattern parameter it must have no target properties assigned to it. To remove a target property so that you can remove a pattern parameter, clear the target property in the Message Flow editor of the Broker Development view. For more information about defining target properties, see "Defining the target properties" on page 1342. When you remove a target property in the Message Flow editor, the associated pattern parameter and parameter group are not automatically deleted in the **Pattern Configuration** tab of the Pattern Authoring editor.
4. You can change the name of a pattern parameter, create help text, create a field prompt, and configure parameter options in the **Basic** tab. In the **Groups and Parameters** list, select the pattern parameter and click **Edit**, or double-click the pattern parameter. The Edit Parameter window opens.
   - To change the name of the pattern parameter, enter a name for the pattern parameter in the **Display Name** field and click **OK**. The name for the pattern parameter is changed to the new name.

- To change parameter options, select or clear the **Hide the parameter**, **Configure during deployment**, and **Mandatory parameter** check boxes, as required.
- To enter a field prompt, type a text into the **Field prompt** field.
- To provide help text for the pattern parameter, type the text in the **Help Text (HTML)** field.
- To close the Edit Parameter window after you have completed your changes, click **OK**.

5. You can change the parameter editor, and set a default value for a pattern parameter in the **Editor** tab. In the **Groups and Parameters** list, select the pattern parameter and click **Edit**, or double-click the pattern parameter. The Edit Parameter window opens. Click the **Editor** tab.
   - To change the parameter editor, select the required editor in the **Parameter editor** list.

     To use an enumerated type for the parameter, select **Drop Down Selection** in the **Parameter editor** list, then select an enumerated type for the parameter in the **Type selection** field. For more information, see "Using enumerated values for pattern parameters" on page 1360.
   - To enter a default value for a pattern parameter, if **Parameter editor** is `Drop Down Selection` or `Check Box`, select a value from the **Default value** list. Otherwise, enter a value in the **Default value** field.
   - To close the Edit Parameter window after you have completed your changes, click **OK**.

What to do next

Next:

You can build the pattern plug-ins, see "Building pattern plug-ins" on page 1395.

Related concepts:

"User-defined patterns" on page 1334
A *user-defined pattern* extends the function of WebSphere Message Broker so that you are able to create patterns that you can reuse within your organization.

"Creating a user-defined pattern" on page 1336
The workflow showing the actions required for pattern authoring.

Related tasks:

"Creating a pattern authoring project" on page 1337
Create a pattern authoring project and choose an exemplar for the project.

"Selecting the source files to use for a user-defined pattern" on page 1338
Select the source files to include in your pattern.

"Extending a user-defined pattern" on page 1340
Extend a user-defined pattern to customize its behavior so that it is easier to use or to provide guidance to pattern users.

"Defining the target properties" on page 1342
Identify the target properties in the workspace project so that you can define the pattern interface and pattern users can modify the pattern parameters.

Adding and editing parameter groups:

Edit parameter groups to control how pattern parameters are displayed to pattern users when they create pattern instances from a user-defined pattern.

Before you begin

Before you start:

Complete the following tasks:

- "Creating a pattern authoring project" on page 1337
- "Selecting the source files to use for a user-defined pattern" on page 1338
- "Defining the target properties" on page 1342

About this task

After defining target properties, a default list of parameter groups is created. You can edit the default groups or add new groups.

Procedure

- To add a new group:
  1. In the **Pattern Configuration** tab of the Pattern Authoring editor, click **Add Group**. The Add Group window opens.
  2. Enter a display name for the group in the **Display name** field.
  3. Enter a description for the group in the **Description** field.
  4. The **Generate help documentation** check box is selected by default. If you do not want to show the help documentation for the parameters in the group, clear the **Generate help documentation** check box. To add help documentation for pattern parameters, see "Defining the user interface" on page 1343.
  5. If you do not want the group to display with a surrounding box in the Configure Pattern Parameters page, clear the **Display parameters in a group box** check box. If a group is displayed in a group box, pattern users can expand or collapse the group box to display or hide the parameters in that group. Parameters in a group that is not shown in a group box are always visible.
  6. Click **OK**. The new group is shown in the Groups and Parameters list.
- To edit an existing group:
  1. In the **Pattern Configuration** tab of the Pattern Authoring editor, double-click the group name that you want to edit, or select the group name and click **Edit**. The Edit Group window opens.
  2. To change the display name, enter a display name for the group in the **Display name** field.
  3. To change the description, enter a description for the group in the **Description** field.
  4. If you want to show help documentation for the parameters in the group, ensure the **Generate help documentation** check box is selected. To add help documentation for pattern parameters, see "Defining the user interface" on page 1343.
  5. If you want the group to display with a surrounding box in the Configure Pattern Parameters page, ensure the **Display parameters in a group box** check box is selected. If a group is displayed in a group box, pattern users can expand or collapse the group box to display or hide the parameters in that group. Parameters in a group that is not shown in a group box are always visible.
  6. Click **OK**. The modified group is shown in the Groups and Parameters list.

What to do next

Next:

You can choose whether a parameter group is enabled based on the value of pattern parameters; see "Enabling parameter groups."

Related concepts:

"User-defined patterns" on page 1334
A *user-defined pattern* extends the function of WebSphere Message Broker so that you are able to create patterns that you can reuse within your organization.

"Creating a user-defined pattern" on page 1336
The workflow showing the actions required for pattern authoring.

Related tasks:

"Creating a pattern authoring project" on page 1337
Create a pattern authoring project and choose an exemplar for the project.

"Selecting the source files to use for a user-defined pattern" on page 1338
Select the source files to include in your pattern.

"Defining the target properties" on page 1342
Identify the target properties in the workspace project so that you can define the pattern interface and pattern users can modify the pattern parameters.

"Extending a user-defined pattern" on page 1340
Extend a user-defined pattern to customize its behavior so that it is easier to use or to provide guidance to pattern users.

Enabling parameter groups:

In a user-defined pattern, control whether parameter groups are enabled based on the values of pattern parameters.

Before you begin

Before you start:

Complete the following tasks:

- "Creating a pattern authoring project" on page 1337
- "Selecting the source files to use for a user-defined pattern" on page 1338
- "Defining the target properties" on page 1342

About this task

By default, when pattern users configure an instance of a user-defined pattern, all the parameter groups that are defined in the user-defined pattern are enabled. Enabled parameter groups and pattern parameters within those groups are visible to pattern users in the Configure Parameters page. You can control whether a parameter group in an instance of a user-defined pattern is enabled by using an XPath expression:

- If the expression evaluates to Boolean true, the parameter group is enabled. When a pattern user configures an instance of a user-defined pattern, the parameter group and pattern parameters within that group are visible to pattern users in the Configure Parameters page. The values of pattern parameters within enabled parameter groups populate the target properties of the user-defined pattern.

- If the expression evaluates to Boolean `false`, the parameter group is disabled. When a pattern user configures an instance of a user-defined pattern, the parameter group is not shown to pattern users in the Configure Parameters page. Pattern parameters within disabled groups are not shown to pattern users and the values of those parameters do not populate the target properties in an instance of the user-defined pattern.

If no XPath expression is entered for a parameter group, the parameter group is enabled. For XPath reference information, including information about XPath functions, see W3C XPath 1.0 Specification.

The `pp:getValue()` function is included, in addition to the functions in the XPath 1.0 Specification. The `pp:getValue()` function takes the parameter ID of a pattern parameter and returns the value of that pattern parameter. To see the parameter ID for a pattern parameter:

1. In the **Pattern Configuration** tab of the Pattern Authoring editor, double-click a parameter group, or select a parameter group and click **Edit**. The Edit Group window opens.
2. Click the **Visibility** tab. The parameter IDs for pattern parameters are shown in the **Parameter ID** column of the Pattern Parameters table. To change parameter IDs, see "Changing pattern parameter IDs" on page 1352.

Procedure

To control whether a pattern parameter group is enabled by using an XPath expression:

1. In the **Pattern Configuration** tab of the Pattern Authoring editor, double-click the parameter group that you want to control, or select the parameter group and click **Edit**. The Edit Group window opens.
2. Click the **Enable** tab. Create an XPath expression for your chosen parameter group:
   - To select a function:
     a. Expand **Boolean**, **Number**, **Pattern**, or **String** in the Functions section, then click a function. The function is displayed in the **Function name** field.
     b. Click **Use**. The function is inserted into the **Expression** field at the cursor.
   - To select an operator:
     a. Click an operator in the Operators section. The operator is displayed in the **Operator** field.
     b. Click **Use**. The operator is inserted into the **Expression** field at the cursor.
   - To select a pattern parameter:
     a. Click a pattern parameter in the Pattern Parameters table. The parameter ID shown in the **Parameter ID** column of the Pattern Parameters table for the chosen parameter is displayed in the **Parameter ID** field.
     b. Click **Use**. The parameter ID is inserted into the **Expression** field at the cursor. To change parameter IDs, see "Changing pattern parameter IDs" on page 1352.
   - You can also edit the expression directly in the **Expression** field.
3. Repeat the actions in step 2, as required, to create your XPath expression.
4. You can now choose whether to test your expression:
   - To test your expression, see "Testing a group enabling expression" on page 1349.

- To accept the expression without testing it, click **OK**. The Edit Group window closes.

Related concepts:

"User-defined patterns" on page 1334
A *user-defined pattern* extends the function of WebSphere Message Broker so that you are able to create patterns that you can reuse within your organization.

"Creating a user-defined pattern" on page 1336
The workflow showing the actions required for pattern authoring.

Related tasks:

"Creating a pattern authoring project" on page 1337
Create a pattern authoring project and choose an exemplar for the project.

"Selecting the source files to use for a user-defined pattern" on page 1338
Select the source files to include in your pattern.

"Extending a user-defined pattern" on page 1340
Extend a user-defined pattern to customize its behavior so that it is easier to use or to provide guidance to pattern users.

"Defining the target properties" on page 1342
Identify the target properties in the workspace project so that you can define the pattern interface and pattern users can modify the pattern parameters.

"Testing a group enabling expression"
In a user-defined pattern, test an XPath expression that is used for enabling parameter groups.

"Changing pattern parameter IDs" on page 1352
Change default pattern parameter IDs to make it easier to write XPath, Java, and PHP code that uses parameter IDs.

Testing a group enabling expression:

In a user-defined pattern, test an XPath expression that is used for enabling parameter groups.

Before you begin

Before you can complete this task, you must have completed the following tasks:

- "Creating a pattern authoring project" on page 1337
- "Selecting the source files to use for a user-defined pattern" on page 1338
- "Defining the target properties" on page 1342
- "Enabling parameter groups" on page 1347

Procedure

To test your XPath expression:

1. If the **Enable** tab in the Edit Group window is not already open, complete the following steps to open the tab:
   a. In the **Pattern Configuration** tab of the Pattern Authoring editor, select the parameter group containing the XPath expression that you want to test and click **Edit**. The Edit Group window opens.
   b. Click the **Enable** tab.
2. To test the expression, click **Evaluate**. The expression is evaluated by using the values in the **Test Values** column of the Pattern Parameters section, and the result is shown in the **Result** field. The result shows whether the parameter group is enabled or disabled, and the value of the expression is shown in

parentheses. If the value is `true`, the associated group is enabled; if the value is `false`, the group is disabled. Use the following table to determine the value of your XPath expression.

Result data type	Evaluates to true	Evaluates to false
Boolean	`true`	`false`
Numeric	Any non-zero value	`0` or `0.0`
String	Any string that returns `true` with a match that is not case-sensitive	Any string that does not return `true` with a match that is not case-sensitive

3. To change the test value for a parameter, select the parameter in the Pattern Parameters section and enter the required test value in the **Test value** field. Click **Set**. The new test value is shown in the Pattern Parameters section.
4. Repeat steps 2 on page 1349 and 3, as required, to test the XPath expression. If you want to modify the expression, see "Enabling parameter groups" on page 1347.
5. When you have finished testing the expression, click **OK**. The Edit Group window closes.

Related concepts:

"User-defined patterns" on page 1334
A *user-defined pattern* extends the function of WebSphere Message Broker so that you are able to create patterns that you can reuse within your organization.

Related tasks:

"Creating a pattern authoring project" on page 1337
Create a pattern authoring project and choose an exemplar for the project.

"Selecting the source files to use for a user-defined pattern" on page 1338
Select the source files to include in your pattern.

"Extending a user-defined pattern" on page 1340
Extend a user-defined pattern to customize its behavior so that it is easier to use or to provide guidance to pattern users.

"Defining the target properties" on page 1342
Identify the target properties in the workspace project so that you can define the pattern interface and pattern users can modify the pattern parameters.

"Enabling parameter groups" on page 1347
In a user-defined pattern, control whether parameter groups are enabled based on the values of pattern parameters.

Creating and configuring categories:

Configure categories to which to assign user-defined patterns in the Patterns Explorer view.

Before you begin

Before you start:

Complete the following task:

- "Creating a pattern authoring project" on page 1337

About this task

You can drag your user-defined pattern into any existing category, or you can create a new category. If you create a category, you can write a category specification that provides information about the category for other users. User-defined patterns are shown within these categories in the Patterns Explorer view.

Procedure

To create a new category, update the category specification, assign your pattern to a category, rename a category, or remove a category, begin by opening the **Categories** tab of the Pattern Authoring editor. You can now choose one or more of the following options:

- To create a new category as a subcategory of an existing category:
  1. Select an existing category.
  2. Click **Add Category**. The Add Category window opens.
  3. Enter the name of your new category and click **OK**. The new category is added to the category list under the category previously selected. The Save Your Pattern File window opens, asking you to confirm whether you want to save your patterns file. Click **Yes**.
- To edit the category specification for a category:
  1. Double-click the category. The HTML editor opens containing the default category specification.
  2. Update the category specification to describe the category, and click **File** > **Save** to save any changes.
- To assign your user-defined pattern to a category, drag your user-defined pattern to your chosen category.
- To rename a category:
  1. Click **Rename Category**. The Edit Category window opens.
  2. Enter the new name, click **OK**. The Save Your Pattern File window opens, asking you to confirm whether you want to save your patterns file.
- To remove a category, click **Remove category**. The Save Your Pattern File window opens, asking you to confirm whether you want to save your patterns file. Click **Yes**.

Related concepts:

"User-defined patterns" on page 1334
A *user-defined pattern* extends the function of WebSphere Message Broker so that you are able to create patterns that you can reuse within your organization.

"Creating a user-defined pattern" on page 1336
The workflow showing the actions required for pattern authoring.

Related tasks:

"Creating a pattern authoring project" on page 1337
Create a pattern authoring project and choose an exemplar for the project.

"Extending a user-defined pattern" on page 1340
Extend a user-defined pattern to customize its behavior so that it is easier to use or to provide guidance to pattern users.

Configuring SOAP nodes for user-defined patterns:

Configure the target properties on SOAPInput, SOAPRequest, and SOAPAsyncRequest nodes by using a WSDL file.

Before you begin

Before you can complete this task, you must have completed the following tasks:

- "Creating a pattern authoring project" on page 1337

Procedure

To configure target properties on SOAP nodes by using a WSDL file:

1. Select an exemplar that uses a SOAPInput, SOAPRequest, or SOAPAsyncRequest node. See "Selecting the source files to use for a user-defined pattern" on page 1338.
2. Define the target properties and ensure that the `WSDL file name` target property is selected as a target property in the SOAPInput, SOAPRequest, or SOAPAsyncRequest node. See "Defining the target properties" on page 1342.

What to do next

When the pattern user generates an instance of the user-defined pattern, the WSDL file name is presented as a pattern parameter. After the pattern user inputs the WSDL file name, the properties in the WSDL file are used to configure the SOAP node target properties. Examples of WSDL properties are the port type, binding, and service port values.

The target properties are configured with the first occurrence of each WSDL property.

Related concepts:

"User-defined patterns" on page 1334
A *user-defined pattern* extends the function of WebSphere Message Broker so that you are able to create patterns that you can reuse within your organization.

"Creating a user-defined pattern" on page 1336
The workflow showing the actions required for pattern authoring.

Related tasks:

"Creating a pattern authoring project" on page 1337
Create a pattern authoring project and choose an exemplar for the project.

"Extending a user-defined pattern" on page 1340
Extend a user-defined pattern to customize its behavior so that it is easier to use or to provide guidance to pattern users.

Changing pattern parameter IDs:

Change default pattern parameter IDs to make it easier to write XPath, Java, and PHP code that uses parameter IDs.

Before you begin

Before you can complete this task, you must have completed the following tasks:

- "Creating a pattern authoring project" on page 1337
- "Selecting the source files to use for a user-defined pattern" on page 1338
- "Defining the target properties" on page 1342

About this task

Pattern parameter IDs uniquely identify pattern parameters. As a pattern author, you use pattern parameter IDs when writing XPath expressions to transform or enable pattern parameters, or when writing Java or PHP code to modify pattern instances. Every pattern parameter in a user-defined pattern has a default pattern parameter ID assigned, but you can change pattern parameter IDs from the default values so that they are easier to identify in your code.

Procedure

To change pattern parameter IDs from the default values:

1. In the **Pattern Configuration** tab of the Pattern Authoring editor, double-click the parameter for which you want to change the parameter ID, or select the parameter and click **Edit**. The Edit Parameter window opens.
2. In the **Parameter ID** field of the **Basic** tab, enter your new parameter ID. The new parameter ID can be made up of ASCII characters, but must start with a non-numeric character.
3. Click **OK**. The Edit Parameter window closes.

What to do next

- To use the new parameter ID in an XPath expression to transform pattern parameters, see "Transforming pattern parameters."
- To use the new parameter ID in an XPath expression to enable pattern parameters, see "Enabling pattern parameters" on page 1357.
- To use the new parameter ID in Java or PHP code to modify pattern instances, see "Modifying pattern instances by using Java or PHP" on page 1364.

Related concepts:

"User-defined patterns" on page 1334
A *user-defined pattern* extends the function of WebSphere Message Broker so that you are able to create patterns that you can reuse within your organization.

Related tasks:

"Creating a pattern authoring project" on page 1337
Create a pattern authoring project and choose an exemplar for the project.

"Selecting the source files to use for a user-defined pattern" on page 1338
Select the source files to include in your pattern.

"Defining the target properties" on page 1342
Identify the target properties in the workspace project so that you can define the pattern interface and pattern users can modify the pattern parameters.

"Extending a user-defined pattern" on page 1340
Extend a user-defined pattern to customize its behavior so that it is easier to use or to provide guidance to pattern users.

Transforming pattern parameters:

Calculate a pattern parameter value from the values entered for other pattern parameters.

Before you begin

Before you start:

Complete the following tasks:

- "Creating a pattern authoring project" on page 1337
- "Selecting the source files to use for a user-defined pattern" on page 1338
- "Defining the target properties" on page 1342

About this task

You can transform pattern parameters by using XPath expressions; for example, to calculate a pattern parameter value from the values entered for other pattern parameters. WebSphere Message Broker supports XPath 1.0. For XPath reference information, including information about XPath functions, see W3C XPath 1.0 Specification.

- The `pp:getValue()` function is included, in addition to the functions in the XPath 1.0 Specification. The `pp:getValue()` function takes the parameter ID of a pattern parameter and returns the value of that pattern parameter. To see the parameter ID for a pattern parameter:
  1. In the **Pattern Configuration** tab of the Pattern Authoring editor, double-click a parameter, or select a parameter and click **Edit**. The Edit Parameter window opens.
  2. Click the **Transform** tab. The parameter IDs for pattern parameters are shown in the **Parameter ID** column of the Pattern Parameters table.
- When a pattern instance is generated by a pattern user, transformation expressions are processed before any Java and PHP code that has been added to modify pattern instances. Transformation of pattern parameters is processed in the following sequence:
  1. Every parameter that has an XPath transformation expression is evaluated.
  2. The value of each parameter that has an XPath transformation expression is updated with the result of its evaluation, overwriting the value entered by the pattern user.
  3. The parameters are evaluated in top to bottom order, as they are listed in the **Pattern Configuration** tab of the Pattern Authoring editor.

Procedure

To transform a pattern parameter by using an XPath expression:

1. In the **Pattern Configuration** tab of the Pattern Authoring editor, double-click the parameter that you want to transform, or select the parameter and click **Edit**. The Edit Parameter window opens.
2. Click the **Transform** tab. Create an XPath expression for your chosen parameter:
   - To select a function:
     a. Expand **Boolean**, **Number**, **Pattern**, or **String** in the Functions section, then click a function. The function is displayed in the **Function name** field.
     b. Click **Use**. The function is inserted into the **Expression** field at the cursor.
   - To select an operator:
     a. Click an operator in the Operators section. The operator is displayed in the **Operator** field.
     b. Click **Use**. The operator is inserted into the **Expression** field at the cursor.
   - To select a pattern parameter:

a. Click a pattern parameter in the Pattern Parameters table. The parameter ID shown in the **Parameter ID** column of the Pattern Parameters table for the chosen parameter is displayed in the **Parameter ID** field.
b. Click **Use**. The parameter ID is inserted into the **Expression** field at the cursor.

- You can also edit the expression directly in the **Expression** field.

3. Repeat the actions in step 2 on page 1354, as required, to create your XPath expression.
4. You can now choose whether to test your expression:
   - To test your expression, see "Testing a transformation expression."
   - To accept the expression without testing it, click **OK**. The Edit Parameter window closes.

Related concepts:

"User-defined patterns" on page 1334
A *user-defined pattern* extends the function of WebSphere Message Broker so that you are able to create patterns that you can reuse within your organization.

"Creating a user-defined pattern" on page 1336
The workflow showing the actions required for pattern authoring.

"Modifying pattern instances by using Java or PHP" on page 1364
Use Java or PHP code to modify pattern instances when the pattern user generates an instance of a user-defined pattern. For example, you can use Java or PHP code to modify the structure of a message flow based on the values of pattern parameters.

Related tasks:

"Creating a pattern authoring project" on page 1337
Create a pattern authoring project and choose an exemplar for the project.

"Selecting the source files to use for a user-defined pattern" on page 1338
Select the source files to include in your pattern.

"Defining the target properties" on page 1342
Identify the target properties in the workspace project so that you can define the pattern interface and pattern users can modify the pattern parameters.

"Changing pattern parameter IDs" on page 1352
Change default pattern parameter IDs to make it easier to write XPath, Java, and PHP code that uses parameter IDs.

"Testing a transformation expression"
Test your XPath transformation expression created in a user-defined pattern to check that it works correctly.

"Extending a user-defined pattern" on page 1340
Extend a user-defined pattern to customize its behavior so that it is easier to use or to provide guidance to pattern users.

Related reference:

"General industry standards supported by WebSphere Message Broker" on page 3607
WebSphere Message Broker supports general industry standards that are associated with message processing.

Testing a transformation expression:

Test your XPath transformation expression created in a user-defined pattern to check that it works correctly.

Before you begin

Before you start:

Complete the following tasks:

- "Creating a pattern authoring project" on page 1337
- "Selecting the source files to use for a user-defined pattern" on page 1338
- "Defining the target properties" on page 1342
- "Transforming pattern parameters" on page 1353

Procedure

To test your XPath expression:

1. If the **Transform** tab in the Edit Parameter window is not already open, complete the following steps to open the tab:
   a. In the **Pattern Configuration** tab of the Pattern Authoring editor, double-click the parameter containing the XPath expression that you want to test, or select the parameter and click **Edit**. The Edit Parameter window opens.
   b. Click the **Transform** tab.
2. To change the test value for a parameter, select the parameter in the Pattern Parameters section and enter the required test value in the **Test value** field. Click **Set**. The new test value is shown in the Pattern Parameters section.
3. To test the expression, click **Evaluate**. The expression is evaluated by using the values in the **Test Values** column of the Pattern Parameters section and the result is shown in the **Result** field. If the expression is not valid an error message is displayed in the **Result** field.
4. Repeat steps 2 and 3, as required, to test the XPath expression. If you want to modify the expression, see "Transforming pattern parameters" on page 1353.
5. When you have finished testing the expression, click **OK**. The Edit Parameter window closes.

Related concepts:

"User-defined patterns" on page 1334
A *user-defined pattern* extends the function of WebSphere Message Broker so that you are able to create patterns that you can reuse within your organization.

Related tasks:

"Creating a pattern authoring project" on page 1337
Create a pattern authoring project and choose an exemplar for the project.

"Selecting the source files to use for a user-defined pattern" on page 1338
Select the source files to include in your pattern.

"Defining the target properties" on page 1342
Identify the target properties in the workspace project so that you can define the pattern interface and pattern users can modify the pattern parameters.

"Transforming pattern parameters" on page 1353
Calculate a pattern parameter value from the values entered for other pattern parameters.

"Extending a user-defined pattern" on page 1340
Extend a user-defined pattern to customize its behavior so that it is easier to use or to provide guidance to pattern users.

Enabling pattern parameters:

You can control whether pattern parameters can be edited by pattern users based on the values of other pattern parameters.

Before you begin

Before you start:

Complete the following tasks:

- "Creating a pattern authoring project" on page 1337
- "Selecting the source files to use for a user-defined pattern" on page 1338
- "Defining the target properties" on page 1342

About this task

By using XPath expressions you can control whether pattern users can edit pattern parameters in the Pattern Instance editor . If an expression evaluates to Boolean true, the pattern user can configure the parameter in the Configure Pattern Parameters page; if an expression evaluates to Boolean `false`, the parameter is read-only, and the pattern user cannot configure it. When the pattern user configures pattern parameters, expressions are evaluated whenever any of the pattern parameters used in an expression change.

WebSphere Message Broker supports XPath 1.0. For XPath reference information, including information about XPath functions, see W3C XPath 1.0 Specification.

The `pp:getValue()` function is included, in addition to the functions in the XPath 1.0 Specification. The `pp:getValue()` function takes the parameter ID of a pattern parameter and returns the value of that pattern parameter. To see the parameter ID for a pattern parameter:

1. In the **Pattern Configuration** tab of the Pattern Authoring editor, double-click a parameter, or select a parameter and click **Edit**. The Edit Parameter window opens.
2. Click the **Enable** tab. The parameter IDs for pattern parameters are shown in the **Parameter ID** column of the Pattern Parameters table.

Procedure

To define an XPath expression to control whether a pattern parameter can be edited:

1. In the **Pattern Configuration** tab of the Pattern Authoring editor, select the parameter that you want to configure. Click **Edit**. The Edit Parameter window opens.
2. Click the **Enable** tab. Create an XPath expression for your selected parameter by applying the following operations as required:
   - To select a function:
     a. Click a function in the Functions section. The function is displayed in the **Function name** field.
     b. Click **Use**. The function is inserted into the **Expression** field at the cursor.
   - To select an operator:
     a. Click an operator in the Operators section. The operator is displayed in the **Operator** field.

b. Click **Use**. The operator is inserted into the **Expression** field at the cursor.

- To select a pattern parameter:
  a. Click a pattern parameter in the Pattern Parameters table. The ID shown in the **Parameter ID** column of the Pattern Parameters table for the selected parameter is displayed in the **Parameter ID** field.
  b. Click **Use**. The parameter ID is inserted into the **Expression** field at the cursor.
- You can also edit the expression directly in the **Expression** field.

A common requirement is for a pattern parameter to be read-only. To ensure that a pattern parameter is read-only, set the XPath expression for the pattern parameter to `false()`.

3. You can now choose whether to test your expression:
   - To test your expression, see "Testing an enabling expression."
   - To accept the expression without testing it, click **OK**. The Edit Parameter window closes.

Related concepts:

"User-defined patterns" on page 1334
A *user-defined pattern* extends the function of WebSphere Message Broker so that you are able to create patterns that you can reuse within your organization.

"Creating a user-defined pattern" on page 1336
The workflow showing the actions required for pattern authoring.

Related tasks:

"Creating a pattern authoring project" on page 1337
Create a pattern authoring project and choose an exemplar for the project.

"Selecting the source files to use for a user-defined pattern" on page 1338
Select the source files to include in your pattern.

"Defining the target properties" on page 1342
Identify the target properties in the workspace project so that you can define the pattern interface and pattern users can modify the pattern parameters.

"Changing pattern parameter IDs" on page 1352
Change default pattern parameter IDs to make it easier to write XPath, Java, and PHP code that uses parameter IDs.

"Testing an enabling expression"
Test an XPath enabling expression that you created in a user-defined pattern to check that it works correctly.

"Extending a user-defined pattern" on page 1340
Extend a user-defined pattern to customize its behavior so that it is easier to use or to provide guidance to pattern users.

Related reference:

"General industry standards supported by WebSphere Message Broker" on page 3607
WebSphere Message Broker supports general industry standards that are associated with message processing.

Testing an enabling expression:

Test an XPath enabling expression that you created in a user-defined pattern to check that it works correctly.

Before you begin

Before you start:

Complete the following tasks:

- "Creating a pattern authoring project" on page 1337
- "Selecting the source files to use for a user-defined pattern" on page 1338
- "Defining the target properties" on page 1342
- "Enabling pattern parameters" on page 1357

Procedure

To test your XPath expression:

1. If the **Enable** tab in the Edit Parameter window is not already open, complete the following steps to open the tab:
   a. In the **Pattern Configuration** tab of the Pattern Authoring editor, select the parameter containing the XPath expression that you want to test and click **Edit**. The Edit Parameter window opens.
   b. Click the **Enable** tab.
2. To test the expression, click **Evaluate**. The expression is evaluated by using the values in the **Test Values** column of the Pattern Parameters section, and the result is shown in the **Result** field. The result shows whether the parameter is enabled or disabled, and the value of the expression is shown in parentheses. If the value is `true`, the associated parameter is enabled; if the value is `false`, the parameter is disabled. Use the following table to determine the value of your XPath expression.

Result data type	Evaluates to True	Evaluates to False
Boolean	`true`	`false`
Numeric	Any non-zero value	`0` or `0.0`
String	Any string that returns `true` with a match that is not case-sensitive	Any string that does not return `true` with a match that is not case-sensitive

3. To change the test value for a parameter, select the parameter in the Pattern Parameters section and enter the required test value in the **Test value** field. Click **Set**. The new test value is shown in the Pattern Parameters section.
4. Repeat steps 2 and 3, as required, to test the XPath expression. If you want to modify the expression, see "Enabling pattern parameters" on page 1357.
5. When you have finished testing the expression, click **OK**. The Edit Parameter window closes.

Related concepts:

"User-defined patterns" on page 1334
A *user-defined pattern* extends the function of WebSphere Message Broker so that you are able to create patterns that you can reuse within your organization.

Related tasks:

"Creating a pattern authoring project" on page 1337
Create a pattern authoring project and choose an exemplar for the project.

"Selecting the source files to use for a user-defined pattern" on page 1338
Select the source files to include in your pattern.

"Defining the target properties" on page 1342
Identify the target properties in the workspace project so that you can define the pattern interface and pattern users can modify the pattern parameters.

"Enabling pattern parameters" on page 1357
You can control whether pattern parameters can be edited by pattern users based on the values of other pattern parameters.

"Extending a user-defined pattern" on page 1340
Extend a user-defined pattern to customize its behavior so that it is easier to use or to provide guidance to pattern users.

Using enumerated values for pattern parameters:

Use enumerated types to provide predefined values for the pattern parameters in your user-defined patterns.

Before you begin

Before you start:

Complete the following tasks:

- "Creating a pattern authoring project" on page 1337
- "Selecting the source files to use for a user-defined pattern" on page 1338
- "Defining the target properties" on page 1342

About this task

You can use enumerated types to provide pattern users with a predefined list of pattern parameter values to select from. For enumerated types that you create, you can add, remove, and edit values. As part of the pattern authoring process, any target properties that use a list of values have enumerated types created for them automatically. You can remove values from this automatically generated list, but you cannot add new values or change existing values.

Procedure

- Create your own enumerated types. See "Creating enumerated types for pattern parameters" on page 1361.
- Edit enumerated types. See "Editing enumerated types for pattern parameters" on page 1362.
- Assign enumerated types to pattern parameters. See "Defining the user interface" on page 1343.

Related concepts:

"User-defined patterns" on page 1334
A *user-defined pattern* extends the function of WebSphere Message Broker so that you are able to create patterns that you can reuse within your organization.

"Creating a user-defined pattern" on page 1336
The workflow showing the actions required for pattern authoring.

Related tasks:

"Creating a pattern authoring project" on page 1337
Create a pattern authoring project and choose an exemplar for the project.

"Selecting the source files to use for a user-defined pattern" on page 1338
Select the source files to include in your pattern.

"Defining the target properties" on page 1342
Identify the target properties in the workspace project so that you can define the

pattern interface and pattern users can modify the pattern parameters.

Creating enumerated types for pattern parameters:

Define enumerated types to provide predefined values for pattern parameters in your user-defined patterns.

Before you begin

Before you start:

Complete the following tasks:
- "Creating a pattern authoring project" on page 1337
- "Selecting the source files to use for a user-defined pattern" on page 1338
- "Defining the target properties" on page 1342

About this task

Enumerated types give pattern users a predefined list of values to select from when they choose values for pattern parameters after generating a new pattern instance.

Procedure

To create a new enumerated type:
1. In the Pattern Authoring editor, click the **Pattern Configuration** tab. Click **Enumerated Types**. The Configure Enumerated Types window opens.
2. Click the **Add** button that is under the **Enumerated type** field. The Enter New Enumeration window opens.
3. Enter a name for the enumeration in the **Enter a name for the enumeration** field. Click **OK**. The new enumeration is displayed in the table of values, and a display name and default value are entered in the table.
4. For manipulating values, either adding or editing, use the following options:
   - To add a new value, click the **Add** button that is next to the table of values. A new display name and value are added to the table.
   - To edit a display name or value, double-click the display name or value in the table. Type a new entry and then press Enter to save the new display name or value.
5. When you have finished adding values and editing display names and values, click **OK**. The Configure Enumerated Types window closes.

What to do next

Next:

You can now assign your enumerated type to a pattern parameter, see "Defining the user interface" on page 1343.

Related concepts:

"User-defined patterns" on page 1334
A *user-defined pattern* extends the function of WebSphere Message Broker so that you are able to create patterns that you can reuse within your organization.

Related tasks:

"Creating a pattern authoring project" on page 1337
Create a pattern authoring project and choose an exemplar for the project.

"Selecting the source files to use for a user-defined pattern" on page 1338
Select the source files to include in your pattern.

"Defining the target properties" on page 1342
Identify the target properties in the workspace project so that you can define the pattern interface and pattern users can modify the pattern parameters.

"Extending a user-defined pattern" on page 1340
Extend a user-defined pattern to customize its behavior so that it is easier to use or to provide guidance to pattern users.

Editing enumerated types for pattern parameters:

Edit enumerated types to add, remove or rename values in existing enumerated types, or to rename an enumerated type. You can edit an existing user-generated enumerated type to add, remove, or change values and display names within the enumerated type, or to change the name of the enumerated type.

Before you begin

Before you start:

Complete the following tasks:

- "Creating a pattern authoring project" on page 1337
- "Selecting the source files to use for a user-defined pattern" on page 1338
- "Defining the target properties" on page 1342
- "Creating enumerated types for pattern parameters" on page 1361

About this task

As part of the pattern authoring process, target properties that use a list of values have enumerated types automatically created for them. You can remove values from this automatically generated list or regenerate the original list of values, but you cannot add new values or edit existing values.

You can duplicate or remove user-generated enumerated types and enumerated types created from target properties.

For enumerated types that you create, you can add, remove, and edit values.

Procedure

To edit enumerated types:

1. In the Pattern Authoring editor, click the **Pattern Configuration** tab. Click **Enumerated Types**. The Configure Enumerated Types window opens.
2. In the **Enumerated type** field, select the enumerated type to edit.
3. You can make the following changes to user-generated enumerated types:
   - Rename the enumerated type. To rename the enumerated type, click **Rename**. The Rename Enumeration window opens. Enter a name for the enumeration and click **OK**. The Rename Enumeration window closes.
   - Add new values. To add a new value, click the **Add** button that is next to the table of values. A new value and display name are added to the table.

- Remove values. To remove a value, select the value to remove and click the **Remove** button that is next to the table of values. The value is removed from the table.

  If a value is the only entry in the table, you cannot remove it.
- Change existing values and display names. To edit a value or display name, double-click the value or display name in the table. Type a new entry and then press Enter to save the new display name or value.
- Duplicate the enumerated type. To duplicate an enumerated type, click **Duplicate**. The duplicate enumerated type is displayed. The name of the duplicate enumerated type is the same as the original, but with _1 added to the end.
- Remove the enumerated type. To remove an enumerated type, click **Remove**. The enumerated type is removed.

  If an enumerated type is being used by a pattern parameter it cannot be removed. You can check if an enumerated type is being used by a parameter in the **This enumerated type is used by the following parameter** field. To modify a parameter so that it does not use an enumerated type, see "Defining the user interface" on page 1343.

4. You can make the following changes to enumerated types that are created from target properties:
   - Rename the enumerated type. To rename the enumerated type, click **Rename**. The Rename Enumeration window opens. Enter a name for the enumeration and click **OK**. The Rename Enumeration window closes.
   - Remove values. To remove a value, select the value to remove and click the **Remove** button that is next to the table of values. The value is removed from the table.

     If a value is the only entry in the table, you cannot remove it.
   - Reset the original values. To restore values you have removed from the list, click **Reset Values**. The original list is displayed.
   - Duplicate the enumerated type. To duplicate an enumerated type, click **Duplicate**. The duplicate enumerated type is displayed. The name of the duplicate enumerated type is the same as the original, but with _1 added to the end.
   - Remove the enumerated type. To remove an enumerated type, click **Remove**. The enumerated type is removed.

     If an enumerated type is being used by a pattern parameter it cannot be removed. You can see if an enumerated type is being used by a parameter in the **This enumerated type is used by the following parameter** field. To modify a parameter so that it does not use an enumerated type, see "Defining the user interface" on page 1343.
5. Repeat the tasks in steps 3 on page 1362 and 4 until your changes are complete. Click **OK**. The Configure Enumerated Types window closes.

What to do next

Next:

You can now build your pattern; see "Building pattern plug-ins" on page 1395.

Related concepts:

"User-defined patterns" on page 1334
A *user-defined pattern* extends the function of WebSphere Message Broker so that you are able to create patterns that you can reuse within your organization.

Related tasks:

"Creating a pattern authoring project" on page 1337
Create a pattern authoring project and choose an exemplar for the project.

"Selecting the source files to use for a user-defined pattern" on page 1338
Select the source files to include in your pattern.

"Defining the target properties" on page 1342
Identify the target properties in the workspace project so that you can define the pattern interface and pattern users can modify the pattern parameters.

"Extending a user-defined pattern" on page 1340
Extend a user-defined pattern to customize its behavior so that it is easier to use or to provide guidance to pattern users.

Modifying pattern instances by using Java or PHP:

Use Java or PHP code to modify pattern instances when the pattern user generates an instance of a user-defined pattern. For example, you can use Java or PHP code to modify the structure of a message flow based on the values of pattern parameters.

When you author user-defined patterns, you can include Java or PHP code to modify pattern instances. This code is run when the pattern user generates an instance of a user-defined pattern and can be used to carry out a number of actions on the pattern.

By using the Java API, you can access all the nodes, connections, and pattern parameters in a user-defined pattern. Examples of changes that you can make by using Java code are:

- Adding, removing, and copying nodes
- Changing connections between nodes
- Renaming nodes
- Changing pattern parameter values and user-defined property values
- Renaming message flows

By using the PHP API for user-defined patterns, you can access the pattern parameters and the pattern instance name of a user-defined pattern. Examples of changes that you can make by using PHP code are:

- Marking up ESQL files to change the operation of the ESQL file depending on the value of pattern parameters
- Taking values from an XML file and using them as pattern parameters
- Changing pattern parameter values
- Renaming message flows
- Running other PHP scripts or Java code

Do not use Java or PHP API code to set the `WSDL file name` property, or any of the properties set by the WSDL file, on a SOAPInput, SOAPRequest, or SOAPAsyncRequest node in your user-defined pattern. To set the `WSDL file name` property, see "Configuring SOAP nodes for user-defined patterns" on page 1351. To modify other properties on these nodes, select the properties as target properties when you create your user-defined pattern. To select target properties, see "Defining the target properties" on page 1342.

Choosing between Java and PHP

WebSphere Message Broker contains a Java to PHP bridge, therefore you can complete some pattern authoring tasks by using either Java or PHP. You can choose which language to use based on the existing skills, assets, or libraries within your organization. The WebSphere Message Broker Toolkit includes code completion and a debugger for Java, which make Java a good choice for longer or more complex code. PHP written in the WebSphere Message Broker Toolkit can be changed and tested without relaunching the workbench, which makes PHP a good choice if you want to change your code and view the results quickly. If you do not have a preference between Java and PHP after considering the skills and assets of your organization and the testing approach, use Java for changes to the structure of the message flow and use PHP to modify text files.

Plug-in packaging

The code you write to modify pattern instances is contained in a separate plug-in from the user-defined pattern. When you create a pattern archive, your code plug-ins are automatically packaged with the generated pattern plug-ins. For more information about packaging user-defined patterns, see "Packaging and distributing pattern plug-ins" on page 1397.

Sequence of actions when a pattern user generates a pattern instance

When a pattern user generates an instance of a user-defined pattern, the following actions occur in sequence:

1. The pattern instance projects are created and all non-message flow files are copied into the workspace.
2. The message flows in the pattern plug-ins are loaded into memory.
3. Pattern parameters that are transformed by using XPath expressions are evaluated. If a pattern parameter is disabled by using an XPath expression, its parameter values are not changed. All target properties are set in all message flows within the user-defined pattern based on pattern parameter values.
4. The Java and PHP code targets are run in top-to-bottom order as they are listed in the Pattern Authoring editor.
5. The message flows are saved into the pattern instance projects.

Related concepts:

"User-defined patterns" on page 1334
A *user-defined pattern* extends the function of WebSphere Message Broker so that you are able to create patterns that you can reuse within your organization.

Related tasks:

"Modifying pattern instances by using the Java API" on page 1367
Add Java code to a code plug-in project to modify a pattern instance when the pattern instance is generated by a pattern user.

"Examples of Java API code" on page 1369
Use the following examples of Java API code for common tasks to help you write your own Java code to modify pattern instances.

"Testing Java code" on page 1387
After writing Java code to modify pattern instances, test the code to check that it works correctly.

"Modifying pattern instances by using PHP" on page 1389
Add PHP code to a code plug-in project to modify a pattern instance when the pattern instance is generated by a pattern user.

"Examples of PHP API code" on page 1390
Use the following examples of PHP API code for common tasks to help you write your own PHP code to modify pattern instances.
"Testing PHP API code" on page 1394
After writing PHP code to modify pattern instances, test the code to check that it works correctly.

Creating a code plug-in project:

Create a Java and PHP plug-in project to contain pattern instance modification code.

Before you begin

Before you can complete this task, you must have completed the following tasks:

- "Creating a pattern authoring project" on page 1337
- "Selecting the source files to use for a user-defined pattern" on page 1338

About this task

You must create a code plug-in project to contain the Java and PHP pattern instance modification code that you write. You can distribute the Java and PHP code plug-in project in the pattern archive for the associated user-defined pattern to pattern users. See "Packaging and distributing pattern plug-ins" on page 1397.

Procedure

To create a Java and PHP code plug-in project:

1. Open the **Pattern Configuration** tab of the Pattern Authoring editor.
2. Click **Add**. The Add Java or PHP Code window opens.
3. Click **New Project**. The New Pattern Authoring Java and PHP Project window opens.
4. Enter a plug-in ID for the new code plug-in. After the plug-in is created, the plug-in ID is shown in the Projects section of the Broker Development view. The plug-in ID is also used to identify the plug-in when you distribute it to pattern users.
5. Optional: If you want to include PHP code in the code plug-in, select **Add PHP support to the project**. If you select this option a `main.php` template and example PHP scripts are added to the code plug-in.
6. Optional: If you want to include a Java class in the code plug-in, complete the following steps:
   a. Select **Add an example pattern authoring Java class to the project**.
   b. Enter a package name for the Java class in the **Package name** field.
   c. Enter a class name for the Java class in the **Class name** field.
7. Click **Finish**. The New Pattern Authoring Java and PHP Project window closes.
8. You can now add Java or PHP code to a user-defined pattern:
   - To add Java code to a user-defined pattern, see "Modifying pattern instances by using the Java API" on page 1367.
   - To add PHP code to a user-defined pattern, see "Modifying pattern instances by using PHP" on page 1389.
   - To return to pattern authoring without adding code, in the Add Java or PHP Code window, click **Cancel**.

Related concepts:

"User-defined patterns" on page 1334
A *user-defined pattern* extends the function of WebSphere Message Broker so that you are able to create patterns that you can reuse within your organization.

Related tasks:

"Creating a pattern authoring project" on page 1337
Create a pattern authoring project and choose an exemplar for the project.

"Selecting the source files to use for a user-defined pattern" on page 1338
Select the source files to include in your pattern.

"Defining the target properties" on page 1342
Identify the target properties in the workspace project so that you can define the pattern interface and pattern users can modify the pattern parameters.

"Extending a user-defined pattern" on page 1340
Extend a user-defined pattern to customize its behavior so that it is easier to use or to provide guidance to pattern users.

"Modifying pattern instances by using the Java API"
Add Java code to a code plug-in project to modify a pattern instance when the pattern instance is generated by a pattern user.

"Modifying pattern instances by using PHP" on page 1389
Add PHP code to a code plug-in project to modify a pattern instance when the pattern instance is generated by a pattern user.

Modifying pattern instances by using the Java API:

Add Java code to a code plug-in project to modify a pattern instance when the pattern instance is generated by a pattern user.

Before you begin

Before you can complete this task, you must have completed the following tasks:

- "Creating a pattern authoring project" on page 1337
- "Selecting the source files to use for a user-defined pattern" on page 1338
- "Creating a code plug-in project" on page 1366

About this task

You can add Java code to a code plug-in project so that it runs when a pattern instance is generated. To add the Java code, select the Java class that you want to run.

The Java class you select can be created in the following ways:

- If you added a Java class when you created the code plug-in, then a template Java class is created and added to the project. This class is based on the Pattern Authoring Class template and contains the basic structure required to function correctly.
- When you follow the steps in this topic to select a Java class, you can optionally create a new Java class. A class created in this way is based on the Pattern Authoring Class template and contains the basic structure required to function correctly.
- You can write your own Java class. This class must implement the `GeneratePatternInstanceTransform` interface.

Regardless of how the class is created, you must add your own code within the class to complete the steps you require when the pattern instance is generated. For examples of Java API code for completing common tasks, see "Examples of Java API code" on page 1369. For reference information about the Java API, see "Java API for user-defined patterns" on page 5345.

Procedure

To select a Java class to run when a pattern instance is generated:

1. If the Add Java or PHP Code window is not open, in the **Pattern Configuration** tab of the Pattern Authoring editor, click **Add**. The Add Java or PHP Code window opens.
2. In the **Type of code** list, select Java.
3. In the **Class name** list, select the name of the class that you want to run when a pattern instance is generated. A class is shown in the **Class name** list if it is in the current workspace and if it implements the `GeneratePatternInstanceTransform` interface.
4. Optional: You can create a new Java class:
   a. Click **New Java Class**. The New Pattern Authoring Java Class window opens.
   b. In the **Source folder** field, click **Browse** and select the folder in which to store the new Java class file.
   c. Optional: In the **Package** field, enter the name of the Java package for the new class. If you leave this field blank, the default package is used.
   d. Optional: Select **Enclosing type** and enter an enclosing type in the **Enclosing type** field.
   e. Enter a name for the new Java class in the **Name** field.
   f. Optional: Change the Superclass for the new Java class in the **Superclass** field.
   g. Optional: To add an interface for the new Java class, click **Add**. The Implemented Interfaces Selection window opens. In the **Choose interfaces** field, enter the name of the interface that you want to add, select the interface in the **Matching items** list and click **OK**. The Implemented Interfaces Selection window closes.
   h. Optional: To remove an interface for the Java class, select the interface in the **Interfaces** list and click **Remove**.
   i. Optional: Click **Next** to view information about the Pattern Authoring Class template, which is used to create the Java class.
   j. Click **Finish**.
5. Click **OK**. The Add Java or PHP Code window closes and the Java class is shown in the Java and PHP Code section of the **Pattern Configuration** tab.
6. Optional: The Java and PHP code listed in the Java and PHP Code section of the **Pattern Configuration** tab runs from top to bottom when a pattern instance is generated. To change the order in which the code is run, select the entry you want to move in the **Java and PHP Code** section and click the **Up** or **Down** button to change the position of the entry in the list.

What to do next

In the Java class that you selected to run when the pattern instance is generated, you must now write the code to perform the steps you require. For examples of Java API code for completing common tasks, see "Examples of Java API code" on page 1369

page 1367 For reference information about the Java API, see "Java API for user-defined patterns" on page 5345. For information about the Java development views and editors within the WebSphere Message Broker Toolkit, see Java Development User Guide plug-in - Views and Editors.

Related concepts:

"User-defined patterns" on page 1334
A *user-defined pattern* extends the function of WebSphere Message Broker so that you are able to create patterns that you can reuse within your organization.

Related tasks:

"Creating a pattern authoring project" on page 1337
Create a pattern authoring project and choose an exemplar for the project.

"Selecting the source files to use for a user-defined pattern" on page 1338
Select the source files to include in your pattern.

"Defining the target properties" on page 1342
Identify the target properties in the workspace project so that you can define the pattern interface and pattern users can modify the pattern parameters.

"Extending a user-defined pattern" on page 1340
Extend a user-defined pattern to customize its behavior so that it is easier to use or to provide guidance to pattern users.

"Modifying pattern instances by using the Java API" on page 1367
Add Java code to a code plug-in project to modify a pattern instance when the pattern instance is generated by a pattern user.

"Testing Java code" on page 1387
After writing Java code to modify pattern instances, test the code to check that it works correctly.

"Examples of Java API code"
Use the following examples of Java API code for common tasks to help you write your own Java code to modify pattern instances.

Examples of Java API code:

Use the following examples of Java API code for common tasks to help you write your own Java code to modify pattern instances.

About this task

The following list shows examples of Java code that you can use to complete common tasks when you are creating user-defined patterns:

Procedure

- "Loading an existing message flow into memory" on page 1370
- "Renaming a node" on page 1371
- "Adding a node and a subflow node" on page 1372
- "Setting the position of a node" on page 1373
- "Copying a node" on page 1374
- "Removing a node" on page 1375
- "Adding connections between nodes" on page 1376
- "Adding and connecting user-defined nodes" on page 1378
- "Removing connections between nodes" on page 1380
- "Changing pattern parameter values" on page 1381
- "Creating or changing user-defined properties" on page 1381

- "Creating ESQL modules" on page 1382
- "Renaming a message flow" on page 1384
- "Running PHP code using the Java API" on page 1385
- "Updating filter tables on Route nodes" on page 1386

What to do next

Next:

After creating your Java code, test it to check that it works correctly; see "Testing Java code" on page 1387.

Related concepts:

"User-defined patterns" on page 1334
A *user-defined pattern* extends the function of WebSphere Message Broker so that you are able to create patterns that you can reuse within your organization.

Related tasks:

"Creating a pattern authoring project" on page 1337
Create a pattern authoring project and choose an exemplar for the project.

"Extending a user-defined pattern" on page 1340
Extend a user-defined pattern to customize its behavior so that it is easier to use or to provide guidance to pattern users.

"Modifying pattern instances by using the Java API" on page 1367
Add Java code to a code plug-in project to modify a pattern instance when the pattern instance is generated by a pattern user.

"Testing Java code" on page 1387
After writing Java code to modify pattern instances, test the code to check that it works correctly.

Related reference:

"Java API for user-defined patterns" on page 5345
Use the Java API to write Java code to modify user-defined pattern instances.

Loading an existing message flow into memory:

Use the Java API to modify a pattern instance to load a message flow into memory and make it available to other Java API methods.

About this task

You must load a message flow into memory to use it with Java API methods within your Java code. To load a message flow into memory, use the getMessageFlow() method of the PatternInstanceManager object, which is automatically passed to your Java code. The getMessageFlow() method takes the message flow project containing the required message flow file and the relative path to the message flow file from this project.

Procedure

- For example, to load a message flow that is in the default schema:

```
public class MyJava implements GeneratePatternInstanceTransform {
    public void onGeneratePatternInstance(PatternInstanceManager patternInstanceManager) {
        MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");
        if (mf1 != null) {
           // Message flow was found
        }
        else {
```

```
            // Message flow was not found
        }
    }
}
```

- The following example shows how to load a message flow that is in the mqsi schema:

```
public class MyJava implements GeneratePatternInstanceTransform {
    public void onGeneratePatternInstance(PatternInstanceManager patternInstanceManager) {
        MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "mqsi/main.msg
        if (mf1 != null) {
            // Message flow was found
        }
        else {
            // Message flow was not found
        }
    }
}
```

Results

Result:

You can now refer to the instance of the MessageFlow object in your Java code.

Related concepts:

"Modifying pattern instances by using Java or PHP" on page 1364
Use Java or PHP code to modify pattern instances when the pattern user generates an instance of a user-defined pattern. For example, you can use Java or PHP code to modify the structure of a message flow based on the values of pattern parameters.

Related tasks:

"Modifying pattern instances by using the Java API" on page 1367
Add Java code to a code plug-in project to modify a pattern instance when the pattern instance is generated by a pattern user.

"Renaming a message flow" on page 1384
Use the Java API to modify a pattern instance to rename message flows.

"Testing Java code" on page 1387
After writing Java code to modify pattern instances, test the code to check that it works correctly.

Related reference:

"Java API for user-defined patterns" on page 5345
Use the Java API to write Java code to modify user-defined pattern instances.

Renaming a node:

Use Java API code to modify a pattern instance to rename a node in a message flow.

About this task

To rename a node, you must first access the required node. You can access the node by using the existing name of the node and the getNodeByName() method. You can then rename the node by using the setNodeName() method, which takes the new node name as a parameter. For example:

Example

```
MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");
Node mqinNode = mf1.getNodeByName("My Input Node");
mqinNode.setNodeName("New Input Node");
```

Related concepts:

"Modifying pattern instances by using Java or PHP" on page 1364
Use Java or PHP code to modify pattern instances when the pattern user generates an instance of a user-defined pattern. For example, you can use Java or PHP code to modify the structure of a message flow based on the values of pattern parameters.

Related tasks:

"Modifying pattern instances by using the Java API" on page 1367
Add Java code to a code plug-in project to modify a pattern instance when the pattern instance is generated by a pattern user.

"Adding a node and a subflow node"
Use the Java API to modify a pattern instance to add a new node or subflow node to a message flow.

"Renaming a message flow" on page 1384
Use the Java API to modify a pattern instance to rename message flows.

"Testing Java code" on page 1387
After writing Java code to modify pattern instances, test the code to check that it works correctly.

Related reference:

"Java API for user-defined patterns" on page 5345
Use the Java API to write Java code to modify user-defined pattern instances.

Adding a node and a subflow node:

Use the Java API to modify a pattern instance to add a new node or subflow node to a message flow.

About this task

You can add a new built-in node, or a new subflow node to a message flow.

- When you add a subflow node by using the Java API, you must link the subflow node with the subflow message flow by using the setSubFlow() method of the subflow node object. For example, if you have assigned your subflow message flow to message flow instance *sub1* and you have assigned your subflow node to subflow node instance *sfNode*, you must use the following statement to link the subflow node with the subflow message flow:

  ```
  sfNode.setSubFlow(sub1);
  ```

Procedure

- The following example shows you how to add a new built-in node:

  ```
  MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");
  MQInputNode mqinNode = new MQInputNode();
  mqinNode.setNodeName("My Input Node");
  mqinNode.setQueueName("INPUTQ");
  mf1.addNode(mqinNode);
  ```

- The following example shows you how to add a new subflow node to a message flow:

  1. A new subflow node is created and assigned to object *sfNode*.
  2. The subflow node name is set to *My Sub Flow Node*.

3. The subflow node is linked to the subflow message flow by using the setSubFlow() method.
4. The new subflow node is added to the message flow held in object *mf1*.

```
MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");
MessageFlow sub1 = patternInstanceManager.getMessageFlow("MyFlowProject", "subflow.msgflow");
SubFlowNode sfNode = new SubFlowNode();
sfNode.setNodeName("My Sub Flow Node");
sfNode.setSubFlow(sub1);
mf1.addNode(sfNode);
```

Related concepts:

"Modifying pattern instances by using Java or PHP" on page 1364
Use Java or PHP code to modify pattern instances when the pattern user generates an instance of a user-defined pattern. For example, you can use Java or PHP code to modify the structure of a message flow based on the values of pattern parameters.

Related tasks:

"Modifying pattern instances by using the Java API" on page 1367
Add Java code to a code plug-in project to modify a pattern instance when the pattern instance is generated by a pattern user.

"Renaming a node" on page 1371
Use Java API code to modify a pattern instance to rename a node in a message flow.

"Setting the position of a node"
Use the Java API to modify a pattern instance to set the position of a node on the canvas in the Broker Development view.

"Copying a node" on page 1374
Use the Java API to modify a pattern instance to copy a node or subflow node to a message flow.

"Removing a node" on page 1375
Use the Java API to modify a pattern instance to remove a node from a message flow.

"Adding connections between nodes" on page 1376
Use the Java API to modify a pattern instance to add connections between nodes.

"Adding and connecting user-defined nodes" on page 1378
Use the Java API to modify a pattern instance to add user-defined nodes and to connect user-defined nodes to other nodes.

"Removing connections between nodes" on page 1380
Use the Java API to modify a pattern instance to remove connections between nodes.

"Testing Java code" on page 1387
After writing Java code to modify pattern instances, test the code to check that it works correctly.

Related reference:

"Java API for user-defined patterns" on page 5345
Use the Java API to write Java code to modify user-defined pattern instances.

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Setting the position of a node:

Use the Java API to modify a pattern instance to set the position of a node on the canvas in the Broker Development view.

About this task

You can set the position of a node on the canvas by using the setLocation() method of the node object. The following example sets the position of a new MQOutput node to coordinates x=300 pixels, y=100 pixels:

Example

```
MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");
MQOutputNode mqoutNode = new MQOutputNode();
mqoutNode.setLocation(300, 100);
```

Related concepts:

"Modifying pattern instances by using Java or PHP" on page 1364
Use Java or PHP code to modify pattern instances when the pattern user generates an instance of a user-defined pattern. For example, you can use Java or PHP code to modify the structure of a message flow based on the values of pattern parameters.

Related tasks:

"Modifying pattern instances by using the Java API" on page 1367
Add Java code to a code plug-in project to modify a pattern instance when the pattern instance is generated by a pattern user.

"Adding a node and a subflow node" on page 1372
Use the Java API to modify a pattern instance to add a new node or subflow node to a message flow.

"Copying a node"
Use the Java API to modify a pattern instance to copy a node or subflow node to a message flow.

"Removing a node" on page 1375
Use the Java API to modify a pattern instance to remove a node from a message flow.

"Renaming a message flow" on page 1384
Use the Java API to modify a pattern instance to rename message flows.

"Testing Java code" on page 1387
After writing Java code to modify pattern instances, test the code to check that it works correctly.

Related reference:

"Java API for user-defined patterns" on page 5345
Use the Java API to write Java code to modify user-defined pattern instances.

Copying a node:

Use the Java API to modify a pattern instance to copy a node or subflow node to a message flow.

About this task

You can copy a built-in or subflow node by using the clone() method. In the following example, a new MQInput node *mqinNode* is created and properties on the node are set. A new MQInput node *mqinNode1* is then created by copying *mqinNode* by using the clone() method. When the node is copied, the node properties are also copied:

Example

```
MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");
MQInputNode mqinNode = new MQInputNode();
mqinNode.setNodeName("My Input Node");
mqinNode.setQueueName("INPUTQ");
MQInputNode mqinNode1 = (MQInputNode) mqinNode.clone();
mqinNode1.setNodeName("Copy of My Input Node");
mf1.addNode(mqinNode1);
```

Related concepts:

"Modifying pattern instances by using Java or PHP" on page 1364
Use Java or PHP code to modify pattern instances when the pattern user generates an instance of a user-defined pattern. For example, you can use Java or PHP code to modify the structure of a message flow based on the values of pattern parameters.

Related tasks:

"Modifying pattern instances by using the Java API" on page 1367
Add Java code to a code plug-in project to modify a pattern instance when the pattern instance is generated by a pattern user.

"Renaming a node" on page 1371
Use Java API code to modify a pattern instance to rename a node in a message flow.

"Adding a node and a subflow node" on page 1372
Use the Java API to modify a pattern instance to add a new node or subflow node to a message flow.

"Setting the position of a node" on page 1373
Use the Java API to modify a pattern instance to set the position of a node on the canvas in the Broker Development view.

"Removing a node"
Use the Java API to modify a pattern instance to remove a node from a message flow.

"Adding connections between nodes" on page 1376
Use the Java API to modify a pattern instance to add connections between nodes.

"Adding and connecting user-defined nodes" on page 1378
Use the Java API to modify a pattern instance to add user-defined nodes and to connect user-defined nodes to other nodes.

"Removing connections between nodes" on page 1380
Use the Java API to modify a pattern instance to remove connections between nodes.

"Testing Java code" on page 1387
After writing Java code to modify pattern instances, test the code to check that it works correctly.

Related reference:

"Java API for user-defined patterns" on page 5345
Use the Java API to write Java code to modify user-defined pattern instances.

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Removing a node:

Use the Java API to modify a pattern instance to remove a node from a message flow.

About this task

To remove a node you must first get the required node from the message flow object. In the following example, the getNodeByName() method is used to get the required node from message flow object *mf1*. The node is then removed by using the removeNode() method. When a node is removed, any connections to or from the node are also removed:

Example

```
MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");
Node mqinNode = mf1.getNodeByName("My Input Node");
mf1.removeNode(mqinNode);
```

Related concepts:

"Modifying pattern instances by using Java or PHP" on page 1364
Use Java or PHP code to modify pattern instances when the pattern user generates an instance of a user-defined pattern. For example, you can use Java or PHP code to modify the structure of a message flow based on the values of pattern parameters.

Related tasks:

"Modifying pattern instances by using the Java API" on page 1367
Add Java code to a code plug-in project to modify a pattern instance when the pattern instance is generated by a pattern user.

"Adding a node and a subflow node" on page 1372
Use the Java API to modify a pattern instance to add a new node or subflow node to a message flow.

"Removing connections between nodes" on page 1380
Use the Java API to modify a pattern instance to remove connections between nodes.

"Testing Java code" on page 1387
After writing Java code to modify pattern instances, test the code to check that it works correctly.

Related reference:

"Java API for user-defined patterns" on page 5345
Use the Java API to write Java code to modify user-defined pattern instances.

Adding connections between nodes:

Use the Java API to modify a pattern instance to add connections between nodes.

About this task

You can connect both subflow and built-in nodes using the Java API. The first example shows you how to connect two built-in nodes. The second example shows you how to connect a built-in node to a subflow node.

To use the terminals of a subflow node, you must link the subflow node with the subflow message flow by using the setSubFlow() method of the subflow node object. For example, if you have assigned your subflow message flow to message flow instance *sub1* and you have assigned your subflow node to subflow node instance *sfNode*, you must use the following statement to link the subflow node with the subflow message flow:

```
sfNode.setSubFlow(sub1);
```

Procedure

- The following example shows you how to connect two built-in nodes:
  1. A MQInput node and a Collector node are created.
  2. The getInputTerminal() method is used to create a dynamic Input terminal called *NEWIN* on the Collector node.
  3. The Input terminal is connected to the Output terminal of the MQInput node by using the connect() method of the message flow object *mf1*.

```
MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");
MQInputNode mqinNode = new MQInputNode();
mqinNode.setNodeName("My Input Node");
mqinNode.setQueueName("INPUTQ");
CollectorNode colNode = new CollectorNode();
colNode.getInputTerminal("NEWIN");
mf1.connect(mqinNode.OUTPUT_TERMINAL_OUT, colNode.getInputTerminal("NEWIN"));
```

  To use a static terminal on a node you must use the appropriate constant defined for it in the API. These constants are listed in the Java API reference, see "Java API for user-defined patterns" on page 5345.

- The following example shows you how to connect a subflow node to a built-in node. You must load the subflow message flow and link it to a subflow node. You must use the getInputTerminal(), getInputTerminals(), getOutputTerminal() or getOutputTerminals() method to access the terminal on the subflow node to which you want to connect. The example code completes the following steps:
  1. The main message flow, `main.msgflow`, and a Compute node in the main message flow are loaded into memory.
  2. The subflow message flow, `subflow.msgflow`, and the subflow node in the main message flow are loaded into memory.
  3. The setSubFlow() method of the subflow node is used to link the subflow message flow *sub1* to the subflow node *sfNode*.
  4. The getOutputTerminal() method is used to get the Process terminal of the subflow node. The connect() method of the message flow object is used to connect this terminal to the Input terminal of the Compute node.

```
MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");
ComputeNode compNode = (ComputeNode)mf1.getNodeByName("My Compute Node");
MessageFlow sub1 = patternInstanceManager.getMessageFlow("MyFlowProject", "subflow.msgflow");
SubFlowNode sfNode = (SubFlowNode)mf1.getNodeByName("My Subflow Node");
sfNode.setSubFlow(sub1);
mf1.connect(sfNode.getOutputTerminal("Process"), compNode.INPUT_TERMINAL_IN);
```

Related concepts:

"Modifying pattern instances by using Java or PHP" on page 1364
Use Java or PHP code to modify pattern instances when the pattern user generates an instance of a user-defined pattern. For example, you can use Java or PHP code to modify the structure of a message flow based on the values of pattern parameters.

Related tasks:

"Modifying pattern instances by using the Java API" on page 1367
Add Java code to a code plug-in project to modify a pattern instance when the pattern instance is generated by a pattern user.

"Adding a node and a subflow node" on page 1372
Use the Java API to modify a pattern instance to add a new node or subflow node to a message flow.

"Setting the position of a node" on page 1373
Use the Java API to modify a pattern instance to set the position of a node on the canvas in the Broker Development view.

"Adding and connecting user-defined nodes"
Use the Java API to modify a pattern instance to add user-defined nodes and to connect user-defined nodes to other nodes.

"Removing connections between nodes" on page 1380
Use the Java API to modify a pattern instance to remove connections between nodes.

"Renaming a message flow" on page 1384
Use the Java API to modify a pattern instance to rename message flows.

"Running PHP code using the Java API" on page 1385
Use the Java API to modify a pattern instance to run PHP scripts from within your Java code.

"Testing Java code" on page 1387
After writing Java code to modify pattern instances, test the code to check that it works correctly.

Related reference:

"Java API for user-defined patterns" on page 5345
Use the Java API to write Java code to modify user-defined pattern instances.

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Adding and connecting user-defined nodes:

Use the Java API to modify a pattern instance to add user-defined nodes and to connect user-defined nodes to other nodes.

About this task

The code required to add and connect user-defined node is different to the code required for built-in nodes and subflow nodes.

When you write Java code for user-defined nodes you must be aware of the following information:

- User-defined nodes are supported by instances of the GenericNode class. To add user-defined nodes to message flows, create instances of GenericNode and add them to the message flow instance.
- To retrieve existing instances of a user-defined node, call getNodeByName() and cast the returned object to a GenericNode object.
- The terminals defined on your user-defined nodes are not automatically available in the Java API. If you create an instance of a GenericNode class, it does not have any input or output terminals listed. The methods getInputTerminals() and getOutputTerminals() return empty lists.
- To get an input terminal for a GenericNode, call getInputTerminal() and pass the terminal name that exists on the generic node. This method returns the input terminal and makes it available in the message flow object that contains your generic node. After you have used getInputTerminal() with a known terminal name, this input terminal is returned if getInputTerminals() is used.
- To get an output terminal for a GenericNode, call getOutputTerminal() and pass the terminal name that exists on the generic node. This method returns the output terminal and makes it available in the message flow object that contains your generic node. After you have used getOutputTerminal() with a known terminal name, this output terminal is returned if getOutputTerminals() is used.

Example

The following example shows how you can add a user-defined node to a message flow and connect it to a built-in node:

1. An MQInput node is created and added to the message flow.
2. A user-defined node is created by using the GenericNode class and is added to the message flow object.
3. The static output terminal of the MQInput is assigned to the variable *outputTerminal*.
4. The input terminal of the user-defined node is assigned to the variable *inputTerminal* by using the getInputTerminal() method with the known terminal name *In*.
5. The nodes are connected by using the connect() method.
6. The final section of code shows that the input node is now available for use in the message flow, by using the getInputTerminals() method of the user-defined node instance.

```
MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");

MQInputNode mqinNode = new MQInputNode();
mqinNode.setNodeName("My Input Node");
mqinNode.setQueueName("IN");
mf1.addNode(mqinNode);

GenericNode myNode = new GenericNode("MyUserDefinedNode");
myNode.setNodeName("MyNode");
mf1.addNode(myNode);

OutputTerminal outputTerminal = mqinNode.OUTPUT_TERMINAL_OUT;
InputTerminal inputTerminal = myNode.getInputTerminal("In");
mf1.connect(outputTerminal, inputTerminal);

InputTerminal[] inputTerminals = myNode.getInputTerminals();
System.out.println("Input terminals on my node:");
for (int i = 0; i < inputTerminals.length; i++) {
   InputTerminal inputTerminal = inputTerminals[i];
   System.out.println(inputTerminal.getName());
}
```

Related concepts:

"Modifying pattern instances by using Java or PHP" on page 1364
Use Java or PHP code to modify pattern instances when the pattern user generates an instance of a user-defined pattern. For example, you can use Java or PHP code to modify the structure of a message flow based on the values of pattern parameters.

Related tasks:

"Modifying pattern instances by using the Java API" on page 1367
Add Java code to a code plug-in project to modify a pattern instance when the pattern instance is generated by a pattern user.

"Renaming a node" on page 1371
Use Java API code to modify a pattern instance to rename a node in a message flow.

"Adding a node and a subflow node" on page 1372
Use the Java API to modify a pattern instance to add a new node or subflow node to a message flow.

"Setting the position of a node" on page 1373
Use the Java API to modify a pattern instance to set the position of a node on the canvas in the Broker Development view.

"Copying a node" on page 1374
Use the Java API to modify a pattern instance to copy a node or subflow node to a message flow.

"Removing a node" on page 1375
Use the Java API to modify a pattern instance to remove a node from a message flow.

"Adding connections between nodes" on page 1376
Use the Java API to modify a pattern instance to add connections between nodes.

"Removing connections between nodes"
Use the Java API to modify a pattern instance to remove connections between nodes.

"Creating or changing user-defined properties" on page 1381
Use the Java API to modify a pattern instance to create or change user-defined properties (UDPs).

"Testing Java code" on page 1387
After writing Java code to modify pattern instances, test the code to check that it works correctly.

Related reference:

"Java API for user-defined patterns" on page 5345
Use the Java API to write Java code to modify user-defined pattern instances.

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Removing connections between nodes:

Use the Java API to modify a pattern instance to remove connections between nodes.

About this task

You can disconnect two nodes by using the disconnect() method of the message flow object. You must provide this method with the names of the terminal instances that you want to disconnect. For example:

Example

```
MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");
MQInputNode mqinNode = (MQInputNode)mf1.getNodeByName("My Input Node");
MQOutputNode mqoutNode = (MQOutputNode)mf1.getNodeByName("My Output Node");
mf1.disconnect(mqinNode.OUTPUT_TERMINAL_OUT, mqoutNode.INPUT_TERMINAL_IN);
```

Related concepts:

"Modifying pattern instances by using Java or PHP" on page 1364
Use Java or PHP code to modify pattern instances when the pattern user generates an instance of a user-defined pattern. For example, you can use Java or PHP code to modify the structure of a message flow based on the values of pattern parameters.

Related tasks:

"Modifying pattern instances by using the Java API" on page 1367
Add Java code to a code plug-in project to modify a pattern instance when the pattern instance is generated by a pattern user.

"Adding connections between nodes" on page 1376
Use the Java API to modify a pattern instance to add connections between nodes.

"Adding and connecting user-defined nodes" on page 1378
Use the Java API to modify a pattern instance to add user-defined nodes and to connect user-defined nodes to other nodes.

"Testing Java code" on page 1387
After writing Java code to modify pattern instances, test the code to check that it works correctly.

Related reference:

"Java API for user-defined patterns" on page 5345
Use the Java API to write Java code to modify user-defined pattern instances.

Changing pattern parameter values:

Use the Java API to modify a pattern instance to change pattern parameter values.

About this task

You can change a pattern parameter value by using the PatternInstanceManager object.

Example

The following example sets the value of the pattern parameter with the pattern parameter ID `pp1` to `true`:

```
patternInstanceManager.setParameterValue("pp1", "true");
```

Related concepts:

"Modifying pattern instances by using Java or PHP" on page 1364
Use Java or PHP code to modify pattern instances when the pattern user generates an instance of a user-defined pattern. For example, you can use Java or PHP code to modify the structure of a message flow based on the values of pattern parameters.

Related tasks:

"Modifying pattern instances by using the Java API" on page 1367
Add Java code to a code plug-in project to modify a pattern instance when the pattern instance is generated by a pattern user.

"Creating or changing user-defined properties"
Use the Java API to modify a pattern instance to create or change user-defined properties (UDPs).

"Testing Java code" on page 1387
After writing Java code to modify pattern instances, test the code to check that it works correctly.

Related reference:

"Java API for user-defined patterns" on page 5345
Use the Java API to write Java code to modify user-defined pattern instances.

Creating or changing user-defined properties:

Use the Java API to modify a pattern instance to create or change user-defined properties (UDPs).

About this task

You can create new UDPs and add them to the message flow, or you can discover existing UDPs and modify them.

Procedure

- The following example shows you how to create a UDP and add it to a message flow:
  1. A UDP called `Property1` is created in parameter group `Group1`. The data type of the UDP is defined as a string and the UDP is given the default value `Hello World!`
  2. The UDP is then added to the message flow by using the addFlowProperty() method.

```
MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");
UserDefinedProperty udp = new UserDefinedProperty("Group1", "Property1", UserDefinedProperty.Usage
mf1.addFlowProperty(udp);
```

- In the following example, the existing UDPs in a message flow are discovered by using the getFlowProperties() method on the message flow. The setName() method is then used to set the name of the first UDP to `Property3`:

```
MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");
Vector<FlowProperty> flowProperties = mf1.getFlowProperties();
flowProperties.get(0).setName("Property3");
```

Related concepts:

"Modifying pattern instances by using Java or PHP" on page 1364
Use Java or PHP code to modify pattern instances when the pattern user generates an instance of a user-defined pattern. For example, you can use Java or PHP code to modify the structure of a message flow based on the values of pattern parameters.

Related tasks:

"Modifying pattern instances by using the Java API" on page 1367
Add Java code to a code plug-in project to modify a pattern instance when the pattern instance is generated by a pattern user.

"Copying a node" on page 1374
Use the Java API to modify a pattern instance to copy a node or subflow node to a message flow.

"Removing a node" on page 1375
Use the Java API to modify a pattern instance to remove a node from a message flow.

"Changing pattern parameter values" on page 1381
Use the Java API to modify a pattern instance to change pattern parameter values.

"Testing Java code" on page 1387
After writing Java code to modify pattern instances, test the code to check that it works correctly.

Related reference:

"Java API for user-defined patterns" on page 5345
Use the Java API to write Java code to modify user-defined pattern instances.

Creating ESQL modules:

Use the Java API to modify a pattern instance to create ESQL modules. You can then associate ESQL modules with nodes that use ESQL. For example, if you create a Compute node, you can use the Java API to create an ESQL module and then associate that module with the node.

About this task

Examples of nodes that use ESQL are Compute, Database, DatabaseInput, and Filter nodes. If you are using a non-default broker schema, you must set the

schema by using the setBrokerSchema() method.

Procedure

- The following example shows you how to create an ESQL module in the mqsi schema. The module is then assigned to a new Compute node by using the setComputeExpression() method:

```
MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");
ESQLModule module = new ESQLModule();
module.setBrokerSchema("mqsi");
module.setEsqlMain("MyESQLMain");
ComputeNode compNode = new ComputeNode();
compNode.setNodeName("My Compute Node");
compNode.setComputeExpression(module);
mf1.addNode(compNode);
```

- The following example shows you how to create an ESQL module in the default schema. The setBrokerSchema() method is not required.

```
MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");
ESQLModule module = new ESQLModule();
module.setEsqlMain("MyESQLMain");
ComputeNode compNode = new ComputeNode();
compNode.setNodeName("My Compute Node");
compNode.setComputeExpression(module);
mf1.addNode(compNode);
```

- The following example shows you how to discover an ESQL module from within an ESQL file by using the getEsqlModules() method. You can then use the ESQL module to set the compute expression on a Compute node by using the setComputeExpression() method.

```
MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");
File esql = new File("FileBatchProcessingSample_Branch.esql");
ESQLFile esqlFile = new ESQLFile(esql);
Vector<ESQLModule> esqlModules = esqlFile.getEsqlModules();
ComputeNode compNode = new ComputeNode();
compNode.setNodeName("My Compute Node");
compNode.setComputeExpression(esqlModules.get(0));
mf1.addNode(compNode);
```

Related concepts:

"Modifying pattern instances by using Java or PHP" on page 1364
Use Java or PHP code to modify pattern instances when the pattern user generates an instance of a user-defined pattern. For example, you can use Java or PHP code to modify the structure of a message flow based on the values of pattern parameters.

Related tasks:

"Modifying pattern instances by using the Java API" on page 1367
Add Java code to a code plug-in project to modify a pattern instance when the pattern instance is generated by a pattern user.

"Loading an existing message flow into memory" on page 1370
Use the Java API to modify a pattern instance to load a message flow into memory and make it available to other Java API methods.

"Renaming a node" on page 1371
Use Java API code to modify a pattern instance to rename a node in a message flow.

"Adding a node and a subflow node" on page 1372
Use the Java API to modify a pattern instance to add a new node or subflow node to a message flow.

"Setting the position of a node" on page 1373
Use the Java API to modify a pattern instance to set the position of a node on the

canvas in the Broker Development view.

"Copying a node" on page 1374
Use the Java API to modify a pattern instance to copy a node or subflow node to a message flow.

"Removing a node" on page 1375
Use the Java API to modify a pattern instance to remove a node from a message flow.

"Adding connections between nodes" on page 1376
Use the Java API to modify a pattern instance to add connections between nodes.

"Adding and connecting user-defined nodes" on page 1378
Use the Java API to modify a pattern instance to add user-defined nodes and to connect user-defined nodes to other nodes.

"Removing connections between nodes" on page 1380
Use the Java API to modify a pattern instance to remove connections between nodes.

"Changing pattern parameter values" on page 1381
Use the Java API to modify a pattern instance to change pattern parameter values.

"Creating or changing user-defined properties" on page 1381
Use the Java API to modify a pattern instance to create or change user-defined properties (UDPs).

"Renaming a message flow"
Use the Java API to modify a pattern instance to rename message flows.

"Running PHP code using the Java API" on page 1385
Use the Java API to modify a pattern instance to run PHP scripts from within your Java code.

"Updating filter tables on Route nodes" on page 1386
Use the Java API to modify a pattern instance to update filter tables on Route nodes.

"Testing Java code" on page 1387
After writing Java code to modify pattern instances, test the code to check that it works correctly.

Related reference:

"Java API for user-defined patterns" on page 5345
Use the Java API to write Java code to modify user-defined pattern instances.

Renaming a message flow:

Use the Java API to modify a pattern instance to rename message flows.

About this task

You can write code to rename a message flow by using the setName() method. In the following example, a message flow file called `main.msgflow` is renamed to `mainGenerated.msgflow`:

Example

```
MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");
mf1.setName(mf1.getName()+"Generated");
```

Related concepts:

"Modifying pattern instances by using Java or PHP" on page 1364
Use Java or PHP code to modify pattern instances when the pattern user generates an instance of a user-defined pattern. For example, you can use Java or PHP code to modify the structure of a message flow based on the values of pattern parameters.

Related tasks:

"Modifying pattern instances by using the Java API" on page 1367
Add Java code to a code plug-in project to modify a pattern instance when the pattern instance is generated by a pattern user.

"Loading an existing message flow into memory" on page 1370
Use the Java API to modify a pattern instance to load a message flow into memory and make it available to other Java API methods.

"Testing Java code" on page 1387
After writing Java code to modify pattern instances, test the code to check that it works correctly.

Related reference:

"Java API for user-defined patterns" on page 5345
Use the Java API to write Java code to modify user-defined pattern instances.

Running PHP code using the Java API:

Use the Java API to modify a pattern instance to run PHP scripts from within your Java code.

About this task

You can run PHP scripts from within the Java API by using the runScript() method of the PatternInstanceManager object. In the following example, the Java API runs a PHP script in the `com.your.company.domain.code` plug-in, `templates/script.php`:

Example

```
@Override
public void onGeneratePatternInstance (PatternInstanceManager patternInstanceManager) {
 patternInstanceManager.runScript("com.your.company.domain.code", "templates/script.php");
}
```

Related concepts:

"Modifying pattern instances by using Java or PHP" on page 1364
Use Java or PHP code to modify pattern instances when the pattern user generates an instance of a user-defined pattern. For example, you can use Java or PHP code to modify the structure of a message flow based on the values of pattern parameters.

Related tasks:

"Modifying pattern instances by using the Java API" on page 1367
Add Java code to a code plug-in project to modify a pattern instance when the pattern instance is generated by a pattern user.

"Testing Java code" on page 1387
After writing Java code to modify pattern instances, test the code to check that it works correctly.

Related reference:

"Java API for user-defined patterns" on page 5345
Use the Java API to write Java code to modify user-defined pattern instances.

Updating filter tables on Route nodes:

Use the Java API to modify a pattern instance to update filter tables on Route nodes.

About this task

You can add and update rows on the filter table of a Route node.

Adding a new row

The following example shows you how to add a new row to a filter table by using the createRow() method:

1. The message flow and the Route node are loaded into memory.
2. The filter table of the Route node is loaded into memory by using the getFilterTable() method of the RouteNode object.
3. A new filter table row is created by using the createRow() method.
4. The value of the `filter pattern` property on this new row is set to `value="123"` by using the setFilterPattern() method.
5. The `routing output terminal` property is set to `NEWOUT` by using the setRoutingOutputTerminal() method.
6. The new row is then added to the filter table by using the addRow() method.

```
MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");
RouteNode routeNode = (RouteNode)mf1.getNodeByName("My Route Node");
RouteNode.FilterTable filterTable = (RouteNode.FilterTable)routeNode.getFilterTable();
RouteNode.FilterTableRow newRow = filterTable.createRow();
newRow.setFilterPattern("value=\"123\"");
newRow.setRoutingOutputTerminal("NEWOUT");
filterTable.addRow(newRow);
```

Updating a row

The following example shows you how to update rows on the filter table of a Route node.

1. The message flow, Route node, and filter table of the Route node are loaded into memory.
2. The rows of the filter table are loaded into memory by using the getRows() method.
3. The `filter pattern` property of the first row of the filter table is set to `value2="456"`.
4. The `routing output terminal` property of the first row of the filter table is set to `NEWOUT2`.

```
MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");
RouteNode routeNode = (RouteNode)mf1.getNodeByName("My Route Node");
RouteNode.FilterTable filterTable = (RouteNode.FilterTable)routeNode.getFilterTable();
Vector<RouteNode.FilterTableRow> filterTableRows = filterTable.getRows();
filterTableRows.get(0).setFilterPattern("value2=\"456\"");
filterTableRows.get(0).setRoutingOutputTerminal("NEWOUT2");
```

Related concepts:

"Modifying pattern instances by using Java or PHP" on page 1364
Use Java or PHP code to modify pattern instances when the pattern user generates an instance of a user-defined pattern. For example, you can use Java or PHP code to modify the structure of a message flow based on the values of pattern parameters.

Related tasks:

"Modifying pattern instances by using the Java API" on page 1367
Add Java code to a code plug-in project to modify a pattern instance when the pattern instance is generated by a pattern user.

"Loading an existing message flow into memory" on page 1370
Use the Java API to modify a pattern instance to load a message flow into memory and make it available to other Java API methods.

"Testing Java code"
After writing Java code to modify pattern instances, test the code to check that it works correctly.

Related reference:

"Java API for user-defined patterns" on page 5345
Use the Java API to write Java code to modify user-defined pattern instances.

Testing Java code:

After writing Java code to modify pattern instances, test the code to check that it works correctly.

Before you begin

Before you test your Java code, you must build the pattern plug-ins; see "Building pattern plug-ins" on page 1395.

About this task

You can test your Java code either by creating an instance of a pattern and checking that it is modified as you expect or by using the Java debugger to step through the code.

Procedure

- To test your user-defined pattern by creating an instance of the pattern, see "Testing a user-defined pattern" on page 1396.
- To use the Java debugger, you must open a temporary workspace to generate an instance of your user-defined pattern, then return to the original workspace to step through the code. To use the Java debugger, complete the following steps:
  1. In your Java code file, add a breakpoint to the required line of code. To add a breakpoint, right-click the marker bar (vertical ruler) to the left of the required code and click **Toggle Breakpoint**.
  2. In the Pattern Authoring editor, click the **Create Pattern** tab.
  3. Click **Debug Pattern**. The Workspace Launcher window opens.
  4. Select the temporary workspace in which to generate an instance of your user-defined pattern. Click **Browse** and select a workspace or enter a workspace in the **Workspace** field. Click **OK**. The selected workspace opens.
  5. In the temporary workspace, you must generate an instance of your user-defined pattern. In the Broker Development view, click the **Patterns Explorer** tab and select your user-defined pattern.
  6. In the **Pattern Specification** tab, click **Create New Instance**. Enter a name for the instance of your user-defined pattern and click **OK**.
  7. Enter any mandatory pattern parameters and click **Generate**. The generation of the pattern instance stops when it reaches the breakpoint in your Java code.

8. In the original workspace, in the Confirm Perspective Switch window, click **Yes**. The original workspace switches to debug mode and shows the code paused at your breakpoint.
9. You can now use the Java debugger to test your code; see "Java Debugger" on page 6723.
10. When you have finished testing the code in the Java debugger, close the temporary workspace and switch back to the Broker Application Development perspective in the original workspace; see "WebSphere Message Broker Toolkit perspectives" on page 34.

What to do next

Next:

You can either change your Java code and retest it or you can package and distribute your user-defined pattern:

- To change and retest your Java code, edit the code file and then repeat the steps in this topic.
- To package and distribute your user-defined pattern, see "Packaging and distributing pattern plug-ins" on page 1397

To ensure that changes in your Java code are included in your generated pattern, change the version number of the Java plug-in, as described in the following steps.

1. Update the version number of the Java plug-in in `manifest.mf`.
2. Update the version number of the pattern on the **Create Pattern** tab.
3. Re-create the pattern plug-ins and pattern archive, as described in "Building pattern plug-ins" on page 1395.

   To check that the pattern archive contains the correct versions of the plug-ins, you can look at the `feature.xml` file.
4. Install the pattern.

Related concepts:

"User-defined patterns" on page 1334
A *user-defined pattern* extends the function of WebSphere Message Broker so that you are able to create patterns that you can reuse within your organization.

Related tasks:

"Creating a pattern authoring project" on page 1337
Create a pattern authoring project and choose an exemplar for the project.

"Extending a user-defined pattern" on page 1340
Extend a user-defined pattern to customize its behavior so that it is easier to use or to provide guidance to pattern users.

"Modifying pattern instances by using the Java API" on page 1367
Add Java code to a code plug-in project to modify a pattern instance when the pattern instance is generated by a pattern user.

"Building pattern plug-ins" on page 1395
You must build the pattern plug-ins before you can distribute user-defined patterns to pattern users.

"Testing a user-defined pattern" on page 1396
Ensure that you test your user-defined pattern before you share it with your pattern users.

Modifying pattern instances by using PHP:

Add PHP code to a code plug-in project to modify a pattern instance when the pattern instance is generated by a pattern user.

Before you begin

Before you can complete this task, you must have completed the following tasks:

- "Creating a pattern authoring project" on page 1337
- "Selecting the source files to use for a user-defined pattern" on page 1338
- "Creating a code plug-in project" on page 1366

About this task

You can add PHP code to a code plug-in project by selecting a PHP script, which runs when a pattern instance is generated. You can choose whether to write the output of this script to a file. For example, you can write a PHP script that creates an ESQL file that is used in your user-defined pattern.

If you added PHP support when you created the code plug-in, a PHP template and example scripts are created and added to the project. Alternatively, you can write your own PHP script. Regardless of how the script is created, you must add your own code within the script to complete the steps you require when the pattern instance is generated. For examples of PHP code for completing common tasks, see "Examples of PHP API code" on page 1390. For reference information about the PHP pattern authoring API, see "PHP API for user-defined patterns" on page 5345.

Procedure

To select a PHP script to run when a pattern instance is generated:

1. If the Add Java or PHP Code window is not open, in the **Pattern Configuration** tab of the Pattern Authoring editor, click **Add**. The Add Java or PHP Code window opens.
2. In the **Type of code** list, select PHP.
3. In the **Project name** list, select the project or plug-in that contains the PHP script that you want to run when a pattern instance is generated. If the required plug-in is installed, but is not in your workspace, click **Browse** and select the required plug-in.
4. In the **PHP file name** list, select the name of the script you want to run when a pattern instance is generated. The **PHP file name** list shows all the PHP scripts in the project or plug-in that you selected.
5. Optional: To write the output of the PHP script to a file, select **Write the output from the PHP file into an output file**.
   a. In the **Pattern instance project** list, select the pattern instance project in which you want to include the output file.
   b. In the **Output file name** field, enter a name for the output file. You can include a path in this field, for example `scripts/example.mqsc` to write the output to the `scripts` folder.
6. Click **OK**. The Add Java or PHP Code window closes and the PHP script that you selected is shown in the **Java and PHP Code** section of the **Pattern Configuration** tab.

7. Optional: The Java and PHP code listed in the Java and PHP Code section of the **Pattern Configuration** tab runs from top to bottom when a pattern instance is generated. To change the order in which the code runs, select the entry you want to move in the Java and PHP Code section and click the **Up** or **Down** button to change the position of the entry in the list.

What to do next

Next:

In the PHP script that you selected to run when the pattern instance is generated, you must now write the code to perform the steps you require. For examples of PHP code for completing common tasks, see "Modifying pattern instances by using PHP" on page 1389. For reference information about the PHP pattern authoring API, see "PHP API for user-defined patterns" on page 5345.

Related concepts:

"User-defined patterns" on page 1334
A *user-defined pattern* extends the function of WebSphere Message Broker so that you are able to create patterns that you can reuse within your organization.

Related tasks:

"Creating a pattern authoring project" on page 1337
Create a pattern authoring project and choose an exemplar for the project.

"Selecting the source files to use for a user-defined pattern" on page 1338
Select the source files to include in your pattern.

"Defining the target properties" on page 1342
Identify the target properties in the workspace project so that you can define the pattern interface and pattern users can modify the pattern parameters.

"Extending a user-defined pattern" on page 1340
Extend a user-defined pattern to customize its behavior so that it is easier to use or to provide guidance to pattern users.

"Testing PHP API code" on page 1394
After writing PHP code to modify pattern instances, test the code to check that it works correctly.

"Examples of PHP API code"
Use the following examples of PHP API code for common tasks to help you write your own PHP code to modify pattern instances.

Related reference:

"PHP API for user-defined patterns" on page 5345
Use the PHP API to write PHP code to develop user-defined pattern applications.

Examples of PHP API code:

Use the following examples of PHP API code for common tasks to help you write your own PHP code to modify pattern instances.

About this task

The following examples show PHP code that you can use to complete common tasks when you are creating user-defined patterns:

Procedure

- Running additional PHP scripts. The following example shows how to use a PHP script to run another PHP script, by using the mb_pattern_run_template() function. The mb_pattern_run_template() function uses the following parameters:
  1. The first parameter required by mb_pattern_run_template() is the name of the project containing the PHP template; `Transform` in this example.
  2. The second parameter required by mb_pattern_run_template() is the name of the PHP script to run; in this example `example.esql.php`, which is in a subfolder called `mqsi`.
  3. The third parameter required by mb_pattern_run_template() is the name of the file to which the output is written; in this example `example.esql`, in subfolder `mqsi`.

  The user-defined pattern in this example contains a Boolean pattern parameter called `includeErrorHandling`. This pattern parameter is used to create a check box in the pattern:

```
<?php
    if ($_MB['PP']['includeErrorHandling'] == 'true') {
        mb_pattern_run_template("Transform", "mqsi/example.esql.php", "mqsi/example.esql");
    }
?>
```

- Using PHP scripts with markup for ESQL. In the following example, the user-defined pattern contains a pattern parameter called `errorQueue`, which can contain the values `none` or `errorAction`:

```
BROKER SCHEMA mqsi
<?php
    if ($_MB['PP']['errorAction'] == 'errorQueue') {
        echo "DECLARE ErrorAction EXTERNAL CHARACTER '".$_MB['PP']['errorAction']."';";

        echo <<<ESQL

        CREATE FILTER MODULE CheckErrorAction
            CREATE FUNCTION Main() RETURNS BOOLEAN
            BEGIN
                IF ErrorAction = 'errorQueue' THEN
                    RETURN TRUE;
                ELSE
                    RETURN FALSE;
                END IF;
            END;
        END MODULE;

        ESQL;
    }
?>
```

- Using PHP scripts to remove files from a project. In the following example, the script first checks the value of the check box pattern parameter, which has the pattern parameter ID `pp1`. If this parameter is set to `false`, the script deletes the `Log.msgflow` and `Log.esql` files from the pattern instance project. Message flow files must be deleted by using the removeMessageFlow() function. Non-message flow files can be deleted by using the standard PHP function unlink():

```
if ($_MB['PP']['pp1'] == 'false') {

    $pim = $_MB["PATTERN_INSTANCE_MANAGER"];

    $logmsgflow = $pim->getMessageFlow("Example_Flows", "mqsi/Log.msgflow");
    $pim->removeMessageFlow($logmsgflow);

    $piworkspace = $pim->getWorkspaceLocation();
```

```
    $piname = $pim->getPatternInstanceName();
    $logesql = $piworkspace . "/" . $piname . "_Example_Flows/mqsi/Log.esql";
    unlink($logesql);
}
```

- Running Java code from PHP by using a static Java method. The following example shows a Java class, *MyClass*, that can be run from within a PHP script. The class contains a static method:

```
package com.your.company.domain.code;

import com.ibm.broker.config.appdev.MessageFlow;
import com.ibm.broker.config.appdev.patterns.GeneratePatternInstanceTransform;
import com.ibm.broker.config.appdev.patterns.PatternInstanceManager;

public class MyClass implements GeneratePatternInstanceTransform {

    public static void doSomethingUseful(String message) {
        System.out.println("Message received [" + message + "]");
    }

    @Override
    public void onGeneratePatternInstance(PatternInstanceManager patternInstanceManager) { }
}
```

  You can run the *MyClass* class from PHP as shown in the following example. In the line `$class = $pim->getPluginClass("com.your.company.domain.code", "com.your.company.domain.code.MyClass")`, the first argument, `com.your.company.domain.code`, is the ID of the plug-in that contains the Java class. The second argument, `com.your.company.domain.code.MyClass`, is the name of the class.

```
<?php
    $pim = $_MB["PATTERN_INSTANCE_MANAGER"];
    $class = $pim->getPluginClass("com.your.company.domain.code", "com.your.company.domain.code.My
    java_import("com.your.company.domain.code.MyClass");
    MyClass::doSomethingUseful("Hello!");
?>
```

- Running Java code from PHP by using a non-static Java method. The following example shows a Java class, *MyClass*, that can be run from within a PHP script. The class contains a method that is not static:

```
package com.your.company.domain.code;

import com.ibm.broker.config.appdev.MessageFlow;
import com.ibm.broker.config.appdev.patterns.GeneratePatternInstanceTransform;
import com.ibm.broker.config.appdev.patterns.PatternInstanceManager;

public class MyClass implements GeneratePatternInstanceTransform {

   public void doSomethingUseful(String message) {
      System.out.println("Message received [" + message + "]");
   }

   @Override
   public void onGeneratePatternInstance(PatternInstanceManager patternInstanceManager) { }
}
```

  You can run the *MyClass* class from PHP as shown in the following example. In the line `$class = $pim->getPluginClass("com.your.company.domain.code", "com.your.company.domain.code.MyClass")`, the first argument, `com.your.company.domain.code`, is the ID of the plug-in that contains the Java class. The second argument, `com.your.company.domain.code.MyClass`, is the name of the class. A new instance of the class is created in the line `$obj = $class->newInstance()`. In the last line of the example, the `doSomethingUseful()` method of the class is run.

```
<?php
    $pim = $_MB["PATTERN_INSTANCE_MANAGER"];
    $class = $pim->getPluginClass("com.your.company.domain.code", "com.your.company.domain.code
    $obj = $class->newInstance();
    $obj->doSomethingUseful("Hello!");
?>
```

- Using PHP scripts to read table values. In the following example, the PHP script extracts the values from an array stored in the superglobal variable *$_MB*. First the script searches inside the array for any values stored in the *event* and *response* variables, and then returns the values:

```
<?php
 $pim = $_MB["PATTERN_INSTANCE_MANAGER"];
 $table = $pim->getParameterTable("table1");
 $count = $table->getRowCount();
 for ($j=0;$j<$count ;$j++ ) {
  echo "".$_MB['PP']['table1'][$j]['event']."';\n";
  echo "".$_MB['PP']['table1'][$j]['response']."';\n";
 }
?>
```

  The array has the following structure:

```
array(4) {
 ["PATTERN_INSTANCE_MANAGER"]=>
 object(Java)#1 (0) {
 }
 ["PP"]=>
  array(10) {
   ["queueName"]=>string(11) "QP.QUEUE.QS"
   ["configurePrefixSuffix"]=>string(5) "false"
   ["queuePrefix"]=>string(3) "QP."
   ["queueSuffix"]=>string(3) ".QS"
   ["logging"]=>string(5) "queue"
   ["generateMessageSet"]=>string(4) "true"
   ["table1"]=>
   array(2) {
     [0]=>
      array(2) {
     ["event"]=>string(6) "event1"
     ["response"]=>string(9) "response1"
      }
     [1]=>
      array(2) {
     ["event"]=>string(6) "event2"
     ["response"]=>string(9) "response2"
      }
   }
 }
```

Related concepts:

"User-defined patterns" on page 1334
A *user-defined pattern* extends the function of WebSphere Message Broker so that you are able to create patterns that you can reuse within your organization.

Related tasks:

"Creating a pattern authoring project" on page 1337
Create a pattern authoring project and choose an exemplar for the project.

"Extending a user-defined pattern" on page 1340
Extend a user-defined pattern to customize its behavior so that it is easier to use or to provide guidance to pattern users.

"Modifying pattern instances by using PHP" on page 1389
Add PHP code to a code plug-in project to modify a pattern instance when the pattern instance is generated by a pattern user.

"Testing PHP API code"
After writing PHP code to modify pattern instances, test the code to check that it works correctly.

Related reference:

"PHP API for user-defined patterns" on page 5345
Use the PHP API to write PHP code to develop user-defined pattern applications.

Testing PHP API code:

After writing PHP code to modify pattern instances, test the code to check that it works correctly.

Before you begin

Before you test your PHP code, you must build the pattern plug-ins; see "Building pattern plug-ins" on page 1395.

About this task

You can check that your PHP scripts work correctly by creating an instance of your user-defined pattern. When you test your user-defined pattern, a new workspace is opened in which to create the instance of your pattern. If you want to change your PHP code and retest it, you are not required to open a new workspace after each change. If you make any other changes to the pattern authoring project, you must rebuild the pattern plug-ins and launch a new workspace.

Procedure

To test your user-defined pattern, see "Testing a user-defined pattern" on page 1396. A new workspace is opened in which you generate an instance of your user-defined pattern. After the pattern instance is generated, check that your PHP code has worked correctly. For example, if you write PHP code to remove files from a pattern instance project, check that the files are deleted.

What to do next

Next:

You can either change your PHP code and retest it or you can package and distribute your user-defined pattern:

- To change and retest your PHP code, repeat the following steps until all your changes are complete. You can keep the test workspace open and continue to change your code and generate pattern instances without closing and relaunching the test workspace after each change:
  1. Edit the PHP code file in your main workspace and save the file.
  2. In the test workspace, generate a new instance of your user-defined pattern and check the results.
- To package and distribute your user-defined pattern, see "Packaging and distributing pattern plug-ins" on page 1397.

Related concepts:

"User-defined patterns" on page 1334
A *user-defined pattern* extends the function of WebSphere Message Broker so that you are able to create patterns that you can reuse within your organization.

Related tasks:

"Creating a pattern authoring project" on page 1337
Create a pattern authoring project and choose an exemplar for the project.

"Extending a user-defined pattern" on page 1340
Extend a user-defined pattern to customize its behavior so that it is easier to use or to provide guidance to pattern users.

"Modifying pattern instances by using PHP" on page 1389
Add PHP code to a code plug-in project to modify a pattern instance when the pattern instance is generated by a pattern user.

"Building pattern plug-ins"
You must build the pattern plug-ins before you can distribute user-defined patterns to pattern users.

"Testing a user-defined pattern" on page 1396
Ensure that you test your user-defined pattern before you share it with your pattern users.

Building pattern plug-ins

You must build the pattern plug-ins before you can distribute user-defined patterns to pattern users.

Before you begin

Before you start:

Complete the following tasks:

- "Creating a pattern authoring project" on page 1337
- "Selecting the source files to use for a user-defined pattern" on page 1338

About this task

Procedure

To build the pattern plug-ins:

1. In the Pattern Authoring editor, click the **Create Pattern** tab.
2. Enter values in the **Plug-in ID** and **Version** fields for your plug-in according to the naming and version numbering conventions within your organization. For more information about naming conventions for Eclipse plug-ins, see http://wiki.eclipse.org/Naming_Conventions. For more information about version numbering conventions for Eclipse plug-ins, see http://wiki.eclipse.org/Version_Numbering.
3. Optional: Select **Create translation plug-ins**, if required.
4. Click **Create Pattern Plug-ins**.

   The plug-ins for your user-defined pattern are created and displayed in the Projects section of the Broker Development view. The plug-ins are named *PluginID*, and *PluginID*.doc, where *PluginID* is the value entered in the **Plug-in ID** field in the **Create Pattern** tab. If **Create translation plug-ins** is selected, two additional plug-ins are created, named *PluginID*.nl1 and *PluginID*.nl1.doc.

What to do next

Next:

You must now test the pattern, see "Testing a user-defined pattern" on page 1396.

Related concepts:

"User-defined patterns" on page 1334
A *user-defined pattern* extends the function of WebSphere Message Broker so that you are able to create patterns that you can reuse within your organization.

"Creating a user-defined pattern" on page 1336
The workflow showing the actions required for pattern authoring.

Related tasks:

"Creating a pattern authoring project" on page 1337
Create a pattern authoring project and choose an exemplar for the project.

"Selecting the source files to use for a user-defined pattern" on page 1338
Select the source files to include in your pattern.

Testing a user-defined pattern

Ensure that you test your user-defined pattern before you share it with your pattern users.

Before you begin

Before you start:

Complete the following tasks:

- "Creating a pattern authoring project" on page 1337
- "Selecting the source files to use for a user-defined pattern" on page 1338
- "Building pattern plug-ins" on page 1395

About this task

Before you distribute your user-defined pattern, you can test it to ensure that it works correctly. To test your user-defined pattern, the Pattern Authoring editor opens a temporary workspace in which you can generate an instance of your user-defined pattern. You can then check that your user-defined pattern works correctly. If your user-defined pattern contains Java pattern authoring API code, see "Testing Java code" on page 1387. If your user-defined pattern contains PHP pattern authoring API code, see "Testing PHP API code" on page 1394.

Procedure

To test your user-defined pattern:

1. In the Pattern Authoring editor, click the **Create Pattern** tab.
2. Click **Test Pattern**. The Workspace Launcher window opens.
3. Select the temporary workspace in which to generate an instance of your user-defined pattern. Click **Browse**. Select a workspace or enter a workspace in the **Workspace** field. Click **OK**. The selected workspace opens.
4. Click the **Patterns Explorer** tab in the Broker Development view. Your new user-defined pattern is shown in the Patterns Explorer view.
5. Test your pattern by generating a new pattern instance. See "Using patterns" on page 1312.
6. If you want to change your pattern, change the pattern authoring project and then rebuild the pattern plug-ins. If you are changing only Java or PHP pattern authoring API code, you do not have to rebuild the pattern plug-ins; see "Testing Java code" on page 1387 and "Testing PHP API code" on page 1394.

What to do next

Next:

After you have tested the pattern, you can share it with the pattern users, see "Packaging and distributing pattern plug-ins."

Related concepts:

"User-defined patterns" on page 1334
A *user-defined pattern* extends the function of WebSphere Message Broker so that you are able to create patterns that you can reuse within your organization.

"Creating a user-defined pattern" on page 1336
The workflow showing the actions required for pattern authoring.

"Modifying pattern instances by using Java or PHP" on page 1364
Use Java or PHP code to modify pattern instances when the pattern user generates an instance of a user-defined pattern. For example, you can use Java or PHP code to modify the structure of a message flow based on the values of pattern parameters.

Related tasks:

"Creating a pattern authoring project" on page 1337
Create a pattern authoring project and choose an exemplar for the project.

"Selecting the source files to use for a user-defined pattern" on page 1338
Select the source files to include in your pattern.

"Building pattern plug-ins" on page 1395
You must build the pattern plug-ins before you can distribute user-defined patterns to pattern users.

Packaging and distributing pattern plug-ins

Package the plug-ins for your user-defined pattern into a pattern archive so that pattern users can download and install the pattern on their own system.

Before you begin

Before you start:

Before you can complete this task, you must have completed the following tasks for your user-defined pattern:

1. "Creating a pattern authoring project" on page 1337
2. "Selecting the source files to use for a user-defined pattern" on page 1338
3. "Building pattern plug-ins" on page 1395
4. "Testing a user-defined pattern" on page 1396

About this task

To enable pattern users to use your user-defined pattern, you must create a pattern archive from the pattern plug-ins. The pattern archive is published to a pattern community site, which is either a website or shared file system directory. The pattern user can download and install the pattern archive from the pattern community site. The following tasks describe this process:

- The pattern author creates a pattern archive, see "Creating a pattern archive" on page 1398.
- The pattern author uploads the pattern archive to a patterns community site, see "Uploading a pattern archive" on page 1399.

- The pattern user downloads and installs the pattern archive, see “Downloading and installing a pattern archive” on page 1400.

If you want to uninstall a user-defined pattern, see “Uninstalling a pattern archive” on page 1405.

Related concepts:

“User-defined patterns” on page 1334
A *user-defined pattern* extends the function of WebSphere Message Broker so that you are able to create patterns that you can reuse within your organization.

“Creating a user-defined pattern” on page 1336
The workflow showing the actions required for pattern authoring.

Creating a pattern archive:

Package your user-defined pattern as a pattern archive so that you can distribute it to pattern users.

Before you begin

Before you start:

Before you can complete this task, you must have completed the following tasks:

- “Creating a pattern authoring project” on page 1337
- “Selecting the source files to use for a user-defined pattern” on page 1338
- “Building pattern plug-ins” on page 1395
- “Testing a user-defined pattern” on page 1396

About this task

When you have tested your user-defined pattern and you are satisfied with its content and quality, package it into a pattern archive so that you can distribute it to your pattern users.

Procedure

To package your user-defined pattern into a pattern archive, complete the following tasks:

1. Open your pattern instance project, click the **Create Pattern** tab.
2. In the Pattern Distribution section, click **Create Pattern Archive**. The "Create a Pattern Archive" window opens.

   The **File name** defaults to the first segment of the pattern project name appended by `.patternzip`
3. To select a location, click **Browse**, and navigate to the required location. The **Location** is the file system location to which the pattern archive is saved. If you want to publish the pattern archive directly to a pattern community shared file system location to which you have write access, select that location.
4. Click **Finish**. A progress monitor is displayed at the bottom of the window.
   - If you click **Cancel** while the creation process is being run, the progress monitor stops and a warning message is displayed saying that the operation has stopped and the pattern archive will not be created. You can choose to restart the creation process by clicking **Finish**, or cancel the creation process by clicking **Cancel**. If the pattern archive has already been created, it is

removed from the target location and its resources are deleted from the workspace. If you click **Cancel** again, the window is closed.

- If an error occurs during the creation process, for example, the disk is full, or a write access permissions problem has occurred, an error message is displayed. You can correct the problem without exiting the window, and try again.

If the process completes successfully, the window closes.

Results

The pattern plug-ins and features are generated in the workspace. If the plug-ins already exist in the workspace, you are asked if you want to overwrite them; if you have already created the `.doc` plug-in or your translated files, you might choose not to overwrite the plug-ins. The pattern is temporarily displayed in the Patterns Explorer view. A pattern archive with the extension `.patternzip` is created in the target location.

What to do next

Next:

- If you published the pattern archive directly to a pattern community shared file system location, pattern users can now download the pattern archive, see "Downloading from a shared file system" on page 1404.
- If you did not publish the pattern archive directly to a pattern community shared file system location, you can upload the pattern archive to a pattern community site to distribute the pattern archive to pattern users, see "Uploading a pattern archive."

Related concepts:

"User-defined patterns" on page 1334
A *user-defined pattern* extends the function of WebSphere Message Broker so that you are able to create patterns that you can reuse within your organization.

"Creating a user-defined pattern" on page 1336
The workflow showing the actions required for pattern authoring.

Related tasks:

"Uploading a pattern archive"
The pattern author can upload a pattern archive to a pattern community site so that it can be distributed to pattern users.

"Downloading and installing a pattern archive" on page 1400
To use a user-defined pattern, download and install the pattern archive for the user-defined pattern.

"Uninstalling a pattern archive" on page 1405
Uninstall the pattern archive to remove a user-defined pattern from the Patterns Explorer view.

Uploading a pattern archive:

The pattern author can upload a pattern archive to a pattern community site so that it can be distributed to pattern users.

Before you begin

Before you start:

Before you can complete this task, the pattern author must have completed the following task:

- "Creating a pattern archive" on page 1398

About this task

To publish a user-defined pattern, the pattern author must upload the pattern archive to a pattern community site, which is either a pattern community website or a pattern community shared file system location. To upload to a pattern community website, the pattern author must have write access to the website.

Procedure

- To upload to a pattern community website:
  - If the pattern community website supports the upload interface, the pattern author can specify the pattern metadata, for example, category, icon, and description, and then upload the pattern archive to the pattern community website.
  - If the pattern community website does not support the upload interface, the pattern archive must be passed to the administrator of the pattern community website so that it can be published.
- To upload to a pattern community shared file system location:
  - If the pattern author has write access to the shared file location, that location can be selected when the pattern archive is created. See "Creating a pattern archive" on page 1398.
  - If the pattern author does not have write access to the shared file location, the pattern archive must be passed to the administrator of the shared file location so that it can be published.

What to do next

Next:

The pattern user can now download and install the pattern archive, see "Downloading and installing a pattern archive."

Related concepts:

"User-defined patterns" on page 1334
A *user-defined pattern* extends the function of WebSphere Message Broker so that you are able to create patterns that you can reuse within your organization.

"Creating a user-defined pattern" on page 1336
The workflow showing the actions required for pattern authoring.

Related tasks:

"Downloading and installing a pattern archive"
To use a user-defined pattern, download and install the pattern archive for the user-defined pattern.

"Uninstalling a pattern archive" on page 1405
Uninstall the pattern archive to remove a user-defined pattern from the Patterns Explorer view.

Downloading and installing a pattern archive:

To use a user-defined pattern, download and install the pattern archive for the user-defined pattern.

Before you begin

Before you start:

Before you can complete this task, you must decide which user-defined pattern you want to use from a pattern community site.

About this task

Before you can update a pattern archive to a newer version of an installed pattern archive, you must uninstall the current version of the pattern archive, see "Uninstalling a pattern archive" on page 1405.

When you have chosen a user-defined pattern that you want to use from a pattern community site, download and install the pattern archive for that user-defined pattern to your local system by using one of the following methods:

- "Downloading from a pattern community website"
- "Downloading from a pattern community website by using a helper application" on page 1402
- "Downloading from a shared file system" on page 1404

What to do next

Next:

You can now use the user-defined pattern in your projects, see "Using patterns" on page 1312.

Related concepts:

"User-defined patterns" on page 1334
A *user-defined pattern* extends the function of WebSphere Message Broker so that you are able to create patterns that you can reuse within your organization.

"Creating a user-defined pattern" on page 1336
The workflow showing the actions required for pattern authoring.

Related tasks:

"Uninstalling a pattern archive" on page 1405
Uninstall the pattern archive to remove a user-defined pattern from the Patterns Explorer view.

Downloading from a pattern community website:

To use a user-defined pattern in your WebSphere Message Broker projects, download and install the pattern archive for the user-defined pattern from a pattern community website.

Before you begin

Before you start:

Before you can complete this task, you must have the URL of the user-defined pattern that you want to download.

In this installation use case, only HTTP URLs that have no security are supported.

About this task

If you want to stop the download process, click **Cancel**, the pattern is not installed and the tasks are rolled back. If you click **Cancel** again, the Download Pattern window closes.

Procedure

1. In the Patterns Explorer view, click **Download**. The Download and Install Pattern window opens.
2. Type in, or copy and paste, the complete URL for the pattern archive into the **File Location** field, click **Download**. The progress monitor starts to run and the status line shows the individual tasks that are being performed.

Results

The new user-defined pattern is displayed in the Patterns Explorer view.

What to do next

Next:

You can now use the user-defined pattern in your projects. See "Using patterns" on page 1312.

Related concepts:

"User-defined patterns" on page 1334
A *user-defined pattern* extends the function of WebSphere Message Broker so that you are able to create patterns that you can reuse within your organization.

Related tasks:

"Downloading from a pattern community website by using a helper application"
To use a user-defined pattern in your WebSphere Message Broker projects, download and install the pattern archive for the user-defined pattern from a pattern community website by using a helper application.

"Downloading from a shared file system" on page 1404
To use a user-defined pattern in your WebSphere Message Broker projects, download and install the pattern archive for the user-defined pattern from a pattern community site that is a shared file system by using the file URL or by browsing for a file.

"Uninstalling a pattern archive" on page 1405
Uninstall the pattern archive to remove a user-defined pattern from the Patterns Explorer view.

Downloading from a pattern community website by using a helper application:

To use a user-defined pattern in your WebSphere Message Broker projects, download and install the pattern archive for the user-defined pattern from a pattern community website by using a helper application.

Before you begin

Before you start:

Before you can complete this task, you must be on the pattern community website that contains the pattern archive that you want to download.

About this task

You can install a pattern archive directly from a pattern community website by using a helper application.

Procedure

1. Navigate to the pattern archive that you want to download.
2. Click the link for the required pattern archive. A new window opens. The content of the window depends on your browser, but the typical options are to either open the file or save the file.
3. Install the pattern archive:
   - Windows On Windows:

     Select the option to open the pattern archive. The Install Pattern Archive window opens and shows a progress monitor.

     If you have more than one WebSphere Message Broker Toolkit installed on your workstation, the helper application is for the WebSphere Message Broker Toolkit that you most recently installed. If you want to control which WebSphere Message Broker Toolkit is extended with your pattern, download the file to disk, follow the instructions in "Downloading from a shared file system" on page 1404, then start the WebSphere Message Broker Toolkit that you want to extend.
   - Linux On Linux:

     The exact steps you must complete to install the pattern archive depend on your Linux distribution and the file explorer and browser you use. Use the following steps as a guide.

     a. Save the pattern archive to your local file system. Select the option in your browser window to save the pattern archive to your local file system. Choose a location for the file and start the download.

     b. After the download completes, open the pattern archive with the **MBPatternInstaller** helper application. Open a file explorer and navigate to the pattern archive. Use the **Open with** menu option on the pattern archive and select `MBPatternInstaller.bin`. The MBPatternInstaller application is located in the WebSphere Message Broker Toolkit installation directory, for example, `/opt/IBM/WMBT700`. The Install Pattern Archive window opens and shows a progress monitor.
4. Close and relaunch the WebSphere Message Broker Toolkit.

Results

The new user-defined pattern is displayed in the Patterns Explorer view.

What to do next

Next:

You can now use the user-defined pattern in your projects. See "Using patterns" on page 1312.

Related concepts:

"User-defined patterns" on page 1334
A *user-defined pattern* extends the function of WebSphere Message Broker so that you are able to create patterns that you can reuse within your organization.

Related tasks:

"Downloading from a pattern community website" on page 1401
To use a user-defined pattern in your WebSphere Message Broker projects, download and install the pattern archive for the user-defined pattern from a pattern community website.

"Downloading from a shared file system"
To use a user-defined pattern in your WebSphere Message Broker projects, download and install the pattern archive for the user-defined pattern from a pattern community site that is a shared file system by using the file URL or by browsing for a file.

"Uninstalling a pattern archive" on page 1405
Uninstall the pattern archive to remove a user-defined pattern from the Patterns Explorer view.

Downloading from a shared file system:

To use a user-defined pattern in your WebSphere Message Broker projects, download and install the pattern archive for the user-defined pattern from a pattern community site that is a shared file system by using the file URL or by browsing for a file.

Before you begin

Before you start:

Before you can complete this task, you must have access to the pattern community site that contains the pattern archive that you want to download.

About this task

- If you want to download and install the pattern archive for the user-defined pattern from a pattern community website by using a helper application and you have more than one WebSphere Message Broker Toolkit installed on your workstation, the helper application is for the WebSphere Message Broker Toolkit that you most recently installed. If you want to control which WebSphere Message Broker Toolkit is extended with your pattern, download the file to disk, complete the following instructions, then start the WebSphere Message Broker Toolkit that you want to extend.
- If you want to stop the download process, click **Cancel**, the pattern is not installed and the tasks are rolled back. If you click **Cancel** again, the Download and Install Pattern window closes.

Procedure

In the Patterns Explorer view, click **Download**. The Download and Install Pattern window opens.

- If you want to download the pattern by using the file URL, type in, or copy and paste, the complete URL for the pattern archive into the **File Location** field, click **Download**. The progress monitor starts to run and the status line shows the individual tasks that are being performed.
- If you want to download the pattern by browsing your local file system:
  1. Click **Browse**. The Select The Pattern Archive To Install window opens.
  2. Navigate to the pattern archive that you want to use, click **Open**.
  3. Click **Download**. The progress monitor starts to run and the status line shows the individual tasks that are being performed.

Results

The new user-defined pattern is displayed in the Patterns Explorer view.

What to do next

Next:

You can now use the user-defined pattern in your projects; see "Using patterns" on page 1312.

Related concepts:

"User-defined patterns" on page 1334
A *user-defined pattern* extends the function of WebSphere Message Broker so that you are able to create patterns that you can reuse within your organization.

Related tasks:

"Downloading from a pattern community website" on page 1401
To use a user-defined pattern in your WebSphere Message Broker projects, download and install the pattern archive for the user-defined pattern from a pattern community website.

"Downloading from a pattern community website by using a helper application" on page 1402
To use a user-defined pattern in your WebSphere Message Broker projects, download and install the pattern archive for the user-defined pattern from a pattern community website by using a helper application.

"Uninstalling a pattern archive"
Uninstall the pattern archive to remove a user-defined pattern from the Patterns Explorer view.

Uninstalling a pattern archive:

Uninstall the pattern archive to remove a user-defined pattern from the Patterns Explorer view.

Before you begin

Before you start:

Before you can complete this task, you must have completed the following task:

- "Downloading and installing a pattern archive" on page 1400

Each user-defined pattern archive has one pattern feature, the name of which is based on the name of the main pattern plug-in. If the name of the plug-in is *com.your.company.Test*, the name of the pattern feature is *com.your.company.Test.Feature_<plug-in version>*. If translation was selected by the pattern author, the pattern archive has a second pattern feature, *com.your.company.Test.Feature_NL.<plug-in version>*. To uninstall a pattern archive you must know the feature name of the pattern archive to ensure that the correct user-defined pattern is removed.

About this task

You can uninstall a pattern archive for a user-defined pattern that you have previously installed. Uninstalling a pattern archive removes the associated user-defined pattern from the Patterns Explorer view.

Procedure

To uninstall the pattern archive:

1. Click **Help** > **Software Updates**. The "Software Updates and Add-ons" window opens.
2. Click the **Installed Software** tab.
3. Browse to and select the pattern feature. If translation was selected by the pattern author, two pattern features exist and both must be selected.
4. Click **Uninstall**. The Eclipse Update window opens.
5. You can either restart the workspace, or click **Apply Changes**.

Results

The pattern archive is removed and the user-defined pattern is removed from the Patterns Explorer view.

Related concepts:

"User-defined patterns" on page 1334
A *user-defined pattern* extends the function of WebSphere Message Broker so that you are able to create patterns that you can reuse within your organization.

"Creating a user-defined pattern" on page 1336
The workflow showing the actions required for pattern authoring.

Related tasks:

"Downloading and installing a pattern archive" on page 1400
To use a user-defined pattern, download and install the pattern archive for the user-defined pattern.

Developing message flow applications by using samples

Use the samples to learn more about the features that are available in WebSphere Message Broker, and how to use them.

Before you begin

Before you start:

If you are not familiar with message flow concepts, message model concepts, and common tasks to manage message flow resources, see "Processing messages" on page 1021.

This section describes one of the four methods that you can use to create message flow applications. The other three methods are described in the following sections:

- "Developing message flow applications by using patterns" on page 1309
- "Developing message flow applications from a wizard" on page 1408
- "Developing message flow applications from scratch" on page 1423

If you are unsure which method to use, see Chapter 9, "Developing message flow applications," on page 1019 for a discussion of the advantages of each option.

About this task

The supplied samples provide tested message flows that focus on a particular feature or function supported by WebSphere Message Broker. Samples are more limited in scope than patterns, but provide comprehensive examples of typical

message processing in particular scenarios. These samples are stand-alone; you can use them without having to create and configure additional resources.

Because they are created to demonstrate a particular facet of the product, the samples are not always designed to use the preferred techniques for a particular task. Therefore, use them as examples to learn how particular functions work, not as complete production-level solutions. You might find them helpful as a starting point to developing your own message flows, or as part of a larger solution.

To access the samples, complete the following steps.

Procedure

1. Open the samples gallery in the WebSphere Message Broker Toolkit by clicking **Help** > **Samples and Tutorials** > **WebSphere Message Broker Toolkit - Message Broker**.
2. Expand the available categories and click a sample to open it.

   The following categories of sample are available:

   Application samples
   The application samples are small end-to-end WebSphere Message Broker applications that were created by using the WebSphere Message Broker Toolkit. The Application samples demonstrate how to transform and route messages through message flows.

   Control and Routing samples
   The control and routing samples demonstrate how to use WebSphere Message Broker to control and route messages.

   File Processing samples
   The file processing samples demonstrate how to process files by using WebSphere Message Broker.

   Industry samples
   The Industry samples demonstrate how you can use WebSphere Message Broker in the retail and healthcare industries.

   Message Formats samples
   The message formats samples demonstrate how to process messages with different formats by using WebSphere Message Broker.

   Message Transformation samples
   The message transformation samples demonstrate alternative ways to develop WebSphere Message Broker applications to transform messages.

   Monitoring samples
   The monitoring samples demonstrate how to monitor your WebSphere Message Broker flows, including how to use WebSphere Business Monitor.

   Security samples
   The security samples demonstrate how to use Identity Security features in WebSphere Message Broker.

   Transports and Connectivity samples
   The transports and connectivity samples demonstrate the many ways of connecting WebSphere Message Broker to other applications, including EIS systems.

Web Service samples
The Web Service samples demonstrate how you can use WebSphere Message Broker as both a consumer and provider of web services.

For a full list of the available samples, see "Samples" on page 98.

3. Follow the instructions to import, deploy, and run the samples. The samples documentation also describes how to remove the samples when you have finished with them.

Related tasks:

"Developing message flow applications by using patterns" on page 1309
Create resources that are used to solve a specific business problem by using patterns.

"Developing message flow applications from a wizard"
A Quick Start wizard sets up the basic resources that are required to develop a message flow application. The wizard sets up and gives names to containers for the resources in which you then develop your message flow.

"Developing message flow applications from scratch" on page 1423
Design, create, and configure message flows by using the WebSphere Message Broker Toolkit.

"Connecting client applications" on page 1537
Connect your client applications to the broker by using one or more of the supported protocols from other resources and software servers in your network.

"Routing messages" on page 2209
Route messages through your message flow or the broker by using one or more of the techniques described in this section.

"Transforming and enriching messages" on page 2227
Transform and enrich messages by using one or more of the techniques described in this section.

"Creating the Default Configuration" on page 106
You can create the Default Configuration of WebSphere Message Broker by using the Default Configuration wizard. You can also remove the Default Configuration by using the link provided.

"Resolving problems when running samples" on page 3366
Use the advice given here to help you to resolve common problems that can arise when you run or remove samples.

Related reference:

"What the Default Configuration wizard creates" on page 107
A table of the components that are created by the wizard, details of how to resolve problems, and how to view errors.

Developing message flow applications from a wizard

A Quick Start wizard sets up the basic resources that are required to develop a message flow application. The wizard sets up and gives names to containers for the resources in which you then develop your message flow.

Before you begin

Before you start: If you are not familiar with message flow concepts, message model concepts, and common tasks to manage message flow resources, see "Processing messages" on page 1021.

This section describes one of the four methods that you can use to create message flow applications. The other three methods are described in the following sections:

- "Developing message flow applications by using patterns" on page 1309
- "Developing message flow applications by using samples" on page 1406
- "Developing message flow applications from scratch" on page 1423

If you are unsure which method to use, see Chapter 9, "Developing message flow applications," on page 1019 for a discussion of the advantages of each one.

To learn about the wizards that you use to start developing message flows, see "Quick Start wizards overview."

About this task

Procedure

- "Creating an application from scratch" on page 1411
- "Creating an application based on WSDL or XSD files" on page 1413
- "Creating an application based on an existing message set" on page 1415
- "Creating an application that uses WebSphere Adapters" on page 1416
- "Creating an application by using the Configure New Web Service Usage wizard" on page 1417
- "Creating an application based on SCA import or export files" on page 1422

What to do next

After you have created a message flow, the next step is to develop it. For more information about how to start, see "Designing a message flow" on page 1455.

Related tasks:

"Developing message flow applications by using patterns" on page 1309
Create resources that are used to solve a specific business problem by using patterns.

"Developing message flow applications by using samples" on page 1406
Use the samples to learn more about the features that are available in WebSphere Message Broker, and how to use them.

"Developing message flow applications from scratch" on page 1423
Design, create, and configure message flows by using the WebSphere Message Broker Toolkit.

"Connecting client applications" on page 1537
Connect your client applications to the broker by using one or more of the supported protocols from other resources and software servers in your network.

"Routing messages" on page 2209
Route messages through your message flow or the broker by using one or more of the techniques described in this section.

"Transforming and enriching messages" on page 2227
Transform and enrich messages by using one or more of the techniques described in this section.

Quick Start wizards overview

You can use a Quick Start wizard to set up the basic resources that are required to develop a message flow application. The wizard sets up and gives names to containers for the resources that you need when you develop a message flow.

The resources that you can set up are described in the following list.

Message flow project
A specialized container in which you create and maintain all the resources that are associated with one or more message flows.

Message set project
A specialized container in which you create and maintain all the resources that are associated with a message set.

Message set
A container for grouping messages and associated message resources (elements, types, groups).

Message flow
A container for a sequence of processing steps that run in the broker when an input message is received.

Working set
A specialized container in which you can group related projects so that you limit the number of resources that are displayed in the Broker Development view.

The following Quick Start wizards and links are available.

- **Start from scratch**, described in "Creating an application from scratch" on page 1411
- **Start from existing message set**, described in "Creating an application based on an existing message set" on page 1415
- **Start from adapter connection**, described in "Creating an application that uses WebSphere Adapters" on page 1416
- **Start from SCA Import or Export**, described in "Creating an application based on SCA import or export files" on page 1422
- **Start from patterns** link which opens the Patterns Explorer view, described in "Developing message flow applications by using patterns" on page 1309
- **Start from samples** link which opens the **Samples and Tutorials** tab, described in "Developing message flow applications by using samples" on page 1406

Related concepts:

"Message flow projects" on page 1035
A message flow project is a specialized container in which you create and maintain all the resources associated with one or more message flows.

"Message set projects" on page 1161
A *message set project* is a specialized container in which you create and maintain all the resources associated with one message set.

"Message sets overview" on page 1162
A message set is a container for grouping messages and associated message resources (elements, types, groups).

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Resources" on page 36
The projects, folders, and files that you work with in the WebSphere Message Broker Toolkit workspace are called *resources*. By default, these resources are stored with their metadata in the workspace directory in your local file system. The workspace directory is created the first time that you start the WebSphere Message Broker Toolkit.

Related tasks:

“Creating a working set” on page 575
Create a working set to limit the number of resources that are displayed in the Broker Development view.

“Creating an application from scratch”
Use the "Start from scratch" wizard to create the basic resources that are required to develop a broker application.

“Creating an application based on WSDL or XSD files” on page 1413
You can use existing WSDL or XSD files as the basis for your solution.

“Creating an application based on an existing message set” on page 1415
Create a new application that is based on an existing message set.

Related reference:

“Broker Application Development perspective” on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

Creating an application from scratch

Use the "Start from scratch" wizard to create the basic resources that are required to develop a broker application.

About this task

The "Start from scratch" wizard creates the following resources:

- A message flow project
- A message set project, and sets up the project dependency
- A message set
- Optional: A message flow
- Optional: A working set

To create these resources, complete the following steps:

Procedure

1. Switch to the Broker Application Development perspective.
2. Open the "Start from scratch" wizard by using one of the following methods:
   - In the Projects section of the Broker Development view, if no projects are listed, the Quick Starts links are displayed. Click **Start from scratch**.
   - In the Projects section toolbar, click **Quick Starts**, the Quick Starts links are displayed. Click **Start from scratch**.

   The New Message Broker Application panel of the wizard is displayed. In this panel, you can type the names of the basic resources that are required to develop a broker application.
3. Type into the appropriate fields, the names of the message flow project, the message set project, the message set, the message flow, and the name of the working set that contains the two new projects. Default names of the message flow project, the message flow, and the working set are already displayed in the appropriate fields, but you can edit these fields by typing your own names for these resources.

   You can change any of the names that are displayed by typing into the appropriate field the name that you want. You can also clear either of the check boxes that relate to the creation of a new message flow or a new working set; if you clear either of the check boxes, you cannot enter text into the associated name field, and the associated resource file is not created.

4. Click **Next**. The Message Set Physical Formats panel is displayed. The panel lists three physical formats: **XML documents**, **Binary data** (Custom Wire Format), and **Text data** (TDS Format).
5. Select one or more of the check boxes to describe the type of message data that you want to process. If you do not select a check box, **XML documents** is selected by default.
6. Click **Finish** to complete the task. The "Start from scratch" wizard closes.

Results

The wizard creates a message flow project, message set project, message set, and, optionally, a message flow, with the names that you have specified. It also creates, optionally, a new working set, with the name that you have specified. The working set contains all the resources you have created, and the Broker Development view shows the new working set as the active working set. If you have chosen not to create a new working set, the projects are created in the active working set currently shown in the Broker Development view.

The XML, CWF, or TDS formats are created with default names for the message set.

The message flow, if created, is opened in the message flow editor.

What to do next

Next: If you have created a message flow, you can now go on to "Defining message flow content" on page 1488.

Related concepts:

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"MRM Custom Wire Format" on page 1214
Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not separated by delimiters.

"MRM TDS format" on page 1221
The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

"Message flow projects" on page 1035
A message flow project is a specialized container in which you create and maintain all the resources associated with one or more message flows.

"Message set projects" on page 1161
A *message set project* is a specialized container in which you create and maintain all the resources associated with one message set.

"Message sets overview" on page 1162
A message set is a container for grouping messages and associated message resources (elements, types, groups).

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Quick Start wizards overview" on page 1409
You can use a Quick Start wizard to set up the basic resources that are required to develop a message flow application. The wizard sets up and gives names to

containers for the resources that you need when you develop a message flow.

"Resources" on page 36
The projects, folders, and files that you work with in the WebSphere Message Broker Toolkit workspace are called *resources*. By default, these resources are stored with their metadata in the workspace directory in your local file system. The workspace directory is created the first time that you start the WebSphere Message Broker Toolkit.

Related tasks:

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Creating a working set" on page 575
Create a working set to limit the number of resources that are displayed in the Broker Development view.

"Creating an application based on WSDL or XSD files"
You can use existing WSDL or XSD files as the basis for your solution.

"Creating an application based on an existing message set" on page 1415
Create a new application that is based on an existing message set.

Related reference:

"Broker Application Development perspective" on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

"Rules for naming workspace objects" on page 6827

Creating an application based on WSDL or XSD files

You can use existing WSDL or XSD files as the basis for your solution.

Procedure

1. Open the New Message Broker Application wizard.
   a. Click **File** > **New** > **Project**, or right-click anywhere in the Broker Development view, and click **New> Project**. The New Project window opens.
   b. Double-click **Message Brokers**. A list of wizards is displayed.
   c. Select **Start from WSDL and/or XSD files**, and click **Next**.

   The first panel of the New Message Broker Application wizard is displayed.
2. Set up the basic resources that are required to develop a broker application that uses existing WSDL and XSD files as a starting point.
   - Type the name of your new application in the **Message flow project name** field.

     The name that you type is also displayed in the **Message set project name** and **Message set name** fields, but with 'MessageSet' appended.

     Similarly, the name that you type is also displayed in the **Message flow name** field (with 'Flow' appended), and in the **Working set name** field.
   - Click **Next**.

     You can change any of the names that are displayed by typing into the appropriate field the name that you want. You can also clear either of the check boxes that relate to the creation of a new message flow or a new working set; if you do this, you cannot enter text into the associated name field.

3. Select the WSDL or XSD files that you want to use as the initial contents of the message set.
   - To choose WSDL or XSD files that exist in your workspace, click **Use resources from the workspace**.

     You are presented with a list of resources from which you can choose. Resources are filtered to show resources only in the active working set.
   - To choose WSDL or XSD files that exist outside your workspace, click **Use external resources** and type a directory name in the **From directory** field. Click **Browse**.

     You are presented with a list of the items in that directory. Make your choice from this list.

   In both cases, a two-pane view is displayed. On the left side, containers (for example, projects, folders, and message sets) are displayed. On the right side, the contents of these containers are shown. Depending on which button you clicked, either a workspace view or a file system view of the resources is displayed.

   If the only use of the XSD file is from the WSDL bindings, you do not have to select an XSD file that a selected WSDL files depends on.

   The view incorporates an option that allows you to copy the source file into the `importFiles` directory of the message set.

   You can use this option as follows:
   - If you choose only WSDL files, you can select the check box.
   - If you choose only XSD files, the option is automatically selected and the check box is greyed out. If you subsequently select a WSDL file, the check box is enabled but the selection state is not changed; that is, the check box remains selected.
   - Regardless of what you select, if the `importFiles` folder exists in the message set project after the import, it is collapsed.
   - If you import only WSDL files, the wizard sets the default message domain to SOAP.
4. Click **Next**. If you selected one or more WSDL files, the WSDL files that you selected are shown in a check box tree, with the acceptable bindings for each file shown as children.
5. Select one or more bindings for each of the WSDL files that you selected. If you do not select at least one binding for each WSDL file, an error message is displayed and the **Next** and **Finish** buttons are disabled.
6. Click **Next**. If you selected one or more XSD files, the XSD files that you selected are displayed in the next pane, with the global elements for each file shown as children.
7. (Optional) Select the global elements from which you want to create message definitions. Click **Next**.
8. (Optional) If any errors or warnings are listed, either click **Finish**, if you want the import to be attempted regardless of the errors or warnings listed, or click **Cancel** to terminate the import. You can then correct any errors and attempt the import again.
9. Click **Finish**.

What to do next

After a WSDL file has been imported into a message set, you can drag the WSDL file onto the message flow editor. The next step is to develop the message flow. For

more information about how to start, see “Designing a message flow” on page 1455.

Related concepts:

“Generate XML schema” on page 1272
You can generate a schema file from a message model.

“Generate WSDL” on page 1274
A *Web Services Description Language* (WSDL) document specifies the interface to a web service, and enables a web service client to start it. A WSDL document that is generated from a message model defines web service requests and responses in terms of the messages that you have defined in that message model.

“XML Schema” on page 1172
XML Schema is an international standard from the World Wide Web Consortium (W3C) that defines a language for describing the structure of XML documents.

Related reference:

“Broker Application Development perspective” on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

Creating an application based on an existing message set

Create a new application that is based on an existing message set.

Before you begin

Before you start:

You must have created a message set by following the instructions in “Creating a message set” on page 2842.

About this task

To create a new application, complete the following steps.

Procedure

1. Click **File** > **New** > **Project**, or right-click anywhere in the Broker Development view and click **New** > **Project**. The New Project wizard opens.
2. Click **Message Brokers**. A list of wizards is displayed.
3. Click **Start from existing message set**, then click **Next**. The first panel of the New Message Broker Application wizard opens.

   The **Start from existing message set** wizard can also be found in the Project section toolbar. Click **Quick Starts** and select the **Start from existing message set** wizard.
4. Set up the basic resources that are required to develop a broker application from an existing message set, then click **Next**.
   - If the message set that you want to use is in a compressed (`.zip`) file, select **Import a message set from a .zip file**, and either type the location of the message set in the **.zip file** and **Compressed message set** fields, or click **Browse** and select, and open, the `.zip` file from the list that is displayed. Then select the required message set.

     If the `.zip` file that you specify does not contain a message set, you receive a message. You can then type a different location for the message set in the **.zip file** field. Otherwise, click **Cancel**.

- If the message set that you want to use is not in a compressed file, click **Create a new message set by copying an existing message set** and type into the **Message set to copy** field the name of the message set file that you want to copy.

  A list is displayed of the message set names from which you can choose. Message sets are filtered to show resources in the active working set only.

5. Enter the names of the projects, the message flow, the message set, and the working set that contains the two new projects. Default names of the message flow project, the message flow, and the working set are already supplied, but you can replace these names. However, if the message set is copied from a `.zip` file that is a project interchange file, you cannot edit the names of the message set project and the message set; the names are imported from the `.zip` file.
6. Click **Finish**.

   The new message set project, message set, message flow project, and message flow are created. A new working set is also created, if required. The new projects are displayed in the specified working set. The contents of the message set project and the message flow project are displayed in the Broker Development view. The message flow is opened in the message flow editor.

What to do next

The next step is to develop the message flow. For more information about how to start, see "Designing a message flow" on page 1455.

Related concepts:

"Message sets overview" on page 1162
A message set is a container for grouping messages and associated message resources (elements, types, groups).

"Message set projects" on page 1161
A *message set project* is a specialized container in which you create and maintain all the resources associated with one message set.

"Message flow projects" on page 1035
A message flow project is a specialized container in which you create and maintain all the resources associated with one or more message flows.

Related reference:

"Broker Application Development perspective" on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

Creating an application that uses WebSphere Adapters

Use the **Start from adapter connection** quick start wizard to create an application that uses WebSphere Adapters.

Before you begin

Before you start:

- Before you run the Adapter Connection wizard, you must gather some information from the Enterprise Information System (EIS). For more information about these prerequisite steps, as well as steps to complete after you have run the wizard, see "Connecting to an EIS by using the Adapter Connection wizard" on page 2037.

Procedure

1. In the Projects section of the Broker Development view, click **Quick Starts**, then click **Start from adapter connection**. The Adapter Connection wizard opens.
2. Follow the on-screen instructions.
3. Click **Finish**.

Results

When you have completed the steps in the wizard, the specified message set project contains a message set with a message type for each business object, and the specified message flow project references the message set project.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

Related tasks:

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

Creating an application by using the Configure New Web Service Usage wizard

Use these instructions to generate a message flow by using the Configure New Web Service Usage wizard.

About this task

This task topic describes how to create a new application by using the Configure New Web Service Usage wizard.

Procedure

1. Open a message set project that contains a WSDL file.
2. Select a WSDL file from either the message set or the `ImportFiles` folder and drag the WSDL file onto the Message Flow editor canvas. Validation occurs and if any of the following errors occurs, a message appears.

- The WSDL file does not come from either a message set or `ImportFiles` folder of the message set project.

  For a multiple-file WSDL, the process also checks that either imports inside the main WSDL have been properly imported into the message set, or imports are available in the `ImportFiles` folder.
- The message set that contains the WSDL file does not support any of the SOAP, XMLNSC, XMLNS, or MRM domains.

  However, if the message set that contains the WSDL file does not support only the SOAP domain, you can generate a flow based on the HTTP nodes, and the process continues.
- No HTTP bindings exist in the WSDL file.
- No port types exist in the WSDL file.

For the flow and subflows to be created correctly, the WSDL file that you are dropping onto the Message Flow editor canvas must be WS-I compliant. If no errors occur, the first page of the **Configure New Web Service Usage** wizard appears. For further information about the fields in the wizard, see Configure Web Service Usage details.

3. In Web service usage, select **Expose message flow as a Web service** or **Invoke Web service from message flow**. If you select **Expose message flow as a Web service**, you can use WebSphere Message Broker with other applications on the Web. If you select **Invoke Web service from message flow**, you use WebSphere Message Broker to start the Web service.
4. Select the **Port type** that you are going to use. By default, the initially selected port type is the first one that has at least one HTTP binding associated with it.

   You receive an error message in the following circumstances:
   - The selected port type does not contain at least one operation.
   - No SOAP bindings (with HTTP transport) in the WSDL document are associated with the port type.
5. Select the **Binding** that you are going to use. You receive an error message in the following circumstances:
   - The selected binding has no operations associated with it.
   - The selected binding has no ports associated with it.

   The **Service Port** field lists all the WSDL ports that point to a selected binding.
6. Select the **Binding operations** that you require. By default, only those operations that are implemented by the binding that you choose are selected.

   If you select one of the operations that is not implemented by the selected binding, you receive a warning message, but you can continue.
7. Click **Next**. For further information on the following fields, see File generation details.
8. Select **HTTP nodes** if you have imported the WSDL file from a message set and do not want the default value of **SOAP nodes**. If you select **HTTP nodes**, you see a message that explains the advantages of using the SOAP nodes. By using SOAP nodes, you can use features such as WS_Security and WS_Addressing. However, if the message set does not support the SOAP domain, you receive an error message.

   If you import the WSDL file from the `ImportFiles` folder, you cannot select **SOAP nodes**.

   All the file names that are generated, together with their location, are listed on this page.

A Details pane appears if any warnings occur about the subflow that is generated.

9. Click **Finish** to complete the wizard, create the subflow, and add appropriate nodes to the main flow. For details about the subflow and nodes that generated by the wizard if you select **Expose message flow as a Web service** as the initial step, see "Web service provider message flow generated."

   For details about the subflow and nodes that are generated by the wizard if you select **Invoke Web service from message flow** as the initial step, see "Web service consumer message flow generated" on page 1420.

What to do next

The next step is to develop the message flow. For more information about how to start, see "Designing a message flow" on page 1455.

Related concepts:

"Generate WSDL" on page 1274
A *Web Services Description Language* (WSDL) document specifies the interface to a web service, and enables a web service client to start it. A WSDL document that is generated from a message model defines web service requests and responses in terms of the messages that you have defined in that message model.

Related reference:

"Broker Application Development perspective" on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

"Configure New Web Service Usage wizard: Configure Web service usage details" on page 6392
Use this panel of the Configure New Web Service Usage wizard to configure a new Web service.

"Configure New Web Service Usage wizard: File generation details" on page 6393
Use the Configure New Web Service Usage wizard to specify file generation details.

Web service provider message flow generated

This provides additional information in relation to the Configure New Web Service Usage wizard about the message flow generated when the flow is a web service provider.

Note that the default name for the generated subflow is prefixed by the name of the WSDL file you selected.

Generated message flow

The message flow generated consists of a:

SoapInput node

This `SOAPInput` node completes the `LocalEnvironment` destination tree with the SOAP operation so that it can be followed either by a:

- `SOAPExtract` node, or by a
- `RouteToLabel` node. In this case, appropriate `Label` nodes need to be in place.

The `out` terminal of the `SOAPInput` node is connected to the `in` terminal of the `SOAPExtract` node.

Subflow node
The subflow node name reflects the name of the WSDL file.

SOAPReply node
This node sends the response message back to the originating client.

Typically, you connect the output of your node, or nodes, that handle your operation, or operations, to the `in` terminal of the `SOAPReply` node.

Generated message subflow

The generated subflow is constructed as follows:

- The input node is connected to the `SOAPExtract` node, which removes the SOAP envelope.

  The `SOAPExtract` node also allows for routing of the SOAP messages, based on the operation being performed. In particular, the SOAP message is routed to a `Label` node within the message flow as identified by the SOAP operation within the message.
- The `Failure` output terminal of the `SOAPExtract` node is connected to the `Output` node used when a process fails named, for example, `failure`.
- A `Label` node is generated for each SOAP operation and each `Label` node is connected to the corresponding `Output` node.
- Each `Output` node in the subflow corresponds to an output terminal for the `SOAPExtract` node in the main message flow.

  Therefore, there is one `failure` output terminal, plus one output terminal for each operation.

  Typically, you connect the output terminal corresponding to the operation you require to the node, or nodes, that handle this operation, for example, `Compute` node.

Related tasks:

"Generating a WSDL definition from a message set" on page 2968
To ensure the highest interoperability of your Web services, use the document style of WSDL whenever possible. If rpc style WSDL is necessary, use literal encoding.

"Creating an application by using the Configure New Web Service Usage wizard" on page 1417
Use these instructions to generate a message flow by using the Configure New Web Service Usage wizard.

Related reference:

"Generate WSDL wizard" on page 6382
The Generate WSDL wizard creates a WSDL definition from a message set.

Web service consumer message flow generated

This provides additional information in relation to the Configure New Web Service Usage wizard about the message flow generated when the flow is a web service consumer.

Note that the default name for the generated subflow is prefixed by the name of the WSDL file you selected.

Generated message flow

The low generated consists of a single node that has a number of output terminals:

- `Failure`
- `Error`

- `Fault`
- One more, corresponding to the name of the selected operation.

Typically, your message flow feeds an input message to the in terminal of the generated subflow node, and handles various outcomes of the web service invocation.

The default name of the subflow node is a combination of selected operation and WSDL file name. You can change the name of the corresponding `.msgflow` file on the second page of the wizard; see "Configure New Web Service Usage wizard: File generation details" on page 6393.

The generated `.msgflow` file is placed into the `gen` folder of the message set project; see "Generated message subflow" for details of this subflow.

Generated message subflow

The generated subflow is constructed as follows:

- A `SOAPRequest` node immediately follows an `Input` node. This is a synchronous request and response node that blocks after sending the request, until the response is received. The `SOAPRequest` node parses the response message.
- The `Failure` and `Error` terminals are connected to the `Output` nodes for failure and error respectively.
- The `Out` terminal is connected to the `SOAPExtract` node.

  The `SOAPExtract` node removes the SOAP envelope so that the body of a SOAP message is extracted.

  The `SOAPExtract` node also allows for routing of the SOAP messages, based on the operation being performed. Note that only the selected operation and fault are handled.

  In particular, the SOAP message is routed to a `Label` node within the message flow as identified by the SOAP operation or a `ws__Fault` label, if fault is returned from the web service.

  Each `Label` node is connected to the corresponding `Output` node.

  The `Failure` terminal of the `SOAPExtract` node is connected to the `Output` node for failure.
- Each `Output` node in the subflow corresponds to an output terminal for the subflow node.

  Therefore, there are three output terminals:

  – `Failure`
  – `Fault`
  – One for the selected operation.

Related tasks:

"Generating a WSDL definition from a message set" on page 2968
To ensure the highest interoperability of your Web services, use the document style of WSDL whenever possible. If rpc style WSDL is necessary, use literal encoding.

"Creating an application by using the Configure New Web Service Usage wizard" on page 1417
Use these instructions to generate a message flow by using the Configure New Web Service Usage wizard.

Related reference:

"Generate WSDL wizard" on page 6382
The Generate WSDL wizard creates a WSDL definition from a message set.

Creating an application based on SCA import or export files

You can create a new application that is based on existing SCA import or export files.

Procedure

1. Open the New Message Broker Application wizard by using either of the following methods:
   - In the Projects section of the Broker Development view, click **Quick Starts**, then click **Start from SCA Import or Export**.
   - Click **File** > **New** > **Project**, double-click **Message Brokers**, then click **Start from SCA Import or Export**.

   The first panel of the New Message Broker Application wizard is shown.
2. Enter names for the projects, message set and message flow, then click **Next**.

   You can change any of the names that are displayed by typing into the appropriate field the name that you want. You can also clear either of the check boxes that relate to the creation of a new message flow or a new working set, but you cannot enter text into the associated name field.
3. Select the project interchange file that contains the SCA import or export files that you want to use as the initial contents of the message set.
   - To choose SCA import or export files that exist in your workspace, click **Use resources from the workspace**.

     A list of available resources is displayed.
   - To choose SCA import or export files that exist outside your workspace, click **Use external resources** and enter a directory name, or click **Browse** to browse for a directory.

     A list of the items in that directory is displayed.

   In both cases, a two-pane view is displayed. On the left side, containers (for example, projects, folders, and message sets) are displayed. On the right side, the contents of these containers are shown. Depending on which option you selected, either a workspace view or a file system view of the resources is displayed.
4. To copy the imported files into the `ImportFiles` folder, select **Copy source file into the 'ImportFiles' directory of the message set project**.
5. To show all resources in the workspace. select **Apply working set filtering to artifact selection(s) on this page**; if you clear the check box, only resources in the current active working set are shown. If no working set is selected, this check box is not displayed.
6. Click **Next**. If you selected one or more files, the files that you selected are shown in a check box tree.
7. Select the project interchange file that contains the SCA components that you want to import into the message set. The project interchange file contains the SCA import components that you want to implement in WebSphere Message Broker, or the SCA export components that you want to invoke from WebSphere Message Broker.
8. Click **Finish**. The SCA components that you have selected are added to the message set that you have specified.

9. Drag an SCA component from the message set in the tree structure in the left pane onto the message flow editor pane.

   Dragging a `.outsca` file:

   - If the `.outsca` file contains either a single request-response operation, or more than one operation (one-way or request-response), you are prompted to select the operation. If the chosen operation is request-response, you can also choose whether to invoke the service synchronously or asynchronously. Synchronous invocation is the default value, and creates a SCARequest node. Choosing the asynchronous option creates a pair of SCAAsyncRequest and SCAAsyncResponse nodes.
   - If the chosen operation is one-way, synchronous invocation is the only option, and a SCARequest node is created.

   Dragging a `.insca` file:

   - If the `.insca` file contains only one-way operations, dragging a `.insca` file onto the canvas creates an SCAInput node.
   - Otherwise, a pair of SCAInput and SCAReply nodes is created.

What to do next

Next: Develop the message flow. For more information about how to start, see "Designing a message flow" on page 1455.

Related concepts:

"Service Component Architecture (SCA) overview" on page 2096
Service Component Architecture (SCA) is a specification that describes a model for building applications and systems using a service-oriented architecture (SOA).

Developing message flow applications from scratch

Design, create, and configure message flows by using the WebSphere Message Broker Toolkit.

Before you begin

Before you start: If you are not familiar with message flow concepts, message model concepts, and common tasks to manage message flow resources, see "Processing messages" on page 1021.

This section describes one of the four methods that you can use to create message flow applications. The other three methods are described in the following sections:

- "Developing message flow applications by using patterns" on page 1309
- "Developing message flow applications by using samples" on page 1406
- "Developing message flow applications from a wizard" on page 1408

If you are unsure which method to use, see Chapter 9, "Developing message flow applications," on page 1019 for a discussion of the advantages of each one.

About this task

To develop a message flow from scratch follow these activities in order.

1. For information about managing message flows, see "Managing message flow resources" on page 1424.

2. To learn about the choices that you have when designing the content of your message flows to provide the processing that your applications require, see “Designing a message flow” on page 1455.
3. To learn about constructing and configuring the content of your message flows, see “Defining message flow content” on page 1488.

Related tasks:

“Developing message flow applications by using patterns” on page 1309
Create resources that are used to solve a specific business problem by using patterns.

“Developing message flow applications by using samples” on page 1406
Use the samples to learn more about the features that are available in WebSphere Message Broker, and how to use them.

“Developing message flow applications from a wizard” on page 1408
A Quick Start wizard sets up the basic resources that are required to develop a message flow application. The wizard sets up and gives names to containers for the resources in which you then develop your message flow.

“Connecting client applications” on page 1537
Connect your client applications to the broker by using one or more of the supported protocols from other resources and software servers in your network.

“Routing messages” on page 2209
Route messages through your message flow or the broker by using one or more of the techniques described in this section.

“Transforming and enriching messages” on page 2227
Transform and enrich messages by using one or more of the techniques described in this section.

Managing message flow resources

Manage your message flows and associated resources in the WebSphere Message Broker Toolkit.

Before you start: Read “Message flows overview” on page 1022.

- “Creating a message flow project” on page 1425
- “Deleting a message flow project” on page 1427
- “Creating a broker schema” on page 1429
- “Creating a message flow” on page 1431
- “Opening an existing message flow” on page 1433
- “Copying a message flow by using copy” on page 1435
- “Analyzing planned changes to message flows” on page 1436
- “Renaming a message flow” on page 1438
- “Moving a message flow” on page 1439
- “Deleting a message flow” on page 1441
- “Deleting a broker schema” on page 1442
- “Version and keyword information for deployable objects” on page 1443
- “Showing resource references” on page 1447
- “Saving a message flow” on page 1448
- “Importing and exporting resources in a Project Interchange file” on page 1452

To learn more about message flows look at the following sample:

- Airline Reservations

In this sample, you can explore message flow resources, and learn how to create, delete, and rename the resources.

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Related tasks:

"Developing message flow applications by using patterns" on page 1309
Create resources that are used to solve a specific business problem by using patterns.

"Developing message flow applications by using samples" on page 1406
Use the samples to learn more about the features that are available in WebSphere Message Broker, and how to use them.

"Developing message flow applications from a wizard" on page 1408
A Quick Start wizard sets up the basic resources that are required to develop a message flow application. The wizard sets up and gives names to containers for the resources in which you then develop your message flow.

"Developing message flow applications from scratch" on page 1423
Design, create, and configure message flows by using the WebSphere Message Broker Toolkit.

"Connecting client applications" on page 1537
Connect your client applications to the broker by using one or more of the supported protocols from other resources and software servers in your network.

"Routing messages" on page 2209
Route messages through your message flow or the broker by using one or more of the techniques described in this section.

"Transforming and enriching messages" on page 2227
Transform and enrich messages by using one or more of the techniques described in this section.

Creating a message flow project

A message flow project is a container for message flows; you must create a project before you can create a message flow.

Before you begin

Before you start:

Read the concept topic about "Message flow projects" on page 1035.

About this task

The project and its resources are stored in a file system or in a shared repository. If you are using a file system, you can use the local file system or a shared drive. If you store files in a repository, you can use all the available repositories that are supported by Eclipse; for example, CVS.

To create a message flow project and other resource files that you need to start developing applications, you can use a Quick Start wizard (as described in "Quick Start wizards overview" on page 1409).

To create only a message flow project, perform the following actions:

Procedure

1. Switch to the Broker Application Development perspective.
2. Click **File** > **New** > **Message Flow Project** or right-click any resource in the Broker Development view and click **New** > **Message Flow Project**.

   You can also press Ctrl+N to display a dialog box, from which you select the wizard to create a new object. Click **Message Brokers** in the left view; the right view displays a list of objects that you can create for WebSphere Message Broker. Click **Message Flow Project** in the right view, then click **Next**. The New Message Flow Project wizard opens.
3. Enter a name for the project. Choose a project name that reflects the message flows that it contains. For example, if you want to use this project for financial processing message flows, you might give it the name Finance_Flows.
4. To use the default location for the new message project directory (the `\workspace` subdirectory of your current installation), leave the **Use default** check box selected (it is selected when the dialog box opens). When you choose this option, you cannot edit the **Directory** field.

   If you do not want to use the default location, clear the **Use default** check box and specify a location for the new message flow project files in **Directory**.
5. If this message flow project depends on other message flow projects or message set projects, click **Next**.

   A list of current projects is displayed. Select one or more message flow projects, one or more message set projects, or both, from the list to indicate the dependencies of this new message flow project. Message flow projects and message set projects are filtered to show only resources in the active working set.

   This message flow project depends on another message flow project if you intend to use common resources in it. Common resources that you can share between message flow projects are:
   a. ESQL subroutines (defined in broker schemas)
   b. Mappings
   c. Message sets
   d. Subflows

   For example, you might want to reuse a subflow that provides standard error processing, such as writing the message to a database, or recording a trace entry.

   This message flow project depends on a message set project if you intend to refer to the message it defines in ESQL in the message flow nodes.

   You can add dependencies after you have created the message flow project by right-clicking the project in the Broker Development view and clicking **Properties**. Click **Project References** and select the dependent message flow or message set project from the list of projects displayed.
6. Click **Finish** to complete the task.

Results

The project file is created in a directory that has the same name as your message flow project in the specified location. All other files that you create that are related to this message flow project are created in this same directory.

A default broker schema (`default`) is also created in the project. You can create and use different schemas in a single project to organize message flow resources and to provide the scope of resource names to ensure uniqueness.

What to do next

Next: create a message flow.

Related concepts:

"Project references" on page 44
When a projects refers to other projects in the workspace, this is called a *project reference*. When one project references another, the files in the referenced project are available for use by the referring project. For example to configure certain nodes, to create libraries of reusable message flows, and to enable Content Assist in the ESQL editor.

"Development repository" on page 45
Use a development repository to benefit from features such as version control and access control of files, which make it easier for teams to work on shared resources.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Broker schemas" on page 1036
A broker schema is a symbol space that defines the scope of uniqueness of the names of resources defined within it. The resources include message flows and other optional resources such as ESQL files, Java files, and mapping files.

"Quick Start wizards overview" on page 1409
You can use a Quick Start wizard to set up the basic resources that are required to develop a message flow application. The wizard sets up and gives names to containers for the resources that you need when you develop a message flow.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Deleting a message flow project"
A message flow project is the container in which you create and maintain all the resources associated with one or more message flows. These resources are created as files, and are displayed in the project in the Broker Development view. If you do not want to retain a message flow project, you can delete it.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

"Creating a broker schema" on page 1429
To organize your message flow project resources, and to define the scope of resource names to ensure uniqueness, you can create broker schemas. A default schema is created when you create the message flow project, but you can create additional schemas.

Related reference:

"Broker Application Development perspective" on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

"Rules for naming workspace objects" on page 6827

Deleting a message flow project

A message flow project is the container in which you create and maintain all the resources associated with one or more message flows. These resources are created as files, and are displayed in the project in the Broker Development view. If you do not want to retain a message flow project, you can delete it.

Before you begin

Before you start:

Complete the following steps.

- "Creating a message flow project" on page 1425
- Read the concept topic about "Message flow projects" on page 1035

About this task

When you delete a message flow project in the WebSphere Message Broker Toolkit, you delete the project and its resources. If you are using a shared repository, the repository might retain a copy of a deleted resource.

To delete a message flow project, complete the following steps.

Procedure

1. Select the message flow project that you want to delete, and click **Edit** > **Delete**. You can also press Delete, or right-click the project in the Broker Development view and click **Delete**.
2. You must choose if you want the contents of the message flow project folder to be deleted with this action in the displayed confirmation dialog box. The dialog box contains two buttons:
   - The first button confirms that all contents are to be deleted.
   - The second button requests that the directory contents are not deleted. The default action is not to delete the contents; therefore, the second button is selected by default when the dialog box is initially displayed.

   a. Select the appropriate button. If you choose not to delete the contents of the message flow project directory, all the files and the directory itself are retained.

      If you later create another project with the same name, and specify the same location for the project (or accept the location as the default value), you can access the files previously created.

      If you choose to delete all the contents, all files and the directory itself are deleted.
3. Click **Yes** to complete the delete request, or **No** to terminate the delete request.

Results

When you click **Yes**, the requested objects are deleted.

If you maintain resources in a shared repository, a copy is retained in that repository. You can follow the instructions provided by the repository supplier to retrieve the resource, if required.

If you are using the local drive or a shared drive to store your resources, no copy of the resource is retained. Be careful to select the correct resource when you complete this task.

Related concepts:

"By name linking" on page 43
You identify objects using a combination of a *namespace* and a name, referred to as a *fully qualified name*. The use of fully qualified names, called *by name linking*, makes it easy to identify and locate objects, and to correct broken references.

"Development repository" on page 45
Use a development repository to benefit from features such as version control and access control of files, which make it easier for teams to work on shared resources.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Creating a message flow project" on page 1425
A message flow project is a container for message flows; you must create a project before you can create a message flow.

Related reference:

"Broker Application Development perspective" on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

Creating a broker schema

To organize your message flow project resources, and to define the scope of resource names to ensure uniqueness, you can create broker schemas. A default schema is created when you create the message flow project, but you can create additional schemas.

Before you begin

Before you start:

Complete the following steps.

- "Creating a message flow project" on page 1425
- Read the concept topic about broker schemas

About this task

To create a broker schema:

Procedure

1. Switch to the Broker Application Development perspective.
2. Click **File** > **New** > **Broker Schema** or right-click any resource in the Broker Development view and click **New** > **Broker Schema**.

   You can also press Ctrl+N. This displays a dialog that allows you to select the wizard to create a new object. Click Message Brokers in the left view. The right view displays a list of objects that you can create for WebSphere Message Broker. Click **Broker Schema** in the right view, then click **Next**. The New Broker Schema wizard displays.
3. Enter the message flow project in which you want the new schema to be created. If a message flow project or one of its resources is highlighted when you start the wizard, that project name appears in the dialog box. If a name does not appear in this field, or if you want to create the schema in another project, click **Browse** and select the correct project from the displayed list. The message flow project list is filtered to show projects in the active working set.

   You can type the project name in, but you must enter a valid name. The dialog box displays a red cross and the error message `The specified project does`

not exist if your entry is not a valid project. You must specify a message flow project; if you select a message set project, you cannot complete the operation.

4. Enter a name for the schema. Choose a name that reflects the resources that it contains. For example, if you want to use this schema for message flows for retail applications, you might give it the name Retail.

   A broker schema name must be a character string that starts with a Unicode character followed by zero or more Unicode characters or digits, and the underscore symbol (_). You can use the period to provide a structure to the name, for example Stock.Common.

5. Click **Finish** to complete the task.

   If category view is selected in the Broker Development view when you create the schema, you see a message to say that the schema has been created, but it might not be visible in the Broker Development view when it is empty. To show the new schema in the Broker Development view, click **Hide Categories**

    on the Broker Development view toolbar.

Results

The schema directory is created in the project directory. If the schema name is structured by using periods, further subdirectories are defined. For example, the broker schema Stock.Common results in a directory Common, in directory Stock, in the message flow project directory.

Related concepts:

"Broker schemas" on page 1036
A broker schema is a symbol space that defines the scope of uniqueness of the names of resources defined within it. The resources include message flows and other optional resources such as ESQL files, Java files, and mapping files.

"Project references" on page 44
When a projects refers to other projects in the workspace, this is called a *project reference*. When one project references another, the files in the referenced project are available for use by the referring project. For example to configure certain nodes, to create libraries of reusable message flows, and to enable Content Assist in the ESQL editor.

"Development repository" on page 45
Use a development repository to benefit from features such as version control and access control of files, which make it easier for teams to work on shared resources.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Creating a message flow project" on page 1425
A message flow project is a container for message flows; you must create a project before you can create a message flow.

"Deleting a message flow project" on page 1427
A message flow project is the container in which you create and maintain all the resources associated with one or more message flows. These resources are created as files, and are displayed in the project in the Broker Development view. If you do not want to retain a message flow project, you can delete it.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can

create one or more message flows and deploy them to one or more brokers.

Related reference:

"Broker Application Development perspective" on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

"Rules for naming workspace objects" on page 6827

Creating a message flow

Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

Before you begin

Before you start:

- Complete the following task: "Creating a message flow project" on page 1425.
- Read the concept topic about "Broker schemas" on page 1036.

About this task

The mode that your broker is working in can affect the number of message flows that you can use; see "Restrictions that apply in each operation mode" on page 3657.

The message flow and its resources are stored in a file system or in a shared repository. If you are using a file system, it can be on the local drive, or a shared drive. If you store files in a repository, you can use all the available repositories that are supported by Eclipse; for example, CVS.

Use this process to create a complete message flow that you can deploy, or a subflow that provides a subset of function (for example, a reusable error processing routine), that you cannot deploy on its own.

To create a message flow and other resource files that you need to start developing applications, you can use a Quick Start wizard.

To create only a message flow, perform the following actions:

Procedure

1. If you have not already created the message flow project in which you want to create the message flow, you can either create it now, or you can create the message flow project as an optional step in creating the message flow (in step 4). If you want to create the message flow project first, see "Creating a message flow project" on page 1425. The project can be empty, or can have message flows defined in it.
2. Complete one of the following actions to open a message flow:
   - Click **File** > **New** > **Message Flow**.
   - Right-click a resource in the Broker Development view and click **New** > **Message Flow**.
   - Press Ctrl+N. This action displays a dialog box in which you can select the wizard to create an object:
     a. Click **Message Brokers** in the left view. The right view displays a list of objects that you can create for WebSphere Message Broker.
     b. Click **Message Flow** in the right view, then click **Next**. The New Message Flow wizard displays.

3. Identify the project in which you want to define the message flow. This field is filtered to show only resources in the active working set.
   - If you have a resource selected in the Broker Development view, the name of the corresponding project is displayed in the Message Flow Project field.
   - If you do not have a resource selected, the first field is blank.
     - If you have already created the message flow project for this message flow, you can perform either of the following actions:
       - Type the name of the project into the field.
       - Click the down-arrow and select the appropriate project from the list displayed.
     - If you have not already created the message flow project, select **New**. The New Message Flow Project wizard starts, and you can use it to create the message flow project for your new message flow, as described in "Creating a message flow project" on page 1425.

       When you have finished creating the message flow project, the New Message Flow Project wizard closes, and the name of your new message flow project is displayed in the Message Flow Project field of the New Message Flow window.

   If your entry is not a valid project name, the window displays a red cross, and the error message `The specified project does not exist`.
4. In the Message flow **Name** field, enter the name of the new message flow. You can use all valid characters for the name, but it is helpful to choose a name that reflects its function, for example, OrderProcessing.
5. Decide whether you want to use the default broker schema. When you create a message flow project, a default schema is created in it, and this default value is assumed unless you clear it. You can create and use different schemas in a single project to organize message flow resources, and to provide the scope of resource names to ensure uniqueness.
   - If you want the message flow to be created in the default broker schema, ensure that you select **Use default** in the Flow organization section.
   - If you want to use a different broker schema, clear **Use default**. You can now perform either of the following actions:
     - Enter the name of the broker schema into the **Schema** field.
     - Click **Browse** to select from the broker schemas in the message flow project.
6. Click **Finish**.

Results

The new message flow (`<message_flow_name>.msgflow`) is displayed in its project in the Broker Development view. The Editor view is empty, and ready to populate.

What to do next

Next, you can do either of the following tasks:

- "Saving a message flow" on page 1448
- "Defining message flow content" on page 1488

Related concepts:

"Development repository" on page 45
Use a development repository to benefit from features such as version control and access control of files, which make it easier for teams to work on shared resources.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Quick Start wizards overview" on page 1409
You can use a Quick Start wizard to set up the basic resources that are required to develop a message flow application. The wizard sets up and gives names to containers for the resources that you need when you develop a message flow.

"Broker schemas" on page 1036
A broker schema is a symbol space that defines the scope of uniqueness of the names of resources defined within it. The resources include message flows and other optional resources such as ESQL files, Java files, and mapping files.

"Packaging and deployment overview" on page 3210
Deployment is the process of transferring data to an execution group on a broker so that it can take effect in the broker. Message flows and associated resources are packaged in broker archive (BAR) files for deployment.

Related tasks:

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

"Creating a message flow project" on page 1425
A message flow project is a container for message flows; you must create a project before you can create a message flow.

Related reference:

"Broker Application Development perspective" on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

"Rules for naming workspace objects" on page 6827

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Opening an existing message flow

Open an existing message flow to change or update its contents, or to add or remove nodes.

Before you begin

Before you start

You must have completed the following task:
- "Creating a message flow" on page 1431

About this task

To open an existing message flow:

Procedure

1. Switch to the Broker Application Development perspective. The Broker Development view is populated with all the message flow and message set projects that you have access to. A message flow is contained in a file called `<message_flow_name>.msgflow`.
2. Right-click the message flow that you want to work with, and click **Open**. Alternatively you can double-click the message flow in the Broker Development view.

   The graphical view of the message flow is displayed in the editor view. You can now work with this message flow; for example, you can add or remove nodes, change connections between nodes, or modify node properties.
3. Click **Open ESQL** for any node in the flow that requires ESQL, or double-click the ESQL file (the .esql file) in the Broker Development view to open it, if you want to work with the ESQL file for this message flow.
4. Click **Open Mappings** for any node in the flow that requires mappings, or double-click the mappings file (the `.msgmap` file) in the Broker Development view to open it, if you want to work with the mappings file for this message flow.
5. Click **Open Java** for any JavaCompute node in the flow, or double-click the Java file in the Broker Development view to open it, if you want to work with the Java file for this message flow.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Adding a message flow node" on page 1494
When you have created a message flow, add nodes to define its function.

"Removing a message flow node" on page 1519
When you have created and populated a message flow, you might need to remove a node to change the function of the flow, or to replace it with another more appropriate node. The node can be a built-in node, a user-defined node, or a subflow node.

"The Message Flow editor experiences problems when opening a message flow, and opens in error mode" on page 3435

Related reference:

"Broker Application Development perspective" on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Copying a message flow by using copy

You might find it useful to copy a message flow as a starting point for a new message flow that has similar function. For example, you might want to replace or remove one or two nodes to process messages in a different way.

Before you begin

Before you start:

To complete this task, you must have created a message flow, as described in "Creating a message flow" on page 1431.

About this task

To copy a message flow:

Procedure

1. Switch to the Broker Application Development perspective.
2. Select the message flow (`<message_flow_name>.msgflow`) that you want to copy in the Broker Development view.
   a. Right-click the file and click **Copy** from the menu.
3. Right-click the broker schema in the message flow project to which you want to copy the message flow and click **Paste**. You can copy the message flow in the same broker schema in the same message flow, or to a different broker schema in the same message flow project, or to a broker schema in a different message flow project.

   When you copy a message flow, the associated files (ESQL and mapping, if present) are not automatically copied to the same target message flow project. If you want these files copied as well, you must do this explicitly following this procedure.

   You might also need to update nodes that have associated ESQL or mappings, to ensure that modules are unique.

   For example, if you have created a message flow Test1 that contains a single Compute node, and you copy message flow Test1 and its associated `.esql` file to the same broker schema in the same message flow project (and give the new copy a different name, for example Test2), two modules named Test1_Compute now exist in the single schema. One is in `Test1.esql`, the second in `Test2.esql`.

   This duplication is not supported, and an error message is written to the Problems view when you have completed the copy action. You must rename the associated ESQL modules in the `.esql` file and update the matching node properties to ensure that every module in a broker schema is unique.

Results

The message flow is copied with all property settings intact. If you intend to use this copy of the message flow for another purpose, for example to retrieve messages from a different input queue, you might have to modify its properties.

You can also use **File** > **Save As** to copy a message flow. This task is described in "Saving a message flow" on page 1448.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an

input message is received.

Related tasks:

Chapter 9, “Developing message flow applications,” on page 1019
Develop message flows to process your business messages and data.

“Opening an existing message flow” on page 1433
Open an existing message flow to change or update its contents, or to add or remove nodes.

“Defining message flow content” on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

Related reference:

“Broker Application Development perspective” on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

“Built-in nodes” on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Analyzing planned changes to message flows

Use impact analysis to analyze the effect of renaming or moving message flows, including subflows.

Before you begin

Before you start:

You must have completed the following tasks:

- “Creating a message flow project” on page 1425.
- “Creating a message flow” on page 1431.
- Ensure that you have enabled indexing, by following the instructions in “Enabling and disabling indexing” on page 1454.

About this task

This information covers the following tasks:

- “Renaming message flows” on page 1437
- “Moving the message flow file” on page 1437

If a node in the message flow contains either an ESQL path expression, or an XPath expression, it might not be possible to unambiguously determine which structure in the message set the expression refers to. Ambiguity occurs when multiple constructs have the same name, and the parent of the construct cannot be determined. For example, consider a message definition that has the following constructs:

Global element declaration with name G1

Global element declaration with name G2

Global element declaration with name G3

G2 contains an element reference to global element G1

G3 contains a local element declaration with name G1

The message flow might contain the following ESQL path:

```
InputRoot.MRM.ns1:G1
```

Or the following XPath path:

```
$InputRoot/MRM/ns1:G1
```

The construct G1 cannot be unambiguously determined. It can refer to G2/G1 or G3/G1. In this case, impact analysis reports both G2/G1 and G3/G1 as potential impacts. The list of secondary impacts might contain references to elements that are not in fact affected by the primary change.

Renaming message flows:
About this task

Use this procedure to analyze the impact of renaming a message flow.

Procedure

1. In the Broker Development view, right-click the flow or subflow that you want to rename, then click **Impact Analysis** > **Rename**.
2. In the Impact Analysis - Rename Artifact window, type the new name of the object, then click **Analyze Impact**.

   The Rename Artifact dialog box shows the results of impact analysis, listing primary and secondary impacts.

   You can view the results of impact analysis in the Impact Analysis dialog box, or the Impact Analysis view of the WebSphere Message Broker Toolkit. For more information about how to view information about selected resources, mark changes as complete, copy results to an external application, and view previous results, see "Impact Analysis view" on page 6801.

Moving the message flow file:
About this task

You can analyze the effects of moving a message flow file.

Procedure

1. Right-click the message flow file, then click **Impact Analysis** > **Move**.
2. Select the new container for the object, then click **Analyze Impact**.

   The Impact Analysis - Move Artifact dialog box shows the results of impact analysis, listing primary and secondary impacts.

   You can view the results of impact analysis in the Impact Analysis dialog box, or the Impact Analysis view of the WebSphere Message Broker Toolkit. For more information about how to view information about selected resources, mark changes as complete, copy results to an external application, and view previous results, see "Impact Analysis view" on page 6801.

:

Related concepts:

"Impact analysis: analyzing the effects of planned changes to your applications" on page 1150
When developing an application, you might want to change the names of some artifacts, or move them. Impact analysis reports the artifacts that are likely to be affected by a particular change.

Related tasks:

"Enabling and disabling indexing" on page 1454
Enable indexing to support impact analysis.

"Analyzing planned changes to message model objects" on page 2897
Use impact analysis to analyze the effect of renaming message model objects.

"Analyzing planned changes to ESQL objects" on page 2403
Use impact analysis to analyze the effect of renaming or moving ESQL objects, or moving or changing the broker schema of an ESQL file.

"Analyzing the impact of changes to message maps" on page 2234
Use impact analysis to analyze the effect of renaming or moving message maps.

"Analyzing planned changes to message set resources" on page 1165
Use impact analysis to analyze the effect of renaming message definition or deployable WSDL files.

Related reference:

"Impact analysis: reference" on page 4174
Some artifacts are excluded from secondary analysis.

"Impact Analysis view" on page 6801
The Impact Analysis view shows information about selected resources, which changes have been marked as complete, and previous results. You can also use this view to copy results to the clipboard.

Renaming a message flow

You can rename a message flow. You might want to rename a flow if you have modified it to provide a different function, and you want the name of the flow to reflect this new function.

Before you begin

Before you start

This task assumes that you have created a message flow. For more information, see "Creating a message flow" on page 1431.

About this task

You can use impact analysis to analyze the effect of renaming message flows. For more information about impact analysis, see "Analyzing planned changes to message flows" on page 1436.

To rename a message flow, complete the following steps.

Procedure

1. To open the Rename Resource dialog box, select the flow that you want to rename (*message_flow_name*.msgflow), then click **File** > **Rename**.
2. Enter a new name for the file.
3. Click **OK** to rename the file, or **Cancel** to cancel the request. If you click **OK**, the file is renamed.

   After you have renamed the flow, any references that you have to this flow (for example, if it is embedded in another message flow) are no longer valid.
4. You must open the affected message flows and correct the references if you are not sure where you have embedded this message flow.
   a. Click **File** > **Save All** The save action saves and validates all resources. Unresolved references are displayed in the Problems view; click each error listed to open the message flow that makes a non-valid reference in the editor view.

b. Right-click the subflow icon and click **Locate Subflow**. The Locate Subflow dialog box opens, listing the available message flow projects.
c. Expand the list and explore the resources available to locate the required subflow.
d. Select the correct subflow, and click **OK**. All references in the current message flow are updated for you. All related errors are removed from the Problems view.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Opening an existing message flow" on page 1433
Open an existing message flow to change or update its contents, or to add or remove nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

Related reference:

"Broker Application Development perspective" on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Moving a message flow

You can move a message flow from one broker schema to another in the same project or to a broker schema in another project. You might want to move a flow, for example, if you are reorganizing the resources in your projects.

Before you begin

Before you start:

Complete the following task:

- "Creating a message flow" on page 1431

You can use impact analysis to analyze the effect of moving message flows and subflows; for more information, see "Analyzing planned changes to message flows" on page 1436.

About this task

To move a message flow:

Procedure

1. Drag the message flow that you want to move from its current location to a broker schema in the same or another message flow project.

   If the target location that you have chosen is not valid, a black no-entry icon appears over the target, an error dialog box is displayed, and the message flow

is not moved. You can move a message flow to another schema in the same project or to a schema in another message flow project. If you have created an empty broker schema for this purpose, it might not be visible in the Broker Development view if category mode is selected. To see an empty schema in the Broker Development view, click Hide Categories  .

Alternatively, you can use the following method:

a. Right-click the message flow that you want to move (*message_flow_name*.msgflow) in the Broker Development view and click **Move**, or **File** > **Move**. The Move dialog box is displayed, and contains a list of all valid projects to which you can move this message flow.

b. Select the project and the broker schema in the project to which you want to move the message flow. You can move a message flow to another schema in the same project or to a schema in another message flow project.

c. Click **OK** to complete the move, or **Cancel** to cancel the move. If you click **OK**, the message flow is moved to its new location.

2. Check the Problems view for errors or warnings that are generated by the move. Errors are indicated by the error icon , warnings are indicated by the warning icon . The errors in this view include those that are caused by broker references. When the move is complete, all references to this message flow (for example, if this message flow is a reusable error routine that you have embedded in another message flow) are checked.

   If you have moved the message flow in the same broker schema (in the same or another project), all references are still valid. However, if you move the message flow from one broker schema to another (in the same or a different project), the references are broken because the resources are linked by a fully-qualified name of which the broker schema is a part. Information about broken references is written to the Problems view; for example, `Linked or nested flow mflow1 cannot be located`.

3. Double-click each error or warning to correct it. The message flow that contains the error is opened in the editor view and the node in error is highlighted.

Results

When you move a message flow, the associated files (for example, all ESQL or mapping files) are not automatically moved to the target broker schema. If you want to move these files as well, you must do so explicitly by following the procedure in this topic.

Related concepts:

"By name linking" on page 43
You identify objects using a combination of a *namespace* and a name, referred to as a *fully qualified name*. The use of fully qualified names, called *by name linking*, makes it easy to identify and locate objects, and to correct broken references.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Opening an existing message flow" on page 1433
Open an existing message flow to change or update its contents, or to add or

remove nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

Related reference:

"Broker Application Development perspective" on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Deleting a message flow

Delete message flows from your message flow project when you no longer need them.

Before you begin

Deleting a message flow in the WebSphere Message Broker Toolkit deletes the project and its resources. If you are using a shared repository, the repository might retain a copy of a deleted resource.

Before you start:

This task assumes that you have created a message flow. For more information, see "Creating a message flow" on page 1431.

Before you delete a message flow, you can list resources that refer to that flow and would therefore be affected by the deletion; for more information, see "Showing resource references" on page 1447.

About this task

To delete a message flow, complete the following steps.

Procedure

1. Select the message flow that you want to delete, then click **Edit** > **Delete**. A confirmation dialog box is displayed.
2. Click **Yes** to delete the message flow, or **No** to cancel the delete request. When you click **Yes**, the requested objects are deleted.

   If you maintain resources in a shared repository, a copy is retained in that repository. You can follow the instructions provided by the repository supplier to retrieve the resource if required.

   If you are using the local file system or a shared file system to store your resources, no copy of the resource is retained. Be careful to select the correct resource when you complete this task.
3. Check the Problems view to see if errors have been returned to the delete request. Errors are generated if you delete a message flow that is embedded in another flow, because the reference is no longer valid.
   a. Click the error in the Problems view. The flow that now has a non-valid reference is displayed.

b. Either remove the node that represents the deleted flow from the parent flow, or create another message flow with the same name to provide whatever processing is required.

Results

When you delete the message flow, the files that are associated with the flow (the ESQL and mapping files, if present) are not deleted by this action. If you want to delete these files also, you must do so explicitly.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Deleting a broker schema

You can delete a broker schema that you have created in a message flow project if you no longer need it.

Before you begin

Before you start:

This topic assumes that you have already created a broker schema, as described in "Creating a broker schema" on page 1429. For more information about schemas, see "Broker schemas" on page 1036.

About this task

To delete a broker schema, complete the following steps.

Procedure

1. If the schema contains resources, delete them. To delete the schema, it must be empty. The schema might not be visible in the Broker Development view when it is empty. To show the new schema in the Broker Development view, click **Hide Categories**  on the Broker Development view toolbar.
2. Select the broker schema, then click **Edit** > **Delete**. A confirmation dialog box is displayed.
3. Click **Yes** to delete the broker schema directory or **No** to cancel the delete request. When you click **Yes**, the requested objects are deleted.

   If you maintain resources in a shared repository, a copy is retained in that repository. You can follow the instructions provided by the repository supplier to retrieve the resource, if required.

If you are using the local file system or a shared file system to store your resources, no copy of the resource is retained. Be careful to select the correct resource when you complete this task.

Results

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Broker schemas" on page 1036
A broker schema is a symbol space that defines the scope of uniqueness of the names of resources defined within it. The resources include message flows and other optional resources such as ESQL files, Java files, and mapping files.

Related tasks:

"Creating a message flow project" on page 1425
A message flow project is a container for message flows; you must create a project before you can create a message flow.

Related reference:

"Broker Application Development perspective" on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

Version and keyword information for deployable objects

Use the Broker Archive editor to view the version and keyword information of deployable objects.

You can display properties of deployed objects, and can modify associated comments:

- "Object version in the Broker Archive editor"
- "Version, deployment time, and keywords of deployed objects"
- "Path and Comment columns" on page 1444

Object version in the Broker Archive editor

The Version column on the Manage page of the Broker Archive editor displays the version tag for the following objects that have a defined version.

- `.dictionary` files
- `.cmf` files
- Embedded JAR files with a version defined in a `META-INF/keywords.txt` file

You cannot edit the Version column.

You can use the **mqsireadbar** command to list the keywords that are defined for each deployable file within a deployable archive file. For more information, see "**mqsireadbar** command" on page 3697.

Version, deployment time, and keywords of deployed objects

In the WebSphere Message Broker Explorer, the Properties QuickView displays the following properties for all deployed objects:

- Version
- Deployment time

- All defined keywords

For example, you deploy a message flow with the following literal strings:

- `$MQSI_VERSION=v1.0 MQSI$`
- `$MQSI Author=fred MQSI$`
- `$MQSI Subflow 1 Version=v1.3.2 MQSI$`

The Properties view displays these properties:

Property	Description
Deployment Time	Date and time of deployment
Modification Time	Date and time of modification. **Note:** This property has no concept of time zone, therefore it is only meaningful if you know in which time zone it was last modified.
Version	`v1.0`
Author	`fred`
Subflow 1 Version	`v1.3.2`

If the keyword information is not available, a message is displayed in the Properties view to indicate the reason; for example, if keyword resolution has not been enabled at deployment time, the Properties view displays the message `Deployed with keyword search disabled`.

Path and Comment columns

If you add source files, the Path column on the Manage tab is populated automatically.

To add a comment, double-click the Comment column and type the text that you require.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Adding keywords to ESQL files" on page 2486
You can add keywords to ESQL files to contain information that you want to associate with a message flow.

"Keywords in subflows" on page 1447
You can embed keywords in each subflow that you use in a message flow.

Related tasks:

"Adding keywords to JAR files" on page 2660
If a BAR file contains JAR files, you can associate keywords with the JAR files.

Related reference:

"Description properties for a message flow" on page 4016
The description properties for a message flow include the `Version`, `Short Description` and `Long Description`. To view and edit the properties of a message flow click **Flow** > **Properties**.

"**mqsireadbar** command" on page 3697
Use the **mqsireadbar** command to read a deployable BAR file and identify its

defined keywords.

"Broker Application Development perspective" on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

"Adding keywords to XSL style sheets" on page 4975
Embedded keywords in an XSL style sheet; their location is not restricted. You can also add a keyword as an XML comment.

"Broker Archive editor" on page 6794
Use the Broker Archive editor to create and manage broker archive (BAR) files.

"Guidance for defining keywords" on page 4017
You can add extra information to an object in the form of one or more keywords.

Message flow version and keywords:

When you are developing a message flow, you can define the version of the message flow as well as other key information that you want to be associated with it.

After the message flow has been deployed, you can view the properties of the message flow in the WebSphere Message Broker Toolkit. These properties include the deployment and modification dates and times (the default information that is displayed) as well as any additional version or keyword information that you have set.

You can define information to give details of the message flow that has been deployed; therefore, you can check that it is the message flow that you expect.

Version

You can set the version of the message flow in the `Version` property.

You can also define a default message flow version in the `Default version tag` of the message flow preferences. All new message flows that are created after this value has been set have this default applied to the `Version` property at the message flow level.

Keywords

Keywords are extracted from the compiled message flow (the `.cmf` file) rather than the message flow source (the `.msgflow` file). Not all the source properties are added to the compiled file. Therefore, add message flow keywords in only these places:

- The `label` property of a Passthrough node
- ESQL comments or string literals
- The `Long Description` property of the message flow

Any keywords that you define must follow certain rules to ensure that the information can be parsed. The following example shows some values that you might want to define in the `Long Description` property:

```
$MQSI Author=John Smith MQSI$
$MQSI Subflow 1 Version=v1.3.2 MQSI$
```

The following table contains the information that the WebSphere Message Broker Toolkit shows.

Message flow name	Example
Deployment Time	28-Aug-2004 15:04
Modification Time	28-Aug-2004 14:27
Version	v1.0
Author	John Smith
Subflow 1 Version	v1.3.2

In this display, the version information has also been defined using the `Version` property of the object. If the version information has not been defined using the property, it is omitted from this display.

If message flows contain subflows, you can embed keywords in each subflow.

Restrictions within keywords

Do not use the following characters within keywords because they cause unpredictable behavior:

```
^ $ . | \ < > ? + * = & [ ] ( )
```

You can use these characters in the values that are associated with keywords; for example:

- `$MQSI RCSVER=$id$ MQSI$` is acceptable
- `$MQSI $name=Fred MQSI$` is not acceptable

Related concepts:

"Version and keyword information for deployable objects" on page 1443
Use the Broker Archive editor to view the version and keyword information of deployable objects.

"Subflows" on page 1030
You can include subflows in your message flows in the same way as you include built-in or user-defined nodes. You can also connect subflows to other nodes in the same way.

"Message set version and keywords" on page 1169
When you develop a message set, you can define the version of the message set, and other key information that you want to be associated with it.

"Adding keywords to ESQL files" on page 2486
You can add keywords to ESQL files to contain information that you want to associate with a message flow.

"Keywords in subflows" on page 1447
You can embed keywords in each subflow that you use in a message flow.

Related tasks:

"Adding keywords to JAR files" on page 2660
If a BAR file contains JAR files, you can associate keywords with the JAR files.

Related reference:

"Guidance for defining keywords" on page 4017
You can add extra information to an object in the form of one or more keywords.

"Description properties for a message flow" on page 4016
The description properties for a message flow include the `Version`, `Short Description` and `Long Description`. To view and edit the properties of a message flow click **Flow** > **Properties**.

"Message flow preferences" on page 4016
You can change preferences that determine properties of message flows when you create them.

"Passthrough node" on page 4628
Use the Passthrough node to enable version control of a subflow at run time.

"Adding keywords to XSL style sheets" on page 4975
Embedded keywords in an XSL style sheet; their location is not restricted. You can also add a keyword as an XML comment.

Keywords in subflows:

You can embed keywords in each subflow that you use in a message flow.

You must use a different keyword in each instance of a subflow, because only the first recorded instance of each keyword within the message flow `.cmf` file is available to applications that use the Administration API (also known as the CMP API), which include the WebSphere Message Broker Toolkit.

The order that subflows appear in the `.cmf` file is not guaranteed.

Related concepts:

"Message flow version and keywords" on page 1445
When you are developing a message flow, you can define the version of the message flow as well as other key information that you want to be associated with it.

"Version and keyword information for deployable objects" on page 1443
Use the Broker Archive editor to view the version and keyword information of deployable objects.

Related reference:

"Guidance for defining keywords" on page 4017
You can add extra information to an object in the form of one or more keywords.

Showing resource references

If you are considering changing a resource, you can see a list of other resources that would be affected by that change.

Before you begin

Before you start:

Before you can search for affected resources, you must enable indexing by following the instructions in "Enabling and disabling indexing" on page 1454.

About this task

To search for affected resources, complete the following steps.

Procedure

1. Right-click a resource in the Broker Development view and click **Show all references**.

   The search results view shows all affected references in a tree format, including the location of that resource, such as a node property or the line number of an ESQL file.

2. Optional: To open a resource in the appropriate editor, double-click a resource. Alternatively, to choose the editor in which to open a resource, right-click the resource and click **Open with**.

What to do next

If you are considering renaming or moving a resource, and you know the new name or location, you can run impact analysis; for more information, see "Impact analysis: analyzing the effects of planned changes to your applications" on page 1150.

Related concepts:

"Resources" on page 36
The projects, folders, and files that you work with in the WebSphere Message Broker Toolkit workspace are called *resources*. By default, these resources are stored with their metadata in the workspace directory in your local file system. The workspace directory is created the first time that you start the WebSphere Message Broker Toolkit.

"Impact analysis: analyzing the effects of planned changes to your applications" on page 1150
When developing an application, you might want to change the names of some artifacts, or move them. Impact analysis reports the artifacts that are likely to be affected by a particular change.

Related tasks:

"Enabling and disabling indexing" on page 1454
Enable indexing to support impact analysis.

Related reference:

"Impact analysis: reference" on page 4174
Some artifacts are excluded from secondary analysis.

Saving a message flow

You might want to save your message flow when you close the WebSphere Message Broker Toolkit, work with another resource, or validate the contents of the flow.

Before you begin

Before you start:

This topic assumes that you have created or opened a message flow, as described in the following topics:

- "Creating a message flow" on page 1431
- "Opening an existing message flow" on page 1433

About this task

To save a message flow, complete the following steps.

Procedure

1. To save the flow without closing it, click **File** > **Save**. You can also save everything by clicking **File** > **Save All**.

   The message flow is saved and the message flow validator validates the contents of the flow. The validator reports all errors that it finds in the Problems view. The flow remains open in the editor view. For example, if you save a message flow and have not set a mandatory property, an error message

appears in the Problems view and the editor marks the node with the error icon  . The message flow in the Broker Development view is also marked with the error icon. This error can occur if you have not edited the properties of an MQInput node to define the queue from which the input node retrieves its input messages.

You might also get warnings when you save a message flow; warnings are also shown in the Problems view. Warnings indicate that, although the configuration of the message flow does not contain an explicit error, the configuration might result in unexpected results when the flow completes. For example, if you have included an input node in your message flow that you have not connected to another node, you get a warning.

2. If you save a message flow that includes a subflow, and the embedded subflow is no longer available, three error messages are added to the Problems view. These errors indicate that the input and output terminals and the embedded subflow itself cannot be located. This error can occur if the embedded subflow has been moved or renamed.

   To resolve this situation, right-click the subflow node in error and click **Locate Subflow**. The Locate Subflow dialog box is displayed, listing the available message flow projects. Expand the list and explore the resources available to locate the required subflow. Select the correct subflow and click **OK**. All references in the current message flow are updated and the errors are removed from the Problems view.

3. You can also save a message flow when you close it. When you click **File** > **Close**, you are asked if you want to save the flow. Click **Yes** to save and close. Validation occurs and all errors and warnings are written to the Problems view.

What to do next

For information about using the **File** > **Save As** option to take a copy of the current flow, see "Copying a message flow by using the save option" on page 1450.

For information about handling errors that occur when you save, see "Correcting errors from saving a message flow" on page 1451.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Packaging and deployment overview" on page 3210
Deployment is the process of transferring data to an execution group on a broker so that it can take effect in the broker. Message flows and associated resources are packaged in broker archive (BAR) files for deployment.

Related tasks:

"Copying a message flow by using the save option" on page 1450
You can copy a message flow by using the **File** > **Save As** option.

"Correcting errors from saving a message flow" on page 1451
Correct the errors that are reported when you save a message flow.

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Opening an existing message flow" on page 1433
Open an existing message flow to change or update its contents, or to add or remove nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

Related reference:

"Broker Application Development perspective" on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Copying a message flow by using the save option:

You can copy a message flow by using the **File** > **Save As** option.

Before you begin

Before you start:

This topic assumes that you have created or opened a message flow, as described in the following topics:
- "Creating a message flow" on page 1431
- "Opening an existing message flow" on page 1433

Procedure

1. Click **File** > **Save As**.
2. Specify the message flow project in which you want to save a copy of the message flow. By default, the current project is selected; you can accept this name, or choose another name from the list.
3. Specify the name for the new copy of the flow. To save this message flow in the same project, you must either give it another name, or confirm that you want to overwrite the current copy.

   To save the message flow in another project, the project must already exist. You can save the flow with the same name or another name in another project.
4. Click **OK**.

   The message flow is saved and its contents are validated. The editor provides a report of errors and warnings in the Problems view.

What to do next

For information about handling errors that occur when you save a message flow, see "Correcting errors from saving a message flow" on page 1451.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Packaging and deployment overview" on page 3210
Deployment is the process of transferring data to an execution group on a broker so that it can take effect in the broker. Message flows and associated resources are packaged in broker archive (BAR) files for deployment.

Related tasks:

"Saving a message flow" on page 1448
You might want to save your message flow when you close the WebSphere Message Broker Toolkit, work with another resource, or validate the contents of the flow.

"Correcting errors from saving a message flow"
Correct the errors that are reported when you save a message flow.

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Opening an existing message flow" on page 1433
Open an existing message flow to change or update its contents, or to add or remove nodes.

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

Related reference:

"Broker Application Development perspective" on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Correcting errors from saving a message flow:

Correct the errors that are reported when you save a message flow.

About this task

When you save a message flow, as described in "Saving a message flow" on page 1448, the flow is validated. Errors and warnings are shown in the Problems view. The following steps describe how to correct errors that occur when you save a flow.

Procedure

1. Examine the list of errors and warnings in the Problems view.
2. Double-click each entry in turn.

   The message flow is opened in the editor view (if it is not already open), and the editor selects the node in which the error was detected. If the error has been generated because you have not set a mandatory property, the editor also opens the Properties view, or dialog box, for that node.

   If you have included a user-defined node in your message flow, and have defined one or more of its properties as configurable, you might get a warning about a custom property editor. If you define a property as a configurable property, and you have specified that it uses a custom property editor, the Broker Archive editor cannot handle the custom property editor, and handles the property as if it is type string. This action restricts your ability change this property at deployment time.
3. Correct the error that is indicated by the message. For example, provide a value for a mandatory property.

4. When you have corrected all the errors, save the message flow again.

   The editor validates all the resources that you have changed, and removes the corresponding graphical indication from the nodes that you have modified successfully. Corrected errors are also removed from the Problems view.

Results

You do not have to correct every error to save your work. The editor saves your resources even if it detects errors or warnings, so that you can continue to work with them at a later date. However, you cannot deploy a resource that has a validation error. You must correct every error before you deploy a resource. Warnings do not prevent successful deployment.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Packaging and deployment overview" on page 3210
Deployment is the process of transferring data to an execution group on a broker so that it can take effect in the broker. Message flows and associated resources are packaged in broker archive (BAR) files for deployment.

Related tasks:

"Saving a message flow" on page 1448
You might want to save your message flow when you close the WebSphere Message Broker Toolkit, work with another resource, or validate the contents of the flow.

"Copying a message flow by using the save option" on page 1450
You can copy a message flow by using the **File** > **Save As** option.

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Opening an existing message flow" on page 1433
Open an existing message flow to change or update its contents, or to add or remove nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

Related reference:

"Broker Application Development perspective" on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Importing and exporting resources in a Project Interchange file

You can import resources to, or export resources from, WebSphere Message Broker by using a Project Interchange file.

About this task

You can work with resources created by other products by importing them into WebSphere Message Broker in a Project Interchange file. For example, you can define a WebSphere Message Broker message set from WebSphere Process Server resource files. In this case, you would export the resources from WebSphere Process Server in the Project Interchange file, then import the file into WebSphere Message Broker.

Similarly, you can share pattern instance projects between WebSphere Message Broker Toolkit workspaces. You can export the pattern instance project into a Project Interchange file, then import the pattern instance project into another workspace.

Procedure

To import or export a Project Interchange file, complete the following steps.

- To import a Project Interchange file:
  1. Click **File** > **Import**. The Import wizard opens.
  2. Expand **Other**, click **Project Interchange**, then click **Next**.
  3. Specify the location of the Project Interchange file that you want to import.
  4. Specify the location to which you want to import the resources. By default, your current workspace is selected.
  5. Select the projects that you want to import, then click **Finish**.
- To export a Project Interchange file:
  1. Click **File** > **Export**. The Export wizard opens.
  2. Expand **Other**, click **Project Interchange**, then click **Next**.
  3. Select the projects that you want to export, and specify the location for the compressed file that is created. You can save the file to a folder on your file system, or to a disk drive.
  4. Click **Finish**. The Project Interchange file is created in the specified location.

Related concepts:

"Interoperability with WebSphere Process Server" on page 2097
WebSphere Process Server is a business integration server that supports solutions based on service-oriented architecture (SOA). It uses Service Component Architecture (SCA) to present all business transactions in a service-oriented way in its runtime environment. WebSphere Message Broker can accept requests from WebSphere Process Server, and can also call service components on WebSphere Process Server.

Related tasks:

"Creating a working set and focus on a pattern instance" on page 1319
Creating a working set for an imported project interchange file that contains a pattern instance project and its associated projects.

"Importing an existing configuration" on page 1329
You can import a pattern configuration XML file from the workspace or file system and populate your current pattern instance with its values.

"Resolving problems when developing message flows with WebSphere Adapters nodes" on page 3428
Advice for dealing with common problems that can arise when you develop message flows that contain WebSphere Adapters nodes.

Enabling and disabling indexing

Enable indexing to support impact analysis.

About this task

Indexing must be enabled to support impact analysis. The act of indexing determines what definitions and relationships exist in WebSphere Message Broker artifacts. These definitions and relationships are used to perform impact analysis. By default, indexing is disabled, for performance reasons.

Procedure

1. Click **Window** > **Preferences**, then click **Broker Development** > **Impact Analysis** in the left pane.
2. Optional: To enable indexing, select **Enable the indexing of Message Broker artifacts**.
   - When indexing is enabled, a full build of all artifacts in the workspace occurs once. Indexing of artifacts occurs on further incremental and full builds. Only artifacts participating in the builds are indexed.
   - Indexes for these artifacts are stored in memory, and written to disk when the WebSphere Message Broker Toolkit is shut down.
3. Optional: To disable indexing, clear the **Enable the indexing of Message Broker artifacts** check box.
   - Indexes of artifacts are removed from memory.
   - When the WebSphere Message Broker Toolkit is shut down, an index of items in memory is written to disk. This index is smaller, because the indexes of artifacts have been removed.
   - Artifacts are not indexed during builds, resulting in better performance.

Related concepts:

"Impact analysis: analyzing the effects of planned changes to your applications" on page 1150
When developing an application, you might want to change the names of some artifacts, or move them. Impact analysis reports the artifacts that are likely to be affected by a particular change.

Related tasks:

"Analyzing planned changes to message model objects" on page 2897
Use impact analysis to analyze the effect of renaming message model objects.

"Analyzing planned changes to ESQL objects" on page 2403
Use impact analysis to analyze the effect of renaming or moving ESQL objects, or moving or changing the broker schema of an ESQL file.

"Analyzing the impact of changes to message maps" on page 2234
Use impact analysis to analyze the effect of renaming or moving message maps.

"Analyzing planned changes to message flows" on page 1436
Use impact analysis to analyze the effect of renaming or moving message flows, including subflows.

"Analyzing planned changes to message set resources" on page 1165
Use impact analysis to analyze the effect of renaming message definition or deployable WSDL files.

Related reference:

"Impact analysis: reference" on page 4174
Some artifacts are excluded from secondary analysis.

"Impact Analysis view" on page 6801
The Impact Analysis view shows information about selected resources, which changes have been marked as complete, and previous results. You can also use this view to copy results to the clipboard.

Designing a message flow

A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

Before you begin

Before you start:

Read the following concept topic: "Message flow nodes" on page 1024.

About this task

When you design a message flow, consider the following questions and options:

- The mode that your broker is working in can affect the types of node that you can use and the number of message flows you can deploy. For more information, see "Restrictions that apply in each operation mode" on page 3657.
- Which nodes provide the function that you require. In many cases, you can choose between several nodes that provide a suitable function. You might have to consider other factors listed here to determine which node is best for your overall needs. You can include built-in nodes, user-defined nodes, and subflow nodes. For more information, see "Deciding which nodes to use" on page 1457.
- Whether it is appropriate to include more than one input node. For more information, see "Using more than one input node" on page 1473.
- How to specify the characteristics of the input message. For more information, see "Defining input message characteristics" on page 1475.
- Whether to determine the path that a message follows through the message flow, based on the content or the characteristics of the message. Several nodes provide checks or examination of the message, and have output terminals that can be connected to direct certain messages to different nodes. For more information, see "Using nodes for decision making" on page 2209.
- Whether you can use subflows that provide a well-defined subset of processing. You might be able to reuse subflows that were created for another project (for example, an error processing routine), or you might create a subflow in your current project, and reuse it in several places in the same message flow. For more information, see "Subflows" on page 1030.
- What response times your applications expect from the message flow. This factor is influenced by several aspects of how you configure your nodes and the message flow. For more information, see "Optimizing message flow response times" on page 3264.
- Whether your message flow processing makes demands on system resources such as stack size. For more information, see "System resources for message flow development" on page 3267.
- Whether you can use the destination list in the local environment that is associated with the message to determine the processing in the message flow (for example, by using RouteToLabel and Label nodes), or the target for the

output messages (for example, by setting the `Destination Mode` property of the MQOutput node to Destination List). For more information, see “Creating destination lists” on page 1477.

- Whether to use WebSphere MQ cluster queues. For more information, see “Using WebSphere MQ cluster queues for input and output” on page 1544.
- Whether to use WebSphere MQ shared queues on z/OS . For more information, see “Using WebSphere MQ shared queues for input and output (z/OS)” on page 1546.
- Whether to validate input messages that are received by the input node, or output messages that are generated by the Compute node, or both. For more information, see “Validating messages” on page 1478.
- Whether to view or record message structure in Trace node output. For more information, see “Viewing the logical message tree in trace output” on page 1481.
- Whether your message flows access data in databases. You must configure brokers, databases, and database connections to enable this function, as described in “Working with databases” on page 2109. You must also configure your message flows; see “Accessing databases from message flows” on page 2112.

  If you include nodes that use ESQL, for information about how to code the appropriate statements, see “Accessing databases from ESQL” on page 2115. If you want to access databases from Java nodes by using JDBC, see “Interacting with databases by using the JavaCompute node” on page 2661 or “Extending the capability of a Java message processing or output node” on page 3069.

  You can also access databases through the Broker Application Development perspective in the WebSphere Message Broker Toolkit; see “Adding database definitions to the WebSphere Message Broker Toolkit” on page 2278.
- Whether your message flows access data in files. By using the FileInput and FileOutput nodes, your message flows can read messages from files and write messages to files in the local file system, or on a network file system that appears local to the broker. For more information, see “Connecting client applications” on page 1537.
- Whether your messages must be handled in a transaction. You can set the properties of some built-in nodes to control how transactions are managed, and how messages are processed in a transaction. For more information, see “Configuring transactionality for message flows” on page 1290.

  If you want to include JMSInput and JMSOutput nodes in your message flow transactions, you must consider the additional information in “Configuring JMS and SOAP nodes to support global transactions” on page 1716.
- Whether you want your messages to go through data conversion. For information about the available options, see “Configuring message flows for data conversion” on page 1293.
- Whether you want to use the MQGet node. For more information about how messages are processed by the MQGet node, and a description of a request-reply scenario that uses this node, see “Using MQGet nodes” on page 1564.
- How your message flows can benefit from user exits. For more information, see “Exploiting user exits” on page 2985.
- What steps to take to ensure that messages are not lost. For more information, see “Ensuring that messages are not lost” on page 1561.
- How errors are handled in the message flow. You can use the facilities provided by the broker to handle any errors that arise during message flow execution (for example, if the input node fails to retrieve an input message, or if writing to a

database results in an error). However, you might prefer to design your message flow to handle errors in a specific way. For more information, see "Handling errors in message flows" on page 2823.

- Whether you want a systems monitoring tool to be able to query, discover, and set certain user-defined properties at run time. For more information, see "Setting message flow user-defined properties at run time in a CMP application" on page 985.

For a basic introduction to developing message flows, see the IBM Redbooks publication WebSphere Message Broker Basics. (This link works only if you are connected to the Internet.)

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"User-defined nodes" on page 6415
You can define your own nodes to use in WebSphere Message Broker message flows.

Deciding which nodes to use

WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

Before you begin

Before you start:

Read the concept topic, "Message flow nodes" on page 1024.

About this task

WebSphere Message Broker also provides an interface that you can use to define your own nodes, known as user-defined nodes.

The mode that your broker is working in can affect the types of node that you can use; see "Restrictions that apply in each operation mode" on page 3657.

Your decision about which nodes to use depends on the processing that you want to perform on your messages.

Input, output, and request nodes

Input and output nodes define points in the message flow to which client applications send messages (input nodes, such as MQInput), and from which client applications receive messages (output nodes, such as

MQOutput). Client applications interact with these nodes by putting messages to, or getting messages from, the I/O resource that is specified by the node as the source or target of the messages. Although a message flow must include at least one input node, it does not have to include an output or request node.

An input node is different from other nodes, because it controls when the rest of the message flow is triggered to do its processing. The input node is designed to check when there is data for the message flow to process, read that data from the transport or server, and present that data to the rest of the flow for processing. The other nodes do processing, but do not control when the flow gets invoked.

You can also use reply, request, and response nodes to interact with other applications from within a message flow; these types of node are supplied for a subset of protocols only.

- If you are creating a message flow for deployment to a broker, you must include at least one input node to receive messages. The input node that you select depends on the source of the input messages, and where in the flow you want to receive the messages.
- If you want to send the messages that are produced by the message flow to a target application, you can include one or more output nodes. The output node that you select depends on the transport across which the target application expects to receive those messages.
- If you want to make a request, in the middle of your flow, to an external system, and put the result into the message tree, use a request node.

Nodes for manipulating, enhancing, and transforming messages

Most enterprises have applications that have been developed over many years, on different systems, using different programming languages, and different methods of communication. WebSphere Message Broker removes the need for applications to understand these differences by providing the ability to configure message flows that transform messages from one format to another.

For example, personal names are held in many forms in different applications. Family name first or last, with or without middle initials, uppercase or lowercase, are just some of the permutations. Because you can configure your message flow to know the requirements of each application, each message can be transformed to the correct format without modifying the sending or receiving application.

You can work with the content of the message to update it in several ways. Your choices here might depend on whether the message flow must handle predefined (modeled) messages, self-defining messages (for example, XML), or both.

A message flow can completely rebuild a message, convert it from one format to another (for example, changing order of fields, byte order, or language), remove content from the message, or introduce specific data into it. For example, a node can interact with a database to retrieve additional information, or to store a copy of the message (whole or part) in the database for offline processing.

The following examples show the importance of message transformation:

- An order entry application has a part ID in the body of the message, but its associated stock application expects it in the message header. The

message is directed to a message flow that knows the two different formats, and can therefore reformat the information as it is needed.

- A data-entry application creates messages containing stock trade information. Some applications that receive this message need the information as provided, but others need additional information added to the message about the price to earnings (PE) ratio. The stock trade messages are directed to a message flow that passes the message unchanged to some output nodes, but calculates and adds the extra information for the others. The message flow does this calculation by looking up the current stock price in a database, and uses this value and the trade information in the original message to calculate the PE value before passing on the updated message.

You can also create message flows that use these nodes to interact with each other. Although the default operation of one message flow does not influence the operation of another message flow, you can force this action by configuring your message flows to store and retrieve information in an external source, such as a database.

These nodes are supplied to transform messages.

Nodes for making decisions

You can use nodes that determine the order and flow of control in the message flow in various ways to decide how messages are processed by the flow. You can also use nodes (TimeoutControl and TimeoutNotification) that determine the time, and frequency of occurrence, of events in the message flow. Routing is independent of message transformation, although the route that a message takes might determine exactly what transformation is performed on it.

For example, a money transfer application always sends messages to one other application. You might decide that every message with a transfer value of more than $10,000 must also be sent to a second application, to enable all high-value transactions to be recorded.

In another example, a national auto club offers a premier service to specific members for orders above a threshold value. Most orders are routed through the typical channels, but, if the membership number and order value meet certain criteria, the order gets special treatment.

You can also establish a more dynamic routing option by building additional routing information into the message when it is processed. Optional sets of rules are set up to receive messages according to values (destinations) set into the message. You can establish these rules such that a message is processed by one or more of the optional sets of rules, in an order determined by the added message content.

These nodes are provided to decide about the route that a message follows through the message flow.

Nodes for controlling time-sensitive operations

You might want a batch application process to run every day at a specific time, or you might want information to be processed and published at fixed intervals (for example, currency exchange rates are calculated and sent to banks), or you might want to take a specified recovery action if certain transactions are not completed within a defined time. For all these cases, two timeout nodes (TimeoutControl and TimeoutNotification) are provided; see “Nodes for controlling time-sensitive operations” on page 1471.

Miscellaneous nodes

Other nodes exist to do the following tasks:

- Collate requests
- Create message collections
- Control the sequence of messages
- Handle and report errors
- Invoke the message flow security manager

See "Miscellaneous nodes" on page 1472 for details.

Related concepts:

"Nodes for connectivity" on page 1028
WebSphere Message Broker supports direct connections from applications, and can send direct requests to other application endpoints. WebSphere Message Broker can also connect to various subsystems including WebSphere MQ, files, and databases, to read and write existing application data.

"Local environment tree structure" on page 1056
The local environment tree is a part of the logical message tree in which you can store information while the message flow processes the message.

Related tasks:

"Working with databases" on page 2109
Create and configure databases to use with your message flow applications.

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Accessing databases from message flows" on page 2112
Create and configure message flows to access user databases.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Using nodes for decision making" on page 2209
You can use several built-in nodes in different ways to control the path that a message takes through the message flow.

Chapter 11, "Packaging and deploying," on page 3209
Package resources that you create in the WebSphere Message Broker Toolkit, such as message flows, and deploy them to execution groups on brokers.

Related reference:

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Input nodes:

You must include at least one input node in your message flow.

An input node is different from other nodes, because it controls when the rest of the message flow is triggered to do its processing. The input node is designed to check when there is data for the message flow to process, read that data from the transport or server, and present that data to the rest of the flow for processing. The other nodes do processing, but do not control when the flow gets invoked.

DatabaseInput node
Use the DatabaseInput node to respond to events in a database. For example, the broker can keep an external system synchronized with a database by sending updates to the target system whenever data is changed in the database.

EmailInput node
Use the EmailInput node to retrieve an email, with or without attachments, from an email server that supports Post Office Protocol 3 (POP3) or Internet Message Access Protocol (IMAP).

FileInput node
Use a FileInput node if the messages are contents of files.

FTEInput node
Use the FTEInput node to receive files using WebSphere MQ File Transfer Edition.

HTTP input node
Use an HTTPInput node if the messages are sent by a Web services client.

Input node
If you are creating a message flow that you want to embed in another message flow (a subflow) that you will not deploy as a stand-alone message flow, you must include at least one Input node to receive messages into the subflow.

An instance of the Input node represents an In terminal. For example, if you have included one instance of the Input node, the subflow icon shows one In terminal, which you can connect to other nodes in the main flow in the same way that you connect any other node.

To deploy a message flow, it must have at least one input node. If your message flow does not contain an input node, you are prevented from adding it to the broker archive file. The input node can be in the main flow, or in a message flow that is embedded in the main flow.

You can use more than one input node in a message flow. For more information, see "Using more than one input node" on page 1473.

JMSInput node
Use a JMSInput node if the messages are sent by a JMS application.

MQInput node
Use an MQInput node if the messages arrive at the broker on a WebSphere MQ queue, and the node is to be at the start of a message flow.

SCAInput node
Use the SCAInput node with the SCAReply node to process messages from WebSphere Process Server.

SOAP input node
Use the SOAPInput node to process client SOAP messages and to configure the message flow to behave like a SOAP Web Services provider.

TCPIPClientInput or TCPIPServerInput node
Use a TCPIPClientInput node or a TCPIPServerInput node to create a TCP/IP connection when messages are sent through raw TCP/IP sockets.

TCPIPClientReceive or TCPIPServerReceive node
Use a TCPIPClientReceive node or a TCPIPServerReceive node to read the messages that arrive in the message flow through a TCP/IP connection.

User-defined input node
Use a user-defined input node if the message source is a client or application that uses a different protocol or transport.

WebSphere Adapters nodes
Use the WebSphere Adapters nodes to interact with Enterprise Information Systems (EIS) such as SAP, Siebel, and PeopleSoft. The following input nodes are available:

- SAPInput node
- SiebelInput node
- PeopleSoftInput node
- JDEdwardsInput
- TwineballInput node

The WebSphere Adapters input nodes monitor an EIS for a particular event. When that event occurs, business objects are sent to the input node. The node constructs a tree representation of the business objects and propagates it to the Out terminal so that the data can be used by the rest of the message flow.

The WebSphere Adapters request nodes can send and receive business data. They request information from an EIS and propagate the data to the rest of the message flow.

Related tasks:

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Output nodes:

If you want to send the messages that are produced by the message flow to a target application, include one or more output nodes in the flow.

EmailOutput node
Use the EmailOutput node to send an email message to one or more recipients.

FileOutput node
Use a FileOutput node if a file is the target of the output messages.

FTEOutput node
Use the FTEOutput node to write messages to files using the WebSphere MQ File Transfer Edition.

HTTPReply node
Use an HTTPReply node if the messages are in response to a Web services client request.

JMSOutput and JMSReply nodes
Use a JMSOutput node if the messages are for a JMS destination.

The JMSReply node has a similar function to the JMSOutput node, but the JMSReply node sends JMS messages only to the reply destination that is supplied in the JMSReplyTo header field of the JMS message tree. Use the JMSReply node to treat a JMS message that is produced from a message flow as a reply to a JMS input message, and when you have no other routing requirements.

MQOutput and MQReply nodes
Use an MQOutput node if the target application expects to receive messages on a WebSphere MQ queue, or on the WebSphere MQ reply-to queue that is specified in the input message MQMD.

Use an MQReply node if the target application expects to receive messages on the WebSphere MQ reply-to queue that is specified in the input message MQMD.

Output node
If you are creating a message flow that you want to embed in another message flow (a subflow) that you will not deploy as a stand-alone message flow, you must include at least one Output node to propagate messages to subsequent nodes that you connect to the subflow.

An instance of the Output node represents an Out terminal. For example, if you have included two instances of the Output node, the subflow icon shows two Out terminals, which you can connect to other nodes in the main flow in the same way that you connect any other node.

Publication node
Use a Publication node to distribute the messages using the publish/subscribe network for applications that subscribe to the broker across all supported protocols. A Publication node is an output node that uses output destinations that are identified by subscribers whose subscriptions match the characteristics of the current message.

SAPReply node
Use this node with the SAPInput node to respond to an incoming SAP event.

SOAPReply node
Use a SOAPReply node if the target application expects to receive SOAP messages in response to a message sent to the SOAPInput node.

SCAReply node
Use the SCAReply node to send a message from the broker to the originating client in response to a message received by a SCAInput node.

TCPIPClientOutput or TCPIPServerOutput node
Use a TCPIPClientOutput node or a TCPIPServerOutput node if the messages are to be sent to the target application through raw TCP/IP sockets.

User-defined output node
Use a user-defined output node if the target is a client or application that uses a different protocol or transport.

Related tasks:

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Request nodes:

If you want to make a request, in the middle of your flow, to an external system, and put the result into the message tree, use a request node.

MQGet node
Use an MQGet node to retrieve a message from a WebSphere MQ queue, if you want to get the message later in the message flow.

HTTPRequest node
Use an HTTPRequest node if your message flow interacts with a web service after it has started.

FileRead node
Use the FileRead node to read a file from the middle of a message flow. The node can:

- Read the entire contents of the file.
- Read a single record.
- Rename or delete the file without reading any data.

WebSphere Adapters nodes
Use the WebSphere Adapters nodes to interact with Enterprise Information Systems (EIS) such as SAP, Siebel, and PeopleSoft. The following request nodes are available:

- SAPRequest node
- SiebelRequest node
- PeopleSoftRequest node
- JDEdwardsRequest node
- TwineballRequest node

SOAP nodes
Use the SOAP nodes to process client SOAP messages and to configure the message flow to behave like a SOAP web Services provider:

- SOAPRequest
- SOAPAsyncRequest
- SOAPAsyncResponse

WebSphere Service Registry and Repository (WSRR) nodes
Use the WebSphere Service Registry and Repository nodes to retrieve web services information:

- Use the EndpointLookup node to retrieve service endpoint information held in the WebSphere Service Registry and Repository.
- Use the RegistryLookup node to retrieve any type of entity held in the WebSphere Service Registry and Repository.

IMSRequest node
Use the IMSRequest node to send a request to run a transaction on a local or remote IBM Information Management System (IMS) system, and wait for a response. IMS Connect must be configured and running on the IMS system.

CORBARequest node

Use the CORBARequest node to call an external CORBA application over Internet Inter-Orb Protocol (IIOP). You can create a message flow that contains a CORBARequest node, which calls a CORBA server. The message flow uses an IDL file to call methods on a remote CORBA object. You can then give existing CORBA applications a new external interface; for example, a SOAP interface.

CICSRequest node

Use the CICSRequest node to call an external CICS Transaction Server for z/OS application over TCP/IP-based IP InterCommunications (IPIC) protocol. You can create a message flow that contains a CICSRequest node, which calls an application on CICS. By using the CICS support that is provided in WebSphere Message Broker you can deploy CICS applications into a service-oriented architecture (SOA).

Database node

Use the Database node to interact with a database that is identified by the node properties. The Database node handles both predefined and self-defining messages. Use the ESQL editor to code ESQL functions to update database content from the message, insert new information into the database, and delete information from the database, using information in the message. Do not use the ESQL code that you develop for use in a Database node in any other type of node.

This node provides a flexible interface with a wide range of functions. It also has properties that you can use to control the way in which the interaction participates in transactions.

You can control the way in which the database is accessed by this node by specifying user and password information for the data source that you specify in the node properties. Use the **mqsisetdbparms** command to initialize and maintain these values.

You can update only databases from this node; you cannot update message content. If you want to update message content, use the Compute or Mapping node.

DataDelete, DataInsert, DataUpdate nodes

The DataDelete, DataInsert, and DataUpdate nodes are specialized forms of the Database node that provide a single mode of interaction (deletion of one or more rows, insertion of one or more rows, or update of one or more existing rows).

The DataDelete, DataInsert, and DataUpdate nodes handle only predefined messages. Use a mapping editor to develop mappings to perform these functions. Do not use the mappings that you develop for these nodes in any other type of node. You can use these nodes to control the transactional characteristics of the updates that they perform.

You can control the way in which the database is accessed by these nodes by specifying user and password information for the data source that you specify in the node property. Use the **mqsisetdbparms** command to initialize and maintain these values.

You can update only databases from these nodes; you cannot update message content. If you want to update message content, use the Compute or Mapping node.

DatabaseRetrieve node

Use the DatabaseRetrieve node to ensure that information in a message is

up to date. Use the node to modify a message using information from a database. For example, you can add information to a message using a key, such as an account number, that is contained in a message. Use the DatabaseRetrieve node to implement message routing with minimal programming logic. For more advanced routing scenarios, use a Compute node or a JavaCompute node.

Warehouse node

The Warehouse node provides a store interface that you can use to store all or part of the message in a database, for example, for audit reasons. The Warehouse node handles only predefined messages. Use the Mapping editor to develop mappings to perform this action. Do not use the mappings that you develop for a Warehouse node in any other type of node.

You can control the way in which the database is accessed by this node by specifying user and password information for the data source that you specify in the node property. Use the **mqsisetdbparms** command to initialize and maintain these values.

You can update only a database from this node; you cannot update message content. If you want to update message content, use the Compute or Mapping node.

SCARequest, SCAAsyncRequest, and SCAAsyncResponse nodes

- The SCARequest node is used to send a request to WebSphere Process Server. The node is configured using a Broker SCA Definition (`.outsca`) file; depending on the contents of the `.outsca` file, requests are either:
  - Two-way, synchronous; the node sends the request, then blocks until it receives a response, or the timeout period is exceeded.
  - One-way; the node sends a request only.
- The SCAAsyncRequest and SCAAsyncResponse nodes are used to construct a pair of message flows that call a WebSphere Process Server service component asynchronously.

  The SCAAsyncRequest node sends an SCA outbound request to a service component that runs in WebSphere Process Server.

  The SCAAsyncResponse node receives the response from a business process that is running in WebSphere Process Server and to which the previous asynchronous request was made. The SCAAsyncResponse node can be in the same message flow or in a separate message flow.

  Calling a WebSphere Process Server service component asynchronously means that the SCAAsyncRequest node sends a request but does not wait for the associated response to be received, although it might wait for an acknowledgment of the request.

  The nodes are used as a pair, and correlate responses and requests. See "SCAAsyncRequest node" on page 4690 and "SCAAsyncResponse node" on page 4698.

Related tasks:

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Nodes for manipulating, enhancing, and transforming messages:

Optionally, include nodes to change messages.

Compute node

Use the Compute node to:

- Manipulate message content
- Transform the message in some way
- Interact with a database to modify the content of the message or the database and pass on one or more new messages

You can use this node to manipulate predefined and self-defining messages.

Use the ESQL editor to create an ESQL module, specific to this node, that contains the statements that define the actions to perform against the message or database. Do not use the ESQL code that you develop for use in a Compute node in any other type of node.

You can control the way in which the database is accessed by this node by specifying user and password information for the data source that you specify in the node property. Use the **mqsisetdbparms** command to initialize and maintain these values.

If possible, perform your message manipulation requirements in a single Compute node. Fewer, more complex Compute nodes perform better than a larger number of simpler nodes because the broker parses the message on entry to each Compute node.

JavaCompute node

Use the JavaCompute node to:

- Examine an incoming message and, depending on its content, propagate it unchanged to one of the node's output terminals. The node behaves in a similar way to a Filter node, but uses Java instead of ESQL to determine which output terminal to use.
- Change part of an incoming message and propagate the changed message to one of the output terminals.
- Interact with a database through a JDBC type 4 connection to modify the content of the message or the database and pass on one or more new messages
- Create and build a new output message that is independent of the input message.

PHPCompute node

Use the PHPCompute node to route and transform an incoming message, using the PHP scripting language. The node functions in a similar way to the JavaCompute node, but uses PHP instead of Java for message transformation and routing.

Mapping node

Use the Mapping node to create a message from the input message by mapping the content of elements of the output message from elements of the input message, or from database content. You can also extract parts of the message, and optionally change their content, to create an output message that is a partial copy of the message that is received by the node. The Mapping node handles only predefined messages.

You can control the way in which the database is accessed by this node by specifying user and password information for the data source that you specify in the node property. Use the **mqsisetdbparms** command to initialize and maintain these values.

Use the Mapping editor to develop mappings to perform simple manipulations on predefined messages. Do not use the mappings that you develop for use in a Mapping node in any other type of node.

Extract node

The Extract node is deprecated in WebSphere Message Broker . Although message flows that contain an Extract node remain valid in WebSphere Message Broker , where possible, redesign your message flows so that any Extract node is replaced by a Mapping node.

With an Extract node, you can create an output message from specified elements of the input message. You can extract parts of the message, and optionally change their content, to create an output message that is a partial copy of the message received by the node. The Extract node handles only predefined messages.

Use the Mapping editor to develop mappings to perform simple manipulations on predefined messages in the Extract node. Do not use the mappings that you develop for use in an Extract node in any other type of node.

XSLTransform node

Use the XSLTransform node (formerly known as the XMLTransformation node) to transform an input XML message into another format using XSLT style sheets and to set the message domain, message set, message type, and message format for the generated message. It is imperative that the data can be parsed into an XML message. The style sheet, using the rules that are defined in it, can perform the following actions:

- Sort the data
- Select data elements to include or exclude based on some criteria
- Transform the data into another format

The Xalan-Java transformation engine (Apache Xalan-java XSLT processor) is used as the underlying transformation engine. For more information about XML Transformations, the W3C specification of the syntax, and semantics of the XSL Transformations language for transforming XML documents into other XML documents, see W3C XSL Transformations.

You can deploy style sheets and XML files to broker execution groups, to help with style sheet and XML file maintenance.

JMSMQTransform node

Use the JMSMQTransform node to transform a message with a JMS message tree into a message that has a tree structure that is compatible with the format of messages that are produced by the WebSphere MQ JMS provider.

The JMSMQTransform node can be used to send messages to existing message flows and to interoperate with WebSphere MQ JMS and WebSphere MQ Publish/Subscribe.

MQJMSTransform node

Use the MQJMSTransform node to receive messages that have a

WebSphere MQ JMS provider message tree format, and transform them into a format that is compatible with messages that are to be sent to JMS destinations.

You can use the MQJMSTransform node to send messages to existing message flows and to interoperate with WebSphere MQ JMS and WebSphere MQ Publish/Subscribe.

SOAPEnvelope and SOAPExtract nodes
Use the SOAPEnvelope and SOAPExtract nodes to add or remove SOAP envelopes from the SOAP message body. You can use the SOAPExtract node both to extract the envelope, and to route the message based on the message content to a Label node.

Header nodes
Use the HTTPHeader, JMSHeader, or MQHeader nodes to manipulate HTTP, JMS, and WebSphere MQ transport headers and their properties without writing compute nodes. You cannot use these nodes to change the message body.

- Use the HTTPHeader node to add, modify, or delete HTTP headers such as HTTPInput, HTTPResponse, HTTPRequest and HTTPReply.
- Use the JMSHeader node to modify contents of the JMS Header_Values and Application properties so that you can control the node's output without programming.
- Use the MQHeader node to add, modify, or delete MQ Message Descriptor (MQMD) and MQ Dead Letter Header (MQDLH) headers.

User-defined processing node
Use a user-defined node processing node to handle specific requirements that are not met by the built-in nodes.

For example, if your node accesses a database, include a user-defined node to interact with the database. You can control the way in which the database is accessed by this node by specifying user and password information for the data source that you specify in the node property. Use the **mqsisetdbparms** command to initialize and maintain these values.

Related tasks:

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Nodes for making decisions:

Optionally, use nodes that determine the order and flow of control in the message flow in various ways to make decisions about how messages are processed by the flow.

Nodes for making decisions

Validate node
Use the Validate node to check that the message that arrives on its input terminal is as expected. You can check that the message has the expected message template properties (that is, the message domain, message set and message type) and that the content of the

message is correct. You can check the message against one or more of message domain, message set, or message type.

The Validate node replaces the Check node, which is deprecated in WebSphere Message Broker . The Validate node works in the same way as the Check node, but it has additional Validation properties to enable the validation of message content by parsers that support that capability.

Filter node

Use the Filter node with an ESQL statement to determine the next node to which the message is sent by this node. Do not use the ESQL code that you develop for use in a Filter node in any other type of node.

The node's terminals are True, False, Unknown, and Failure; the message is propagated to the True terminal if the test succeeds, and to the False terminal if it fails. If the statement cannot be resolved (for example, it tests the value of a field that is not in the input message), the message is propagated to the Unknown terminal. If any other error is detected, the message is propagated to the Failure terminal.

The test in the ESQL statement can depend on message content, database content, or a combination of the two.

If you refer to a database, you can control the way in which it is accessed by this node by specifying user and password information for each data source defined in the registry on the broker system. Use the **mqsisetdbparms** command to initialize and maintain these values.

Use this node in preference to the Compute node to provide message selection and routing; the Filter node is more efficient for this task.

FlowOrder node

You can connect the terminals of this node to force the message to be processed by one sequence of nodes, followed by a second sequence of nodes.

Passthrough node

Use the Passthrough node to enable version control of a subflow at run time. Use this node to add a label to your subflow. By combining this label with a reserved word replacement from your version control system, you can identify which version of a subflow is included in a deployed message flow. You can use this label for your own purposes. If you have included the correct version keywords in the label, you can see the value of the label:

- Stored in the broker archive (BAR) file, using the **mqsireadbar** command
- As last deployed to a particular broker, on the properties of a deployed message flow in the WebSphere Message Broker Toolkit
- In the broker, if you enable user trace for that message flow

Route node

Use the Route node to direct messages that meet certain criteria down different paths of a message flow. For example, you can forward a message to different service providers, based on the

request details. You can also use the Route node to bypass unnecessary steps. For example, you can check to see if certain data is in a message, and perform a database lookup operation only if the data is missing. If you set the `Distribution Mode` property to `All`, you can trigger multiple events that each require different conditions. For example, you can log requests that relate to a particular account identifier, and send requests that relate to a particular product to be audited.

Use the Route node to implement message routing with minimal programming logic. For more advanced routing scenarios, use a Compute node or a JavaCompute node.

RouteToLabel node
Use the RouteToLabel node following a Compute node or a JavaCompute node for complex routing. Define a list of destinations in a Compute or JavaCompute node that are acted on by the RouteToLabel node, which interrogates the destinations and passes the message on to the corresponding Label node.

DatabaseRoute node
Use the DatabaseRoute node to route a message using information from a database in conjunction with applied XPath routing expressions. The node looks up a collection of named column values from a located database row and synchronously applies one or more XPath expressions to these acquired values. Use the DatabaseRoute node to implement message routing with minimal programming logic. For more advanced routing scenarios, use a Compute node or a JavaCompute node.

Label node
Use the Label node as a target for the next sequence of one or more nodes that are to process a message. Use this node in combination with the RouteToLabel node for all types of messages, or with the SOAPExtract node for SOAP messages.

The Label node only routes the message to the next node in the flow and performs no processing.

ResetContentDescriptor node
Use the ResetContentDescriptor node to set new message properties that are used when the message bit stream is next parsed by a subsequent node in the message flow.

Related tasks:

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Nodes for controlling time-sensitive operations:

Run processes at specific times, or at fixed intervals, and take action if transactions are not completed within a defined time.

TimeoutControl node
Use a TimeoutControl node and a TimeoutNotification node together in a

message flow to control events that occur at a specific time or at defined time intervals. The TimeoutControl node receives an input message that contains a timeout request. All or part of this input message is validated and stored to be propagated by an associated TimeoutNotification node in the message flow. The input message is also propagated unchanged to the next node in the message flow.

More than one TimeoutControl node can be associated with each TimeoutNotification node.

TimeoutNotification node
Use a stand-alone TimeoutNotification node to generate messages that are propagated at configured times or time intervals to the next node in the message flow for further processing.

Related tasks:

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Miscellaneous nodes:

Nodes to perform a variety of tasks.

Nodes for collating requests

Use the AggregateControl, AggregateReply, and AggregateRequest nodes to collate related requests and responses. Use these nodes to generate several requests in response to one input message, to control and coordinate the responses that are received in response to those requests, and to combine the information that is provided by the responses to continue processing.

Node for creating message collections

Use the Collector node to generate collections of messages and make multiple synchronous or asynchronous requests in parallel. The Collector node does not need an initial fan-out stage, and can group unrelated input messages by correlating their content. You can configure dynamic input terminals on a Collector node to receive messages from different sources. You can also configure properties on the Collector node, known as event handlers, to determine how messages are added to a message collection, and when a message collection has completed.

Nodes for controlling the sequence of messages
Use the Sequence node to apply a sequence number to a message received from an input source, and the Resequence node to change the sequence of messages that contain a sequence number. You can divide messages into sequence groups and use the Sequence and Resequence nodes to control the sequences in groups of messages that arrive at the nodes.

Nodes for handling and reporting errors

Use the following nodes to affect error handling and reporting:

Trace node
Include a Trace node to generate one or more trace entries to record what is happening in the message flow at this point.

TryCatch node
Include a TryCatch node to control the error processing when exceptions are thrown.

Throw node
Include a Throw node to force an exception to be thrown, and specify the identity of the exception, to make it easier to diagnose the problem.

Node for invoking the message flow security manager
Use the SecurityPEP node to invoke the message flow security manager at any point in the message flow between an input node and an output (or request) node. The SecurityPEP node enables you to invoke the security manager even if your input nodes do not support message flow security. You can also use the SecurityPEP node to invoke different aspects of security (for example, authentication and authorization) at different points in the message flow.

Related tasks:

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Using more than one input node

You can include more than one input node in a single message flow.

Before you begin

Before you start:

Read the following concept topic:

- "Message flow nodes" on page 1024

About this task

You might find this useful in the following situations:

- The message flow provides common processing for messages that are received from multiple transports. For example, a single message flow might handle:
  - Data in messages received from WebSphere MQ, and therefore through a WebSphere MQ queue and an MQInput node
  - Messages that are received from native IP connections (a Real-timeInput node)
- You need to set standard properties on the MQInput node if input messages:
  - are predefined, and
  - are all received from WebSphere MQ, and
  - do not include an MQRFH2 header.

If the required standard properties are not always the same for every message, you can include more than one input node and configure each to handle a particular set of properties.

This requirement is not necessary for self-defining messages.

- Each input node in a message flow causes the broker to start a separate thread of execution. Including more than one input node might improve the message flow performance. However, if you include multiple input nodes that access the same input source (for example, a WebSphere MQ queue), the order in which the messages are processed cannot be guaranteed. If you want the message flow to process messages in the order in which they are received, this option is not appropriate.

  If you are not concerned about message order, consider using additional instances of the same message flow rather than multiple input nodes. If you set the `Additional Instances` property of the message flow when you deploy it to the broker, multiple copies of the message flow are started in the execution group. This is the most efficient way of handling multiple instances.

Look at the following sample :

- Scribble

This sample uses two input nodes: an MQInput node and a Real-timeInput node. You can use these two input nodes to enable the sample's message flow to accept input from both WebSphere MQ transport and native IP connections. You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Optimizing message flow response times" on page 3264
You can use different solutions to improve message flow response times.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

Chapter 11, "Packaging and deploying," on page 3209
Package resources that you create in the WebSphere Message Broker Toolkit, such as message flows, and deploy them to execution groups on brokers.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Defining input message characteristics

When a message is received by an input node in a message flow, the node detects how to interpret that message by determining the domain in which the message is defined and starting the appropriate parser.

Before you begin

Before you start:

Read the following concept topic:

- "Parsers" on page 1072

About this task

You can provide message domain information to the input node in one of two ways:

1. You can configure the built-in input nodes to indicate the message domain, and therefore the parser to be started, for each message that is received.
2. You can set values in the input message itself that specify this information. Include an MQRFH2 header, which contains a folder that defines the message characteristics. This approach is more flexible because it means that the input node can start the appropriate parser based on the content of each message.

If the input message is defined in the MRM domain, and is therefore interpreted by the MRM parser, you must specify the following additional properties:

- The `Message set` within which the message is defined
- The `Message type`, which is defined by the message model
- The `Message format`, which defines the physical characteristics of the message

The way that these properties are set depends upon the type of message, or node, that you want to use:

- If the message is a WebSphere MQ message, these properties can be set either in the input node or in the MQRFH2 header of the incoming message. If the properties are set in both, the properties of the MQRFH2 header take precedence. If the properties are not found in either the node or the MQRFH2 header, the default value is empty and the `BLOB` parser is used.
- If the message is a JMS message, the property that is set on the node takes precedence. If the `Message domain` is empty, the `Message domain` is, by default, derived according to certain criteria following a predetermined order of precedence; see "JMS message payload and appropriate parser" on page 1698.
- If the input message belongs to a `Message domain` other than those for which a parser is supplied, you must provide a user-defined parser to handle it, and a user-defined input node to accept it for processing in the message flow. Check the documentation provided with the user-defined parser and node for further information.
- If the `Message domain` is in a TimeoutControl node, an empty `Message domain` has either of the following results:
  - If the `Stored message location` property is also empty, the full message is stored. When the message comes back at TimeoutNotification, it is parsed in the same way as the original message.
  - If the `Stored message location` property is not empty, a partial message is stored and no parser is associated, therefore, by default, it is treated as `BLOB`.

- If the `Message domain` is in a ResetContentDescriptor node, an empty `Message domain` has either of the following results:
  - If `Reset message domain` is cleared, the domain is not reset.
  - If `Reset message domain` is selected, the default is `BLOB`.
- If the input node cannot determine the message characteristics, the default value is empty and the message is considered to be in the `BLOB` domain, and the `BLOB` parser is started.

Import either of the following samples, or another sample that uses a `Message set`, and look at the values on the **Input Message Parsing** properties tab of the input node in the sample's message flow.

- Video Rental
- Comma Separated Value (CSV)

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Parsers" on page 1072
A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from the internal message tree representation.

"Logical tree structure" on page 1042
The logical tree structure is the internal (broker) representation of a message. It is also known as the message assembly.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"How the message tree is populated" on page 1047
The message tree is initially populated by the input node of the message flow.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"MQRFH2 structure" on page 6397

Creating destination lists

Create a list of destinations to indicate where a message is sent.

Before you begin

Before you start:

Read the concept topic "Message flow nodes" on page 1024.

About this task

You can include a Compute node in your message flow, and configure it to create a destination list in the local environment subtree. You can then use the destination list in the following nodes:

- The MQOutput and JMSOutput nodes, to put output messages to a specified list of destinations.
- The RouteToLabel node, to pass messages to Label nodes.

  For details about how this technique is used, look at the following sample:

  – Airline Reservations

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

For more information about accessing the LocalEnvironment subtree, destination list contents, and example procedures for setting values for each of these scenarios, see "Accessing the local environment tree" on page 2463.

For more information about how to populate destination in the LocalEnvironment subtree, and how to build JMS destination lists, see "Populating Destination in the local environment tree" on page 2467.

You might find it useful to create the contents of the destination list from an external database that is accessed by the Compute node. You can then update the destinations without needing to update and redeploy the message flow.

The use of the destination list to define which applications receive the output messages is in contrast to the publish/subscribe application model, in which the recipients of the publications are those subscribers that are currently registered with the broker. The processing that is completed by the message flow does not have any effect on the current list of subscribers.

Related concepts:

"Local environment tree structure" on page 1056
The local environment tree is a part of the logical message tree in which you can store information while the message flow processes the message.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Validating messages

The broker provides validation based on the message model for predefined messages.

Before you begin

Before you start:

Read the concept topics about message flows and parsers, especially "MRM parser and domain" on page 1111 and "XMLNSC parser" on page 1090.

About this task

Validation applies only to messages that you have modeled and deployed to the broker. Specifically, the message domains that support validation are MRM, XMLNSC, SOAP, and IDOC.

The broker does not provide any validation for self-defining messages. The XMLNSC, and SOAP domains validate predefined messages directly against message model schema filesXML Schema files. The MRM and IDOC parsers validate predefined messages against the message dictionary generated from a message set.

Message flows are designed to transform and route messages that conform to certain rules. By default, parsers perform some validity checking on a message, but only to ensure the integrity of the parsing operation. However, you can validate a message more stringently against the message model contained in the message set by specifying validation options on certain nodes in your message flow.

You can use validation options to validate the following messages:
- Input messages that are received by an input node
- Output messages that are created, for example, by a Compute, Mapping, or JavaCompute node

These validation options can ensure the validity of data entering and leaving the message flow. The options provide you with some degree of control over the validation performed to:

- Maintain a balance between performance requirements and security requirements

- Validate at different stages of message flow completion; for example, on input of a message, before a message is propagated, or at any point in between
- Cope with messages that your message model does not fully describe

You can also specify what action to take when validation fails.

Message validation involves navigating a message tree, and checking the validity of the tree. Message validation is an extension of tree creation when the input message is parsed, and of bit stream creation when the output message is written.

Validation options are available on the following nodes:

Node type	Nodes with validation options
Input node	FileInput, FTEInput, HTTPInput, JMSInput, MQInput, SCAInput, , SOAPInput,TimeoutNotification,
Output node	FileOutput, FTEOutput, HTTPReply, JMSOutput, JMSReply, MQOutput, MQReply, SCAReply, SOAPReply
Other nodes	Compute, CICSRequest, DatabaseRetrieve, HTTPRequest, JavaCompute, Mapping, MQGet, ResetContentDescriptor, SCAAsyncResponse, SCARequest, SOAPRequest, SOAPAsyncResponse, Validate, XSLTransform

Validation options can also be specified on the ESQL CREATE statement and the ASBITSTREAM function.

To validate input messages that are received on an input node, you can specify validation properties on the input node. The input message is then validated when the message bit stream is parsed to form the message tree.

You can also use the `Parse Timing` property of the input node to control whether the entire message is parsed and validated at this time, or whether individual fields in the message are parsed and validated only when referenced.

To validate output messages that are created by a transformation node, specify validation properties either on the node itself, or on the output node that sends the message. The validation takes place when the message bit stream is created from the message tree by the output node.

Alternatively, use a Validate node to validate a message tree at a particular place in your message flow, or use the ESQL ASBITSTREAM function in a Compute, Filter, or Database node.

A limited amount of validation occurs by default if you leave the validation settings unaltered. At this default level, an exception is thrown if one of the following statements is true:

- A data mismatch occurs; for example, the parser cannot interpret the data that is provided for the field type specified.
- The order of elements in the output message does not match the order of elements in the logical message tree (MRM CWF, and MRM TDS fixed-length models only).

Additionally, the MRM parser performs limited remedial action under the following circumstances:

- Extraneous fields are discarded on output for fixed formats (CWF and TDS fixed-length models only).

- If mandatory content is missing, default values are supplied, if available, on output for fixed formats (CWF and TDS fixed-length models only).
- If the data type of an element in the tree does not match that specified in the dictionary, the data type is converted on output to match the dictionary definition, if possible, for all formats.

However, by using validation options you can request more thorough validation of messages. For example, you might want to validate one or more of the following conditions, and throw an exception, or log the errors:

- The whole message at the start of the message flow
- That complex elements have the correct `Composition` and `Content Validation`
- That all data fields contain the correct type of data
- That data fields conform to the value constraints in the message model
- That all mandatory fields are present in the message
- That only the expected fields are present in the message
- That message elements are in the correct order

The samples illustrate some of these validation options.

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

When using validation options, it is important to understand the following behavior.

- The `Parse Timing` property, which controls whether *on-demand* parsing (sometimes called partial parsing) takes place, affects the timing of the validation of input messages, including message headers.

  For more information about the `Parse Timing` property, see "Parsing on demand" on page 4173.
- If a message tree is passed to an output node, by default the output node inherits the validation options in force for the message tree. You can override these options by specifying a new set of validation options on the output node.
- If a message tree is passed as input to a Compute, Mapping, XSLTransform, DatabaseRetrieve, or JavaCompute node, any new output message trees that the node creates have the validation options specified by the node itself (even if the whole message is copied). You can override this behavior and specify that the messages that are created by the node inherit the validation options of the input message tree.
- (MRM domain only) When the bit stream is written, and validation options are applied, the entire message is validated. The message tree might contain an unresolved type (for example, if a Compute node copied an unresolved type from an input message to an output message without resolving it). If such a type is encountered, a validation error occurs because it is not possible to validate the type. To prevent this error, ensure that all unresolved types are resolved before they are copied to output messages.
- (MRM domain only) Do not select the `Truncate fixed length strings` check box because validation is done before truncation, and a fixed-length field fails validation if its length exceeds the length that is defined in the message set. For

more information about the `Truncate fixed length strings` property, see "Custom Wire Format message set properties" on page 5375 and "TDS Format message set properties" on page 5381.

For information about how you can control validation by using different properties, see "Validation properties" on page 4169.

Related concepts:

"Parsers" on page 1072
A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from the internal message tree representation.

"MRM parser and domain" on page 1111
You can use the MRM domain to parse and write a wide range of message formats.

"SOAP parser and domain" on page 1082
You can use the SOAP parser to create a common WSDL-based logical tree format for working with Web services, independent of the physical bitstream format.

"XML parsers and domains" on page 1084
You can use XML domains to parse and write messages that conform to the W3C XML standard.

"XMLNSC parser" on page 1090
The XMLNSC parser is a flexible, general-purpose XML parser that offers high performance XML parsing and optional XML Schema validation.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"Predefined and self-defining messages" on page 1076
Both predefined and self-defining messages are supported.

Related reference:

"Parsing on demand" on page 4173
On-demand parsing, referred to as partial parsing, is used to parse an input message bit stream only as far as is necessary to satisfy the current reference. The parsers that can perform partial parsing of input messages are the JSON, MRM, XML, XMLNS, and XMLNSC parsers.

"Validation properties" on page 4169
You can control validation by setting properties on the Validate and Parser Options tabs for the nodes that are listed in the following table.

"CREATE statement" on page 5082
The CREATE statement creates a new message field.

"ASBITSTREAM function" on page 5224
The ASBITSTREAM field function generates a bit stream for the subtree of a given field according to the rules of the parser that owns the field.

Viewing the logical message tree in trace output

To view the structure of the logical message tree at any point in the message flow, include a Trace node and write some or all the message (including headers and all four message trees) to the trace output destination.

About this task

You might find trace output useful to check or record the content of a message before and after a node has changed it, or on its receipt by the input node. For example, if you include a Compute node that builds a destination list in the local environment tree, you might want a record of the structure that it has created as part of an audit trail, or you might want to check that the Compute node is working as you expect it to.

UNIX On UNIX, syslog entries are restricted in length and messages that are sent to the syslog are truncated by the newline character. To record a large amount of data in a log on UNIX, set the `Destination` property on the Trace node to `File` or `User Trace` instead of `Local Error Log`.

Procedure

1. Switch to the Broker Application Development perspective.
2. Open the message flow for which you want to view messages. Open an existing message flow, or create a message flow.
3. Include a Trace node wherever you want to view part or all the message tree structure. You can include as many Trace nodes as you choose; however, each node that you introduce can affect the performance of message flow processing.
4. Set the Trace node properties to trace the message, or parts of the message, that you want to view. Specify the parts of the message by using ESQL field references. Several examples are included later in this topic.
5. If you have added a Trace node to investigate a particular behavior of your message flow, and have now resolved your concerns or checked that the message flow is working correctly, remove the Trace node or nodes, and redeploy the message flow.

Example

Assume that you have configured a message flow that receives an XML message on a WebSphere MQ queue in an MQInput node. The input message includes an MQRFH2 header. The message has the following content:

```
<Trade type='buy'
 Company='IBM'
 Price='200.20'
 Date='2000-01-01'
 Quantity='1000'/>
```

You can include and configure a Trace node to produce output that shows one or more of the trees created from this message: the message body, environment, local environment, and exception trees. If you choose to record the content of the message body, the Properties tree and the contents of all headers (in this example, at least an MQMD and an MQRFH2) are included. You specify what you want to be recorded when you set the Trace node property `Pattern`. You can use most of the correlation names to define this pattern (you cannot use those names that are specific to the Compute node).

Message body

If you want the Trace node to write the message body tree including Properties and all headers, set `Pattern` to $Root. If you want only the message data, set `Pattern` to ${Body}.

The trace output generated for the message tree of the preceding message with Pattern set to $Root would look like the following example:

```
Root
      Properties
         CreationTime=GMTTIMESTAMP '1999-11-24 13:10:00'          (a GMT timestamp field)

  ... and other fields ...

      MQMD
         PutDate=DATE '19991124'                                  (a date field)

         PutTime=GMTTIME '131000'                                 (a GMTTIME field)

  ... and other fields ...

      MQRFH
          mcd
          msd='xml'                                              (a character string field)

         .. and other fields ...

      XML
         Trade
          type='buy'                                             (a character string field)

         Company='IBM'                                           (a character string field)

         Price='200'                                             (a character string field)

         Date='2000-01-01'                                       (a character string field)

         Quantity='1000'                                         (a character string field)
```

Environment

To trace any data in the environment tree, set Pattern to ${Environment}. This setting produces output like the following example:

```
(0x1000000)Environment = (
     (0x1000000)Variables = (
       (0x1000000)MyVariable1 = (
         (0x2000000) = '3'
       )
       (0x1000000)MyVariable2 = (
         (0x2000000) = 'Hello'
       )
     )
 )
```

To trace particular variables in the variables folder of the environment tree, you can use a more specific pattern, for example ${Environment.Variables.MyVariable1}. This setting returns the value only (for example, it returns just the value 3).

LocalEnvironment

To trace data in the local environment tree, set Pattern to ${LocalEnvironment}. The output you get is like the following example, which shows that a destination list has been created in the local environment tree:

```
(0x1000000)Destination = (
    (0x1000000)MQ               = (
      (0x1000000)DestinationData = (
        (0x3000000)queuename = 'MQOUT'
      )
    )
     (0x1000000)MQDestinationList = (
      (0x1000000)DestinationData = (
        (0x3000000)queuename = 'OLDMQOUT'
      )
    )
    (0x1000000)RouterList       = (
      (0x1000000)DestinationData = (
        (0x3000000)labelname = 'continue'
      )
      (0x1000000)DestinationData = (
        (0x3000000)labelname = 'custdetails'
      )
      (0x1000000)DestinationData = (
        (0x3000000)labelname = 'trade'
      )
    )
  )
```

Another example, shown here, includes a WrittenDestination folder. This example represents a trace that has been written by a Trace node that is included after an MQOutput node, where the Out terminal of the MQOutput node is connected to a sequence of nodes including the Trace node. When an Out terminal is connected, the local environment is augmented with information about the action that the output node has performed.

```
(0x1000000)Destination       = (
    (0x1000000)MQ               = (
      (0x1000000)DestinationData = (
        (0x3000000)queuename = 'MQOUT'
      )
    )
  (0x1000000)WrittenDestination = (
    (0x1000000)MQ = (
      (0x1000000)DestinationData = (
        (0x3000000)queueName        = 'MQOUT'
        (0x3000000)queueManagerName = 'MQSI_SAMPLE_QM'
        (0x3000000)replyIdentfier   = X'414d51204d5153495f53414d504c455f1f442f3b12600100'
        (0x3000000)msgId            = X'414d51204d5153495f53414d504c455f1f442f3b12600100'
        (0x3000000)correlId         = X'000000000000000000000000000000000000000000000000'
 (0x03000000):GroupId          = X'414d512042524f4b45523220202020203f59934620001803'
      )
    )
  )
)
```

ExceptionList

To trace data in the exception list, set `Pattern` to ${ExceptionList}.

You can also view message structure in the message flow, and other information, when you use the flow debugger.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Logical tree structure" on page 1042
The logical tree structure is the internal (broker) representation of a message. It is also known as the message assembly.

"Correlation names" on page 1069
A correlation name is a field reference that identifies a well-defined starting point in the logical message tree and is used in field references to describe a standard part of the tree format.

"Flow debugger overview" on page 3158
Use the flow debugger in the WebSphere Message Broker Toolkit to track messages through your message flows.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Manipulating message body content" on page 2418
The message body is always the last child of root, and is identified by its parser name, for example XML or MRM. You can refer to, modify, and create message body data.

"Manipulating other parts of the message tree" on page 2452
You can access message tree headers, the properties tree, the local environment tree, the environment tree and the exception list tree.

Chapter 10, "Testing and debugging message flow applications," on page 3143
Use the flow debugger to track messages through your message flows and check for errors, or use the Test Client to test your message flows in a safe environment before they are used on a production system.

Related reference:

"Trace node" on page 4942
Use the Trace node to generate trace records that you can use to monitor the behavior of a message flow.

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

"MQOutput node" on page 4612
Use the MQOutput node to send messages to clients that connect to the broker using the WebSphere MQ Enterprise Transport and that use the MQI and AMI application programming interfaces.

"Exception list structure" on page 4224
An exception list contains information about exceptions, such as error numbers, the name of the node that generated the exception, and the reason for the exception.

Changing the parser used in a message flow

If you need to change the parser that is used in a message flow, use the ResetContentDescriptor node to request that the message is reparsed by a different parser.

About this task

Use the ResetContentDescriptor node to set new message properties that are used when the message bit stream is next parsed by a subsequent node in the message flow.

For example, if the format of an incoming message is unknown when it enters a message flow, the BLOB parser is started. Later on in the message flow, you might decide that the message is predefined as a message in the MRM domain, and you can use the ResetContentDescriptor node to set the correct values to use when the message is parsed by a subsequent node in the message flow.

You can select from the following parsers:
- MRM
- XMLNSC
- DataObject
- XMLNS
- JMSMap
- JMSStream
- MIME
- BLOB
- XML (this domain is deprecated; use XMLNSC)
- IDOC (this domain is deprecated; use MRM)
- JSON

If you specify MRM, XMLNSC, DataObject, or IDOC as the new parser, you can also specify a different message template (message set, message type, and message format).

Whether or not the node reparses the message straight away depends on the settings of the `Parse timing` option in the node properties. `Parse timing` is, by default, set to `On Demand`, which causes parsing of the message to be delayed. For more details on controlling when the message is parsed, see "Parsing on demand" on page 4173.

Related concepts:

"Specifying namespaces in the Message Type property" on page 1208
When using the MRM domain, the *Message Type* property is used to specify the name of the message.

"Multipart messages" on page 1191
A *multipart message* contains one or more other messages within its structure. The contained message is sometimes referred to as an *embedded message*.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Parsers" on page 1072
A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from the internal message tree representation.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Validating messages" on page 1478
The broker provides validation based on the message model for predefined messages.

"Manipulating messages in the XMLNSC domain" on page 2546
If you are writing ESQL to process messages in the XMLNSC domain, it is helpful to learn about the structure of the message tree that the XMLNSC parser builds.

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

Related reference:

"ResetContentDescriptor node" on page 4663
Use the ResetContentDescriptor node to request that the message is reparsed by a different parser.

"Validation properties" on page 4169
You can control validation by setting properties on the Validate and Parser Options tabs for the nodes that are listed in the following table.

"Parsing on demand" on page 4173
On-demand parsing, referred to as partial parsing, is used to parse an input message bit stream only as far as is necessary to satisfy the current reference. The parsers that can perform partial parsing of input messages are the JSON, MRM, XML, XMLNS, and XMLNSC parsers.

Providing user-defined properties to control behavior

User-defined properties can be set at design time, deployment time, or run time.

About this task

For example, user-defined properties can be queried, discovered, and set at run time to dynamically change the behavior of a message flow. You can use the Administration API for WebSphere Message Broker (also known as the CMP API) to manipulate these properties, which can be used by a systems monitoring tool to perform automated actions in response to situations that it detects in the monitored systems.

Procedure

- You can use the Message Flow editor to define a user-defined property when you construct a message flow. For more information, see "Message Flow editor" on page 6810.
- You can set user-defined properties at deployment time to configure a message flow, as described in "Configuring a message flow at deployment time with user-defined properties" on page 2626.
- You can use the CMP API to manipulate user-defined properties on a message flow dynamically at run time, as described in "Setting message flow user-defined properties at run time in a CMP application" on page 985.

Related concepts:

"User-defined properties" on page 1147
A user-defined property (UDP) is a property that is defined when you construct a message flow by using the Message Flow editor. This property can be used by the ESQL or Java program inside message flow nodes, such as a Compute node.

"User-defined properties in ESQL" on page 2376
Access user-defined properties (UDPs) as variables in your ESQL program by specifying the EXTERNAL keyword on a DECLARE statement. For example, the ESQL statement `DECLARE today EXTERNAL CHARACTER 'monday'` defines a user-defined property called today with an initial value monday.

Related tasks:

"Accessing message flow user-defined properties from a JavaCompute node" on page 2659
Customize a JavaCompute node to access properties that you have associated with the message flow in which the node is included.

Related reference:

"Message Flow editor" on page 6810
The Message Flow editor is the default editor provided by the Broker Application Development perspective for defining a graphical representation of a message flow in the WebSphere Message Broker Toolkit, and for setting properties for individual message flow nodes.

Defining message flow content

You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

About this task

When you create a message flow, the editor view is initially empty. You must create the contents of the message flow by using a combination of the following tasks:

- "Adding a message flow node" on page 1494
- "Adding a subflow" on page 1501
- "Renaming a message flow node" on page 1502
-
- "Connecting message flow nodes" on page 1520
- "Inserting nodes into existing message flows" on page 1525
- "Adding a bend point" on page 1527
- "Aligning and arranging nodes" on page 1530

While you are developing message flows, you might want to record notes about flow development or particular nodes in the flow:

- "Adding annotations to a message flow or node" on page 1531
- "Editing annotations on a message flow or node" on page 1533
- "Copying annotations on a message flow or node" on page 1534
- "Showing and hiding annotations on a message flow or node" on page 1535
- "Deleting annotations from a message flow or node" on page 1536

When you finalize the contents of the message flow, you might also need to complete the following tasks:

- "Removing a message flow node" on page 1519
- "Removing a node connection" on page 1526
- "Removing a bend point" on page 1528

To learn more about message flow content, you can import either of the following samples:

- Airline Reservations
- Error Handler

Follow the supplied instructions to build the sample yourself. You can also try adding and deleting nodes, adding subflows, and connecting nodes together.

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

For a basic introduction to developing message flows, see the IBM Redbooks publication WebSphere Message Broker Basics.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related reference:

"Broker Application Development perspective" on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Using the node palette

The node palette contains all the built-in nodes, which are organized into categories.

Before you begin

Before you start:

Read the following concept topic "Message flow node palette" on page 1027.

About this task

You can add the nodes that you use most often to the Favorites category by following the instructions in "Adding nodes to the Favorites category on the palette" on page 1492.

You can change the palette preferences in the WebSphere Message Broker Toolkit. The changes that you can make are described in the following topics.

Procedure

- "Changing the palette layout" on page 1490
- "Changing the palette settings" on page 1490
- "Customizing the palette" on page 1491

Related concepts:

"Message flow node palette" on page 1027
The palette in the Message Flow editor contains all the built-in nodes, which are organized into categories, or drawers. A drawer is a container for a list of icons, such as the **Favorites** drawer.

Related tasks:

"Choosing the location of a user-defined node in the palette" on page 3092
Use the Palette editor to edit palette-specific information for user-defined nodes, add and delete separators, and rearrange user-defined nodes in the palette.

Related reference:

"Message Flow editor" on page 6810
The Message Flow editor is the default editor provided by the Broker Application Development perspective for defining a graphical representation of a message flow in the WebSphere Message Broker Toolkit, and for setting properties for individual message flow nodes.

Changing the palette layout:

You can change the layout of the palette in the Message Flow editor.

Procedure

1. Switch to the Broker Application Development perspective
2. Right-click the palette to display the pop-up menu.
3. Click **Layout**.
4. Click one of the available views:

   Columns
   Displays named icons in one or more columns. Change the number of columns by clicking on the right edge of the palette and dragging.

   List Displays named icons in a single-column list. The list view is the default layout.

   Icons Only
   Displays a list of icons only.

   Details
   Displays descriptions of the icons.

Related tasks:

"Customizing the palette" on page 1491
If you customize the message flow node palette, you can make it easier to find the nodes that you use most often.

"Changing the palette settings"
How to use the **Palette Settings** dialog box.

Related reference:

"Message Flow editor" on page 6810
The Message Flow editor is the default editor provided by the Broker Application Development perspective for defining a graphical representation of a message flow in the WebSphere Message Broker Toolkit, and for setting properties for individual message flow nodes.

Changing the palette settings:

How to use the **Palette Settings** dialog box.

About this task

Change the palette settings in the Message Flow editor using the **Palette Settings** dialog box.

Procedure

1. Switch to the Broker Application Development perspective.
2. Right-click the palette to display the pop-up menu.
3. Click **Settings**. The **Palette Settings** dialog box opens.
4. Use the dialog to change the appropriate setting:

- Click **Change** to change the font on the palette.
- Click **Restore Default** to restore the default palette settings.
- In the **Layout** list, click the appropriate radio button to change the palette layout. (See "Changing the palette layout" on page 1490 for more information.)
- Select **User large icons** to toggle between large and small icons in the palette.
- In the **Drawer options** list, click the appropriate radio button to change the way that drawers are handled in the palette. A drawer is a container for a list of icons, such as the **Favorites** drawer on the Message Flow editor's palette, or the **Entity** drawer on the Broker Topology editor's palette.

Related tasks:

"Changing the palette layout" on page 1490
You can change the layout of the palette in the Message Flow editor.

"Customizing the palette"
If you customize the message flow node palette, you can make it easier to find the nodes that you use most often.

Related reference:

"Message Flow editor" on page 6810
The Message Flow editor is the default editor provided by the Broker Application Development perspective for defining a graphical representation of a message flow in the WebSphere Message Broker Toolkit, and for setting properties for individual message flow nodes.

Customizing the palette:

If you customize the message flow node palette, you can make it easier to find the nodes that you use most often.

About this task

This saves time and on-screen space. For example:

- Change the order of the drawers in the palette so that the ones that you use most often are at the top.
- Hide any drawers that you do not use, to save on-screen space.
- Pin open the drawers that contain the nodes that you use most often.
- Create your own drawers to hold user-defined nodes that you create.

Customize the palette for the Message Flow editor using the Customize Palette dialog box:

Procedure

1. Switch to the Broker Application Development perspective.
2. Right-click the palette, then click **Customize**. The Customize Palette dialog box opens.
   - To change the order of entries and drawers in the palette, click the appropriate item in the list to highlight it, then click **Move Down** or **Move Up**. You cannot move any category above the Favorites category.
   - To hide an entry or drawer, click the appropriate item in the list to highlight it, then select the **Hide** check box.
   - To create a new separator, click **New** > **Separator**.
   - To create a new drawer:
     a. Click **New** > **Drawer**.

b. Type a name and description for the drawer.
c. If required, select the **Open drawer at start-up** check box.
d. If required, select the **Pin drawer open at start-up** check box.

3. Click **OK** or **Apply** to save your changes.

Results

You have customized the message flow node palette.

Related tasks:

"Changing the palette layout" on page 1490
You can change the layout of the palette in the Message Flow editor.

"Changing the palette settings" on page 1490
How to use the **Palette Settings** dialog box.

Related reference:

"Message Flow editor" on page 6810
The Message Flow editor is the default editor provided by the Broker Application Development perspective for defining a graphical representation of a message flow in the WebSphere Message Broker Toolkit, and for setting properties for individual message flow nodes.

Adding nodes to the Favorites category on the palette:

The nodes on the palette are organized in categories. The first category is Favorites, which is usually empty. You can drag the nodes that you use most often to the Favorites category.

Before you begin

Before you start:

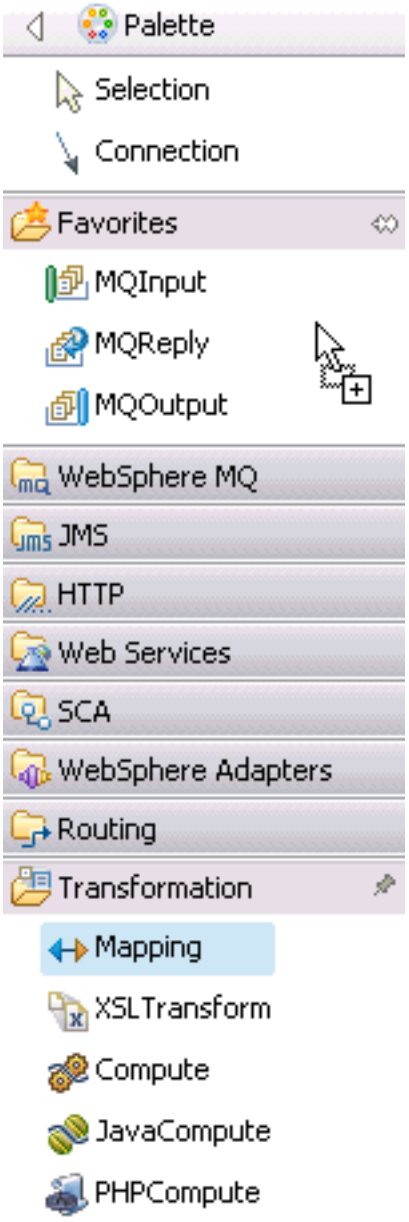
Read the concept topic about the message flow node palette.

About this task

The following steps describe how to drag nodes into the Favorites category.

Procedure

1. Switch to the Broker Application Development perspective.
2. On the palette, open the Favorites category.
3. On the palette, open the category that contains the node that you want to add to the Favorites category.
4. Use the mouse to drag the node into the Favorites category, as shown in the following example:



Results

Alternatively, right-click the palette and choose the appropriate option to add or remove nodes from the Favorites category.

Related tasks:

"Changing the palette layout" on page 1490
You can change the layout of the palette in the Message Flow editor.

"Changing the palette settings" on page 1490
How to use the **Palette Settings** dialog box.

"Customizing the palette" on page 1491
If you customize the message flow node palette, you can make it easier to find the nodes that you use most often.

"Adding a message flow node" on page 1494
When you have created a message flow, add nodes to define its function.

"Creating a user-defined node in the WebSphere Message Broker Toolkit" on page 3081
Create the representation of a user-defined node created in Java and C only, in the WebSphere Message Broker Toolkit.

Related reference:

"Message Flow editor" on page 6810
The Message Flow editor is the default editor provided by the Broker Application Development perspective for defining a graphical representation of a message flow in the WebSphere Message Broker Toolkit, and for setting properties for individual message flow nodes.

"Plug-in Development projects and files" on page 6825

Adding a message flow node

When you have created a message flow, add nodes to define its function.

Before you begin

Before you start:

- Create a message flow or open an existing message flow; see "Creating a message flow" on page 1431 and "Opening an existing message flow" on page 1433.
- Read the concept topic about message flow nodes, see "Message flow nodes" on page 1024.

About this task

To add a node to a message flow:

Procedure

1. Open the message flow with which you want to work.
2. Open the Palette.
   - Hold the mouse pointer over the palette bar while it is in collapsed mode. The palette bar expands. When you move the mouse pointer away from the palette bar, it collapses again.
   - Click the **Show Palette** icon at the top of the palette bar. The palette bar expands and it remains expanded when the mouse is moved away from the palette bar. To collapse the palette bar again, click the **Hide Palette** icon at the top of the palette bar while it is in expanded mode.
3. Click **Selection** above the palette of nodes.
4. Decide which node you want to add: a built-in node or a user-defined node. You can select any of the nodes that are displayed in the node palette, but you can add only one node at a time.

   Nodes are grouped in categories according to the function that they provide. To see descriptions of the nodes in the palette, either hold the mouse pointer over a node in the palette, or switch to the Details view by following the instructions in "Changing the palette layout" on page 1490.
5. Drag the node from the node palette onto the canvas.

   When you add a node to the canvas, the editor automatically assigns a name to the node, but the name is highlighted and you can change it by entering a name of your choice. If you rename the node, the name that you choose must be unique in the message flow. If you do not change the default name at this time, you can change it later, see "Renaming a message flow node" on page 1502. The default name is set to the type of node for the first instance. For

example, if you add an MQInput node to the canvas, it is given the name MQInput; if you add a second MQInput node, the default name is MQInput1; the third is MQInput2, and so on.

6. Repeat steps 4 on page 1494 and 5 on page 1494 to add further nodes.
7. You can also add nodes from other flows into this flow:
   a. Open the other message flow.
   b. Select the node or nodes that you want to copy from the editor or outline views, and press Ctrl+C or click **Edit** > **Copy**.
   c. Return to the flow with which you are currently working.
   d. Press Ctrl+V or click **Edit** > **Paste**. This action copies the node or nodes into your current flow. The node names and properties are preserved in the new copy.

Results

When you have added the nodes that you want in this message flow, you can connect them to specify the flow of control through the message flow, and you can configure their properties.

You can also add user-defined nodes to your message flow. To find out about user-defined nodes, see "User-defined nodes" on page 2989.

What to do next

Now you can configure the nodes, see "Configuring a message flow node" on page 1503.

Related concepts:

"Message flow node palette" on page 1027
The palette in the Message Flow editor contains all the built-in nodes, which are organized into categories, or drawers. A drawer is a container for a list of icons, such as the **Favorites** drawer.

Related tasks:

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

"Opening an existing message flow" on page 1433
Open an existing message flow to change or update its contents, or to add or remove nodes.

"Adding a node by using the keyboard" on page 1498
You can use the keyboard to perform tasks in the Message Flow editor, such as adding a node to the canvas.

"Dragging a resource from the Broker Development view" on page 1499
Drag a node or a related resource into the Message Flow editor.

"Renaming a message flow node" on page 1502
You can change the name of any type of node (a built-in node, user-defined node, or subflow node) to reflect its purpose.

"Adding a subflow" on page 1501
In a message flow, you can include an embedded message flow, also known as a *subflow*. For example, you might define a subflow that provides error handling, and include it in a message flow connected to a failure terminal on a node that can generate an error in some situations.

"Removing a message flow node" on page 1519
When you have created and populated a message flow, you might need to remove a node to change the function of the flow, or to replace it with another more appropriate node. The node can be a built-in node, a user-defined node, or a subflow node.

"Connecting message flow nodes" on page 1520
When you include more than one node in your message flow, you must connect the nodes to indicate how the flow of control passes from input to output. The nodes can be built-in nodes, user-defined nodes, or subflow nodes.

Related reference:

"WebSphere Message Broker Explorer and WebSphere Message Broker Toolkit keyboard shortcuts" on page 6828
You can navigate all interfaces in the WebSphere Message Broker Explorer and WebSphere Message Broker Toolkit by using the keyboard.

Installing a user-defined node:

Develop message flows that use a user-defined node.

About this task

You can develop message flows by using the user-defined node in the same way as the built-in nodes. Before you can use a user-defined node, you must install it into your WebSphere Message Broker Toolkit. A user-defined node is installed differently depending on how it was packaged.

Procedure

1. Install the user-defined node project:
   - If the user-defined node was packaged as plug-in JAR files, copy your JAR files into the dropins folder in your WebSphere Message Broker Toolkit installation location, and run the **mb -clean -initialize** command to pick up the new files.

     Note: For some WebSphere Message Broker Toolkit installations, the dropins folder (for example `C:\Program Files (x86)\IBM\WMBT700`) is not created. In this case, you can create the dropins folder manually.
   - If the user-defined node was packaged as an update site, install the plug-ins from the update site, see "Installing from an update site" on page 3124.
2. Stop and restart your WebSphere Message Broker Toolkit. The user-defined nodes are displayed in the palette in the Message Flow editor under the category that you specified for the user-defined node project.

Related concepts:

"User-defined nodes" on page 2989
User-defined nodes are the main mechanism for extending the functions of WebSphere Message Broker.

"Using a subflow as a user-defined node" on page 3008
You can develop a user-defined node that packages a subflow from scratch, in the same way that you can create any other user-defined node that has its implementation based on Java, or base it on an existing subflow.

Related tasks:

"Debugging the message flow in simulation mode" on page 3098
Compile, deploy, test, and debug the message flow that includes your user-defined node.

"Creating a user-defined node from a subflow" on page 3076
Create user-defined nodes either from scratch, or by using an existing subflow.

"Packaging as plug-in JAR files" on page 3122
Export the user-defined node project as plug-in JAR files to make it available for other users.

"Packaging as an update site" on page 3122
Create an installation site for Web page distribution of the user-defined node projects that includes or references other projects or plug-ins. You can either create a new update site, or use an existing one.

"Installing from an update site" on page 3124
If you have packaged your user-defined nodes into an update site, use the software update mechanism to install the user-defined nodes.

"Testing a subflow user-defined node project" on page 3097
Test how the user-defined node is displayed, and how it behaves in your environment.

Message sets packaged with a user-defined node:

Adding message sets to an update site.

About this task

Message sets are automatically packaged into user-defined node JAR files and update sites. If you drag a user-defined node from the palette onto the canvas, the WebSphere Message Broker Toolkit detects whether the required message sets are available in the current workspace.

Procedure

1. If the required message sets are unavailable in the current workspace, a warning is displayed in the **Problems** tab, and the **Import Required Message Sets** window opens.
2. To import the message set from the user-defined node plug-in, click **Import selected message sets**, or you can import the message sets from another source.

   If the message set is available in the same execution group at execution time, you do not have to import the message sets.

Related concepts:

"User-defined nodes" on page 2989
User-defined nodes are the main mechanism for extending the functions of WebSphere Message Broker.

"Using a subflow as a user-defined node" on page 3008
You can develop a user-defined node that packages a subflow from scratch, in the same way that you can create any other user-defined node that has its implementation based on Java, or base it on an existing subflow.

Related tasks:

"Creating a user-defined node from a subflow" on page 3076
Create user-defined nodes either from scratch, or by using an existing subflow.

"Testing a subflow user-defined node project" on page 3097
Test how the user-defined node is displayed, and how it behaves in your environment.

"Packaging and distributing a user-defined node project" on page 3121
Export the user-defined node project to make it available for other users.

Adding a node by using the keyboard:

You can use the keyboard to perform tasks in the Message Flow editor, such as adding a node to the canvas.

Before you begin

Before you start:
- Ensure that you have created or opened a message flow. For more information, see "Creating a message flow" on page 1431 or "Opening an existing message flow" on page 1433.
- Read the concept topic, "Message flow nodes" on page 1024.

Procedure

1. Open the message flow to which you want to add a node.
2. Open the Palette view or the Palette bar.
3. Select a node in the Palette view or Palette bar by using the up and down arrows to highlight the node that you want to add to the canvas.
4. Add the node to the canvas by using one of the following methods:
   - Press **Alt + L**, then press **N**.
   - Press **Shift + F10** to open the pop-up menu for the Palette, and press **N**.

   The node that you selected in the Palette bar or Palette view is placed on the canvas in the Editor view.

   When you add a node to the canvas, the editor automatically assigns a name to the node, but the name is highlighted and you can change it by entering a name of your choice. If you do not change the default name at this time, you can change it later by following the instructions in "Renaming a message flow node" on page 1502. The default name is set to the type of node for the first instance. For example, if you add an MQInput node to the canvas, it is given the name MQInput. If you add a second MQInput node, the default name is MQInput1; the third is MQInput2, and so on.

Results

You can move the node that you have placed on the canvas by using the keyboard controls described in "WebSphere Message Broker Explorer and WebSphere Message Broker Toolkit keyboard shortcuts" on page 6828.

Related concepts:

"Message flow node palette" on page 1027
The palette in the Message Flow editor contains all the built-in nodes, which are organized into categories, or drawers. A drawer is a container for a list of icons, such as the **Favorites** drawer.

Related tasks:

"Adding a message flow node" on page 1494
When you have created a message flow, add nodes to define its function.

"Dragging a resource from the Broker Development view" on page 1499
Drag a node or a related resource into the Message Flow editor.

Related reference:

"WebSphere Message Broker Explorer and WebSphere Message Broker Toolkit keyboard shortcuts" on page 6828
You can navigate all interfaces in the WebSphere Message Broker Explorer and WebSphere Message Broker Toolkit by using the keyboard.

“Message Flow editor” on page 6810
The Message Flow editor is the default editor provided by the Broker Application Development perspective for defining a graphical representation of a message flow in the WebSphere Message Broker Toolkit, and for setting properties for individual message flow nodes.

“Message Flow editor menus” on page 6813

“Accessibility features for WebSphere Message Broker” on page 135
Accessibility features help users who have a physical disability, such as restricted mobility or limited vision, to use information technology products successfully.

Dragging a resource from the Broker Development view:

Drag a node or a related resource into the Message Flow editor.

Before you begin

Before you start:

Complete the following tasks:
- Create or open a message flow, as described in “Creating a message flow” on page 1431 and “Opening an existing message flow” on page 1433.
- Read concept information in “Message flow nodes” on page 1024.

About this task

To create a node, drag a resource from the Broker Development view to an empty canvas. To modify an existing node, drag a resource onto that node. The following resources are supported:
- An Adapter file
- An ESQL file
- A Java file
- A subflow
- A WSDL file
- An XSL file

Procedure
1. Open the message flow with which you want to work.
2. Drag one of the supported resources from the Broker Development view onto the canvas.
   - If you drop the resource on an empty canvas, a node is created and configured automatically.

     The following table shows the results when you drag a resource from the Broker Development view onto an empty canvas:

Resource	Node created	Property set
Adapter file	A “PeopleSoftInput node” on page 4630, “SAPInput node” on page 4676, or “SiebelInput node” on page 4740 is created	`Adapter component`
CORBA IDL file	A “CORBARequest node” on page 4349 is created	`IDL file`

Resource	Node created	Property set
ESQL file	A “Compute node” on page 4340 is created	`ESQL Module`
Java file	A “JavaCompute node” on page 4514 is created	`Java Class`
WSDL file	A “SOAPInput node” on page 4795 or “SOAPRequest node” on page 4828 is created	`WSDL file name`
XSL file	An “XSLTransform node” on page 4968 is created	`Stylesheet`

- If you drop the resource onto an existing node, the relevant node property is updated with the name of the resource file. For example, if you drop a Java file onto a JavaCompute node, the `Java Class` property is set to the class name of the Java file that you are dropping. If you drop an ESQL file over any node that uses ESQL, such as a Database node, the `ESQL Module` property is set.

Related concepts:

“Message flow nodes” on page 1024
A message flow node is a processing step in a message flow. It can be a built-in node, a user-defined node, or a subflow node.

Related tasks:

“Adding a message flow node” on page 1494
When you have created a message flow, add nodes to define its function.

“Adding a subflow” on page 1501
In a message flow, you can include an embedded message flow, also known as a *subflow*. For example, you might define a subflow that provides error handling, and include it in a message flow connected to a failure terminal on a node that can generate an error in some situations.

“Creating a message flow” on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

“Opening an existing message flow” on page 1433
Open an existing message flow to change or update its contents, or to add or remove nodes.

Related reference:

“Compute node” on page 4340
Use the Compute node to construct one or more new output messages.

“JavaCompute node” on page 4514
Use the JavaCompute node to work with messages by using the Java language.

“PeopleSoftInput node” on page 4630
Use the PeopleSoftInput node to interact with a PeopleSoft application.

“SAPInput node” on page 4676
Use the SAPInput node to accept input from an SAP application.

“SiebelInput node” on page 4740
Use the SiebelInput node to interact with a Siebel application.

“SOAPInput node” on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

"XSLTransform node" on page 4968
Use the XSLTransform node to transform an XML message to another form of message, according to the rules provided by an XSL (Extensible Stylesheet Language) style sheet, and to set the `Message domain`, `Message set`, `Message type`, and `Message format` for the generated message.

Adding a subflow

In a message flow, you can include an embedded message flow, also known as a *subflow*. For example, you might define a subflow that provides error handling, and include it in a message flow connected to a failure terminal on a node that can generate an error in some situations.

Before you begin

Before you start:

To complete this task, you must have created a message flow to use as a subflow, and a message flow in which to insert the subflow. For more details, see "Creating a message flow" on page 1431.

About this task

You can embed subflows into your message flow if either of the following statements is true:

- The flow that you want to embed is defined in the same message flow project.
- The flow is defined in a different message flow project, and you have specified the dependency of the current message flow project on that other project.

You can embed subflows into other subflows.

Procedure

To add a subflow to a message flow, complete the following steps:

1. Open the message flow in which you want to embed the subflow.
2. To add a subflow, drag the message flow that you want to add from the Broker Development view to the editor. Alternatively, click **Flow** > **Add subflow**, then select the flow that you want to add from the list. The embedded subflow is shown in the Message Flow editor as a single node with the terminals that represent the Input and Output nodes that you have included in the subflow.
3. Connect the subflow node to one or more of the nodes in the main message flow. For more details, see "Connecting message flow nodes" on page 1520.
4. To add and connect further subflow nodes, repeat steps 2 and 3.
5. To work with the contents of the subflow, double-click the subflow icon. The subflow opens in the Message Flow editor.

What to do next

You can also package your subflow as a user-defined node so that you can use it in a message flow; for more information, see "Using a subflow as a user-defined node" on page 3008.

Related concepts:

"Subflows" on page 1030
You can include subflows in your message flows in the same way as you include built-in or user-defined nodes. You can also connect subflows to other nodes in the

same way.

"Using a subflow as a user-defined node" on page 3008
You can develop a user-defined node that packages a subflow from scratch, in the same way that you can create any other user-defined node that has its implementation based on Java, or base it on an existing subflow.

Related tasks:

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

"Opening an existing message flow" on page 1433
Open an existing message flow to change or update its contents, or to add or remove nodes.

"Connecting message flow nodes" on page 1520
When you include more than one node in your message flow, you must connect the nodes to indicate how the flow of control passes from input to output. The nodes can be built-in nodes, user-defined nodes, or subflow nodes.

"Creating a user-defined node from a subflow" on page 3076
Create user-defined nodes either from scratch, or by using an existing subflow.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Renaming a message flow node

You can change the name of any type of node (a built-in node, user-defined node, or subflow node) to reflect its purpose.

Before you begin

Before you start:

Complete the following tasks:

- Create a message flow, as described in "Creating a message flow" on page 1431.
- Read the concept topic, "Message flow nodes" on page 1024.

About this task

When you first add a node to the canvas, the editor automatically assigns a name to the node; the name is highlighted, and you can change it by entering a name of your choice. If you do not change the default name at this time, you can change it later, as described in this topic. For example, you might include a Compute node to calculate the price of a specific part in an order; you could change the name of the node to Calculate_Price.

When you rename a node, use only the supported characters for this entity. The editor prevents you from entering unsupported characters.

To rename a node:

Procedure

1. Switch to the Broker Application Development perspective.
2. Open the message flow with which you want to work.
3. You can rename a node in three ways:
   - Right-click the node and click **Rename**. The name is highlighted; enter a name of your choice and press **Enter**.
   - Click the node to select it, then click the name of the node so that it is highlighted; enter a name of your choice and press **Enter**.
   - Click the node to select it, then on the Description tab of the Properties view, enter a name of your choice in the Node name field.

   The name that you enter must be unique in the message flow.

What to do next

If you generate ESQL code for a Compute, Database, or Filter node, the code is contained in a module that is associated with the node. The name of the module in the ESQL file must match the name specified for the module in the *ESQL Module* property of the corresponding node. You can modify the module name, and change it from its default value (which is the name of the message flow, concatenated with the name of the node with which the module is associated). However, ensure that the module in the ESQL file matches the node property.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

"Opening an existing message flow" on page 1433
Open an existing message flow to change or update its contents, or to add or remove nodes.

"Connecting message flow nodes" on page 1520
When you include more than one node in your message flow, you must connect the nodes to indicate how the flow of control passes from input to output. The nodes can be built-in nodes, user-defined nodes, or subflow nodes.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Configuring a message flow node

When you have included an instance of a node in your message flow, you can configure its properties to customize how it works.

Before you begin

Before you start:

- Read the concept topic about message flow nodes

- Add a node

Viewing the properties of a node:
About this task

To view a node's properties:

Procedure

1. In the Message Flow editor, open the message flow with which you want to work.
2. To open the Properties view, right-click a node and click **Properties**.
   The selected node's properties are displayed. You can edit the properties.

Editing the properties of a node:
About this task

Properties are organized into related groups and displayed on tabs. Each tab is listed on the left of the Properties view. Click each tab to view the properties that you can edit.

Procedure

- Every node has at least one tab, **Description**, where you can change the name of the node and enter short and long descriptions. The description fields are optional because they are used only for documentation purposes.
- If a property is mandatory (that is, one for which you must enter a value), the property name is marked with an asterisk, as shown in the following example:
  `Queue Name*` _____
- Many nodes have properties that require XPath expressions, typically to identify the location of a particular resource. For example, for the MQInput node, you can specify the location of the security token and password by using either ESQL or XPath expressions. For help in constructing an XPath expression, you can open the XPath Expression Builder by clicking **Edit** next to each XPath property. For more information about using XPath expressions, see "Using XPath" on page 1506.

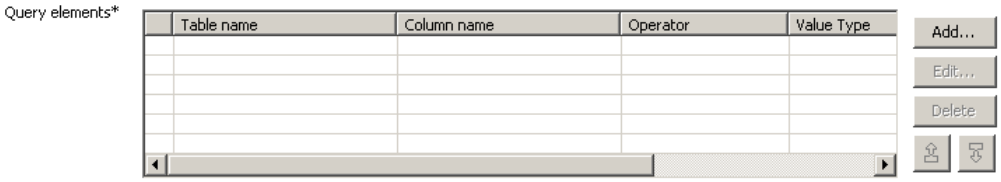
What to do next

For details of how to configure each individual built-in node, see the node description. You can find a list of the nodes, with links to the individual topics, in "Built-in nodes" on page 4293.

If you have included a user-defined node, refer to the documentation that came with the node to understand if, and how, you can configure its properties.

Editing complex properties:
About this task

A complex property is a property to which you can assign multiple values. Complex properties are displayed in a table in the Properties view, where you can add, edit, and delete values, and change the order of the values in the table. This example shows the `Query elements` complex property of the DatabaseRoute node.



Procedure

- To add a value to a complex property, click **Add**, enter the required fields in the dialog box that opens, then click **OK**. The values appear in the table. Repeat this step to enter as many values as are required.
- To edit a value, click any element in a row, click **Edit**, edit any of the values in the dialog box, then click **OK**.
- To delete a value, click any element in a row and click **Delete**. The entire row is deleted.
- To change the order of values in the table, click any element in a row and click the up icon  or down icon  to move the row.

Promoting properties:
About this task

You can promote node properties to their containing message flow; for more information, see “Promoting a property” on page 1298. Use this technique to set some values at the message flow level, without having to change individual nodes. This can be useful, for example, when you embed a message flow in another flow, and want to override some property such as output queue or data source with a value that is correct in this context. You cannot promote complex properties. For a full list of properties that are unavailable for promotion, as well as instructions for how to promote properties, see “Promoting a property” on page 1298.

Overriding properties at deployment time:
About this task

You can override a small number of node property values when you deploy a message flow. These property values are known as configurable properties, and you can use them to modify some characteristics of a deployed message flow without changing the message flow definitions. For example, you can update queue manager and data source information.

Even though you can set values for configurable properties at deployment time, you must set values for these properties within the message flow if they are mandatory. Each built-in node reference topic contains a table of properties, which identifies the configurable and mandatory properties.

What to do next

Next: connect the nodes.

Related concepts:

“Promoted properties” on page 1145
A promoted property is a message flow node property that has been promoted to the level of the message flow in which it is included.

"Packaging and deployment overview" on page 3210
Deployment is the process of transferring data to an execution group on a broker so that it can take effect in the broker. Message flows and associated resources are packaged in broker archive (BAR) files for deployment.

Related tasks:

"Promoting a property" on page 1298
You can promote a node property to the message flow level to simplify the maintenance of the message flow and its nodes, and to provide common values for multiple nodes in the flow by converging promoted properties.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

"Adding a message flow node" on page 1494
When you have created a message flow, add nodes to define its function.

"Defining a promoted property" on page 1297
When you create a message flow, you can promote properties from individual nodes in that message flow to the message flow level. Properties promoted in this way override the property values that you have set for the individual nodes.

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

Using XPath:

XPath provides an alternative method to ESQL for entering expressions in the property fields of specific built-in nodes.

About this task

In addition to ESQL as a message transformation language, WebSphere Message Broker supports an alternative expression grammar in property fields. For more information, see "ESQL-to-XPath mapping table" on page 5046.

You can use ESQL or XPath expressions in built-in nodes, within your message flows, to query or update message trees that are specified as accessible, and that you expect to be processed by a given node.

This section provides information on:
- "XPath overview" on page 1507
- "Namespace support" on page 1508
- "XPath Expression Builder" on page 1510
- "Creating XPath expressions" on page 1515
- "Selecting the grammar mode" on page 1516

There examples on configuring the XPath cache in the Configuring Applications section here: "Configuring the XPath cache" on page 765.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

XPath overview:

The XML Path Language (XPath) is used to uniquely identify or address parts of an XML document. An XPath expression can be used to search through an XML document, and extract information from any part of the document, such as an element or attribute (referred to as a *node* in XML) in it. XPath can be used alone or in conjunction with XSLT.

Some of the built-in nodes provided in the WebSphere Message Broker Toolkit can use XPath expressions to specify the part of a message that is processed by the node. For example, you can use an XPath expression to identify fields in a message and determine if they match a specified value, or to set the field value by updating it with the results of a database query.

You can use XPath 1.0 path expressions in your flow to access specific parts of an incoming message, create or locate parts of an outgoing message, and perform complex message processing that might involve values present in message trees accessible by a node so that you can transform, filter, or retrieve values from a message.

For example, the Route node applies XPath 1.0 general expressions to the content of message trees associated with the incoming message assembly for this node. Following evaluation of an expression the result is cast as a Boolean (true or false) result, and this in turn is used to determine if a copy of the incoming message is routed down an output terminal associated with the processed expression.

If you have XML schema definition (`.mxsd`) files present in your workspace, any elements, attributes or data types defined in such definitions can be loaded into the Data types viewer and selected to automatically generate a path expressions mapping to the definition concerned.

Equally, depending on the XPath expressions supported by the property concerned, you can select XPath functions and operators to include in an expression, or you can build your own expressions manually.

The Data types viewer contains a list of variables relating to trees that can be accessed by expressions for the node property concerned.

For example, `$InputRoot` gives access to the incoming message tree. The fixed format of standard folders you can expect to exist under this tree, for example, `Properties` and `MQMD` are described without the need to import an `.mxsd` definition for them. These structures can be navigated in the viewer and, on selection of an

element within them, a path expression that maps to the element in question is built automatically through the XPath 1.0 language.

For further information on XPath 1.0 see W3C XPath 1.0 Specification.

You can use the XPath Expression Builder to visually build XPath expressions to set the relevant properties in your nodes. You launch the XPath Expression Builder from buttons located along side property fields present in the Properties viewer, for those nodes that support the use of XPath expressions as property values.

The XPath files in WebSphere Message Broker are supplied in three property editors; see "XPath property editors" on page 5047 for more details.

The XPath editor supports both content-assist directly on the text field and also an **Edit...** button that launches the XPath builder dialog. The dialog gives you a larger area in which to build your XPath expressions.

The content assist based control contains two different proposal lists in the following order:

1. Nodes and Variables
2. Functions and Operators

The node and variable proposals are displayed the first time that you use the XPath editor. In this view, the status bar reads `Press` **`Ctrl+Space`** `to show Function and Operation Proposals`.

Pressing **`Ctrl+Space`** when you are in the function and operator level proposals selects the Node and Variable proposals.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

Namespace support:

The XPath Expression builder provides qualified support for namespaces.

The XPath Expression builder dialog contains a namespace settings table that:

- Supports simplified expressions that enable qualified namespace matching at run time

- Can be auto-generated based on imported schema definitions and generated expressions (based on selections made in the dialog when you build an expression)
- Allows you to add your own entries to the table

The table encapsulates deployable data passed to the runtime environment, as part of the nodes attribute data, and is used by the node to modify expressions through prefix-to-URI substitution. The final expressions support namespace matching, because they are processed against a target tree when employed by their associated message processing engine, that is, the XPath 1.0 runtime engine or ESQL runtime engine.

When you enter an ESQL field reference expression in a read-only or read-write path field, or an XPath 1.0 path expression in a read-only or read-write path field, or a general expression field (general expressions can contain zero or more path expressions), WebSphere Message Broker understands the language from the syntax you use.

XPath is the default for general expression fields that are validated by ensuring they conform to the XPath 1.0 grammar. For path expression fields XPath is assumed if the expression is valid and begins with a $ sign.

The language you can use is dictated by the property editor currently in use for a node's property field.

Namespace prefixes are used in an XPath or ESQL expression to make the statements shorter and easier to understand, while still supporting the ability to qualify an element name match by also matching on its associated namespace URI.

For example, consider the following message, where namespace prefix `b` is overridden through an inner declaration:

```
<b:a xmlns:b='xyz'>
 <!-- the namespace of elements 'a' and 'c' using prefix 'b'  is xyz -->
 <b:c>
  <b:d  xmlns:b='qrs'>
   <!-- the namespace of elements 'd' and 'e' using prefix 'b' is now qrs -->
   <b:e>100</b:e>
  </b:d>
 </b:c>
</b:a>
```

Note that the scope of a namespace declaration declaring a prefix extends from the beginning of the start tag in which it appears to the end of the corresponding end tag, excluding the scope of any inner declarations with the same namespace prefix. In the case of an empty tag, the scope is the tag itself: `>`.

To navigate to element `e` in the above message use the following human-readable XPath expression:

```
/b:a/b:c/b2:d/b2:e
```

Note, that to prevent the auto-generated prefix to the URI map produced in the expression dialog overloading the same prefix (in this case `b`), the inner `b` prefix is appended with a numeric value to distinguish it from the outer `b` prefix. This strategy is repeated for each prefix name clash.

This notation is similar to the equivalent human-readable ESQL expression:

```
Root.b:a.b:c.b2:d.b2:e
```

To support namespace prefixes within expressions, the XPath Expression Builder Dialog automatically generates a prefix to a URI namespace settings table (based on the content of imported schema definitions, through which expressions are generated) .

Without the use of namespace prefixes to URI mapping data in this table, the runtime environment would be forced to take a less efficient approach, where portable but verbose XPath expressions would be required by it to provide namespace matching support.

The previous expression:

```
/b:a/b:c/b2:d/b2:e
```

would take the form:

```
/*[namespace-uri()='xyz' and local-name()='a']/*[namespace-uri()='xyz'
   and local-name()='c']/*[namespace-uri()='qrs' and
   local-name()='d']/*[namespace-uri()='qrs' and local-name()='e']
```

Related tasks:

"Using XPath" on page 1506
XPath provides an alternative method to ESQL for entering expressions in the property fields of specific built-in nodes.

"Writing ESQL" on page 2413
How you can use ESQL to customize nodes.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"Predefined XPath mapping functions" on page 5001
Some XPath functions are available for use with message maps.

XPath Expression Builder:

You can launch the XPath Expression Builder from most property fields that support, or expect, XPath expressions as a value that can be entered into the field.

The use of the XPath Expression Builder is optional, in that it is an aid to you in developing message flow applications. The XPath Expression Builder helps you to construct message processing expressions in either XPath or ESQL. You are free to enter expressions by hand, or use the XPath Expression Builder to help construct such expressions.

The XPath Expression Builder does not support use of the `$Body` variable. You can use the `$Body` variable when you enter an expression by hand, but the XPath Expression Builder and associated validation in the WebSphere Message Broker Toolkit do not support it. Use the `$Root` variable instead.

You can populate the fields, regardless of the state of the node; that is, whether the node is detached or connected, or fully, partially, or completely unconfigured.

You launch the XPath Expression Builder from a button in the following locations:

- Table cells, located to the right of the text entry field within the cell.
- **Add** or **Edit** dialog boxes used to construct rows in tables, located to right of the property field concerned.
- Tabs in the property viewer for a node, to the right of a property field.

Variables (or in ESQL terminology, correlation names) provide a list of all message tree start points that are applicable to the property field from which the dialog was launched.

If a field is a read-only or a read-write path field, expressions must start with such a variable to indicate which tree in which message assembly the path expression is mapping to.

XPath variable names map to existing correlation names found in ESQL field reference expressions, but to conform to the ESQL grammar they are designated as variable references by prefixing them with the dollar ($) character.

For example:

ESQL `Root.XMLNSC.CUST_DETAILS.NAME`

XPATH
`$Root/XMLNSC/CUST_DETAILS/NAME`

The variable indicates to which tree and where in that tree the expression is anchored.

The XPath Expression Builder dialog box supports validation, which you can turn off on the XPath preferences page by clearing the **Validate when creating XPath expressions** check box.

If you select the `$Root` or `$Body` variables and create an expression that refers to the body of the message, the XPath expression contains the message element. This expression is correct for message bodies owned by the XMLNSC, XMLNS, XML, and DataObject domains.

For message bodies that are owned by the MRM, MIME, SOAP, and IDOC domains, you must remove the message element from the expression.

For example, the XPath expression `$Body/my_message/my_field` is correct for XMLNSC, but must be changed to `$Body/my_field` to be correct for MRM.

Views

There are three main views when functions are supported.

Whether a view is displayed, and what is displayed in it, depends on what type of property editor you have used to launch the dialog, and its tailored settings; for example, for path type fields you do not see a functions pane. The operators that are supported can change as can the list of applicable variables.

Data Types Viewer
This view shows the different schema types, elements, and attributes that you can use within the XPath expression that you are creating, as well as the allowable variable references.

XPath Function
This view shows four main top level categories, which are:

String This category corresponds to the description in the XPath 1.0 specification of `section-String-Functions`.

Boolean
This category corresponds to the description in the XPath 1.0 specification of `section-Boolean-Functions`.

Numeric
This category corresponds to the description in the XPath 1.0 specification of `section-Number-Functions`.

Nodeset
This category corresponds to the description in the XPath 1.0 specification of `section-Node-Set-Functions`.

For information on the format of XPath 1.0 expressions see the W3C XPath 1.0 Specification.

Operators
This view shows a list of all of the available operators that you can use within the given XPath expression.

Namespace settings

If you expand **Namespace settings** in the XPath Expression Builder dialog you see a table of Prefix and Namespace pair strings. This table is automatically updated when XPath expressions are created. If the default prefix generated is not what you want, you can change it by clicking **Change Prefix**.

To add a prefix and namespace map entry click **Add** and complete the fields in the dialog.

To edit or delete an entry in the table, select the item and click **Edit** or **Delete** respectively.

Edit opens another field dialog allowing you to change the prefix and namespace.

For information about the preferences supplied with the XPath editor, see "XPath editor preferences" on page 1513.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

Related tasks:

"Writing ESQL" on page 2413
How you can use ESQL to customize nodes.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"Predefined XPath mapping functions" on page 5001
Some XPath functions are available for use with message maps.

XPath editor preferences:

This topic describes the possible options available to you when you use the XPath editor.

The following lists describe the custom preferences used in the XPath editor.

- Validation option:

  Validation when building XPath expressions
  Default: checked.

  This option is used to perform validation each time you update the XPath expression. As validation requires re-parsing the expression against the XML Schema document you can turn this option off, for example, if you are dealing with very large complex XPath expressions.

- Content Assist display options:

  Show XML Schema model groups
  Default: not checked.

  This option allows you to view XML schema model groups, or not.

  Show type in XML Schema tree
  Default: checked.

  This option allows you to view the `<type name>` in both the Content Assist view and the XPath expression builder, or not.

  Show function parameters
  Default: checked.

  This option allows you to have function parameters shown, or not.

  Show function return type
  Default: checked.

  This option allows you to have function return types shown, or not.

  Show content assist description
  Default: checked.

  This option allows you to view the description of a given selected entry in the Content Assist view, or not..

- Auto-Activation option:

  Enable auto activation
  Default: checked.

  When this option is active, the Content Assist field appears whenever the cursor is after one of the following:

  – / - Forward slash character
  – [- left bracket character
  – (- Left parentheses character
  – , - Comma character

  You set the delay time, before the Content Assist field appears, in the **Auto activation delay** field. The time is in milliseconds and the range is a positive number between zero and 9999.

- Content assist color options:

This preference allows you to customize the background and foreground colors for the Content Assist fields. The default background color is (red, green, blue - 254, 241, 233) and the default foreground color is (red, green, blue - 0, 0, 0) - black.

Related tasks:

"Using XPath" on page 1506
XPath provides an alternative method to ESQL for entering expressions in the property fields of specific built-in nodes.

Related reference:

"XPath Expression Builder" on page 1510
You can launch the XPath Expression Builder from most property fields that support, or expect, XPath expressions as a value that can be entered into the field.

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

XPath expressions supported by default:

XPath supports a number of expressions by default.

The following expressions are supported:

Read-only fields

```
$Root, $Body, $Properties, $LocalEnvironment,  $DestinationList,
$ExceptionList, $InputRoot , $InputBody, $InputProperties,
$InputLocalEnvironment, $InputDestinationList,
$InputExceptionList,$Environment.
```

To exclude variables for a node property from the default list, specify a comma separated string of variables. For example, specifying:

```
"com.ibm.etools.mft.ibmnodes.editors.xpath.XPathReadOnlyPropertyEditor", "InputRoot ,
 InputBody, InputProperties, InputLocalEnvironment, InputDestinationList,
 InputExceptionList"
```

restricts the XPath field to support only:

```
$Root, $Body, $Properties, $LocalEnvironment,  $DestinationList, $ExceptionList'
$Environment
```

Read-write fields

```
$InputRoot , $InputBody, $InputProperties, $InputLocalEnvironment,
$InputDestinationList, $InputExceptionList,  $OutputRoot ,
$OutputLocalEnvironment, $OutputDestinationList,
$OutputExceptionList, $Environment.
```

To exclude variables for a node property from the default list, specify a comma separated string of variables. For example, specifying:

```
"com.ibm.etools.mft.ibmnodes.editors.xpath.XPathReadWritePropertyEditor", "InputRoot ,
 InputBody, InputProperties, InputLocalEnvironment, InputDestinationList,
 InputExceptionList"
```

restricts the XPath field to support only:

```
$OutputRoot, $OutputLocalEnvironment, $OutputDestinationList, $OutputExceptionList,
 $Environment
```

Expression fields

```
$Root, $Body, $Properties, $LocalEnvironment,  $DestinationList,
$ExceptionList, $InputRoot , $InputBody, $InputProperties,
$InputLocalEnvironment, $InputDestinationList, $InputExceptionList,
```

```
$OutputRoot , $OutputLocalEnvironment, $OutputDestinationList,
$OutputExceptionList, $Environment.
```

To exclude variables for a node property from the default list, specify a comma separated string of variables. For example, specifying:

```
"com.ibm.etools.mft.ibmnodes.editors.xpath.XPathPropertyEditor", "InputRoot ,
InputBody, InputProperties, InputLocalEnvironment, InputDestinationList,
InputExceptionList, OutputRoot , OutputLocalEnvironment, OutputDestinationList,
OutputExceptionList"
```

restricts the XPath field to support only:

```
$Root, $Body, $Properties, $LocalEnvironment,  $DestinationList, $ExceptionList,
 $Environment.
```

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"Predefined XPath mapping functions" on page 5001
Some XPath functions are available for use with message maps.

Creating XPath expressions:

A number of built-in primitive nodes have properties that can be specified using an XPath 1.0 expression; most commonly where this language is used to form a path expression to locate incoming message body elements received by a node.

About this task

Other less common node property fields support the entry of general XPath 1.0 expressions that support a wider range of the language to perform more complex evaluations in the broker's XPath 1.0 runtime engine.

The XPath Expression builder provides a tree view of a message, and supports the automatic generation of an XPath 1.0 path expression, through the selection of an element within the tree.

The Schema Viewer section provides a tree view of the input message. To visually build your XPath expression follow these steps:

Procedure

1. Add the relevant node to your message flow
2. In the Properties viewer, enter the correlation name, or press Ctrl + Space to use content assist, or press Edit to use the Expression editor. Content assist is also invoked by simply typing $ in cell-based property fields. See "Correlation names" on page 1069 for further information on correlation names.
3. Expand the tree, navigate to the field for which you want to build an expression, and click to select it. A field is either an element, or an attribute. Double click the field to add it to the XPath expression. You can also drag fields, functions, and operators to the desired location in the XPath expression when using the XPath Expression builder.
4. To set conditions, enter them as you would a normal XPath Expression.

Results

The complete XPath expression is shown either:

- In the XPath Expression pane if you are using the XPath Expression builder.

  The Expression builder dialog is an optional aid for generating expressions that, when complete, form the value in a node's property field.

  If you do not use the Expression builder dialog, the expressions entered manually are validated using the property editor.
- In the Property field if it is in the node itself.

Messages are displayed at the top of the XPath Editor window to alert you to the fact that a path or expression you have entered is not valid.

Note: The editor does not prevent you from entering, and saving, an expression that is not valid.

Example

Here is the XPath expression built in the XPath Expression Builder to filter the Employee business objects for all employees who are managers: `$Root/XMLNSC/getEmployeeInfo/Emp[isManager=true()]`.

- `$Root/XMLNSC/`: The body section of the message; that is, the last child of root. This example assumes that the XMLNSC domain is being used.
- `/getEmployeeInfo`: The name of the operation in the interface.
- `/Emp`: The name of the input message type.
- `[isManager=true()]`: Checks whether the `isManager` field is set to true.

In this case the same expression works for both request and response flows, because the input and output messages for the operation are identical.

For more information on XPath 1.0, see W3C XPath 1.0 Specification.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

Related tasks:

"Writing ESQL" on page 2413
How you can use ESQL to customize nodes.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"Predefined XPath mapping functions" on page 5001
Some XPath functions are available for use with message maps.

Selecting the grammar mode:

The grammar mode allows you to use only a restricted set of expressions, in either XPath or ESQL, and checks whether the syntax you have entered is valid.

About this task

WebSphere Message Broker supports the following field categories:

- Read-only path field
- Read-write path field
- Expression field

Each of these field types can be either fixed or mixed language, that is, ESQL, XPath, or either.

If you use XPath syntax, and the expressions are not supported for the property you are using, the syntax is rejected during the validation process.

ESQL and XPath have similar restrictions on the syntax that is permitted for the first two of these field types. There are restrictions to the expression fields as well, but as this type of field supports general expressions that can be used in either language, the range of syntax available is greater than in the first two.

WebSphere Message Broker uses code assistance in the grammar management of XPath 1.0 to validate the syntax of expressions you enter. This assistance is always available, regardless of the grammar mode you are using.

By default, you are operating in the restricted grammar mode.

Code assistance enables you to construct syntactically correct expressions but it does not validate those expressions. Validation is performed by property editors in which such expressions are entered.

If you attempt to use an expression that is not valid, the property editor marks it as such, either from a syntax or schema validation perspective.

You receive error or warning messages depending on the preference choices you set in **Windows>Preferences>Broker Development>XPath>Validation** .

If, under the above validation settings, particular checks are to be marked as errors, error markers are shown in the problems viewer. This behavior results in a message flow being marked as broken, and it cannot then be imported into, or compiled in, a deployable broker archive (BAR) file using the Broker Archive editor.

If you want to use the appropriate unrestricted grammar to input a specific field type, property editors do not force restricted forms of ESQL or XPath 1.0 expressions for such fields that expect them. Instead, you can enter the full range of syntax in the context of the field category concerned, namely, path or general expression, without the validation checks being applied. This means that if you need to, you can deploy the full range of syntax supported by the ESQL or XPath 1.0 runtime environment. Note, however, that such expressions might not be in a form that can be converted to the other language.

To use unrestricted grammar, carry out the following procedure:

Procedure

1. Click **Window -> Preferences** and expand **Broker Development**.
2. Expand **XPath** and click **Grammar**.
3. Clear the **Use XPath and ESQL equivalent grammar** check box.

What to do next

Note that expressions are still checked for valid syntax appropriate in the context of the field type, but you can now use the full range of grammar supported by the runtime environment.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related reference:

"Predefined XPath mapping functions" on page 5001
Some XPath functions are available for use with message maps.

Using dynamic terminals

You can add, rename, and remove dynamic terminals on a node in the Message Flow editor.

Before you begin

Before you start:

- Add a node that supports dynamic terminals; for more details, see "Adding a message flow node" on page 1494 and "Message flow node terminals" on page 1034.

About this task

Some message flow nodes support dynamic input or output terminals, including the Collector, Route, and DatabaseRoute nodes. When you have added a node to the flow editor, you can add, remove, or change dynamic terminals.

Procedure

- **Adding a dynamic terminal**
  1. Right-click the node and click **Add Input Terminal** or **Add Output Terminal**.
  2. Enter a name for the new terminal and click **OK**. The name must be unique for the terminal type. For example, if an input terminal called *In* already exists, you cannot create a dynamic input terminal with the name *In*.

     The new terminal is displayed on the node. If a node has five or more terminals, they are displayed as a terminal group. The following example shows a Route node with more than four output terminals.  To connect a particular output terminal, click the terminal group to open the Terminal Selection dialog box, or right-click the node and select **Create Connection**.
- **Renaming a dynamic terminal**
  1. Right-click the node and click **Rename Input Terminal** or **Rename Output Terminal**. These options are available only if you have added one or more appropriate terminals to this node.

2. Select from the list the name of the terminal that you want to change. Only dynamic terminals are listed because you cannot change the name of a static terminal.
3. Enter a new name for the terminal and click **OK**. Do not rename a dynamic terminal if one of the node properties is configured to use that name.

- **Removing a dynamic terminal**
  1. Right-click the node and click **Remove Input Terminal** or **Remove Output Terminal**, These options are available only if you have added one or more appropriate terminals to this node.
  2. Select from the list the name of the terminal that you want to remove and click **OK**. Only dynamic terminals are listed because you cannot remove a static terminal. Do not remove a dynamic terminal if one of the node properties is configured to use that terminal.

What to do next

When you have added dynamic terminals to a node, connect them to other nodes in the message flow; for more information, see "Connecting message flow nodes" on page 1520.

Related concepts:

"Message flow node terminals" on page 1034
A terminal is the point at which one node in a message flow is connected to another node.

Related tasks:

"Adding a message flow node" on page 1494
When you have created a message flow, add nodes to define its function.

"Connecting message flow nodes" on page 1520
When you include more than one node in your message flow, you must connect the nodes to indicate how the flow of control passes from input to output. The nodes can be built-in nodes, user-defined nodes, or subflow nodes.

Related reference:

"Collector node" on page 4333
Use the Collector node to create message collections based on rules that you configure in the node.

"Route node" on page 4669
Use the Route node to direct messages that meet certain criteria down different paths of a message flow.

"DatabaseRoute node" on page 4373
Use the DatabaseRoute node to route messages using information from a database in conjunction with XPath expressions.

Removing a message flow node

When you have created and populated a message flow, you might need to remove a node to change the function of the flow, or to replace it with another more appropriate node. The node can be a built-in node, a user-defined node, or a subflow node.

Before you begin

Before you start:

This topic assumes that you have completed one or more of the following tasks.

- Create a message flow, or open an existing message flow, as described in "Creating a message flow" on page 1431 and "Opening an existing message flow" on page 1433.
- Add a node to the message flow, as described in "Adding a message flow node" on page 1494.
- Add a subflow, as described in "Adding a subflow" on page 1501.
- Read background information about nodes in "Message flow nodes" on page 1024.

About this task

To remove a node from a message flow, complete the following steps.

Procedure

1. Open the message flow with which you want to work.
2. Highlight the node that you want to remove and click **Edit** > **Delete**.

   The node is removed from the flow. If you have created any connections between that node and any other node, those connections are also deleted when you delete the node.
3. If you delete a node in error, you can restore it by clicking **Edit** > **Undo Delete** or pressing Ctrl+Z. The node and its connections, if any, are restored.
4. If you undo the deletion, but decide that it is correct to delete the node, click **Edit** > **Redo Delete**.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

"Opening an existing message flow" on page 1433
Open an existing message flow to change or update its contents, or to add or remove nodes.

"Adding a message flow node" on page 1494
When you have created a message flow, add nodes to define its function.

"Removing a node connection" on page 1526
The message flow editor displays the nodes and connections in the editor view. You can remove connections to change the way in which the message flow processes messages.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Connecting message flow nodes

When you include more than one node in your message flow, you must connect the nodes to indicate how the flow of control passes from input to output. The nodes can be built-in nodes, user-defined nodes, or subflow nodes.

Before you begin

Before you start:

Complete the following tasks.

- Add a note to your message flow, as described in "Adding a message flow node" on page 1494.
- Add a subflow to your message flow, as described in "Adding a subflow" on page 1501.
- Read the concept topic, "Message flow connections" on page 1032.

About this task

Your message flow might contain just one MQInput node, one Compute node, and one MQOutput node. Or it might involve a large number of nodes, and perhaps embedded message flows, that provide a number of paths through which a message can travel depending on its content. You might also have some error processing routines included in the flow. You might also need to control the order of processing.

You can connect a single output terminal of one node to the input terminal of more than one node (this is known as fan-out). If you do this, the same message is propagated to all target nodes, but you have no control over the order in which the subsequent paths through the message flow are executed (except with the FlowOrder node).

You can also connect the output terminal of several nodes to a single node input terminal (this is known as fan-in). Again, the messages that are received by the target node are not received in any guaranteed order.

When you have completed a connection, it is displayed as a black line, and is drawn as close as possible to a straight line between the connected terminals. This behavior might result in the connection passing across other nodes. To avoid this problem, you can add bend points to the connection.

In the Message Flow editor, you can display node and connection metadata by holding the mouse pointer over a node or subflow in a message flow. To view metadata information for a node, subflow, or connection:

1. Switch to the Broker Application Development perspective.
2. Open a message flow.
3. In the Message Flow editor, hold the mouse pointer over a node, a subflow, or a node connection in the open message flow by placing the mouse over the element.

A custom tooltip is displayed below the element.

- To turn the pop-up window into a scrollable window, press **F2**.
- To hide the pop-up window, either press Esc or move the mouse away from the node.

If you define a complex message flow, you might have to create a large number of connections. The principle is the same for every connection. You create connections either by using the mouse, or by using the Terminal Selection dialog. See "Creating node connections with the mouse" on page 1522 and "Creating node connections with the Terminal Selection dialog box" on page 1523 for more information.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

"Adding a message flow node" on page 1494
When you have created a message flow, add nodes to define its function.

"Removing a node connection" on page 1526
The message flow editor displays the nodes and connections in the editor view. You can remove connections to change the way in which the message flow processes messages.

"Adding a bend point" on page 1527
When you are working with a message flow, and connecting your chosen nodes together to determine the flow of control, you might find that a connection that you have made crosses over an intervening node and makes the flow of control difficult to follow. To help you to display the message flow nodes and their connections in a clear way, you can add bend points to the connections that you have made to improve the organization of the display. The addition of bend points has no effect on the execution of the nodes or the operation of the message flow.

"Removing a bend point" on page 1528
When you are working with a message flow in the editor view, you might want to simplify the display of the message flow by removing a bend point that you previously added to a connection between two nodes.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Creating node connections with the mouse:

Use the mouse to connect one node to another.

Before you begin

Before you start:

Read the concept topic about connections.

About this task

Procedure

1. Switch to the Broker Application Development perspective.
2. Open the message flow with which you want to work.
3. Click the terminal from which the connection is to be made; that is, the terminal from which the message is propagated from the current node.

For example, you can click the Failure, Out, or Catch terminal of the MQInput node. Hold the mouse pointer over each terminal to see the name of the terminal. You do not need to keep the mouse button pressed.

Alternatively, click **Connection** on the palette, then click the node from which the connection is to be made. The Terminal Selection dialog box opens for you to choose the terminal from which to make a connection. Click **OK**. If a node has five or more input or output terminals (for example, if you have added dynamic terminals), they are displayed in a group. The following example shows a node with more than four output nodes.  To select a particular output terminal, click the grouped output terminal to open the Terminal Selection dialog box.

4. Click the input terminal of the next node in the message flow (to which the message passes for further processing). The connection is made when you click a valid input terminal. The connection appears as a black line between the two terminals.

Results

In the Message Flow editor, you can display node and connection metadata by holding the mouse pointer over a node or subflow in a message flow. To view metadata information for a node, subflow, or connection:

1. Switch to the Broker Application Development perspective.
2. Open a message flow.
3. In the Message Flow editor, hold the mouse pointer over a node, a subflow, or a node connection in the open message flow by placing the mouse over the element.

A custom tooltip is displayed below the element.

- To turn the pop-up window into a scrollable window, press **F2**.
- To hide the pop-up window, either press Esc or move the mouse away from the node.

What to do next

Next: add a bend point, as described in "Adding a bend point" on page 1527.

Related concepts:

"Message flow node terminals" on page 1034
A terminal is the point at which one node in a message flow is connected to another node.

Related tasks:

"Connecting message flow nodes" on page 1520
When you include more than one node in your message flow, you must connect the nodes to indicate how the flow of control passes from input to output. The nodes can be built-in nodes, user-defined nodes, or subflow nodes.

"Creating node connections with the Terminal Selection dialog box"
Use the Terminal Selection dialog box to connect one node to another.

Creating node connections with the Terminal Selection dialog box:

Use the Terminal Selection dialog box to connect one node to another.

Before you begin

Before you start:

Read the concept topic about connections.

About this task

1. Switch to the Broker Application Development perspective.
2. Open the message flow with which you want to work.
3. Click **Connection** above the node palette.
4. Click the node from which you want the connection to be made. The Terminal Selection dialog box is displayed.
5. Select the terminal from the list of valid terminals on this node. Click **OK**. The dialog box closes.
6. Click the node to which to make the connection. If this node has only one input terminal, the connection is made immediately. If this node has more than one input terminal, the Terminal Selection dialog box is displayed again, listing the input terminals of the selected node. Click the correct terminal and click **OK**.

Alternatively, you can make a connection in the following way:

Procedure

1. Click **Selection** above the node palette.
2. Right-click the node from which you want to make the connection and click **Create Connection**. The Terminal Selection dialog box is displayed.
3. Select the terminal from the list of valid terminals on this node. Click **OK**. The dialog box closes.
4. Click the node to which to make the connection. If this node has only one input terminal, the connection is made immediately. If this node has more than one input terminal, the Terminal Selection dialog box is displayed again, listing the input terminals of the selected node. Click the correct terminal and click **OK**.

Results

In the Message Flow editor, you can display node and connection metadata by holding the mouse pointer over a node or subflow in a message flow. To view metadata information for a node, subflow, or connection:

1. Switch to the Broker Application Development perspective.
2. Open a message flow.
3. In the Message Flow editor, hold the mouse pointer over a node, a subflow, or a node connection in the open message flow by placing the mouse over the element.

A custom tooltip is displayed below the element.

- To turn the pop-up window into a scrollable window, press **F2**.
- To hide the pop-up window, either press Esc or move the mouse away from the node.

What to do next

Next: add a bend point, as described in “Adding a bend point” on page 1527.

Related concepts:

“Message flow node terminals” on page 1034
A terminal is the point at which one node in a message flow is connected to another node.

Related tasks:

“Connecting message flow nodes” on page 1520
When you include more than one node in your message flow, you must connect the nodes to indicate how the flow of control passes from input to output. The nodes can be built-in nodes, user-defined nodes, or subflow nodes.

“Creating node connections with the mouse” on page 1522
Use the mouse to connect one node to another.

Inserting nodes into existing message flows

You can insert a node into an existing message flow without having to delete existing connections and create connections for the new node.

Before you begin

Before you start:

The following instructions assume that you have created a message flow, as described in “Creating a message flow” on page 1431.

About this task

In earlier versions of WebSphere Message Broker, to insert a new node into a message flow, you had to delete the existing connection between two nodes, insert the new node, then create new connections between the existing nodes and the new node. From WebSphere Message Broker Version 7.0 onwards, you can insert a new node in one step by dropping it onto an existing connection, as described by the following steps.

Procedure

1. In the Message Flow editor, open the message flow into which you want to insert a new node.
2. Choose the node that you want to insert from the message flow node palette.
3. Drag the node onto the connection between the two nodes where you want the new node to be added. When the mouse pointer is over the connection, a label is displayed saying "Add a node here". When you release the mouse button, the node is inserted in the message flow and is automatically connected to the preceding and succeeding nodes.

Related concepts:

“Message flow node palette” on page 1027
The palette in the Message Flow editor contains all the built-in nodes, which are organized into categories, or drawers. A drawer is a container for a list of icons, such as the **Favorites** drawer.

“Managing message flow resources” on page 1424
Manage your message flows and associated resources in the WebSphere Message Broker Toolkit.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related reference:

"Message Flow editor" on page 6810
The Message Flow editor is the default editor provided by the Broker Application Development perspective for defining a graphical representation of a message flow in the WebSphere Message Broker Toolkit, and for setting properties for individual message flow nodes.

Removing a node connection

The message flow editor displays the nodes and connections in the editor view. You can remove connections to change the way in which the message flow processes messages.

Before you begin

Before you start:

Complete the following tasks:
- "Connecting message flow nodes" on page 1520.
- Read the concept topic, "Message flow connections" on page 1032.

About this task

To remove a connection that you have created between two nodes:

Procedure

1. Open the message flow that you want to work with.
2. Click **Selection** above the node palette.
3. Select the connection that you want to delete. When you hold the mouse pointer over the connection, the editor highlights the connection that you have selected by thickening its line, adding an arrowhead at the target terminal end, and annotating the connection with the name of the two terminals connected, for example Out->In.

   When you select the connection, the editor appends a small black square at each end and at every bend point of the connection, and a small arrowhead at the target terminal end. The annotation disappears when you select the connection.
4. Check that the selected connection is the one that you want to delete.
5. Right-click the connection and click **Delete**, press the Delete key, or click **Edit** > **Delete**. If you want to delete further connections, repeat these actions from step 3.
6. If you delete a connection in error, you can restore it by right-clicking in the editor view and clicking **Undo Delete**. The connection is restored.
7. If you undo the delete, but decide that it is the correct delete action, you can right-click in the editor view and click **Redo Delete**. You can also delete a connection by selecting it in the Outline view and pressing the Delete key.

Results

If you delete a node, its connections are automatically removed; you do not have to do this as a separate task.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

"Adding a message flow node" on page 1494
When you have created a message flow, add nodes to define its function.

"Connecting message flow nodes" on page 1520
When you include more than one node in your message flow, you must connect the nodes to indicate how the flow of control passes from input to output. The nodes can be built-in nodes, user-defined nodes, or subflow nodes.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Adding a bend point

When you are working with a message flow, and connecting your chosen nodes together to determine the flow of control, you might find that a connection that you have made crosses over an intervening node and makes the flow of control difficult to follow. To help you to display the message flow nodes and their connections in a clear way, you can add bend points to the connections that you have made to improve the organization of the display. The addition of bend points has no effect on the execution of the nodes or the operation of the message flow.

Before you begin

Before you start:

- Connect the nodes
- Read the concept topic about bend points

About this task

To add a bend point:

Procedure

1. Switch to the Broker Application Development perspective.
2. Open the message flow that you want to work with.
3. Click **Selection** above the node palette.
4. Select the connection to which you want to add a bend point. The editor appends a small black square to each end of the connection to highlight it.
   a. Check that this is the correct connection. The editor also adds a small point (a handle) in the connection halfway between the in and out terminals that are joined by this connection.
5. Hold the mouse pointer over this point until the editor displays a black cross to indicate that you now have control of this bend point.

a. Hold down the left mouse button and move your mouse to move the black cross and bend point across the editor view.

6. As you drag your mouse, the connection is updated, retaining its start and end points with a bend point at the drag point. You can move the bend point anywhere in the editor view to improve the layout of your message flow.
7. Release the mouse button when the connection is in the correct place. The editor now displays the bend point that you have created with a small square (such as those at the ends of the connection), and displays another two small points in the connection, one between your newly-created bend point and the out terminal, the other between the new bend point and the in terminal.

Results

If you want to add more than one bend point to the same connection, repeat these actions from step 4 on page 1527 using the additional small points inserted into the connection.

What to do next

Next: align and arrange the nodes.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Connecting message flow nodes" on page 1520
When you include more than one node in your message flow, you must connect the nodes to indicate how the flow of control passes from input to output. The nodes can be built-in nodes, user-defined nodes, or subflow nodes.

"Removing a node connection" on page 1526
The message flow editor displays the nodes and connections in the editor view. You can remove connections to change the way in which the message flow processes messages.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Removing a bend point

When you are working with a message flow in the editor view, you might want to simplify the display of the message flow by removing a bend point that you previously added to a connection between two nodes.

Before you begin

Before you start:

- Add a bend point
- Read the concept topic about bend points

About this task

To remove a bend point:

Procedure

1. Switch to the Broker Application Development perspective.
2. Open the message flow that you want to work with.
3. Click **Selection** above the node palette.
4. Select the connection from which you want to remove the bend point. The editor highlights the connection and its current bend points by thickening its line and appending a small black square to each end of the connection, and by indicating each bend point with a small black square. Check that this is the correct connection.
5. Right-click over the selected connection, if you added this bend point in the current edit session.
   a. Click **Undo Create Bend Point**.

   The editor removes the selected bend point.

   If you right-click in the editor view without a connection being selected, you can also click **Undo Create Bend Point** from the menu. However, this removes the last bend point that you created in any connection, which might not be the one that you want to remove.
6. Move the bend point to straighten the line if you added this bend point in a previous edit session, because you cannot use the undo action. When the line is straight, the bend point is removed automatically.

   When the bend point has been removed, the connection remains highlighted. Both ends of the connection, and any remaining bend points, remain displayed as small black squares. The editor also inserts small points (handles) into the connection between each bend point and between each terminal and its adjacent bend point, which you can use to add more bend points.
7. If you want to remove another bend point from the same connection, repeat these actions from step 4.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Connecting message flow nodes" on page 1520
When you include more than one node in your message flow, you must connect the nodes to indicate how the flow of control passes from input to output. The nodes can be built-in nodes, user-defined nodes, or subflow nodes.

"Adding a bend point" on page 1527
When you are working with a message flow, and connecting your chosen nodes together to determine the flow of control, you might find that a connection that you have made crosses over an intervening node and makes the flow of control difficult to follow. To help you to display the message flow nodes and their connections in a clear way, you can add bend points to the connections that you have made to improve the organization of the display. The addition of bend points has no effect on the execution of the nodes or the operation of the message flow.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Aligning and arranging nodes

When you are working in the Message Flow editor, you can decide how your nodes are aligned in the editor view.

Before you begin

This option is closely linked to the way in which your nodes are arranged. The default for both alignment and arrangement is left to right, which means that the in terminal of a node appears on its left edge, and its out terminals appear on its right edge. You can also change this characteristic of a node by rotating the icon display to right to left, top to bottom, and bottom to top.

Before you start

To complete this task, you must have completed the following task:

- "Adding a message flow node" on page 1494

About this task

To modify the way in which nodes and connections are displayed in the editor:

Procedure

1. Switch to the Broker Application Development perspective.
2. Open the message flow that you want to work with.
3. Click **Selection** above the node palette.
4. Right-click in the editor window and select **Manhattan Layout** if you want the connections between the nodes to be displayed in Manhattan style; that is with horizontal and vertical lines joined at right angles.
5. If you want to change the layout of the complete message flow:
   a. Right-click in the editor view and click **Layout**. The default for the alignment is left to right, such that your message flow starts (with an input node) on the left and control passes to the right.
   b. From the four further options displayed, **Left to Right**, **Right to Left**, **Top to Bottom**, and **Bottom to Top**, click the option that you want for this message flow. The message flow display is updated to reflect your choice. As a result of the change in alignment, all the nodes in the message flow are also realigned.

      For example, if you have changed from a left to right display (the default) to a right to left display, each node in the flow has now also changed to right to left (that is, the in terminal now appears on the right edge, the out terminals appear on the left edge).
6. You might want to arrange an individual node in a different direction from that in which the remaining nodes are arranged in the message flow:
   a. Right-click the node that you want to change and click **Rotate**. Four further options are displayed: **Left to Right**, **Right to Left**, **Top to Bottom**, and **Bottom to Top**. The option that represents the current arrangement of the node is not available for selection.
   b. Click the option that you want for this node.

Results

If you change the alignment of the message flow, or the arrangement of an individual node, or both, these settings are saved when you save the message flow. They are applied when another user accesses this same message flow, either through a shared repository or through shared files or import and export. When you reopen the message flow, you see these changed characteristics. The alignment and arrangement that you have selected for this message flow have no effect on the alignment and arrangement of any other message flow.

You can also access the editor toolbar to select other options related to the display and arrangement of nodes, for example, snap to grid. These options are defined in "Message Flow editor" on page 6810.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Adding a message flow node" on page 1494
When you have created a message flow, add nodes to define its function.

"Connecting message flow nodes" on page 1520
When you include more than one node in your message flow, you must connect the nodes to indicate how the flow of control passes from input to output. The nodes can be built-in nodes, user-defined nodes, or subflow nodes.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"Message Flow editor" on page 6810
The Message Flow editor is the default editor provided by the Broker Application Development perspective for defining a graphical representation of a message flow in the WebSphere Message Broker Toolkit, and for setting properties for individual message flow nodes.

Adding annotations to a message flow or node

You can add annotations to a message flow, a node, or multiple nodes. You can use these annotations to record reminders, issues arising during the development of a message flow, or informal documentation to facilitate team development.

Before you begin

Before you start:

The following instructions assume that you have created a message flow, as described in "Creating a message flow" on page 1431.

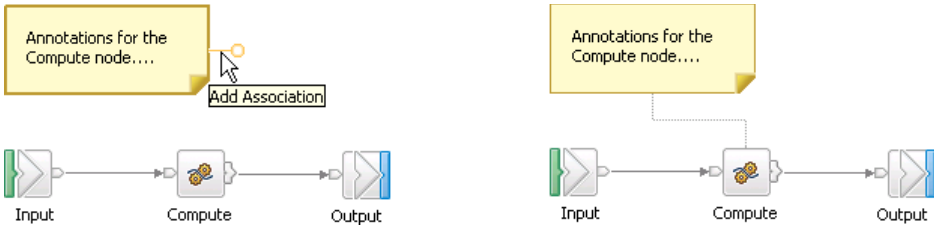
About this task

Procedure

- **Adding an annotation to a message flow**
  1. Open the message flow that you want to annotate in the Message Flow editor.
  2. In the message flow node palette, click **Note**, then click the canvas of the Message Flow editor. A blank note, in editing mode, is added to the canvas.
  3. Enter your comments to the note.
  4. To exit editing mode, click the canvas outside the note. The note resizes according to the amount of text that you enter.
  5. To save the contents of the note, save the message flow.
- **Associating an annotation with a node**
  1. Add a note to the canvas, as described in the preceding steps.
  2. To associate a note with a node in the message flow, move the mouse pointer over the edge of the note until a connector appears.
  3. Click the connector, then click the node with which you want to associate the note. The sticky note is associated with the selected node. This association is shown in the Message Flow editor by a dotted line.

     You can repeat these steps to associate an annotation with more than one node.

     The following diagram shows that a connector appears when you move the mouse cursor over the edge of a note, and that a dotted line represents the association between a node and a note.



What to do next

After you have added an annotation to a message flow or node, you can modify the annotation in the following ways:

- "Editing annotations on a message flow or node" on page 1533
- "Copying annotations on a message flow or node" on page 1534
- "Showing and hiding annotations on a message flow or node" on page 1535
- "Deleting annotations from a message flow or node" on page 1536

Related concepts:

"Message flow node palette" on page 1027
The palette in the Message Flow editor contains all the built-in nodes, which are organized into categories, or drawers. A drawer is a container for a list of icons, such as the **Favorites** drawer.

"Managing message flow resources" on page 1424
Manage your message flows and associated resources in the WebSphere Message Broker Toolkit.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related reference:

"Message Flow editor" on page 6810
The Message Flow editor is the default editor provided by the Broker Application Development perspective for defining a graphical representation of a message flow in the WebSphere Message Broker Toolkit, and for setting properties for individual message flow nodes.

Editing annotations on a message flow or node:

After you have added an annotation to a message flow or node, in the form of a sticky note, you can edit the contents of that note.

Before you begin

Before you start:

The following instructions assume that you have added an annotation to a message flow or node, as described in "Adding annotations to a message flow or node" on page 1531.

About this task

You can add annotations to message flows or nodes to create reminders, or record details about message flow development. For example, you might want to add a list of modifications that need to be made to a message flow. As you complete the modifications, you might want to edit the annotation to reflect progress. The following steps describe how to edit an annotation.

Procedure

- **Editing the content of a note**
  1. In the Message Flow editor, open the message flow that contains the note that you want to edit.
  2. To put the note into editing mode, click the text of the note. The note changes color, acquires a blue border, and a cursor appears.
  3. Edit the content of the note. You can use standard cut, copy, and paste operations.
  4. To exit editing mode, click the canvas outside the note. The note resizes according to the amount of text that you enter.
  5. To save the contents of the note, save the message flow.
- **Moving a sticky note**
  - To move a sticky note, click inside the note (not on the text) so that black dots appear at each corner. You can now drag the note to a new position on the canvas. If the note is associated with one or more nodes, the association line also moves.
  - You can also move a sticky note by using cut and paste operations. Right-click the note, click **Cut**, right-click the canvas where you want the note to appear, and click **Paste**.
- **Changing the association of a sticky note**
  - To associate a sticky note with a different node, move the mouse pointer over the old node, click the end of the association line, hold down the mouse button, then drag the association line to the new node. When you release the mouse button, the sticky note is associated with the new node, as indicated by a dotted line.

– To associate a node with a different sticky note, move the mouse pointer over the old sticky note, click the end of the association line, hold down the mouse button, then drag the association line to the new sticky note. When you release the mouse button, the node is associated with the new sticky note, as indicated by a dotted line.

Related tasks:

"Adding annotations to a message flow or node" on page 1531
You can add annotations to a message flow, a node, or multiple nodes. You can use these annotations to record reminders, issues arising during the development of a message flow, or informal documentation to facilitate team development.

"Copying annotations on a message flow or node"
You can copy annotations that you have added to a message flow or node to the same instance of the Message Flow editor, or to a different instance of the editor.

"Showing and hiding annotations on a message flow or node" on page 1535
After you have added an annotation to a message flow or node, you can hide it without having to delete it.

"Deleting annotations from a message flow or node" on page 1536
Typically an annotation on a message flow or node contains temporary information, therefore you can delete it when you no longer need it.

Related reference:

"Message Flow editor" on page 6810
The Message Flow editor is the default editor provided by the Broker Application Development perspective for defining a graphical representation of a message flow in the WebSphere Message Broker Toolkit, and for setting properties for individual message flow nodes.

Copying annotations on a message flow or node:

You can copy annotations that you have added to a message flow or node to the same instance of the Message Flow editor, or to a different instance of the editor.

Before you begin

Before you start:

The following instructions assume that you have added an annotation to a message flow or node, as described in "Adding annotations to a message flow or node" on page 1531.

Procedure

- In the Message Flow editor, open the message flow that contains the annotation that you want to copy. A copy of the note appears in the new location. If the original note is associated with one or more message flow nodes, those associations are not copied. You can also cut and paste a note in the same way.
- Right-click the note that you want to copy, then click **Copy**.
- Optional: If you want to copy annotations to a different Message Flow editor, switch to the Message Flow editor to which you want to copy the note.
- Right-click the Message Flow editor canvas where you want the copy of the note to appear, then click **Paste**.

Related tasks:

"Adding annotations to a message flow or node" on page 1531
You can add annotations to a message flow, a node, or multiple nodes. You can use these annotations to record reminders, issues arising during the development of a

message flow, or informal documentation to facilitate team development.

"Editing annotations on a message flow or node" on page 1533
After you have added an annotation to a message flow or node, in the form of a sticky note, you can edit the contents of that note.

"Showing and hiding annotations on a message flow or node"
After you have added an annotation to a message flow or node, you can hide it without having to delete it.

"Deleting annotations from a message flow or node" on page 1536
Typically an annotation on a message flow or node contains temporary information, therefore you can delete it when you no longer need it.

Related reference:

"Message Flow editor" on page 6810
The Message Flow editor is the default editor provided by the Broker Application Development perspective for defining a graphical representation of a message flow in the WebSphere Message Broker Toolkit, and for setting properties for individual message flow nodes.

Showing and hiding annotations on a message flow or node:

After you have added an annotation to a message flow or node, you can hide it without having to delete it.

Before you begin

Before you start:

The following instructions assume that you have added an annotation to a message flow or node, as described in "Adding annotations to a message flow or node" on page 1531.

About this task

Procedure

1. Open a message flow that contains notes.
2. Right-click a note, then click **Hide Notes**. All notes in the message flow are hidden.
3. To show all sticky notes, right-click the Message Flow editor canvas, then click **Show Notes**.

Related tasks:

"Adding annotations to a message flow or node" on page 1531
You can add annotations to a message flow, a node, or multiple nodes. You can use these annotations to record reminders, issues arising during the development of a message flow, or informal documentation to facilitate team development.

"Editing annotations on a message flow or node" on page 1533
After you have added an annotation to a message flow or node, in the form of a sticky note, you can edit the contents of that note.

"Copying annotations on a message flow or node" on page 1534
You can copy annotations that you have added to a message flow or node to the same instance of the Message Flow editor, or to a different instance of the editor.

"Deleting annotations from a message flow or node" on page 1536
Typically an annotation on a message flow or node contains temporary information, therefore you can delete it when you no longer need it.

Related reference:

"Message Flow editor" on page 6810
The Message Flow editor is the default editor provided by the Broker Application Development perspective for defining a graphical representation of a message flow in the WebSphere Message Broker Toolkit, and for setting properties for individual message flow nodes.

Deleting annotations from a message flow or node:

Typically an annotation on a message flow or node contains temporary information, therefore you can delete it when you no longer need it.

Before you begin

Before you start:

The following instructions assume that you have added an annotation to a message flow or node, as described in "Adding annotations to a message flow or node" on page 1531.

Procedure

- **Deleting a single annotation from a message flow**
  1. In the Message Flow editor, open the message flow that contains the note that you want to delete.
  2. Right-click the note that you want to delete, then click **Delete**. The note is deleted. If the note was associated with any nodes, those associations are also deleted.
- **Deleting multiple annotations from a message flow**
  1. In the Message Flow editor, open the message flow that contains the notes that you want to delete.
  2. Hold down Ctrl, highlight all the notes that you want to delete by clicking them (not on the text), then press **Delete**. The selected notes are deleted. If any of the notes are associated with nodes, those associations are also deleted.
- **Deleting an association between a note and a message flow node**
  1. In the Message Flow editor, open the message flow that contains the note association that you want to delete.
  2. Right-click the dotted association line between the note and the node, then click **Delete**. The association is deleted, but the note and node remain.
- **Deleting multiple associations between a note and message flow nodes**
  1. In the Message Flow editor, open the message flow that contains the note associations that you want to delete.
  2. Hold down Ctrl, select all the dotted association lines that you want to delete by clicking them, then press **Delete**. The selected associations are deleted, but the notes and nodes remain.

Related tasks:

"Adding annotations to a message flow or node" on page 1531
You can add annotations to a message flow, a node, or multiple nodes. You can use these annotations to record reminders, issues arising during the development of a message flow, or informal documentation to facilitate team development.

"Editing annotations on a message flow or node" on page 1533
After you have added an annotation to a message flow or node, in the form of a sticky note, you can edit the contents of that note.

"Copying annotations on a message flow or node" on page 1534
You can copy annotations that you have added to a message flow or node to the same instance of the Message Flow editor, or to a different instance of the editor.

"Showing and hiding annotations on a message flow or node" on page 1535
After you have added an annotation to a message flow or node, you can hide it without having to delete it.

Related reference:

"Message Flow editor" on page 6810
The Message Flow editor is the default editor provided by the Broker Application Development perspective for defining a graphical representation of a message flow in the WebSphere Message Broker Toolkit, and for setting properties for individual message flow nodes.

Connecting client applications

Connect your client applications to the broker by using one or more of the supported protocols from other resources and software servers in your network.

About this task

- "Processing WebSphere MQ messages"
- "Processing HTTP messages" on page 1579
- "Processing Web service messages" on page 1601
- "Processing TCP/IP messages" on page 1733
- "Processing email messages" on page 1786
- "Working with files" on page 1807
- "Working with JMS" on page 1709
- "Connecting to Enterprise Information Systems" on page 1912
- "Working with WebSphere Process Server" on page 2095
- "Working with databases" on page 2109
- "Working with IMS" on page 2128
- "Working with CORBA" on page 2144
- "Working with CICS Transaction Server for z/OS" on page 2172

Related tasks:

"Processing messages" on page 1021
Process your business messages and data by interacting with a broker, which you can configure to provide services and to communicate with other applications and systems.

"Routing messages" on page 2209
Route messages through your message flow or the broker by using one or more of the techniques described in this section.

"Transforming and enriching messages" on page 2227
Transform and enrich messages by using one or more of the techniques described in this section.

Processing WebSphere MQ messages

WebSphere Message Broker provides a number of nodes for handling messages received from or sent to WebSphere MQ applications.

About this task

WebSphere Message Broker provides the following nodes for working with message received from or sent to WebSphere MQ applications:

MQInput node
Use an MQInput node if the messages arrive at the broker on a WebSphere MQ queue, and the node is to be at the start of a message flow.

MQGet node
Use an MQGet node to retrieve a message from a WebSphere MQ queue, if you want to get the message later in the message flow.

MQReply node
Use an MQReply node if the target application expects to receive messages on the WebSphere MQ reply-to queue that is specified in the input message MD.

MQOutput node
Use an MQOutput node if the target application expects to receive messages on a WebSphere MQ queue, or on the WebSphere MQ reply-to queue that is specified in the input message MD.

MQHeader node
Use the MQHeader nodes to manipulate WebSphere MQ transport headers and their properties without writing compute nodes. Use the MQHeader node to add, modify, or delete Message Descriptor (MD) and Dead Letter Header (DLH) headers.

JMSMQTransform node
Use the JMSMQTransform node to transform a message with a JMS message tree into a message that has a tree structure that is compatible with the format of messages that are produced by the WebSphere MQ JMS provider.

The JMSMQTransform node can be used to send messages to existing message flows and to interoperate with WebSphere MQ JMS and WebSphere MQ Publish/Subscribe.

MQJMSTransform node
Use the MQJMSTransform node to receive messages that have a WebSphere MQ JMS provider message tree format, and transform them into a format that is compatible with messages that are to be sent to JMS destinations.

You can use the MQJMSTransform node to send messages to existing message flows and to interoperate with WebSphere MQ JMS and WebSphere MQ Publish/Subscribe.

Use the following links to find more information about configuring your message flows and related resources for handling messages received from or sent to WebSphere MQ applications:
- "Application programming interfaces" on page 1539
- "WebSphere MQ Enterprise Transport" on page 1542
- "Using WebSphere MQ cluster queues for input and output" on page 1544
- "Using WebSphere MQ shared queues for input and output (z/OS)" on page 1546
- "Configuring flows to handle WebSphere MQ message groups" on page 1553
- "Enabling WebSphere MQ applications" on page 1557
- "Ensuring that messages are not lost" on page 1561
- "Using MQGet nodes" on page 1564

Related tasks:

"Processing messages" on page 1021
Process your business messages and data by interacting with a broker, which you can configure to provide services and to communicate with other applications and systems.

"Developing message flow applications from scratch" on page 1423
Design, create, and configure message flows by using the WebSphere Message Broker Toolkit.

"Connecting client applications" on page 1537
Connect your client applications to the broker by using one or more of the supported protocols from other resources and software servers in your network.

"Routing messages" on page 2209
Route messages through your message flow or the broker by using one or more of the techniques described in this section.

"Transforming and enriching messages" on page 2227
Transform and enrich messages by using one or more of the techniques described in this section.

"Testing a message flow that has WebSphere MQ nodes" on page 3149
You can configure settings in the Test Client for testing message flows that have WebSphere MQ nodes.

Related reference:

"MQJMSTransform node" on page 4610
Use the MQJMSTransform node to receive messages that have a WebSphere MQ JMS provider message tree format, and transform them into a format that is compatible with messages that are to be sent to JMS destinations.

"JMSMQTransform node" on page 4547
Use the JMSMQTransform node to transform a message with a JMS message tree into a message that has a message tree structure that is compatible with the format of messages that are produced by the WebSphere MQ JMS provider.

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

"MQGet node" on page 4578
Use the MQGet node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and the MQI and AMI application programming interfaces.

"MQReply node" on page 4621
Use the MQReply node to send a response to the originator of the input message.

"MQOutput node" on page 4612
Use the MQOutput node to send messages to clients that connect to the broker using the WebSphere MQ Enterprise Transport and that use the MQI and AMI application programming interfaces.

"MQHeader node" on page 4590
Use the MQHeader node to add, modify, or delete MQ Message Descriptor (MQMD) and MQ Dead Letter Header (MQDLH) headers.

Application programming interfaces

WebSphere Message Broker supports several programming interfaces that are in use by WebSphere MQ applications today; it does not provide any unique programming interfaces.

- Message Queue Interface (MQI)

The MQI provides a few calls that allow an application to interact with other applications in a network of WebSphere MQ queue managers. The calls support a large range of parameters that allow a rich choice of processing options for each and every message.

Client applications using the MQI can run on any supported WebSphere MQ operating system, and therefore any limitations that are enforced for language or function are defined by the relevant product for that operating system.

The MQI is described in the *Application Programming Reference* and *Application Programming Guide* sections in the WebSphere MQ Version 7 Information Center online . Details are also provided of the programming language and operating system support available for clients that use this interface.

- Application Messaging Interface (AMI)

  The AMI is designed to simplify the application programmer's task by centralizing the selection of optional parameters outside the application program. It also provides support for the more advanced functions available from the broker. The AMI is designed for general messaging applications with and without a broker.

  The principal functions of the AMI are administrator-defined packets of options known as policies and services. An application specifies a service to determine the underlying messaging support required, and associates a policy with sending or receiving a message to control attributes for message processing, such as priority.

  The policies and services mean that the application does not have to understand details of the MQRFH2 header and the MQI interface.

  Client applications using the AMI are restricted to the operating systems and programming languages supported by this interface. The AMI is defined in the *Publish/Subscribe User's Guide* section in the WebSphere MQ Version 7 Information Center online.

If you have existing end-user applications that are written to these interfaces, they can typically run unchanged in a broker environment. You must create the message flows to interact with these applications from the supported protocols, using the appropriate input and output nodes. WebSphere Message Broker provides built-in input and output nodes for its supported protocols and you can create your own user-defined nodes to support additional protocols.

You can also create new end-user applications to interact with the broker.

Message headers

WebSphere Message Broker provides parsers for many WebSphere MQ headers, and can therefore accept messages that contain these headers from the WebSphere MQ Enterprise Transport protocol.

Messages must include a WebSphere MQ Message Descriptor (MQMD) as the first header, which must precede user or application data in every message. The MQMD contains basic control information that must travel with the message, including:

- The message identifier
- The destination of the reply, if one is to be sent
- Reply and report options (for example, confirm on delivery report)
- The format of any following data in the message

When a message is processed by a WebSphere Message Broker broker, it typically (but not necessarily) has one or more additional headers. The header following the MQMD is always identified in the format field within the MQMD, and itself contains another format field to identify either the header that follows, or the format of the user data.

The additional headers can include:

MQRFH
: The Rules and Formatting header is used by WebSphere MQ Publish/Subscribe.

MQRFH2
: The MQRFH2 is an updated version of MQRFH and allows Unicode strings to be transported without translation, and it can carry numeric data types. The MQRFH2 header carries a description of the message contents, so that WebSphere Message Broker can select the correct message parser when content-based processing is carried out on the message. In addition, this header contains publish/subscribe command messages.

  Use the MQRFH2 header in all new applications written for the WebSphere Message Broker environment that use a supported protocol based on WebSphere MQ technology. The MQRFH2 header must be immediately before the body of the message (that is, the last header).

  If an MQRFH2 header is not included (which is normally the case of the application uses a supported protocol that is not based on WebSphere MQ technology), you must configure the message flow that processes its messages to specify the message characteristics (by setting the input node properties).

Related concepts:

"Publish/Subscribe" on page 2215
Publish/subscribe is a style of messaging application in which the providers of information (publishers) are decoupled from the consumers of that information (subscribers).

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Routing using publish/subscribe applications" on page 2215
You can route your messages to applications using the publish/subscribe method of messaging.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"Publish/subscribe" on page 6395
Use the reference information in this section to help you develop publish/subscribe applications.

"MQRFH2 header" on page 6397
The MQRFH2 header is used to pass messages to and from a message broker that belongs to WebSphere Message Broker.

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Related information:

WebSphere MQ Enterprise Transport

WebSphere MQ Enterprise Transport is a service that connects applications to messaging middleware.

The WebSphere MQ Enterprise Transport is the transport used by WebSphere MQ. The WebSphere MQ Enterprise Transport supports WebSphere MQ applications that connect to WebSphere Message Broker to benefit from message routing and transformation options.

The WebSphere MQ Enterprise Transport provides all the reliable messaging features available in WebSphere MQ. This transport provides persistent and non-persistent messaging and supports transactions. To use the WebSphere MQ Enterprise Transport, you must deploy a message flow that contains an MQInput node to your broker. If this message flow sends output messages to other WebSphere MQ applications, it must also include an MQOutput, MQReply, or Publication node.

The WebSphere MQ Enterprise Transport is a queued transport; applications communicate with the broker by writing data to and reading data from message queues. Use the WebSphere MQ Enterprise Transport when you require assured delivery of messages or need to use transactional support.

WebSphere MQ Enterprise Transport is used by WebSphere MQ clients or application programs that are written to the Application Messaging Interface (AMI) or Message Queue Interface (MQI). The client uses the services provided by the message flows deployed within one or more brokers in the broker network by interacting with the queues serviced by those message flows.

The queue specified in the MQInput node determines the queue on which the broker receives publications from publishing applications. Subscribers connect to the broker by sending a registration request to the broker's SYSTEM.BROKER.CONTROL.QUEUE. The subscriber specifies a queue on which they want to receive any publications on the registered topic in the registration request.

All WebSphere Message Broker applications, like WebSphere MQ applications, can use all the supported WebSphere MQ interfaces to put messages to the message flow queues. In fact, every WebSphere MQ application is a potential WebSphere Message Broker application.

These applications use one of two techniques to gain access to broker services:

- An application can use a WebSphere MQ client connection. You can use all the WebSphere MQ clients supported by Version 6.0 or later. You can therefore connect applications running in a wide variety of environments into your broker network. An application running on the same system as the queue manager to which it connects can also use a client connection.
- An application running on the same system as a broker can use a local connection to the queue manager that hosts that broker. If it uses this method, the client must execute on the same system. It can connect to a queue manager supporting a broker, or to any other queue manager in the WebSphere MQ network that has a defined path to the broker's queue manager. (This option is not possible on z/OS, where clients are not supported.)

Multiple applications can communicate using separate local queue managers by using WebSphere MQ intercommunication (*remote queue definitions*, or *clustering*). For more details, see "Concepts of intercommunication" in the *Intercommunication* section of the WebSphere MQ information center.

WebSphere Message Broker does not impose any particular conditions or restrictions on applications.

Receiving applications can get the messages put to the output queue or queues of a message flow when they have been processed by that message flow. The applications must be connected, either by a client/server connection, or across a local connection, to the queue manager that owns the queue or queues defined as the target for their messages. If the message flow provides a publish/subscribe service, the Publication node puts the messages to the queue specified by the subscriber as its local receiver queue.

Applications that connect using WebSphere MQ Enterprise Transport use a mixture of point-to point and publish/subscribe models.

The following built-in nodes are provided to support this protocol:
- MQInput
- MQOutput
- MQReply
- MQGet
- Publication
- SCA nodes, when communicating with WebSphere Process Server applications.

Related concepts:

"Nodes for connectivity" on page 1028
WebSphere Message Broker supports direct connections from applications, and can send direct requests to other application endpoints. WebSphere Message Broker can also connect to various subsystems including WebSphere MQ, files, and databases, to read and write existing application data.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Connecting client applications" on page 1537
Connect your client applications to the broker by using one or more of the supported protocols from other resources and software servers in your network.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

"MQOutput node" on page 4612
Use the MQOutput node to send messages to clients that connect to the broker using the WebSphere MQ Enterprise Transport and that use the MQI and AMI application programming interfaces.

"MQReply node" on page 4621
Use the MQReply node to send a response to the originator of the input message.

"Publication node" on page 4643
Use the Publication node to filter output messages from a message flow and transmit them to subscribers who have registered an interest in a particular set of topics.

"SCAInput node" on page 4707
Use the SCAInput node with the SCAReply node to process messages from WebSphere Process Server.

"SCAReply node" on page 4726
Use the SCAReply node to send a message from the broker to the originating client in response to a message received by a SCAInput node.

"SCAAsyncRequest node" on page 4690
Use the SCAAsyncRequest node with the SCAAsyncResponse node to construct a pair of message flows that invoke a WebSphere Process Server service component asynchronously.

"SCARequest node" on page 4719
Use the SCARequest node to send a request to WebSphere Process Server. The node is a synchronous request and response node, and blocks after sending the request until the response is received. The node can also send one-way requests.

"SCAAsyncResponse node" on page 4698
Use the SCAAsyncResponse node with the SCAAsyncRequest node to construct a pair of message flows that start a component asynchronously.

Using WebSphere MQ cluster queues for input and output

Design your broker network to use WebSphere MQ queues, if appropriate for your business needs.

About this task

The use of queue manager clusters has the following significant benefits:

1. Reduced system administration

   Clusters need fewer definitions to establish a network; you can set up and change your network more quickly and easily.

2. Increased availability and workload balancing

   You can benefit by defining instances of the same queue to more than one queue manager, therefore distributing the workload through the cluster.

If you use clusters with WebSphere Message Broker, consider the following queues:

For SYSTEM.BROKER queues:
The SYSTEM.BROKER queues are defined for you when you create WebSphere Message Broker components, and are not defined as cluster queues. Do not change this attribute.

For message flow input queues:
If you define an input queue as a cluster queue, consider the implications for the order of messages or the segments of a segmented message. The implications are the same as for any WebSphere MQ cluster queue. In particular, the application must ensure that, if it is sending segmented messages, all segments are processed by the same target queue, and therefore by the same instance of the message flow at the same broker.

For message flow output queues:

- WebSphere Message Broker always specifies MQOO_BIND_AS_Q_DEF when it opens a queue for output. If you expect segmented messages to be put to an output queue, or want a series of messages to be handled by the same process, you must specify DEFBIND(OPEN) when you define that queue. This option ensures that all segments of a single message, or all messages in a sequence, are put to the same target queue and are processed by the same instance of the receiving application.
- If you create your own output nodes, specify MQOO_BIND_AS_Q_DEF when you open the output queue, and DEFBIND(OPEN) when you define the queue, if you need to ensure message order, or to ensure a single target for segmented messages.

For publish/subscribe applications:

- If the target queue for a publication is a cluster queue, you must deploy the publish/subscribe message flow to all the brokers on queue managers in the cluster. However, the cluster does not provide any of the failover function to the broker network and function. If a broker to which a message is published, or a subscriber registers, is unavailable, the distribution of the publication or registration is not taken over by another broker.
- When a client registers a subscription with a broker that is running on a queue manager that is a member of a cluster, the broker forwards a proxy registration to its neighbors in the broker domain; the registration details are not advertised to other members of the cluster.
- A client might choose to become a clustered subscriber, so that its subscriber queue is one of a set of clustered queues that receive a particular publication. In this case, when registering a subscription, use the name of an "imaginary" queue manager that is associated with the cluster; this queue manager is not the one to which the publication is sent, but an alias for the broker to use. As an administrative activity, a blank queue manager alias definition is made for this queue manager on the broker that satisfies this subscription for all clustered subscribers. When the broker publishes to a subscriber queue that names this queue manager, resolution of the queue manager name results in the publication being sent to any queue manager that hosts the subscriber cluster queue, and only one clustered subscriber receives the publication.

  For example, if the clustered subscriber queue was SUBS_QUEUE and the "imaginary" subscriber queue manager was CLUSTER_QM, the broker definition is:

  ```
  DEFINE QREMOTE(CLUSTER_QM) RQMNAME(' ') RNAME(' ')
  ```

  This configuration sends broker publications for SUBS_QUEUE on CLUSTER_QM to one instance of the cluster queue named SUBS_QUEUE anywhere in the cluster.

To understand more about clusters, and the implications of using cluster queues, see the *Queue Manager Clusters* section of the WebSphere MQ Version 7 Information Center online.

Related concepts:

“Message flows overview” on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Using WebSphere MQ shared queues for input and output (z/OS)"
On z/OS systems, you can define WebSphere MQ shared queues as input and output queues for message flows. You might need to serialize access to those messages.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Related information:

WebSphere MQ Version 7 Information Center online

Using WebSphere MQ shared queues for input and output (z/OS)

On z/OS systems, you can define WebSphere MQ shared queues as input and output queues for message flows. You might need to serialize access to those messages.

About this task

Shared queues are available only on z/OS. Use the WebSphere MQ for z/OS product facilities to define these queues and specify that they are shared.

You cannot use shared queues for broker component queues such as the SYSTEM.BROKER.CONTROL.QUEUE.

If you use shared queues, you can provide failover support between different images that are running WebSphere Message Broker on a sysplex.

Some messaging transactions depend on the exact sequence of messages from a queue, and for that sequence to be maintained in the event of a failure of the queue manager. In these instances, you must serialize the access to those messages.

Serialization of messages is achieved by using specialized connection options, and a unique connection token. This connection token is used when the application that empties the messages from a queue issues a connect call to the WebSphere MQ queue manager that owns that queue.

For example, WebSphere Message Broker can use this feature when multiple brokers, with multiple execution groups, are each running message flows that empty from a shared input queue. If one broker queue manager fails, the message flow can be started automatically on another broker. The transactional integrity and original sequencing of the messages on the shared queue are maintained.

The following examples demonstrate how these features can be applied:

• "Serialization of input between separate brokers on z/OS" on page 1547

- "Serialization of input between separate execution groups running on the same broker on z/OS" on page 1549
- "Serialization of input in an execution group on z/OS" on page 1551

To configure shared input queues and define serialization tokens for message flows, follow the instructions in "Serialization token - user tasks on z/OS" on page 1552.

For more information, see the *z/OS Concepts and Planning* section of the WebSphere MQ Version 7 Information Center online.

Related tasks:

"Customizing the z/OS environment" on page 591
If you are planning to use a z/OS environment, consider whether to create your brokers on z/OS. You must also complete a number of tasks to configure your environment.

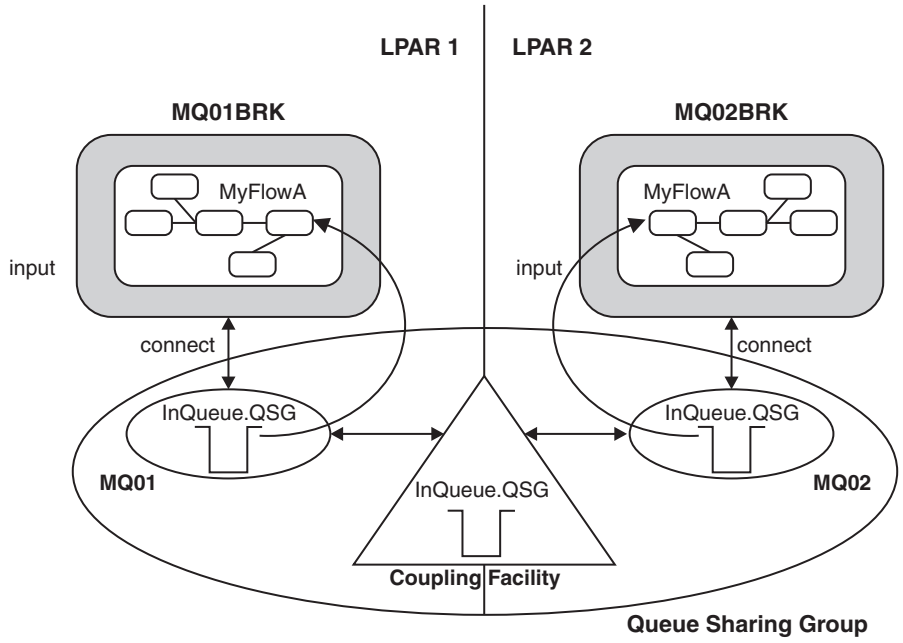
Serialization of input between separate brokers on z/OS: This example demonstrates that only one input node at a time takes messages from a shared queue when the same serialization token is used by message flows running on separate brokers.

Two brokers (`MQ01BRK` and `MQ02BRK`) are configured in this example. The respective queue managers are called `MQ01` and `MQ02`. The queue managers participate in the same queue sharing group. Each queue manager has a shared queue `INQueue.QSG` that has been defined with a disposition of `QSG`, and a local queue called `INQueue`. The queue managers can be running in the same Logical Partition (LPAR) or separate LPARs. The Coupling Facility shown in the following diagrams is a zSeries component that allows z/OS WebSphere MQ queue managers in the same system image, or different system images, to share queues.

An identical message flow `MyFlowA` is deployed to an execution group called `MYGroupA` on each broker. Note that the message flows do not have to be identical; the significant point is that an identical serialization token is used in both flows.

The simple message flow in this example consists of an `MQInput` node connected to an `MQOutput` node. The `MQInput` node in both message flows gets messages from the shared queue `INQueue.QSG`; the node attribute `Serialization Token` is configured as `MyToken123ABC` in both `MQInput` nodes.

The message flow property `additional Instances` takes the default value of zero in both message flows, which ensures that input is serialized within the flow.



A typical sequence of events for this example follows:

1. The first broker MQ01BRK starts and runs message flow MyFlowA in execution group MyGroupA. The input node MyInputNode connects to queue manager MQ01 using a serialization token MyToken123ABC. The input node successfully opens shared queue INQUeue.QSG and gets input messages.
2. The second broker MQ02BRK starts and begins to run its copy of message flow MyFlowA in execution group MyGroupA. The Input node MyInputNode attempts to connect to queue manager MQ02 , also using a serialization token MyToken123ABC.

   The following SDSF console message is logged:

   ```
   BIP2656I MQ02BRK MyGroupA 17 UNABLE TO OPEN QUEUE
   'INQueue.QSG' ON WEBSPHERE BUSINESS INTEGRATION QUEUE
   MANAGER 'MQ02': COMPLETION CODE 2; REASON CODE 2271.
   :ImbCommonInputNode(759) BECAUSE SERIALIZATION TOKEN
   MyToken123ABC is already in use. NO USER ACTION REQUIRED.
   ```

   Note that this message is output every 30 minutes.

   Message flow MyFlowA in execution group MyGroupA running on broker MQ02BRK is unable to process input because the serialization token it has passed is already in use within the queue sharing group. This is indicated by the reason code 2271 (MQRC_CONN_TAG_IN_USE) in message bip2623.
3. Broker MQ01BRK stops. Message flow MyFlowA in execution group MyGroupA in broker MQ02BRK2 is now able to get messages from the shared queue INQueue.QSG.

   A sequence of SDSF console messages is logged, of which the following two are relevant:

   ```
    BIP2091I MQ02BRK MyGroupA 17 THE BROKER HAS
   RECONNECTED TO WEBSPHERE BUSINESS INTEGRATION
   SUCCESSFULLY : ImbCommonInputNode(785)

    BIP9142I MQ01BRK 0 THE COMPONENT HAS STOPPED. :
   ImbControlService(594)
   ```

The preceding sequence of events also occurs should broker MQ01BRK fail, rather than stop through a request from the operator, or if a new broker configuration is deployed to MQ01BRK that deletes or modifies message flow MyFlowA.

This arrangement can also be used where the requirement is to migrate message processing between brokers running in different z/OS system images that are attached to the same Coupling Facility.

Related concepts:

"z/OS customization overview" on page 592
After you have used SMP/E to install WebSphere Message Broker for z/OS, the installed executable code is located inside the file system. JCL samples are located in the PDS <hlq>.SBIPSAMP, the JCL procedures are located in the PDS <hlq>.SBIPPROC, and load module for synchronizing statistics with SMF are located in the PDS <hlq>.SBIPAUTH.

Overview of message serialization on z/OS
Some messaging transactions depend on the exact sequence of messages from a queue, and for that sequence to be maintained in the event of a failure of the queue manager. In these instances you must serialize the access to those messages.

"Serialization of input between separate execution groups running on the same broker on z/OS"
This example demonstrates that only one MQInput node at a time is allowed to take messages from a shared queue when the same serialization token is used by message flows running in separate execution groups on the same broker.

"Serialization of input in an execution group on z/OS" on page 1551
To allow concurrent processing within a message flow, while still serializing messages between message flows in separate execution groups, the scope of the serialization token is restricted within a single execution group.

Serialization of input between separate execution groups running on the same broker on z/OS:

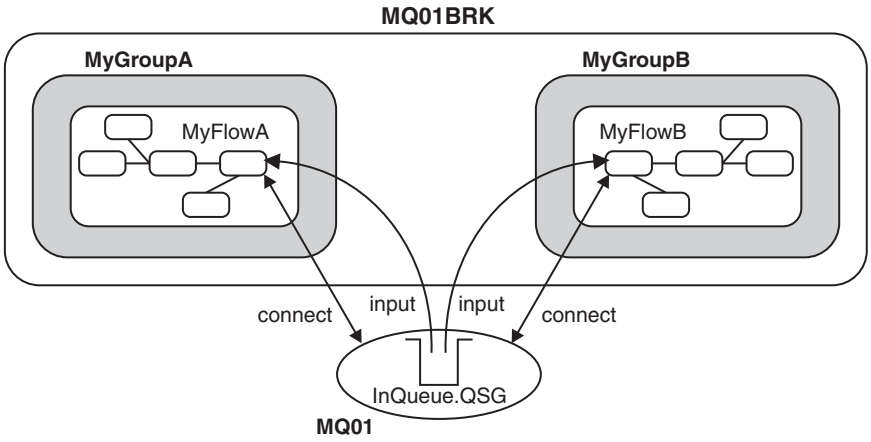
This example demonstrates that only one MQInput node at a time is allowed to take messages from a shared queue when the same serialization token is used by message flows running in separate execution groups on the same broker.

An identical message flow MyFlowA is deployed to two execution groups called MYGroupA and MYGroupB on broker MQ01BRK.

In this case it is not a requirement that the queue manager participates in a queue sharing group. The input queue INQueue is defined as local with disposition QMGR.

As in "Serialization of input between separate brokers on z/OS" on page 1547:

- Note that the message flows do not have to be identical; the significant point is that an identical serialization token is used in both flows.
- The simple message flow in this example consists of an MQInput node connected to an MQOutput node. The MQInput node in both message flows gets messages from the shared queue INQueue.QSG; the node attribute Serialization Token is configured as MyToken123ABC in both MQInput nodes.
- The message flow property additional Instances takes the default value of zero in both message flows, which ensures that input is serialized within the flow.



A typical sequence of events for this example follows:

1. Broker MQ01BRK starts and the first message flow to begin is MyFlowA in execution group MyGroupA. The MQInput node MyInputNode connects to queue manager MQ01 using the serialization token MyToken123ABC. The MQInput node successfully opens shared queue INQUeue and gets input messages.
2. The second execution group MyGroupB starts and message flow MyFlowA in execution group MyGroupB begins. The MQInput node MyInputNode now attempts to connect to queue manager MQ01 using serialization token MyToken123ABC. The following SDSF console message is logged:

   ```
   BIP2656I MQ01BRK MyGroupB 11 UNABLE TO OPEN QUEUE
   'INQueue' ON WEBSPHERE BUSINESS INTEGRATION QUEUE
   MANAGER 'MQ01':  BECAUSE SERIALIZATION TOKEN
   MyToken123ABC is already in use. NO USER ACTION REQUIRED
   ```

   Message flow MyFlowA in execution group MyGroupB is unable to process input because the serialization token it has passed is already in use within the queue manager (by the MQInput node in message flow MyFlowA in execution group MyGroupA). This is indicated by the reason code 2271 (MQRC_CONN_TAG_IN_USE) in message bip2623.
3. The first execution group is deleted or canceled.

   If the first execution group is canceled by the operator, ends with an abend, or is deleted during a redeployment of the broker configuration, the input node in the second execution group is now able to get input messages from queue INQueue.

   A sequence of SDSF console messages is logged, of which the following one is relevant:

   ```
    BIP2091I MQ01BRK MyGroupB 11 THE BROKER HAS
   RECONNECTED TO WEBSPHERE BUSINESS INTEGRATION
   SUCCESSFULLY : ImbCommonInputNode(785)
   ```

Message flow MyFlowA in execution group MyGroupB is now able to recover processing of messages from the shared queue INQueue.QSG.

Note that, although serialization of input can be achieved in a similar manner by configuring the input queue for exclusive input, this does not ensure message integrity during a recovery situation. This can be achieved only through the use of the serialization token as described in this example.

Related concepts:

"z/OS customization overview" on page 592
After you have used SMP/E to install WebSphere Message Broker for z/OS, the installed executable code is located inside the file system. JCL samples are located in the PDS `<hlq>.SBIPSAMP`, the JCL procedures are located in the PDS `<hlq>.SBIPPROC`, and load module for synchronizing statistics with SMF are located in the PDS `<hlq>.SBIPAUTH`.

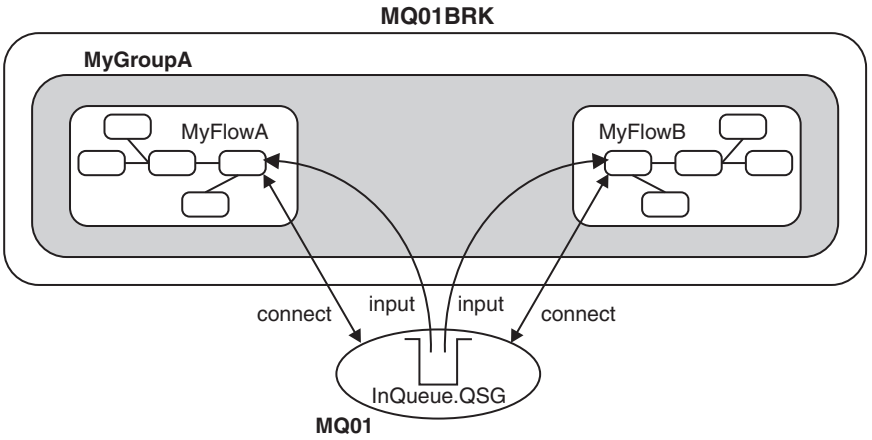
Overview of message serialization on z/OS
Some messaging transactions depend on the exact sequence of messages from a queue, and for that sequence to be maintained in the event of a failure of the queue manager. In these instances you must serialize the access to those messages.

"Serialization of input between separate brokers on z/OS" on page 1547

"Serialization of input in an execution group on z/OS"
To allow concurrent processing within a message flow, while still serializing messages between message flows in separate execution groups, the scope of the serialization token is restricted within a single execution group.

Serialization of input in an execution group on z/OS:

To allow concurrent processing within a message flow, while still serializing messages between message flows in separate execution groups, the scope of the serialization token is restricted within a single execution group.



This example demonstrates that the serialization token is restricted within a single execution group running on a broker::

- Two `MQInput` nodes in separate message flows (in this case `MyFlowA` and `MyFlowB`) are running within the same execution group `MyGroupA`. Both `MQInput` nodes concurrently get messages from the shared input queue even though they are using the same serialization token.
- If serialization is required within a single message flow, the message flow attribute `additional instances` must be set to zero which is the default setting. However, if greater throughput is required and serialization of input within the flow is not important, you can set `additional instances` to a value greater than zero.
- The use of the `serialization token` attribute on the `MQInput` node does not serialize input between message flows operating within the same execution group. However, setting the attribute has no adverse affect on the processing within that execution group

- In this way it is possible to maximize throughput in a message flow on one broker while still serializing input between brokers. This is useful where the requirement is to have one or more brokers acting as an immediate standby, should the currently active broker need to be stopped for servicing, or fail unexpectedly.

Related concepts:

"z/OS customization overview" on page 592
After you have used SMP/E to install WebSphere Message Broker for z/OS, the installed executable code is located inside the file system. JCL samples are located in the PDS `<hlq>.SBIPSAMP`, the JCL procedures are located in the PDS `<hlq>.SBIPPROC`, and load module for synchronizing statistics with SMF are located in the PDS `<hlq>.SBIPAUTH`.

Overview of message serialization on z/OS
Some messaging transactions depend on the exact sequence of messages from a queue, and for that sequence to be maintained in the event of a failure of the queue manager. In these instances you must serialize the access to those messages.

"Serialization of input between separate brokers on z/OS" on page 1547

"Serialization of input between separate execution groups running on the same broker on z/OS" on page 1549
This example demonstrates that only one `MQInput` node at a time is allowed to take messages from a shared queue when the same serialization token is used by message flows running in separate execution groups on the same broker.

Serialization token - user tasks on z/OS:

Configure shared input queues and define serialization tokens for message flows.

About this task

Configure a shared input queue for message flows

The broker makes use of WebSphere MQ queue-sharing groups on z/OS.

Queue managers that can access the same set of shared queues form a group called a queue-sharing group (QSG) and they communicate with each other by means of a coupling facility (CF) that stores the shared queues. A shared queue is a type of local queue whose messages can be accessed by one or more queue managers that are in a QSG.

To further enhance the availability of messages in a QSG, WebSphere MQ detects if another queue manager in the group disconnects from the CF in an unusual way, and completes pending units of work for that queue manager where possible; this is known as peer recovery.

To understand more fully the concepts of shared-queues and queue-sharing groups, see the *Concepts and Planning Guide* section of the WebSphere MQ Version 7 Information Center online, and perform the following steps:

- Add a QSG to the DB2 tables
- Add a queue manager to a QSG
- Create the shared queue as a member of the QSG

Define a serialization token

Define the same value for the serialization token attribute for each MQInput node that is required to access the shared queue.

For the situations described in the preceding text to work you must:

- Ensure that the Coupling Facility Structure is at `CFLEVEL(3)` or above, and that you set `RECOVER=YES`.

  If you do not do this, when an MQInput node attempts to get a message from the shared queue, the action fails with the WebSphere MQ return code BIP2048 (`MQRC_PERSISTENT_NOT_ALLOWED`)
- Set the Backout Threshold for the shared queue to at least `2`.

  This value prevents input messages that are in progress being sent to the Dead Letter Queue because, during recovery, a message is restored to the input queue before another broker is able to get it and resume processing.

Related concepts:

"z/OS customization overview" on page 592
After you have used SMP/E to install WebSphere Message Broker for z/OS, the installed executable code is located inside the file system. JCL samples are located in the PDS `<hlq>.SBIPSAMP`, the JCL procedures are located in the PDS `<hlq>.SBIPPROC`, and load module for synchronizing statistics with SMF are located in the PDS `<hlq>.SBIPAUTH`.

Overview of message serialization on z/OS
Some messaging transactions depend on the exact sequence of messages from a queue, and for that sequence to be maintained in the event of a failure of the queue manager. In these instances you must serialize the access to those messages.

Related information:

WebSphere MQ Version 7 Information Center online

Configuring flows to handle WebSphere MQ message groups

WebSphere MQ allows multiple messages to be treated as a group, or as segments of one larger message. WebSphere Message Broker provides support for WebSphere MQ message grouping and partial support for message segmenting.

You can use the MQInput and MQOutput nodes to receive and send messages that are part of a WebSphere MQ message group. You can use the MQOutput node to send messages that are segments of a larger message.

For guidance about configuring the MQInput and MQOutput nodes to receive and send messages that are part of a WebSphere MQ message group, see:

- "Receiving messages in a WebSphere MQ message group" on page 1554
- "Sending messages in a WebSphere MQ message group" on page 1555
- "Sending message segments in a WebSphere MQ message" on page 1557

For more information about WebSphere MQ message groups, see the *Application Programming Guide* section of the WebSphere MQ Version 7 Information Center online.

Related reference:

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

"MQOutput node" on page 4612
Use the MQOutput node to send messages to clients that connect to the broker using the WebSphere MQ Enterprise Transport and that use the MQI and AMI application programming interfaces.

Related information:

WebSphere MQ Version 7 Information Center online

Receiving messages in a WebSphere MQ message group:

You can configure the MQInput node to receive messages that are in a WebSphere MQ message group.

About this task

The following properties on the MQInput node control the processing of messages in a WebSphere MQ message group:

- `Logical order`
- `Order mode`
- `All messages available`
- `Commit by message group`

To ensure that your message flow receives group messages in the order that has been assigned by the sending application, select `Logical order`. If you do not select this option, messages that are sent as part of a group are not received in a predetermined order. This property maps to the MQGMO_LOGICAL_ORDER option of the MQGMO of the MQI. More information about the options to which this property maps is available in the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online.

If you specify a value of `By Queue Order` on the `Order mode` property, the message flow processes the messages in the group in the order that is defined by the queue attributes; this order is guaranteed to be preserved when the messages are processed. This behavior is identical to the behavior that is exhibited if the `Additional instances` property is set to zero. The message flow processes the messages on a single thread of execution, and a message is processed to completion before the next message is retrieved from the queue. If you do not specify this value, it is possible that multiple threads within a single message flow are processing multiple messages, and the final message in a group, which prompts the commit or roll back action, is not guaranteed to be processed to completion after all other messages in the group.

To ensure that only a single instance of the message flow processes the group messages in the order that has been assigned by the sending application, select `Logical order` **and** specify a value of `By Queue Order` on the `Order mode` property.

If you select `All messages available`, message retrieval and processing is performed only when all messages in a single group are available. This means that messages in a group are not received until all the messages in the group are present on the input queue. It is good practice to select this check box when your message flow needs to process group messages. If you do not select this check box, the message flow receives the messages as they arrive on the input queue; if a message in the group fails to arrive on the input queue, the message flow waits for it and cannot process any further messages until this message arrives. This property maps to the MQGMO_ALL_MESSAGES_AVAILABLE option of the

MQGMO of the MQI. More information about the options to which this property maps is available in the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online.

If you select `Commit by message group`, message processing is committed only after the final message of a group has been received and processed. If you leave this check box cleared, a commit is performed after each message has been propagated completely through the message flow. This property is relevant only if you have selected `Logical order`. It is good practice to select this check box together with the `All messages available` check box because this ensures that the complete message group is retrieved and processed in the same unit of work without risk of the message flow waiting indefinitely for messages in the group to arrive on the input queue.

To ensure that the message flow that processes group messages does not wait for unavailable messages:

- Avoid having multiple message flows reading from the same input queue when group messages are being retrieved.
- Avoid deploying additional instances of a flow that retrieves group messages.
- Avoid using expired messages in message groups.
- When expired messages are to be used, ensure either that all messages have the same expiry time or that the first message in the group is set to expire before the rest of the group. If the first message in a group cannot be retrieved, the group can never be started in logical order.

If a message flow waits for a group message that does not arrive within the wait interval, a BIP2675 warning message is issued. From that point on, the message flow always attempts to retrieve the next group message and does not process any other input messages until the next group message is received.

Therefore, if the expected group message does not arrive, or has expired, the message flow must be stopped manually, and any incomplete message group cleared from the input queue.

A message flow cannot receive all the messages in a group in one operation.

If you specify a value of `Yes` or `No` on the `Transaction mode` property, all the segments in a message are received in the message flow as a single message. As a result, the message flow might receive very large messages which might lead to storage problems in the broker. If you specify a value of `Automatic` on the `Transaction mode` property, message segments are received as individual messages.

Related reference:

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

Related information:

WebSphere MQ Version 7 Information Center online

Sending messages in a WebSphere MQ message group:

The MQOutput node can send multiple messages that form a WebSphere MQ message group. Configure a Compute or JavaCompute node to set the MQMD fields to specify message group options.

About this task

The message flow must set the following MQMD fields:

- **GroupId**
- **MsgSeqNumber**
- **MsgFlags**

You can use the following example of ESQL code in a Compute node, which shows how to set these fields:

```
DECLARE MSGNUM INT 0;
  DECLARE MSGTOTAL INT 5;
  WHILE MSGNUM < MSGTOTAL DO
    SET MSGNUM = MSGNUM + 1;
    CALL CopyMessageHeaders();
    -- Manually set the groupId since we cant ask the queue manager to generate one.
    -- the UUIDASBLOB function could be used here to generate one, but this must be done
    -- outside the loop to keep the same groupId throughout!
    SET OutputRoot.MQMD.GroupId = X'000000000000000000000000000000000000000000000001';
    SET OutputRoot.MQMD.MsgSeqNumber = MSGNUM;
    SET OutputRoot.MQMD.MsgFlags = MQMF_MSG_IN_GROUP;
    IF (MSGNUM = MSGTOTAL) THEN
      SET OutputRoot.MQMD.MsgFlags = MQMF_LAST_MSG_IN_GROUP;
    END IF;
    SET OutputRoot.XML.TestCase = MSGNUM;
    PROPAGATE;
  END WHILE;
  RETURN FALSE;
```

You can use the following example of Java code in a JavaCompute node, which shows how to set these fields:

```
public class mq_group_java_JavaCompute extends MbJavaComputeNode {
      public void evaluate(MbMessageAssembly assembly) throws MbException {
        MbOutputTerminal out = getOutputTerminal("out");
        int MSGTOTAL = 5;
        for(int MSGNUM = 1 ; MSGNUM <= MSGTOTAL;MSGNUM++){
           MbMessage newMessage = new MbMessage();
           MbElement root = newMessage.getRootElement();
           MbElement mqmd = root.createElementAsFirstChild("MQMD");
           mqmd.createElementAsLastChild(MbElement.TYPE_NAME_VALUE,
           "GroupId", new byte[]{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1});
           mqmd.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "MsgSeqNumber", MSGNUM);
           if (MSGNUM == MSGTOTAL){
               mqmd.createElementAsLastChild(MbElement.TYPE_NAME_VALUE,
               "MsgFlags", 16); // MQMF_LAST_MSG_IN_GROUP 0x00000010
           }
           else{
               mqmd.createElementAsLastChild(MbElement.TYPE_NAME_VALUE,
               "MsgFlags", 8);  // MQMF_MSG_IN_GROUP      0x00000008
           }
           MbElement xmlnsc = root.createElementAsLastChild("XMLNSC");
           MbElement body = xmlnsc.createElementAsLastChild(MbElement.TYPE_NAME,"TestCase",null);
           body.setValue(MSGNUM);
           MbMessageAssembly newAssembly = new MbMessageAssembly(assembly,newMessage);
           out.propagate(newAssembly);
        }
    }
}
```

If the message flow is sending multiple messages from one input message, it can create a **GroupId** value, increment the **MsgSeqNumber** value, and set the **MsgFlags** field. The examples of ESQL and Java code show how you can set these values. However, if the message flow is sending multiple messages from more than

one input message, it must store the **GroupId** and **MsgSeqNumber** values between flow instances, which is achieved by using shared variables.

For more information about message grouping, see the *Application Programming Guide* section of the WebSphere MQ Version 7 Information Center online. For more information about the WebSphere MQ fields that are significant in message grouping, see the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online.

Related reference:

"MQOutput node" on page 4612
Use the MQOutput node to send messages to clients that connect to the broker using the WebSphere MQ Enterprise Transport and that use the MQI and AMI application programming interfaces.

Related information:

WebSphere MQ Version 7 Information Center online

Sending message segments in a WebSphere MQ message:

The MQOutput node can send multiple message segments that form a WebSphere MQ message. Configure a Compute node to set the MQMD fields to specify message segment options.

About this task

You can either select `Segmentation allowed` on the node, or set the required fields in the MQMD in the message flow:

- GroupId
- MsgFlags
- Offset

Use the example ESQL code in "Sending messages in a WebSphere MQ message group" on page 1555 and change the code to set these fields.

For more information about message grouping and segmentation, see the *Application Programming Guide* section of the WebSphere MQ Version 7 Information Center online. For more information about the WebSphere MQ fields that are significant in message grouping and segmentation, see the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online.

Related reference:

"MQOutput node" on page 4612
Use the MQOutput node to send messages to clients that connect to the broker using the WebSphere MQ Enterprise Transport and that use the MQI and AMI application programming interfaces.

Related information:

WebSphere MQ Version 7 Information Center online

Enabling WebSphere MQ applications

If you want to connect WebSphere MQ applications to the broker, you must define and secure the required resources.

About this task

- "Defining WebSphere MQ resources" on page 1558
- "Securing WebSphere MQ resources" on page 1559

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Chapter 11, "Packaging and deploying," on page 3209
Package resources that you create in the WebSphere Message Broker Toolkit, such as message flows, and deploy them to execution groups on brokers.

Defining WebSphere MQ resources:

An application client can run on a computer anywhere in the WebSphere MQ network. If your applications use WebSphere MQ facilities to connect to the broker, and to interact with it (by using the MQI and AMI), you must set up the WebSphere MQ resources that they require.

About this task

The way that you set up applications is identical to the setup for clients for an WebSphere MQ server. To support client connections to a broker:

Procedure

1. If the application runs on the same computer as the broker, it can establish a local connection with the broker queue manager by using MQCONN, and you do not have to define any WebSphere MQ resources to support it.
2. If the application runs on the same computer as another queue manager in the WebSphere MQ network, it can establish a local connection to that queue manager. In this scenario, you must define the appropriate resource to support communications between the queue manager to which the client has connected and the queue manager that hosts the broker that provides the required service.
3. If the application runs on a computer that does not support a queue manager, it must make a client connection to a queue manager on another computer:
   - The broker queue manager

     You must set up the appropriate client connection and server connection definitions to support this option.
   - Another queue manager in the network

     You must set up the appropriate client connection and server connection definitions to support this option, and ensure that definitions are in place to support communications between the queue manager to which the client has connected, and the queue manager that hosts the broker that provides the required service.

Results

An application can get messages only from queues that are owned by the queue manager to which it is connected (this restriction is true for all WebSphere MQ applications). Therefore, if an application expects to receive messages from a queue populated by a service within a particular broker and owned by that broker queue manager, it must connect to that broker queue manager (by using a local or WebSphere MQ client connection).

An application that puts messages, however, can be connected to any queue manager in the WebSphere MQ network, if the queue manager can resolve the target destination in some way. In all cases, the queue manager to which the client application is connected must know the location of the queue or queues to which the application puts messages (for example, by using remote queue definitions).

When you define a WebSphere MQ queue for a node in a message flow, you must not give it a name that starts with SYSTEM_BROKER. Names that include these characters are reserved for queues that are defined for internal use by WebSphere Message Broker components.

If your application is a subscriber that receives messages published by other applications, it can specify a temporary dynamic queue as its subscriber queue. If it does so, the broker automatically deregisters the subscription when the queue is deleted.

For more details about applications, putting and getting messages, and the use of WebSphere MQ clients, see the *Clients* and *Application Programming Guide* sections of the WebSphere MQ Version 7 Information Center online.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Chapter 11, "Packaging and deploying," on page 3209
Package resources that you create in the WebSphere Message Broker Toolkit, such as message flows, and deploy them to execution groups on brokers.

Related reference:

"WebSphere MQ Enterprise Transport" on page 1542
WebSphere MQ Enterprise Transport is a service that connects applications to messaging middleware.

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Related information:

WebSphere MQ Version 7 Information Center online

Securing WebSphere MQ resources:

Secure the WebSphere MQ resources that your broker configuration requires.

About this task

This information does not apply to z/OS.

Brokers depend on a number of WebSphere MQ resources to operate successfully. You must control access to these resources to ensure that the brokers can access the resources on which they depend, and that these same resources are protected from other users.

Some authorizations are granted on your behalf when commands are issued. Others depend on the configuration of your broker network.

- When you issue the command **mqsicreatebroker**, it grants put and get authority on your behalf to the group mqbrkrs for the following queues:
  - SYSTEM.BROKER.ADAPTER.FAILED
  - SYSTEM.BROKER.ADAPTER.INPROGRESS
  - SYSTEM.BROKER.ADAPTER.NEW
  - SYSTEM.BROKER.ADAPTER.PROCESSED
  - SYSTEM.BROKER.ADAPTER.UNKNOWN
  - SYSTEM.BROKER.ADMIN.QUEUE
  - SYSTEM.BROKER.ADMIN.REPLYTODM
  - SYSTEM.BROKER.AGGR.CONTROL
  - SYSTEM.BROKER.AGGR.REPLY
  - SYSTEM.BROKER.AGGR.REQUEST
  - SYSTEM.BROKER.AGGR.TIMEOUT
  - SYSTEM.BROKER.AGGR.UNKNOWN
  - SYSTEM.BROKER.AUTH
  - SYSTEM.BROKER.CONTROL.QUEUE
  - SYSTEM.BROKER.DEPLOY.QUEUE
  - SYSTEM.BROKER.DEPLOY.REPLY
  - SYSTEM.BROKER.EDA.COLLECTIONS
  - SYSTEM.BROKER.EDA.EVENTS
  - SYSTEM.BROKER.EXECUTIONGROUP.QUEUE
  - SYSTEM.BROKER.EXECUTIONGROUP.REPLY
  - SYSTEM.BROKER.INTER.BROKER.COMMUNICATIONS
  - SYSTEM.BROKER.MODEL.QUEUE
  - SYSTEM.BROKER.TIMEOUT.QUEUE
  - SYSTEM.BROKER.WS.ACK
  - SYSTEM.BROKER.WS.INPUT
  - SYSTEM.BROKER.WS.REPLY
- When you start the WebSphere Message Broker Toolkit, it connects to a broker by using a WebSphere MQ client/server connection. For details of WebSphere MQ channel security, see "Setting up WebSphere MQ client security" in the *Clients* section of the WebSphere MQ Version 7 Information Center online.
- When you start the WebSphere Message Broker Explorer, it connects to a broker by using a WebSphere MQ client/server connection. For details of WebSphere MQ channel security, see "Setting up WebSphere MQ client security" in the *Clients* section of the WebSphere MQ Version 7 Information Center online.
- When you create and deploy a message flow that includes nodes which reference WebSphere MQ queues, grant get, inq, and put authority to the user ID under which the broker is running.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Chapter 11, "Packaging and deploying," on page 3209
Package resources that you create in the WebSphere Message Broker Toolkit, such as message flows, and deploy them to execution groups on brokers.

Related reference:

"WebSphere MQ Enterprise Transport" on page 1542
WebSphere MQ Enterprise Transport is a service that connects applications to messaging middleware.

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

Related information:

WebSphere MQ Version 7 Information Center online

Ensuring that messages are not lost

Messages that flow through your broker domain represent business data that you want to safeguard. Configure the messages, your environment, or both, to ensure that you do not lose messages.

About this task

Messages that are generated both by your applications and by runtime components for inter-component communication are important to the operation of your brokers. Messages used internally between components always use the WebSphere MQ protocol. Application messages can use all supported transport protocols.

For application and internal messages traveling across WebSphere MQ, two techniques protect against message loss:

- Message persistence

  If a message is persistent, WebSphere MQ ensures that it is not lost when a failure occurs, by copying it to disk.
- Sync point control

  An application can request that a message is processed in a synchronized unit-of-work (UOW)

For more information about how to use these options, refer to the *System Administration Guide* section of the WebSphere MQ Version 7 Information Center online.

Internal messages:
About this task

WebSphere Message Broker components use WebSphere MQ messages to communicate events and data between broker processes and subsystems. The components ensure that the WebSphere MQ features are used to protect against message loss. You do not need to take any additional steps to configure WebSphere MQ or WebSphere Message Broker to protect against loss of internal messages.

Application messages:
About this task

If delivery of application messages is critical, you must design application programs and the message flows that they use to ensure that messages are not lost. The techniques used depend on the protocol used by the applications.

WebSphere MQ Enterprise Transport

If you are using the built-in input nodes that accept messages across the WebSphere MQ protocol, you can use the following guidelines and recommendations:

- Using persistent messages

  WebSphere MQ messaging products provide *message persistence*, which defines the longevity of the message in the system and guarantees message integrity. Nonpersistent messages are lost in the event of system or queue manager failure. Persistent messages are always recovered if a failure occurs.

  You can control message persistence in the following ways:

  – Program your applications that put messages to a queue using the MQI or AMI to indicate that the messages are persistent.
  – Define the input queue with message persistence as the default setting.
  – Configure the output node to handle persistent messages.
  – Program your subscriber applications to request message persistence.

  When an input node reads a message from an input queue, the default action is to use the persistence defined in the WebSphere MQ message header (MQMD), that has been set either by the application creating the message, or by the default persistence of the input queue. The message retains this persistence throughout the message flow, unless it is changed in a subsequent message processing node.

  You can override the persistence value of each message when the message flow terminates at an output node. This node has a property that allows you to specify the message persistence of each message when it is put to the output queue, either as the required value, or as a default value. If you specify the default, the message takes the persistence value defined for the queues to which the messages are written.

  If a message passes through a Publication node, the persistence of messages sent to subscribers is determined by the registration options of the subscribers. If a subscriber has requested persistent message delivery, and is authorized to do so by explicit or implicit (inherited) ACL, the message is delivered persistently regardless of its existing persistence property. Also, if the user has requested nonpersistent message delivery, the message is delivered nonpersistent regardless of its existing persistence property.

  If a message flow creates a message (for example, in a Compute node), the persistence in the MQMD of the new message is copied from the persistence in the MQMD of the incoming message.

- Processing messages under sync point control

  The default action of a message flow is to process incoming messages under sync point in a broker-controlled transaction. This means that a message that fails to be processed for any reason is backed out by the broker. Because it was received under sync point, the failing message is reinstated on the input queue and can be processed again. If the processing fails, the error handling procedures that are in place for this message flow (defined either by how you have configured the message flow, or by the broker) are executed.

  For full details of input node processing, see “Managing errors in the input node” on page 2828.

WebSphere MQ Web Services Transport
If you are using the HTTPInput, HTTPRequest, SOAPInput, SOAPRequest nodes, or a SOAPAsyncRequest and SOAPAsyncResponse node pair that accept messages from Web services applications, no facilities are available to protect against message loss. You can, however, provide recovery procedures by configuring the message flow to handle its own errors.

Other transports and protocols
If you have created your own user-defined input nodes that receive messages from another transport protocol, you must rely on the support provided by that transport protocol, or provide your own recovery procedures.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Nodes for connectivity" on page 1028
WebSphere Message Broker supports direct connections from applications, and can send direct requests to other application endpoints. WebSphere Message Broker can also connect to various subsystems including WebSphere MQ, files, and databases, to read and write existing application data.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Managing errors in the input node" on page 2828
When you design your message flow, consider which terminals on the input node to connect.

"Connecting client applications" on page 1537
Connect your client applications to the broker by using one or more of the supported protocols from other resources and software servers in your network.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"WebSphere MQ Enterprise Transport" on page 1542
WebSphere MQ Enterprise Transport is a service that connects applications to messaging middleware.

Related information:

WebSphere MQ Version 7 Information Center online

Using MQGet nodes

The MQGet node processes messages in a particular way, and you can use it in request-response message flows.

Procedure

- "How the MQGet node processes messages"
- "A request-response scenario that uses an MQGet node" on page 1569

Related tasks:

"Processing WebSphere MQ messages" on page 1537
WebSphere Message Broker provides a number of nodes for handling messages received from or sent to WebSphere MQ applications.

Related reference:

"MQGet node" on page 4578
Use the MQGet node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and the MQI and AMI application programming interfaces.

How the MQGet node processes messages:

The MQGet node processes each message that it receives.

This topic contains the following sections:

- "Propagating the message"
- "Constructing OutputLocalEnvironment" on page 1566
- "Constructing the Output message" on page 1567

Propagating the message

1. If an MQ Message Descriptor header (MQMD) is present in the input tree, the MQGet node uses it. If not, the node creates a default MQMD.
2. The node also creates a default MQ Get Message Options (MQGMO) structure based on the values that you have set for the node properties. If an MQGMO is present in the input tree, the node uses its content to modify the default one.

   When you include an MQGMO to override the default one, you must specify all the options that you are replacing. For example, if you set the option field to MQGMO_CONVERT, that value overrides all options that you set with the WebSphere Message Broker Toolkit. If you do not use an overriding MQGMO, WebSphere Message Broker uses the following values:
   - If `Wait interval` is not zero, MQGMO_WAIT is set; otherwise, MQGMO_NOWAIT is used.
   - If `Transaction mode` is set to `Yes`, MQGMO_SYNCPOINT is used.
   - If `Transaction mode` is set to `No`, MQGMO_NOSYNCPOINT is used.
   - If `Transaction mode` is set to `Automatic`, MQGMO_SYNCPOINT_IF_PERSISTENT is used.
   - The only other option that is used by default in the node properties is MQGMO_COMPLETE_MSG, which is set if `Transaction mode` is set to `Yes` or `No`. This option is not set when your broker is running on z/OS.
   - No other options are used by default.
3. The node makes the MQGet call to WebSphere MQ.
4. The node analyzes the completion code (CC), and propagates the message to the appropriate terminal:

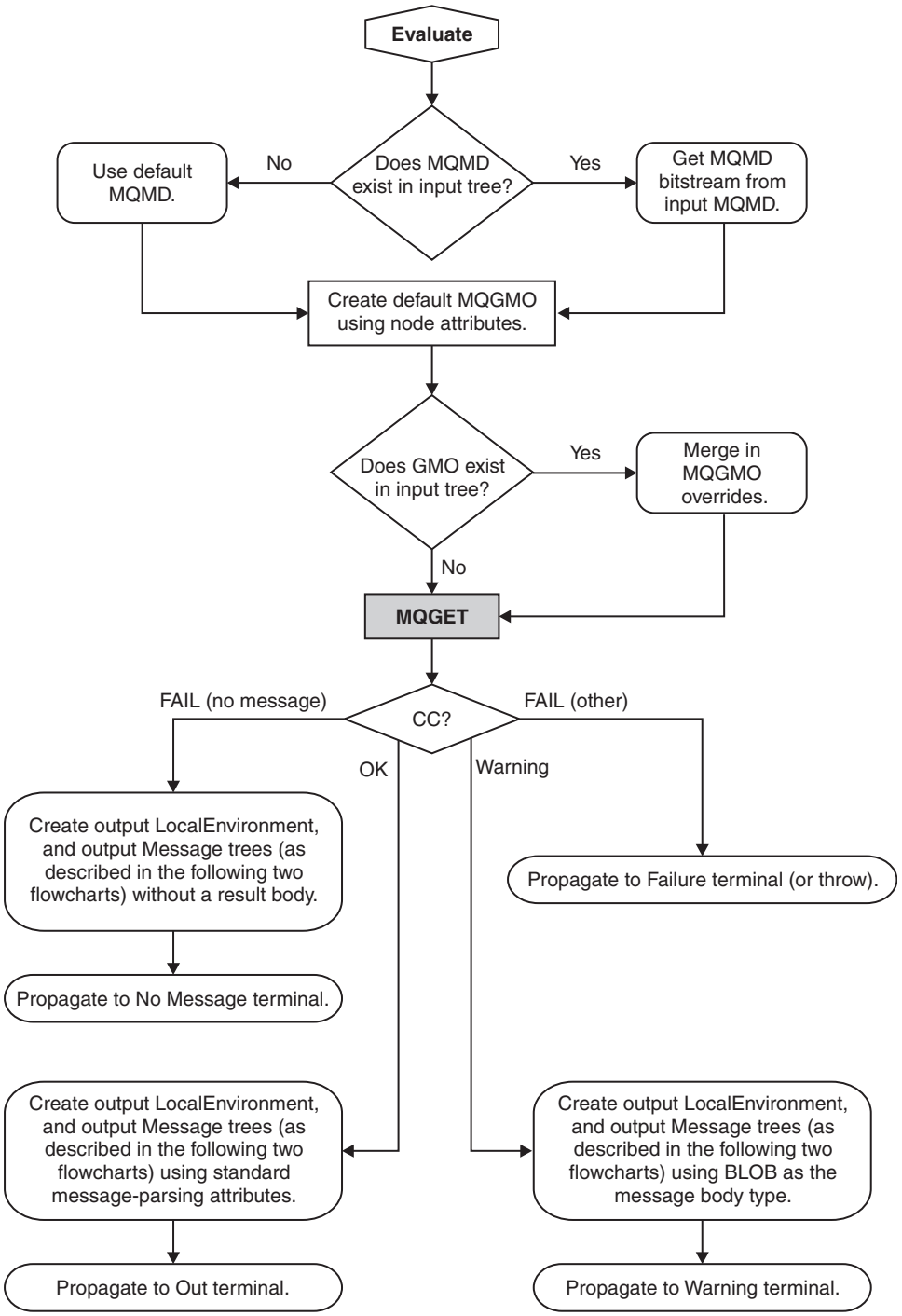
OK The node creates the output LocalEnvironment and the output message trees using standard message-parsing techniques, then propagates the message to the Out terminal.

Warning
The node creates the output LocalEnvironment and the output message trees using BLOB as the message body type, then propagates the message to the Warning terminal, if it is connected. If the Warning terminal is not connected, no propagation occurs, and the flow ends.

Fail (no message)
The node creates the output LocalEnvironment and the output message trees by copying the input trees, then propagates the message to the No Message terminal, if it is connected. If the No Message terminal is not connected, no propagation occurs. The output message that is propagated to the No Message terminal is constructed from the input message only, according to the values of the `Generate Mode` property, and the `Copy Message` or `Copy Local Environment` properties.

Fail (other)
The node propagates the message to the Failure terminal. If the Failure terminal is not connected, the broker throws an exception and returns control to the closest upstream node that can process it. For more information, see "Handling errors in message flows" on page 2823.

The following diagram shows this processing:



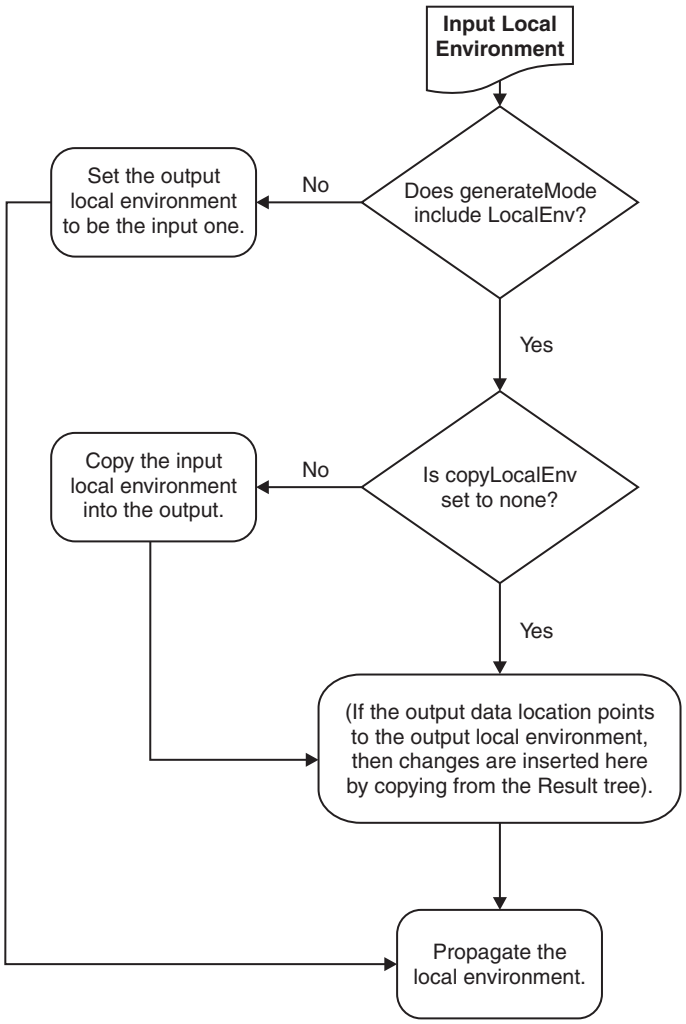
Constructing OutputLocalEnvironment

1. If the `Generate Mode` property on the MQGet node is set to an option that does not include `LocalEnvironment`, the node copies the input local environment tree to the output local environment tree.

   If this copy is made, any updates that are made in this node to the output local environment tree are not propagated downstream.
2. If the `Copy Local Environment` property is set to an option other than `None`, the node copies the input local environment tree to the output local environment tree.

3. If the output data location points to the output local environment tree, the node applies changes in that tree by copying from the result tree.
4. The local environment tree is propagated.

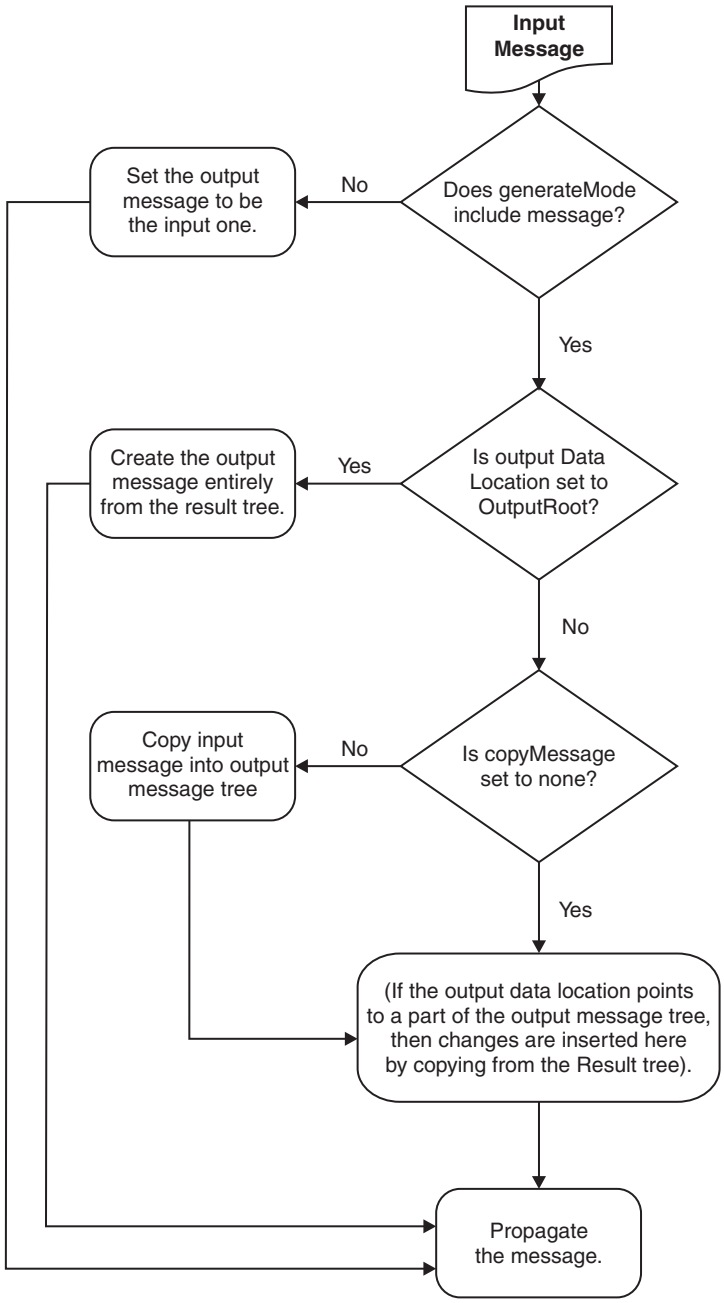
The following diagram shows this processing:



Constructing the Output message

1. If the `Generate Mode` property on the MQGet node is set to an option that does not include `Message`, the node copies the input message tree to the output message tree. Go to step 5.
2. If the `Output Data Location` property is set to `OutputRoot`, the node creates the output message tree entirely from the result tree. Go to step 5.
3. If the `Copy Message` property is set to a value other than `None`, the node copies the input message tree to the output message tree.
4. If the `Output Data Location` property points to a part of the output message tree, the node applies changes in that tree by copying from the result tree at the point that is defined by the `Result Data Location` property.
5. The message tree is propagated.

The following diagram shows this processing:



For an example of how this processing is implemented in a message flow, see "A request-response scenario that uses an MQGet node" on page 1569.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related reference:

"A request-response scenario that uses an MQGet node" on page 1569
Read about a scenario in which an MQGet node is used in a request-response flow, and how the node processes the input messages to construct the output messages, based on both the content of the local environment tree and the input parameters that you set.

"MQGet node" on page 4578
Use the MQGet node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and the MQI and AMI application programming interfaces.

A request-response scenario that uses an MQGet node:

Read about a scenario in which an MQGet node is used in a request-response flow, and how the node processes the input messages to construct the output messages, based on both the content of the local environment tree and the input parameters that you set.

A request-response flow is a specialized form of a point-to-point application. For a general description of these applications, see "Nodes for connectivity" on page 1028. For an example of a request-response message flow, see the following sample:

- Coordinated Request Reply

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

You can include an MQGet node anywhere in a message flow, including a flow that implements a request-response scenario. The MQGet node receives an input message on its input terminal from the preceding node in the message flow, issues an MQGET call to retrieve a message from the WebSphere MQ queue that you have configured in its properties, and builds a result message tree. Finally, it uses the input tree and the result tree to create an output tree that is then propagated to its Output, Warning, or Failure terminal, depending on the configuration of the node and the result of the MQGET operation.

How the MQGet node handles the local environment:
The MQGet node examines the local environment tree that is propagated from the preceding node, uses the content that is related to the MQGMO (MQ Get Message Options) and the MQMD (MQ Message Descriptor header), and updates the local environment:

- The node reads the MQGMO structure from `${inputMQParmsLocation}.MQGMO.*`.
- The node copies the WebSphere MQ completion and reason codes to `${outputMQParmsLocation}.CC` and `${outputMQParmsLocation}.RC`.
- The node writes the complete MQGMO that is used for the MQGET call into `${outputMQParmsLocation}.MQGMO` if `${inputMQParmsLocation}.MQGMO` exists in the input tree.
- The node writes the MQMD that is passed to the MQGET call (that contains the values that are specified in the input message or are generated by the node) into `${inputMQParmsLocation}.MQMD`, deleting any existing content.

Set the value to `${inputMQParmsLocation}` in the MQGet node property Input MQ Parameters Location on the **Request Properties** tab.

Set the value to `${outputMQParmsLocation}` in the MQGet node property Output MQ Parameters Location on the **Result Properties** tab.

For more information about these properties, see "MQGet node" on page 4578.

In summary:

${inputMQParmsLocation}

- *QueueName*: Optional override for MQGet node *Queue Name* property
- *InitialBufferSize*: Optional override for MQGet node *Initial Buffer Size* property
- *MQGMO.**: Optional MQGET message options that are used by the MQGet node

${outputMQParmsLocation}

- *CC*: MQGET call completion code
- *RC*: MQGET call result code
- *MQGMO.**: MQGET message options that are used if present in `${inputMQParmsLocation}`
- *MQMD*: unparsed MQ Message Descriptor for received messages[1]
- *Browsed*: Set to true if the message is browsed. Not present if the message is removed from the queue

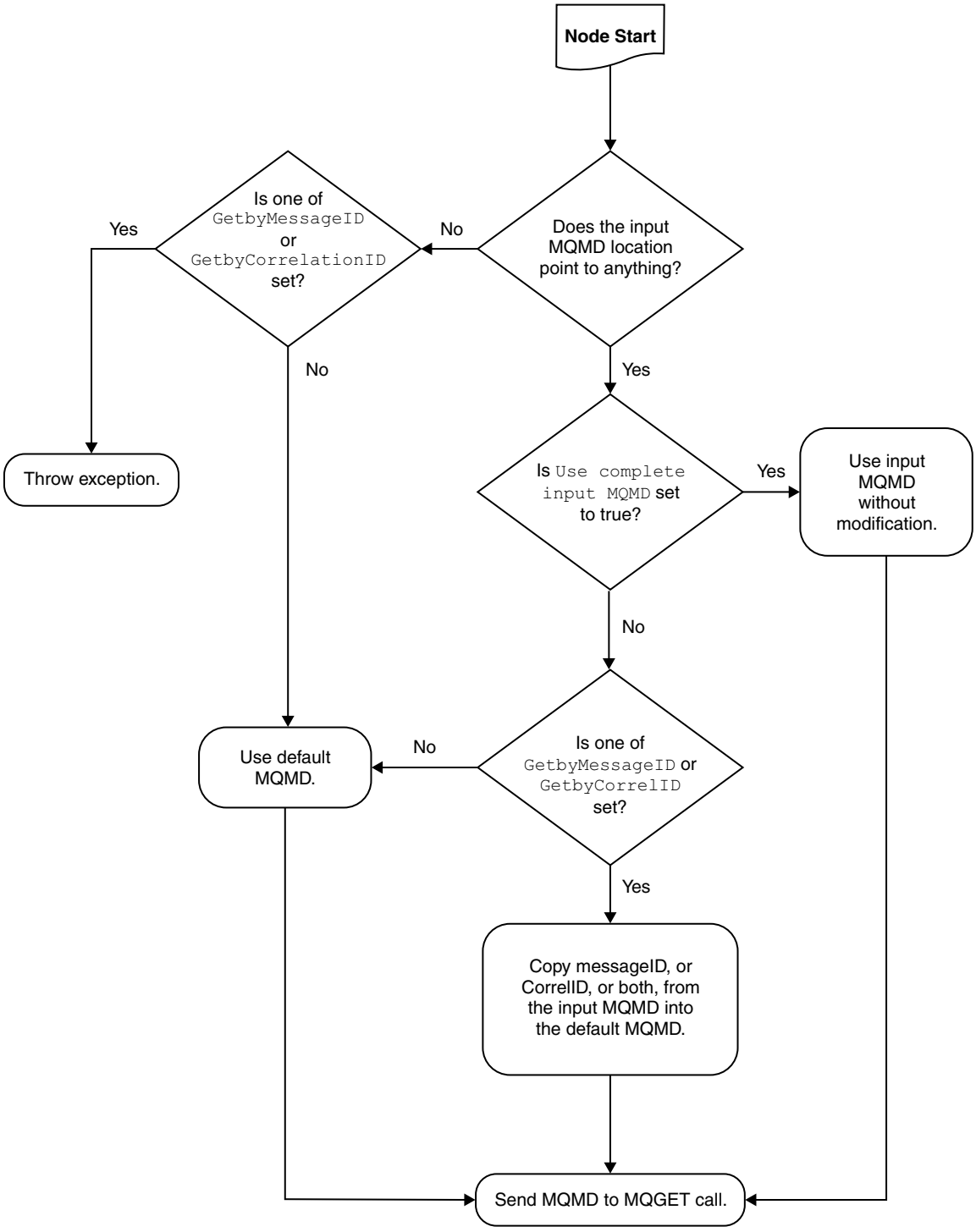
You can parse the MQMD (for example, by using ESQL), where `${outputMQParmsLocation}` is LocalEnvironment.MQ.GET:

```
DECLARE ptr REFERENCE TO OutputLocalEnvironment.MyMQParms;
CREATE FIRSTCHILD OF ptr DOMAIN('MQMD') PARSE(InputLocalEnvironment.MQ.GET.MQMD)
```

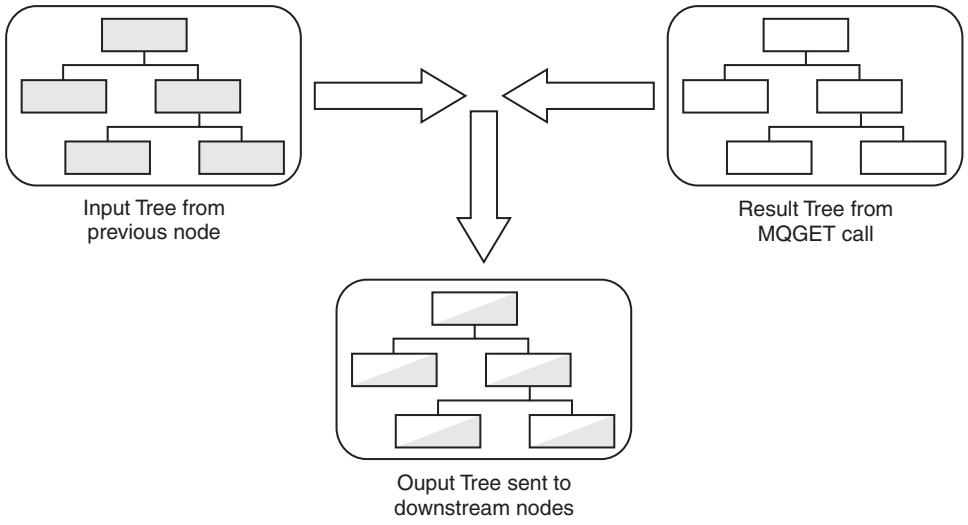
How the MQMD for the MQGET call is constructed:

- A default MQMD is prepared. For further information about the MQMD, see the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online.
- If you do not supply an input MQMD, the default MQMD is used.
- If you do supply an input MQMD, the default MQMD is used after the following modifications:
  - If the property `Use all input MQMD fields` is set, all MQMD fields supplied are copied into the default MQMD from the input MQMD.
  - If the property `Use all input MQMD fields` is not set and the properties `Get by Message ID` or `Get by Correlation ID` are selected, the respective IDs are copied into the default MQMD from the input MQMD.

The following diagram shows how the MQGet node constructs the MQMD that is used on the call to WebSphere MQ:



How the output message tree is constructed:
The following diagram outlines how the MQGet node constructs the output message tree, combining the input tree from the previous node with the result tree from the MQGET call:



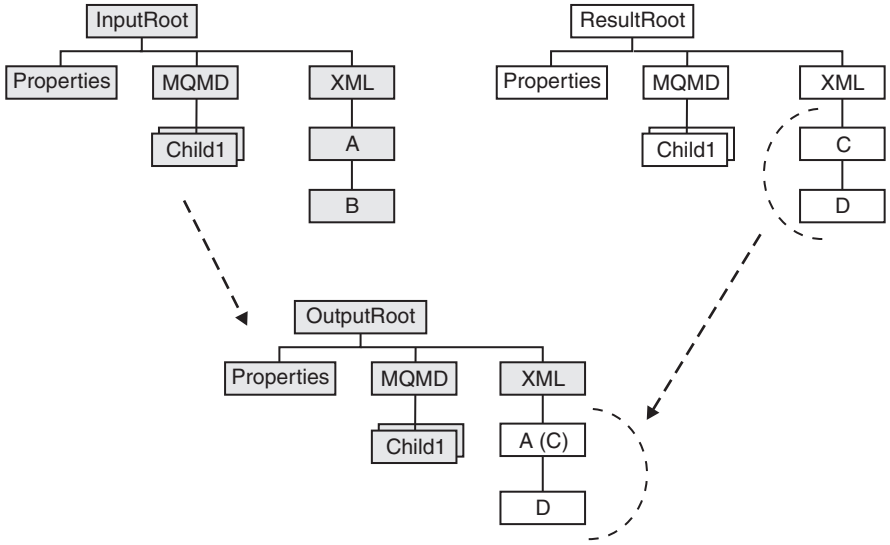
In this example, the MQGet node properties are configured as shown in the following table.

Property	**Action**
Copy Message	Copy Entire Message
Generate Mode	Message
Output Data Location	`OutputRoot.XMLNS.A`
Result Data Location	`ResultRoot.XMLNS.C`

The MQGet node constructs the output tree according to the following sequence:

1. The whole of the input tree is copied to the output tree, including the XML branch with child A, and A's child B.
2. From the result tree, the XML branch's child C, and C's child D, are put into the output tree at position `OutputRoot.XMLNS.A`. Any previous content of A (values and children) is lost, and replaced with C's content, including all values and children it has, in this case child D.
3. The position in the output tree retains the name A.

The following diagram shows this behavior:



For some examples of message trees that are constructed by the MQGet node according to the rules described above, see "MQGet node message tree examples."

Related concepts:

"Nodes for connectivity" on page 1028
WebSphere Message Broker supports direct connections from applications, and can send direct requests to other application endpoints. WebSphere Message Broker can also connect to various subsystems including WebSphere MQ, files, and databases, to read and write existing application data.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related reference:

"MQGet node message tree examples"
The MQGet node generates message trees based on the input message assembly that it receives, and the options that you have set on the node properties.

"How the MQGet node processes messages" on page 1564
The MQGet node processes each message that it receives.

"MQGet node" on page 4578
Use the MQGet node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and the MQI and AMI application programming interfaces.

Related information:

WebSphere MQ Version 7 Information Center online

MQGet node message tree examples:

The MQGet node generates message trees based on the input message assembly that it receives, and the options that you have set on the node properties.

The message trees, shown in the following table, are generated according to the rules described in “A request-response scenario that uses an MQGet node” on page 1569.

With a message assembly like this:	The message that the MQGet node returns is:
InputRoot **MQMD** {input message MQMD} **MQRFH2** {input message MQRFH2} **XMLNS** {input message body} **InputLocalEnvironment** **MQ** **GET** **MQGMO** MatchOptions = MQMO_MATCH_CORREL_ID **MQMD (with no children)** **Variables** **MQMD** {input MQMD} (with CorrelID = {correct Correlation ID as binary})	**ResultRoot** **MQMD** {result message MQMD} **MQRFH2** {result message MQRFH2} **XML** {result message body}

With the following node property settings:	The resulting output message assembly is:
Input MQMD Location InputLocalEnvironment.Variables.MQMD **Copy Message** Copy Entire Message **Copy Local Environment** Copy Entire LocalEnvironment **Generate Mode** Message and LocalEnvironment **Output Data Location** InputLocalEnvironment.Variables.ReturnedMessage	**OutputRoot** **MQMD** {input message MQMD} **MQRFH2** {input message MQRFH2} **XMLNS** {input message body} **OutputLocalEnvironment** **MQ** **GET** **MQGMO** {MQGMO used for MQGET} **MQMD** {MQMD used for MQGET} **CC = 0** **RC = 0** **Variables** **MQMD** {input MQMD} (with CorrelID = {correct Correlation ID as binary}) **ReturnedMessage** **MQMD** {result message MQMD} **MQRFH2** {result message MQRFH2} **XML** {result message body}

With the following node property settings:	The resulting output message assembly is:
Result Data Location ResultRoot.XML	**OutputRoot** **MQMD** {input message MQMD} **MQRFH2** {input message MQRFH2} **XMLNS** {input message body} **OutputLocalEnvironment** **MQ** **GET** **MQGMO** {MQGMO used for MQGET} **MQMD** {MQMD used for MQGET} **CC = 0** **RC = 0** **Variables** **MQMD** {input MQMD} (with CorrelID = {correct Correlation ID as binary}) **ReturnedMessage (with any attributes and value from ResultRoot.XML)** {result message body} This tree is effectively the result of doing an assignment from `${resultDataLocation}` to `${outputDataLocation}`. The value of the source element is copied, as are all children including attributes.

With the following node property settings:	The resulting output message assembly is:
Copy Local Environment None	**OutputRoot** **MQMD** {input message MQMD} **MQRFH2** {input message MQRFH2} **XMLNS** {input message body} **OutputLocalEnvironment** **MQ** **GET** **MQGMO** {MQGMO used for MQGET} **MQMD** {MQMD used for MQGET} **CC = 0** **RC = 0** **Variables** **ReturnedMessage (with any attributes and value from ResultRoot.XML)** {result message body} This tree has the MQMD that is used for the MQGET call in the OutputLocalEnvironment, because the input MQ parameters location had an MQMD element under it. Even though the input tree is not copied, the presence of the MQMD element causes the MQMD that is used for the MQGET call to be placed in the output tree.

With the following node property settings:	The resulting output message assembly is:
Output Data Location <blank> **Copy Local Environment** Copy Entire Local Environment	**OutputRoot** **MQMD** {result message MQMD} **MQRFH2** {result message MQRFH2} **XMLNS** {result message body} **OutputLocalEnvironment** **MQ** **GET** **MQGMO** {MQGMO used for MQGET} **MQMD** {MQMD used for MQGET} **CC = 0** **RC = 0** **Variables** **MQMD** {input MQMD} (with CorrelID = {correct Correlation ID as binary}) The value that you set for the Copy Message property makes no difference to the eventual output tree in this case.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related reference:

"A request-response scenario that uses an MQGet node" on page 1569
Read about a scenario in which an MQGet node is used in a request-response flow, and how the node processes the input messages to construct the output messages, based on both the content of the local environment tree and the input parameters that you set.

"How the MQGet node processes messages" on page 1564
The MQGet node processes each message that it receives.

"MQGet node" on page 4578
Use the MQGet node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and the MQI and AMI application programming interfaces.

Processing HTTP messages

Hypertext Transfer Protocol (HTTP) is an Internet protocol that is used to transfer and display hypertext and XML documents on the Web.

You can configure message flows that include the HTTP or SOAP nodes to access the HTTP transport to work with the following resources:

- SOAP-based Web services
- Other Web services standards, such as REST or XML-RPC
- General HTTP messaging, where the payload might be XML

HTTP nodes can process non-secure (HTTP) messages and secure (HTTPS or HTTP over SSL) messages.

For SOAP-based Web services, several advantages exist if you use the SOAP nodes and the SOAP message domain instead of the HTTP transport nodes and XMLNSC message domain.

- Support for WS-Addressing, WS-Security and SOAP headers.
- A common SOAP logical tree format, independent of the bitstream format.
- Runtime checking against WSDL.
- Automatic processing of SOAP with Attachments (SwA).
- Automatic processing of Message Transmission Optimization Mechanism (MTOM).

Although the HTTP nodes can process SwA messages, you must use the MIME message domain and design your flow to handle the attachments explicitly, and use custom logic to extract and parse the SOAP.

For more information about using SOAP messages and nodes, see “What is SOAP?” on page 1604

You can choose how your HTTP nodes interact with the TCP/IP network:

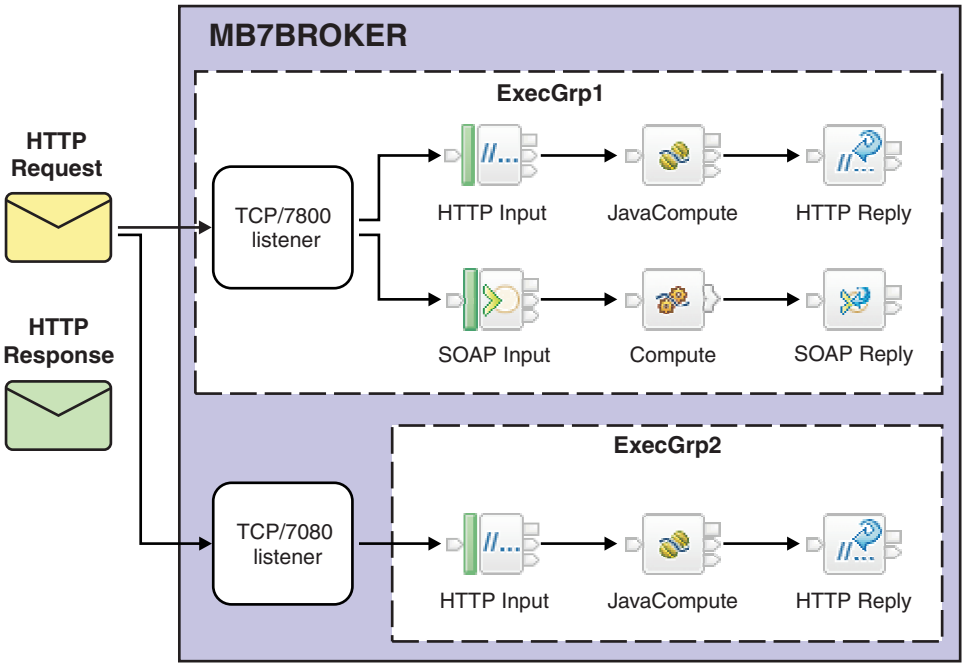
- You can use the broker-wide listener, which receives HTTP messages on one port, and HTTPS messages on a second port.

  This option is set as the default configuration for both existing and new brokers.
- You can use the listener that is embedded within each execution group, which also has two ports for HTTP and HTTPS messages.
- You can use a mixture of broker and execution group listeners by keeping the broker listener active, and configuring a subset of execution groups to use the embedded listener.

For more information about why you might choose each option, and how to configure them, see “HTTP listeners” on page 1589.

SOAP nodes always use the listener that is embedded in the execution group, and only this listener; you cannot configure them to use the broker-wide listener.

The following diagram shows the use of both types of listener, configured on default ports, for HTTP messages.



You must always use the correct reply node that matches your input node; you cannot combine an HTTPReply node with a SOAPInput, or a SOAPReply node with an HTTPInput node. The broker generates an exception when the reply is attempted.

You can include the reply node in the same message flow, or in a different message flow:

- If you have configured the HTTPInput node to use the execution group listener, you must deploy the second message flow to the same execution group.

  This restriction always applies to SOAPInput nodes, which use only the execution group listener.
- If you have configured the HTTPInput node to use the broker-wide listener, you can deploy the second message flow to any other execution group defined to the broker.
- You must pass the correct reply identifier from the input message flow to the reply node.

If you choose to handle HTTP messages by using the execution group listener, you must carefully check the URL specifications in your HTTPInput and SOAPInput nodes. If both URL specifications match an incoming message, the wrong type of node might get the message, and processing might fail or produce unexpected results. This situation occurs if you specify identical values for the Path suffix for URL properties of the HTTPInput node and the SOAPInput node. It can also occur if you use wildcards in either or both specifications, and an incoming message matches both properties.

For more information about using the WebSphere Broker HTTP Transport, see the following topics:

- “Working with HTTP flows” on page 1585
- “HTTP listeners” on page 1589
- “HTTP message format” on page 1582
- “HTTP headers” on page 1583

- "Web services example messages" on page 1599

For information about using HTTPS, see "Implementing SSL authentication" on page 504.

You can also use the HTTP Proxy servlet in an external Web servlet container to provide listener support for a larger number of concurrent HTTP sessions. For more information about the servlet and its uses, see "HTTP proxy servlet overview" on page 856.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"What is SOAP?" on page 1604
SOAP is an XML message format used in Web service interactions. SOAP messages are typically sent over HTTP or JMS, but other transport protocols can be used. The use of SOAP in a specific Web service is described by a WSDL definition.

"HTTP listeners" on page 1589
You can choose between broker-wide listeners and execution group (embedded) listeners to manage HTTP messages in your HTTP flows. Learn about the two types of listener, how ports are assigned to them, and how you can switch from one to the other for individual execution groups.

"HTTP proxy servlet overview" on page 856
The HTTP proxy servlet is a Java servlet that you can use in an external web servlet container, such as IBM WebSphere Application Server or Apache Tomcat, to receive HTTP requests from web services client applications, to replace the support provided by the broker and execution group HTTP listeners.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Implementing SSL authentication" on page 504
Use SSL authentication to enhance security in your broker environment.

"Resolving problems when you use HTTP and SOAP nodes" on page 3407
Use the advice given here to help you to resolve common problems that can arise when you develop Web Services message flows that contain HTTP and SOAP nodes.

Chapter 11, "Packaging and deploying," on page 3209
Package resources that you create in the WebSphere Message Broker Toolkit, such as message flows, and deploy them to execution groups on brokers.

Related reference:

"HTTPInput node" on page 4474
Use the HTTPInput node to receive an HTTP message from an HTTP client for processing by a message flow.

"HTTPReply node" on page 4484
Use the HTTPReply node to return a response from the message flow to an HTTP client. This node generates the response to an HTTP client from which the input message was received by the HTTPInput node, and waits for confirmation that it has been sent.

"HTTPRequest node" on page 4488
Use the HTTPRequest node to interact with a web service.

Related information:

"Web services: when to use SOAP or HTTP nodes"
HTTP and SOAP nodes can both be used to interact with Web services. Typically you use SOAP nodes when working with SOAP-based Web services.

Web services: when to use SOAP or HTTP nodes

HTTP and SOAP nodes can both be used to interact with Web services. Typically you use SOAP nodes when working with SOAP-based Web services.

For SOAP-based Web services, several advantages exist if you use the SOAP nodes and the SOAP message domain instead of the HTTP transport nodes and XMLNSC message domain.

- Support for WS-Addressing, WS-Security and SOAP headers.
- A common SOAP logical tree format, independent of the bitstream format.
- Runtime checking against WSDL.
- Automatic processing of SOAP with Attachments (SwA).
- Automatic processing of Message Transmission Optimization Mechanism (MTOM).

Although the HTTP nodes can process SwA messages, you must use the MIME message domain and design your flow to handle the attachments explicitly, and use custom logic to extract and parse the SOAP.

Cases where it might be better to use HTTP nodes include:

- Message flows in which a single request node handles multiple SOAP request and responses from more than one WSDL.
- Message flows that interact with Web services that use different standards, such as REST or XML-RPC.
- Message flow that never use WS-Addressing, WS-Security, SwA, or MTOM.

Related concepts:

"What is a Web service?" on page 1603
A Web service is defined by the World Wide Web Consortium (W3C) as a software system designed to support interoperable machine-to-machine interaction over a network.

"Processing HTTP messages" on page 1579
Hypertext Transfer Protocol (HTTP) is an Internet protocol that is used to transfer and display hypertext and XML documents on the Web.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

HTTP message format

An HTTP message contains components that are appropriate to its type.

The bit stream containing headers and body is parsed and represented within the message tree when an input request is received by an HTTPInput node, or when a response from a Web service is received by the HTTPRequest node. A bit stream is created by parsers from the appropriate parts of the message tree when a reply is sent to the client by the HTTPReply node, or when a message is sent by the HTTPRequest node. For further details about these actions, see the individual node descriptions.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Generate WSDL" on page 1274
A *Web Services Description Language* (WSDL) document specifies the interface to a web service, and enables a web service client to start it. A WSDL document that is generated from a message model defines web service requests and responses in terms of the messages that you have defined in that message model.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Generating a WSDL definition from a message set" on page 2968
To ensure the highest interoperability of your Web services, use the document style of WSDL whenever possible. If rpc style WSDL is necessary, use literal encoding.

Chapter 11, "Packaging and deploying," on page 3209
Package resources that you create in the WebSphere Message Broker Toolkit, such as message flows, and deploy them to execution groups on brokers.

HTTP headers

When an HTTPInput or HTTPRequest node receives a message, it parses the HTTP headers to create elements in the message tree. When an HTTPReply or HTTPRequest node sends a message, it parses the HTTP headers from the message tree into a bit stream.

The HTTP headers in a message depend on the type of message that is processed. There are four message types recognized in a message flow, and a parser is associated with each of these.

1. Input. An input message is received by the HTTPInput node from a client. The HTTP headers in the input message (data up to and including the CRLFCRLF) are parsed by the HTTPInput parser and are included in the message tree under the correlation name HTTPInput. The headers shown in the following table are expected in an input message; others might also be present.

Header	Content	Example
Host	The host name to which the client issued the message.	localhost
Content-Length	The length of the body of the input message in decimal (that follows the CRLFCRLF after the last header).	520
Content-Type	The type of the body data.	text/xml; charset=utf-8
SOAPAction		"" (empty string)

The headers in the following table might also be automatically generated by the HTTPInput node depending on the request.

Header	Content	Example
X-Original-HTTP-Command	An expanded version of the original inbound request	POST http://localhost:7800/Wss001/services/Wss001 HTTP/1.1
X-Remote-Addr	The IP address of the client (or proxy if the client is connecting through a proxy)	127.0.0.1

Header	Content	Example
X-Remote-Host	The host name or address of the client (or proxy if the client is connecting through a proxy)	localhost
X-Server-Name	The broker's machine name	localhost
X-Server-Port	The broker's port	7800
X-Query-String	The query string if present in the inbound URL (optional)	a=b&x=y
X-Scheme	The scheme through which the client is connected, either http or https	http

2. Reply. A reply message is sent by the HTTPReply node to the client that sent the corresponding input message. The headers in the reply message are created in the message tree under the correlation name HTTPReply, which is also the name of the parser used to parse that part of the message tree to a bit stream. You can create your own HTTPReply header in a Compute node, or you can configure the HTTPReply node to create it by using default values, or values taken from the HTTPReply or HTTPResponse trees in the input message, or both.

   You can set the HTTPReply Status Code in the local environment; for more information, see the instructions for setting the HTTP Status Code for a reply in "Working with HTTP flows" on page 1585.

   If the HTTPReply node creates a default HTTPReply header, it contains the headers and values shown in the following table.

Header	Value
Content-Length (if present in the input message)	The calculated length of the reply message body in decimal.
Content-Type	text/xml; charset=*ccsid of the message body*

3. Request. A request message is sent by the HTTPRequest node. The HTTP headers in this message must be created in the message tree under the correlation name HTTPRequest, and are parsed by the HTTPRequest parser when the message tree is parsed to a bit stream. You can create your own HTTPRequest header in a Compute node, or you can configure the HTTPRequest node to create it using default values, or values taken from the HTTPInput or HTTPRequest trees in the input message, or both. If the HTTPRequest node creates a default HTTPRequest header, it contains the headers and values shown in the following table.

Header	Value
Host	Value set in the *Default Web Service URL* property.
Content-Length	The calculated length of the request message body in decimal.
Content-Type	text/xml; charset=*ccsid of the message body*
SOAPAction	"" (empty string)
Content-Encoding	"gzip" or "deflate" if the `Use compression` property is set to `gzip`, `zlib (deflate)`, or `deflate`.
Accept-Encoding	"gzip, deflate" if the `Accept compressed responses by default` property is selected.

4. Response. A response message is received by the HTTPRequest node from the application to which the corresponding request message was sent. The HTTP headers in the response message (data up to and including the CRLFCRLF) are parsed by the HTTPResponse parser and are included in the message tree under the correlation name HTTPResponse. The header shown in the following table is expected in a response message (though not required); others might also be present.

Header	Content	Example
Content-Length	The length of the response message body in decimal.	1585

"Web services example messages" on page 1599 provides example messages that include these headers.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Generate WSDL" on page 1274
A *Web Services Description Language* (WSDL) document specifies the interface to a web service, and enables a web service client to start it. A WSDL document that is generated from a message model defines web service requests and responses in terms of the messages that you have defined in that message model.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Generating a WSDL definition from a message set" on page 2968
To ensure the highest interoperability of your Web services, use the document style of WSDL whenever possible. If rpc style WSDL is necessary, use literal encoding.

Chapter 11, "Packaging and deploying," on page 3209
Package resources that you create in the WebSphere Message Broker Toolkit, such as message flows, and deploy them to execution groups on brokers.

Related reference:

"HTTPReply node" on page 4484
Use the HTTPReply node to return a response from the message flow to an HTTP client. This node generates the response to an HTTP client from which the input message was received by the HTTPInput node, and waits for confirmation that it has been sent.

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"HTTPRequest node" on page 4488
Use the HTTPRequest node to interact with a web service.

Working with HTTP flows

Read this information if you are using HTTP message flows to interact with HTTP applications, including web services and RESTful services.

About this task

You might find it useful to read this information with the "Web services scenarios" on page 1620 section.

- Using secure connections with HTTPS

- Setting the HTTP Status Code for a reply
- Using LocalEnvironment.Destination.HTTP.RequestIdentifier
- Setting the HTTPRequest node URL dynamically
- Setting `Generate default HTTP headers from reply or response` for the HTTPReply node
- Setting `Generate default HTTP headers from input` for the HTTPRequest node

You must decide which listener you want the HTTP nodes to use:

- The broker listener (the httplistener component) has an HTTPConnector for handling HTTP messages, and an HTTPSConnector for handling HTTPS (HTTP over SSL) messages. Each connector has its own assigned port.

  By default, all HTTPInput and HTTPReply nodes use the broker-wide listener, which interfaces with the HTTP nodes by using a WebSphere MQ queue.
- An embedded listener is part of each execution group. The embedded listener has an HTTPConnector and an HTTPSConnector, and each connector has its own assigned port.

  You can configure one or more execution groups so that all HTTPInput and HTTPReply that you deploy to that execution group use the embedded listener.

For a discussion of the advantages of each listener, see "HTTP listeners" on page 1589.

You can use `ProxyConnectHeaders` only with HTTPS (SSL) connections; these headers do not work with HTTP connections.

Using secure connections with HTTPS

For information about setting up HTTPS connections, see "Implementing SSL authentication" on page 504.

Setting the HTTP Status Code for a reply

The default HTTP Status Code is 200, which indicates success. If you want a different status code to be returned, take one of the following actions:

- Set your status code in the field Destination.HTTP.ReplyStatusCode in the local environment tree (correlation name OutputLocalEnvironment). This field overrides any status code that is set in an HTTPResponseHeader header. This action is the preferred option, because it provides the greatest flexibility.
- Set your status code in the field X-Original-HTTP-Status-Code in the HTTPReplyHeader header.
- Set your status code in the field X-Original-HTTP-Status-Code in the HTTPResponseHeader header. This option is useful if you include an HTTPRequest node before the HTTPReply node in your flow, because the HTTPResponseHeader header is created for you. In this scenario, an HTTPResponseHeader header has been created in the logical tree, representing the HTTP headers in the response from another Web service. If you have selected the `Generate default HTTP headers from reply or response` property on the HTTPReply node, values for the response header are set as default values when the reply message is created.

Using LocalEnvironment.Destination.HTTP.RequestIdentifier

When the HTTPInput node receives an input request message, it sets the local environment field Destination.HTTP.RequestIdentifier to a unique value that identifies the Web service client that sent the request. You can refer to this value, and you can save it to another location if appropriate.

For example, if you design a pair of message flows that interact with an existing WebSphere MQ application (as described in "Broker calls existing Web service" on page 1621), you can save the identifier value in the request flow, and restore it in the reply flow, to ensure that the correct client receives the reply. If you use this technique, you must not change the data, and you must retain the data as a BLOB.

The HTTPReply node extracts the identifier value from the local environment tree and sets up the reply so that it is sent to the specific client. However, if you are using an HTTPReply node in a flow that does not have an HTTPInput node, and this field has been deleted or set incorrectly, message BIP3143S is issued.

If you design a message flow that includes both an HTTPInput and an HTTPReply node, the identifier value is set into the local environment by the HTTPInput node, but the HTTPReply node does not use it. Therefore, if your message flow includes both HTTP nodes and a Compute node in the same flow, you do not have to include the local environment tree when you specify which components of the message tree are copied from input message to output message by the Compute node (the `Compute mode` property).

Setting the HTTP request URL dynamically

You can set the property `Default Web service URL` on the HTTPRequest node to determine the destination URL for a Web service request. You can configure a Compute node before the HTTPRequest node within the message flow to override the value set in the property. Code ESQL that stores a URL string in LocalEnvironment.Destination.HTTP.RequestURL; the node retrieves and uses the URL string in place of the node property value.

Although you can also set the request URL in the special header X-Original-HTTP-URL in the HTTPRequestHeader header section of the request message (which overrides all other settings) in a Compute node, use the local environment tree content for this purpose for greater flexibility.

Setting `Generate default HTTP headers from reply or response` for the HTTPReply node

If you select `Generate default HTTP headers from reply or response` in the HTTPReply node properties, the node includes a minimum set of headers in the response that is sent to the Web service client.

To set headers explicitly, create them in an HTTPReplyHeader header. For example, a Compute node propagates a message in the XMLNSC domain and modifies the Content-Type as follows:

```
CALL CopyMessageHeaders();
SET OutputRoot.HTTPReplyHeader."Content-Type" = 'text/xml';
SET OutputRoot.XMLNSC = InputRoot.XMLNSC;
```

Do not use the ContentType property to set the Content-Type unless you are working in the MIME domain. The ContentType property is intended to set the value of Content-Type used in MIME.

The full set of HTTP headers used in the reply is built by selecting the headers using the algorithm defined in the following steps:

1. Select one or more headers in an HTTPReplyHeader header.
2. If no Content-Type header is yet defined, create one by using a non-empty value in the ContentType property.

3. Select one or more headers in an HTTPResponseHeader header (an HTTPResponseHeader header is propagated on return from an HTTPRequest node).
4. If no Content-Type header is yet defined, create one with the default value `text/xml; charset=`*`ccsid of the message body`*.
5. Create or overwrite the Content-Length header.

Attention: The HTTPReply node always rewrites the Content-Length header, even if you have cleared `Generate default HTTP headers from reply or response`. This action ensures that the content is correct.

If an HTTPReplyHeader header section existed in the message received by the HTTPReply node, and the Output terminal of the HTTPReply node is connected, the HTTPReplyHeader header section is updated with all changed or added values.

Setting `Generate default HTTP headers from input` for the HTTPRequest node

If you select `Generate default HTTP headers from input` in the HTTPRequest node properties, the node includes a minimum set of headers in the request that is sent to the server.

To explicitly set headers, create them in an HTTPRequestHeader header. For example, a Compute node propagating a message in the XMLNSC domain can modify the Content-Type as follows:

```
CALL CopyMessageHeaders();
SET OutputRoot.HTTPRequestHeader."Content-Type" = 'text/xml';
SET OutputRoot.XMLNSC = InputRoot.XMLNSC;
```

Do not use the ContentType property to set the Content-Type unless you are working in the MIME domain. The ContentType property is intended to set the value of Content-Type used in MIME.

The full set of HTTP headers used in the request is built by selecting the headers using the algorithm defined in the following steps:

1. Set the Host header, based on either the request URL or the incoming HTTPRequestHeader header section of the message.
2. Select one or more headers in an HTTPRequestHeader header.
3. If no Content-Type header is yet defined, create one by using a non-empty value in the ContentType property.
4. Select one or more headers in an HTTPInputHeader header (an HTTPInputHeader header is created automatically by an HTTPInput node).
5. If no Content-Type header is yet defined, create one with the default value `text/xml; charset=`*`ccsid of the message body`*
6. If no SOAPAction header is yet defined, create one with the default value `''`.
7. Create or overwrite the Content-Length header.

Attention: The HTTPRequest node always rewrites the Content-Length header, even if you have cleared `Generate default HTTP headers from input or request`. This action ensures that the content is correct.

If an HTTPRequestHeader header exists in the received message, the HTTPRequestHeader header is updated with all changed or added values.

Related concepts:

"Processing HTTP messages" on page 1579
Hypertext Transfer Protocol (HTTP) is an Internet protocol that is used to transfer and display hypertext and XML documents on the Web.

"HTTP listeners"
You can choose between broker-wide listeners and execution group (embedded) listeners to manage HTTP messages in your HTTP flows. Learn about the two types of listener, how ports are assigned to them, and how you can switch from one to the other for individual execution groups.

"Generate WSDL" on page 1274
A *Web Services Description Language* (WSDL) document specifies the interface to a web service, and enables a web service client to start it. A WSDL document that is generated from a message model defines web service requests and responses in terms of the messages that you have defined in that message model.

Related tasks:

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

Chapter 11, "Packaging and deploying," on page 3209
Package resources that you create in the WebSphere Message Broker Toolkit, such as message flows, and deploy them to execution groups on brokers.

"Checking the results of deployment" on page 3243
After you have made a deployment, check that the operation has completed successfully.

"Resolving problems when you use HTTP and SOAP nodes" on page 3407
Use the advice given here to help you to resolve common problems that can arise when you develop Web Services message flows that contain HTTP and SOAP nodes.

Related reference:

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsichangetrace** command" on page 3822
Use the **mqsichangetrace** command to set the tracing characteristics for a broker.

"HTTPInput node" on page 4474
Use the HTTPInput node to receive an HTTP message from an HTTP client for processing by a message flow.

"HTTPReply node" on page 4484
Use the HTTPReply node to return a response from the message flow to an HTTP client. This node generates the response to an HTTP client from which the input message was received by the HTTPInput node, and waits for confirmation that it has been sent.

"HTTPRequest node" on page 4488
Use the HTTPRequest node to interact with a web service.

HTTP listeners:

You can choose between broker-wide listeners and execution group (embedded) listeners to manage HTTP messages in your HTTP flows. Learn about the two types of listener, how ports are assigned to them, and how you can switch from one to the other for individual execution groups.

Your choice of listener affects message flows that handle inbound Web service requests by using HTTPInput or HTTPReply nodes. Message flows that do not handle inbound requests but that instigate outbound requests by using HTTPRequest nodes are not affected.

- "Execution group (embedded) listeners"
- "Broker-wide listeners"
- "Using both broker-wide and embedded listeners" on page 1591
- "Configuring listeners" on page 1591

Execution group (embedded) listeners

Each execution group has an embedded listener. The listener is associated with an HTTPConnector object and an HTTPSConnector object. The HTTPConnector object controls the runtime properties that affect the handling of HTTP messages. For example, run the following command to change the port on which the embedded listener for execution group *default* on broker *myBroker* listens for HTTP messages:

```
mqsichangeproperties myBroker -e default -o HTTPConnector
  -n explicitlySetPortNumber -v 8085
```

The HTTPSConnector object controls the runtime properties that affect the handling of HTTPS (HTTP Secure) messages. Run the following command to display these properties for execution group *default* on broker *myBroker*:

```
mqsireportproperties myBroker -e default -o HTTPSConnector -r
```

For further details, including more examples, see "Execution group HTTP listener parameters (SOAP and HTTP nodes)" on page 3805.

Each connector has its own assigned port, which is allocated from a range of numbers, as required. The default range for the HTTPConnector is `7800 - 7842`; the default range for the HTTPSConnector is `7843 - 7884`. The first execution group to start an embedded listener is allocated port `7800`, the second is allocated `7801`, and so on.

If you deploy a message flow to multiple execution groups, the port number is incremented by one for each successive deployment. Assume that no embedded listeners have as yet been started for these execution groups. In this case, the message flow that is deployed to the first execution group receives requests on port 7800. The next message flow uses port 7801, and so on, up to the specified limit of 7842. In this scenario, you typically use an intermediary router that listens on one port, then distributes the requests across the range of ports that you are using.

You can change these port number ranges, and you can allocate a specific port to an execution group, by using the **mqsichangeproperties** command.

If you restart an execution group, the embedded listeners continue to use the same ports as before the restart.

Broker-wide listeners

The broker-wide listener is associated with an HTTPConnector object for properties related to handling HTTP messages, and an HTTPSConnector object for properties related to handling HTTPS messages. For example, run the following command to change the port on which the broker-wide listener for *myBroker* listens for HTTP messages:

```
mqsichangeproperties myBroker -b httplistener -o HTTPConnector
  -n port -v 8085
```

Broker-wide listener properties that apply to HTTP and HTTPS messages are controlled by the HTTPListener object. For example, to disable the broker-wide listener for *myBroker*, run the following command:

```
mqsichangeproperties myBroker -b httplistener -o HTTPListener
  -n startListener -v false
```

Each connector has its own assigned port; default values are `7080` for HTTP and `7083` for HTTPS. You can change these port numbers by using the **mqsichangeproperties** command.

You can configure one or more execution groups so that HTTP nodes that you deploy to those execution groups use the embedded listener.

Using both broker-wide and embedded listeners

Because the option to use the embedded listener is at the execution group level, you can change your configuration such that some execution groups continue to use the broker-wide listener, while specific execution groups use the embedded listener.

However, if you disable the broker-wide listener, the execution group listeners are used for all HTTP nodes, even if you have not explicitly enabled support for them. Therefore, if you set all relevant broker and execution group properties to `false`, the execution group listeners handle all HTTP messages.

The HTTPRequest node communicates directly with the HTTP transport, and is therefore unaffected by your choice.

If you change the listener and port that are processing your HTTP or HTTPS messages, you must ensure that you also update your applications to use the updated configuration.

Configuring listeners

When you have decided on the configuration that you want:

- If you want to use the broker-wide listener for HTTP nodes in all execution groups, you do not have to change your configuration.
- If you are currently using the broker listener for one or more execution groups, and want to switch to using embedded listeners, follow the instructions in "Switching from a broker-wide listener to embedded listeners" on page 1593.
- If you are currently using the execution group listener for one or more execution groups, and want to switch to using the broker-wise listener, follow the instructions in "Switching from embedded listeners to a broker-wide listener" on page 1592.

Related concepts:

"Processing HTTP messages" on page 1579
Hypertext Transfer Protocol (HTTP) is an Internet protocol that is used to transfer and display hypertext and XML documents on the Web.

Related tasks:

"Processing Web service messages" on page 1601
Use WebSphere Message Broker nodes and services to connect to other Web

services providers and consumers.

"Implementing SSL authentication" on page 504
Use SSL authentication to enhance security in your broker environment.

"Switching from a broker-wide listener to embedded listeners" on page 1593
Configure your broker and execution groups so that some or all of the HTTP nodes use an execution group (embedded) listener.

"Switching from embedded listeners to a broker-wide listener"
Configure your broker and execution groups to use the broker-wide listener for HTTP nodes in one or more execution groups.

Related reference:

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"Broker-wide HTTP listener parameters" on page 3809
Select the resources and properties associated with the broker-wide HTTP listener that you want to change.

"Execution group HTTP listener parameters (SOAP and HTTP nodes)" on page 3805
Select the resources and properties associated with the HTTPInput, HTTPReply, SOAPInput, SOAPReply, and SOAPAsyncResponse nodes that you want to change.

"HTTPInput node" on page 4474
Use the HTTPInput node to receive an HTTP message from an HTTP client for processing by a message flow.

"HTTPReply node" on page 4484
Use the HTTPReply node to return a response from the message flow to an HTTP client. This node generates the response to an HTTP client from which the input message was received by the HTTPInput node, and waits for confirmation that it has been sent.

Switching from embedded listeners to a broker-wide listener:

Configure your broker and execution groups to use the broker-wide listener for HTTP nodes in one or more execution groups.

About this task

You can change the configuration for execution groups so that HTTP nodes use the broker-wide listener. HTTP nodes use the broker-wide listener by default, but you might have configured some execution groups to use the embedded listener, and now want to switch back to using the broker-wide listener.

The commands shown in the examples here are split across multiple lines for ease of reading; when you enter the command, you must use a single line.

Procedure

1. Check that the broker is running.
2. If you disabled the broker-wide listener, run the **mqsichangeproperties** command to restart it. For example:

```
mqsichangeproperties MB7BROKER -b httplistener -o HTTPListener
 -n startListener -v true
```

3. To switch to using the broker-wide listener for a specific execution group, use the **mqsichangeproperties** command to change the execution group configuration.
   - For example:

     ```
     mqsichangeproperties MB7BROKER -e exgroup1 -o ExecutionGroup
      -n httpNodesUseEmbeddedListener -v false
     ```

     MB7BROKER is the name of your broker; *exgroup1* is the name of your execution group.

     If you want to change the status for all execution groups, you can omit the specific execution group name:

     ```
     mqsichangeproperties MB7BROKER -o ExecutionGroup
      -n httpNodesUseEmbeddedListener -v false
     ```

     This command does not change the status of the listener for SOAP messages processed by SOAP nodes; only messages to and from HTTP nodes are affected.
4. Stop and restart the broker to ensure that your changes take effect.

Example

For more information about this command, and examples of changing other properties associated with a broker or execution group, see the description of the **mqsichangeproperties** command.

Related concepts:

"HTTP listeners" on page 1589
You can choose between broker-wide listeners and execution group (embedded) listeners to manage HTTP messages in your HTTP flows. Learn about the two types of listener, how ports are assigned to them, and how you can switch from one to the other for individual execution groups.

Related tasks:

"Starting and stopping a broker" on page 921
Run the appropriate command to start or stop a broker.

Related reference:

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"Broker-wide HTTP listener parameters" on page 3809
Select the resources and properties associated with the broker-wide HTTP listener that you want to change.

Switching from a broker-wide listener to embedded listeners:

Configure your broker and execution groups so that some or all of the HTTP nodes use an execution group (embedded) listener.

About this task

You can change the configuration for one or more execution groups so that HTTP nodes deployed in these execution groups use the embedded listener.

Procedure

1. Check that the broker is running.
2. If you want all HTTP nodes in all execution groups to use the embedded listener, you can change the broker configuration to disable the broker-wide listener. Run the **mqsichangeproperties** command to change the broker configuration. Do not run this command if you want to keep the broker-wide listener active for at least one of your execution groups.

   ```
   mqsichangeproperties MB7BROKER -b httplistener -o HTTPListener
    -n startListener -v false
   ```

   All execution groups detect this change of status, and use the embedded listener when they are restarted, regardless of their own specific configuration. Therefore, you can switch to using embedded listeners for all execution groups by running this single command.

   If you disable the broker-wide listener in this way, you can configure an execution group to use the same port or ports that the broker-wide listener was using for HTTP, HTTPS, or both. Reusing the port numbers means that you do not have to change your client applications to send messages to a different port number.
3. To switch to using the embedded listener for a specific execution group, use the **mqsichangeproperties** command to change the execution group configuration.
   - For example:

     ```
     mqsichangeproperties MB7BROKER -e exgroup1 -o ExecutionGroup
      -n httpNodesUseEmbeddedListener -v true
     ```

     MB7BROKER is the name of your broker; *exgroup1* is the name of your execution group.
4. Stop and restart the broker to implement your changes.

Example

For more information about this command, and examples of changing other properties associated with brokers or execution groups, see the description of the **mqsichangeproperties** command.

Related concepts:

"HTTP listeners" on page 1589
You can choose between broker-wide listeners and execution group (embedded) listeners to manage HTTP messages in your HTTP flows. Learn about the two types of listener, how ports are assigned to them, and how you can switch from one to the other for individual execution groups.

Related tasks:

"Starting and stopping a broker" on page 921
Run the appropriate command to start or stop a broker.

Related reference:

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"Broker-wide HTTP listener parameters" on page 3809
Select the resources and properties associated with the broker-wide HTTP listener

that you want to change.

Using timeouts with HTTP and SOAP nodes:

Connect the HTTP Timeout terminal of the HTTPInput or SOAPInput nodes to further nodes to process timeouts.

Before you begin

Before you start:

- Read the message flows overview.
- Read about the options that you have for processing Web service messages, and learn more about SOAP and HTTP.

About this task

You can configure message flows that start with an HTTPInput or SOAPInput node by connecting the HTTP Timeout terminal to further nodes for processing timeouts:

- On SOAPInput nodes, messages are propagated through this terminal only when you are using an HTTP binding.
- On HTTPInput nodes, messages are propagated through this terminal only when you have configured your execution groups such that the HTTP nodes are using the embedded execution group listener.

If these conditions are not met when you deploy the BAR file for the message flow that includes one of these nodes, a warning is generated, and the path of the message flow that you have connected to the HTTP Timeout terminal is ignored. No further warnings are generated until the next restart.

To set a static timeout value in an input node:

1. Create a message flow, or open an existing flow.
2. In the Message Flow editor, select the input node for this message flow. The node properties are displayed in the Properties view (below the editor pane).
3. Set an appropriate time for the timeout interval in the property `Maximum client wait time`. The default interval is 180 seconds.

   If this time expires, and you have not connected one or more nodes to the HTTP Timeout terminal, the listener that received the client request message responds with a SOAP Fault message indicating that a timeout has occurred.
4. If you want to provide customized timeout processing, connect one or more nodes to the HTTP Timeout terminal. You must include in this sequence the reply node that matches the input node. Therefore, if your message flow starts with an HTTPInput node, you must include an HTTPReply; if your message flow starts with a SOAPInput node, you must include a SOAPReply node.

To set a dynamic timeout value in an input node:

- Override the timeout value set on the input node by using the Java plug-in API to update it in a JavaCompute node using the `MbUtilities.changeIdentifierTimeout()` method. The following code shows an example of the `changeIdentifierTimeout` method:

```
MbMessage localEnv = assembly.getLocalEnvironment();
MbElement rootElem = localEnv.getRootElement();
MbElement repIdElement = rootElem.getFirstElementByPath(
```

```
   "/Destination/SOAP/Reply/ReplyIdentifier");
Object repId = repIdElement.getValue();
boolean success = changeIdentifierTimeout((byte[])repId, timeout);
```

- Override the timeout value set on the input node by using the "CHANGEIDENTIFIERTIMEOUT function" on page 5292.

You can derive the value that you use to replace the existing value by several means; for example:

- Create a configurable service of type UserDefined to define a timeout value, and retrieve the appropriate property.
- Read a record from a database.
- Use a value from a field within the message body.

By propagating from the HTTP Timeout terminal you can then change the contents of the responses that your message flow sends to the client. The processing on the sequence of nodes that you connect to the HTTP Timeout terminal is also subject to a further timeout, so that the client always gets a response within a known timeout interval.

When a message is propagated from the HTTP Timeout terminal the message tree contains the input headers of the original input message and a message body that is the fault timeout message. The original message body along with other information relating to the timeout can be accessed in the LocalEnvironment message tree. For example, the following record can be found in the LocalEnvironment:

```
(0x01000000:Name):HTTP = (
  (0x01000000:Name):Input = (
    (0x01000000:Name):Timeout = (
      (0x03000000:NameValue):OriginalClientLastWaitTime = 10 (INTEGER)
      (0x03000000:NameValue):OriginalClientWaitTime     = 15 (INTEGER)
      (0x03000000:NameValue):OriginalMessageMadeTheFlow = TRUE (BOOLEAN)
      (0x03000000:NameValue):OriginalRequestIdentifier  =
         X'485454500000000000000000000000c00c000000000000' (BLOB)
      (0x03000000:NameValue):OriginalInboundMessage     = X'3c3e' (BLOB)
    )
  )
)
```

For SOAPInput nodes the SOAPReply node connected on the HTTP Timeout terminal path must send a SOAP fault response message and the reply status code of 500 cannot be altered. For HTTPInput nodes any response message can be sent from the HTTP Timeout terminal and the reply status code can be changed by updating the `LocalEnvironment.Destination.HTTP.ReplyStatusCode` message tree field.

Related concepts:

"What is SOAP?" on page 1604
SOAP is an XML message format used in Web service interactions. SOAP messages are typically sent over HTTP or JMS, but other transport protocols can be used. The use of SOAP in a specific Web service is described by a WSDL definition.

"Processing HTTP messages" on page 1579
Hypertext Transfer Protocol (HTTP) is an Internet protocol that is used to transfer and display hypertext and XML documents on the Web.

"The Administration API for WebSphere Message Broker" on page 54
The Administration API for WebSphere Message Broker is a programming interface that your applications can use to control brokers and their resources through a remote interface.

Related tasks:

"Developing applications that use the Administration API" on page 956
Develop Java applications that use the Administration API (also known as the CMP API) to communicate with, deploy to, and manage brokers and their associated resources.

"Managing brokers from JavaCompute nodes" on page 997
You can use the CMP API to manage brokers and their associated resources from JavaCompute nodes in deployed message flows.

"Working with properties of a configurable service of type UserDefined at run time in a JavaCompute node" on page 987
Use the CMP API in a JavaCompute node to query, set, create, and delete properties dynamically at run time in configurable services that you have defined with type UserDefined.

"Writing ESQL" on page 2413
How you can use ESQL to customize nodes.

"Processing Web service messages" on page 1601
Use WebSphere Message Broker nodes and services to connect to other Web services providers and consumers.

Related reference:

"HTTPInput node" on page 4474
Use the HTTPInput node to receive an HTTP message from an HTTP client for processing by a message flow.

"SOAPInput node" on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

"Administration API" on page 3672
Use the Administration API for WebSphere Message Broker (CMP API) Java classes and methods to develop CMP applications.

Using compression with HTTP and SOAP nodes:

You can configure HTTP and SOAP nodes to use HTTP compression and decompression when sending and receiving messages.

The nodes can be configured to compress request messages, accept and decompress responses, and decompress input messages.

Background information

The HTTP transport supports the sending of compressed data. Specific HTTP header fields are used to indicate that the data is compressed, and which compression technique was used for the compression. When an HTTP client makes an HTTP request, it can specify that it accepts a compressed response, and also which types of compressed data it accepts. The three compression techniques supported by the HTTP transport are GZIP, deflate, and compress.

The Accept-Encoding field is used in an HTTP request to indicate which encodings are accepted in a response message to the request. This field is used to specify the values of `gzip`, `deflate`, `compress` and `identity`, which are not case-sensitive. A value of `identity` indicates that the data must not be compressed. A wildcard of `*` indicates that any encoding is accepted. The Accept-Encoding field can also be empty, indicating that no encoding is accepted.

The Content-Encoding header field indicates the encoding that is applied to the data and it can contain the tokens of `gzip`, `deflate`, and `compress`, but not `identity`. When data is compressed for an HTTP message, the Content-Encoding field contains the name of the compression technique that was used, enabling the recipient to identify the compression scheme and correctly decompress the message data.

The TE header field is used in a request header to indicate which extension transfer encodings it will accept in the response. This is similar to the Accept-Encoding field.

The Transfer-Encoding field is similar to the Content-Encoding field, except that it is a property of the message and not of the entity. The Transfer-Encoding field is primarily used to indicate that a response message is chunked. It is possible to receive a Transfer-Encoding header that indicates other transfer encodings such as "chunked, gzip". In this case the gzip applies to the transfer of the data, not the actual data itself.

HTTP compression in WebSphere Message Broker

WebSphere Message Broker supports compression and decompression with the HTTP and SOAP nodes using the Accept-Encoding and Content-Encoding fields as follows:

- In a request node, such as an HTTPRequest or SOAPRequest node, if you select the property to allow compressed responses, the broker sets the Accept-Encoding field on the outbound request.
- When compressing a request, the request nodes set the Content-Encoding header field to indicate that the data is compressed. When receiving a response, the request nodes check the Content-Encoding field to determine if the response is compressed. Multiple values in the header field indicate that the data has been compressed more than once using multiple compression functions.
- When receiving a request, the input nodes check the contents of the Content-Encoding field to determine if the contents of the message are compressed. In response and input nodes, a Content-Encoding value of `x-gzip` is treated as `gzip`.
- WebSphere Message Broker supports only the `gzip` and `deflate` compression encodings; the `compress` encoding is not supported.
- Compression and decompression is handled only through the Accept-Encoding and Content-Encoding header fields; the TE and Transfer-Encoding header fields are not supported.

When compressing a request HTTP message, the node checks the Content-Encoding field to determine if the message data is already compressed. If the data is already compressed in the specified scheme then no further compression is needed. However, if the existing data is already compressed in an encoding that is not specified in the node properties, the node further compresses the compressed bit stream using the encoding specified in the node properties. The value of the Content-Encoding field is updated to indicate the additional encoding applied to the data. For example, a Content-Encoding value of `deflate,gzip` indicates that the message should be first decompressed using deflate and then further decompressed using gzip.

The HTTP and SOAP nodes do not support quality values in the Accept-Encoding field, which allow a user to specify a preferred weighting of compression types for responses. Any quality values in the Accept-Encoding field are ignored.

Using HTTP compression with the HTTP and SOAP nodes

The request nodes can request and process compressed responses. You can configure the request nodes to indicate that compression in responses is allowed, and the node automatically decompresses a compressed response. The Accepts-Encoding header is set to indicate that GZIP and deflate compression techniques are accepted. If the Accept-Encoding header is already set, the node does not override it.

If the request or the AsyncResponse nodes receive a GZIP or deflate compressed response, it is decompressed and any header indication that the message is compressed is removed. If the nodes receive an invalid compressed response or an unrecognized compression function, an exception is raised indicating that the data could not be decompressed.

The request nodes can also send compressed requests. You can configure the node to specify which compression technique is used for the compressed requests that it sends. The value of the Content-Encoding header is set to indicate the compression that is used. You can override this value in the local environment for an individual message. If you override the local environment with a value that is not recognized by the node, the existing node value for Use compression is used.

The input nodes can decompress input data that is compressed using the GZIP and deflate compression scheme. If the input node receives a message that is not compressed validly, a fault message is returned to the client and the input message is not propagated to the message flow. This can occur if:

- The Content-Encoding header is set to an unrecognized compression function
- The message body is not compressed correctly using the named Content-Encoding value.

Related reference:

"HTTPInput node" on page 4474
Use the HTTPInput node to receive an HTTP message from an HTTP client for processing by a message flow.

"HTTPRequest node" on page 4488
Use the HTTPRequest node to interact with a web service.

"SOAPAsyncRequest node" on page 4750
Use the SOAPAsyncRequest node with the SOAPAsyncResponse node to construct a pair of message flows that call a Web service asynchronously.

"SOAPRequest node" on page 4828
Use the SOAPRequest node to send a SOAP request to the remote Web service.

"SOAPInput node" on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

Web services example messages

Examples of complete HTTP messages show typical content in specific scenarios.

The following examples show complete HTTP messages. The first message is a request sent by an HTTPRequest node to a Web service that provides a lookup service:

```
POST /greenpages/servlet/rpcrouter HTTP/1.0
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: 520
SOAP Action: ""
Cookie: JSESSIONID=0000B5OSLFIUDMQZFAUXKHD5ZDQ:-1

<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schema.xmlsoap.org.soap/envelope/"
                   xmlns:xsi="http://www.w3/2001/XMLSchema-instance"
                   xmlns:xsd="http://www.w3/2001/XMLSchema">
<SOAP-ENV:Body>
<ns1:getUserByName xmlns:ns1="http://tempuri.org/imb.GreenPages"
                   SOAP-ENV:encodingStyle="http:/schemas.xmlsoap.org/soap/encoding/">
<nameField xsl:type="xsd:string">bloggs, joe</nameField>
</ns1:getUserByName>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The `Cookie` is an example of a value that can be retrieved from the HTTPRequest tree.

The second message is the corresponding Web service response returned to the HTTPRequest node:

```
HTTP/1.0 200 OK
Server: WebSphere Application Server/4.0
Content-Type: text/xml; charset=utf-8
Content-Length: 1585
Content-Language: en
Connection: close

<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schema.xmlsoap.org.soap/envelope/"
                   xmlns:xsi="http://www.w3/2001/XMLSchema-instance"
                   xmlns:xsd="http://www.w3/2001/XMLSchema">
<SOAP-ENV:Body>
<ns1:getUserByNameResponse xmlns:ns1="http://tempuri.org/imb.GreenPages"
                   SOAP-ENV:encodingStyle="http:/schemas.xmlsoap.org/soap/encoding/">
<return xmlns:ns2="http://www.greenpages.com/schemas/GreenPagesRemoteInterface"
        xsi:type="ns2:imb.UserRecord">
<fullName xsi:type="xsd:string">Joseph Bloggs</fullName>
<empNum xsi:type="xsd:int">65874</empNum>
<deskPhone xsi:type="xsd:string">(718)545-3623</deskPhone>
</return>
</ns1:getUserByNameResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

For more information about HTTP return codes, see HTTP Response codes.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Generate WSDL" on page 1274
A *Web Services Description Language* (WSDL) document specifies the interface to a web service, and enables a web service client to start it. A WSDL document that is generated from a message model defines web service requests and responses in terms of the messages that you have defined in that message model.

"HTTP message format" on page 1582
An HTTP message contains components that are appropriate to its type.

"HTTP headers" on page 1583
When an HTTPInput or HTTPRequest node receives a message, it parses the HTTP headers to create elements in the message tree. When an HTTPReply or HTTPRequest node sends a message, it parses the HTTP headers from the message tree into a bit stream.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Generating a WSDL definition from a message set" on page 2968
To ensure the highest interoperability of your Web services, use the document style of WSDL whenever possible. If rpc style WSDL is necessary, use literal encoding.

Chapter 11, "Packaging and deploying," on page 3209
Package resources that you create in the WebSphere Message Broker Toolkit, such as message flows, and deploy them to execution groups on brokers.

Processing Web service messages

Use WebSphere Message Broker nodes and services to connect to other Web services providers and consumers.

About this task

A Web service is a software system designed to support interoperable computer-to-computer interaction over a network. It has an interface described by an XML-based specification; specifically, the Web Service Definition Language, or WSDL.

Web services fulfill a specific task or a set of tasks. A Web service is described using a standard, formal XML notation, called its service description, that provides all the details necessary to interact with the service, including message formats (that detail the operations), transport protocols, and location.

The nature of the interface hides the implementation details of the service, so that it can be used independently of the hardware or software platform on which it is implemented. The interface is also independent of the programming language in which it is written. This interface handles Web service-based applications as loosely coupled, component-oriented, cross-technology implementations. Web services can be used alone, or with other Web services, to carry out a complex aggregation or a business transaction.

The following topics describe how to work with Web services:

- "WebSphere Message Broker and Web services" on page 1602
- "What is SOAP?" on page 1604
- "What is WSDL?" on page 1615
- "What is SOAP MTOM?" on page 1616
- "WS-Addressing" on page 1617
- "WebSphere Service Registry and Repository" on page 1875
- "Message flows for Web services" on page 1619
- "WS-Security" on page 765

WebSphere Message Broker supplies a Java servlet that you can use in an external Web servlet container such as IBM WebSphere Application Server or Apache Tomcat, to receive HTTP requests from Web services client applications. The HTTP

proxy servlet is described in “HTTP proxy servlet overview” on page 856.

Related concepts:

“Message flows overview” on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

“Designing a message flow” on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

“Defining message flow content” on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

“Connecting client applications” on page 1537
Connect your client applications to the broker by using one or more of the supported protocols from other resources and software servers in your network.

“Routing messages” on page 2209
Route messages through your message flow or the broker by using one or more of the techniques described in this section.

“Transforming and enriching messages” on page 2227
Transform and enrich messages by using one or more of the techniques described in this section.

WebSphere Message Broker and Web services

A WebSphere Message Broker application can participate in a Web services environment as a service requester, as a service provider, or both.

The SOAP domain supports these formats:

- Common Web services message formats SOAP 1.1, SOAP 1.2, SOAP with Attachments (SwA), and MTOM.
- Consistent SOAP logical tree format, which is independent of the exact message format.
- WS-Addressing and WS-Security standards.

The following nodes are provided for use in the SOAP domain:

- “SOAPInput node” on page 4795
- “SOAPReply node” on page 4819
- “SOAPRequest node” on page 4828
- “SOAPAsyncRequest node” on page 4750
- “SOAPAsyncResponse node” on page 4777
- “SOAPEnvelope node” on page 4786
- “SOAPExtract node” on page 4790

Use the SOAP nodes and SOAP domain where possible; see “Web services: when to use SOAP or HTTP nodes” on page 1582.

Web services support conforms to the following open standards:

- SOAP 1.1 and 1.2
- SOAP Messages with Attachments
- MTOM
- HTTP 1.1

- WSDL 1.1
- WS-Addressing (new SOAP domain only)
- WS-Security (new SOAP domain only)

WSDL is also validated against the WS-I Basic Profile Version 1.1. Conformance to the guidelines in this specification improves interoperability with other applications.

For more information about how a WebSphere Message Broker application can participate in a Web services environment, see the WebSphere Message Broker web page on developerWorks.

Related concepts:

"What is a Web service?"
A Web service is defined by the World Wide Web Consortium (W3C) as a software system designed to support interoperable machine-to-machine interaction over a network.

"WebSphere Message Broker compliance with Web services standards" on page 6704
WebSphere Message Broker complies with the supported Web services standards and specifications, in that you can generate and deploy Web services that are compliant.

"WS-I Basic Profile Version 1.1" on page 6700
WS-I Basic Profile Version 1.1 (WS-I BP 1.1) is a set of non-proprietary Web services specifications, along with clarifications and amendments to those specifications, which together promote interoperability between different implementations of Web services.

What is a Web service?:

A Web service is defined by the World Wide Web Consortium (W3C) as a software system designed to support interoperable machine-to-machine interaction over a network.

A Web service fulfills a specific task or a set of tasks, and is described by a service description in a standard XML notation called Web Services Description Language (WSDL). The service description provides all of the details necessary to interact with the service, including message formats (that detail the operations), transport protocols, and location.

Other systems use SOAP messages to interact with the Web service, typically by using HTTP with an XML serialization in conjunction with other Web-related standards.

The WSDL interface hides the details of how the service is implemented, and the service can be used independently of the hardware or software platform on which it is implemented, and independently of the programming language in which it is written.

Applications that are based on Web services are loosely-coupled, component-oriented, cross-technology implementations. Web services can be used alone, or in conjunction with other Web services to carry out a complex aggregation or a business transaction.

Related concepts:

"WebSphere Message Broker and Web services" on page 1602
A WebSphere Message Broker application can participate in a Web services environment as a service requester, as a service provider, or both.

"Web services external standards" on page 6696
WebSphere Message Broker support for Web services conforms to a number of industry standards and specifications.

What is SOAP?:

SOAP is an XML message format used in Web service interactions. SOAP messages are typically sent over HTTP or JMS, but other transport protocols can be used. The use of SOAP in a specific Web service is described by a WSDL definition.

There are two versions of SOAP in common use: SOAP 1.1 and SOAP 1.2. Both are supported in WebSphere Message Broker. SOAP is defined in the following documents issued by World Wide Web Consortium (W3C):

- Simple Object Access Protocol (SOAP) 1.1 (W3C note).
- SOAP Version 1.2 Part 0: Primer (W3C recommendation).
- SOAP Version 1.2 Part 1: Messaging Framework (W3C recommendation).
- SOAP Version 1.2 Part 2: Adjuncts (W3C recommendation).

Support for SOAP in WebSphere Message Broker includes:

- SOAP parser and domain. See "SOAP parser and domain" on page 1082.
- SOAP nodes to send and receive messages in SOAP format.
- IBM supplied message definitions for SOAP 1.1 and SOAP 1.2. These message definitions support validation, ESQL content assist, and the creation of message maps for use with SOAP messages, in the SOAP and other XML domains. See "IBM supplied messages that you can import" on page 6367.
- HTTP and JMS transport to send SOAP messages. See the W3C SOAP over JMS specification: http://www.w3.org/TR/soapjms/

WSDL validation in WebSphere Message Broker refers to the WS-I Basic Profile. For more information, see the WS-I, and in particular the WS-I Basic Profile document:

- http://www.ws-i.org/
- http://www.ws-i.org/deliverables

Related concepts:

"WebSphere Message Broker and Web services" on page 1602
A WebSphere Message Broker application can participate in a Web services environment as a service requester, as a service provider, or both.

"SOAP 1.1 and 1.2" on page 6696
SOAP is a lightweight, XML-based, protocol for exchange of information in a decentralized, distributed environment.

"The structure of a SOAP message" on page 1605
A SOAP message is encoded as an XML document, consisting of an `<Envelope>` element, which contains an optional `<Header>` element, and a mandatory `<Body>` element. The `<Fault>` element, contained in `<Body>`, is used for reporting errors.

"SOAP tree overview" on page 1611
This tree format allows you to access the key parts of the SOAP message in a convenient way.

Related reference:

"IBM supplied messages that you can import" on page 6367
You can import IBM supplied messages to create a new message definition file.

The structure of a SOAP message:

A SOAP message is encoded as an XML document, consisting of an <Envelope> element, which contains an optional <Header> element, and a mandatory <Body> element. The <Fault> element, contained in <Body>, is used for reporting errors.

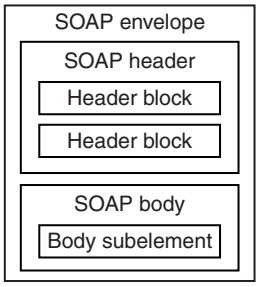
The SOAP envelope
<Envelope> is the root element in every SOAP message, and contains two child elements, an optional <Header> element, and a mandatory <Body> element.

The SOAP header
<Header> is an optional subelement of the SOAP envelope, and is used to pass application-related information that is to be processed by SOAP nodes along the message path; see "The SOAP header" on page 1606.

The SOAP body
<Body> is a mandatory subelement of the SOAP envelope, which contains information intended for the ultimate recipient of the message; see "The SOAP body" on page 1608.

The SOAP fault
<Fault> is a subelement of the SOAP body, which is used for reporting errors; see "The SOAP fault" on page 1608.

XML elements in <Header> and <Body> are defined by the applications that make use of them, although the SOAP specification imposes some constraints on their structure. The following diagram shows the structure of a SOAP message.



The following code is an example of a SOAP message that contains header blocks (the <m:reservation> and <n:passenger> elements) and a body (containing the <p:itinterary> element).

```
<?xml version='1.0' Encoding='UTF-8' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
  <m:reservation xmlns:m="http://travelcompany.example.org/reservation"
  env:role="http://www.w3.org/2003/05/soap-envelope/role/next">
   <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</m:reference>
   <m:dateAndTime>2007-11-29T13:20:00.000-05:00</m:dateAndTime>
  </m:reservation>
  <n:passenger xmlns:n="http://mycompany.example.com/employees"
  env:role="http://www.w3.org/2003/05/soap-envelope/role/next">
   <n:name>Fred Bloggs</n:name>
  </n:passenger>
 </env:Header>
 <env:Body>
  <p:itinerary xmlns:p="http://travelcompany.example.org/reservation/travel">
```

```
   <p:departure>
     <p:departing>New York</p:departing>
     <p:arriving>Los Angeles</p:arriving>
     <p:departureDate>2007-12-14</p:departureDate>
     <p:departureTime>late afternoon</p:departureTime>
     <p:seatPreference>aisle</p:seatPreference>
   </p:departure>
   <p:return>
     <p:departing>Los Angeles</p:departing>
     <p:arriving>New York</p:arriving>
     <p:departureDate>2007-12-20</p:departureDate>
     <p:departureTime>mid-morning</p:departureTime>
     <p:seatPreference></p:seatPreference>
   </p:return>
  </p:itinerary>
 </env:Body>
</env:Envelope>
```

Related concepts:

"The SOAP header"
The SOAP header (the <Header> element) is an optional sub-element of the SOAP envelope, and is used to pass application-related information that is processed by SOAP nodes along the message flow.

"The SOAP body" on page 1608
The SOAP body (the <Body> element) is a mandatory sub-element of the SOAP envelope, which contains information intended for the ultimate recipient of the message.

"The SOAP fault" on page 1608
The SOAP fault (the <Fault> element) is a sub-element of the SOAP body, which is used for reporting errors.

"SOAP 1.1 and 1.2" on page 6696
SOAP is a lightweight, XML-based, protocol for exchange of information in a decentralized, distributed environment.

The SOAP header:

The SOAP header (the <Header> element) is an optional sub-element of the SOAP envelope, and is used to pass application-related information that is processed by SOAP nodes along the message flow.

The immediate child elements of the header are called *header blocks*. A header block is an application-defined XML element, and represents a logical grouping of data which can be targeted at SOAP nodes that might be encountered in the path of a message from a sender to an ultimate receiver.

SOAP header blocks can be processed by SOAP intermediary nodes, and by the ultimate SOAP receiver node. However, in a real application, not every node processes every header block. Each node is typically designed to process particular header blocks, and each header block is processed by particular nodes.

The SOAP header enables you to add features to a SOAP message in a decentralized manner without prior agreement between the communicating parties. SOAP defines some attributes that can be used to indicate what can deal with a feature and whether it is optional or mandatory. Such control information includes, for example, passing directives or contextual information related to the processing of the message. This control information enables a SOAP message to be extended in an application-specific manner.

Although the header blocks are application-defined, SOAP-defined attributes on the header blocks indicate how the header blocks must be processed by the SOAP nodes. SOAP-defined attributes include:

encodingStyle
Indicates the rules used to encode the parts of a SOAP message. SOAP defines a narrower set of rules for encoding data than the flexible encoding that XML enables.

actor (SOAP 1.1) or role (SOAP 1.2)
In SOAP 1.2, the role attribute specifies whether a particular node will operate on a message. If the role specified for the node matches the role attribute of the header block, the node processes the header. If the roles do not match, the node does not process the header block. In SOAP 1.1, the actor attribute performs the same function.

Roles can be defined by the application, and are designated by a URI. For example, `http://example.com/Log` might designate the role of a node which performs logging. Header blocks that are processed by this node specify `env:role="http://example.com/Log"` (where the namespace prefix `env` is associated with the SOAP namespace name of `http://www.w3.org/2003/05/soap-envelope`).

The SOAP 1.2 specification defines three standard roles in addition to those which are defined by the application:

`http://www.w3.org/2003/05/soap-envelope/none`
None of the SOAP nodes on the message path should process the header block directly. Header blocks with this role can be used to carry data that is required for processing of other SOAP header blocks.

`http://www.w3.org/2003/05/soap-envelope/next`
All SOAP nodes on the message path are expected to examine the header block (provided that the header has not been removed by a node earlier in the message path).

`http://www.w3.org/2003/05/soap-envelope/ultimateReceiver`
Only the ultimate receiver node is expected to examine the header block.

mustUnderstand
This attribute is used to ensure that SOAP nodes do not ignore header blocks which are important to the overall purpose of the application. If a SOAP node determines, by using the **role** or **actor** attribute, that it should process a header block, the action taken depends on the value of the **mustUnderstand** attribute.

- `1` (SOAP 1.1) or `true` (SOAP 1.2): The node must either process the header block in a manner consistent with its specification, or not at all (and throw a fault).
- `0` (SOAP 1.1) or `false` (SOAP 1.2): The node is not obliged to process the header block.

In effect, the **mustUnderstand** attribute indicates whether processing of the header block is mandatory or optional.

relay (SOAP 1.2 only)
When a SOAP intermediary node processes a header block, the SOAP intermediary node removes the header block from the SOAP message. By default, the SOAP intermediary node also removes all header blocks that it

ignored (because the **mustUnderstand** attribute had a value of `false`). However, when the relay attribute is specified with a value of `true`, the SOAP intermediary node retains the unprocessed header block in the message.

Related concepts:

"The SOAP body"
The SOAP body (the <Body> element) is a mandatory sub-element of the SOAP envelope, which contains information intended for the ultimate recipient of the message.

"The SOAP fault"
The SOAP fault (the <Fault> element) is a sub-element of the SOAP body, which is used for reporting errors.

The SOAP body:

The SOAP body (the <Body> element) is a mandatory sub-element of the SOAP envelope, which contains information intended for the ultimate recipient of the message.

The body element and its associated child elements are used to exchange information between the initial SOAP sender and the ultimate SOAP receiver. SOAP defines one child element for the body: the `<Fault>` element, which is used for reporting errors. Other elements in the body are defined by the Web service that uses them.

Related concepts:

"The SOAP header" on page 1606
The SOAP header (the <Header> element) is an optional sub-element of the SOAP envelope, and is used to pass application-related information that is processed by SOAP nodes along the message flow.

"The SOAP fault"
The SOAP fault (the <Fault> element) is a sub-element of the SOAP body, which is used for reporting errors.

The SOAP fault:

The SOAP fault (the <Fault> element) is a sub-element of the SOAP body, which is used for reporting errors.

If present, the SOAP fault element must appear as a body entry and must not appear more than once in a body element. The sub-elements of the SOAP fault element are different in SOAP 1.1 and SOAP 1.2.

SOAP 1.1

In SOAP 1.1, the SOAP fault contains the following sub-elements:

`<faultcode>`
The`<faultcode>` element is a mandatory element in the `<Fault>` element. It provides information about the fault in a form that can be processed by software. SOAP defines a small set of SOAP fault codes covering basic SOAP faults, and this set can be extended by applications.

`<faultstring>`
The `<faultstring>` element is a mandatory element in the `<Fault>` element. It provides information about the fault in a form intended for a human reader.

<faultactor>
The <faultactor> element contains the URI of the SOAP node that generated the fault. A SOAP node that is not the ultimate SOAP receiver must include the <faultactor> element when it creates a fault; an ultimate SOAP receiver is not obliged to include this element, but might do so.

<detail>
The <detail> element carries application-specific error information related to the <Body> element. It must be present if the contents of the <Body> element were not successfully processed. The <detail> element must not be used to carry information about error information belonging to header entries. Detailed error information belonging to header entries must be carried in header entries.

SOAP 1.2

In SOAP 1.2, the SOAP fault contains the following sub-elements:

<Code> The <Code> element is a mandatory element in the <Fault> element. It provides information about the fault in a form that can be processed by software. It contains a <Value> element and an optional <Subcode> element.

<Reason>
The <Reason> element is a mandatory element in the <Fault> element. It provides information about the fault in a form intended for a human reader. The <Reason> element contains one or more <Text> elements, each of which contains information about the fault in a different language.

<Node> The <Node> element contains the URI of the SOAP node that generated the fault. A SOAP node that is not the ultimate SOAP receiver must include the <Node> element when it creates a fault; an ultimate SOAP receiver is not obliged to include this element, but might do so.

<Role> The <Role> element contains a URI that identifies the role in which the node was operating at the point the fault occurred.

<Detail>
The <Detail> element is an optional element, which contains application-specific error information related to the SOAP fault codes describing the fault. The presence of the <Detail> element has no significance as to which parts of the faulty SOAP message were processed.

Related concepts:

"The SOAP header" on page 1606
The SOAP header (the <Header> element) is an optional sub-element of the SOAP envelope, and is used to pass application-related information that is processed by SOAP nodes along the message flow.

"The SOAP body" on page 1608
The SOAP body (the <Body> element) is a mandatory sub-element of the SOAP envelope, which contains information intended for the ultimate recipient of the message.

SOAP nodes:

The SOAP nodes act as points in the flow where Web service processing is configured and applied. Properties on the SOAP nodes control the processing carried out and can be configured by supplying a WSDL definition, or by manually configuring properties, or both.

SOAP nodes

- The SOAPInput and SOAPReply nodes are used in a message flow which implements a Web service. These SOAP nodes are used to construct a message flow that implements a Web service provider. The SOAPInput node listens for incoming Web service requests, and the SOAPReply sends responses back to the client; see “SOAPInput node” on page 4795 and “SOAPReply node” on page 4819.
- A client can send an HTTP GET request to the endpoint exposed by the flow, suffixed with a query string `?wsdl`, and receive a response with the WSDL definition used to configure the flow. For a full description, see “Using WSDL to configure message flows” on page 1664.
- The SOAPRequest node is used in a message flow to call a Web service provider synchronously. Calling a Web service synchronously means that the node sends a Web service request and waits, blocking the message flow, for the associated Web service response to be received before the message flow continues; see “SOAPRequest node” on page 4828.
- The SOAPAsyncRequest and SOAPAsyncResponse nodes are used to construct a message flow (or pair of flows) which calls a Web service asynchronously. Calling a Web service asynchronously means that the SOAPAsyncRequest node sends a Web service request, but the request does not block the message flow by waiting for the associated Web service response to be received because the Web service response is received at the SOAPAsyncResponse node, which is in a separate flow. The Node Correlator identifies the logical pairing of the responses against the original requests. Multiple requests can, therefore, be handled in parallel; see “SOAPAsyncRequest node” on page 4750 and “SOAPAsyncResponse node” on page 4777.
- You can work on the payload of the SOAP body using the SOAPExtract and SOAPEnvelope nodes. The SOAPExtract node can interoperate with the SOAP domain. The SOAP nodes do not require the SOAPEnvelope node, because they can directly handle non-SOAP messages, but the SOAPEnvelope node is still required for the HTTP nodes. See “SOAPExtract node” on page 4790 and “SOAPEnvelope node” on page 4786.
- You can change the Operation mode of the SOAP nodes so that they act in gateway mode. In gateway mode, a WSDL is not required to configure the nodes since they handle generic request/response and one-way SOAP messages that are not tied to a specific WSDL. For more details, see “Gateway operation mode for SOAP nodes” on page 1645.

The W3C SOAP specification refers to "SOAP nodes" meaning a unit of application logic (see Web Services Glossary). Typically, references to "SOAP nodes" in the WebSphere Message Broker Information Center are referring to WebSphere Message Broker SOAP nodes.

If you choose to handle HTTP messages by using the execution group listener, you must carefully check the URL specifications in your HTTPInput and SOAPInput nodes. If both URL specifications match an incoming message, the wrong type of node might get the message, and processing might fail or produce unexpected results. This situation occurs if you specify identical values for the Path suffix for URL properties of the HTTPInput node and the SOAPInput node. It can also occur if you use wildcards in either or both specifications, and an incoming message matches both properties.

Related concepts:

"What is SOAP?" on page 1604
SOAP is an XML message format used in Web service interactions. SOAP messages are typically sent over HTTP or JMS, but other transport protocols can be used. The use of SOAP in a specific Web service is described by a WSDL definition.

"The SOAP body" on page 1608
The SOAP body (the <Body> element) is a mandatory sub-element of the SOAP envelope, which contains information intended for the ultimate recipient of the message.

"Using WSDL to configure message flows" on page 1664
You can use WSDL to configure message flows.

Related reference:

"HTTPReply node" on page 4484
Use the HTTPReply node to return a response from the message flow to an HTTP client. This node generates the response to an HTTP client from which the input message was received by the HTTPInput node, and waits for confirmation that it has been sent.

"HTTPInput node" on page 4474
Use the HTTPInput node to receive an HTTP message from an HTTP client for processing by a message flow.

"HTTPRequest node" on page 4488
Use the HTTPRequest node to interact with a web service.

"SOAPInput node" on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

"SOAPReply node" on page 4819
Use the SOAPReply node to send SOAP messages from the broker to the originating client in response to a message received by a SOAPInput node.

"SOAPRequest node" on page 4828
Use the SOAPRequest node to send a SOAP request to the remote Web service.

"SOAPAsyncRequest node" on page 4750
Use the SOAPAsyncRequest node with the SOAPAsyncResponse node to construct a pair of message flows that call a Web service asynchronously.

"SOAPAsyncResponse node" on page 4777
Use the SOAPAsyncResponse node in conjunction with the SOAPAsyncRequest node to construct a pair of message flows that call a Web service asynchronously.

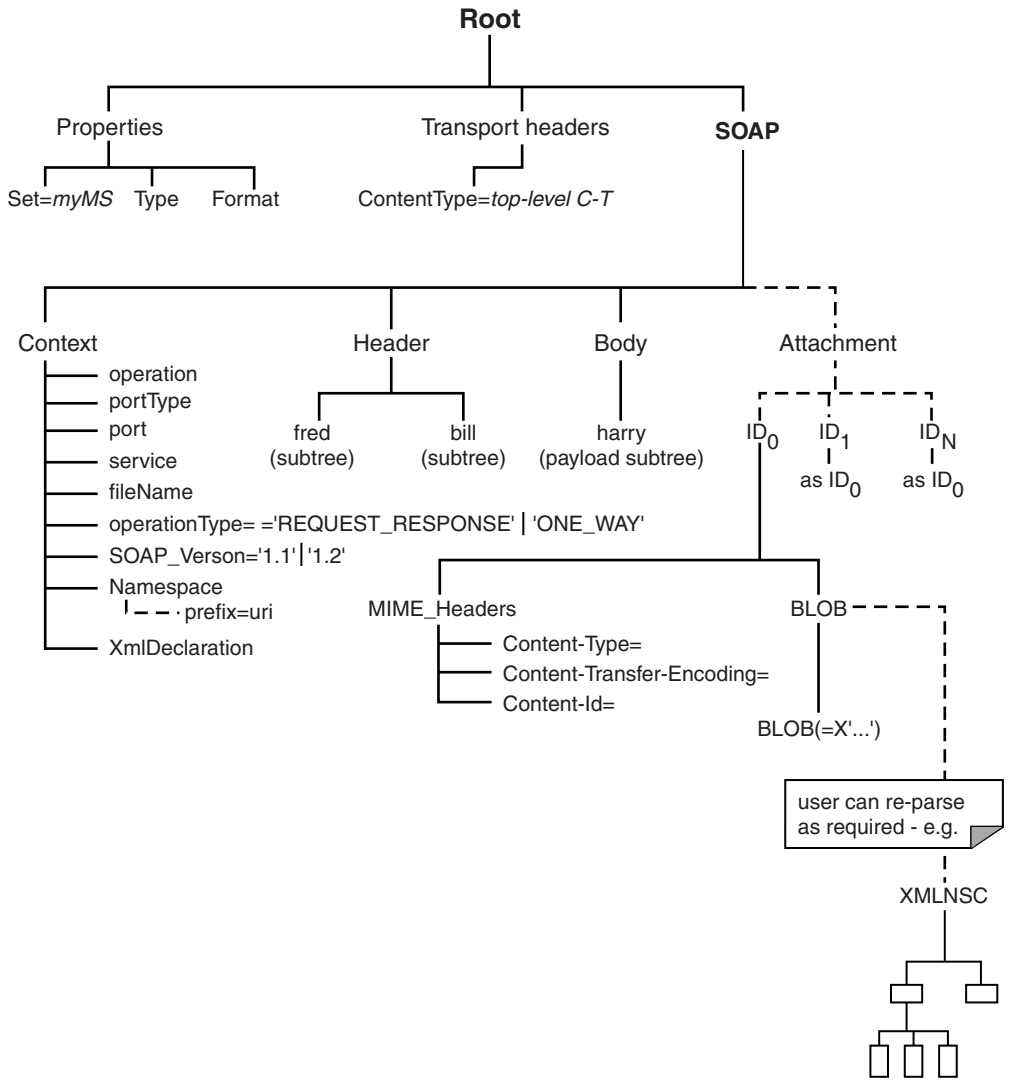
Related information:

"Web services: when to use SOAP or HTTP nodes" on page 1582
HTTP and SOAP nodes can both be used to interact with Web services. Typically you use SOAP nodes when working with SOAP-based Web services.

SOAP tree overview:

This tree format allows you to access the key parts of the SOAP message in a convenient way.

This is a diagrammatic representation of the SOAP domain tree:



The SOAP tree contains the following elements:

SOAP.Header
Contains the SOAP header blocks (children of `Envelope.Header`)

SOAP.Body
Contains the SOAP payload (children of `Envelope.Body`)

The content of the `Body` subtree depends on the WSDL style.

SOAP.Attachment
Contains attachments for an `SwA` message in their non encoded format.

Note that attachments for an `MTOM` message are represented inline as part of the SOAP content in a base 64 representation.

SOAP.Context
Contains the following information:

- Input; populated by the SOAPInput node:
  - `operation` - the WSDL operation name. In Gateway mode, the operation is assumed to be the name of the element that is the first

child of the SOAP Body element, if present, otherwise it is the constant name 'ComIbmBrokerGenericGatewayOperation'.
  - `portType` - the WSDL port type name. In Gateway mode, this item is empty.
  - `port` - the WSDL port name (if known). In Gateway mode, this item is empty.
  - `service` - the WSDL service name (if known). In Gateway mode, the service has the constant name 'ComIbmBrokerGenericGatewayService'.
  - `fileName` - the original WSDL file name. In Gateway mode, this item is empty.
  - `operationType` - one of 'REQUEST_RESPONSE', 'ONE_WAY', 'SOLICIT_RESPONSE', 'NOTIFICATION'. In Gateway mode, without WSDL, this field contains 'GATEWAY'. This means 'REQUEST_RESPONSE' or 'GATEWAY_ONE_WAY', which means that the node has detected the operation type to be one-way.
  - `SOAP_Version` - one of '1.1' or '1.2'.
  - `Namespace` - Contains `nameValue` child elements; the `name` is the Namespace prefix, and the `value` is the Namespace URI as it appears in the bit stream.
  - `XmlDeclaration` - represents the standard XML declaration.
- Output; the following fields can be placed in `SOAP.Context` to provide override information when SOAPRequest or SOAPAsyncRequest nodes serialize a SOAP message:
  - `SOAP_Version` - one of '1.1' or '1.2'
  - `Namespace` - Contains `nameValue` child elements that define the namespace prefix (the `name`) to be used for a specified namespace URI (the `value`).

    An output message uses the namespace prefixes defined here to qualify any elements in the corresponding namespaces.

    If the `SOAP.Context` was originally created at an input node, it might already contain all the namespace prefix definitions that you need.

    If `SOAP.Context` does not exist, or the outgoing message uses additional namespaces, the SOAP parser generates any required namespace prefixes automatically.

    Alternatively, you can specify your own namespace prefix; the specific name of a namespace prefix does not usually affect the meaning of a message, with one important exception. If the message content contains a `qualified name`, the message must contain a matching namespace prefix definition.

    For example, if the output message is a SOAP Fault containing a `<faultcode>` element with the value `soapenv:Server`, a namespace prefix (which is case sensitive) for `soapenv` must be defined in the logical tree:

```
-- Build SOAP Fault message. Note that as well as defining the correct
-- namespace for the Fault element, it is also necessary to bind the
-- namespace prefix used in the faultcode element (this is set up under
-- SOAP.Context.Namespace)

-- Send back a new user defined SOAP 1.2 fault message
DECLARE soapenv NAMESPACE 'http://www.w3.org/2003/05/soap-envelope';
DECLARE xml     NAMESPACE 'http://www.w3.org/XML/1998/namespace';
DECLARE myNS    NAMESPACE 'http://myNS';

SET OutputRoot.SOAP.Context.Namespace.(SOAP.NamespaceDecl)xmlns:soapenv = soapenv;
SET OutputRoot.SOAP.Context.Namespace.(SOAP.NamespaceDecl)xmlns:myNS = myNS;
```

```
SET OutputRoot.SOAP.Body.soapenv:Fault.soapenv:Code.soapenv:Value = 'soapenv:Receiver';
SET OutputRoot.SOAP.Body.soapenv:Fault.soapenv:Code.soapenv:Subcode.soapenv:Value = 'my:subcode value';
SET OutputRoot.SOAP.Body.soapenv:Fault.soapenv:Reason.soapenv:Text = 'my Reason string';
SET OutputRoot.SOAP.Body.soapenv:Fault.soapenv:Reason.soapenv:Text.(SOAP.Attribute)xml:lang = 'en';
SET OutputRoot.SOAP.Body.soapenv:Fault.soapenv:Node = 'my Node string';
SET OutputRoot.SOAP.Body.soapenv:Fault.soapenv:Role = 'my Role string';
SET OutputRoot.SOAP.Body.soapenv:Fault.soapenv:Detail.my:Text = 'my detail string';
```

```
-- Send back a new user defined SOAP 1.1 fault message
DECLARE soapenv NAMESPACE 'http://schemas.xmlsoap.org/soap/envelope/';
SET OutputRoot.SOAP.Context.Namespace.(SOAP.NamespaceDecl)xmlns:soapenv = soapenv;

SET OutputRoot.SOAP.Body.soapenv:Fault.faultcode = 'soapenv:Receiver';
SET OutputRoot.SOAP.Body.soapenv:Fault.faultstring = 'my fault string';
SET OutputRoot.SOAP.Body.soapenv:Fault.faultactor = 'my fault actor';
SET OutputRoot.SOAP.Body.soapenv:Fault.detail.Text = 'my detail string';
```

Only `Namespace`, `SOAP_Version`, and `XmlDeclaration` influence the bit stream generated for a SOAP tree; the other fields are for information only.

Related concepts:

"The structure of a SOAP message" on page 1605
A SOAP message is encoded as an XML document, consisting of an `<Envelope>` element, which contains an optional `<Header>` element, and a mandatory `<Body>` element. The `<Fault>` element, contained in `<Body>`, is used for reporting errors.

"The SOAP body" on page 1608
The SOAP body (the <Body> element) is a mandatory sub-element of the SOAP envelope, which contains information intended for the ultimate recipient of the message.

"The SOAP header" on page 1606
The SOAP header (the <Header> element) is an optional sub-element of the SOAP envelope, and is used to pass application-related information that is processed by SOAP nodes along the message flow.

"What is SOAP?" on page 1604
SOAP is an XML message format used in Web service interactions. SOAP messages are typically sent over HTTP or JMS, but other transport protocols can be used. The use of SOAP in a specific Web service is described by a WSDL definition.

Related reference:

"XMLNSC: Namespace declarations" on page 2554
The XMLNSC parser provides full support for namespaces.

"IBM supplied messages that you can import" on page 6367
You can import IBM supplied messages to create a new message definition file.

Web services: when to use SOAP or HTTP nodes:

HTTP and SOAP nodes can both be used to interact with Web services. Typically you use SOAP nodes when working with SOAP-based Web services.

For SOAP-based Web services, several advantages exist if you use the SOAP nodes and the SOAP message domain instead of the HTTP transport nodes and XMLNSC message domain.

- Support for WS-Addressing, WS-Security and SOAP headers.
- A common SOAP logical tree format, independent of the bitstream format.
- Runtime checking against WSDL.
- Automatic processing of SOAP with Attachments (SwA).

- Automatic processing of Message Transmission Optimization Mechanism (MTOM).

Although the HTTP nodes can process SwA messages, you must use the MIME message domain and design your flow to handle the attachments explicitly, and use custom logic to extract and parse the SOAP.

Cases where it might be better to use HTTP nodes include:

- Message flows in which a single request node handles multiple SOAP request and responses from more than one WSDL.
- Message flows that interact with Web services that use different standards, such as REST or XML-RPC.
- Message flow that never use WS-Addressing, WS-Security, SwA, or MTOM.

Related concepts:

"What is a Web service?" on page 1603
A Web service is defined by the World Wide Web Consortium (W3C) as a software system designed to support interoperable machine-to-machine interaction over a network.

"Processing HTTP messages" on page 1579
Hypertext Transfer Protocol (HTTP) is an Internet protocol that is used to transfer and display hypertext and XML documents on the Web.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

What is WSDL?:

WSDL is an XML notation for describing a web service. A WSDL definition tells a client how to compose a web service request and describes the interface that is provided by the web service provider.

WebSphere Message Broker supports WSDL 1.1, as defined in the following document issued by the World Wide Web Consortium (W3C): Web Services Description Language (WSDL) 1.1. WebSphere Message Broker support for WSDL also adheres to the Web Services Interoperability Organization (WS-I) Basic profile 1.1; see Web Services Interoperability Organization (WS-I).

A WSDL definition is divided into separate sections that specify the logical interface and the physical details of a web service. The physical details include both endpoint information, such as HTTP port number, and binding information, which specifies how the SOAP payload is represented and which transport is used.

Support for WSDL in WebSphere Message Broker includes:

- Import of WSDL to create message definitions in a message set; see "Importing from WSDL" on page 2946.
- Generation of WSDL from a message set; see "WSDL generation" on page 6340.
- WSDL editor with text and graphical design views.
- Use of WSDL to configure nodes in the SOAP domain; for example, you can drag WSDL onto a node, and a client can request the WSDL that was used to configure a SOAPInput node. For more details, see "Using WSDL to configure message flows" on page 1664
- Use WSDL to create a skeleton message flow by dragging WSDL onto the message flow editor canvas. For more details, see "Using WSDL to configure message flows" on page 1664.

When you import or generate WSDL, the WSDL is validated against the WS-I Basic Profile. You must fix validation errors before the message set can be deployed. Validation warnings do not prevent deployment, but can indicate potential interoperability problems. The validated WSDL becomes an integral part of the message set.

The WSDL editor supports a graphical design view so that you can navigate from the WSDL to its associated message definitions. The message set contains all the message definitions required by message flows that are working with the Web service described by the WSDL. At development time, the message definitions support ESQL Content Assist and the creation of mappings. At run time, the deployed message set supports schema validation in the SOAP, XMLNSC, and MRM domains. In the SOAP domain, runtime checks are also made against the WSDL itself, and WSDL information is included in the SOAP logical tree.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"WSDL validation" on page 1661
The WS-I Validator can be used to check your WSDL definitions against the Basic Profile.

"WSDL Version 1.1" on page 6699
Web Services Description Language (WSDL) is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information.

What is SOAP MTOM?:

SOAP Message Transmission Optimization Mechanism (MTOM) is the use of MIME to optimize the bitstream transmission of SOAP messages that contain significantly large base64Binary elements.

The MTOM message format allows bitstream compression of binary data. Data that would otherwise have to be encoded in the SOAP message is instead transmitted as raw binary data in a separate MIME part. A large chunk of binary data takes up less space than its encoded representation, so MTOM can reduce transmission time, although it can increase processor usage. Candidate elements to be transmitted in this way are defined as base64Binary in the WSDL (XML Schema).

An MTOM message is identified by a Content-Type with a type of `application/xop+xml`.

The SOAP domain handles inbound MTOM messages automatically, and MTOM parts are reincorporated automatically into the SOAP Body.

The use of outbound MTOM messages can be configured on the SOAPReply, SOAPRequest, and SOAPAsyncRequest nodes; for details, see "Using SOAP MTOM with the SOAPReply, SOAPRequest, and SOAPAsyncRequest nodes" on page 1678.

For details of the external specification published by the World Wide Web Consortium (W3C), see "SOAP MTOM" on page 6697.

Related concepts:

"What is SOAP?" on page 1604
SOAP is an XML message format used in Web service interactions. SOAP messages are typically sent over HTTP or JMS, but other transport protocols can be used. The use of SOAP in a specific Web service is described by a WSDL definition.

Related tasks:

"Processing Web service messages" on page 1601
Use WebSphere Message Broker nodes and services to connect to other Web services providers and consumers.

Related reference:

"SOAPReply node" on page 4819
Use the SOAPReply node to send SOAP messages from the broker to the originating client in response to a message received by a SOAPInput node.

"SOAPRequest node" on page 4828
Use the SOAPRequest node to send a SOAP request to the remote Web service.

"SOAPAsyncRequest node" on page 4750
Use the SOAPAsyncRequest node with the SOAPAsyncResponse node to construct a pair of message flows that call a Web service asynchronously.

WS-Addressing:

Web Services Addressing (WS-Addressing) is a World Wide Web Consortium (W3C) specification that aids interoperability between Web services by defining a standard way to address Web services and provide addressing information in messages.

Start here to find out how WebSphere Message Broker supports WS-Addressing.

The WS-Addressing specification introduces two primary concepts: endpoint references, and message addressing properties. This topic contains an overview of each concept. For further details, select the following links to access the WS-Addressing specifications:

- W3C WS-Addressing specifications
- W3C submission WS-Addressing specification

Endpoint references (EPRs)

EPRs provide a standard mechanism to encapsulate information about specific endpoints. EPRs can be propagated to other parties and then used to target the Web service endpoint that they represent. The following table summarizes the information model for EPRs.

Abstract Property name	Property type	Multiplicity	Description
[address]	xs:anyURI	1..1	The absolute URI that specifies the address of the endpoint.
[reference parameters]*	xs:any	0..unbounded	Namespace qualified element information items that are required to interact with the endpoint.
[metadata]	xs:any	0..unbounded	Description of the behavior, policies and capabilities of the endpoint.

The following prefix and corresponding namespace is used in the previous table.

Prefix	Namespace
xs	http://www.w3.org/2001/XMLSchema

The following XML fragment illustrates an endpoint reference. This element references the endpoint at the URI http://example.com/fabrikam/acct, has metadata specifying the interface to which the endpoint reference refers, and has application-defined reference parameters of the http://example.com/fabrikam namespace.

```
<wsa:EndpointReference xmlns:wsa="http://www.w3.org/2005/08/addressing"
                       xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
                       xmlns:fabrikam="http://example.com/fabrikam"
                       xmlns:wsdli="http://www.w3.org/2005/08/wsdl-instance"
                       wsdli:wsdlLocation="http://example.com/fabrikam
                       http://example.com/fabrikam/fabrikam.wsdl">
    <wsa:Address>http://example.com/fabrikam/acct</wsa:Address>
    <wsa:Metadata>
        <wsaw:InterfaceName>fabrikam:Inventory</wsaw:InterfaceName>
    </wsa:Metadata>
    <wsa:ReferenceParameters>
        <fabrikam:CustomerKey>123456789</fabrikam:CustomerKey>
        <fabrikam:ShoppingCart>ABCDEFG</fabrikam:ShoppingCart>
    </wsa:ReferenceParameters>
</wsa:EndpointReference>
```

Message addressing properties (MAPs)

MAPs are a set of well defined WS-Addressing properties that can be represented as elements in SOAP headers. MAPs can provide either a standard way of conveying information, such as the endpoint to which message replies should be directed, or information about the relationship that the message has with other messages. The MAPs that are defined by the WS-Addressing specification are summarized in the following table.

Abstract WS-Addressing MAP name	MAP content type	Multiplicity	Description
[action]	xs:anyURI	1..1	An absolute URI that uniquely identifies the semantics of the message. This proprety corresponds to the [address] property of the endpoint reference to which the message is addressed. This value is required.
[destination]	xs:anyURI	1..1	The absolute URI that specifies the address of the intended receiver of this message. This value is optional because, if not present, it defaults to the anonymous URI that is defined in the specification, indicating that the address is defined by the underpinning protocol.
[reference parameters]*	xs:any	0..unbounded	Correspond to the [reference parameters] property of the endpoint reference to which the message is addressed. This value is optional.
[source endpoint]	EndpointReference	0..1	A reference to the endpoint from which the message originated. This value is optional.
[reply endpoint]	EndpointReference	0..1	An endpoint reference for the intended receiver of replies to this message. This value is optional.

Abstract WS-Addressing MAP name	MAP content type	Multiplicity	Description
[fault endpoint]	EndpointReference	0..1	An endpoint reference for the intended receiver of faults relating to this message. This value is optional.
[relationship]*	xs:anyURI plus optional attribute of type xs:anyURI	0..unbounded	A pair of values that indicate how this message relates to another message. The content of this element conveys the [message id] of the related message. An optional attribute conveys the relationship type. This value is optional.
[message id]	xs:anyURI		An absolute URI that uniquely identifies the message. This value is optional.

The abstract names in the previous tables are used to refer to the MAPs throughout this documentation.

The following example of a SOAP message contains WS-Addressing MAPs:

```
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
            xmlns:wsa="http://www.w3.org/2005/08/addressing"
            xmlns:fabrikam="http://example.com/fabrikam">
    <S:Header>
        ...
     <wsa:To>http://example.com/fabrikam/acct</wsa:To>
              <wsa:ReplyTo>
        <wsa:Address> http://example.com/fabrikam/acct</wsa:address>
              </wsa:ReplyTo>
            <wsa:Action>...</wsa:Action>
              <fabrikam:CustomerKey wsa:IsReferenceParameter='true'>123456789
              </fabrikam:CustomerKey>
             <fabrikam:ShoppingCart wsa:IsReferenceParameter='true'>ABCDEFG
             </fabrikam:ShoppingCart>
              ...
    </S:Header>
        <S:Body>
          ...
        </S:Body>
</S:Envelope>
```

Related concepts:

"What is SOAP?" on page 1604
SOAP is an XML message format used in Web service interactions. SOAP messages are typically sent over HTTP or JMS, but other transport protocols can be used. The use of SOAP in a specific Web service is described by a WSDL definition.

Related tasks:

"Processing Web service messages" on page 1601
Use WebSphere Message Broker nodes and services to connect to other Web services providers and consumers.

Message flows for Web services

Message flows that need to work with Web services can use either the SOAP domain or one of the XML domains.

The following topic describes fundamental scenarios. Any use of WS-Addressing or WS-Security requires use of the SOAP domain, but otherwise these scenarios apply to both types of message flow.

- "Web services scenarios" on page 1620

The following topics describe both types of flow.

- "SOAP domain message flows" on page 1635
- "XML domain message flows" on page 1643

The following topics describe web services tasks.

- "Working with WS-Addressing" on page 1650
- "Working with WSDL" on page 1661
- "Using timeouts with HTTP and SOAP nodes" on page 1595
- "Using SOAP MTOM with the SOAPReply, SOAPRequest, and SOAPAsyncRequest nodes" on page 1678

Related concepts:

"What is SOAP?" on page 1604
SOAP is an XML message format used in Web service interactions. SOAP messages are typically sent over HTTP or JMS, but other transport protocols can be used. The use of SOAP in a specific Web service is described by a WSDL definition.

"WS-Addressing" on page 1617
Web Services Addressing (WS-Addressing) is a World Wide Web Consortium (W3C) specification that aids interoperability between Web services by defining a standard way to address Web services and provide addressing information in messages.

"WS-Security" on page 765
Web Services Security (WS-Security) describes enhancements to SOAP messaging to provide quality of protection through message integrity, message confidentiality, and single message authentication. WS-Security mechanisms can be used to accommodate a wide variety of security models and encryption technologies.

Web services scenarios:

These common Web services scenarios are organized according to the role that is played by the broker.

A key consideration is whether a WSDL description for the Web service already exists.

In scenario A, the WSDL description exists and is imported and used by the message flow.

In scenario B, the WSDL description is generated in an existing message set. The WSDL is used by the message flow and might also be exported for use by an external client.

These scenarios are generic and can be implemented by using the SOAP domain, or an appropriate non-SOAP domain (XMLNSC, MRM, MIME) and basic transport nodes. If you need to use WS-Addressing or WS-Security for a particular implementation, use the SOAP domain.

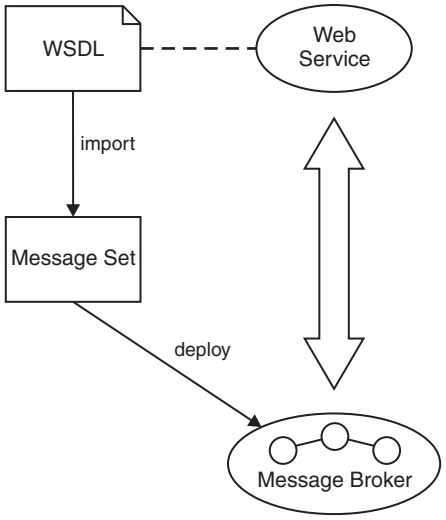
Scenario A: You want the broker to invoke an existing Web service:
See "Broker calls existing Web service" on page 1621

Scenario B: You want the broker to expose a Web service to a non-Web service client: See "Broker implements non-Web-service interface to new Web service" on page 1634

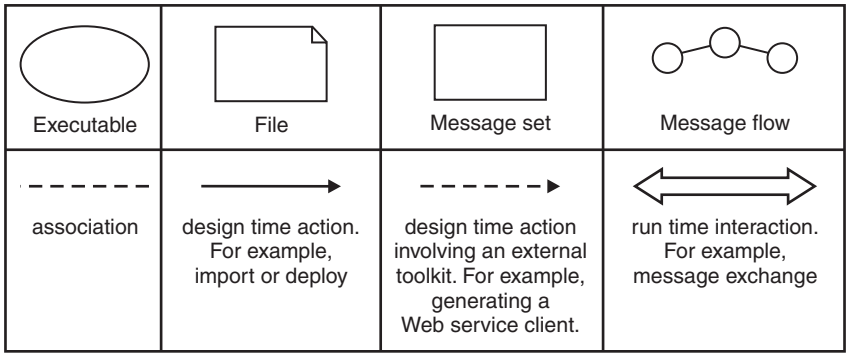
Broker calls existing Web service:

In this scenario, the broker calls an existing Web service implementation. The WSDL for the Web service already exists, and is imported to create a message set.

A message flow based on this message set sends a Web service request and receives the response, for example by using a SOAPRequest node.



Key to symbols:



Possible uses

- You want to call a Web service to do some processing as part of your message flow.
- You have an existing Web service and you want to provide a different interface to it. This could be an alternative Web services interface or a WebSphere MQ interface.
- You have an existing Web service and you want to change its implementation in some way without changing its interface; that is, the broker acts as an intermediary to the Web service. For instance a message flow could be used to enable auditing, or to transparently propagate the Web service response to another application.

Design steps

1. Import WSDL to create a message set containing definitions for the SOAP messages described by the WSDL.
2. Create a message flow to invoke the Web service. If the SOAP domain is used, the message flow uses a SOAPRequest node, SOAPAsyncRequest node, or a SOAPAsyncResponse node. The nodes are configured by using the WSDL imported in Step 1. If required, you can create a skeleton flow from scratch by dropping the WSDL onto a blank message flow editor canvas. If you use the SOAP domain, you must create the message flow by using transport nodes, and an XML or MIME domain. For example, if the WSDL binding specifies HTTP transport, and the request message is SOAP, you can use an HTTPRequest node with the XMLNSC domain. You can then configure the node manually with the endpoint information for the Web service.
3. Build a broker archive file for deployment. The broker archive file contains your message flow and the message set that contains the imported WSDL. The SOAP domain always requires the WSDL to be deployed, because messages are verified against it at run time; also WSDL information is included in the logical tree. The message set includes XML Schema definitions that can be used for message validation in the SOAP, XMLNSC, or MRM domains.

At run time

Your message flow creates an appropriately formatted Web service request, invokes the Web service, and parses the Web service response. If you use the SOAP domain, your message flow uses the SOAP logical tree model. If you do not use the SOAP domain, your message flow uses the logical tree for your selected domain; for example, you use the MIME domain if your Web service messages use SOAP with Attachments.

Example 1

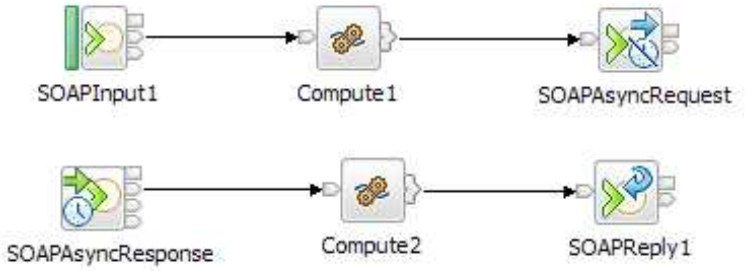
Web service intermediary

In this example a client application uses a Web service called Account, which is made available by another organization. The client is widely distributed in your company. The client uses an operation called getBalance, but the Account service is being modified to change the definition of the getBalance operation. You can construct message flows to provide an interface to the Account service, instead of modifying the client. The message flows can call the Account service to do the work, and the new Web service delegates to the original Web service. The client can now continue to use the Account service, by using the new message flows.

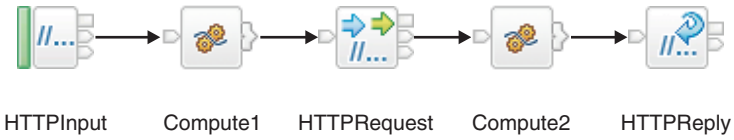
In the examples of typical message flow patterns shown here, the intermediate request node calls the Account service:

- Using SOAPInput, SOAPRequest, and SOAPReply nodes:

- Using SOAPInput, SOAPAsyncRequest, SOAPAsyncResponse, and SOAPReply nodes:



- Using HTTPInput, HTTPRequest, and HTTPReply nodes:



In the message flows in the example, Compute1 modifies the original getBalance message as required by the modified Account service, while Compute2 restores the response message to the original format. If you have imported or generated WSDL, you have a message model for the getBalance operation. If you have a message model defined for the getBalance operation, you can use Mapping nodes instead of Compute nodes.

HTTP details

If you use HTTP transport nodes, as shown in the example, you can configure the HTTPRequest node to generate HTTP headers from the headers that are received by the HTTPInput node. This configuration enables cookies and other application-specific headers to be passed through the message flow. The HTTPReply node can be used task to extract headers from the Web service response, to return to the originating client. To create this configuration, select **Generate default HTTP headers from** on both the HTTPRequest and HTTPReply nodes. Typically, you do not need the original request message to generate the reply to the client, and can select **Replace input message with Web service response** on the HTTPRequest node. If you do want to preserve data from the input request, you can store this in the LocalEnvironment in Compute1, and retrieve it in Compute2 for inclusion in the reply.

Example 2

Using a Web service

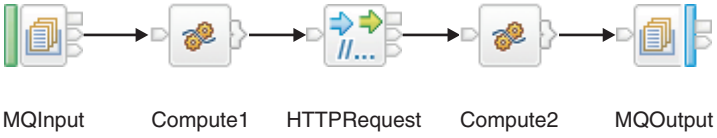
In this example, a WebSphere MQ message flow implements a process for the Human Resource department of your company. As part of this processing, the message flow calls a Web service to retrieve employee ID numbers. Employee ID numbers are maintained in the company's employee directory, which is accessed through a Web service.

In the examples of typical message flow patterns shown here the intermediate request node retrieves the employee ID:

- Using MQInput, SOAPRequest and MQOutput nodes:

- Using MQInput, SOAPAsyncRequest, SOAPAsyncResponse, and MQOutput nodes:

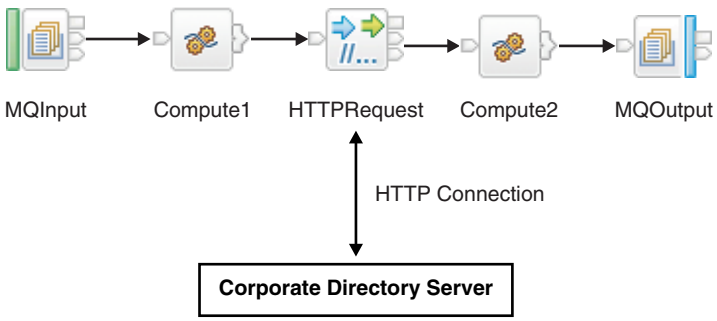
- Using MQInput, HTTPRequest, and MQOutput nodes:



In the message flows in the example, Compute1 prepares the Web service request message and Compute2 processes the response. For example, by incorporating the employee ID in another message. If you have a message model defined, you can use Mapping nodes instead of Compute nodes in these examples.

HTTP details

If you use HTTP transport nodes, as shown in the example, you typically clear the **Replace input message with Web service response** in the HTTPRequest node properties. The response from the corporate directory server is placed in a temporary location in the message tree. The temporary location is specified in the **Response message location** in tree property in the same node. In Compute2, you can code ESQL to retrieve the result, and update the final message.



Using the SOAP domain for these scenarios is preferred. For more information about choosing a domain for Web services, see “WebSphere Message Broker and Web services” on page 1602.

Related concepts:

“XML domain message flows” on page 1643
If you are not using the SOAP domain, your message flow must take account of the bitstream format of the Web service messages with which you are working. A different logical tree format is used by each domain.

“Broker implements new Web service interface”
In this scenario, the broker implements a new Web service interface. The WSDL for the Web service is generated from a message set and made available to clients. A message flow based on this WSDL and message set receives a request and then builds a response message by using data obtained from an existing non-Web-service application.

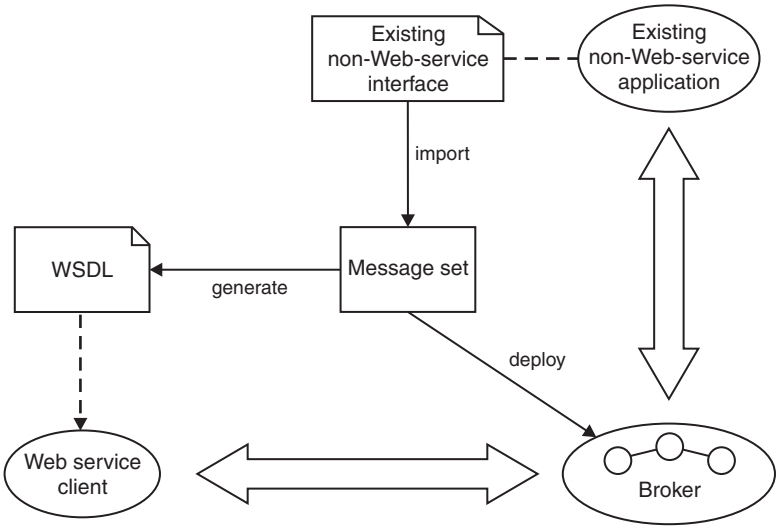
“Broker implements existing Web service interface” on page 1630
In this scenario, the broker implements an existing Web service interface. The WSDL for the Web service already exists, and is imported to create a message set. A message flow based on this message set receives a request, then builds a response message by using data obtained from an existing non-Web-service application.

“Broker implements non-Web-service interface to new Web service” on page 1634
In this Web service scenario, the broker provides compatibility with earlier versions for existing non-Web-service clients to call a new Web services implementation provided by a SOAP toolkit.

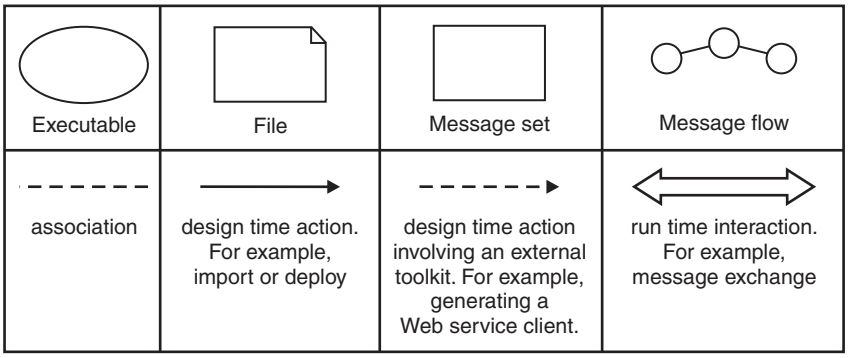
Broker implements new Web service interface:

In this scenario, the broker implements a new Web service interface. The WSDL for the Web service is generated from a message set and made available to clients. A message flow based on this WSDL and message set receives a request and then builds a response message by using data obtained from an existing non-Web-service application.

The following diagram shows a message set being created from an interface definition (for example, a header file) of an existing application that is not currently accessible as a Web service. A WSDL file is generated from the message set and exported for use by a Web service client. A message flow that uses the message set and WSDL is created to call the application. The message flow and message set are deployed to a broker, providing a Web service interface to the original application.



Key to symbols:



This scenario is sometimes referred to as a Web service facade. The design of the Web service interface typically involves some regrouping, restriction, or enhancement of the existing interface, and is not constrained by an existing WSDL definition.

Possible uses

- The broker provides a Web services interface to an existing application, optionally providing other mix-in capabilities such as auditing the requests made.
- Over time the implementation can be changed without affecting the interface presented to the Web services client.

Design steps

1. Create a message set for the business messages, possibly by importing an existing interface definition such as a C header file or COBOL copybook.
2. Generate a WSDL definition from the message set.
3. Use a SOAP toolkit such as Rational Application Developer to create a suitable Web services client based on the WSDL.
4. Develop a message flow to implement the Web service.

At run time

Your message flow receives a Web service request, converts it into a form expected by the existing application and invokes the existing application. The response from the existing application is converted into a valid Web service response.

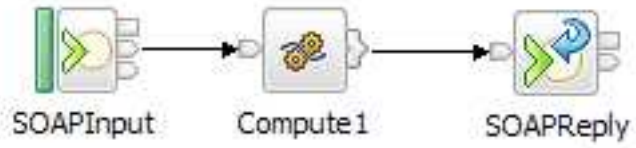
Example 1

In this example, an existing message flow is modified to provide a Web service. If the existing message flow models its data in a message set, a WSDL definition can be generated from that message set and made available to clients.

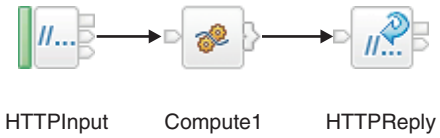
Most message flows that currently use WebSphere MQ for input or output can be adapted to support Web services as a replacement or additional protocol.

The following are typical message flow patterns. In each case the input and reply nodes replace or complement the original MQInput and MQOutput nodes. The main part of the flow is understood to do some useful processing.

- Using SOAPInput and SOAPReply nodes:



- Using HTTPInput and HTTPReply nodes:



If you use the SOAP domain, the logical tree shape will be different from the original domain and you will need to take account of this in the message flow. If you use the HTTP nodes with the original domain, the logical tree shape does not change. For information about choosing the domain, see "WebSphere Message Broker and Web services" on page 1602.

HTTP details

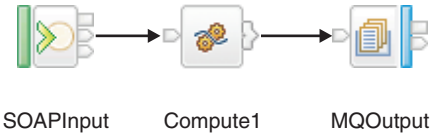
If you use the HTTP nodes, you can configure the HTTPReply node to generate a set of default HTTP headers for the reply message sent to the client. Generating a set of default HTTP headers reduces the modifications that you must make to convert the message flow from one that processes WebSphere MQ messages to a flow that processes HTTP messages.

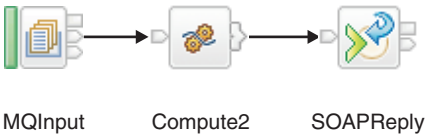
Example 2

In this example, a message flow is created to provide asynchronous access to a WebSphere MQ application.

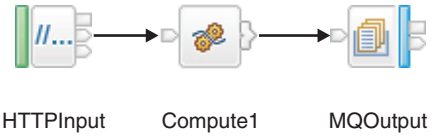
The following are typical message flow patterns. In each case the flow receives the Web service request and build the response by using data returned from the application over WebSphere MQ.

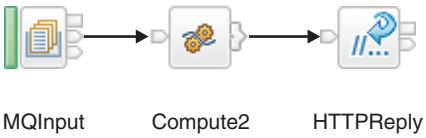
- Using two message flows with SOAPInput and SOAPReply nodes:





- Using two message flows with HTTPInput and HTTPReply nodes:





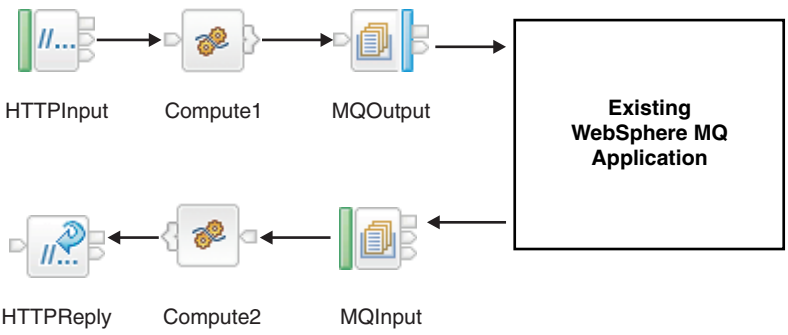
In each case, the first message flow receives inbound requests from a Web service client. The Compute1 node transforms the request and an MQOutput node sends the modified request to the existing application.

In the second message flow, an MQInput node receives the response from the application. The Compute2 node then transforms the message and propagates it to a reply node that responds to the original Web service client.

The Compute1 node must also save some correlation information to be retrieved by the Compute2 node, ensuring that the replies from the WebSphere MQ application are returned to the client that sent the original request.

HTTP details

Using HTTPInput and MQOutput nodes as the outbound message and MQInput and HTTPReply nodes as the response message:



One way to preserve the correlation information is for the Compute1 node to encode the correlation identifier in the outbound message. (Alternatively, the identifier can be stored in a database). The SOAPInput and HTTPInput nodes place the identifier as a field in the local environment tree and the Compute1 node can read and store this value. The location of the identifier differs between the SOAPInput and HTTPInput nodes, as described in the following sections.

SOAP nodes

The Compute2 node reads the SOAP reply identifier from the message and sets LocalEnvironment.Destination.SOAP.Reply.ReplyIdentifier by using this value. The SOAPReply node uses the reply identifier to ensure that the message reaches the correct HTTP client. In the ESQL module for the Compute1 node, include a code statement like the following statement:

```
SET OutputRoot.XMLNS.A.MessageID =
CAST(InputLocalEnvironment.Destination.SOAP.Reply.ReplyIdentifier AS CHARACTER);
```

In the ESQL module for the Compute2 node, include a code statement like the following statement:

```
SET OutputLocalEnvironment.Destination.SOAP.Reply.ReplyIdentifier =
CAST(InputRoot.XMLNS.A.MessageID AS BLOB);
```

HTTP nodes

The Compute2 node reads the HTTP request identifier from the message and sets LocalEnvironment.Destination.HTTP.RequestIdentifier by using this value. The HTTPReply node uses the request identifier to ensure that the message reaches the correct HTTP client. In the ESQL module for the Compute1 node, include a code statement like the following statement:

```
SET OutputRoot.XMLNS.A.MessageID =
CAST(InputLocalEnvironment.Destination.HTTP.RequestIdentifier AS CHARACTER);
```

In the ESQL module for the Compute2 node, include a code statement like the following statement:

```
SET OutputLocalEnvironment.Destination.HTTP.RequestIdentifier =
CAST(InputRoot.XMLNS.A.MessageID AS BLOB);
```

Related concepts:

"XML domain message flows" on page 1643
If you are not using the SOAP domain, your message flow must take account of the bitstream format of the Web service messages with which you are working. A different logical tree format is used by each domain.

"Broker calls existing Web service" on page 1621
In this scenario, the broker calls an existing Web service implementation. The

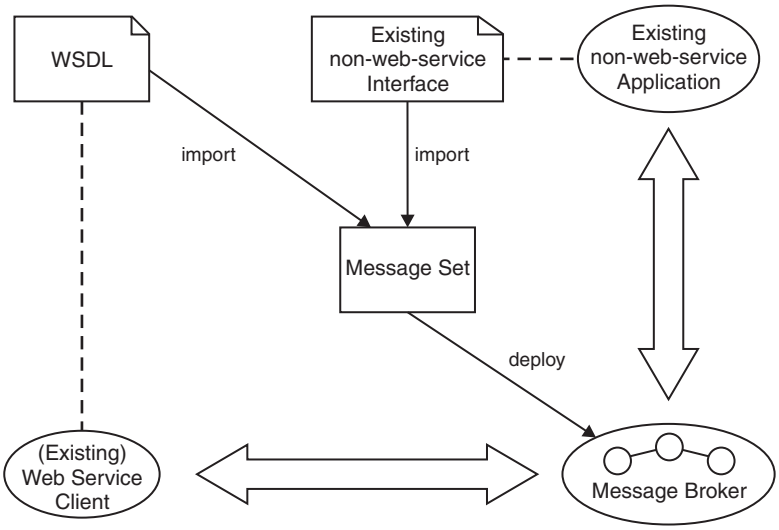
WSDL for the Web service already exists, and is imported to create a message set.

"Broker implements existing Web service interface"
In this scenario, the broker implements an existing Web service interface. The WSDL for the Web service already exists, and is imported to create a message set. A message flow based on this message set receives a request, then builds a response message by using data obtained from an existing non-Web-service application.

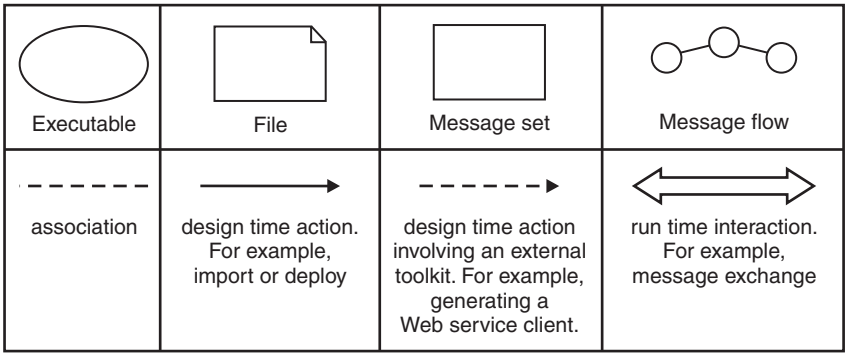
"Broker implements non-Web-service interface to new Web service" on page 1634
In this Web service scenario, the broker provides compatibility with earlier versions for existing non-Web-service clients to call a new Web services implementation provided by a SOAP toolkit.

Broker implements existing Web service interface:

In this scenario, the broker implements an existing Web service interface. The WSDL for the Web service already exists, and is imported to create a message set. A message flow based on this message set receives a request, then builds a response message by using data obtained from an existing non-Web-service application.



Key to symbols:



Possible uses

- The broker provides a Web service implementation with a different quality of service from existing implementations.
- The broker provides a migration strategy for the existing implementation.

Design steps

1. Import WSDL to create a message set containing definitions for the SOAP messages described by the WSDL.
2. Adapt the message set for the required existing interface, possibly by importing an existing interface definition such as a C header file or COBOL copybook.
3. Develop a message flow to implement the Web service.

At run time

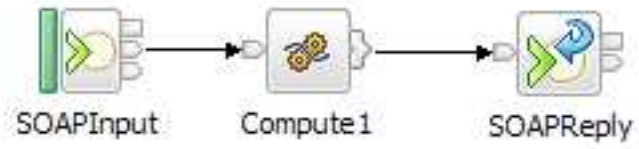
Your message flow receives a Web service request, converts it into a form expected by the existing application and invokes the existing application. The response from the existing application is converted into a valid Web service response.

Example 1

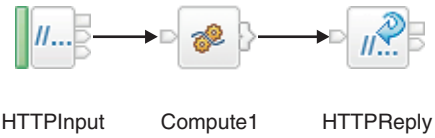
In this example, an existing HTTP Web service client provides information on a particular subject (stock prices or exchange rates, for example). You want to replace this service with an inhouse database lookup solution, but want to make no changes to the clients because these are widely deployed.

Typical message flow patterns are shown in the following examples. In each case the intermediate request node retrieves the information from the database:

1. Using SOAPInput and SOAPReply nodes:



2. Using HTTPInput and HTTPReply nodes:



In the flows above, the input node receives the Web service request. Compute1 then retrieves the required information from the database and generates the required Web service response by using this data. The response is returned to the client by the reply node. In the examples you can use Mapping nodes instead of Compute nodes.

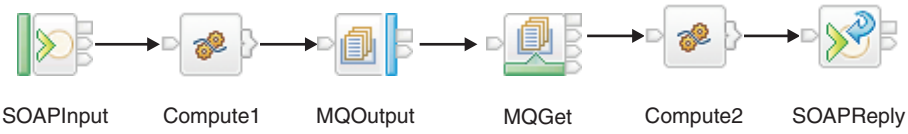
Example 2

In this example, an existing application is exposed as a Web service, but there is a constraint on the interface with the Web service, because a widely distributed

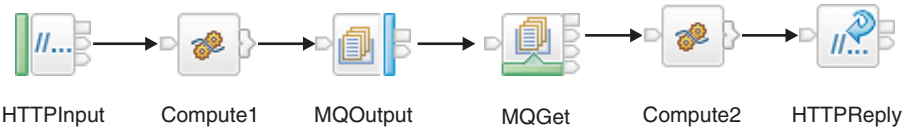
client already uses a similar service that is defined by an existing WSDL definition. The broker offers the same interface to the client, this might be because the original service offers a different quality of service or is to be discontinued.

Typical message flow patterns are shown in the following examples. In each case the message flows receive the Web service request and build the response by using data returned from the application over WebSphere MQ.

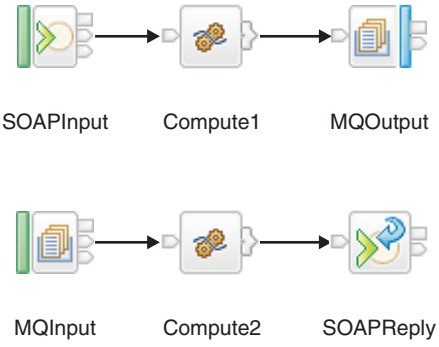
1. Using SOAPInput, SOAPReply and MQGet nodes:



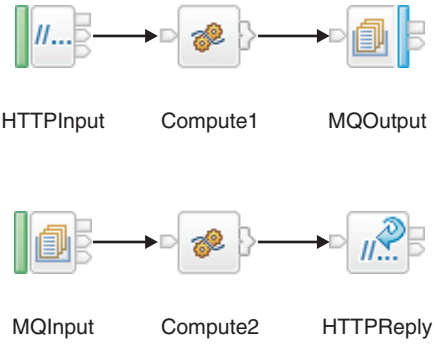
2. Using HTTPInput, HTTPReply and MQGet nodes:



3. Using two message flows with SOAPInput, SOAPReply nodes:



4. Using two message flows with HTTPInput and HTTPReply nodes:



The steps to develop the message flow are:

1. Create a message model for the existing application interface, for example, by importing a C header file for the application.
2. Import an existing WSDL definition for the client.

3. Create a flow by using the message set to implement the Web service interface and mediate with the existing application.

Message flows 1 and 2 show a synchronous call to the application by using MQOutput and MQGet nodes. You can set a timeout in the MQGet node, to allow for failure of the remote application. The request-reply translation is handled in a single transaction enabling simple rollback and recovery. However, each incoming request has to be fully processed and responded to before moving onto the next request. This behavior might affect performance if the application cannot respond quickly. The message flows shown in examples 3 and 4, show an asynchronous equivalent. In each case the first flow stops after sending the message to the application, and becomes available to handle further requests. However, this scenario requires a correlation context to be saved in the request flow, and restored in the reply flow. You can store the correlation context on a queue, then use an MQGet node in the reply flow to retrieve it. This flow design enables the requests to be dispatched to the application promptly, and replies to be returned in the order that they are received. In the examples you can use Mapping nodes instead of Compute nodes.

The use of the SOAP domain for these scenarios is preferred. For more information about choosing a domain for Web services, see "WebSphere Message Broker and Web services" on page 1602.

For more information about the asynchronous request-reply scenario, see "A request-response scenario that uses an MQGet node" on page 1569.

The asynchronous request-reply scenario is also detailed in the following sample which can be adapted for Web service usage:

- Coordinated Request Reply

Another Web services scenario is described in the sample:

- Web services using HTTP nodes

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Related concepts:

"XML domain message flows" on page 1643
If you are not using the SOAP domain, your message flow must take account of the bitstream format of the Web service messages with which you are working. A different logical tree format is used by each domain.

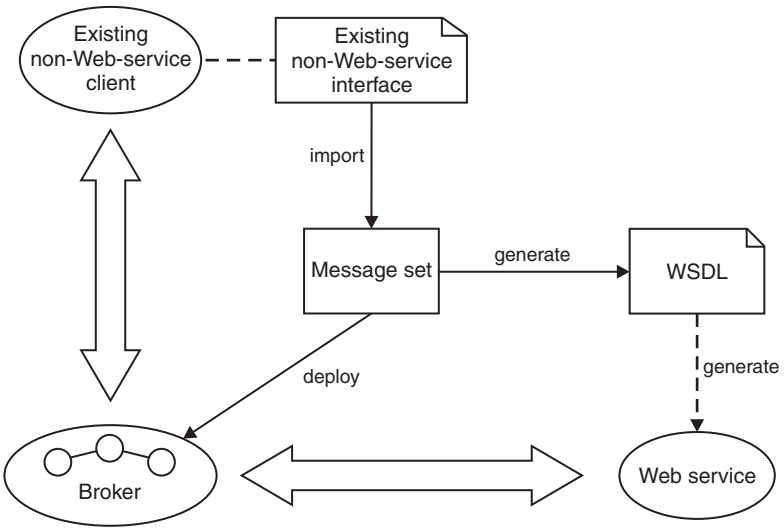
"Broker calls existing Web service" on page 1621
In this scenario, the broker calls an existing Web service implementation. The WSDL for the Web service already exists, and is imported to create a message set.

"Broker implements new Web service interface" on page 1625
In this scenario, the broker implements a new Web service interface. The WSDL for the Web service is generated from a message set and made available to clients. A message flow based on this WSDL and message set receives a request and then builds a response message by using data obtained from an existing non-Web-service application.

"Broker implements non-Web-service interface to new Web service" on page 1634
In this Web service scenario, the broker provides compatibility with earlier versions for existing non-Web-service clients to call a new Web services implementation provided by a SOAP toolkit.

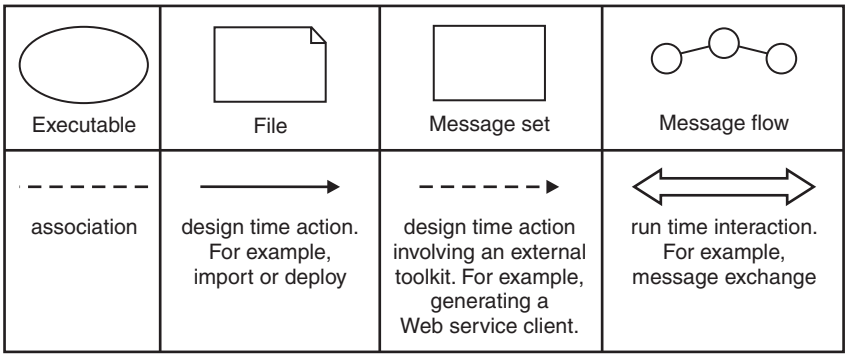
Broker implements non-Web-service interface to new Web service:

In this Web service scenario, the broker provides compatibility with earlier versions for existing non-Web-service clients to call a new Web services implementation provided by a SOAP toolkit.



The diagram shows a message set being created from an interface definition (for example, a header file) that is used by an existing client application. A WSDL file is generated from the message set and is used to create a new Web service implementation. A message flow that uses the message set is created to call the new Web service. The message flow and message set are deployed to a broker, providing the original application interface to the new Web service implementation.

Key to symbols:



Possible uses

You want to migrate an application to a Web service implementation, for example an EJB implementation hosted by an application server to offer better scalability.

However, a significant number of your users have existing clients that cannot be immediately replaced. Existing clients can use the broker to use the new Web service implementation.

Design steps

1. Create a message set for the business messages, for example, by importing an existing interface definition such as a C header file or COBOL copybook.
2. Generate a WSDL definition from the message set.
3. Use a SOAP toolkit or application server to create a suitable Web services implementation based on the WSDL.
4. Develop a message flow to mediate between the original existing client and the new Web service.

At run time

Your message flow receives a request from the existing client, converts it into a Web services request and invokes the Web service. The response from the Web service is converted into a form understood by the existing client.

Related concepts:

"XML domain message flows" on page 1643
If you are not using the SOAP domain, your message flow must take account of the bitstream format of the Web service messages with which you are working. A different logical tree format is used by each domain.

"Broker calls existing Web service" on page 1621
In this scenario, the broker calls an existing Web service implementation. The WSDL for the Web service already exists, and is imported to create a message set.

"Broker implements new Web service interface" on page 1625
In this scenario, the broker implements a new Web service interface. The WSDL for the Web service is generated from a message set and made available to clients. A message flow based on this WSDL and message set receives a request and then builds a response message by using data obtained from an existing non-Web-service application.

"Broker implements existing Web service interface" on page 1630
In this scenario, the broker implements an existing Web service interface. The WSDL for the Web service already exists, and is imported to create a message set. A message flow based on this message set receives a request, then builds a response message by using data obtained from an existing non-Web-service application.

SOAP domain message flows:

SOAP domain message flows typically use SOAP nodes, WSDL, and a common logical tree format that is independent of the exact format of the Web service message.

The following nodes are provided for use in the SOAP domain:

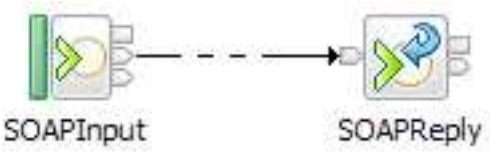
- "SOAPInput node" on page 4795
- "SOAPReply node" on page 4819
- "SOAPRequest node" on page 4828
- "SOAPAsyncRequest node" on page 4750
- "SOAPAsyncResponse node" on page 4777

The following nodes can also be used to simplify SOAP payload processing in a message flow; these nodes are not specific to the SOAP domain.

- "SOAPEnvelope node" on page 4786
- "SOAPExtract node" on page 4790

The SOAP nodes are used together in the following basic patterns:

- As a Web service provider, for example:



- As a Web service consumer, for example:

Or:

- As a Web service facade, for example, by combining the provider and consumer scenarios:

You can use the SOAPExtract node in conjunction with these patterns to extract the SOAP payload. If you are working with the HTTP nodes, you can use the SOAPEnvelope node to rebuild a SOAP envelope.

The main SOAP domain nodes are typically configured by WSDL, and in this mode, a prerequisite for a SOAP domain message flow is a message set containing deployable WSDL. To create a message set containing deployable WSDL, either import existing WSDL or generate WSDL from an existing message set. For more information about creating a new message definition from WSDL, see “Importing from WSDL” on page 2946. For more information about generating WSDL from an existing message set, see “WSDL generation” on page 6340.

Alternatively you can create a new message set and skeleton message flow in one step using the procedure described in “Creating an application based on WSDL or XSD files” on page 1413.

The WSDL then appears in the workbench under **Deployable WSDL** below the message set, although if you have selected `Hide Categories` on the message set, the category heading itself is not shown.

Deployable WSDL can then be used to configure SOAP nodes. You can do this by dragging the WSDL resource onto the node or by selecting the required WSDL resource from the node properties.

Alternatively a new skeleton message flow can be created by dragging and dropping the WSDL on to a blank canvas in the Message Flow editor.

The WSDL is deployed with your completed message flow, enabling the broker to raise exceptions if a Web service message does not correspond to the specified WSDL description.

You can change the operation mode of the SOAP domain nodes so that they act in gateway mode. In gateway mode, a WSDL is not required to configure the nodes because they handle generic request/response and one-way SOAP messages that are not tied to a specific WSDL. For more details, see "Gateway operation mode for SOAP nodes" on page 1645.

A client can send an HTTP GET request to the endpoint exposed by a SOAPInput node, suffixed with a query string `?wsdl`, and receive a response with the WSDL definition used to configure the flow; see "Using WSDL to configure message flows" on page 1664.

The SOAP domain uses a common logical tree format that is independent of the exact format of the Web service message. For details of the SOAP tree format, see "SOAP tree overview" on page 1611. Useful WSDL information is included in the logical tree under `SOAP.Context`.

Related concepts:

"XML domain message flows" on page 1643
If you are not using the SOAP domain, your message flow must take account of the bitstream format of the Web service messages with which you are working. A different logical tree format is used by each domain.

"Web services scenarios" on page 1620
These common Web services scenarios are organized according to the role that is played by the broker.

Related tasks:

"Example usage of WS-Addressing"
Set up a sample message flow by using WS-Addressing, and test the flow.

Example usage of WS-Addressing:

Set up a sample message flow by using WS-Addressing, and test the flow.

About this task

Complete the steps in the following set of topics.

1. Build the main message flow that includes SOAP nodes, a Filter node, and a Mapping node by following the instructions in "Building the main message flow" on page 1638.
2. Build a logging message flow, so that you can send a reply to an address different from the originating client, by following the instructions in "Building the logger message flow" on page 1640.
3. Deploy the message flows by following the instructions in "Deploying the message flows" on page 1641.
4. Test the message flows, by using a tool that uses HTTP. This task illustrates that the contents of the SOAP message determine where the replies are routed. Follow the instructions in "Testing the message flows" on page 1642.

Related concepts:

"How to use WS-Addressing" on page 1650
An overview of how you use WS-Addressing with WebSphere Message Broker.
"Message flows for Web services" on page 1619
Message flows that need to work with Web services can use either the SOAP domain or one of the XML domains.
"WS-Addressing" on page 1617
Web Services Addressing (WS-Addressing) is a World Wide Web Consortium (W3C) specification that aids interoperability between Web services by defining a standard way to address Web services and provide addressing information in messages.

Related reference:

"Broker Application Development perspective" on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.
"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Building the main message flow:

You can construct a sample main message flow to use with WS-Addressing.

About this task

These steps are the first in a set of instructions on setting up your system to use WS-Addressing with WebSphere Message Broker; they explain how to set up a message flow to use this feature. This topic describes how to construct a sample main message flow when using WS-Addressing.

Procedure

1. Switch to the Broker Application Development perspective.
2. Create message flow and message set projects using the Start from WSDL and/or XSD files wizard. See "Creating an application based on WSDL or XSD files" on page 1413 for instructions.
3. Select the **Web Services** folder on the message flow palette to display the contents, and drag a SOAPInput node onto the canvas.
4. Add a SOAPExtract node to the message flow to remove the SOAP envelope from the incoming message, followed by a SOAPReply node. Wire the Out terminal of the SOAPInput node to the In terminal of the SOAPExtract node, and wire the Out terminal of the SOAPExtract node to the In terminal of the SOAPReply node.
5. Select the WSDL file that you need under **Deployable WSDL** from the Active Working Set, and drag it onto the SOAPInput node. The SOAPInput node is configured with the WSDL.
6. Select the **Construction** folder on the message flow palette to display the contents.
7. Select a Trace node and move the mouse to the right of the SOAPExtract node.
   a. Click the left mouse button to add the node to the message flow. The name is selected automatically.
   b. Press **Enter** to accept the default name.

c. Wire the submitPORequest terminal of the SOAPExtract node to the In terminal of the Trace node.

8. Select the Trace node to display the properties.
   a. Use the menu to set **Destination** to `File`
   b. Set the **File path** that you require.
   c. Enter the **Pattern** that you require.
9. Expand the **Routing** folder on the palette and select **Filter**.
10. Add the Filter node to the right of the Trace node.
    a. Type the name for the node that you require and press **Enter**.
    b. Wire the Out terminal of the Trace node to the In terminal of the Filter node.
11. Select the Filter node to display the properties.
    a. Enter the **Data source** name that you require.
    b. Change the name of **Filter expression** to the name that you selected for the Filter node.
    c. Clear the **Throw exception on database error** check box.
12. Double-click the Filter node to open the ESQL editor. Create or change the ESQL for the node; for more information, see "Creating ESQL for a node" on page 2394 and "Modifying ESQL for a node" on page 2398.
13. Expand the **Transformation** folder on the palette and select a Mapping node.
14. Add the Mapping node to the right of the Filter node.
    a. Type the name for the node that you require and press **Enter**.
    b. Wire the True terminal of the Filter node to the In terminal of the Mapping node.
    c. Wire the Out terminal of the Mapping node to the In terminal of the Reply node.
15. Select the Mapping node to display the properties, and change the name of Mapping routine to the name that you selected for the Mapping node.
16. Double-click the Mapping node to open the mapping editor.
    a. Select **submitPORequest** as the map source.
    b. Select **SOAP_Domain_msg** as the map target.
    c. Click **OK**
    d. Click **OK** on the tip that is displayed to open the mapping editor.
17. Select **Properties** in both the **source** and **target** pane, right-click, and click **Map by Name**.
    a. Map the source properties to the target properties using drag-and-drop mapping. The Map by Name dialog box appears.
    b. Click **OK** to perform the mapping.
18. Expand **SOAP_Domain_Msg**, then **Body** and **message items** in the target pane.
19. Right-click **Wildcard Message** in the target pane, and click **Create new Submap**.
    a. Expand **Wildcard**.
    b. Scroll down and click **submitPOResponse**.
    c. Click **OK** to create the submap.
20. Use drag-and-drop mapping to select the items that you need in the **Source** pane.

21. Select the first item that you need in the **Target** pane, right-click, and **Enter Expression**. Enter the value that you require in the expression editor and press **Enter** to complete the mapping.

    Repeat the above steps for all the items that you require in the **Target** pane, and save the submap and map by pressing **Ctrl+S**.
22. Expand the **Construction** folder on the message flow editor and select a Throw node.
23. Add the Throw node above the Mapping node
    a. Type the name for the node that you require and press **Enter**.
    b. Wire the False and Unknown terminals of the Filter node to the In terminal of the Throw node.
24. Select the Throw node and, in the Node Properties pane, enter the text that you require in **Message text**.
25. Select the SOAPInput node to display the Node Properties.
    a. Select the **WS Extensions** tab.
    b. Select **Use WS-Addressing** .
26. Save the message flow.

Related concepts:

"WS-Addressing" on page 1617
Web Services Addressing (WS-Addressing) is a World Wide Web Consortium (W3C) specification that aids interoperability between Web services by defining a standard way to address Web services and provide addressing information in messages.

Related reference:

"Broker Application Development perspective" on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

Building the logger message flow:

This is the second of a set of instructions on setting up your system to use WS-Addressing with WebSphere Message Broker, and illustrates the use of a reply being sent to an address other than the originating client.

About this task

This topic describes the construction of a sample logger message flow when using WS-Addressing. This flow echoes back the input while creating a trace entry in a file to indicate that the flow has been invoked.

Procedure

1. Switch to the Broker Application Development perspective.
2. Select the message flow name that you used in "Building the main message flow" on page 1638
3. Press the right mouse button and select **New->MessageFlow**.
   a. Enter the name that you require for this message flow; for example, `Logger`.
   b. Press **Finish** to create the flow.
4. Select the **HTTP** folder on the message flow palette to display the contents.
5. Select an **HTTPInput** node and move the mouse to the left side of the canvas.

a. Click the left mouse button to add the node to the message flow and enter the name `Logger`.
b. Press **Enter** to finish.

6. Select the **HTTPReply** node from the palette and move the mouse to the right of the HTTPInput node, leaving room for a node in between.
   a. Click the left mouse button to add the node to the message flow and enter the name that you require; for example, `Logger`.
   b. Press **Enter** to finish.
7. Select the **Construction** folder on the message flow palette to display the contents.
8. Select a **Trace** node and move the mouse to the right of the HTTPInput node.
   a. Click the left mouse button to add the node to the message flow and enter the name that you require; for example `Trace`.
   b. Press **Enter**.
   c. Wire the `out` terminal of the HTTPInput node to the `In` terminal of the Trace node.
   d. Wire the `out` terminal of the Trace node to the `In` terminal of the HTTPReply node.
9. Select the HTTPInput node to display the properties. In the **Basic** tab:
   a. Enter the **Data source** name that you require.
   b. Change the name of the input node `Logger` as the **Path suffix for URL**.
10. Select the **Input Message Parsing** tab and select `XMLNSC` as the **Message domain**.
11. Select the Trace node to display the properties.
    a. Set **Destination** to `File`
    b. Set the **File path** that you require.
    c. Enter the **Pattern** that you require.
12. Save the message flow.

Related concepts:

"WS-Addressing" on page 1617
Web Services Addressing (WS-Addressing) is a World Wide Web Consortium (W3C) specification that aids interoperability between Web services by defining a standard way to address Web services and provide addressing information in messages.

Related reference:

"Broker Application Development perspective" on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

Deploying the message flows:

This is the third of a set of instructions on setting up your system to use WS-Addressing with WebSphere Message Broker, and illustrates the deployment of the message flows.

About this task

This topic describes the deployment of the message flows that you have already constructed.

Procedure

1. Select the **LocalProject** project under **Broker Archives** in the navigator pane.
   a. Press the right mouse button.
   b. Select **New** > **Message Broker Archive**.
   c. Use the drop-down menu to change the **Project** to `LocalProject`.
   d. Enter the name you selected for the main message flow described in "Building the main message flow" on page 1638 in **Name**.
   e. Press **Finish** to open the Broker Archive Editor.
2. In the Broker Archive Editor, complete the following steps.
   a. In the **Filter working set** list box select the working set name that you used for the main message flow.
   b. In **Message Flows** select the message flow names that you used for the main message flow and logger message flow.
   c. In **Message Sets** select the message set name that you used for the main message set.
   d. Press **Build broker archive** and confirm that the build operation was successful.
   e. Save the updated broker archive file.
3. To deploy the message flows to the default execution groups, right-click the broker archive (BAR) file that you have just built, then click **Deploy File**.
4. Select the default execution group and click **OK** to start the deployment. Ensure that you receive a successful response message, and press **OK** to dismiss the information dialog.
5. Use the **Deployment Log** to confirm that the deploy operation was successful:

Related concepts:

"WS-Addressing" on page 1617
Web Services Addressing (WS-Addressing) is a World Wide Web Consortium (W3C) specification that aids interoperability between Web services by defining a standard way to address Web services and provide addressing information in messages.

Related tasks:

"Example usage of WS-Addressing" on page 1637
Set up a sample message flow by using WS-Addressing, and test the flow.

Related reference:

"Broker Application Development perspective" on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

Testing the message flows:

This is the fourth of a set of instructions about how to set up your system to use WS-Addressing with WebSphere Message Broker; these instructions illustrate how to test the message flows.

About this task

This topic describes how to test the message flows that you have already constructed. In this scenario, you use a tool that uses HTTP protocol rather than WebSphere MQ protocol. You can use any tool that has the facilities that are described in the following procedure.

Procedure

1. Start the tool and select `http://localhost:nnnn/`, where *nnnn* is the address of the port that you are using.
2. Set the URL to the host, port, and selection for the deployed flow. The SOAPInput nodes listen on a different port from the HTTP nodes, and the listener is built into the execution group rather than using a different process.
3. The SOAPInput node expects a SOAPAction entry in the HTTP headers, therefore you must add one.
   a. Click **Add New Header**.
   b. Enter the **Value** part of the header. The value must match the `SOAPAction` attribute of the `SOAP:operation` element in your code.
   c. Select **New Header** in the **Name** pane.
   d. Change the name from **New Header** to `SOAPAction` and click **Enter**.
4. Select **Load File** and go to the directory that contains the XML file you want to use.
5. Select the file and click **Open**. Note the following conditions:
   - If the message does not include any WS-Addressing entries, the `ReplyTo` and `FaultTo` locations default to `anonymous`. This means that the results are returned on the original client connection.
   - If the message includes a WS-Addressing header (`ReplyTo`) with a value of `anonymous`, the reply is returned to the original client by using the original TCP/IP connection.
   - If the message includes a WS-Addressing header with a value of `FaultTo` explicitly included, the reply is returned to that address rather than the default of using the location that was specified in the `ReplyTo` header.
6. Click **Send** to test the flow. The result appears in the right pane.

Related concepts:

"WS-Addressing" on page 1617
Web Services Addressing (WS-Addressing) is a World Wide Web Consortium (W3C) specification that aids interoperability between Web services by defining a standard way to address Web services and provide addressing information in messages.

Related tasks:

"Example usage of WS-Addressing" on page 1637
Set up a sample message flow by using WS-Addressing, and test the flow.

Related reference:

"Broker Application Development perspective" on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

XML domain message flows:

If you are not using the SOAP domain, your message flow must take account of the bitstream format of the Web service messages with which you are working. A different logical tree format is used by each domain.

If the messages are SOAP, you can use either the XMLNSC domain or the MRM XML domain. Both domains offer validation. The XMLNSC domain is more efficient, while the MRM XML domain can be useful if you have specific message transformation requirements (for example, if your message flow also uses binary data formats).

If the messages use MIME (for example, SOAP with Attachments or MTOM), you can use the MIME domain. In this case, your message flow typically needs to identify at least the MIME part that corresponds to the SOAP payload, then explicitly parse this part by using the XMLNSC or MRM domain.

In the SOAP domain, WSDL is used to configure your nodes automatically with the appropriate endpoint information. If you are not using the SOAP domain, select and configure the transport nodes manually. Typical WSDL bindings are:

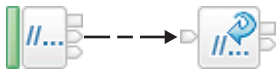
- SOAP/HTTP; in which case, implement a flow by using HTTP nodes. Use the HTTPInput and HTTPReply nodes if a flow implements a Web service, or use the HTTPRequest node if a flow calls a Web service.
- SOAP/JMS; where you implement a flow by using JMS or MQ nodes.

You can configure message flows that receive input messages from clients by using one transport, and interact with a Web service or legacy application by using another.

You can propagate a message to more than one location. For example, the Web service response to be returned to a client by an HTTPReply node might first be sent to an auditing application by using an MQOutput node, after making any required adjustments to the message headers.

Nodes are used together in the following basic patterns, by using HTTP nodes as example transports:

- As a Web service provider, for example:



- As a Web service consumer, for example:

- As a Web service facade, for example:

If required, you can use the SOAPExtract and SOAPEnvelope nodes in conjunction with these patterns to respectively extract the SOAP payload and rebuild a SOAP Envelope.

To enable your message flow to validate messages, deploy an appropriate message set with the flow. An appropriate message set is created either by importing existing WSDL or by generating WSDL from an existing message set. For details

about importing existing WSDL, see “Importing from WSDL” on page 2946. For details about generating WSDL from an existing message set, see “WSDL generation” on page 6340.

You can also create a new message set and flow based on existing WSDL or XSD files; for details, see “Creating an application based on WSDL or XSD files” on page 1413.

The generated message set contains message definitions for the relevant SOAP Envelope version and for the XML payload data defined by the WSDL. In the XMLNSC or MRM XML domains, messages can be validated against the message set; for details, see “Validating messages” on page 1478.

Related concepts:

“Processing HTTP messages” on page 1579
Hypertext Transfer Protocol (HTTP) is an Internet protocol that is used to transfer and display hypertext and XML documents on the Web.

“Generate WSDL” on page 1274
A *Web Services Description Language* (WSDL) document specifies the interface to a web service, and enables a web service client to start it. A WSDL document that is generated from a message model defines web service requests and responses in terms of the messages that you have defined in that message model.

“Message flows overview” on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

Chapter 9, “Developing message flow applications,” on page 1019
Develop message flows to process your business messages and data.

Chapter 11, “Packaging and deploying,” on page 3209
Package resources that you create in the WebSphere Message Broker Toolkit, such as message flows, and deploy them to execution groups on brokers.

Related reference:

“Built-in nodes” on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Gateway operation mode for SOAP nodes:

The SOAPInput, SOAPRequest, and SOAPAsyncRequest nodes have two operation modes: WSDL mode, and Gateway mode.

If the node is configured to be in WSDL mode, which is the default, the web service operations performed by the node are specified by a WSDL that configures the node. In Gateway mode, SOAP nodes in a message flow handle generic SOAP request messages in the following scenarios.

Provider scenario
: In a provider scenario, WebSphere Message Broker receives generic SOAP/HTTP or SOAP/JMS requests by using a SOAPInput node, and sends a reply to the originating client by using a SOAPReply node. A single SOAPInput node can receive any SOAP request message, and is not configured with a WSDL.

Consumer scenario
: In a consumer scenario, WebSphere Message Broker can route a SOAP request to any external web service provider by using the SOAPRequest

node or a pair of SOAPAsyncRequest and SOAPAsyncResponse nodes. Endpoint information can be specified in the local environment, which is used dynamically at run time to send the outbound request message.

Façade scenario
In a façade scenario, WebSphere Message Broker can receive SOAP requests from multiple different clients, and route them to any one of multiple back end web service providers, or existing systems. Endpoint information for the back-end web service provider can be specified dynamically at run time by setting values in the local environment.

The SOAP nodes act and are configured differently in the two different operation modes.

- If you configure a node in Gateway mode, some node properties are disabled because they are not applicable in Gateway mode.
  - The WSDL-specific properties are disabled when the node is switched to Gateway mode. The values are not cleared, and can be used if the node is switched back to WSDL mode.
  - The Response Message Parsing properties on the SOAPRequest node are disabled in Gateway mode.
  - Validation of the operation according to the WSDL does not take place because no WSDL exists against which to validate in Gateway mode. If a SOAPAsyncRequest and SOAPAsyncResponse node pair is used in Gateway mode, the validation properties on the SOAPAsyncResponse node are disabled at run time, regardless of the validation settings in the BAR file.
  - No message set is associated with the node, and no schema is used. Therefore, no message validation takes place in Gateway mode. You can configure the flow downstream to enable validation.
  - Outbound MTOM is not supported in Gateway mode because validation cannot take place. However, you can enable validation, and therefore enable MTOM, by setting the Message Set in the local environment. For example:

    ```
    SET OutputRoot.Properties.MessageSet = 'myMessageSet';
    ```

    You might need to also enable validation on the Compute node or the SOAPReply node.
  - In the `User-defined SOAP headers` table on the SOAPInput node, all operations are set to `*`. The `WSDL-defined SOAP headers` table is cleared.
- Transport properties must be configured. The transport-specific properties are mandatory depending on the `Transport` property; for example, the JMS transport properties are mandatory if the selected transport is `JMS`.
- A SOAPInput node can only be configured to receive messages of a single specified transport, for example, HTTP. Use separate input nodes to send or receive messages with different transports.
- A SOAPRequest node can only be configured to send messages of a single specified transport, for example, JMS. However, you can change the transport for any message using the local environment.
- A SOAPAsyncRequest node can only be configured to send and receive messages over a single transport, for example, JMS, and this transport is always used for the paired SOAPAsyncResponse node to receive the response message. However, you can change the outbound request transport for any message by using the local environment. For example, if a SOAPAsyncRequest node is configured to use JMS transport, its paired SOAPAsyncResponse node always expects to receive responses over JMS, and this cannot be changed. At run time, the SOAPAsyncRequest node also then assumes JMS as the default transport.

However, you can instruct it to instead send the request over HTTP by using the local environment. This request includes the WSA:ReplyTo JMS address for the SOAPAsyncResponse node.

- If a SOAPInput node receives a one-way SOAP request, the node attempts to detect that it is a one-way message. However, the node cannot detect all cases, and therefore it is sometimes necessary to instruct the SOAPReply node that it is a one-way message, by using the local environment. For example:

  ```
  SET OutputLocalEnvironment.Destination.SOAP.Reply.Gateway.OneWay = True;
  ```

  For more information, see "One-way messages in Gateway mode" on page 1648.
- If you use the SOAPAsyncRequest node in Gateway mode, you must set the WS-Addressing `Action` property in the local environment in the message flow before the SOAPAsyncRequest node. Set this property by using `OutputLocalEnvironment.Destination.SOAP.Request.WSA.Action`.
- If you use the SOAPRequest node in Gateway mode and the remote service provider expects a SOAPAction, set the SOAPAction in the flow. In Gateway mode the SOAPAction from a WSDL is not available to the node. For example, to set the SOAPAction using ESQL:

  ```
  SET OutputRoot.HTTPRequestHeader.SOAPAction = 'mySoapAction';
  ```

  By default the SOAPRequest node sends an empty SOAPAction of `""`.
- If you use the SOAPRequest node in Gateway mode and are using WS-Addressing, you must set the WS-Addressing `Action` property in the local environment in the message flow before the SOAPRequest node. Set this property by using `OutputLocalEnvironment.Destination.SOAP.Request.WSA.Action`.
- In Gateway mode, you can add inbound "must understand" headers to the SOAPInput node or to the SOAPRequest and SOAPAsyncRequest nodes by specifying the details on the node property. However, if you will be adding services dynamically, where all the "must understand" headers cannot be known in advance, you can add these headers with a wildcard (`*`) for name, namespace, operation, or any combination of the three. This removes the need to redeploy your message flow when new services are added. However, consider that if you add a wildcard for the name, namespace, and also operation, this means that all headers with a "must understand" flag are allowed into the flow.
- If you use the SOAPReply node as part of a façade message flow, with the SOAPInput node set to Gateway mode, and the SOAPRequest node acting in WSDL mode, disable validation on the SOAPReply node or add an explicit Message Set in the Properties folder as documented above. If you do not disable validation or reference a Message Set in the Properties folder, parsing errors occur when the message is serialized.
- In Gateway mode, the SOAP nodes send SOAP 1.1 messages by default, although they also accept inbound SOAP 1.2 messages. To send an outbound SOAP 1.2 message, set the `SOAP.Context` field to indicate that SOAP 1.2 is required. For example, to set this field using ESQL:

  ```
  SET OutputRoot.SOAP.Context.SOAP_Version = '1.2';
  ```

  The outbound message then uses a SOAP 1.2 SOAP Envelope. You can also set the namespace prefix used by the SOAP Envelope. For example, by using ESQL:

  ```
  DECLARE soapenc NAMESPACE 'http://www.w3.org/2003/05/soap-envelope';
  SET OutputRoot.SOAP.Context.Namespace.(SOAP.NamespaceDecl)xmlns:soap12 = soapenc;
  SET OutputRoot.SOAP.Context.SOAP_Version = '1.2';
  ```

  In this example, `soap12` is the prefix used in the outbound message.

Related concepts:

"SOAP nodes" on page 1609
The SOAP nodes act as points in the flow where Web service processing is configured and applied. Properties on the SOAP nodes control the processing carried out and can be configured by supplying a WSDL definition, or by manually configuring properties, or both.

"One-way messages in Gateway mode"
When you configure a SOAP node using WSDL, the WSDL specifies whether a given node operation is one-way or not. However, configuring a node in Gateway mode without WSDL means that this WSDL information is not available. Therefore, SOAP nodes configured as Gateways automatically attempt to detect one-way Operations based on the message content.

Related reference:

"SOAPInput node" on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

"SOAPRequest node" on page 4828
Use the SOAPRequest node to send a SOAP request to the remote Web service.

"SOAPAsyncRequest node" on page 4750
Use the SOAPAsyncRequest node with the SOAPAsyncResponse node to construct a pair of message flows that call a Web service asynchronously.

"Local environment overrides for the SOAPRequest node" on page 4850
You can dynamically override values in the local environment in the same way as setting values in other elements of a message.

One-way messages in Gateway mode:

When you configure a SOAP node using WSDL, the WSDL specifies whether a given node operation is one-way or not. However, configuring a node in Gateway mode without WSDL means that this WSDL information is not available. Therefore, SOAP nodes configured as Gateways automatically attempt to detect one-way Operations based on the message content.

SOAPInput node one-way Operation detection

The SOAPInput node detects one-way messages in different ways depending on the transport used, and whether or not WS-Addressing is configured on the node.

- If the node uses HTTP transport and has WS-Addressing configured, the operation is determined to be one-way if the inbound message uses the special `WSA:None` address for the `WSA:ReplyTo` and `WSA:FaultTo` addresses.
- If the node uses HTTP transport and does not have WS-Addressing configured, no auto detection of one-way messages takes place.
- If the node uses JMS transport and has WS-Addressing configured, the operation is determined to be one-way if either of the following conditions are true:
  - The inbound message uses the special `WSA:None` address (http://www.w3.org/2005/08/addressing/none) for the `WSA:ReplyTo` and `WSA:FaultTo` addresses.
  - There is no JMS ReplyTo destination specified in the inbound message, and the inbound message uses the special `WSA:Anonymous` address (http://www.w3.org/2005/08/addressing/anonymous or http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous) for the `WSA:ReplyTo` and `WSA:FaultTo` addresses.

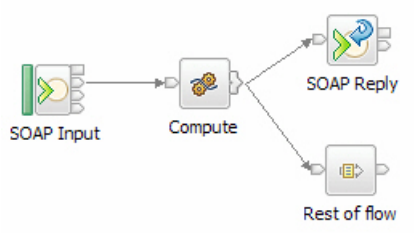
- If the node uses JMS transport and does not have WS-Addressing configured, the operation is determined to be one-way if there is no JMS ReplyTo destination specified in the inbound message.

In Gateway mode, the `SOAP.Context.operationType` field is set to `GATEWAY` if the Operation is determined to be request-response, or `GATEWAY_ONE_WAY` if the Operation is determined to be one-way. If the operation is determined to be one-way, no reply is necessary or allowed. However, if the Operation is not determined to be one-way, the flow is configured with the assumption that the flow sends a reply. Therefore, if the Operation is one-way, you must specify that the Operation is one-way in order to allow the flow to free up resources, and, if HTTP is being used, to send an HTTP 202 acknowledgment back to the originating client. Do this by setting the following field in the local environment before wiring the message to a SOAPReply node:

```
SET OutputLocalEnvironment.Destination.SOAP.Reply.Gateway.OneWay = 'true';
```

This setting instructs the SOAPReply node to complete the Message Exchange Pattern before sending an HTTP 202 acknowledgment, if required, and freeing up its resources.

One approach for using this setting would be in a Gateway flow like this:



In this flow, the Compute node determines if the message is one-way. If the message is one-way, the Compute node sets the local environment one-way setting, and sends a message to the SOAPReply node to complete the Message Exchange Pattern. If the flow is a Gateway flow and the one-way local environment option is set, any message received by the SOAPReply node causes it to ignore the message content and complete the Message Exchange Pattern. The flow can then continue through the other terminal of the Compute node.

It is not an error to send a message to the SOAPReply node with the one-way local environment option set if the message has been automatically determined to be a one-way message.

SOAPRequest node one-way Operation detection

In Gateway mode, the SOAPRequest node automatically detects whether a message is one-way only if WSA is used and the `WSA:ReplyTo` and `WSA:FaultTo` addresses are set to the special `WSA:None` address. To manually instruct the node that the message is one-way, set the following option in the local environment:

```
SET OutputLocalEnvironment.Destination.SOAP.Request.Gateway.OneWay = 'true';
```

For the SOAPRequest node, specifying that a message is one-way indicates to the node that a response is not expected, except for an HTTP 202 acknowledgment if HTTP transport is used. If JMS transport is used, it also allows the message to be

sent under the control of any existing transaction, if the `Transaction mode` is to `Yes` or `Automatic`.

Related concepts:

"SOAP nodes" on page 1609
The SOAP nodes act as points in the flow where Web service processing is configured and applied. Properties on the SOAP nodes control the processing carried out and can be configured by supplying a WSDL definition, or by manually configuring properties, or both.

"Gateway operation mode for SOAP nodes" on page 1645
The SOAPInput, SOAPRequest, and SOAPAsyncRequest nodes have two operation modes: WSDL mode, and Gateway mode.

Related reference:

"SOAPInput node" on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

"SOAPRequest node" on page 4828
Use the SOAPRequest node to send a SOAP request to the remote Web service.

Working with WS-Addressing:

Use the following tasks to help you work with WS-Addressing.

- "How to use WS-Addressing"
- "WS-Addressing with the SOAPInput node" on page 1651
- "WS-Addressing with the SOAPReply node" on page 1653
- "WS-Addressing with the SOAPRequest node" on page 1653
- "WS-Addressing with the SOAPAsyncRequest and SOAPAsyncResponse nodes" on page 1655
- "WS-Addressing information in the local environment" on page 1656

Related concepts:

"What is SOAP?" on page 1604
SOAP is an XML message format used in Web service interactions. SOAP messages are typically sent over HTTP or JMS, but other transport protocols can be used. The use of SOAP in a specific Web service is described by a WSDL definition.

Related tasks:

"Processing Web service messages" on page 1601
Use WebSphere Message Broker nodes and services to connect to other Web services providers and consumers.

How to use WS-Addressing:

An overview of how you use WS-Addressing with WebSphere Message Broker.

Sending a message to an endpoint reference (EPR)

When sending a message to an endpoint reference, the following processes take place:

- The [`destination`] Message Addressing Property (MAP) is completed from the [`address`] property in the EPR.
- [`reference parameters`] are copied to top level SOAP headers from the [`reference parameters`] property of the EPR.
- The [`action`] property is required, but is not populated from the EPR.

In this context, action is an absolute Internationalized Resource Identifier (IRI) that uniquely identifies the semantics implied by this message, and it must be the same as the HTTP SOAPAction if a non-empty SOAPAction is specified.

- The [message id] property must be specified if this message is part of a request-response Message Exchange Pattern (MEP); the message id is generated by default.

When replying with a non-fault message, the following processes take place:

- The EPR to reply to is selected from the [reply endpoint] MAP.
  - If this property contains the special address none, no reply is sent.
  - If this property contains the special address anonymous, the reply is sent on the return channel of the transport on which the request was received. This is the default value in the absence of any other supplied EPR.
  - Otherwise, the reply is sent to the [address] property in the Reply EPR,
- The [message id] property of the inbound message request is placed into the [relationship] property of the response, along with the predefined relationship of the reply part of the Universal Resource Identifier (URI) - which indicates that this message is a reply.

  For further information on the URI see Web Services Addressing URI Specification.
- A new [message id] property is specified for the reply, and this is generated by default.

Related concepts:

"What is SOAP?" on page 1604
SOAP is an XML message format used in Web service interactions. SOAP messages are typically sent over HTTP or JMS, but other transport protocols can be used. The use of SOAP in a specific Web service is described by a WSDL definition.

Related tasks:

"Processing Web service messages" on page 1601
Use WebSphere Message Broker nodes and services to connect to other Web services providers and consumers.

WS-Addressing with the SOAPInput node:

Various options are available when you use WS-Addressing with the SOAPInput node.

The SOAPInput node has a property for processing WS-Addressing information present in the incoming message called Use WS-Addressing.

If you select this property, the WS-Addressing information is processed and the process itself is called engaging WS-Addressing. The default is that WS-Addressing is not engaged.

You can also specify this property in the WSDL, and this property is configurable from the WSDL, automatically by the WebSphere Message Broker Toolkit, when the WSDL is dropped onto the node. The behavior of the node when WS-Addressing is engaged or not engaged is as follows:

Addressing not engaged
: No WS-Addressing processing is performed. If a message is received that contains any WS-Addressing headers they are ignored, and no WS-Addressing processing of any kind is performed, unless they are marked as MustUnderstand.

The inbound WS-Addressing headers in this case are visible in the message when it leaves the SOAPInput node under the Header folder of the SOAP parser in the message tree.

A fault is returned to the client if WS-Addressing headers exist in the incoming message, and they meet both of the following criteria:

- Marked as `MustUnderstand`
- Targeted at the role the SOAPInput node is operating in

Engaging WS-Addressing is how you instruct the node to 'understand' the WS-Addressing headers. In this case the WS-Addressing headers remain in the SOAP Header section of the SOAP parser, and no other SOAP node acts upon them. In all cases, they are treated as a SOAP header with no special meaning assigned to the WS-Addressing headers.

Addressing engaged:

WS-Addressing processing is performed as stated in the WS-Addressing specification. This processing means that messages that contain either submission addressing headers or final addressing headers are accepted.

A fault is returned if both submission addressing headers and final headers are present, and either of the following conditions is met:

- Neither is marked with a role.
- They are both marked with same role and the SOAPInput node is acting in that role.

Assuming that the WS-Addressing headers are valid and the `Place WS-Addressing Headers into LocalEnvironment` check box is selected on the SOAPInput node, all headers (including detectable inbound reference parameters) are removed from the inbound message tree and are placed into the local environment tree under the `SOAP.Input.WSA` folder. Moving the WS-Addressing headers to the local environment indicates that they have been processed by the broker. The headers are removed from the message tree because they have been processed on input; otherwise they would not be valid if the message tree was sent out without further changes. They are stored in the local environment to allow you to inspect them.

Only reference parameters from the final specification are detectable because they have an attribute called `IsReferenceParameter` that allows them to be detected. Submission reference parameter headers do not have this attribute, therefore they are not detectable, and they are not moved into the local environment tree from the message tree.

You can change WS-Addressing reply headers before the SOAPReply node is reached. For more information about changing WS-Addressing information in the local environment, see “WS-Addressing information in the local environment” on page 1656.

Related concepts:

“WS-Addressing” on page 1617
Web Services Addressing (WS-Addressing) is a World Wide Web Consortium (W3C) specification that aids interoperability between Web services by defining a standard way to address Web services and provide addressing information in messages.

Related tasks:

“Example usage of WS-Addressing” on page 1637
Set up a sample message flow by using WS-Addressing, and test the flow.

Related reference:

"SOAPInput node" on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

"SOAPReply node" on page 4819
Use the SOAPReply node to send SOAP messages from the broker to the originating client in response to a message received by a SOAPInput node.

WS-Addressing with the SOAPReply node:

Various options are available when you use WS-Addressing with the SOAPReply node.

The SOAPReply node uses WS-Addressing if WS-Addressing is engaged on the SOAPInput node that is referenced by the reply identifier of the message entering the reply node.

The SOAPReply node uses addressing information in the `Destination.SOAP.Reply.WSA` folder of the local environment to determine where to send the reply and with what Message Addressing Properties (MAPs).

If the `Destination.SOAP.Reply.WSA` does not exist, or is completely empty when inspected by the SOAPReply node, the node uses the default addressing headers that were part of the incoming message. Therefore, you do not have to propagate the local environment in the default case, and addressing still works as expected.

In the case where folders exist beneath the `Reply.WSA` folder, these folders are used to update the output message. Therefore, you can change, add, or remove parts of the default reply information generated by the input node, because any changes that you made to the tree are reflected in the outgoing message by the SOAPReply node. For details about WS-Addressing information in the local environment, see "WS-Addressing information in the local environment" on page 1656.

If WS-Addressing is not engaged, this node does not perform any WS-Addressing processing.

Related concepts:

"WS-Addressing" on page 1617
Web Services Addressing (WS-Addressing) is a World Wide Web Consortium (W3C) specification that aids interoperability between Web services by defining a standard way to address Web services and provide addressing information in messages.

Related tasks:

"Example usage of WS-Addressing" on page 1637
Set up a sample message flow by using WS-Addressing, and test the flow.

Related reference:

"SOAPReply node" on page 4819
Use the SOAPReply node to send SOAP messages from the broker to the originating client in response to a message received by a SOAPInput node.

WS-Addressing with the SOAPRequest node:

Various options are available when you use WS-Addressing with the SOAPRequest node.

The SOAPRequest node has a property called `Use WS-Addressing`, for processing WS-Addressing information that is present in the incoming message.

If you select this property, the WS-Addressing information is processed and the process itself is called `engaging WS-Addressing`. The default is that WS-Addressing is not engaged.

You can also specify this property in the WSDL and this is configurable from the WSDL, automatically by the WebSphere Message Broker Toolkit, when the WSDL is dragged onto the node. The behavior of the node when WS-Addressing is engaged or not is as follows:

Addressing not engaged
The node does not add any WS-Addressing headers to the outgoing message, and does not process any WS-Addressing headers that might be present in the response message that is received by the node.

Addressing engaged:
The node first looks at the `Destination.SOAP.Request.WSA` folder in the local environment. If this folder is empty, the node automatically generates all required WS-Addressing Message Addressing Properties (MAPs) in the outgoing message, by using the following default values:

- `Action`, from the WSDL configuration file. If this is not explicitly specified, this defaults to the value that is defined in the WSDL Binding specification.
- `To`, from the `Web Service URL` node property.
- `ReplyTo`, by using the special `Anonymous` address (assuming that the Operation being used is not a one-way message exchange program, in which case a `ReplyTo` by using the special `None` address is specified).
- `MessageID`, a unique UUID is used.

If the `Destination.SOAP.Request.WSA` folder in the LocalEnvironment is not empty, any user supplied MAPs override the default ones that were listed previously, on a property by property basis.

After the response to the request is received and if the `Place WS-Addressing Headers into LocalEnvironment` check box is selected on the SOAPRequest node, the SOAPRequest node removes all WS-Addressing headers from the response message and places them in the `SOAP.Response.WSA` folder. This folder allows you to query the headers in a similar manner to the way the SOAPInput node deals with the Input WS-Addressing headers.

Related concepts:

"WS-Addressing" on page 1617
Web Services Addressing (WS-Addressing) is a World Wide Web Consortium (W3C) specification that aids interoperability between Web services by defining a standard way to address Web services and provide addressing information in messages.

"WS-Addressing information in the local environment" on page 1656
WS-Addressing header information can be placed in the local environment tree where it is visible to a message flow. WS-Addressing header information is only processed by the SOAP nodes.

Related tasks:

"Example usage of WS-Addressing" on page 1637
Set up a sample message flow by using WS-Addressing, and test the flow.

Related reference:

"SOAPRequest node" on page 4828
Use the SOAPRequest node to send a SOAP request to the remote Web service.

WS-Addressing with the SOAPAsyncRequest and SOAPAsyncResponse nodes:

The remote Web service must understand WS-Addressing to be able to work with SOAPAsyncRequest and SOAPAsyncResponse nodes.

The SOAPAsyncRequest and SOAPAsyncResponse nodes require WS-Addressing; therefore, the remote Web service must understand WS-Addressing to process the WS-Addressing headers that are sent from the SOAPAsyncRequest node, and to allow the response to be sent back to the corresponding SOAPAsyncResponse node, which is specified in the `address` property of the `ReplyTo` Message Addressing Property (MAP).

SOAPAsyncRequest node

The SOAPAsyncRequest node has a property called `Use WS-Addressing` that is read-only and has a default value of `true`, indicating that WS-Addressing is mandatory for this node. This property has the effect of permanently engaging WS-Addressing for this node and cannot be changed by the node, or by the WSDL that is used to configure this node.

The node first looks at the `Destination.SOAP.Request.WSA` folder in the local environment. If this folder is empty, the node automatically generates all required WS-Addressing MAPs in the outgoing message, using the following default values:

- `Action`, from the WSDL configuration file. If this value is not specified explicitly, the default value is defined by the WSDL Binding specification.
- `To`, from the `Web Service URL` node property.
- `ReplyTo`, the address of the corresponding SOAPAsyncResponse node.
- `MessageID`, a unique UUID is used.

If the `Destination.SOAP.Request.WSA` folder in the local environment is not empty, any user-supplied MAPs override the default ones listed previously on a property by property basis.

However, because of the nature of the SOAP asynchronous node pair, you cannot specify the `address` property of the `ReplyTo` Message Exchange Program (MEP), and this property is ignored if specified.

When the main MAPs are generated, the node looks in several places to obtain various pieces of context information to send in a `<wmb:context>` element under the `ReferenceParameters` section of the `ReplyTo` endpoint reference. If these locations exist and are not empty, the following additional information is added to the `<wmb:context>`:

- `Destination.SOAP.Request.UserContext`

  This information is added under a subfolder called `UserContext`.
- `Destination.SOAP.Reply.ReplyIdentifier`

  This information is added under a subfolder called `ReplyID`.

Use the user context to specify an arbitrary amount of data that will be sent with the message from the SOAPAsyncRequest node to the SOAPAsyncResponse node.

By using the user context, you can pass state from one node to the other. Ensure that the amount of data that you send is small because this data is placed in the message.

Use the reply identifier to automatically correlate a SOAPInput node in the flow that contains the SOAPAsyncRequest node, with a SOAPReply node in the flow that contains the SOAPAsyncResponse node.

SOAPAsyncResponse node

After the response to the request is received, the SOAPAsyncResponse node can remove all WS-Addressing headers from the response message and places them in the `SOAP.Response.WSA` folder so that you can query the headers, if you select the node property `Place WS-Addressing headers in local environment`.

If the response message contains a user context that was specified by the SOAPAsyncRequest node, the user context is placed in the `SOAP.Response.UserContext` folder in the local environment.

If the response message contains a reply identifier that was specified by the SOAPAsyncRequest node, the reply identifier is placed in the `Destination.SOAP.Reply.ReplyIdentifier` folder in the local environment.

Related concepts:

"WS-Addressing" on page 1617
Web Services Addressing (WS-Addressing) is a World Wide Web Consortium (W3C) specification that aids interoperability between Web services by defining a standard way to address Web services and provide addressing information in messages.

"WS-Addressing information in the local environment"
WS-Addressing header information can be placed in the local environment tree where it is visible to a message flow. WS-Addressing header information is only processed by the SOAP nodes.

Related tasks:

"Example usage of WS-Addressing" on page 1637
Set up a sample message flow by using WS-Addressing, and test the flow.

Related reference:

"SOAPAsyncRequest node" on page 4750
Use the SOAPAsyncRequest node with the SOAPAsyncResponse node to construct a pair of message flows that call a Web service asynchronously.

"SOAPAsyncResponse node" on page 4777
Use the SOAPAsyncResponse node in conjunction with the SOAPAsyncRequest node to construct a pair of message flows that call a Web service asynchronously.

WS-Addressing information in the local environment:

WS-Addressing header information can be placed in the local environment tree where it is visible to a message flow. WS-Addressing header information is only processed by the SOAP nodes.

Inbound messages

Inbound information is placed in the local environment by the SOAP node only if addressing is engaged on the node and you select the `Place WS-Addressing Headers into LocalEnvironment` property on the SOAPInput, SOAPAsyncResponse, or SOAPRequest nodes.

The following table describes the node specific WS-Addressing information in the local environment tree.

Node	Populates local environment property
SOAPInput	`LocalEnvironment.SOAP.Input.WSA.`*`type`*
SOAPAsyncResponse	`LocalEnvironment.SOAP.Response.WSA.`*`type`*
SOAPRequest	`LocalEnvironment.SOAP.Request.WSA.`*`type`*

Where *type* is the structure of the subsection of the local environment WS-Addressing XML schema. For details about how *type* maps to the WS-Addressing properties defined by the WS-Addressing specification, see the "Local environment property type" on page 1658 section of this topic.

The local environment information for inbound messages is for your information only. If you engage addressing on the node, and select the `Place WS-Addressing Headers into LocalEnvironment` property on the node, WS-Addressing information is available for you to look at and use in your flow. The WS-Addressing properties are placed in the local environment after processing by the node. Note that the WS-Addressing folder and all its children are owned by an XMLNSC parser, therefore you can copy elements directly into any other tree that is owned by an XMLNSC parser. However, be aware that if you copy this folder (or any of its children) to a tree that is not owned by an XMLNSC parser, information in the tree is discarded unless you create an XMLNSC parser in the target tree first. This behavior can occur if you, for example, copy from the `InputLocalEnvironment` tree to the `OutputLocalEnvironment` tree.

Outbound messages

You can place outbound WS-Addressing header information in the local environment; however, this practice is necessary only to override the defaults that are generated by the node automatically . Outbound addressing headers are created only if WS-Addressing is enabled on the node.

The following table describes the node specific WS-Addressing information in the local environment tree that can be used to override the defaults for outbound messages.

Node	Populates local environment property
SOAPReply	`LocalEnvironment.Destination.SOAP.Reply.WSA.`*`type`*
SOAPRequest	`LocalEnvironment.Destination.SOAP.Request.WSA.`*`type`*
SOAPAsyncRequest	`LocalEnvironment.Destination.SOAP.Request.WSA.`*`type`*

Where *type* is the structure of the subsection of the local environment WS-Addressing XML schema. For details about how the *type* maps to the WS-Addressing properties defined by the WS-Addressing specification, see the "Local environment property type" on page 1658 section of this topic.

You can modify local environment information for outbound messages. The SOAPReply, SOAPRequest, and SOAPAsyncRequest nodes generate default local environment settings that you can override. One exception to this table is that any attempt to override the WS-Addressing `ReplyTo` address on the SOAPAsyncRequest node is ignored.

For example, the following code shows how to set WS-Addressing information in the local environment for the SOAPRequest node. The WS-Addressing ReplyTo.Address and FaultTo.Address values should be entered as a single string, without line breaks.

```
 SET OutputRoot = InputRoot;
 SET OutputLocalEnvironment.Destination.SOAP.Request.WSA.To.Address = 'jms:jndi:INPUTQ';
 SET OutputLocalEnvironment.Destination.SOAP.Request.WSA.ReplyTo.Address = 'jms:jndi:RESPONSEQ?jndiConnectionFactoryName=QC
jndiInitialContextFactory=com.sun.jndi.fscontext.RefFSContextFactory&
jndiURL=file://C:/SOAPJNDIBindings';
 SET OutputLocalEnvironment.Destination.SOAP.Request.WSA.From.Address = 'jms:jndi:INPUTQ';
 SET OutputLocalEnvironment.Destination.SOAP.Request.WSA.FaultTo.Address = 'jms:jndi:RESPONSEQ?jndiConnectionFactoryName=QC
jndiInitialContextFactory=com.sun.jndi.fscontext.RefFSContextFactory&
jndiURL=file://C:/SOAPJNDIBindings';
 SET OutputLocalEnvironment.Destination.SOAP.Request.WSA.Action = 'http://WMB_BankImport/NewOperation';
 SET OutputLocalEnvironment.Destination.SOAP.Request.WSA.MessageID = 'test:my:msg:ID:1234578';
```

Local environment property *type*

The local environment property *type* in the preceding tables corresponds to the WS-Addressing part of the local environment XML schema. The following table shows the corresponding message addressing properties (MAPs) of the WS-Addressing local environment schema for all nodes.

Element	Corresponds to abstract WS-Addressing MAP name
To	[destination endpoint]
From	[source endpoint]
ReplyTo	[reply endpoint]
FaultTo	[fault endpoint]
Action	[action]
MessageId	[message id]
RelatesTo	[relationship]
ReferenceParameters	[reference parameters]
Version	This element does not correspond to a MAP, but it is used to identify the version of WS-Addressing. The two main versions of WS-Addressing are Submission and Final. The default version that is used by all nodes is Final. Therefore, for outbound messages, set this element only if you want the version to be Submission. For incoming messages, this element is populated automatically with the version of the WS-Addressing headers that the inbound message used.

For more details about the message addressing properties defined by the WS-Addressing specification, see “WS-Addressing” on page 1617.

For outbound WS-Addressing, you can set an additional local environment property.

Element	Description
AddMustUnderstandAttribute	This element places the SOAP `mustUnderstand` attribute on each WS-Addressing header before the message is sent.

Related concepts:

"WS-Addressing" on page 1617
Web Services Addressing (WS-Addressing) is a World Wide Web Consortium (W3C) specification that aids interoperability between Web services by defining a standard way to address Web services and provide addressing information in messages.

Related reference:

"SOAPInput node" on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

"SOAPReply node" on page 4819
Use the SOAPReply node to send SOAP messages from the broker to the originating client in response to a message received by a SOAPInput node.

"SOAPRequest node" on page 4828
Use the SOAPRequest node to send a SOAP request to the remote Web service.

"SOAPAsyncRequest node" on page 4750
Use the SOAPAsyncRequest node with the SOAPAsyncResponse node to construct a pair of message flows that call a Web service asynchronously.

"SOAPAsyncResponse node" on page 4777
Use the SOAPAsyncResponse node in conjunction with the SOAPAsyncRequest node to construct a pair of message flows that call a Web service asynchronously.

"Example use of WS-Addressing information in the local environment"
This example shows the setting of reference parameters in the local environment along with the corresponding messages as they appear on the wire.

Example use of WS-Addressing information in the local environment:

This example shows the setting of reference parameters in the local environment along with the corresponding messages as they appear on the wire.

In this example the Web service exposes a simple ping operation.

ESQL to add reference parameters

The following ESQL example shows how to specify addressing headers in the local environment.

```
DECLARE Example_ns NAMESPACE 'http://ibm.namespace';

SET OutputLocalEnvironment.Destination.SOAP.Request.WSA.ReplyTo.ReferenceParameters.Parameter1 = 'Message Broker';
SET OutputLocalEnvironment.Destination.SOAP.Request.WSA.ReplyTo.ReferenceParameters.Example_ns:Parameter2.
 (SOAP.NamespaceDecl)xmlns:Example_ns = 'http://ibm.namespace';
SET OutputLocalEnvironment.Destination.SOAP.Request.WSA.ReplyTo.ReferenceParameters.Example_ns:Parameter2 = 'Ping';

SET OutputLocalEnvironment.Destination.SOAP.Request.WSA.FaultTo.ReferenceParameters.Parameter1 = 'Ping';
SET OutputLocalEnvironment.Destination.SOAP.Request.WSA.FaultTo.ReferenceParameters.Example_ns:Parameter2.
 (SOAP.NamespaceDecl)xmlns:Example_ns = 'http://ibm.namespace';
SET OutputLocalEnvironment.Destination.SOAP.Request.WSA.FaultTo.ReferenceParameters.gns:Parameter2 = 'FAULT';
```

Request message

The following example is of an outgoing SOAP envelope in a message from a SOAPRequest node with `ReplyTo` and `FaultTo` reference parameters generated after using the above ESQL. It also shows the other message addressing properties

(MAPs) that are not set in the local environment, but are generated automatically by the node as a result of engaging WS-Addressing.

```
<NS1:Envelope xmlns:NS1="http://www.w3.org/2003/05/soap-envelope" xmlns:wsa="http://www.w3.org/2005/08/addressing">
  <NS1:Header>
    <wsa:To>http://localhost:7801/Service</wsa:To>
    <wsa:ReplyTo>
      <wsa:Address>http://www.w3.org/2005/08/addressing/anonymous</wsa:Address>
      <wsa:ReferenceParameters>
        <Example_ns:Parameter2 xmlns:Example_ns="http://ibm.namespace">Ping</Example_ns:Parameter2>
        <Parameter1>Message Broker</Parameter1>
      </wsa:ReferenceParameters>
    </wsa:ReplyTo>
    <wsa:FaultTo>
      <wsa:Address>http://www.w3.org/2005/08/addressing/anonymous</wsa:Address>
      <wsa:ReferenceParameters>
        <Example_ns:Parameter2 xmlns:Example_ns="http://ibm.namespace">FAULT</Example_ns:Parameter2>
        <Parameter1>Ping</Parameter1>
      </wsa:ReferenceParameters>
    </wsa:FaultTo>
    <wsa:MessageID>urn:uuid:020C911C16EB130A8F1204119836321</wsa:MessageID>
    <wsa:Action>http://ibm.com/Service/Ping</wsa:Action>
  </NS1:Header>
  <NS1:Body>
    <NS2:Ping xmlns:NS2="http://ibm.com"></NS2:Ping>
  </NS1:Body>
</NS1:Envelope>
```

In the above example, reference parameters are set for the `ReplyTo` and `FaultTo` endpoint references (EPRs). If this message is sent to a SOAPInput node with WS-Addressing engaged, these `ReferenceParameters` are placed in the local environment of the flow that contains the SOAPInput node for use by the flow if the `Place WS-Addressing Headers into LocalEnvironment` property is selected. This option changes only what is placed in the local environment; it does not change the contents of the response message.

Response message

The following SOAP envelope is a response to the outgoing preceding message, as sent by a SOAPReply node. This example shows the MAP processing that happens automatically by the SOAPReply node. In this example, the `FaultTo` reference parameters are not present because the reply is not a SOAP fault. This response also shows where the reference parameters that belonged to the `ReplyTo` EPR appear in the response message.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:wsa="http://www.w3.org/2005/08/
 addressing">
  <soapenv:Header>
    <wsa:To>http://www.w3.org/2005/08/addressing/anonymous</wsa:To>
    <Example_ns:Parameter2 wsa:IsReferenceParameter="true" xmlns:Example_ns="http://ibm.namespace">Ping</Example_ns:
 Parameter2>
    <Parameter1 wsa:IsReferenceParameter="true">Message Broker</Parameter1>
    <wsa:ReplyTo>
      <wsa:Address>http://www.w3.org/2005/08/addressing/anonymous</wsa:Address>
    </wsa:ReplyTo>
    <wsa:Action>http://ibm.com/Service/PingResponse</wsa:Action>
    <wsa:RelatesTo RelationshipType="http://www.w3.org/2005/08/addressing/reply">urn:uuid:020C911C16EB130A8F1204119836321
 </wsa:RelatesTo>
  </soapenv:Header>
  <soapenv:Body>
    <NS1:PingResponse xmlns:NS1="http://ibm.com">
      <NS1:PingResult>Ping</NS1:PingResult>
    </NS1:PingResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Related concepts:

"WS-Addressing information in the local environment" on page 1656
WS-Addressing header information can be placed in the local environment tree where it is visible to a message flow. WS-Addressing header information is only processed by the SOAP nodes.

Working with WSDL:

Use the following tasks to help you work with WSDL.

- "WSDL validation"
- "Using WSDL to configure message flows" on page 1664
- "WSDL URI formats for JMS" on page 1668
- "WSDL styles" on page 1670
- "Working with rpc-encoded SOAP messages" on page 1671

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"WSDL validation"
The WS-I Validator can be used to check your WSDL definitions against the Basic Profile.

"WSDL Version 1.1" on page 6699
Web Services Description Language (WSDL) is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information.

WSDL validation:

The WS-I Validator can be used to check your WSDL definitions against the Basic Profile.

For more information about the WS-I Basic Profile refer to the WS-I, and in particular the WS-I Basic Profile document:

- Web Services Interoperability Organization (WS-I)
- WS-I deliverables index

You can use the WS-I Validator to check your WSDL definitions against the Basic Profile; see "WS-I Basic Profile Version 1.1" on page 6700.

You can run the validator in either of the following ways:

- Manually against a specific `.wsdl` resource in the workbench.

  This option enables you to investigate and fix WS-I compliance problems; all validation issues are displayed as task list errors and warnings.
- Automatically, when WSDL is imported or generated.

  WSDL can be imported using the WSDL Quick Start wizard, the WSDL Importer wizard, or the **mqsicreatemsgdefsfromwsdl** command.

  WSDL can be generated from a message set by using the WSDL Generator wizard.

You can control the behavior of the validator using **Preferences > Web services > Profile Compliance and Validation**. The significant settings are:

- WS-I AP compliance level (WS-I Attachments Profile 1.0)

- WS-I SSBP compliance level (WS-I Simple SOAP Binding Profile 1.0)

You can select one of the following values:

Suggest
Run the validator with errors treated as unrecoverable, but warnings only notified. This is the default setting.

Require
Run the validator with errors and warnings treated as unrecoverable.

Ignore Do no run the validator.

The AP selection applies automatically to the SSBP field, therefore Ignore is not explicitly selectable unless the AP selection is set to Ignore.

The following terms refer to the three broad categories of WSDL definition:

- `document-literal` means the combination `style="document"` and `use="literal"`
- `rpc-literal` means the combination `style="rpc"` and `use="literal"`
- `rpc-encoded` means the combination `style="rpc"` and `use="encoded"`

The following are typical validation problems using the preceding terminology:

Your WSDL is rpc-encoded
WSDL with `use="encoded"` is not WS-I compliant and can lead to operational problems because products of different vendors can make different assumptions about the expected SOAP payload.

WS-I: (BP2406) The use attribute of a `soapbind:body`, `soapbind:fault`, `soapbind:header`, and `soapbind:headerfault` does not have the value of "literal".

Your WSDL is document-literal, but one or more WSDL part definitions refer to XML Schema types.
In `document-literal` WSDL, the SOAP body payload is the XML Schema element that is referred to by the appropriate WSDL part.

If a type is specified instead of an element, the SOAP payload is potentially ambiguous (the payload name is not defined) and interoperability problems are likely.

WS-I: (BP2012) A `document-literal` binding contains `soapbind:body` elements that refer to message part elements that do not have the element attribute.

Your WSDL is rpc-literal or rpc-encoded, but one or more WSDL part definitions refer to XML Schema elements.
In rpc-style WSDL, the SOAP body payload is the WSDL operation name, and its children are the WSDL parts that are specified for that operation.

If an element is specified instead of a type, the SOAP message payload is potentially ambiguous (the payload name might be the WSDL part name or the XML Schema element name), and interoperability problems are likely.

WS-I: (BP2013) An rpc-literal binding contains `soapbind:body` elements that refer to message part elements that do not have the type attribute.

Your WSDL includes SOAP header, headerfault or fault definitions that refer to XML Schema types.

In rpc-style WSDL, the SOAP body is correctly defined through XML Schema types as described above.

SOAP headers and faults, however, do not correspond to an rpc function call in the same way as the body.

In particular, there is no concept of 'parameters' to a header or fault, and a header or fault must always be defined in terms of XML Schema elements to avoid potential ambiguity. Effectively, header and fault definitions in WSDL are always `document-literal`.

WS-I: (BP2113) The `soapbind:header`, `soapbind:headerfault`, or `soapbind:fault` elements refer to `wsd:part` elements that are not defined using only the "element" attribute.

Your WSDL is rpc-literal or rpc-encoded, but no namespace was specified for an operation.

In rpc-style WSDL, the SOAP message payload is the WSDL operation name, qualified by a namespace that is specified as part of the WSDL binding.

If no namespace is specified then the SOAP message payload is potentially ambiguous (the payload name might be in no namespace, or might default to use a different namespace, such as the target namespace of the WSDL definition) and interoperability problems are likely.

WS-I: (BP2020) An rpc-literal binding contains `soapbind:body` elements that either do not have a namespace attribute, or have a namespace attribute value that is not an absolute URI.

Your WSDL includes a SOAP/JMS binding.

If your WSDL uses a SOAP/JMS transport URI it is not WS-I compliant. An error is shown if strict WS-I validation is enabled. To disable strict WS-I validation, click **Window > Preferences > Broker Development > Message Sets > Validation** and select the **WS-I BP 1.1: Allow SOAP/JMS as transport URI**. By default strict WS-I validation is disabled.

Web service interoperability is improved if you implement the following actions:

- Use document-style WSDL whenever possible.
- Use literal encoding, if rpc-style WSDL is necessary.
- Ensure that the WSDL operation definitions are qualified by a valid namespace attribute, if rpc-encoded WSDL must be used.

Related concepts:

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

"Message definition files" on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

"Namespaces in the message model" on page 1201
Use namespaces to qualify message model object names.

"Importing WSDL files to create message definitions" on page 1267
Import WSDL files by using the **New Message Definition File From WSDL file** wizard, the **Start from WSDL and/or XSD files** Quick Start wizard, or the **`mqsicreatemsgdefsfromwsdl`** command.

"WSDL Version 1.1" on page 6699
Web Services Description Language (WSDL) is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information.

Related tasks:

"Working with a message set" on page 2840
Complete a variety of tasks that are involved in working with a message set.

"Creating a message definition file" on page 2865
Creating an empty message definition file to contain your message model objects.

"Importing from WSDL" on page 2946
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to create a new message definition from WSDL.

"Importing WSDL definitions from the command line" on page 2948
WSDL definitions can be imported using the (**mqsicreatemsgdefsfromwsdl**) command.

"Generating a WSDL definition from a message set" on page 2968
To ensure the highest interoperability of your Web services, use the document style of WSDL whenever possible. If rpc style WSDL is necessary, use literal encoding.

Related reference:

"What is WSDL?" on page 1615
WSDL is an XML notation for describing a web service. A WSDL definition tells a client how to compose a web service request and describes the interface that is provided by the web service provider.

"**mqsicreatemsgdefsfromwsdl** command" on page 3712
Use the **mqsicreatemsgdefsfromwsdl** command to import a single WSDL definition.

"Importing from WSDL: generated objects and restrictions" on page 6355
Several objects are generated when you import from WSDL but restrictions might apply.

Using WSDL to configure message flows:

You can use WSDL to configure message flows.

Message flows that work with Web services typically use the SOAP nodes. For details about the SOAP nodes, see "WebSphere Message Broker and Web services" on page 1602.

In WSDL mode, the SOAP nodes are configured by using a WSDL file that was previously imported or generated in a message set, and is displayed under **Deployable WSDL** in the workbench. You can drag the WSDL file onto a SOAP node, or specify it by using the `WSDL file name` property on the node.

You can change the operation mode of the SOAP nodes so that they act in gateway mode. In gateway mode, a WSDL is not required to configure the nodes because they handle generic request/response and one-way SOAP messages that are not tied to a specific WSDL. For more details, see "Gateway operation mode for SOAP nodes" on page 1645.

When you drag a WSDL file onto a SOAP node, the node properties are configured from the properties in the WSDL `address` URI. The transport properties on the SOAP node are populated according to the first binding imported from the WSDL file. Therefore, if the first imported binding describes a JMS transport, the JMS Transport properties are populated; if the first imported binding describes an

HTTP transport, the HTTP Transport properties are populated. If you select another imported binding, the transport properties are populated accordingly. The `portType` appears differently depending on the transport selected.

The WSDL `address` element URI can exist in two different formats, W3C format, or IBM (deprecated) format. The format of the WSDL URI affects the names of the WSDL properties that the parser looks for to populate the SOAP node properties. For example, the `JNDI context parameters` table is not populated when you import an IBM-style WSDL because it does not support these properties in the WSDL address URI. The table is populated only if JNDI context parameters are present in a W3C-style WSDL. For details, see "WSDL URI formats for JMS" on page 1668.

If you supply a service definition, endpoint properties are set automatically, but you can also set or override these properties manually.

Optionally, WSDL definitions can be split into multiple files. The typical arrangement is that a top-level service definition file imports a binding file, the binding file imports an interface file, and this interface file imports or includes schema definition files.

A WSDL `portType` (the logical WSDL interface) is not sufficient on its own to configure a SOAP node; a specific binding is required so that the SOAP payload is well-defined at run time.

A binding defines a use, which can be `document` (the default) or `rpc`. If the use is `document`, the SOAP payload is described by an XML Schema element in the WSDL. If the use is `rpc`, the SOAP payload is the WSDL operation name in a specified namespace.

To create your own message flow, configure the nodes as described. However, you can create a new skeleton message flow by dragging a WSDL definition onto a blank canvas in the Message Flow editor, and selecting a specific WSDL binding. You can also choose the type of flow (service provider or consumer) and the operations to be handled by the flow. The key nodes and properties in the generated message flow are configured, but you must complete the configuration and add any other nodes that you require before deploying the flow. For details about configuring a new Web service by using the wizard, see "Configure New Web Service Usage wizard" on page 6392.

Configuring the SOAP nodes

The following nodes are configured explicitly by WSDL:

- "SOAPInput node" on page 4795
- "SOAPRequest node" on page 4828
- "SOAPAsyncRequest node" on page 4750

The following nodes are configured implicitly by WSDL, because they inherit the WSDL configuration of the node with which they are paired:

- "SOAPReply node" on page 4819
- "SOAPAsyncResponse node" on page 4777

A SOAPReply node is always used with a SOAPInput node. For details of Web service scenarios, see "Web services scenarios" on page 1620.

A SOAPAsyncResponse node is always used with a SOAPAsyncRequest node, associated by the `Unique Identifier` property. For SOAP node usage patterns, see "Web services scenarios" on page 1620.

Related concepts:

"SOAP nodes" on page 1609
The SOAP nodes act as points in the flow where Web service processing is configured and applied. Properties on the SOAP nodes control the processing carried out and can be configured by supplying a WSDL definition, or by manually configuring properties, or both.

"Importing WSDL files to create message definitions" on page 1267
Import WSDL files by using the **New Message Definition File From WSDL file** wizard, the **Start from WSDL and/or XSD files** Quick Start wizard, or the **`mqsicreatemsgdefsfromwsdl`** command.

"Relationship of WSDL to Message Model" on page 1269
If a broker is to communicate with an existing Web service, it typically needs to send and receive SOAP messages. To take this approach, use the MRM domain. You must ensure that the broker message model and the WSDL definition used by the Web service describe the same messages. In general, you can achieve this result by importing the WSDL for the existing Web service by using the broker tooling. Currently only WSDL version 1.1 is supported.

Related tasks:

"Generating a WSDL definition from a message set" on page 2968
To ensure the highest interoperability of your Web services, use the document style of WSDL whenever possible. If rpc style WSDL is necessary, use literal encoding.

"Migrating a flow supporting ?wsdl queries" on page 220
If you want WSDL and XML Schema information to be made available for existing SOAPInput-SOAPReply flows that implement web services, you must explicitly set the SOAPInput node property `Enable support for ?wsdl` and redeploy the flow.

"Creating an application by using the Configure New Web Service Usage wizard" on page 1417
Use these instructions to generate a message flow by using the Configure New Web Service Usage wizard.

Related reference:

"Configure New Web Service Usage wizard" on page 6392
This provides additional reference information in relation to the Configure New Web Service Usage wizard.

"Generate WSDL wizard" on page 6382
The Generate WSDL wizard creates a WSDL definition from a message set.

Querying WSDL with ?wsdl:

You can interrogate web services using ?wsdl.

A web service client can send an HTTP GET request with a `?wsdl` query string to a WebSphere Message Broker web service, and receive a representation of the WSDL that was used to configure the input node that provides the endpoint for the service. You can do this only for input nodes that use HTTP and not JMS transport. The protocol of the HTTP GET request must match the protocol of the flow; therefore if the flow uses SSL, the HTTP GET request must begin `https://`.

The client starts by sending a simple `?wsdl` query and retrieves the complete WSDL definition by following a chain of referenced imports or includes. For example, if the web service endpoint is `http://localhost:7800/test1`, the initial client request is:

```
GET http://localhost:7800/test1?wsdl
```

This request returns the top-level WSDL service definition, which might include imports for further sections of the WSDL definition. For example, if the returned WSDL has a line:

```
<wsdl:import ... location="http://localhost:7800/test1?wsdl=wsdl0"/>
```

then the client sends a corresponding request to retrieve that section of the WSDL:

```
GET http://localhost:7800/test1?wsdl=wsdl0
```

One or more WSDL sections can also have imports for XML Schema data, for example:

```
<xsd:import ... schemaLocation="http://localhost:7800/test1?xsd=xsd0"/>
```

The client again sends a corresponding request to retrieve that data:

```
GET http://localhost:7800/test1?xsd=xsd0
```

Only semantically correct references can be followed:

- `<wsdl:import>` elements that are immediate children of `wsdl:definition` elements
- `<xsd:import>` and `<xsd:include>` elements that are immediate children of `xsd:schema` elements

where `wsdl` is a shorthand (namespace prefix) for `http://schemas.xmlsoap.org/wsdl/`, and `xsd` is a shorthand for `http://www.w3.org/2001/XMLSchema`. A request made to a URI from an element which superficially looks like an `<import>` or `<include>`, for example, an element in a comment, results in a SOAP Fault being returned. Only the simple `?wsdl` query string, and subsequent queries that exactly match those queries specified in semantically valid imports and includes, result in data being returned.

The WSDL definition returned is logically equivalent to the deployable WSDL in the toolkit, with inline schemas externalized. It might not be physically identical to the original imported WSDL definition. Although a SOAPInput is configured with a specific WSDL binding, the WSDL returned also includes other bindings that are not used by the flow if these were part of the original WSDL definition that was imported.

Related concepts:

"SOAP nodes" on page 1609
The SOAP nodes act as points in the flow where Web service processing is configured and applied. Properties on the SOAP nodes control the processing carried out and can be configured by supplying a WSDL definition, or by manually configuring properties, or both.

"Importing WSDL files to create message definitions" on page 1267
Import WSDL files by using the **New Message Definition File From WSDL file** wizard, the **Start from WSDL and/or XSD files** Quick Start wizard, or the **mqsicreatemsgdefsfromwsdl** command.

"Relationship of WSDL to Message Model" on page 1269
If a broker is to communicate with an existing Web service, it typically needs to send and receive SOAP messages. To take this approach, use the MRM domain. You must ensure that the broker message model and the WSDL definition used by the Web service describe the same messages. In general, you can achieve this result by importing the WSDL for the existing Web service by using the broker tooling. Currently only WSDL version 1.1 is supported.

Related tasks:

"Generating a WSDL definition from a message set" on page 2968
To ensure the highest interoperability of your Web services, use the document style of WSDL whenever possible. If rpc style WSDL is necessary, use literal encoding.

"Migrating a flow supporting ?wsdl queries" on page 220
If you want WSDL and XML Schema information to be made available for existing SOAPInput-SOAPReply flows that implement web services, you must explicitly set the SOAPInput node property `Enable support for ?wsdl` and redeploy the flow.

"Creating an application by using the Configure New Web Service Usage wizard" on page 1417
Use these instructions to generate a message flow by using the Configure New Web Service Usage wizard.

Related reference:

"Configure New Web Service Usage wizard" on page 6392
This provides additional reference information in relation to the Configure New Web Service Usage wizard.

"Generate WSDL wizard" on page 6382
The Generate WSDL wizard creates a WSDL definition from a message set.

WSDL URI formats for JMS:

You must use WSDL to configure SOAP nodes. When using WSDL with a JMS transport, different URI formats can exist in the `address` element in the WSDL, which affect how properties are parsed and applied to the configured nodes.

Two different URI formats can exist in the WSDL `address` element. Several node properties are initially set from properties in the imported WSDL, which is parsed according to which type of URI is found in the WSDL element. The first type is the W3C SOAP JMS specification format. For example:

```
<soap:address location="jms:jndi:REPLYTOQ2?jndiConnectionFactoryName=QCF&amp;
 jndiInitialContextFactory=com.sun.jndi.fscontext.RefFSContextFactory&amp;
 jndiURL=file:/C:/mqsi6/webservices/SOAP/JMS/JNDI&amp;
 targetService=SOAPJMSGenMessageSetSOAP_JMS_Service&amp;
 timeToLive=30000"
/>
```

The second URI format for the `address` element is a proprietary IBM format which is currently deprecated. For example:

```
<soap:address location="jms:/queue?destination=jms/RequestQ&amp;
 connectionFactory=jms/WMBQCF&amp;
 targetService=SOAPJMSGenMessageSetSOAP_JMS_Service&amp;
 initialContextFactory=com.sun.jndi.fscontext.RefFSContextFactory&amp;
 jndiProviderURL=file:/C:/mqsi6/webservices/SOAP/JMS/JNDI"
/>
```

There are several differences between these URI formats. WebSphere Message Broker accepts both URI formats. Different WSDL properties are used to set the SOAP node properties depending on which URI format is used in the WSDL address element.

The following table shows how WSDL properties are parsed into SOAPInput node properties. The columns headed "W3C names in URI" and "W3C allowed values" indicate the property names that the parser looks for when a W3C-style URI is found, and the allowed values for those properties. The columns headed "IBM names in URI" and "IBM allowed values" indicate the property names that the parser looks for when an IBM-style URI is found in the WSDL, and the allowed values for those properties. Where more than one property name is shown in a table cell, the node property is set to the value of the first of those property names found in the WSDL `address` element. Any properties found in the WSDL `address` element that are not parsed into node properties are discarded.

SOAPInput node property name	W3C SOAP/JMS specification names	W3C names in URI	W3C allowed values	IBM names in URI	IBM allowed values
Source	`soapjms:destinationName`	`jms=dest` (in URI)	`<string>`	`destination`	`<string>`
Connection factory name	`soapjms:jndiConnectionFactoryName`	`jndiConnectionFactoryName`	`<string>`	`connectionFactory`	`<string>`
Initial context factory	`soapjms:jndiInitialContextFactory`	`jndiInitialContextFactory`	`<string>`	`initialContextFactory`	`<string>`
JNDI URL bindings location	`soapjms:jndiURL`	`jndiURL`	`<URL>`	`jndiProviderURL`	`<URL>`
JNDI parameters	`soapjms:jndiContextParameter`	`parameter=value`	`<string for name and value>`	N/A	`<string>`
Delivery mode	`soapjms:deliveryMode`	`deliveryMode`	`NON_PERSISTENT` `PERSISTENT`[1]	`deliveryMode` `persistence`	`<int 1 | 2>`
Message priority	`soapjms:priority`	`priority`	`<int 0-9>`	`priority` `Priority`	`<int 0-9>`
Target service	`soapjms:targetService`	`targetService`	`<string>`	`targetService`	`<string>`

Notes:

1. WebSphere Message Broker accepts the values 1 and 2 when parsing a W3C-style URI for compatibility reasons, but the W3C specification allows only the string values `NON_PERSISTENT` and `PERSISTENT` for this property.

The following table shows how WSDL properties are parsed into SOAPRequest andSOAPAsyncRequest node properties. The columns headed "W3C names in URI" and "W3C allowed values" indicate the property names that the parser looks for when a W3C-style URI is found, and the allowed values for those properties. The columns headed "IBM names in URI" and "IBM allowed values" indicate the property names that the parser looks for when an IBM-style URI is found in the WSDL, and the allowed values for those properties. Where more than one property name is shown in a table cell, the node property is set to the value of the first of those property names found in the WSDL address element. Any properties found in the WSDL `address` element that are not parsed into node properties are populated in the `User Parameters` table.

SOAPRequest or SOAPAsyncRequest node property name	W3C SOAP/JMS specification names	W3C names in URI	W3C allowed values	IBM names in URI	IBM allowed values
Destination	`soapjms:destinationName`	`jms=dest` (in URI)	`<string>`	`destination`	`<string>`

SOAPRequest or SOAPAsyncRequest node property name	W3C SOAP/JMS specification names	W3C names in URI	W3C allowed values	IBM names in URI	IBM allowed values
Connection factory name	soapjms:jndiConnectionFactoryName	jndiConnectionFactoryName	<string>	connectionFactory	<string>
Initial context factory	soapjms:jndiInitialContextFactory	jndiInitialContextFactory	<string>	initialContextFactory	<string>
JNDI URL bindings location	soapjms:jndiURL	jndiURL	<URL>	jndiProviderURL	<URL>
JNDI parameters	soapjms:jndiContextParameter	Parameter=value	<string for name and value>	N/A	<string>
Delivery mode	soapjms:deliveryMode	deliveryMode	NON_PERSISTENT PERSISTENT[1]	deliveryMode persistence	<int 1 | 2>
Message expiration	soapjms:timeToLive	timeToLive	<int>	timeToLive	<int>
Message priority	soapjms:priority	priority	<int 0-9>	priority Priority	<int 0-9>
Reply to destination	soapjms:replyToName	replyToName	<string>	replyToName replyTo replyToDestination replyDestination	<string>
Target service	soapjms:targetService	targetService	<string>	targetService	<string>
User parameters	UserProperties	<any other property name>	<string>	<any other property name>	<string>

Notes:

1. WebSphere Message Broker accepts the values 1 and 2 when parsing a W3C-style URI for compatibility reasons, but the W3C specification allows only the string values NON_PERSISTENT and PERSISTENT for this property.

Related concepts:

"Using WSDL to configure message flows" on page 1664
You can use WSDL to configure message flows.

Related reference:

"SOAPInput node" on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

"SOAPRequest node" on page 4828
Use the SOAPRequest node to send a SOAP request to the remote Web service.

WSDL styles:

The SOAP nodes are configured using a specific WSDL binding that has a style of either document (the default) or rpc. All operations defined in a specific WSDL binding are usually defined with the same use, which can be either literal (the default) or encoded.

The following terms are used to describe the three general types of WSDL bindings:

1. `document-literal (style="document", use="literal")`
2. `rpc-literal (style="rpc", use="literal")`
3. `rpc-encoded (style="rpc", use="encoded")`

The shape of the runtime SOAP message defined by the WSDL depends on the binding type, as shown in the following table:

WSDL binding type	Description
`document-literal`	The SOAP payload is described by XML schema. The `wrapped document-literal` convention constructs the XML schema so that the first child of the SOAP Body matches the operation name.
`rpc-literal`	The SOAP payload is described by the WSDL (operation and part name) and then by XML schema.
`rpc-encoded`	The SOAP payload has the same general shape as `rpc-literal`, but can carry SOAP encoding annotations designed to give the receiver additional information about the message that is sent. The annotations are slightly different between SOAP 1.1 and SOAP 1.2.

All three WSDL styles are supported by WebSphere Message Broker. The best style for a new service is `document-literal`, and it is also the least likely to cause interoperability problems. You can use the `wrapped document-literal` convention to explicitly associate the SOAP payload with the operation.

Both `document-literal` and `rpc-literal` are WS-I compliant. The `rpc-encoded` style is not WS-I compliant and can cause interoperability problems if the web service client and server use different technologies. Some common issues encountered with WSDL styles are described in "WSDL validation" on page 1661.

It is simple to create and parse SOAP messages described by `document-literal` or `rpc-literal` WSDL, because the payload is standard XML described by the message set created from the WSDL.

For more information, see "Working with rpc-encoded SOAP messages."

Related reference:

"What is WSDL?" on page 1615
WSDL is an XML notation for describing a web service. A WSDL definition tells a client how to compose a web service request and describes the interface that is provided by the web service provider.

Working with rpc-encoded SOAP messages:

The SOAP nodes are configured using a specific WSDL binding that has a style of either `document` (the default) or `rpc`. All the operations defined within a particular WSDL binding are usually defined with the same use, which can be either `literal` (the default) or `encoded`.

SOAP messages that are `rpc-encoded` can carry SOAP encoding annotations that are designed to give the receiver additional information about the message being sent. The following four types of annotations are common:

1. xsi:type

2. encodingStyle
3. Arrays
4. Multi-reference

Arrays and multi-reference elements can cause interoperability problems, and require specific intervention by the message flow developer, as follows:

- If the message flow builds outbound messages incorporating SOAP arrays, use ESQL to add any required attributes, as follows:
  - `arrayType` for SOAP 1.1
  - `arraySize` and `itemType` for SOAP 1.2
- If the message flow receives messages using SOAP arrays, disable validation.
- If the message flow receives messages using multi-reference encoding, disable validation and, if necessary, navigate the resulting logical tree using the `href` (or `ref`) and `id` attributes.

xsi:type

An `xsi:type` attribute can be added to an element to specify its type. For example:

```
<data xsi:type="xsd:string">text</data>
```

where the namespace prefix `xsi` is defined as follows:

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

For a web service built from WSDL and XML schema, the type information is already available and the `xsi:type` is redundant.

Generally, when building an outbound message, you do not add `xsi:type` information. The following ESQL example shows how to add `xsi:type` information if it is required:

```
DECLARE xsd NAMESPACE 'http://www.w3.org/2001/XMLSchema';
DECLARE xsi NAMESPACE 'http://www.w3.org/2001/XMLSchema-instance';

SET OutputRoot.SOAP.Context.Namespace.(SOAP.NamespaceDecl)xmlns:xsd = xsd;
SET OutputRoot.SOAP.Context.Namespace.(SOAP.NamespaceDecl)xmlns:xsi = xsi;

SET OutputRoot.SOAP.Body.rpc:op1.part1.data.(SOAP.Attribute)xsi:type = 'xsd:string';
```

When parsing an inbound message, any `xsi:type` information is added to the logical tree in the same way as other attributes, as follows:

```
(0x03000000:PCDataField):data = 'text' (CHARACTER)
(
  (0x03000100:Attribute)http://www.w3.org/2001/XMLSchema-instance:type = 'xsd:string' (CHARACTER)
)
```

encodingStyle

An `encodingStyle` attribute can be added to an element to specify the SOAP encoding style used. For example:

```
<tns:op1 soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
```

where the namespace prefix `soapenv` is defined as the namespace of your SOAP Envelope, as follows:

```
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" (SOAP 1.1)
xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope"   (SOAP 1.2)
```

The value itself is a list of URIs, but the values in common use are as follows:

```
"http://schemas.xmlsoap.org/soap/encoding/" (SOAP 1.1)
"http://www.w3.org/2003/05/soap-encoding"   (SOAP 1.2)
```

In SOAP 1.1, the `encodingStyle` attribute can be added to any element. In SOAP 1.2 the `encodingStyle` attribute can be added only to children of Body, Header and Detail.

Generally, you build an outbound message, you do not add the `encodingStyle` attribute. The following ESQL example shows how to add `encodingStyle` information, if it is required:

```
DECLARE soapenv NAMESPACE 'http://schemas.xmlsoap.org/soap/envelope/';
DECLARE soapenc NAMESPACE 'http://schemas.xmlsoap.org/soap/encoding/';

SET OutputRoot.SOAP.Body.tns:op1.(SOAP.Attribute)soapenv:encodingStyle = soapenc;
```

When parsing an inbound message, any `encodingStyle` attributes are added to the logical tree in the same way as other attributes, as follows:

```
(0x03000100:Attribute)http://schemas.xmlsoap.org/soap/envelope/:encodingStyle =
     'http://schemas.xmlsoap.org/soap/encoding/' (CHARACTER)
```

Arrays

A SOAP array is an element containing a sequence of child elements of the same type. In the following XML schema example, there is a type called *data* with two elements: a simple string and an array. In the schema, the field called *array* has an unspecified number of children of type string. The name of those child elements is not specified.

```
<xsd:complexType name="ArrayOfString">
  <xsd:complexContent mixed="false">
    <xsd:restriction base="soapenc:Array">
 <xsd:attribute wsdl:arrayType="xsd:string[]" ref="soapenc:arrayType"/>
    </xsd:restriction>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="data">
  <xsd:sequence>
     <xsd:element name="simple" type="xsd:string"/>
     <xsd:element name="array"  type="tns:ArrayOfString"/>
  </xsd:sequence>
</xsd:complexType>
```

The namespaces used in the example are from WSDL 1.1 and SOAP 1.1, as follows:

```
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
```

The following example is part of a valid instance document matching the schema:

```
<array soapenc:arrayType="xsd:string[]">
  <item>item1</item>
  <item>item2</item>
</array>
```

When you build an outbound message, you must add the appropriate attributes. For example, the following ESQL shows how to add the `arrayType` attribute:

```
DECLARE xsd     NAMESPACE 'http://www.w3.org/2001/XMLSchema';
DECLARE soapenc NAMESPACE 'http://schemas.xmlsoap.org/soap/encoding/';
```

```
SET OutputRoot.SOAP.Context.Namespace.(SOAP.NamespaceDecl)xmlns:xsd = xsd;

SET OutputRoot.SOAP.Body.rpc:op1.p1.array.(SOAP.Attribute)soapenc:arrayType = 'xsd:string[]';
```

The attributes used by SOAP 1.1 and SOAP 1.2 are different. SOAP 1.1 uses the `arrayType` attribute. The size of the array can be specified, but is not required. SOAP 1.2 uses two separate attributes. The equivalent SOAP 1.2 attributes for the previous example are `soapenc:itemType="xsd:string"` and `soapenc:arraySize="2"`, where the namespace prefixes are defined as follows:

```
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soapenc=" http://www.w3.org/2003/05/soap-encoding"
```

You must disable validation if your message flow receives SOAP encoded messages containing SOAP arrays. The instance document cannot be validated against the schema when parsing an inbound message, because the name of the array items is not defined by the schema.

Multi-reference

The following example shows a SOAP 1.1 request message for a WSDL `rpc-encoded` operation called `op1`:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
                  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
  <soapenv:Body   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                  xmlns:rpc="http://example/rpc">
    <rpc:op1>
       <p1>
         <simple>text</simple>
         <array soapenc:arrayType="xsd:string[]">
           <Item>item1</Item>
           <Item>item2</Item>
         </array>
       </p1>
    </rpc:op1>
  </soapenv:Body>
</soapenv:Envelope>
```

A SOAP implementation can reorganize this logical message to use multi-reference elements, as follows:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
                  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
  <soapenv:Body   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                  xmlns:rpc="http://example/rpc">
    <rpc:op1>
      <p1 href="#id1"/>
    </rpc:op1>
    <rpc:data id="id1">
      <simple>text</simple>
      <array href="#id2"/>
    </rpc:data>
    <soapenc:Array id="id2" soapenc:arrayType="xsd:string[]">
      <Item>Array Element 0</Item>
      <Item>Array Element 1</Item>
    </soapenc:Array>
  </soapenv:Body>
</soapenv:Envelope>
```

The message is logically equivalent to the first example, but the children of elements `p1` and `array` have been split out into separate sibling elements and then referenced using the `href` attribute.

The introduced elements, such as <data>:

- Can be referenced from more than one location, which allows the message to state that such elements are identical, as opposed to separate items which have the same value. The message size could be reduced if a shared element is large and referenced many times.
- Can directly or indirectly reference themselves, which allows the message to represent a graph that contains circular references.

Neither of these considerations applies to the previous example.

Generally, when you build an outbound message, you do not encode multi-reference elements unless the message represents a graph. Otherwise, the multi-reference encoding is optional. The following ESQL example shows how to encode multi-reference elements:

```
-- ESQL namespace prefixes
DECLARE soapenc NAMESPACE 'http://schemas.xmlsoap.org/soap/encoding/';
DECLARE xsd     NAMESPACE 'http://www.w3.org/2001/XMLSchema';
DECLARE rpc NAMESPACE 'http://example/rpc';

-- define XML namespace prefixes
SET OutputRoot.SOAP.Context.Namespace.(SOAP.NamespaceDecl)xmlns:soapenc = soapenc;
SET OutputRoot.SOAP.Context.Namespace.(SOAP.NamespaceDecl)xmlns:xsd = xsd;
SET OutputRoot.SOAP.Context.Namespace.(SOAP.NamespaceDecl)xmlns:rpc = rpc;

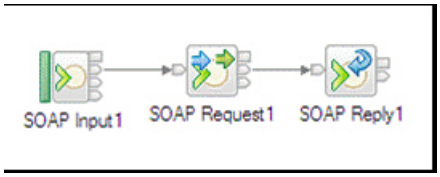
-- build request message
SET OutputRoot.SOAP.Body.rpc:op1.p1.(SOAP.Attribute)href = '#id1';

SET OutputRoot.SOAP.Body.rpc:data.(SOAP.Attribute)id = 'id1';
SET OutputRoot.SOAP.Body.rpc:data.simple = 'text';
SET OutputRoot.SOAP.Body.rpc:data.array.(SOAP.Attribute)href = '#id2';

SET OutputRoot.SOAP.Body.soapenc:Array.(SOAP.Attribute)id = 'id2';
SET OutputRoot.SOAP.Body.soapenc:Array.(SOAP.Attribute)soapenc:arrayType = 'xsd:string[]';
SET OutputRoot.SOAP.Body.soapenc:Array.Item[1] = 'item1';
SET OutputRoot.SOAP.Body.soapenc:Array.Item[2] = 'item2';
```

When parsing an inbound message, multi-reference elements do not match the XML schema from the web service WSDL. If validation is enabled on a SOAP node, then an exception is thrown. If you use multi-reference encoding, then you must disable validation.

When you have a logical tree built from the message, you can propagate this tree to another SOAP node for output. For example, with a façade flow as shown in the following diagram, if the SOAPRequest node receives a response using multi-reference encoding, you can propagate the logical tree for the response to the SOAPReply node, and a SOAP message with multi-reference encoding is returned to the original client.



The original client must also understand the multi-reference encoding.

To manipulate the logical tree for a multi-reference encoded message, navigate the `href` and `id` attributes in ESQL. These attributes are slightly different in SOAP 1.1 and SOAP 1.2, as follows:

- In SOAP 1.1, the referencing element has an attribute `href="#id"` and the item referenced `id="id"`.
- In SOAP 1.2, the referencing element has an attribute `ref="id"` and the item referenced `id="id"`.

Related concepts:

"WSDL styles" on page 1670
The SOAP nodes are configured using a specific WSDL binding that has a style of either `document` (the default) or `rpc`. All operations defined in a specific WSDL binding are usually defined with the same use, which can be either `literal` (the default) or `encoded`.

Using timeouts with HTTP and SOAP nodes:

Connect the HTTP Timeout terminal of the HTTPInput or SOAPInput nodes to further nodes to process timeouts.

Before you begin

Before you start:

- Read the message flows overview.
- Read about the options that you have for processing Web service messages, and learn more about SOAP and HTTP.

About this task

You can configure message flows that start with an HTTPInput or SOAPInput node by connecting the HTTP Timeout terminal to further nodes for processing timeouts:

- On SOAPInput nodes, messages are propagated through this terminal only when you are using an HTTP binding.
- On HTTPInput nodes, messages are propagated through this terminal only when you have configured your execution groups such that the HTTP nodes are using the embedded execution group listener.

If these conditions are not met when you deploy the BAR file for the message flow that includes one of these nodes, a warning is generated, and the path of the message flow that you have connected to the HTTP Timeout terminal is ignored. No further warnings are generated until the next restart.

To set a static timeout value in an input node:

1. Create a message flow, or open an existing flow.
2. In the Message Flow editor, select the input node for this message flow. The node properties are displayed in the Properties view (below the editor pane).
3. Set an appropriate time for the timeout interval in the property `Maximum client wait time`. The default interval is 180 seconds.

   If this time expires, and you have not connected one or more nodes to the HTTP Timeout terminal, the listener that received the client request message responds with a SOAP Fault message indicating that a timeout has occurred.
4. If you want to provide customized timeout processing, connect one or more nodes to the HTTP Timeout terminal. You must include in this sequence the

reply node that matches the input node. Therefore, if your message flow starts with an HTTPInput node, you must include an HTTPReply; if your message flow starts with a SOAPInput node, you must include a SOAPReply node.

To set a dynamic timeout value in an input node:

- Override the timeout value set on the input node by using the Java plug-in API to update it in a JavaCompute node using the `MbUtilities.changeIdentifierTimeout()` method. The following code shows an example of the `changeIdentifierTimeout` method:

```
MbMessage localEnv = assembly.getLocalEnvironment();
MbElement rootElem = localEnv.getRootElement();
MbElement repIdElement = rootElem.getFirstElementByPath(
   "/Destination/SOAP/Reply/ReplyIdentifier");
Object repId = repIdElement.getValue();
boolean success = changeIdentifierTimeout((byte[])repId, timeout);
```

- Override the timeout value set on the input node by using the "CHANGEIDENTIFIERTIMEOUT function" on page 5292.

You can derive the value that you use to replace the existing value by several means; for example:

- Create a configurable service of type UserDefined to define a timeout value, and retrieve the appropriate property.
- Read a record from a database.
- Use a value from a field within the message body.

By propagating from the HTTP Timeout terminal you can then change the contents of the responses that your message flow sends to the client. The processing on the sequence of nodes that you connect to the HTTP Timeout terminal is also subject to a further timeout, so that the client always gets a response within a known timeout interval.

When a message is propagated from the HTTP Timeout terminal the message tree contains the input headers of the original input message and a message body that is the fault timeout message. The original message body along with other information relating to the timeout can be accessed in the LocalEnvironment message tree. For example, the following record can be found in the LocalEnvironment:

```
(0x01000000:Name):HTTP = (
  (0x01000000:Name):Input = (
    (0x01000000:Name):Timeout = (
      (0x03000000:NameValue):OriginalClientLastWaitTime = 10 (INTEGER)
      (0x03000000:NameValue):OriginalClientWaitTime     = 15 (INTEGER)
      (0x03000000:NameValue):OriginalMessageMadeTheFlow = TRUE (BOOLEAN)
      (0x03000000:NameValue):OriginalRequestIdentifier  =
         X'485454500000000000000000000000c00c000000000000' (BLOB)
      (0x03000000:NameValue):OriginalInboundMessage     = X'3c3e' (BLOB)
    )
  )
)
```

For SOAPInput nodes the SOAPReply node connected on the HTTP Timeout terminal path must send a SOAP fault response message and the reply status code of 500 cannot be altered. For HTTPInput nodes any response message can be sent from the HTTP Timeout terminal and the reply status code can be changed by updating the `LocalEnvironment.Destination.HTTP.ReplyStatusCode` message tree field.

Related concepts:

"What is SOAP?" on page 1604
SOAP is an XML message format used in Web service interactions. SOAP messages are typically sent over HTTP or JMS, but other transport protocols can be used. The use of SOAP in a specific Web service is described by a WSDL definition.

"Processing HTTP messages" on page 1579
Hypertext Transfer Protocol (HTTP) is an Internet protocol that is used to transfer and display hypertext and XML documents on the Web.

"The Administration API for WebSphere Message Broker" on page 54
The Administration API for WebSphere Message Broker is a programming interface that your applications can use to control brokers and their resources through a remote interface.

Related tasks:

"Developing applications that use the Administration API" on page 956
Develop Java applications that use the Administration API (also known as the CMP API) to communicate with, deploy to, and manage brokers and their associated resources.

"Managing brokers from JavaCompute nodes" on page 997
You can use the CMP API to manage brokers and their associated resources from JavaCompute nodes in deployed message flows.

"Working with properties of a configurable service of type UserDefined at run time in a JavaCompute node" on page 987
Use the CMP API in a JavaCompute node to query, set, create, and delete properties dynamically at run time in configurable services that you have defined with type UserDefined.

"Writing ESQL" on page 2413
How you can use ESQL to customize nodes.

"Processing Web service messages" on page 1601
Use WebSphere Message Broker nodes and services to connect to other Web services providers and consumers.

Related reference:

"HTTPInput node" on page 4474
Use the HTTPInput node to receive an HTTP message from an HTTP client for processing by a message flow.

"SOAPInput node" on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

"Administration API" on page 3672
Use the Administration API for WebSphere Message Broker (CMP API) Java classes and methods to develop CMP applications.

Using SOAP MTOM with the SOAPReply, SOAPRequest, and SOAPAsyncRequest nodes:

The use of outbound MTOM messages can be configured on the SOAPReply, SOAPRequest, and SOAPAsyncRequest nodes.

The nodes have a property called `Allow MTOM`, which defines whether MTOM can be used.

An MTOM output message is written if all of the following criteria are met:

- The `Allow MTOM` property is selected on the **WS Extensions** tab.
- Validation is enabled. The `Validate` property on the SOAPRequest and SOAPAsyncRequest nodes controls validation of the anticipated response

message and not validation of the outgoing request. MTOM output is therefore suppressed unless you set `Validate` to `Content and value` on a preceding input node or transformation node.

- No child elements exist below `SOAP.Attachment` in the logical tree. If child elements are present, SOAP with Attachments (SwA) is used.
- Elements exist in the output message that are identified as base64Binary in the associated XML Schema and whose length does not fall below a default threshold size of 1000 bytes.

You can use the local environment setting `MTOMThreshold` to override the MTOM element size threshold. The MTOM element size threshold is set to a default value of 1000 bytes.

The `Allow MTOM` node property and the `MTOMThreshold` setting can both be overridden in the local environment.

The overrides that apply at a SOAPReply node are:

- `LocalEnvironment.Destination.SOAP.Reply.AllowMTOM`, which can have a value of true or false
- `LocalEnvironment.Destination.SOAP.Reply.MTOMThreshold`, which is an integer value in bytes

The equivalent overrides for a SOAPRequest or SOAPAsyncRequest node are:

- `LocalEnvironment.Destination.SOAP.Request.AllowMTOM`, which can have a value of true or false
- `LocalEnvironment.Destination.SOAP.Request.MTOMThreshold`, which is an integer value in bytes

Related concepts:

"SOAP nodes" on page 1609
The SOAP nodes act as points in the flow where Web service processing is configured and applied. Properties on the SOAP nodes control the processing carried out and can be configured by supplying a WSDL definition, or by manually configuring properties, or both.

Processing JMS messages

JMS is the standard J2EE messaging API for building enterprise messaging applications. WebSphere Message Broker provides built-in input and output nodes for its supported protocols.

The JMS specification offers an abstract, provider-independent messaging interface, so that you can write messaging applications to this common API regardless of the underlying messaging implementation.

JMS offers two common ways of messaging on which to build messaging applications: point-to-point and publish/subscribe (pub/sub). JMS applications send messages to JMS destinations, which can be queues (in the point-to-point domain) or topics (in the pub/sub domain).

JMS is defined in the following documents from Sun Microsystems:

- JMS specifications from Sun Microsystems

JMS can be used as a transport for SOAP as described in the following document issued by the World Wide Web Consortium (W3C):

- SOAP over Java Message Service 1.0 (W3C recommendation)

Related concepts:

"SOAP over JMS" on page 6698
SOAP over Java Message Service 1.0 is a specification that describes how SOAP can bind to a messaging system that supports the Java Message Service (JMS).

Related tasks:

"Working with JMS" on page 1709
Learn about the concepts and tasks involved in configuring message flows to support JMS messages.

Related reference:

"WebSphere Broker JMS Transport" on page 1681
The WebSphere Broker JMS Transport is a service that connects applications that send and receive messages that conform to the Java Message Service (JMS) standard.

"JMS message structure" on page 1688
JMS messages have a defined structure that includes headers and payloads.

"JMSInput node" on page 4532
Use the JMSInput node to receive messages from JMS destinations. JMS destinations are accessed through a connection to a JMS provider.

"JMSOutput node" on page 4549
Use the JMSOutput node to send messages to JMS destinations.

"SOAPInput node" on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

Java Message Service (JMS) API

WebSphere Message Broker supports the Java Message Service (JMS) application programming interface (API).

The JMS is an application programming interface that provides Java language functions for handling messages. Developed by messaging vendors, including IBM, in partnership with Sun Microsystems, Inc., the JMS API provides a common interface to access different enterprise messaging systems, including WebSphere MQ. This interface is appropriate for point-to-point and publish/subscribe applications.

Messaging clients in JMS are called *JMS clients*, and the messaging system is called the *JMS provider*. A *JMS application* is a business system that comprises JMS clients and at least one JMS provider. Client applications that use the JMS interface are written in the Java programming language, and are therefore restricted to the levels of JVM that are supported on the operating system in the business environment.

If you have existing end-user applications that are written to these interfaces, they can typically run unchanged in a broker environment. You must create the message flows to interact with these applications from the supported protocols by using the appropriate input and output nodes. WebSphere Message Broker provides built-in input and output nodes for its supported protocols, and other nodes that support transformation to and from JMS message formats. You can also create your own user-defined nodes to support additional protocols.

You can also create end-user applications to interact with the broker.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Configuring resources for processing JMS messages" on page 1714
A number of nodes are provided in WebSphere Message Broker for processing and routing JMS messages. Follow the links in this topic to find out how to configure the JMS nodes and broker resources for processing JMS messages.

Related reference:

"WebSphere Broker JMS Transport"
The WebSphere Broker JMS Transport is a service that connects applications that send and receive messages that conform to the Java Message Service (JMS) standard.

"Troubleshooting JMS nodes" on page 1730
Review possible problems with nodes using JMS transport.

"JMSInput node" on page 4532
Use the JMSInput node to receive messages from JMS destinations. JMS destinations are accessed through a connection to a JMS provider.

"JMSOutput node" on page 4549
Use the JMSOutput node to send messages to JMS destinations.

"JMSMQTransform node" on page 4547
Use the JMSMQTransform node to transform a message with a JMS message tree into a message that has a message tree structure that is compatible with the format of messages that are produced by the WebSphere MQ JMS provider.

"MQJMSTransform node" on page 4610
Use the MQJMSTransform node to receive messages that have a WebSphere MQ JMS provider message tree format, and transform them into a format that is compatible with messages that are to be sent to JMS destinations.

"JMSHeader node" on page 4529
Use the JMSHeader node to modify contents of the JMS Header_Values and Application properties so that you can control the node's output without programming.

WebSphere Broker JMS Transport

The WebSphere Broker JMS Transport is a service that connects applications that send and receive messages that conform to the Java Message Service (JMS) standard.

You can use the WebSphere Broker JMS Transport to support the following operations:

- Receive a JMS message as input.
- Create a JMS message for output.
- Work with message flows that do not expect JMS messages.

All JMS messages must conform to the Java Message Service Specification, version 1.1.

To use the WebSphere Broker JMS Transport, you must deploy a message flow that contains at least one of the built-in JMS nodes, which support the exchange of JMS messages. You can create message flows to receive messages from JMS destinations, and send messages to JMS destinations. These destinations are accessible through a connection to a JMS provider. In sending and receiving messages, the JMS nodes behave like JMS clients.

You can also use two transformation nodes, the JMSMQTransform node and the MQJMSTransform node, with the JMSInput and JMSOutput nodes so that they can interact with nodes that expect a propagated message to contain an MQMD (and MQRFH2) header.

- The JMSMQTransform node takes the output of the JMSInput node, and produces a message that can be handled by an MQOutput node.
- The MQJMSTransform node transforms a message with an MQMD (and optional MQRFH2) header into a message that is expected by the JMSOutput node.

If you want to change or add content to the headers in JMS messages, you can include a JMSHeader node in the message flow to modify fields without programming.

You can configure the JMS nodes to work with the WebSphere MQ JMS provider, WebSphere Application Server Version 6.0, the IBM Service Integration Bus, and all JMS providers that conform to the Java Message Service Specification, version 1.1. WebSphere Message Broker supports Java 6 (also known as Java 1.6).

Refer to the following topics for further information about JMS messages and JMS providers:

- "Simplified JMS message representation" on page 1683
- "JMS message transformation" on page 1684
- "JMS message structure" on page 1688
- "JMS message selector" on page 1703
- "JMS Transactionality" on page 1705
- "JMS message domain properties" on page 1707
- "JMS properties for application communication models" on page 1708

Related concepts:

"Nodes for connectivity" on page 1028
WebSphere Message Broker supports direct connections from applications, and can send direct requests to other application endpoints. WebSphere Message Broker can also connect to various subsystems including WebSphere MQ, files, and databases, to read and write existing application data.

"Simplified JMS message representation" on page 1683
The JMSInput node receives an input message as a Java object, and not as a bit stream wire format (as is the case with an MQInput node). The message does not populate an MQMD and RFH2 header, but instead populates a new message tree that represents a JMS message in a more native way.

"JMS message transformation" on page 1684
The JMSInput and JMSOutput nodes expect JMS messages, and therefore expect a native JMS message tree representation.

"Connection to different JMS providers" on page 1709
The JMSInput and JMSOutput nodes are compatible with all JMS providers that conform to the Java Message Service Specification, version 1.1.

Related tasks:

"Configuring resources for processing JMS messages" on page 1714
A number of nodes are provided in WebSphere Message Broker for processing and routing JMS messages. Follow the links in this topic to find out how to configure the JMS nodes and broker resources for processing JMS messages.

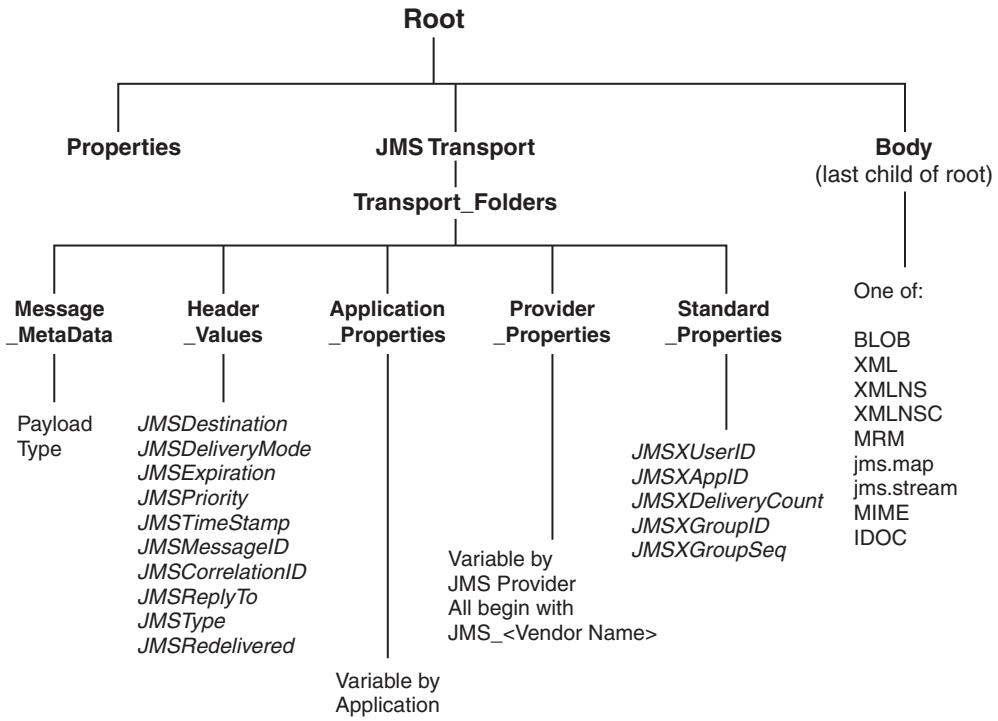
Related reference:

"Troubleshooting JMS nodes" on page 1730
Review possible problems with nodes using JMS transport.
"JMSInput node" on page 4532
Use the JMSInput node to receive messages from JMS destinations. JMS destinations are accessed through a connection to a JMS provider.
"JMSOutput node" on page 4549
Use the JMSOutput node to send messages to JMS destinations.
"JMSMQTransform node" on page 4547
Use the JMSMQTransform node to transform a message with a JMS message tree into a message that has a message tree structure that is compatible with the format of messages that are produced by the WebSphere MQ JMS provider.
"MQJMSTransform node" on page 4610
Use the MQJMSTransform node to receive messages that have a WebSphere MQ JMS provider message tree format, and transform them into a format that is compatible with messages that are to be sent to JMS destinations.
"JMSHeader node" on page 4529
Use the JMSHeader node to modify contents of the JMS Header_Values and Application properties so that you can control the node's output without programming.

Simplified JMS message representation:

The JMSInput node receives an input message as a Java object, and not as a bit stream wire format (as is the case with an MQInput node). The message does not populate an MQMD and RFH2 header, but instead populates a new message tree that represents a JMS message in a more native way.

To represent a JMS message in a message tree, a new canonical form has been created. This message tree allows for representation of JMS message header data, and message properties. The JMS message tree is in a format that is recognizable to Java programmers.



For details about the structure and content of the JMS message tree, see "Representation of messages in the JMS Transport" on page 1691.

Related concepts:

"JMS message transformation"
The JMSInput and JMSOutput nodes expect JMS messages, and therefore expect a native JMS message tree representation.

Related reference:

"JMS message types" on page 1690
JMS defines six message interface types; a base message type and five subtypes. The message types are defined according to the type of the message *payload*, where the payload is the body of a message that holds the content.

"JMS message structure" on page 1688
JMS messages have a defined structure that includes headers and payloads.

"JNDI administered objects" on page 1702
JNDI (Java Naming and Directory Interface) is a standard Java extension that provides a uniform API for accessing various directory and naming services.

"JMS message for output" on page 1700
When the JMSOutput node receives a JMS message, it calls the JMSTransport parser to return an XML bit stream containing the JMSTransport section of the message, so that it can be examined and processed.

"JMSInput node" on page 4532
Use the JMSInput node to receive messages from JMS destinations. JMS destinations are accessed through a connection to a JMS provider.

JMS message transformation:

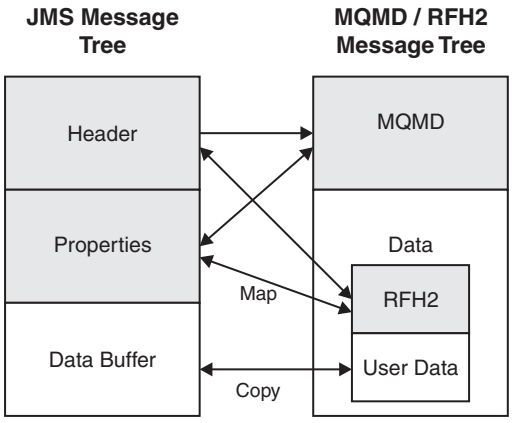
The JMSInput and JMSOutput nodes expect JMS messages, and therefore expect a native JMS message tree representation.

You can use the following nodes to transform messages between a WebSphere MQ JMS message tree and a JMS message tree:

- JMSMQTransform node
- MQJMSTransform node

These nodes do not have any configurable properties. The JMSMQTransform node transforms a native JMS message tree to a WebSphere MQ JMS message tree, and the MQJMSTransform node performs the transformation in the opposite direction.

The following diagram provides an overview of the mapping scheme that is used:



This mapping diagram uses the same scheme as the WebSphere MQ JMS provider to convert between a JMS message and an MQMD or MQRFH2 message.

When transforming between a WebSphere MQ message tree and a native JMS message tree, the transformation nodes copy elements from different parts of a message tree:

- The following fields are copied from the JMS message to the MQMD, if they exist in the incoming JMS message:

JMS field	**MQMD field**
JMSMessageID	MsgId
JMSCorrelationID	CorrelId
JMSPriority	Priority
JMSDeliveryMode	Persistence
JMSQApplid	PutApplName
JMSUser	UserIdentifier
JMSXDeliveryCount	BackoutCount - 1
JMSTimeStamp	PutDate, PutTime

- The following fields are copied from the JMS message to the MQRFH2 JMS folder:

JMS field	**MQRFH2 JMS field**
JMSDestination	Dst

JMS field	MQRFH2 JMS field
JMSDeliveryMode	Dlv
JMSExpiration	Exp
JMSPriority	Pri
JMSTimestamp	Tms
JMSCorrelationID	Cid
JMSReplyTo	Rto

- The following fields are copied from the MQMD to the JMS message:

MQMD field	JMS field
Expiry	JMSExpiration
Persistence	JMSDeliveryMode
Priority	JMSPriority
MsgId	JMSMessageID
CorrelId	JMSCorrelationID
BackoutCount = 0	JMSRedelivered = false
BackoutCount > 0	JMSRedelivered = true
GroupId	JMSGroupid
MsgSeqNumber	JMSGroupseq
UserIdentifier	JMSUser
PutApplName	JMSApplid
PutDate, PutTime	JMSTimeStamp

- The following fields are copied from the MQRFH2 JMS folder to the JMS message:

MQRFH2 JMS field	JMS field
Dst	JMSDestination
Dlv	JMSDeliveryMode
Pri	JMSPriority
Cid	JMSCorrelationID
Rto	JMSReplyTo

Example message flow scenario: JMSInput node to MQOutput node

1. A JMSInput node is configured to subscribe to topic ABC.
2. An application that is connected to the JMS server publishes on topic ABC.
3. A publication is received at the JMSInput node.
4. The node extracts data from the JMS message.

5. The JMS message is passed to the JMSMQTransform node where the message is converted to a WebSphere MQ message.
6. The MQOutput node receives the WebSphere MQ message, and publishes the message on a WebSphere MQ queue.

The final destination is a WebSphere MQ queue, therefore the message must pass through a JMSMQTransform node to convert the message tree to a WebSphere MQ JMS format before it reaches the MQOutput node.

Example message flow scenario: MQInput node to JMSOutput node

1. An MQInput node receives a message from a WebSphere MQ queue.
2. The MQInput node creates a WebSphere MQ message.
3. The MQ message is passed to the MQJMSTransform node where the message tree is converted to a JMS format.
4. The JMSOutput node receives the JMS message and publishes the JMS message on topic XYZ.

Additional examples

These examples show some of the solutions that you can achieve when you use the JMS Transport. Other solutions are possible; for example, the message can be passed to a Compute node or a JavaCompute node and the contents can be modified as required.

Look at the following sample for examples of the JMS nodes being used in message flows:

- JMS Nodes

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Related concepts:

"JMS parsers and domains" on page 1116
The JMSMap and JMSStream domains can be used for modeling messages that are produced by the implementations of the Java Messaging Service standard.

Related reference:

"JMS message selector" on page 1703
A message selector allows a JMS consumer to be more selective about the messages that it receives from a particular topic or queue.

"JMSInput node" on page 4532
Use the JMSInput node to receive messages from JMS destinations. JMS destinations are accessed through a connection to a JMS provider.

"JMSOutput node" on page 4549
Use the JMSOutput node to send messages to JMS destinations.

"JMS message as input" on page 1693
The JMS message is a Java object, and therefore you cannot parse the message as a bit stream. When the message is received, the header data, property data, and payload data are extracted by using the JMS API.

"JMS message for output" on page 1700
When the JMSOutput node receives a JMS message, it calls the JMSTransport parser to return an XML bit stream containing the JMSTransport section of the message, so that it can be examined and processed.

JMS message structure:

JMS messages have a defined structure that includes headers and payloads.

The following figure depicts the JMS message structure:

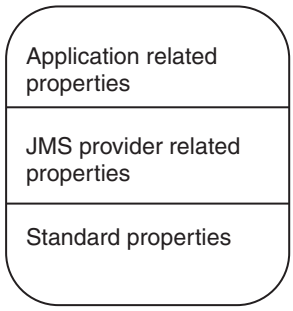
Header

A header must be present in every JMS message, and it is assigned automatically. Most of the values in the header are set by the JMS provider when the message is put to a JMS destination. Some values can be declared by the JMS client when it creates a JMS session, or when it creates the message consumer or producer; for example, *JMSDeliveryMode*, *JMSExpiration*, *JMSReplyTo*, and *JMSCorrelationID* are created when the JMS client creates a JMS session or creates the message consumer or producer.

The data elements of each header comprise name-value pairs and they can be any of the Java following types: Boolean, byte, short, char, long, int, float, double, string, or byte[].

Properties

The properties are optional, and can be divided into the following subsections:



- *Application-related properties*

  A Java application can assign application-related properties, which are set before the message is delivered. The property names of the application are meaningful only to the sending and receiving applications.
- *Provider-related properties*

  Every JMS provider can define proprietary properties that can be set either by the client or automatically by the provider. Provider-related properties are prefixed with *JMS_* followed by the vendor name and the specific property name. For example, the WebSphere MQ JMS client sets the provider property to be *JMS_IBM_MsgType*.
- *Standard properties*

  These properties are set by the JMS provider when a message is sent. The JMS provider vendor can choose to not support any standard properties, to support some standard properties, or to support all standard properties. Standard property names start with *JMSX*; for example: *JMSXUserid* or *JMSXDeliveryCount*.

The properties are handled as name-value pairs, and they can be any of the following Java types: Boolean, byte, short, char, long, int, float, double, string, or byte[].

Payload

The payload type defines the JMS message. It can be one of the six JMS message types that are described in "JMS message types" on page 1690.

JMS does not define a wire format. The Java Message Service Specification, version 1.1 describes the physical representation of how a message is structured.

Related reference:

"JMS message types" on page 1690
JMS defines six message interface types; a base message type and five subtypes. The message types are defined according to the type of the message *payload*, where the payload is the body of a message that holds the content.

"Representation of messages in the JMS Transport" on page 1691
Messages that are sent in the JMS Transport are represented by the JMS Transport message tree.

"JMS message as input" on page 1693
The JMS message is a Java object, and therefore you cannot parse the message as a bit stream. When the message is received, the header data, property data, and payload data are extracted by using the JMS API.

"JMS message for output" on page 1700
When the JMSOutput node receives a JMS message, it calls the JMSTransport parser to return an XML bit stream containing the JMSTransport section of the message, so that it can be examined and processed.

"JNDI administered objects" on page 1702
JNDI (Java Naming and Directory Interface) is a standard Java extension that provides a uniform API for accessing various directory and naming services.

JMS message types:

JMS defines six message interface types; a base message type and five subtypes. The message types are defined according to the type of the message *payload*, where the payload is the body of a message that holds the content.

JMS specifies only the interface and does not specify the implementation. This approach allows for vendor-specific implementation and transportation of messages while using a common interface.

The following table describes the six message types:

Message type	Description
Message	The base class. This message type is used for event notification, and does not have a payload.
BytesMessage	The payload is stored as an array of bytes. This message type is useful for exchanging data in a format that is native to the application, and when JMS is used as a transport between two systems, where the JMS client does not know the message payload type. Use this message type to transmit XML messages to ensure that the message is transmitted efficiently, and is not subject to unnecessary data conversion.
TextMessage	Data is stored as a string. This message type is useful for exchanging simple text messages.
StreamMessage	A Stream message is a sequence of primitive Java types. The message object tracks the order and the types of these primitives within the stream. Formal conversion rules apply; for example, an exception is thrown if a JMS application tries to read a double value as a short value. Refer to the Java Message Service Specification, version 1.1 for a full list of the conversion rules. `21ABCDEFGH32.345` is an example of a StreamMessage payload. It consists of the following three fields: • An Integer, `21` • A String, `ABCDEFGH` • A Float, `32.345` If the data structure is unknown, the generic method `readObject()` can be used to return the next object in the stream. If the structure of the data is known, the JMS client can be specific about the type of object being accessed.

Message type	Description
MapMessage	The payload of a MapMessage is stored as a set of name-value pairs. The name is defined as a string and the value is typed. The MapMessage is useful for delivering keyed data that can change from one message to the next. `NumberOfCopies:5` is an example of a MapMessage payload, where `NumberOfCopies` is the key and `5` is the value. Data can be accessed by using `getMapNames()`, which returns a Java Enumeration object. It is possible to iterate through the MapMessage by using `hasMoreElements()` to retrieve the mapped name-value pairs.
ObjectMessage	The Object message carries a serializable Java Object as its payload. It is useful for exchanging Java objects.

Related reference:

"JMS message structure" on page 1688
JMS messages have a defined structure that includes headers and payloads.

"Representation of messages in the JMS Transport"
Messages that are sent in the JMS Transport are represented by the JMS Transport message tree.

"JMS message as input" on page 1693
The JMS message is a Java object, and therefore you cannot parse the message as a bit stream. When the message is received, the header data, property data, and payload data are extracted by using the JMS API.

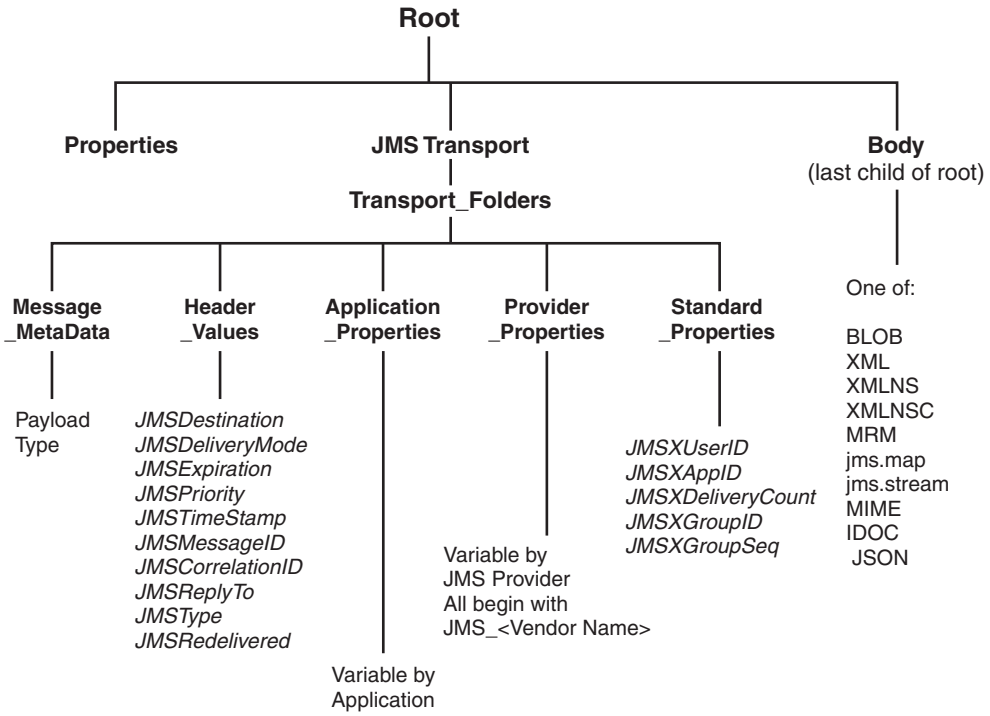
"JMS message for output" on page 1700
When the JMSOutput node receives a JMS message, it calls the JMSTransport parser to return an XML bit stream containing the JMSTransport section of the message, so that it can be examined and processed.

"JNDI administered objects" on page 1702
JNDI (Java Naming and Directory Interface) is a standard Java extension that provides a uniform API for accessing various directory and naming services.

Representation of messages in the JMS Transport:

Messages that are sent in the JMS Transport are represented by the JMS Transport message tree.

The following figure depicts the JMS message tree that is propagated from the JMSInput node and can be propagated to the JMSOutput and JMSReply nodes. The JMSOutput and JMSReply nodes populate details of the JMS message that is sent to the LocalEnvironment WrittenDestination folder, as described in "JMSOutput node" on page 4549.



JMSTransport

- *Header_Values* subfolder:

  This subfolder is mandatory and is always created.

- Properties subfolders:

  JMS message properties are optional; if they are present in the message, they are stored in the appropriate properties subfolder.

- *Message_MetaData* subfolder:

  This subfolder is included to preserve the payload type of the JMS message. The folder is used by the JMSOutput node when it creates a JMS message. The payload type can be one of the values shown in the following table.

Message type	**Payload values**
Base JMS message with no payload	jms_none
TextMessage	jms_text
BytesMessage	jms_bytes
MapMessage	jms_map
StreamMessage	jms_stream
ObjectMessage	jms_object

Body

The message payload is stored in the body folder, which is the last child of Root. The payload is transferred by using one of the following message domain parsers:

- XML
- XMLNS
- XMLNSC

- BLOB
- JMSMap
- JMSStream
- MRM
- MIME
- IDOC
- JSON

Related concepts:

"Simplified JMS message representation" on page 1683
The JMSInput node receives an input message as a Java object, and not as a bit stream wire format (as is the case with an MQInput node). The message does not populate an MQMD and RFH2 header, but instead populates a new message tree that represents a JMS message in a more native way.

Related reference:

"JMS message types" on page 1690
JMS defines six message interface types; a base message type and five subtypes. The message types are defined according to the type of the message *payload*, where the payload is the body of a message that holds the content.

"JMS message structure" on page 1688
JMS messages have a defined structure that includes headers and payloads.

"JMS input message header and property data" on page 1694
The JMSInput node obtains header and property data from JMS messages.

"JMSInput node" on page 4532
Use the JMSInput node to receive messages from JMS destinations. JMS destinations are accessed through a connection to a JMS provider.

"JMSOutput node" on page 4549
Use the JMSOutput node to send messages to JMS destinations.

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

JMS message as input:

The JMS message is a Java object, and therefore you cannot parse the message as a bit stream. When the message is received, the header data, property data, and payload data are extracted by using the JMS API.

For information on the header, properties, and payload of JMS messages, refer to "JMS message structure" on page 1688.

The following topics describe how the different parts of the JMS message are obtained, and how the message is parsed:

- "JMS input message header and property data" on page 1694
- "JMS Message payload" on page 1696
- "JMS message payload and appropriate parser" on page 1698

Related concepts:

"JMS message transformation" on page 1684
The JMSInput and JMSOutput nodes expect JMS messages, and therefore expect a native JMS message tree representation.

Related reference:

"JMS message structure" on page 1688
JMS messages have a defined structure that includes headers and payloads.

"JMS message types" on page 1690
JMS defines six message interface types; a base message type and five subtypes. The message types are defined according to the type of the message *payload*, where the payload is the body of a message that holds the content.

"Representation of messages in the JMS Transport" on page 1691
Messages that are sent in the JMS Transport are represented by the JMS Transport message tree.

"JNDI administered objects" on page 1702
JNDI (Java Naming and Directory Interface) is a standard Java extension that provides a uniform API for accessing various directory and naming services.

"JMS message for output" on page 1700
When the JMSOutput node receives a JMS message, it calls the JMSTransport parser to return an XML bit stream containing the JMSTransport section of the message, so that it can be examined and processed.

"JMSInput node" on page 4532
Use the JMSInput node to receive messages from JMS destinations. JMS destinations are accessed through a connection to a JMS provider.

JMS input message header and property data:

The JMSInput node obtains header and property data from JMS messages.

Header data

The JMSInput node extracts header data from messages by using JMS API methods. Header data is stored as name-value pairs in the *Header_Values* folder. The API methods return the value; for example, to get the value for the header field *JMSTimestamp*, the JMSInput node uses the `getJMSTimestamp( )` method. A similar method is provided for each of the following fixed header fields:

- JMSDestination
- JMSDeliveryMode
- JMSExpiration
- JMSPriority
- JMSTimeStamp
- JMSMessageID
- JMSCorrelationID
- JMSReplyTo
- JMSType
- JMSRedelivered

Property data

In a similar way to how the header data is obtained, the JMSInput node extracts property data from messages by using JMS API methods. Property data is stored as name-value pairs in the properties folders. The API method returns a value for every property name with which it is supplied.

XML representation of header and property data

The JMSInput node uses the header and property data to create an XML representation of the JMSTransport folders. The node passes the XML data to the JMSTransport parser as a byte array. The byte array is then used to populate or to refresh the elements in the message tree.

Preservation of Java type

A scheme is not required to preserve knowledge of the Java type because the header value Java types are fixed and known. The JMS message properties are optional, therefore a scheme is required to preserve the Java type of the property values. The scheme used is that which is implemented by the WebSphere MQ JMS client and the Real-timeInput node.

Java type information is represented as a metadata in the form of a keyword `dt='DataType'` where *Datatype* is a string. The Java type is passed in the XML as part of the element name `<ElementName dt='DataType'>Value</ElementName>`. *Datatype* can be any of the following values:

Datatype value	Definition
String	Any sequence of characters, excluding `<` and `&`
Boolean	The character `0` or `1`, where 1 is equal to "true"
bin.hex	Hexadecimal digits representing octets
I1	A number, expressed by using the digits 0..9, with optional sign (no fractions or exponent). The value must lie in the range -128 to 127 inclusive.
I2	A number, expressed by using the digits 0..9, with optional sign (no fractions or exponent). The value must lie in the range -32768 to 32767 inclusive.
I4	A number, expressed by using the digits 0..9, with optional sign (no fractions or exponent). The value must lie in the range -2147483648 to 2147483647 inclusive.
I8	A number, expressed by using the digits 0..9, with optional sign (no fractions or exponent). The value must lie in the range -9223372036854775808 to 9223372036854775807 inclusive.
int	A number, expressed by using the digits 0..9, with optional sign (no fractions or exponent). The value must lie in the same range as the datatype value`I8`. This number can be used in place of one of the `I*` types if the sender does not want to associate a particular precision with the property.
R4	A floating point number, expressed by using the digits 0..9, optional sign, optional fractional digits, optional exponent. Magnitude <= 3.40282347E+38, and>= 1.175E-37
R8	A floating point number, expressed by using the digits 0..9, optional sign, optional fractional digits, optional exponent. Magnitude <= 1.7976931348623E+308, and>= 2.225E-307

Related reference:

"JMS message structure" on page 1688
JMS messages have a defined structure that includes headers and payloads.

"JMS message types" on page 1690
JMS defines six message interface types; a base message type and five subtypes. The message types are defined according to the type of the message *payload*, where

the payload is the body of a message that holds the content.

"Representation of messages in the JMS Transport" on page 1691
Messages that are sent in the JMS Transport are represented by the JMS Transport message tree.

"JMS Message payload"
How payload is extracted from the JMS message for each of the JMS Message types.

"JMS message payload and appropriate parser" on page 1698
Configure the JMSInput node properties to specify the message domain that will be used to parse the JMS message payload.

"JMS message for output" on page 1700
When the JMSOutput node receives a JMS message, it calls the JMSTransport parser to return an XML bit stream containing the JMSTransport section of the message, so that it can be examined and processed.

"Real-timeInput node" on page 4646
The Real-timeInput node, available in earlier versions of WebSphere Message Broker, is not supported in WebSphere Message Broker Version 7.0. See for information about migrating your message flows from WebSphere Message Broker to WebSphere Message Broker Version 7.0.

JMS Message payload:

How payload is extracted from the JMS message for each of the JMS Message types.

The payload for some of the JMS message types can be extracted as a whole from the message object by using the JMS API. The payload is passed as a bit stream to a broker parser. This is true for the following message types:

- BytesMessage
- TextMessage
- ObjectMessage

  Additional processing is required to deal with the ObjectMessage payload because the JMS ObjectMessage payload is a serialized Java Object.

  The JMSInput node obtains the payload by calling `getObject( )` on the message. `getObject( )` returns a de-serialized object of the original class. This class definition must be made available to the JMSInput node, and you should ensure that it is accessible through the broker's Java class path. (The class path is defined in the mqsiprofile batch file, which is in the broker's executable directory; for example, on Windows, this is `mqsiprofile.cmd` in the *`install_dir`*`/bin` directory.) The JMSInput node invokes the BLOB parser, which creates the message body by using a bit stream that is created from the object.

  The Java Object can be subsequently re-serialized in a JavaCompute node or a user-defined extension, and is updated by using its method calls.

The payload for MapMessage and StreamMessage can be extracted only as individual elements and must be reformatted by the JMSInput node before it can be used to create the message body.

- **MapMessage payload**

  The JMSMap domain is a synonym for the broker XML parser, which expects a stream of XML data. MapMessage payload data however, is extracted as sets of name-value pairs from the message object. The JMS API is used to obtain the name-value pairs.

The JMSInput node appends each name-value pair to a bit stream as an XML element and value, and preserves the type of the value by using the `dt=` attribute.

The following example shows the XML that is generated by the JMSInput node for the MapMessage payload:

```
<map>
    <Item_8_of_10_Char dt='char'>A</Item_8_of_10_Char>
     <Item_5_of_10_Double dt='r8'>999999.0</Item_5_of_10_Double>
     <Item_10_of_10_String>Last Map Item</Item_10_of_10_String>
     <Item_9_of_10_Boolean dt='boolean'>0</Item_9_of_10_Boolean>
     <Item_2_of_10_Integrer dt='i4'>999</Item_2_of_10_Integrer>
     <Item_3_of_10_Short dt='i2'>9999</Item_3_of_10_Short>
     <Item_7_of_10_Byte dt='i1'>9</Item_7_of_10_Byte>
     <Item_6_of_10_Float dt='r4'>2.24</Item_6_of_10_Float>
     <Item_1_of_10_String>P2P Map Msg Number:1</Item_1_of_10_String>
     <Item_4_of_10_Long dt='i8'>99999</Item_4_of_10_Long>
          </map>
```

In this example, the message contains 10 fields. The field names have been generated by a JMS Client application, and take the form `item_n_of_x_t`, where:

- `n` is the sequence number in which the item was added to the message,
- `x` is the total number of items in the map,
- `t` is the type of the value.

The map data is not returned from the JMS API the order in which it was received.

- **StreamMessage payload**

  The StreamMessage payload data is a sequence of fields, where each field has a specific type. The fields do not have associated names and so a default element name `elt` is used to generate the XML elements. Similar to the MapMessage, the JMS API allows for fields only to be retrieved individually. The JMSInput node extracts fields from the JMS message and appends each to a bit stream in XML format.

  The following is an example of the XML that is generated by the JMSInput node for the StreamMessage payload:

```
<stream>
     <elt>P2P Stream Message  Number :7</elt>
     <elt dt='i4'>999</elt>
     <elt dt='i2'>9999</elt>
     <elt dt='i8'>99999</elt>
     <elt dt='r8'>999999.0</elt>
     <elt dt='r4'>2.24</elt>
     <elt dt='i1'>9</elt>
     <elt dt='char'>A</elt>
     <elt dt='boolean'>0</elt>
     <elt>Last Stream Item</elt>
</stream>
```

  In this example, 10 typed values are added to the StreamMessage by a JMS client application.

Related concepts:

"JMS parsers and domains" on page 1116
The JMSMap and JMSStream domains can be used for modeling messages that are produced by the implementations of the Java Messaging Service standard.

Related reference:

"JMS message structure" on page 1688
JMS messages have a defined structure that includes headers and payloads.

"JMS message types" on page 1690
JMS defines six message interface types; a base message type and five subtypes. The message types are defined according to the type of the message *payload*, where the payload is the body of a message that holds the content.

"Representation of messages in the JMS Transport" on page 1691
Messages that are sent in the JMS Transport are represented by the JMS Transport message tree.

"JMS input message header and property data" on page 1694
The JMSInput node obtains header and property data from JMS messages.

"JMS message payload and appropriate parser"
Configure the JMSInput node properties to specify the message domain that will be used to parse the JMS message payload.

"JMS message for output" on page 1700
When the JMSOutput node receives a JMS message, it calls the JMSTransport parser to return an XML bit stream containing the JMSTransport section of the message, so that it can be examined and processed.

JMS message payload and appropriate parser:

Configure the JMSInput node properties to specify the message domain that will be used to parse the JMS message payload.

When the JMSInput node creates a message tree from the JMS message payload, the appropriate message domain for that payload must be used. Therefore, the JMSInput node must know the type of JMS message that it expects to receive. The JMSInput node extracts the payload from the JMS message using the appropriate JMS API, then passes the payload data to the parser for the domain. The parser creates the body portion of the message tree.

The message domain is derived according to the following criteria and in the following order of precedence:

1. "The Message domain property is set to a specific domain"
2. "The Message domain property is left blank (default) and the JMSType header field from the JMS input message is used to indicate the domain" on page 1699
3. " The Message Domain property is left blank (default) and the JMSType header field from the JMS input message is also left blank" on page 1699

The Message domain property is set to a specific domain

In this case, the node expects to receive only the allowed JMS message types for that domain, as shown by the following table:

	Valid JMS message types				
Message domain	**BytesMessage**	**TextMessage**	**MapMessage**	**StreamMessage**	**ObjectMessage**
BLOB	•	•			•
XMLNS		•			
XMLNSC		•			
MRM	•	•			
JMSMap			•		
JMSStream				•	
MIME	•	•			

Message domain	Valid JMS message types				
	BytesMessage	**TextMessage**	**MapMessage**	**StreamMessage**	**ObjectMessage**
IDOC	•	•			
XML		•			

- If the JMSInput node receives a JMS message type that is not valid for the message domain that is configured in the JMSInput node, the node issues a warning and backs out the message either to the source JMS provider destination, or to the backout destination.
- If you specify the MRM domain, you must also specify the `Message set`, `Message type` and `Message format` node properties.
- If you specify the IDOC domain, you must also specify the `Message set` and `Message format` node properties.
- If you specify the XMLNSC domain, and you want to validate input messages, you must also specify the `Message set` node property.

The Message domain property is left blank (default) and the JMSType header field from the JMS input message is used to indicate the domain

The JMSType header field must be set according to the URI format shown in the following table. The domain in the `mcd:` string can be uppercase or lowercase.

JMSType	Message domain
`mcd://BLOB`	BLOB
`mcd://MRM/[set]/[type]/[?format=fmt]`	MRM
`mcd://XMLNS`	XMLNS
`mcd://XMLNSC/[set]`	XMLNSC
`mcd://IDOC/[set]/[?format=fmt]`	IDOC
`mcd://MIME`	MIME
`mcd://XML`	XML

- If the JMSInput node receives a JMS message type that is not valid for the message domain configured in the JMSType header, the node issues a warning and backs out the message either to the source JMS provider destination, or to the backout destination.
- If the JMSType field does not conform to this URI format, the message is handled in the BLOB domain.
- For details of the `[type]` syntax, refer to “Specifying namespaces in the Message Type property” on page 1208.
- If the XMLNSC domain is specified, use `[set]` only if you want to validate input messages.

The Message Domain property is left blank (default) and the JMSType header field from the JMS input message is also left blank

The message domain is set according to the JMS message Java class as follows:

JMS message type	Message domain
TextMessage	XML
BytesMessage	BLOB

JMS message type	Message domain
MapMessage	JMSMap
StreamMessage	JMSStream
ObjectMessage	BLOB

Related concepts:

"JMS parsers and domains" on page 1116
The JMSMap and JMSStream domains can be used for modeling messages that are produced by the implementations of the Java Messaging Service standard.

"Specifying namespaces in the Message Type property" on page 1208
When using the MRM domain, the *Message Type* property is used to specify the name of the message.

"Multipart messages" on page 1191
A *multipart message* contains one or more other messages within its structure. The contained message is sometimes referred to as an *embedded message*.

Related reference:

"JMS message structure" on page 1688
JMS messages have a defined structure that includes headers and payloads.

"JMS message types" on page 1690
JMS defines six message interface types; a base message type and five subtypes. The message types are defined according to the type of the message *payload*, where the payload is the body of a message that holds the content.

"Representation of messages in the JMS Transport" on page 1691
Messages that are sent in the JMS Transport are represented by the JMS Transport message tree.

"JMS input message header and property data" on page 1694
The JMSInput node obtains header and property data from JMS messages.

"JMS Message payload" on page 1696
How payload is extracted from the JMS message for each of the JMS Message types.

"JMS message for output"
When the JMSOutput node receives a JMS message, it calls the JMSTransport parser to return an XML bit stream containing the JMSTransport section of the message, so that it can be examined and processed.

JMS message for output:

When the JMSOutput node receives a JMS message, it calls the JMSTransport parser to return an XML bit stream containing the JMSTransport section of the message, so that it can be examined and processed.

The node extracts the *Message_MetaData* and obtains the payload type information to identify which JMS message type to create for output. If the *Message_MetaData* folder is not present, the output node creates a BytesMessage by default.

Header data

The JMSOutput node extracts the JMS header data from the XML string, and uses this data to populate the values for the JMS header fields in the message.

Property data

The JMSOutput node extracts the property values from the XML string. The XML elements contain type information that identifies which Java Object type to create for each property value.

Message payload

The message payload is obtained from the JMS message as a bit stream. For TextMessage and BytesMessage payloads, the bit stream can be passed to the JMS API directly to create the appropriate payload.

For MapMessage and StreamMessage payloads, the individual elements must be extracted from the XML bit stream. The output node calls the appropriate JMS API method to create the map or stream fields in the message.

For an ObjectMessage payload, the JMSOutput node reserializes the bit stream payload by using the object class. The object class must be available in the Java class path for the broker. The class path is defined in the **mqsiprofile** batch file, which is in the directory that contains the executable files for the broker; for example, on Windows, the file is `mqsiprofile.cmd` in the *install_dir*`/bin` directory.

Sending JMS messages

The JMSOutput node generates and supports:

Sending a datagram message
A message with sufficient information to reach its destination, but without an expectation of there being a response as defined in the node attributes.

Sending a Reply message
The message is treated as a reply as defined by the JMSReplyTo property value.

Sending a Request message
The JMSOutput node sends a message to a defined JMS destination with the expectation of a response from the recipient.

See "Using the Message Destination Mode" on page 4551 for more information about how you perform these tasks.

Message publication

The message is published to the JMS destination that has been specified as a property of the JMSOutput node. However, if the JMSReplyTo header field is set in the JMS message, the JMSOutput node treats the message as a reply to a previous request, and publishes the message to the JMS destination of the previous request.

Related reference:

"JMS message structure" on page 1688
JMS messages have a defined structure that includes headers and payloads.

"JMS message types" on page 1690
JMS defines six message interface types; a base message type and five subtypes. The message types are defined according to the type of the message *payload*, where the payload is the body of a message that holds the content.

"JMSOutput node" on page 4549
Use the JMSOutput node to send messages to JMS destinations.

"Representation of messages in the JMS Transport" on page 1691
Messages that are sent in the JMS Transport are represented by the JMS Transport message tree.

"JMS message as input" on page 1693
The JMS message is a Java object, and therefore you cannot parse the message as a bit stream. When the message is received, the header data, property data, and payload data are extracted by using the JMS API.

"JNDI administered objects"
JNDI (Java Naming and Directory Interface) is a standard Java extension that provides a uniform API for accessing various directory and naming services.

JNDI administered objects:

JNDI (Java Naming and Directory Interface) is a standard Java extension that provides a uniform API for accessing various directory and naming services.

JMS clients use JNDI to browse a naming service to obtain references to administered objects. *Administered objects* are JMS connection factory and JMS destination objects, where JMS destination objects are topics and queues. Administered objects are created and configured by a system administrator.

To create and configure JNDI administered objects, refer to the JMS provider documentation. If you are using the WebSphere MQ JMS provider, see the sample JMSAdmin definitions file that is included with WebSphere MQ and refer to the *Using Java* section in the WebSphere MQ Version 7 Information Center online.

Location of JNDI administered objects

JNDI administered objects are stored in the bindings. This storage can be either file system based or based on LDAP (Lightweight Directory Access Protocol). LDAP is a software protocol that enables everyone to locate organizations, individuals, and other resources; for example, locating files and devices in a network, either on the public Internet or on a corporate intranet.

LDAP is part of X.500, which is a standard for directory services in a network.

Naming service

A naming service associates names with distributed objects so that the administered objects are located by using names and not complex network addresses. JNDI provides an abstraction that hides the specifics of the naming service, which makes client applications more portable.

A JMS client specifies a *JNDI InitialContext* to obtain a JNDI connection to the JMS messaging server. The InitialContext is the starting point in any JNDI lookup and acts like the root of a file system. The JMS directory service that is being used determines the properties that are used to create an InitialContext.

Related reference:

"JMS message structure" on page 1688
JMS messages have a defined structure that includes headers and payloads.

"JMS message types" on page 1690
JMS defines six message interface types; a base message type and five subtypes. The message types are defined according to the type of the message *payload*, where

the payload is the body of a message that holds the content.

"Representation of messages in the JMS Transport" on page 1691
Messages that are sent in the JMS Transport are represented by the JMS Transport message tree.

"JMS message as input" on page 1693
The JMS message is a Java object, and therefore you cannot parse the message as a bit stream. When the message is received, the header data, property data, and payload data are extracted by using the JMS API.

"JMS message for output" on page 1700
When the JMSOutput node receives a JMS message, it calls the JMSTransport parser to return an XML bit stream containing the JMSTransport section of the message, so that it can be examined and processed.

Related information:

WebSphere MQ Version 7 Information Center online

JMS message selector:

A message selector allows a JMS consumer to be more selective about the messages that it receives from a particular topic or queue.

A message selector uses message properties and headers as criteria in conditional expressions. These expressions use Boolean logic to declare which messages are delivered to a client, such as the JMSInput node.

The following table demonstrates the construction of a message selector. It comprises an identifier, such as the *JMSPriority* header, or an application controlled property *myProperty1*. The selector string must specify an operator followed by a literal.

Element	Valid values
Identifiers	• Property or header field reference (such as *JMSPriority*, *myProperty1*) • The following values are not possible: NULL, TRUE, FALSE, NOT, AND, OR, BETWEEN, LIKE, IN, IS
Operators	AND, OR, LIKE, BETWEEN, =, <>, <,>, <=, >=, IS NULL, IS NOT NULL
Literals	• The two Boolean literals, TRUE and FALSE • Exact number literals that have no decimal point; for example, +25, -399, 40 • Approximate number literals. These literals can use scientific notation or decimal; for example, -21.4E4, 5E2, +34.4928

The JMSInput node provides a free format string *PropertySelector*, to specify selectors that filter or include application properties. The node also has properties for specific header properties, where the identifier is implicit and is generated by the node. For the header selectors, the operator and literal part of the string must be specified.

If more than one selector is specified the node generates a composite selector string, where the individual selector strings are concatenated with the `AND` operator, and each selector string part is wrapped with parentheses.

The following are examples for each of the selector properties:

Selector property	Description
PropertySelector	`OrderValue> 100.00` This string is used directly as shown.
TimeStamp	`BETWEEN 1057576423231 AND 10575788993265` Messages that are put between these two Java times only (where Java time is milliseconds since 01 Jan 1970) is delivered to the JMSInput node. In this case, the string generated is prefixed with the identifier *JMSTimestamp*.
Delivery Mode	`PERSISTENT` This setting means that only messages marked by the sender as being `PERSISTENT` should be delivered to the JMSInput node. In this case, the string that is generated is prefixed with the identifier *JMSDeliveryMode*.
Priority	`>= 5 AND <= 8` This setting means that only messages marked by the sender as having a priority `5`, `6`, `7`, or `8` are delivered to the JMSInput node. In this case, the string generated is prefixed with the identifier *JMSPriority*.
Message ID	`> WMBRK123456` This setting returns messages with a Message ID that is greater than the value specified. In this case, the string generated is prefixed with the identifier *JMSMessageID*.
Redelivered	`FALSE` This setting means that messages that have not been redelivered are received by the node. In this case, the string generated is prefixed with the identifier *JMSRedelivered*.
Correlation ID	`= WMBRKABCDEFG` This setting returns messages whose Correlation ID is equal to the value `WMBRKBABCDEFG`. In this case, the string generated is prefixed with the identifier *JMSCorrelationID*.

Related concepts:

"JMS Transactionality" on page 1705
JMS destinations that supply messages to an input node, or receive messages from an output node, can be sync-point coordinated as part of a message flow global transaction.

Related reference:

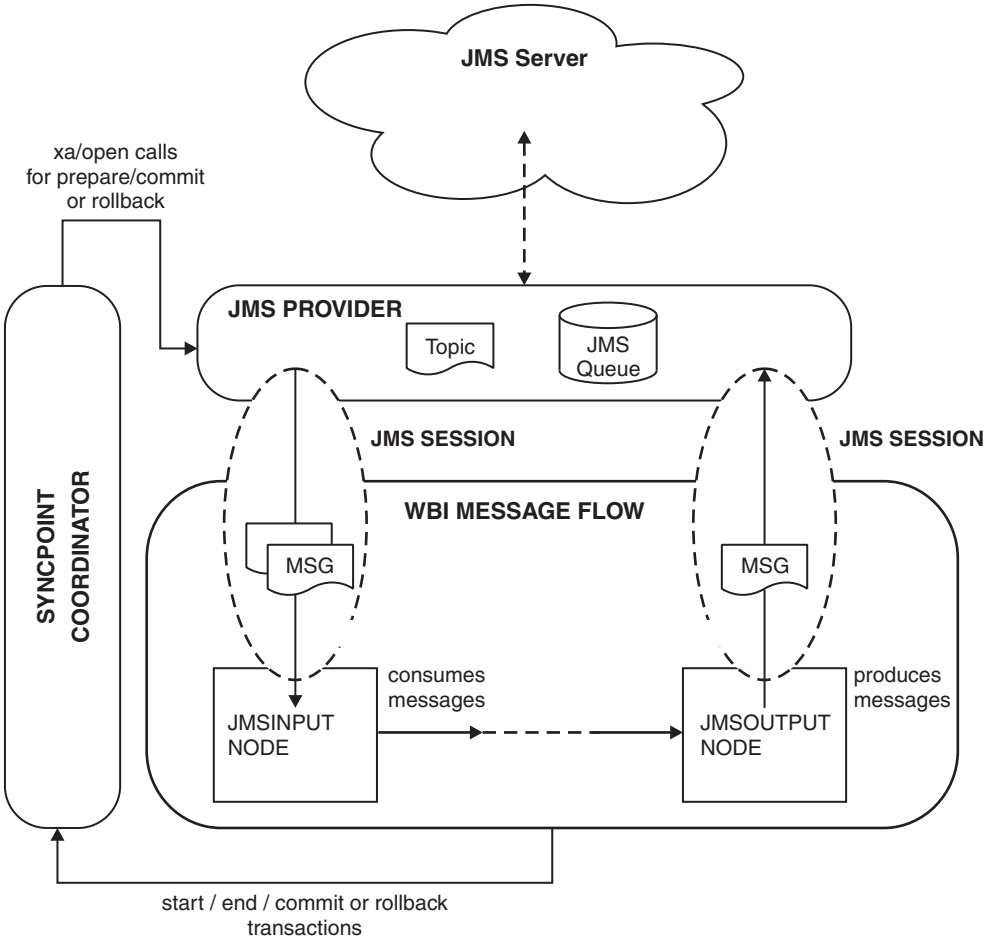
"JMS properties for application communication models" on page 1708
JMS clients can operate with both publish/subscribe and point-to-point messages. The publish/subscribe and point-to-point application communication models use virtual channels called *destinations*. In the publish/subscribe model, the destinations are *topics*. For the point-to-point model, the destinations are known as *queues*.

"JMS message domain properties" on page 1707
The JMSInput node can receive message payloads that correspond to all of the JMS message types that are specified in the JMS Specification, version 1.1.

JMS Transactionality:

JMS destinations that supply messages to an input node, or receive messages from an output node, can be sync-point coordinated as part of a message flow global transaction.

Transactions involving the sync-point coordinator



In this diagram, messages are consumed from a queue by a JMSInput node, and are produced to a JMS queue by a JMSOutput node. The nodes are connected to, and are in session with, a JMS provider. Any message flow input node can inform the external sync-point coordinator when a message flow transaction starts and ends, and whether any resources that have been affected by the flow should be committed or rolled back.

The sync-point coordinator sends XA/Open compliant requests to all participating resource managers to inform them to prepare. Any changes are either committed or rolled back. Resource managers, for example, WebSphere MQ, DB2 and any XA compliant JMS provider can participate in a global transaction.

The external sync-point coordinator is WebSphere MQ on operating systems other than z/OS, and RRS (Resource Recovery Services) on z/OS.

On z/OS, the only JMS provider that is supported is the IBM WebSphere MQ Java Client, and the only transport mode supported for that client is BIND mode.

Nodes that use JMS transport, such as the JMS and SOAP nodes, can participate in a global transaction only if the JMS provider to which they connect supports the XA/Open interface through the JMS `XAResource` Class. An example JMS provider is the WebSphere MQ Java Client.

You can specify a generic connection factory (`recoverXAQCF`) for recovery of XA coordinated transactions.

In-doubt transactions

In-doubt transactions can occur when a resource manager does not reply to a call from the sync-point manager, where the call is to commit or to rollback resources. During start up of the broker's WebSphere MQ queue manager, an initial recovery step is taken to ensure that any in-doubt transactions are resolved before the broker message flows start to process new input. A JMS provider that participates in broker global transactions is included in this recovery step.

On operating systems other than z/OS, WebSphere MQ requires an administration task to be carried out before deployment. This task registers a broker component, which is a shared library, with the queue manager by referring the shared library to a switch file.

When the broker's WebSphere MQ queue manager starts up, it loads the switch file. The switch file forwards XA/Open transaction calls from the sync-point coordinator to the JMS Provider. This ensures that the JMS resources that participate in the transaction can be coordinated in synchronization with other resource managers that are involved in the same transaction.

Additional configuration is required to enable global transaction support for the nodes using JMS transport; see "Configuring JMS and SOAP nodes to support global transactions" on page 1716.

Related tasks:

"Configuring JMS and SOAP nodes to support global transactions" on page 1716
To include nodes that use JMS transport, such as the JMS and SOAP nodes, in globally coordinated transactions, you must complete additional configuration.

Related reference:

"JMS message types" on page 1690
JMS defines six message interface types; a base message type and five subtypes. The message types are defined according to the type of the message *payload*, where the payload is the body of a message that holds the content.

"JMS message structure" on page 1688
JMS messages have a defined structure that includes headers and payloads.

"JNDI administered objects" on page 1702
JNDI (Java Naming and Directory Interface) is a standard Java extension that provides a uniform API for accessing various directory and naming services.

"JMSInput node" on page 4532
Use the JMSInput node to receive messages from JMS destinations. JMS destinations are accessed through a connection to a JMS provider.

"JMSOutput node" on page 4549
Use the JMSOutput node to send messages to JMS destinations.

"SOAPInput node" on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

"SOAPRequest node" on page 4828
Use the SOAPRequest node to send a SOAP request to the remote Web service.

"SOAPAsyncRequest node" on page 4750
Use the SOAPAsyncRequest node with the SOAPAsyncResponse node to construct a pair of message flows that call a Web service asynchronously.

JMS message domain properties:

The JMSInput node can receive message payloads that correspond to all of the JMS message types that are specified in the JMS Specification, version 1.1.

The JMS Specification is defined in Java Message Service Specification, version 1.1. For more information, see "JMS message types" on page 1690.

Set the JMSInput node properties to specify how the message payload is to be parsed. For more information, see "JMSInput node" on page 4532 and "JMS message payload and appropriate parser" on page 1698.

The JMSOutput node has no message domain properties.

Related concepts:

"JMS Transactionality" on page 1705
JMS destinations that supply messages to an input node, or receive messages from an output node, can be sync-point coordinated as part of a message flow global transaction.

"Specifying namespaces in the Message Type property" on page 1208
When using the MRM domain, the *Message Type* property is used to specify the name of the message.

"Multipart messages" on page 1191
A *multipart message* contains one or more other messages within its structure. The contained message is sometimes referred to as an *embedded message*.

Related reference:

"JMS properties for application communication models" on page 1708
JMS clients can operate with both publish/subscribe and point-to-point messages. The publish/subscribe and point-to-point application communication models use virtual channels called *destinations*. In the publish/subscribe model, the destinations are *topics*. For the point-to-point model, the destinations are known as *queues*.

"JMS message types" on page 1690
JMS defines six message interface types; a base message type and five subtypes. The message types are defined according to the type of the message *payload*, where the payload is the body of a message that holds the content.

"JMSInput node" on page 4532
Use the JMSInput node to receive messages from JMS destinations. JMS destinations are accessed through a connection to a JMS provider.

"JMS message selector" on page 1703
A message selector allows a JMS consumer to be more selective about the messages that it receives from a particular topic or queue.

"JMS message payload and appropriate parser" on page 1698
Configure the JMSInput node properties to specify the message domain that will be used to parse the JMS message payload.

JMS properties for application communication models:

JMS clients can operate with both publish/subscribe and point-to-point messages. The publish/subscribe and point-to-point application communication models use virtual channels called *destinations*. In the publish/subscribe model, the destinations are *topics*. For the point-to-point model, the destinations are known as *queues*.

The following application communication model properties can be configured for JMSInput and JMSOutput nodes:

Property	Description
Connection Factory Name	A string name that is passed to JNDI to look up the administered connection factory object. The connection factory object is used to create a connection to the JMS destination. • For a client operating as a publish/subscribe client, the connection factory name is for a TopicConnectionFactory. • For a client operating as a point-to-point client, the connection factory name is for a QueueConnectionFactory.
Subscription Topic	The string name that is passed to JNDI to look up the JMS topic destination. The topic is used to create a JMS session when the node is being used to process publish/subscribe messages.
Durable Subscription ID	This is a JMSInput node property only. It is a string identifier that is specified if the node is to subscribe to a durable subscription topic. A durable subscription is one that outlasts the client's connection to a message server. When a durable subscriber is disconnected from the server, the server stores messages that are published. Therefore, when the durable subscriber reconnects, the message server sends all the unexpired messages. Durable subscriptions cannot be unsubscribed from a message flow. A separate administration task unsubscribes a previously registered durable subscription. Some JMS providers supply an administration tool to perform this action.
Source Queue	The string name that is passed to JNDI to look up the JMS queue destination. The queue is used to create a JMS session when the node is being used to process point-to-point messages.

The `Subscription Topic` and `Source Queue` properties are mutually exclusive because they configure the node to work with either the publish/subscribe message model or the point-to-point message model.

A `Durable subscription ID` is not valid without a `Subscriber Topic` property.

Related concepts:

"Nodes for connectivity" on page 1028
WebSphere Message Broker supports direct connections from applications, and can send direct requests to other application endpoints. WebSphere Message Broker can also connect to various subsystems including WebSphere MQ, files, and databases, to read and write existing application data.

Related reference:

"JMS message domain properties" on page 1707
The JMSInput node can receive message payloads that correspond to all of the JMS message types that are specified in the JMS Specification, version 1.1.

"JMS message selector" on page 1703
A message selector allows a JMS consumer to be more selective about the messages that it receives from a particular topic or queue.

Working with JMS

Learn about the concepts and tasks involved in configuring message flows to support JMS messages.

Before you begin

Before you start: Read the following topics for an overview of the concepts related to processing JMS messages

- "WebSphere Broker JMS Transport" on page 1681
- "Java Message Service (JMS) API" on page 1680
- "JMS Transactionality" on page 1705
- "JMS message selector" on page 1703
- "JMS properties for application communication models" on page 1708
- "JMS message domain properties" on page 1707

About this task

The following areas are included:

- "Configuring resources for processing JMS messages" on page 1714
- "Troubleshooting JMS nodes" on page 1730

Connection to different JMS providers:

The JMSInput and JMSOutput nodes are compatible with all JMS providers that conform to the Java Message Service Specification, version 1.1.

If you want these nodes to participate in coordinated transactions, the JMS provider must support the XAResource interface, as defined in the Java Message Service Specification, version 1.1.

JBoss exception handling

If you use JBoss from a JMSInput node, set the `jmsAsyncExceptionHandling` property to true on the JMSProviders configurable service so that the broker can detect a failure in the JMS connection. This property is set to true by default on the provided JBoss configurable service definition.

Related concepts:

"JMS message transformation" on page 1684
The JMSInput and JMSOutput nodes expect JMS messages, and therefore expect a native JMS message tree representation.

Related tasks:

"Configuring JMS nodes to communicate with WebSphere Application Server service integration bus" on page 1710
You can configure a stand-alone JMS client and JMS nodes to communicate with service integration bus (SIBus) in WebSphere Application Server Version 6 and Version 7.

"Configuring JMS nodes to communicate with Oracle AQ" on page 1712
You can configure the JMS nodes to communicate with Oracle AQ (Oracle 11g and above). This communication requires an LDAP 3 compliant server to hold definitions for JNDI lookup by the JMS nodes.

Related reference:

"JMS message types" on page 1690
JMS defines six message interface types; a base message type and five subtypes. The message types are defined according to the type of the message *payload*, where

the payload is the body of a message that holds the content.

"JMS message structure" on page 1688
JMS messages have a defined structure that includes headers and payloads.

"JNDI administered objects" on page 1702
JNDI (Java Naming and Directory Interface) is a standard Java extension that provides a uniform API for accessing various directory and naming services.

"JMS message for output" on page 1700
When the JMSOutput node receives a JMS message, it calls the JMSTransport parser to return an XML bit stream containing the JMSTransport section of the message, so that it can be examined and processed.

"JMSInput node" on page 4532
Use the JMSInput node to receive messages from JMS destinations. JMS destinations are accessed through a connection to a JMS provider.

Configuring JMS nodes to communicate with WebSphere Application Server service integration bus:

You can configure a stand-alone JMS client and JMS nodes to communicate with service integration bus (SIBus) in WebSphere Application Server Version 6 and Version 7.

About this task

Procedure

1. Complete the following steps in WebSphere Application Server. For more information, see the WebSphere Application Server documentation.
   a. Create a messaging bus.
   b. Add a bus member.
   c. Restart the WebSphere Application Server server.
   d. Create a queue destination on the bus.
   e. Create a JMS queue on the default messaging provider.
   f. Create a Queue Connection Factory (QCF) on the default messaging provider.

      Ensure that the messaging provider URL is specified in the QCF definition, particularly if the JMS client and messaging bus are on different computers. The provider endpoint URL must have the following format:

      ```
      bus_member_host_name:7276:BootstrapBasicMessaging
      ```

      where 7276 is the default SIB endpoint address. Do not use 127.0.0.1 or localhost for the bus member host name.
2. Test the WebSphere Application Server configuration by using a stand-alone JMS client and completing the following steps.
   a. Put the following two JAR files in your class path: `com.ibm.ws.sib.client.thin.jms_7.0.0.jar` and `com.ibm.ws.ejb.thinclient_7.0.0.jar`.

      Copy these JAR files from the WebSphere Application Server Version 7 installation directory under the runtimes subdirectory. If you are using a non-IBM JRE, you also need the `com.ibm.ws.orb_7.0.0.jar` file.
   b. Ensure that the provider URL is set to `iiop://WAS_server_host_name:boot_strap_port`.
   c. Ensure that you specify the correct boot strap port.

d. Ensure that the Queue Connection Factory and JMS Queue properties are set to the values defined in the WebSphere Application Server configuration.
e. Compile the JMS client code.
f. Run the JMS client with the following IBM ORB debug parameters turned on.

```
java -Dcom.ibm.CORBA.Debug=true -Dcom.ibm.CORBA.CommTrace=true -Dcom.ibm.CORBA.D ebug.Outpu
```

This command produces CORBA debug output in the `client.log` file in the same directory.

3. Complete the following steps in WebSphere Message Broker.
   a. Stop the broker.
   b. Create a directory (for example, `c:\WebSphere_WAS_Client`) and copy the following two JAR files from WebSphere Application Server Version 7 Thin Client for JMS.
      - `com.ibm.ws.sib.client.thin.jms_7.0.0.jar`
      - `com.ibm.ws.ejb.thinclient_7.0.0.jar`

      Alternatively, you can copy these files from the WebSphere Application Server installation directory *WAS home*`/runtimes`.
   c. Configure the JMS service in WebSphere Message Broker by using the **mqsichangeproperties** command. The JMS provider WebSphere_WAS_Client exists; therefore you can change the client JAR file path for that provider.

```
mqsichangeproperties broker_name -c JMSProviders -o WebSphere_WAS_Client -n jarsURL -v WAS_
```

   d. Configure the JMSInput node as shown in the following example. For more information about these properties, see "JMSInput node" on page 4532.
      - Specify the name of the JMS provider; for example, Client for WebSphere Application Server.
      - Specify the initial context factory; for example, `com.ibm.websphere.naming.WsnInitialContextFactory`.
      - Specify the location of the JNDI bindings in the format `iiop://`*WAS_server_host_name*`:`*WAS_server_boot_strap_port*.
      - Set the connection factory name to QCF.
   e. Ensure that the JMS connection has been established before the message flow starts by using the Windows Event Viewer.

Related concepts:

"Connection to different JMS providers" on page 1709
The JMSInput and JMSOutput nodes are compatible with all JMS providers that conform to the Java Message Service Specification, version 1.1.

Related tasks:

"Configuring the broker to enable a JMS provider's proprietary API" on page 748
Some JMS providers provide an alternative interface to the standard JMS specification for particular JMS API calls. In these cases, IBM supplies a Java class to interface with that proprietary API.

"Working with JMS" on page 1709
Learn about the concepts and tasks involved in configuring message flows to support JMS messages.

Related reference:

"JMSInput node" on page 4532
Use the JMSInput node to receive messages from JMS destinations. JMS destinations are accessed through a connection to a JMS provider.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

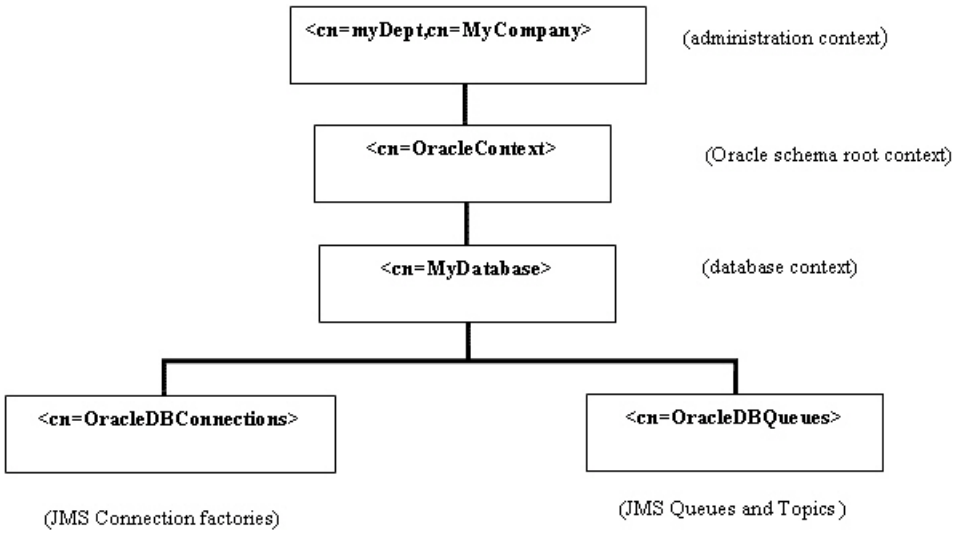
Configuring JMS nodes to communicate with Oracle AQ:

You can configure the JMS nodes to communicate with Oracle AQ (Oracle 11g and above). This communication requires an LDAP 3 compliant server to hold definitions for JNDI lookup by the JMS nodes.

About this task

Procedure

1. Complete the following steps in Oracle AQ, referring to Oracle documentation for the specific details of each step.
   a. You must install the Oracle Internet Directory Server (OID) to host the JNDI administered objects for Oracle AQ.
      - You can configure Oracle AQ to register JMS connection factories and destinations (queues and topics) automatically with the OID server when these JMS connection factories and destinations are created in the Oracle database.
   b. Create the database tables to hold the JMS queues and topics.
   c. Create the JMS queues and topics and associate them with the tables created in step 1b.
2. Add definitions for the JMS connections to the LDAP server to permit the broker JMS nodes to complete JND lookup and connect to the Oracle AQ server.
   a. Register JMS connection factories with the OID LDAP server by using the administrative tools provided by Oracle.
   b. The following diagram describes the shape of the directory tree for JNDI administered objects for Oracle AQ.



3. Copy the Oracle AQ JMS client JAR files to a local directory that is accessible by the broker.
   - The `aqapi.jar` file is found on the Oracle AQ server in directory *oracle_install_path*`/rdbms/jlib`.
   - The `ojdbc5.jar` file is found on the Oracle AQ server in directory *oracle_install_path*`/jdbc/lib`.
   - The orai18n.jar found on the Oracle AQ server in directory *oracle_install_path*`/jlib`.
4. Modify the JMSProviders configurable service for Oracle AQ; for example:

```
mqsichangeproperties MyBroker –c JMSProviders –o Oracle_AQ
   -n jarsURL,
      InitialContextFactory,
      jndiBindingsLocation
   -v location of the Oracle Jars,
      com.sun.jndi.ldap.LdapCtxFactory,
      ldap://LDAP_server_address:LDAP_listener_port
```

5. Configure the properties on the JMS Connection tab of the JMS node (input, output, or reply node) as shown in the following example. For more information about these properties, see "JMSInput node" on page 4532.
   - Set the `JMS provider name` property to `Oracle_AQ`.
   - Set the `Initial context factory` property; for example:

     ```
     com.sun.jndi.ldap.LdapCtxFactory
     ```

     For all nodes that refer to the JMSProviders configurable service, if this property is set on the configurable service, it overrides the property that is set on the node.
   - Set the `Location JNDI bindings` property; for example:

     ```
     ldap://LDAP_server_address:LDAP_listener_port
     ```

     For all nodes that refer to the JMSProviders configurable service, if this property is set on the configurable service, it overrides the property that is set on the node.
   - Set the `Connection factory name` property. This name must be the fully qualified path in the LDAP directory; for example:

     ```
     cn=QCF,cn=oracledbconnections,cn=ORCL,cn=OracleContext,
     ou=MyDept,o=MyCompany
     ```

     Where
     - cn=QCF is the JMS connection factory name.
     - cn=oracledbconnections is the branch for JMS connection factory definitions.
     - cn=ORCL is the Oracle AQ database name.
     - cn=OracleContext is the root of the Oracle RDBMS schema.
     - ou=MyDept, o=MyCompany is the installation-specific LDAP administrative context.
6. On the Basic tab, configure the JMS destinations (queue or topic) properties.
   - Set the `Source queue` property on the JMSInput node. This queue must be the fully qualified path in the LDAP directory; for example:

     ```
     cn=JMS.Queue,cn=oracleDBQueues,cn=ORCL,cn=OracleContext,
     ou=MyDept,o=MyCompany
     ```

     Where
     - cn=JMS.Queue is the JMS queue.
     - cn=oracleDBQueues is the branch for JMS queues and topic definitions.
     - cn=ORCL is the Oracle AQ database name.

- cn=OracleContext, is the root of the Oracle RDBMS schema.
- ou=MyDept, o=MyCompany is the installation-specific LDAP administrative context.

7. Before the message flow starts, ensure that the JMS connection has been established by using the Windows Event Viewer.

Related concepts:

"Connection to different JMS providers" on page 1709
The JMSInput and JMSOutput nodes are compatible with all JMS providers that conform to the Java Message Service Specification, version 1.1.

Related tasks:

"Configuring the broker to enable a JMS provider's proprietary API" on page 748
Some JMS providers provide an alternative interface to the standard JMS specification for particular JMS API calls. In these cases, IBM supplies a Java class to interface with that proprietary API.

"Working with JMS" on page 1709
Learn about the concepts and tasks involved in configuring message flows to support JMS messages.

Related reference:

"JMSInput node" on page 4532
Use the JMSInput node to receive messages from JMS destinations. JMS destinations are accessed through a connection to a JMS provider.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

Configuring resources for processing JMS messages:

A number of nodes are provided in WebSphere Message Broker for processing and routing JMS messages. Follow the links in this topic to find out how to configure the JMS nodes and broker resources for processing JMS messages.

About this task

WebSphere Message Broker provides the following nodes for working with JMS messages:

JMSInput node
Use a JMSInput node if the messages are received from a JMS application.

JMSOutput node
Use a JMSOutput node if the messages are sent to a JMS destination.

JMSReply node
The JMSReply node has a similar function to the JMSOutput node, but the JMSReply node sends JMS messages only to the reply destination that is supplied in the JMSReplyTo header field of the JMS message tree. Use the JMSReply node to treat a JMS message that is produced from a message flow as a reply to a JMS input message, and when you have no other routing requirements.

JMSMQTransform node
Use the JMSMQTransform node to transform a message with a JMS

message tree into a message that has a tree structure that is compatible with the format of messages that are produced by the WebSphere MQ JMS provider.

The JMSMQTransform node can be used to send messages to existing message flows and to interoperate with WebSphere MQ JMS and WebSphere MQ Publish/Subscribe.

MQJMSTransform node
Use the MQJMSTransform node to receive messages that have a WebSphere MQ JMS provider message tree format, and transform them into a format that is compatible with messages that are to be sent to JMS destinations.

You can use the MQJMSTransform node to send messages to existing message flows and to interoperate with WebSphere MQ JMS and WebSphere MQ Publish/Subscribe.

JMSHeader node
Use a JMSHeader node to change JMS Header_Values properties, or add, modify, or delete JMS Application properties without programming.

SOAPInput node
Use a SOAPInput node for SOAP messages received from a JMS application.

SOAPReply node
The SOAPReply node sends SOAP messages using JMS transport only to the reply destination specified in the received message. Use the SOAPReply node to treat a JMS message that is produced from a message flow as a reply to a JMS input message.

SOAPRequest node
Use a SOAPRequest node to send a SOAP request to a remote Web service. This node is a synchronous request and response node, and blocks after sending the request until the response is received.

SOAPAsyncRequest and SOAPAsyncResponse nodes
Use a SOAPAsyncRequest node with a SOAPAsyncResponse node to construct a pair of message flows that call a Web service asynchronously.

To use JMS nodes in your message flows, there are additional configuration steps in the broker environment that you might need to complete. See the following topics for information about additional configuration tasks that you might need to complete:

- "Configuring JMS and SOAP nodes to support global transactions" on page 1716
- "Windows systems: modifying the queue manager authorization" on page 1723
- "Securing JMS connections and JNDI lookups" on page 1725
- "Configuring the broker to enable a JMS provider's proprietary API" on page 748
- "Processing bytes messages with JMS nodes" on page 1729

Related tasks:

"Working with JMS" on page 1709
Learn about the concepts and tasks involved in configuring message flows to support JMS messages.

"Configuring resources for processing JMS messages" on page 1714
A number of nodes are provided in WebSphere Message Broker for processing and routing JMS messages. Follow the links in this topic to find out how to configure the JMS nodes and broker resources for processing JMS messages.

Related reference:

"JMSHeader node" on page 4529
Use the JMSHeader node to modify contents of the JMS Header_Values and Application properties so that you can control the node's output without programming.

"JMSInput node" on page 4532
Use the JMSInput node to receive messages from JMS destinations. JMS destinations are accessed through a connection to a JMS provider.

"JMSMQTransform node" on page 4547
Use the JMSMQTransform node to transform a message with a JMS message tree into a message that has a message tree structure that is compatible with the format of messages that are produced by the WebSphere MQ JMS provider.

"JMSOutput node" on page 4549
Use the JMSOutput node to send messages to JMS destinations.

"JMSReply node" on page 4562
Use the JMSReply node to send messages to JMS destinations.

"MQJMSTransform node" on page 4610
Use the MQJMSTransform node to receive messages that have a WebSphere MQ JMS provider message tree format, and transform them into a format that is compatible with messages that are to be sent to JMS destinations.

"SOAPInput node" on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

"SOAPReply node" on page 4819
Use the SOAPReply node to send SOAP messages from the broker to the originating client in response to a message received by a SOAPInput node.

"SOAPRequest node" on page 4828
Use the SOAPRequest node to send a SOAP request to the remote Web service.

"SOAPAsyncRequest node" on page 4750
Use the SOAPAsyncRequest node with the SOAPAsyncResponse node to construct a pair of message flows that call a Web service asynchronously.

"SOAPAsyncResponse node" on page 4777
Use the SOAPAsyncResponse node in conjunction with the SOAPAsyncRequest node to construct a pair of message flows that call a Web service asynchronously.

"Using LocalEnvironment variables with JMSOutput and JMSReply nodes" on page 4242
The LocalEnvironment data elements related to the processing of JMS Messages in the JMSOutput and JMSReply nodes.

Configuring JMS and SOAP nodes to support global transactions:

To include nodes that use JMS transport, such as the JMS and SOAP nodes, in globally coordinated transactions, you must complete additional configuration.

About this task

If you require transaction coordination, choose a JMS provider that conforms to the Java Message Service Specification, version 1.1 and that supports the JMS XAResource API through the JMS session.

If the message designer has specified a non-XA-compliant provider, the non-transactional mode only is supported. In this case, you must set the Transaction mode property to None for all JMS and SOAP nodes that use JMS transport.

To configure the nodes:

Procedure

1. Switch to the Broker Application Development perspective.
2. Set the message flow property Coordinated Transaction to yes in the BAR file properties.
3. For each node that uses JMS transport that is required in the global transaction, set the Transaction mode property to Global in the message flow editor.
4. Create a Queue Connection Factory and use either the default name, *recoverXAQCF*, or supply your own name. See the JMSInput or JMSOutput node for further details about creating JNDI administered objects.
5. On distributed systems, you must set up a stanza for each JMS provider that you want to use, before deployment.

   The following table shows the JMS provider switch files that are provided on each operating system.

Platform	32-bit file	64-bit file
AIX		`libJMSSwitch.so`
HP-Itanium		`libJMSSwitch.so`
Linux on POWER		`libJMSSwitch.so`
Linux on IBM z Systems		`libJMSSwitch.so`
Linux on x86	`libJMSSwitch.so`	
Linux on x86-64		`libJMSSwitch.so`
Solaris on SPARC		`libJMSSwitch.so`
Solaris on x86-64		`libJMSSwitch.so`
Windows on x86	`JMSSwitch.dll`	
Windows on x86-64	`JMSSwitch32.dll`	`JMSSwitch.dll`

   Select the appropriate link for details of this task on the operating system, or systems, that your enterprise uses:

   - Linux UNIX Linux and UNIX systems
   - Windows Windows systems

     On Windows only, you must also modify the queue manager authorization, as described in "Windows systems: modifying the queue manager authorization" on page 1723.

   For further information, see the following topics:

   - "Configuring for coordinated transactions" on page 4535 in the JMSInput node topic
   - "Configuring for coordinated transactions" on page 4553 in the JMSOutput node topic

   z/OS On z/OS, the only JMS provider that is supported is the IBM WebSphere MQ Java Client, and the only transport mode supported for that client is BIND mode; no further configuration steps are required.

What to do next

The JMS provider might supply additional JAR files that are required for transactional support; for more informaiton, see the documentation that is supplied with the JMS provider. For example, on distributed systems, the WebSphere MQ JMS provider supplies an extra JAR file, `com.ibm.mqetclient.jar`.

You must add any additional JAR files to the broker `shared_classes` directory:

- Linux UNIX On Linux and UNIX: `var/mqsi/shared-classes`.
- Windows On Windows, `%ALLUSERSPROFILE%\Application Data\IBM\MQSI\shared-classes`, where `%ALLUSERSPROFILE%` is the environment variable that defines the system working directory. The default directory depends on the operating system:
  - On Windows XP and Windows Server 2003: `C:\Documents and Settings\All Users\Application Data\IBM\MQSI\shared-classes`
  - On Windows Vista and Windows Server 2008: `C:\ProgramData\IBM\MQSI\shared-classes`

  The value can vary for different computers.

For more information, see the section about making the JMS provider client available to the JMS nodes in "JMSInput node" on page 4532.

Optional: To secure the JMS connection factory, the JNDI bindings, or both, see "Securing JMS connections and JNDI lookups" on page 1725.

Related concepts:

"JMS Transactionality" on page 1705
JMS destinations that supply messages to an input node, or receive messages from an output node, can be sync-point coordinated as part of a message flow global transaction.

Related tasks:

"Linux and UNIX systems: configuring the queue manager to coordinate JMS resources" on page 1719
Define a stanza in the broker's queue manager `qm.ini` file for each new JMS provider, where the JMS provider can be specified by a JMS node included in a message flow that is running on the broker.

"Windows systems: configuring the queue manager to coordinate JMS resources" on page 1721
Use WebSphere MQ Explorer to configure the XA resource managers for the queue manager.

"Windows systems: modifying the queue manager authorization" on page 1723
Authorize the broker and queue manager to access shared resources that are associated with the JMS provider.

"Securing JMS connections and JNDI lookups" on page 1725
If you want additional security for JMS connectivity and the JMS nodes or SOAP nodes using JMS transport, two configuration options are supported.

Related reference:

"JMSInput node" on page 4532
Use the JMSInput node to receive messages from JMS destinations. JMS destinations are accessed through a connection to a JMS provider.

"JMSOutput node" on page 4549
Use the JMSOutput node to send messages to JMS destinations.

"SOAPInput node" on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

"SOAPRequest node" on page 4828
Use the SOAPRequest node to send a SOAP request to the remote Web service.

"SOAPAsyncRequest node" on page 4750
Use the SOAPAsyncRequest node with the SOAPAsyncResponse node to construct a pair of message flows that call a Web service asynchronously.

Linux and UNIX systems: configuring the queue manager to coordinate JMS resources:

Define a stanza in the broker's queue manager `qm.ini` file for each new JMS provider, where the JMS provider can be specified by a JMS node included in a message flow that is running on the broker.

About this task

The parameters that are defined in `XAOpenString` are comma delimited and positional. Represent missing optional parameters by a comma if you include other parameters later in the string.

The following stanza entry is an example that you can add when using WebSphere MQ Java as the JMS provider:

```
XAResourceManager:
    Name=WBIWMQJMS
    SwitchFile=/install_dir/lib/JMSSwitch.so
    XAOpenString=<Initial Context Factory>,
              <location of JNDI bindings>'
              <LDAP Principal>,
              <LDAP Credentials>,
              <Recovery Connection Factory Name>,
              <JMS Principal>,
              <JMS Credentials>
              ThreadOfControl=THREAD
```

where:

install_dir
Is the location of the WebSphere Message Broker installation. This value is mandatory where the LDAP parameters are omitted, but a user-defined Queue Connection Factory is specified for recovery.

<Initial Context Factory>
Is the Initial Context Factory identifier for the JMS provider; this value is required.

<Location of JNDI bindings>
Is either the file path to the bindings file, or the LDAP directory location of the JNDI administered objects that can be used to create an initial context factory for the JMS connection. When supplying the file path to the bindings file, do not include the file name. See the JMSInput or JMSOutput node for further details on creating the JNDI administered objects; this value is required.

<LDAP Principal>
Is an optional parameter used to specify the principal (user ID) that might be required when an LDAP database is used to hold the JNDI administered objects.

<LDAP Credentials>
Is an optional parameter used to specify the Credentials (password) that might be required if a password protected LDAP database is used to hold the JNDI administered objects.

<Recovery Connection Factory Name>
Is an optional parameter used to specify the name of a Queue Connection Factory object in the JNDI administered objects for recovery purposes, when the non default name is required.

<JMS Principal>
Is an optional parameter for the user ID required to connect to a JMS provider, using a secure JMS Connection Factory.

<JMS Credentials>
Is an optional parameter for the password required to connect to the same JMS provider in conjunction with the JMS principal.

Switch files are installed in the *install_dir* `/lib` directory. To simplify the contents of the `qm.ini` file, create a symbolic link to the switch file for the queue manager to retrieve from `/var/mqm/exits` (for 32-bit brokers) or `/var/mqm/exits64` (for 64-bit brokers). For example:

```
ln -s install_dir/lib/libJMSSwitch.so /var/mqm/exits/JMSSwitch

ln -s install_dir/lib/libJMSSwitch.so /var/mqm/exits64/JMSSwitch
```

If you create a link for both 32-bit and 64-bit switch files on a single computer, ensure that you specify the same name in `/exits` and in `/exits64`, as shown in the example.

The values for the Initial Context factory and Location of JNDI bindings in the stanza must match the values that you specified in the JMS or SOAP nodes in the message flows.

All LDAP parameters must match the values that you specified on the **mqsicreatebroker** or **mqsichangebroker** command.

The Recovery Factory Name must match a Queue Connection Factory name that is created in the JNDI administered objects. If you do not specify a name, a default factory called `recoverXAQCF` is used. In either case, this value must refer to a JNDI administered object that has already been created.

The JMS Principal and JMS Credentials must be configured together.

The following example shows the format of a stanza in the `qm.ini` file that describes a JMS provider for global transactions:

```
XAResourceManager:
    Name=XAJMS_PROVIDER1
    SwitchFile=/opt/var/mqsi/lib/JMSSwitch.so
    XAOpenString= com.sun.jndi.fscontext.RefFSContextFactory,
              /Bindings/JMSProvider1_Bindings_Directory,
              ,
              ,
              ,
              myJMSuser1,
              passwd
              ThreadOfControl=THREAD
```

where:

XAJMS_PROVIDER1
Is the user-defined name for the resource manager

/opt/var/mqsi
Is the <Installation Path>

com.sun.jndi.fscontext.RefFSContextFactory
Is the <Initial Context Factory>

/Bindings/JMSProvider1_Bindings_Directory
Is the location of the bindings

myJMSuser1
Is the <JMS Principal>

passwd
Is the password used in <JMS Credentials>

In this example, the optional fields `<LDAP Principal>`, `<LDAP Credentials>`, and `<Recovery Connection Factory Name>` are not required, therefore the positional comma delimiters only are configured in the `XAOpenString` stanza.

Related tasks:

"Configuring JMS and SOAP nodes to support global transactions" on page 1716
To include nodes that use JMS transport, such as the JMS and SOAP nodes, in globally coordinated transactions, you must complete additional configuration.

Related reference:

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

"JMSInput node" on page 4532
Use the JMSInput node to receive messages from JMS destinations. JMS destinations are accessed through a connection to a JMS provider.

"JMSOutput node" on page 4549
Use the JMSOutput node to send messages to JMS destinations.

"SOAPInput node" on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

"SOAPRequest node" on page 4828
Use the SOAPRequest node to send a SOAP request to the remote Web service.

"SOAPAsyncRequest node" on page 4750
Use the SOAPAsyncRequest node with the SOAPAsyncResponse node to construct a pair of message flows that call a Web service asynchronously.

Windows systems: configuring the queue manager to coordinate JMS resources:

Use WebSphere MQ Explorer to configure the XA resource managers for the queue manager.

About this task

Complete the following steps:

Procedure

1. Open WebSphere MQ Explorer.
2. Select the queue manager for your broker and click **Properties**.
3. Select **XA resource managers** in the left pane and click **Add**.
4. Complete the fields to define a new resource manager:
   - **Name**: Enter the name of the resource manager; for example, WBIWMQJMS.
   - **SwitchFile**: On Windows on x86, enter the full path of the switch file; for example, *install_dir*\bin\JMSSwitch.dll. On Windows on x86-64, enter **JMSSwitch**.
   - **XAOpenString**: Enter the following values, which are comma delimited and positional. Represent missing optional parameters by a comma if you include other parameters later in the string.

     Initial Context Factory
     The Initial Context Factory identifier for the JMS provider; this value is required.

     Location of JNDI bindings
     Either the file path to the bindings file, or the LDAP directory location of the JNDI administered objects that can be used to create an initial context factory for the JMS connection. If you supply the file path to the bindings file, do not include the file name. See the JMSInput or JMSOutput node for further details about creating the JNDI administered objects; this value is required.

     LDAP Principal
     Optional: The principal (user ID) that might be required when an LDAP database is used to hold the JNDI administered objects.

     LDAP Credentials
     Optional: The credentials (password) that might be required if a password protected LDAP database is used to hold the JNDI administered objects.

     Recovery Connection Factory Name
     Optional: The name of a Queue Connection Factory object in the JNDI administered objects for recovery purposes, when the non default name is required.

     JMS Principal
     The user ID that is required to connect to a JMS provider, using a secure JMS Connection Factory.

     JMS Credentials
     The password that is required to connect to the same JMS provider in conjunction with the JMS principal.

     The values for the Initial Context factory and Location of JNDI bindings in the stanza must match the values that you specified in the JMS or SOAP nodes in the message flows.

     All LDAP parameters must match the values that you specified on the **mqsicreatebroker** or **mqsichangebroker** command.

     The Recovery Factory Name must match a Queue Connection Factory name that is created in the JNDI administered objects. If you do not specify a name, a default factory called `recoverXAQCF` is used. In either case, this value must refer to a JNDI administered object that has already been created.

     The JMS Principal and JMS Credentials must be configured together.

- **XACloseString**: Leave this field blank.
- **ThreadOfControl**: Set the value `Thread`.

5. Click **OK** to complete the XA resource manager definition.
6. Click **OK** to close the queue manager properties dialog.
7. Click **File** > **Exit** to close WebSphere MQ Explorer.
8. On Windows on x86, copy the switch file (for example, `JMSSwitch.dll`) to the `\exits` subdirectory in the WebSphere MQ installation directory. On Windows on x86-64, copy the switch file `JMSSwitch32.dll` to the `\exits` subdirectory in the WebSphere MQ installation directory, and rename it to `JMSSwitch.dll`. Copy the switch file `JMSSwitch.dll` to the `\exits64` subdirectory in the WebSphere MQ installation directory.

What to do next

Next: modify the queue manager authorization.

Related tasks:

"Configuring JMS and SOAP nodes to support global transactions" on page 1716
To include nodes that use JMS transport, such as the JMS and SOAP nodes, in globally coordinated transactions, you must complete additional configuration.

"Windows systems: modifying the queue manager authorization"
Authorize the broker and queue manager to access shared resources that are associated with the JMS provider.

Related reference:

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

"JMSInput node" on page 4532
Use the JMSInput node to receive messages from JMS destinations. JMS destinations are accessed through a connection to a JMS provider.

"JMSOutput node" on page 4549
Use the JMSOutput node to send messages to JMS destinations.

"SOAPInput node" on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

"SOAPRequest node" on page 4828
Use the SOAPRequest node to send a SOAP request to the remote Web service.

"SOAPAsyncRequest node" on page 4750
Use the SOAPAsyncRequest node with the SOAPAsyncResponse node to construct a pair of message flows that call a Web service asynchronously.

Windows systems: modifying the queue manager authorization:

Authorize the broker and queue manager to access shared resources that are associated with the JMS provider.

Before you begin

Before you start:

Set up the JMSProviders configurable service; for more information, see “Making the JMS provider client available to the JMS nodes” on page 4534 (in the JMSInput node topic) or “Making the JMS provider client available to the JMS nodes” on page 4550 (in the JMSOutput node topic).

About this task

Complete the following steps on the Windows system on which the broker is running:

Procedure

1. If you defined the broker queue manager when you created the broker by running the **mqsicreatebroker** command, the two components share the same administrative ID, defined as the broker service ID, and you do not have to take any further action.
2. If you specified an existing queue manager when you created the broker, check that its administrative ID is the same ID as that used for the service ID of the broker. If the ID is not the same, change the queue manager ID to be the same as the broker service ID:
   a. Click **Start** > **Run** and enter dcomcnfg. The Component Services window opens.
   b. In the left pane, under Console Root, expand **Component Services** > **Computers** > **My Computer** and click **DCOM Config**.
   c. In the right pane, under DCOM Config, right-click the WebSphere MQ service labelled **IBM MQSeries Services**, and click **Properties**.
   d. Click the **Identity** tab.
   e. Select **This user** and enter the user ID and password for the broker service ID to associate that ID with the queue manager.
   f. Click **OK** to confirm the change.

Related tasks:

“Configuring JMS and SOAP nodes to support global transactions” on page 1716
To include nodes that use JMS transport, such as the JMS and SOAP nodes, in globally coordinated transactions, you must complete additional configuration.

Related reference:

“**mqsicreatebroker** command” on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

“**mqsichangebroker** command” on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

“JMSInput node” on page 4532
Use the JMSInput node to receive messages from JMS destinations. JMS destinations are accessed through a connection to a JMS provider.

“JMSOutput node” on page 4549
Use the JMSOutput node to send messages to JMS destinations.

“SOAPInput node” on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

“SOAPRequest node” on page 4828
Use the SOAPRequest node to send a SOAP request to the remote Web service.

"SOAPAsyncRequest node" on page 4750
Use the SOAPAsyncRequest node with the SOAPAsyncResponse node to construct a pair of message flows that call a Web service asynchronously.

Securing JMS connections and JNDI lookups:

If you want additional security for JMS connectivity and the JMS nodes or SOAP nodes using JMS transport, two configuration options are supported.

About this task

When you include JMS nodes in your message flow, you can optionally secure JMS connection resources. You can secure one, both, or neither of these options, depending on the level of security and access that you want to enforce.

Procedure

1. To secure a JMS connection:
   a. Specify the Connection Factory Name property on the node. You must set this property for every node using JMS transport.
   b. Use the **mqsisetdbparms** command to authorize the user ID and password for the specified connection factory. For example:

      ```
      mqsisetdbparms MyBroker1 -n jms::tcf1 -u myuserid -p secret
      ```

      where `tcf1` is the name of the connection factory that matches the node property that you set.
2. To secure JNDI bindings lookups:
   a. Specify the Initial Context Factory property on the node. You must set this property for every node using JMS transport.
   b. Use the **mqsisetdbparms** command to authorize the user ID and password for the specified context factory. For example:

      ```
      mqsisetdbparms MyBroker1 -n jndi::com.sun.jndi.fscontext.RefFSContextFactory
           -u myuserid -p secret
      ```

      where `com.sun.jndi.fscontext.RefFSContextFactory` is the name of the initial context factory that you set.

Related concepts:

"JMS Transactionality" on page 1705
JMS destinations that supply messages to an input node, or receive messages from an output node, can be sync-point coordinated as part of a message flow global transaction.

Related tasks:

"Configuring JMS and SOAP nodes to support global transactions" on page 1716
To include nodes that use JMS transport, such as the JMS and SOAP nodes, in globally coordinated transactions, you must complete additional configuration.

"Configuring the broker to enable a JMS provider's proprietary API" on page 748
Some JMS providers provide an alternative interface to the standard JMS specification for particular JMS API calls. In these cases, IBM supplies a Java class to interface with that proprietary API.

Related reference:

"JMSInput node" on page 4532
Use the JMSInput node to receive messages from JMS destinations. JMS destinations are accessed through a connection to a JMS provider.

"JMSOutput node" on page 4549
Use the JMSOutput node to send messages to JMS destinations.

"JMSReply node" on page 4562
Use the JMSReply node to send messages to JMS destinations.

"SOAPInput node" on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

"SOAPRequest node" on page 4828
Use the SOAPRequest node to send a SOAP request to the remote Web service.

"SOAPAsyncRequest node" on page 4750
Use the SOAPAsyncRequest node with the SOAPAsyncResponse node to construct a pair of message flows that call a Web service asynchronously.

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

Configuring the broker to enable a JMS provider's proprietary API:

Some JMS providers provide an alternative interface to the standard JMS specification for particular JMS API calls. In these cases, IBM supplies a Java class to interface with that proprietary API.

About this task

For example, BEA WebLogic uses a component called a *Client Interposed Transaction Manager* to allow a JMS client to obtain a reference to the XAResource that is associated with a user transaction.

If the WebSphere Message Broker JMS nodes use BEA WebLogic as the JMS provider, and the nodes must participate in a globally coordinated message flow, you must modify the configurable services properties that are associated with that vendor. The following table shows the properties that have been added to the configurable service for BEA WebLogic.

JMS provider	Property	Purpose	Default value
BEA_WebLogic	`proprietaryAPIHandler`	The name of the IBM supplied Java class to interface with a JMS provider's proprietary API.	`com.ibm.broker.apihandler.BEAWebLogicAPIHandler`
	`proprietaryAPIAttr1`	The Initial Context Factory class name for the vendor	`weblogic.jndi.WLInitialContextFactory`
	`proprietaryAPIAttr2`	The URL of the WebLogic bindings	*URL JNDI bindings*
	`proprietaryAPIAttr3`	The DNS name of the JMS server	*Server name*

In the list of JMS provider configurable services, the name of the IBM supplied Java class is set to the default value for the `proprietaryAPIHandler` property. Typically, you do not need to change this value, unless you are instructed to do so by an IBM Service team representative.

Procedure

- Use the **mqsichangeproperties** command to modify values of the properties for this JMS provider.

  The following example shows how to change the values of the properties `proprietaryAPIAttr2` and `proprietaryAPIAttr3` for the JMS provider

configurable service definition called BEA_Weblogic, where these properties represent the URL of the WebLogic bindings and the DNS Server name of the BEA WebLogic JMS Server:

```
mqsichangeproperties MB7BROKER -c JMSProviders -o BEA_Weblogic
-n proprietaryAPIAttr2,proprietaryAPIAttr3 -v t3://9.20.94.16:7001,BEAServerName
```

- Use the **mqsireportproperties** command to display the properties for a JMS provider.

  The following example shows how to display the properties for all the broker's JMS provider resources (the default JMS provider resources and those configurable services that are defined with the **mqsicreateconfigurableservice** command):

  ```
  mqsireportproperties MB7BROKER -c JMSProviders -o BEA_WebLogic –r
  ```

  The result of this command has the following format:

  ```
  ReportableEntityName=''
  JMSProviders
    BEA_Weblogic=''
      jarsURL='default_Path'
      nativeLibs='default_Path'
      proprietaryAPIAttr1='weblogic.jndi.WLInitialContextFactory'
      proprietaryAPIAttr2='t3://9.20.94.16:7001'
      proprietaryAPIAttr3='BEAServerName'
      proprietaryAPIAttr4='default_none'
      proprietaryAPIAttr5='default_none'
      proprietaryAPIHandler='com.ibm.broker.apihandler.BEAWebLogicAPIHandler'
  ```

  The default location for the JMS provider JAR files is the broker's shared-classes directory. You can specify an alternative location for the JAR files by using the **mqsichangeproperties** command, as shown in the following example:

  ```
  mqsichangeproperties MB7BROKER -c JMSProviders -o BEA_WebLogic -n jarsURL
  -v /var/mqsi/WebLogic
  ```

  On Windows, the file location cannot be a mapped network drive on a remote Windows computer; the directory must be local or on a Storage Area Network (SAN) disk.

- Use the **mqsicreateconfigurableservice** command to add a JMS provider.

  The following example shows how to add a JMS provider called BEAV91 for broker MB7BROKER, specifying the name of an IBM supplied Java class called `com.ibm.broker.apihandler.BEAWebLogicAPIHandler` to handle vendor-specific API calls:

  ```
  mqsicreateconfigurableservice MB7BROKER -c JMSProviders -o BEAV91
  -n proprietaryAPIHandler,proprietaryAPIAttr1,proprietaryAPIAttr2,proprietaryAPIAttr3
  –v com.ibm.broker.apihandler.BEAWebLogicAPIHandler,weblogic.jndi.WLInitialContextFactory,
  t3://9.20.94.16:7001,BEAServerName
  ```

- If you have defined a user-defined JMS provider configurable service, set the value for the `proprietaryAPIHandler` property manually.

Related concepts:

"Configurable services" on page 1296
Configurable services are typically runtime properties. You can use them to define properties that are related to external services on which the broker relies; for example, an SMTP server or a JMS provider.

Related reference:

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"JMSInput node" on page 4532
Use the JMSInput node to receive messages from JMS destinations. JMS destinations are accessed through a connection to a JMS provider.

"JMSOutput node" on page 4549
Use the JMSOutput node to send messages to JMS destinations.

"SOAPInput node" on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

"SOAPRequest node" on page 4828
Use the SOAPRequest node to send a SOAP request to the remote Web service.

"SOAPAsyncRequest node" on page 4750
Use the SOAPAsyncRequest node with the SOAPAsyncResponse node to construct a pair of message flows that call a Web service asynchronously.

Enabling batch acknowledgment for JMS messages:

Configure JMS message flows to send a batch acknowledgment for receipt of non-transactional JMS messages.

About this task

You can configure message flows that use the JMS transport to send an acknowledgment to the JMS server of all non-transactional JMS messages that have not previously been acknowledged. You might want to enable batch acknowledgment if there is high network latency and want to send an acknowledgment of received messages only after a threshold has been reached.

JMS flows send acknowledgment responses in accordance with the acknowledge setting on the JMS Session created by the broker. The default setting is `AUTO_ACKNOWLEDGE`, which causes JMS flows to send an acknowledgment response after receiving each non-transactional message. You can change this setting to use `CLIENT_ACKNOWLEDGE` instead by setting the properties `clientAckBatchSize` and `clientAckBatchTime` on the JMSProviders configurable service. If `CLIENT_ACKNOWLEDGE` is used, the broker calls the acknowledge() method only after a set time interval has passed, or a set number of messages are received.

Related tasks:

"Configuring the JMSInput node for batch message processing" on page 751
Configure JMS message flows to send a batch acknowledgment for receipt of non-transactional JMS messages.

Related reference:

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

"JMSInput node" on page 4532
Use the JMSInput node to receive messages from JMS destinations. JMS destinations are accessed through a connection to a JMS provider.

Processing bytes messages with JMS nodes:

The default behavior of WebSphere Message Broker when processing bytes messages can affect clients that are designed to use the readUTF() and writeUTF() methods. Construct an equivalent UTF bit stream by using a Compute node.

About this task

By default, WebSphere Message Broker processes bytes messages by using the readBytes() and writeBytes() JMS methods. By using these methods, the payload is written or read as a raw byte array. For the input message, the behavior is based on the serialization of the message tree; for the output message, the resulting bit stream is passed to the user-specified parser to construct a logical tree.

This behavior can affect clients that are designed to use the readUTF() and writeUTF() methods. A UTF string contains encoded length information as well as the raw bit stream. To construct an equivalent UTF bit stream that can be read by the readUTF() method, complete the following steps.

Procedure

1. Add a Compute node immediately before a JMSOutput node.
2. Double-click the Compute node to open the corresponding ESQL file.
3. Use the ESQL shown in the following example to construct an equivalent UTF bit stream from an XMLNSC input message. This bit stream can be understood by a client that uses the readUTF() message.

```
CALL CopyMessageHeaders();

DECLARE byteData BLOB ASBITSTREAM (InputRoot.XMLNSC ccsid
InputProperties.CodedCharSetId);
DECLARE stringData CHARACTER CAST(byteData AS CHARACTER ccsid
InputProperties.CodedCharSetId );
DECLARE dataLen INTEGER LENGTH (byteData);

DECLARE blobLen BLOB CAST(dataLen AS BLOB ENCODING
InputProperties.Encoding);
DECLARE str2byteBlobLen CHARACTER SUBSTRING (CAST(blobLen AS
CHARACTER) FROM 15 FOR 4);

SET OutputRoot.BLOB.BLOB = CAST(str2byteBlobLen as BLOB) ||
byteData ;
```

Related tasks:

"Configuring resources for processing JMS messages" on page 1714
A number of nodes are provided in WebSphere Message Broker for processing and routing JMS messages. Follow the links in this topic to find out how to configure the JMS nodes and broker resources for processing JMS messages.

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"JMSOutput node" on page 4549
Use the JMSOutput node to send messages to JMS destinations.

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.
"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

Troubleshooting JMS nodes:

Review possible problems with nodes using JMS transport.

The following errors might occur:

- "Managing badly formed messages"
- "Diagnosing problems when using globally coordinated transactions"
- "Problems with JNDI Administered Objects" on page 1731

In all cases of error, if the underlying cause is a JMS exception that has been thrown by the JMS provider, the broker BIP event message includes the text message from the JMS exception to help diagnosis.

Managing badly formed messages:
If a message cannot be processed by the JMS input node, or has been rolled back as part of a global transaction, the message is backed out to the source destination. The message is then delivered again to the input node.

To prevent badly formed messages from interrupting the processing of valid messages, the node properties can be configured as described in the following table.

Property	Description
`Backout destination`	This property specifies a JMS destination to which backed out messages are routed if the JMS message property *JMSX_DeliveryCount*, which is set by the JMS provider, exceeds the backout threshold. The JMS destination must be applicable to the message model being used by the node; for example, if a subscription topic has been configured on the node, the JMS destination must also be a topic.
`Backout threshold`	This property specifies the integer value that controls a message that is sent to the backout destination. A threshold value of 3 indicates that if the input node receives a message where the value of the *JMSX_DeliveryCount* property exceeds 3, the message is sent to the backout destination and is removed from the source destination. See "Configuring the backout threshold property" on page 4544.

Diagnosing problems when using globally coordinated transactions:
This problem is not applicable to z/OS.

In addition to the broker service trace, another trace log is provided to diagnose problems that could occur when a node using JMS transport participates in a global message flow transaction. That is, at least one JMS node in the message flow has the `Transaction Mode` property set to `Global`.

To capture the trace log, complete the following steps:

1. Define an environment variable called *XAJMS_TRACEFILE* that is available to the broker queue manager.
2. Set the value of the environment variable. This value must be a character string that represents the location and file name of the trace log. For example, on Windows, the variable can be configured as shown in the following example:

   ```
   XAJMS_TRACEFILE = c:\JMSSwitchLog
   ```
3. When the broker queue manager starts, it performs a recovery step to resolve any previous broker transactions that the JMS provider considers to be in doubt. This queue manager process writes to two trace logs during this stage. The two trace logs are:
   - *XAJMS_TRACEFILE valuePID*.txt, where *PID* is the process ID of the queue manager start process. This file is produced from the broker JMSSwitch library; for more information, see "JMS Transactionality" on page 1705.

     The previous example produces a file called JMSSwitchLog2596.txt, where the queue manager start up process ID is 2596.
   - *XAJMS_TRACEFILEXARecoveryTrace*.txt, which is produced by the recovery component of the broker that connects to the JMS provider.
4. After the broker queue manager has completed recovery, the broker starts and creates a file called *XAJMS_TRACEFILE valuePID*.txt, where *PID* is the process ID of the queue manager start process. This file is produced from the broker JMSSwitch library; for more information, see "JMS Transactionality" on page 1705.

Neither of these trace files require extra formatting.

Problems with JNDI Administered Objects:
Description of problem: The JMS node is unable to obtain the Initial Context Factory or a JNDI administered object, such as the Connection Factory or JMS destination, and message BIP4640 is issued.

Corrective action
1. Verify that the JNDI bindings have been built correctly, and can be reached at the location specified in the node.
2. Check that the values specified in the node for the Connection Factory Name and Source Queue or Destination Queue properties exist in the JNDI bindings.
3. Ensure that the correct keyword is used to match the location of the bindings:
   - file:// when the administered objects have been created in a .bindings file
   - ldap:// when the administered objects exist in an LDAP directory
   - iiop:// when CORBA is used to access the administered objects
4. When the bindings are file based, do not specify the .bindings file name in the node property.
5. Ensure that the Initial Context Factory Class name is set correctly, as specified in the documentation for the JMS provider.
6. Ensure that the Initial Context Factory Class name does not include a file path.
7. Ensure that a JMS destination (Topic or Source Queue or Destination Queue) specified in the node property exists in the JNDI administered objects.
8. Ensure that the JMS Provider jarsURL property has been set correctly by using the **mqsichangeproperties** command. To verify the value, use the following command:

   ```
   mqsireportproperties MB7BROKER -c JMSProviders -o JMSProvider –r
   ```

The JMS nodes continue to attempt to obtain the JNDI administered objects. Correct any problems and rebuild the bindings. The JMS node should automatically detect the changes and attempt to start.

- If the problem is resolved by rebuilding the bindings, the JMS node detects the changes automatically and attempts to start.
- If the problem is resolved by updating the node properties, you must redeploy the flow before it can connect successfully.
- If the problem is resolved by updating the properties of the JMSProviders configurable service, you must restart the execution group for the changes to take effect.

Description of problem: A JMS node is unable to connect for a JMS provider and issues message BIP4648.

Corrective action:

1. Verify that the JMS provider server is running. If it is offline, start the server.
2. Verify that the JMS provider server is available from the broker environment.
3. Ensure that the JMS provider Java `.jar` files have been placed into the broker shared-classes directory on distributed systems, or that on z/OS, that these `.jar` files have been defined to the broker CLASSPATH and any native libraries defined in the broker LIBPATH.

The JMS nodes continue to attempt to connect to the JMS provider. Correct any problems and the JMS node should automatically detect the changes and attempt to connect to the provider.

Description of problem: A JMS node is unable to obtain a JMS destination and issues message BIP4642.

Corrective action

1. Investigate the cause of the problem described by the JMS exception message that might be included in the BIP event message.
2. Check that the name of the JMS destination that is defined in the relevant node property (Topic, Source Queue or Destination Queue) has been defined correctly in the JNDI administered objects.
3. Verify that the underlying system resource that is used by the JMS provider for the JMS destination has been configured correctly

Description of problem: A JMS input node does not attempt to reconnect to a JMS provider following a connection failure, or a restart of the JMS provider.

Corrective action: If the JMS provider is implemented by using a model that pushes on the JMS client, rather than a traditional polling model, the JMS provider might not throw an exception when calling receive() on a broker connection. To resolve this problem, set the jmsAsyncExceptionHandling property of the JMSProviders configurable service to true for this JMS provider.

Related concepts:

"JMS Transactionality" on page 1705
JMS destinations that supply messages to an input node, or receive messages from an output node, can be sync-point coordinated as part of a message flow global transaction.

Related tasks:

"Using logs" on page 3526
There are a variety of logs that you can use to help with problem determination and troubleshooting.

Related reference:

"JMS properties for application communication models" on page 1708
JMS clients can operate with both publish/subscribe and point-to-point messages. The publish/subscribe and point-to-point application communication models use virtual channels called *destinations*. In the publish/subscribe model, the destinations are *topics*. For the point-to-point model, the destinations are known as *queues*.

"JMS message domain properties" on page 1707
The JMSInput node can receive message payloads that correspond to all of the JMS message types that are specified in the JMS Specification, version 1.1.

"JMS message selector" on page 1703
A message selector allows a JMS consumer to be more selective about the messages that it receives from a particular topic or queue.

"SOAPInput node" on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

"SOAPRequest node" on page 4828
Use the SOAPRequest node to send a SOAP request to the remote Web service.

"SOAPAsyncRequest node" on page 4750
Use the SOAPAsyncRequest node with the SOAPAsyncResponse node to construct a pair of message flows that call a Web service asynchronously.

"Service trace" on page 6875
Service trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Event Log or to user trace. Service trace is inactive by default; you must activate it explicitly by using a command.

Processing TCP/IP messages

You can use WebSphere Message Broker to connect to applications that use raw TCP/IP sockets for transferring data.

About this task

If you have existing applications that use raw TCP/IP sockets for transferring data, you can use the WebSphere Message Broker TCP/IP nodes to connect to the applications, without needing to enable them for WebSphere MQ.

WebSphere Message Broker implements access to the TCP/IP input and output streams through the following nodes:

- "TCPIPClientInput node" on page 4854
- "TCPIPClientOutput node" on page 4867
- "TCPIPClientReceive node" on page 4877
- "TCPIPServerInput node" on page 4890
- "TCPIPServerOutput node" on page 4903
- "TCPIPServerReceive node" on page 4913

For information about how to use the TCP/IP support, see "Working with TCP/IP" on page 1750.

The following topics contain information that you need to understand before you can use TCP/IP in a WebSphere Message Broker application:

- "WebSphere Broker TCP/IP Transport" on page 1735
- "TCP/IP nodes" on page 1738
- "Connection management" on page 1742
- "Configuring TCP/IP client nodes to use SSL" on page 551
- "Scenarios for WebSphere Message Broker and TCP/IP" on page 1746

Related concepts:

"WebSphere Broker TCP/IP Transport" on page 1735
The WebSphere Broker TCP/IP Transport is a service that connects applications that use raw TCP/IP sockets for transferring data.

"TCP/IP nodes" on page 1738
WebSphere Message Broker implements access to the TCP/IP input and output streams through a series of nodes.

"Connection management" on page 1742
TCP/IP connections are requested by the client connection manager and accepted by the server connection manager.

"Scenarios for WebSphere Message Broker and TCP/IP" on page 1746
Two example scenarios show how you might use TCP/IP and WebSphere Message Broker as part of a business solution.

"Scenarios: TCP/IP" on page 1746
Two scenarios illustrate how TCP/IP might be used as part of a business solution.

"Scenarios: Message Broker using TCP/IP" on page 1749
WebSphere Message Broker can be added to systems that use TCP/IP for transport, to generate a more flexible architecture for communication between components.

Related tasks:

"Working with TCP/IP" on page 1750
You can use WebSphere Message Broker TCPIP nodes and TCP/IP configurable services to perform various tasks.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

Related reference:

"TCPIPClientInput node" on page 4854
Use the TCPIPClientInput node to create a client connection to a raw TCP/IP socket, and to receive data over that connection.

"TCPIPClientOutput node" on page 4867
Use the TCPIPClientOutput node to create a client connection to a raw TCP/IP socket, and to send data over that connection to an external application.

"TCPIPClientReceive node" on page 4877
Use the TCPIPClientReceive node to receive data over a client TCP/IP connection.

"TCPIPServerInput node" on page 4890
Use the TCPIPServerInput node to create a server connection to a raw TCPIP socket, and to receive data over that connection.

"TCPIPServerOutput node" on page 4903
Use the TCPIPServerOutput node to create a server connection to a raw TCP/IP socket, and to send data over the connection to an external application.

"TCPIPServerReceive node" on page 4913
Use the TCPIPServerReceive node to receive data over a server TCP/IP connection.

“**mqsicreateconfigurableservice** command” on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

“**mqsireportproperties** command” on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

“**mqsichangeproperties** command” on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

WebSphere Broker TCP/IP Transport

The WebSphere Broker TCP/IP Transport is a service that connects applications that use raw TCP/IP sockets for transferring data.

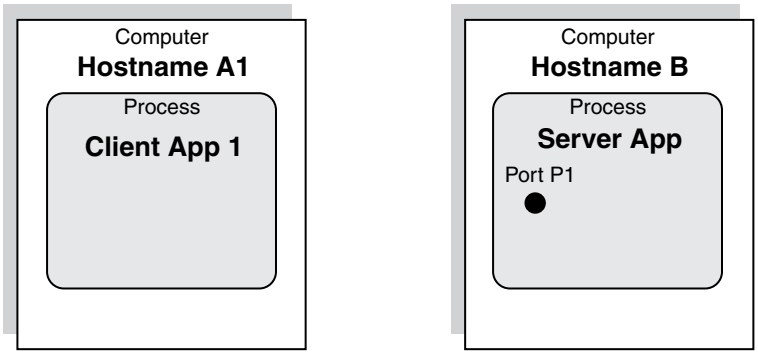
TCP/IP sockets provide a simple way of connecting computer programs together, and this type of interface is commonly added to existing stand-alone applications. TCP/IP provides a mechanism for transferring data between two applications, which can be running on different computers. The transfer of data is bidirectional; provided that the TCP/IP connection is maintained and no data is lost, the sequence of the data is kept. A significant advantage of using TCP/IP directly is that it is quick and simple to configure, which makes it a useful mechanism for processes that do not require message persistence (for example, monitoring).

However, the use of TCP/IP sockets for transferring information between programs does have some limitations:

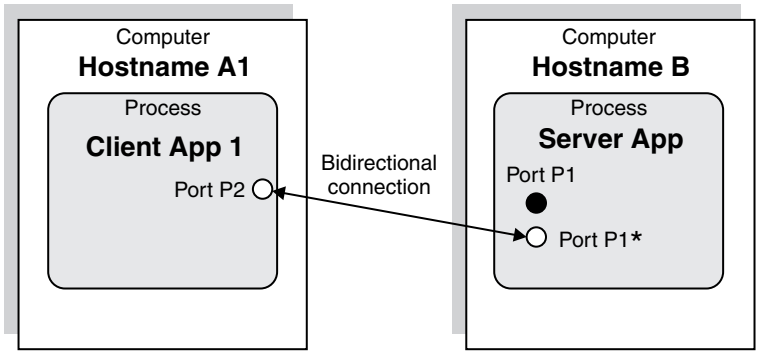
- It is non-transactional
- It is not persistent (the data is written to an in-memory buffer between the sender and receiver)
- It has no built-in security
- It provides no standard way of signaling the start and end of a message

For these reasons, it can be preferable to use a transport mechanism like WebSphere MQ, which has none of these limitations. However, if you have existing applications that use raw TCP/IP sockets for transferring data, you can use the WebSphere Message Broker TCPIP nodes to connect to the applications without needing to enable them for WebSphere MQ, so that you can develop a WebSphere Message Broker solution quickly.

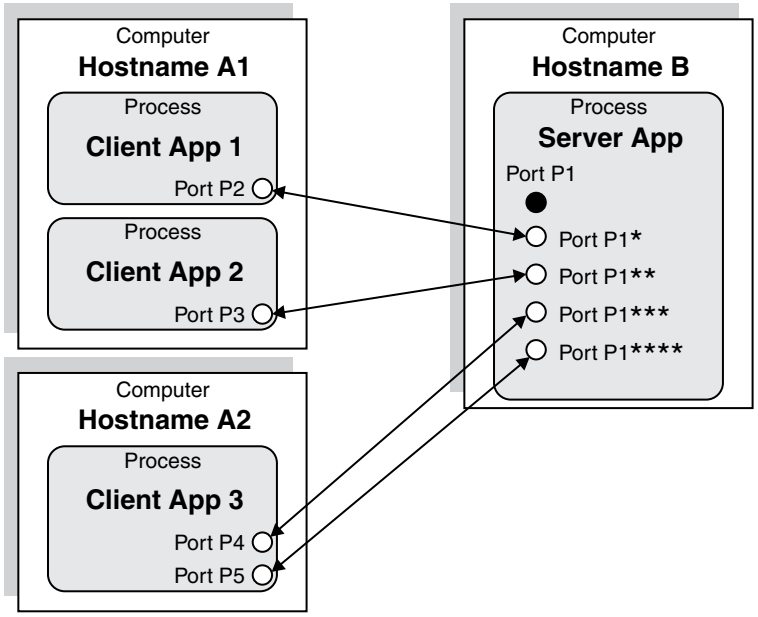
A TCP/IP connection between two applications has a client end and a server end, which means that one application acts as a server and the other as a client. The terms client and server refer only to the mechanism used to establish a connection; they do not refer to the pattern of data exchange. When the connection has been established, both client and server can perform the same operations and can both send and receive data. The following diagram illustrates the locations of client and server applications:



1. The server application listens on a local port (on the computer that is running the application) for requests for connections to be made by a client application.
2. The client application requests a connection from the server port, which the server then accepts.
3. When the server accepts the request, a port is created on the client computer and is connected to the server port.
4. A socket is created on both ends of the connection, and the details of the connection are encapsulated by the socket.
5. The server port remains available to listen for further connection requests:

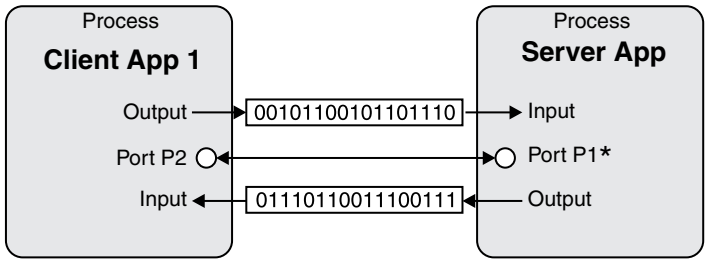


The server can accept more connections from other client applications. These connections can be in the same process, in a different process on the same computer, or on a different computer:



Only one server application can exist, but any number of different client processes can connect to the server application. Any of these applications (client or server) can be multithreaded, which enables them to use multiple connections.

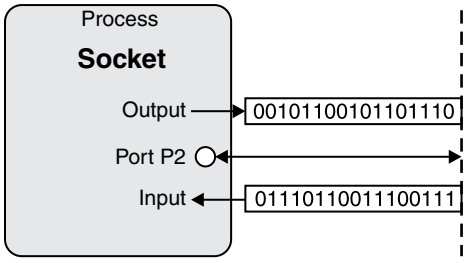
When the connection has been established, two data streams exist: one for inbound data and another for outbound data:



The client and server ends of the connection are identical and both can perform the same operations. The only difference between them is that the output stream of the client is the input stream of the server, and the input stream of the client is the output stream of the server.

The two streams of data are independent and can be accessed simultaneously from both ends. The client does not need to send data before the server.

The example illustrated in the previous diagram can be simplified in the following way, showing that the client and server have access to a socket that has an input stream and an output stream:



Related concepts:

"TCP/IP nodes"
WebSphere Message Broker implements access to the TCP/IP input and output streams through a series of nodes.

"Connection management" on page 1742
TCP/IP connections are requested by the client connection manager and accepted by the server connection manager.

"Scenarios for WebSphere Message Broker and TCP/IP" on page 1746
Two example scenarios show how you might use TCP/IP and WebSphere Message Broker as part of a business solution.

Related tasks:

"Processing TCP/IP messages" on page 1733
You can use WebSphere Message Broker to connect to applications that use raw TCP/IP sockets for transferring data.

"Working with TCP/IP" on page 1750
You can use WebSphere Message Broker TCPIP nodes and TCP/IP configurable services to perform various tasks.

Related reference:

"TCPIPClientInput node" on page 4854
Use the TCPIPClientInput node to create a client connection to a raw TCP/IP socket, and to receive data over that connection.

"TCPIPClientOutput node" on page 4867
Use the TCPIPClientOutput node to create a client connection to a raw TCP/IP socket, and to send data over that connection to an external application.

"TCPIPClientReceive node" on page 4877
Use the TCPIPClientReceive node to receive data over a client TCP/IP connection.

"TCPIPServerInput node" on page 4890
Use the TCPIPServerInput node to create a server connection to a raw TCPIP socket, and to receive data over that connection.

"TCPIPServerOutput node" on page 4903
Use the TCPIPServerOutput node to create a server connection to a raw TCP/IP socket, and to send data over the connection to an external application.

"TCPIPServerReceive node" on page 4913
Use the TCPIPServerReceive node to receive data over a server TCP/IP connection.

TCP/IP nodes

WebSphere Message Broker implements access to the TCP/IP input and output streams through a series of nodes.

Two sets of TCP/IP nodes exist: TCPIPServer nodes and TCPIPClient nodes. Both sets have identical function in terms of accessing the data streams; however, one set uses client connections and the other set uses server connections. As a result,

the nodes establish the connections in different ways but they use the streams in the same way when the connections have been established.

The main difference between the properties of the nodes is that the TCPIPServer nodes do not allow the host name to be changed (because it must always be `localhost`). All TCPIPServer nodes that use the same port must be in the same execution group because the port is tied to the running process. TCPIPClient nodes on the same port can be used in different execution groups, but client connections cannot be shared because the client connections are tied to a particular execution group, which maps to a process. Within the two sets of nodes (TCPIPClient and TCPIPServer), are three types of node:

- TCPIPServerInput and TCPIPClientInput
- TCPIPServerReceive and TCPIPClientReceive
- TCPIPServerOutput and TCPIPClientOutput

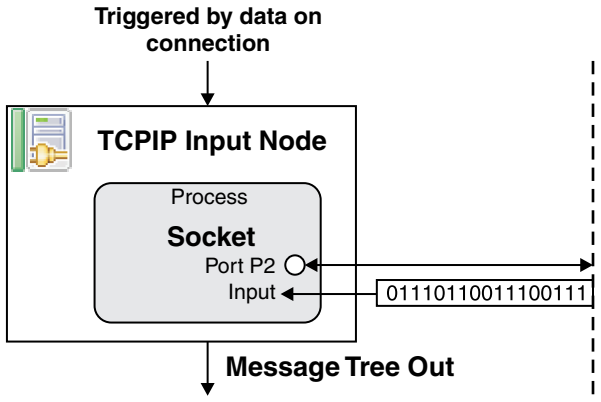
The input and receive nodes access the input stream to retrieve data, and the output nodes access the output stream to send data. No single node can access both streams at the same time. To access both streams simultaneously, you must connect multiple nodes in a message flow.

Input nodes

The input node allows access to a connection's input stream. The node is triggered by the arrival of data in the stream and starts processing the message flow. The input node controls thread and transaction management. The TCP/IP nodes are not transactional in the way that they interact with TCP/IP, but other nodes in the same flow can be transactional (for example, WebSphere MQ nodes). The input node does not create a thread for every connection being used, but waits for two requirements to be met:

- A connection is available that still has an open input stream
- Data is available on the input stream (at least 1 byte)

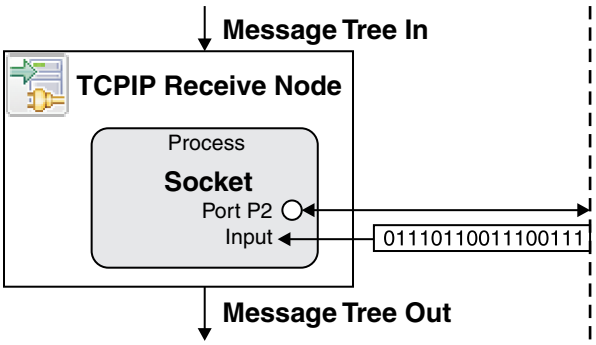
For example, 1,000 TCP/IP connections can be handled by one input node that has only one additional instance. This situation is possible because the node does not poll the connections, but is triggered when the specified conditions are met.



Receive nodes

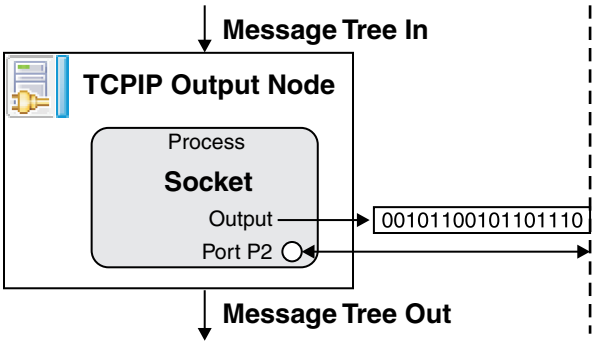
The receive node is triggered to read data from a connection when a message arrives on its In terminal. It waits for data to arrive, then sends it to the Out

terminal. You can configure the receive node to use a particular connection (by specifying a connection's ID) or to use any available connection. If the node is configured to use any available connection, it receives data from the first connection that has data available.



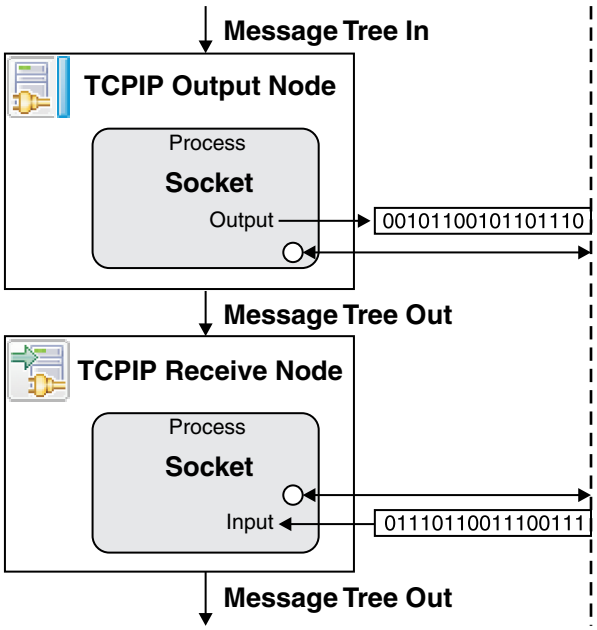
Output nodes

The output node sends data to a connection. It is triggered by a message arriving on its In terminal, then it sends the data contained in the message to the stream. The same message that is received in the node is sent to the Out terminal.



Combining nodes

The six client and server nodes can be combined to provide more complex operations. For example, an output node followed by a receive node enables a synchronous request of data:



If the message flows used are single threaded and only one connection ever exists, the sequence of nodes requires no further configuration. Two additional mechanisms are included to enable multithreading and multiple connections:

- A connection ID to ensure that the same connection is used by multiple nodes
- The ability to reserve connections so that they can be accessed only when the ID is specified

One connection in multiple nodes

Every connection has a unique identifier assigned to it when it is created. Whenever a node uses a connection, the ID that is used is written to the local environment. Any nodes that use it later in the flow can access the same connection by specifying the ID; the receive and output nodes find the ID by searching in a specified location in the local environment. By default, the location in which a node writes its connection details is different from the location in which the next node looks to see if there is an ID to use. The nodes can be configured to use the ID that was sent by a previous node. For example, the combination of the output and receive nodes shown in “Combining nodes” on page 1740 can be configured so that the receive node uses the WrittenDestination data from the preceding output node.

The use of the ID enables a series of nodes to access the same connection, but does not prevent two message flow threads accessing the same connection. When a connection is used for the first time, it can be reserved so that no other nodes can access it unless they know the ID. For example, the combination of the output and receive nodes shown in “Combining nodes” on page 1740 reserves the connection so that no other threads can access it before the receive node uses it. By default, the receive node then releases the connection when it has finished.

The ability to reserve connections (and access them by specifying the correct ID) enables you to build up complex interactions with TCP/IP connections that span whole flows and even multiple message flows. As a result, the TCP/IP interactions can be used with other asynchronous transport mechanisms like WebSphere MQ.

Reserved connections must be released at some time, otherwise they remain unavailable indefinitely. For more information about reserved and available connections, see "Connection management."

Correlating replies across flows

You can use the TCP/IP nodes in asynchronous patterns, in which data is sent out through a TCP/IP output node and received back through a TCP/IP input node. The spanning of two message flows causes any state in the first flow to be lost and therefore inaccessible from the second flow. The TCP/IP nodes enable you to store some reply details on a connection, and these details are then available for the input node to use when a new event arrives on the same connection. By default, this data is taken from the local environment, but you can configure the nodes to take the data from any location, including the Correlid field and message IDs in WebSphere MQ headers.

Related concepts:

"WebSphere Broker TCP/IP Transport" on page 1735
The WebSphere Broker TCP/IP Transport is a service that connects applications that use raw TCP/IP sockets for transferring data.

"Connection management"
TCP/IP connections are requested by the client connection manager and accepted by the server connection manager.

"Scenarios for WebSphere Message Broker and TCP/IP" on page 1746
Two example scenarios show how you might use TCP/IP and WebSphere Message Broker as part of a business solution.

Related tasks:

"Working with TCP/IP" on page 1750
You can use WebSphere Message Broker TCPIP nodes and TCP/IP configurable services to perform various tasks.

Related reference:

"TCPIPClientInput node" on page 4854
Use the TCPIPClientInput node to create a client connection to a raw TCP/IP socket, and to receive data over that connection.

"TCPIPClientOutput node" on page 4867
Use the TCPIPClientOutput node to create a client connection to a raw TCP/IP socket, and to send data over that connection to an external application.

"TCPIPClientReceive node" on page 4877
Use the TCPIPClientReceive node to receive data over a client TCP/IP connection.

"TCPIPServerInput node" on page 4890
Use the TCPIPServerInput node to create a server connection to a raw TCPIP socket, and to receive data over that connection.

"TCPIPServerOutput node" on page 4903
Use the TCPIPServerOutput node to create a server connection to a raw TCP/IP socket, and to send data over the connection to an external application.

"TCPIPServerReceive node" on page 4913
Use the TCPIPServerReceive node to receive data over a server TCP/IP connection.

Connection management

TCP/IP connections are requested by the client connection manager and accepted by the server connection manager.

The execution group process contains the connection manager, which makes the connections. Only one execution group can have server nodes using a specific port at any one time; deployment to a second execution group causes a deployment error. Client nodes can be deployed to different execution groups, but each execution group has its own pool of connections, and therefore its own minimum and maximum number of connections.

TCPIP nodes do not directly create or manage any TCP/IP connections, but acquire them from the connection manager's internal pool. For example, two output nodes using the same connection details share the same connection manager. The TCPIP nodes can define the connection details to be used by specifying one of the following values:

- Host name and port
- Name of a configurable service.

If a host name and port are specified, the node uses these values when requesting connections. If a configurable service is specified, the node obtains the values for the port and host name from the values defined in the configurable service. The connection manager supports other configurable parameters in addition to the host name and port, and you can define all of these values when you are using a configurable service. When the host name and port are specified on the node, the connection manager obtains the rest of the required values from the default configurable service. However, if a configurable service is defined that is using the host name and port number, the values from that configurable service are used.

The connection manager is created when the first node that requires connections from it is deployed. The connection manager is deleted when the last remaining node using it has been removed from the execution group (which means that the connection manager is no longer being used by any deployed nodes). For example, this process can occur when existing flows are redeployed, because redeployment involves deleting all existing nodes before creating them again.

Server connections

The server connection manager starts listening for server connections when it starts, and it keeps accepting connections until the maximum number of connections (as specified in the configurable service) is reached. Any attempts to make connections after this point are refused. TCP/IP servers do not create connections; they accept connection requests only from other applications. As a result, you cannot force the creation of connections within a message flow.

Client connections

The client connection manager starts and keeps making client connections until the minimum number of connections (as defined in the configurable service) is reached. By default, the minimum number of connections is zero, which means that no connections are made. Whenever the number of connections drops below the minimum value, the connection manager starts creating more client connections. The client output and receive nodes initiate the creation of new client connections whenever none are available for them to use, unless the maximum number (as defined in the configurable service) has been reached.

Reserving and releasing connections

Each connection has an input stream and an output stream, both of which have two main states within the connection manager: available and reserved.

When a node requests a connection for input or output, without specifying the ID of a particular connection, it is given any available connection on the required stream. If no connections are available, and if the node is a client node, a new connection is made, but only if the maximum number of connections has not yet been reached. Any connection in the available state can be used by only one node at a time, but when a node has finished using it, any other node (from any flow or thread) can access it.

You can restrict access to a stream on a connection by reserving the connection. When a connection is in the reserved state, no other node can access the stream without specifying the ID of the connection. For example, an input node can request an available connection, and, when it has finished reading the data, put the stream into the reserved state. While the stream is in the reserved state, no input node (including the node that put the stream into the reserved state) can access it because input nodes can access only available streams. The only nodes that can access the stream must have the connection ID, which is written to the outgoing local environment when the data is passed down the message flow. As a result, receive nodes can read more data on the same connection, but only if the receive node is configured to use the ID from the local environment of the input node.

When a connection is reserved, ownership of the connection is given to a current thread of processing. This processing can span separate message flows, if required.

The reserve mechanism provides the following options:

- Leave unchanged
- Reserve
- Release
- Reserve and release at the end of the flow

For all nodes, the stream is left available (not reserved) by default. For many types of processing you can leave this default unchanged; for example, when you are moving data from an input stream to a file. The main purpose of reserving a stream is to connect a series of nodes to give complex processing on a stream in an ordered, controlled, synchronous sequence.

You can use the `Reserve and release at end of flow` option to reserve a connection and to ensure that the connection's stream is released when one iteration of the message flow has finished processing (including any error conditions that might occur).

If you require the processing to span multiple message flows (for example, for asynchronous request and reply), you must reserve a stream without releasing it at the end of the message flow. See the following sample for an example:

- TCPIP Client Nodes

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

A disadvantage of reserving a stream between message flows is the potential for a stream never to be released. To avoid this problem, set an expiry time on the connection so that it is closed after a specified period of inactivity.

Another benefit of reserving an input stream is that the connection cannot be closed until it is either released or expired (even if an end application closes its end of the connection), which is useful when the end of the stream is being used to delimit messages in the stream.

File descriptors

If your WebSphere Message Broker application is running on Sun Solaris 10 on SPARC, you might need to increase the number of file descriptors. The following error in the `syslog` indicates that additional file descriptors are required:

```
Failed to create a client connection using hostname: '', port: ''. Reason: 'Invalid argument'
```

You can also try the following two methods to resolve the error:

- Change the *MQSIJVERBOSE* environment variable, for example:

  ```
  export MQSIJVERBOSE=-Djava.nio.channels.spi.SelectorProvider=sun.nio.ch.PollSelectorProvider
  ```

- Change the limit of maximum file handles to *value* instead of *RLIM64_INFINITY*

Related concepts:

"WebSphere Broker TCP/IP Transport" on page 1735
The WebSphere Broker TCP/IP Transport is a service that connects applications that use raw TCP/IP sockets for transferring data.

"TCP/IP nodes" on page 1738
WebSphere Message Broker implements access to the TCP/IP input and output streams through a series of nodes.

"Scenarios for WebSphere Message Broker and TCP/IP" on page 1746
Two example scenarios show how you might use TCP/IP and WebSphere Message Broker as part of a business solution.

Related tasks:

"Working with TCP/IP" on page 1750
You can use WebSphere Message Broker TCPIP nodes and TCP/IP configurable services to perform various tasks.

Related reference:

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

"TCPIPClientInput node" on page 4854
Use the TCPIPClientInput node to create a client connection to a raw TCP/IP socket, and to receive data over that connection.

"TCPIPClientOutput node" on page 4867
Use the TCPIPClientOutput node to create a client connection to a raw TCP/IP socket, and to send data over that connection to an external application.

"TCPIPClientReceive node" on page 4877
Use the TCPIPClientReceive node to receive data over a client TCP/IP connection.

"TCPIPServerInput node" on page 4890
Use the TCPIPServerInput node to create a server connection to a raw TCPIP socket, and to receive data over that connection.

"TCPIPServerOutput node" on page 4903
Use the TCPIPServerOutput node to create a server connection to a raw TCP/IP socket, and to send data over the connection to an external application.

"TCPIPServerReceive node" on page 4913
Use the TCPIPServerReceive node to receive data over a server TCP/IP connection.

Scenarios for WebSphere Message Broker and TCP/IP

Two example scenarios show how you might use TCP/IP and WebSphere Message Broker as part of a business solution.

- "Scenarios: TCP/IP"
- "Scenarios: Message Broker using TCP/IP" on page 1749

Related concepts:

"WebSphere Broker TCP/IP Transport" on page 1735
The WebSphere Broker TCP/IP Transport is a service that connects applications that use raw TCP/IP sockets for transferring data.

"TCP/IP nodes" on page 1738
WebSphere Message Broker implements access to the TCP/IP input and output streams through a series of nodes.

"Scenarios: TCP/IP"
Two scenarios illustrate how TCP/IP might be used as part of a business solution.

"Scenarios: Message Broker using TCP/IP" on page 1749
WebSphere Message Broker can be added to systems that use TCP/IP for transport, to generate a more flexible architecture for communication between components.

Related tasks:

"Working with TCP/IP" on page 1750
You can use WebSphere Message Broker TCPIP nodes and TCP/IP configurable services to perform various tasks.

Scenarios: TCP/IP:

Two scenarios illustrate how TCP/IP might be used as part of a business solution.

- "Expense submission"
- "Price-change notification" on page 1747.

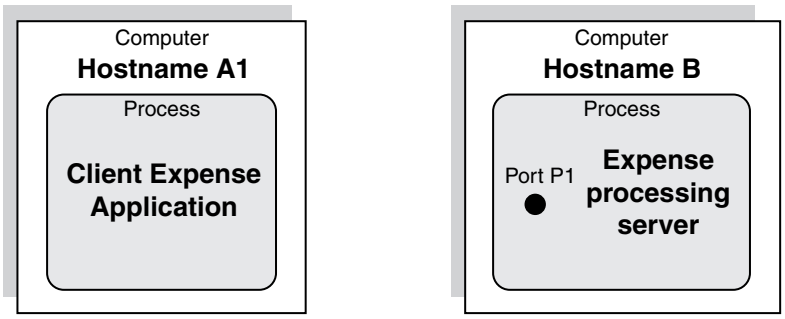
Expense submission: **Scenario:**

A company has an expense-submission system based on a central expense-processing application, which receives completed expense forms from end users. The users complete the forms by using a local application, which stores the form until it is completed. When it has been completed, the form is transferred to the central system where it is processed. Any further notifications are sent to the user by email.

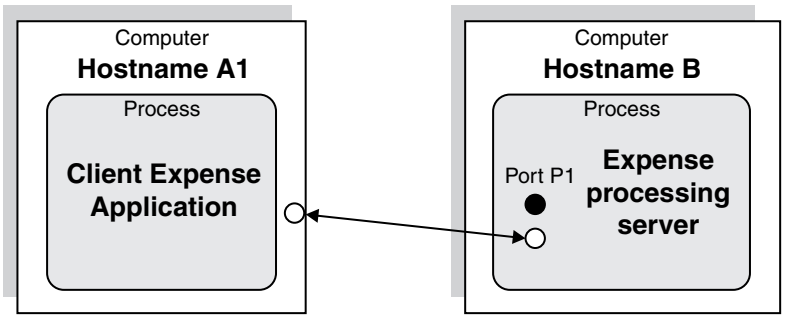
How TCP/IP is used:

One client application exists for each end user, and one central application processes all the expense forms. Each client application connects to a TCP/IP server port on the server application and sends the expense form in a fixed-field-size structure similar to a COBOL structure. The flow of processing is:

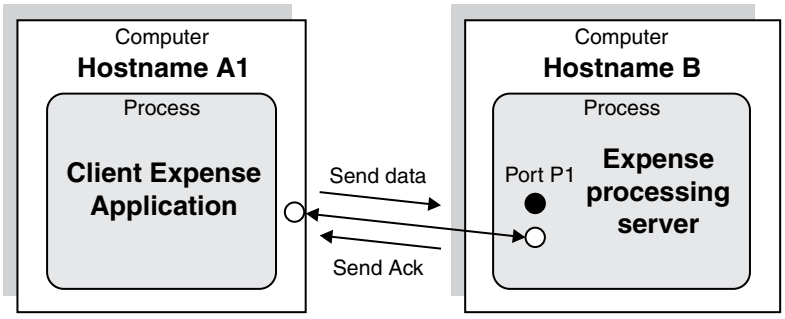
1. The user enters information into the form in the client expense application.



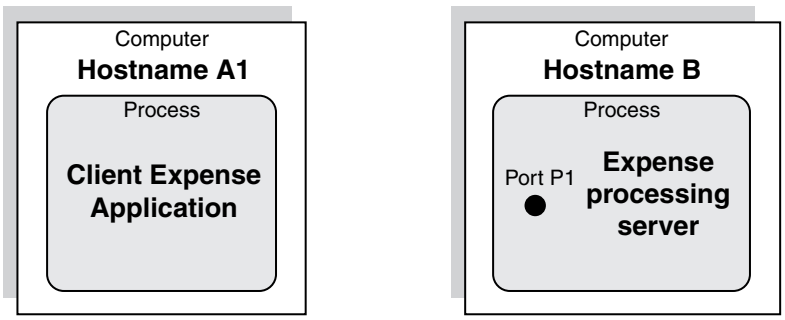
2. The user submits the completed form and the client application connects to the server.



3. The client application sends the data to the server and receives an acknowledgment.



4. The connection is closed by the client.

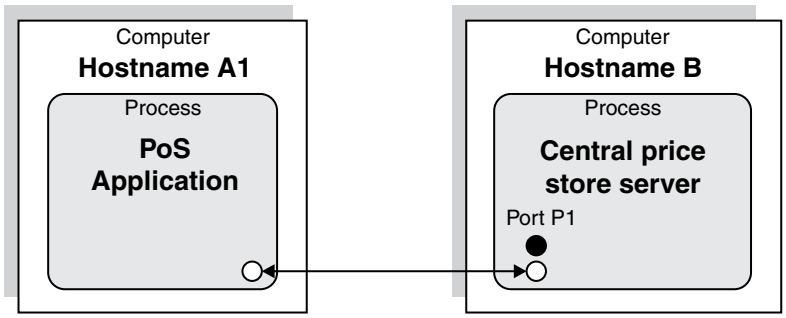


Price-change notification: **Scenario:**

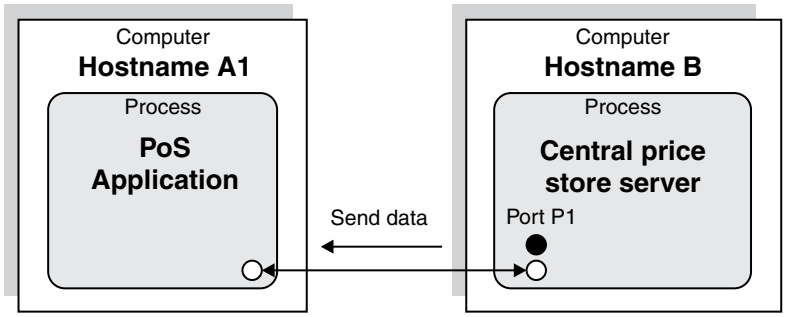
A company has a central server, which stores the catalog price of everything that the company sells. This information is required by all Point of Sale (PoS) terminals in all the stores, and each PoS terminal must be notified when any price changes. The PoS terminals connect to the central server and wait for any process changes. The server sends any process changes to all connected client applications.

How TCP/IP is used:

1. When the PoS application starts, it connects to the central server.



2. Whenever the server has a new price, it publishes it to all the connected clients.



3. The PoS stays connected until it shuts down.

See "Scenarios: Message Broker using TCP/IP" on page 1749 for an example of how these scenarios can be modified to use WebSphere Message Broker.

Related concepts:

"WebSphere Broker TCP/IP Transport" on page 1735
The WebSphere Broker TCP/IP Transport is a service that connects applications that use raw TCP/IP sockets for transferring data.

"TCP/IP nodes" on page 1738
WebSphere Message Broker implements access to the TCP/IP input and output streams through a series of nodes.

"Scenarios for WebSphere Message Broker and TCP/IP" on page 1746
Two example scenarios show how you might use TCP/IP and WebSphere Message Broker as part of a business solution.

"Scenarios: Message Broker using TCP/IP" on page 1749
WebSphere Message Broker can be added to systems that use TCP/IP for transport, to generate a more flexible architecture for communication between components.

Related tasks:

"Working with TCP/IP" on page 1750
You can use WebSphere Message Broker TCPIP nodes and TCP/IP configurable

services to perform various tasks.

Related reference:

"TCPIPClientInput node" on page 4854
Use the TCPIPClientInput node to create a client connection to a raw TCP/IP socket, and to receive data over that connection.

"TCPIPClientOutput node" on page 4867
Use the TCPIPClientOutput node to create a client connection to a raw TCP/IP socket, and to send data over that connection to an external application.

"TCPIPClientReceive node" on page 4877
Use the TCPIPClientReceive node to receive data over a client TCP/IP connection.

"TCPIPServerInput node" on page 4890
Use the TCPIPServerInput node to create a server connection to a raw TCPIP socket, and to receive data over that connection.

"TCPIPServerOutput node" on page 4903
Use the TCPIPServerOutput node to create a server connection to a raw TCP/IP socket, and to send data over the connection to an external application.

"TCPIPServerReceive node" on page 4913
Use the TCPIPServerReceive node to receive data over a server TCP/IP connection.

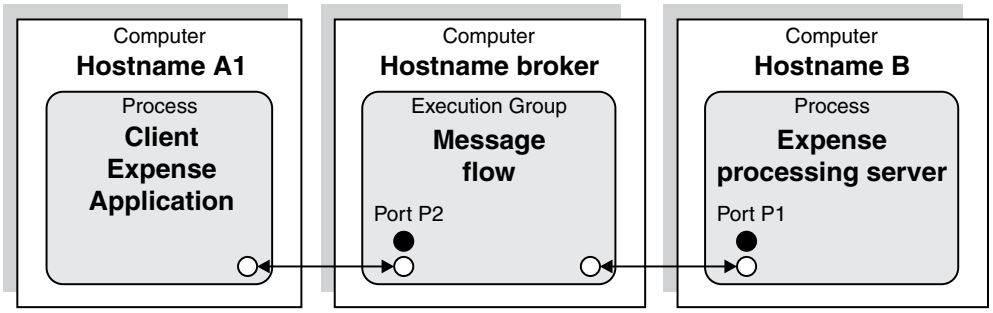
Scenarios: Message Broker using TCP/IP:

WebSphere Message Broker can be added to systems that use TCP/IP for transport, to generate a more flexible architecture for communication between components.

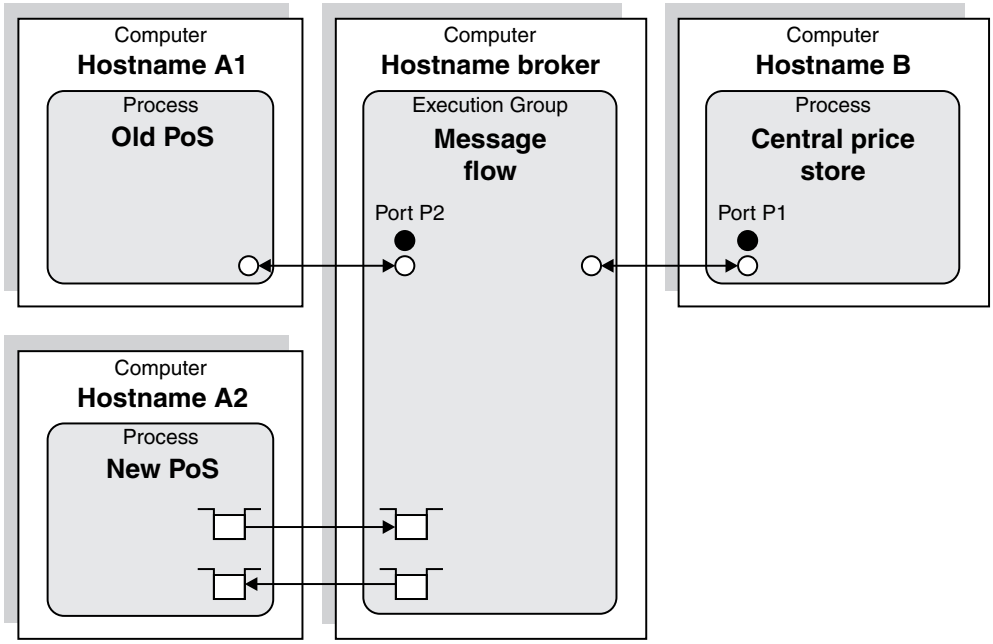
The scenarios in the "Scenarios: TCP/IP" on page 1746 topic show how systems can be created to use TCP/IP as a transport mechanism. The following sections show how WebSphere Message Broker can be added to those systems to generate a more flexible architecture for communication between components:

- "Expense submission" illustrates TCP/IP to TCP/IP routing
- "Price-change notification" on page 1750 illustrates routing and transformation to other formats

Expense submission: The expense submission scenario shown in the "Scenarios: TCP/IP" on page 1746 topic requires a direct connection from the client applications to the end server. With that model it is difficult to add new consumers of the expense submission information, and it is also difficult to change the end application that processes them. However, by adding WebSphere Message Broker as an intermediary router, the two systems can be separated without any changes to their interfaces, as shown in the following diagram:



Price-change notification: The price-change notification scenario shown in the "Scenarios: TCP/IP" on page 1746 topic can be modified to use a message broker for routing and transformation. It could also allow support for other protocols like WebSphere MQ, which would allow new applications to be written to different interfaces without the need for changing the current client or server applications:



Related concepts:

"WebSphere Broker TCP/IP Transport" on page 1735
The WebSphere Broker TCP/IP Transport is a service that connects applications that use raw TCP/IP sockets for transferring data.

"TCP/IP nodes" on page 1738
WebSphere Message Broker implements access to the TCP/IP input and output streams through a series of nodes.

"Scenarios for WebSphere Message Broker and TCP/IP" on page 1746
Two example scenarios show how you might use TCP/IP and WebSphere Message Broker as part of a business solution.

"Scenarios: TCP/IP" on page 1746
Two scenarios illustrate how TCP/IP might be used as part of a business solution.

Related tasks:

"Working with TCP/IP"
You can use WebSphere Message Broker TCPIP nodes and TCP/IP configurable services to perform various tasks.

Working with TCP/IP

You can use WebSphere Message Broker TCPIP nodes and TCP/IP configurable services to perform various tasks.

About this task

- "Transferring XML data from a TCP/IP server socket to a WebSphere MQ queue" on page 1752
- "Transferring binary (CWF) data from a TCP/IP server socket to a flat file" on page 1754

- "Receiving data on a TCP/IP server socket and sending data back to the same connection" on page 1756
- "Sending XML data from a WebSphere MQ queue to a TCP/IP client socket" on page 1757
- "Sending CWF data from a flat file to a TCP/IP client socket" on page 1759
- "Sending data to a TCP/IP client connection and receiving data back on the same connection (synchronous)" on page 1760
- "Establishing a client session over a TCP/IP connection" on page 1765
- "Broadcasting data to all currently available connections" on page 1767
- "Configuring a server socket so that connections expire after a specified time" on page 1768
- "Configuring a client socket to make 100 connections at deployment or startup time" on page 1769
- "Configuring a server socket to receive XML data ending in a null character" on page 1771
- "Configuring a server socket to receive XML data and discover the end of a record (by using the message model)" on page 1772
- "Configuring a server output node to close all connections" on page 1774
- "Configuring a client socket to store reply correlation details" on page 1775
- "Writing close connection details to a file" on page 1777
- "Configuring a client node to dynamically call a port" on page 1779
- "Configuring a server receive node to wait for data on a specified port" on page 1781
- "Sending and receiving data through a TCP/IP client connection, delimiting the record by closing the output stream (asynchronous)" on page 1782
- "Sending and receiving data on the same TCP/IP client connection, closing input and output streams (synchronous)" on page 1784.
- "Configuring TCP/IP client nodes to use SSL" on page 551

Related concepts:

"WebSphere Broker TCP/IP Transport" on page 1735
The WebSphere Broker TCP/IP Transport is a service that connects applications that use raw TCP/IP sockets for transferring data.

"TCP/IP nodes" on page 1738
WebSphere Message Broker implements access to the TCP/IP input and output streams through a series of nodes.

"Scenarios: TCP/IP" on page 1746
Two scenarios illustrate how TCP/IP might be used as part of a business solution.

"Scenarios: Message Broker using TCP/IP" on page 1749
WebSphere Message Broker can be added to systems that use TCP/IP for transport, to generate a more flexible architecture for communication between components.

Related tasks:

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

Related reference:

"TCPIPClientInput node" on page 4854
Use the TCPIPClientInput node to create a client connection to a raw TCP/IP socket, and to receive data over that connection.

"TCPIPClientOutput node" on page 4867
Use the TCPIPClientOutput node to create a client connection to a raw TCP/IP socket, and to send data over that connection to an external application.

"TCPIPClientReceive node" on page 4877
Use the TCPIPClientReceive node to receive data over a client TCP/IP connection.

"TCPIPServerInput node" on page 4890
Use the TCPIPServerInput node to create a server connection to a raw TCPIP socket, and to receive data over that connection.

"TCPIPServerOutput node" on page 4903
Use the TCPIPServerOutput node to create a server connection to a raw TCP/IP socket, and to send data over the connection to an external application.

"TCPIPServerReceive node" on page 4913
Use the TCPIPServerReceive node to receive data over a server TCP/IP connection.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

Transferring XML data from a TCP/IP server socket to a WebSphere MQ queue:

Transfer XML data from a TCP/IP server socket to a WebSphere MQ queue, by creating a message flow with TCPIPServerInput and MQOutput nodes.

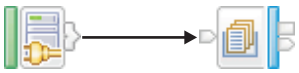
About this task

Scenario: A client application opens a TCP/IP socket and sends an XML document. The end of the document is signalled by the closure of the client connection.

Instructions: The following steps describe how to write a message flow that can receive the XML document and write it to a WebSphere MQ queue:

Procedure

1. Create a message flow called TCPIP_Task1 with a TCPIPServerInput node and an MQOutput node. For more information about how to do this, see "Creating a message flow" on page 1431.
2. Connect the Out terminal of the TCPIPServerInput node to the In terminal of the MQOutput node.



3. Set the following properties of the TCPIPServerInput node:
   a. On the **Basic** tab, set the `Connection details` property to `14141`.
   b. On the **Input Message Parsing** tab, set the `Message domain` property to `XMLNSC`.
4. On the MQOutput node, set the `Queue name` property (on the **Basic** tab) to `TCPIP.TASK1.OUT1`.
5. Save the message flow.

Related concepts:

"WebSphere Broker TCP/IP Transport" on page 1735
The WebSphere Broker TCP/IP Transport is a service that connects applications that use raw TCP/IP sockets for transferring data.

"TCP/IP nodes" on page 1738
WebSphere Message Broker implements access to the TCP/IP input and output streams through a series of nodes.

"Scenarios: TCP/IP" on page 1746
Two scenarios illustrate how TCP/IP might be used as part of a business solution.

"Scenarios: Message Broker using TCP/IP" on page 1749
WebSphere Message Broker can be added to systems that use TCP/IP for transport, to generate a more flexible architecture for communication between components.

Related tasks:

"Working with TCP/IP" on page 1750
You can use WebSphere Message Broker TCPIP nodes and TCP/IP configurable services to perform various tasks.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

Related reference:

"TCPIPClientInput node" on page 4854
Use the TCPIPClientInput node to create a client connection to a raw TCP/IP socket, and to receive data over that connection.

"TCPIPClientOutput node" on page 4867
Use the TCPIPClientOutput node to create a client connection to a raw TCP/IP socket, and to send data over that connection to an external application.

"TCPIPClientReceive node" on page 4877
Use the TCPIPClientReceive node to receive data over a client TCP/IP connection.

"TCPIPServerInput node" on page 4890
Use the TCPIPServerInput node to create a server connection to a raw TCPIP socket, and to receive data over that connection.

"TCPIPServerOutput node" on page 4903
Use the TCPIPServerOutput node to create a server connection to a raw TCP/IP socket, and to send data over the connection to an external application.

"TCPIPServerReceive node" on page 4913
Use the TCPIPServerReceive node to receive data over a server TCP/IP connection.

"MQOutput node" on page 4612
Use the MQOutput node to send messages to clients that connect to the broker using the WebSphere MQ Enterprise Transport and that use the MQI and AMI application programming interfaces.

Transferring binary (CWF) data from a TCP/IP server socket to a flat file:

Transfer binary Custom Wire Format (CWF) data from a TCP/IP server socket to a flat file, by the use of a message set and a message flow with TCPIPServerInput and FileOutput nodes.

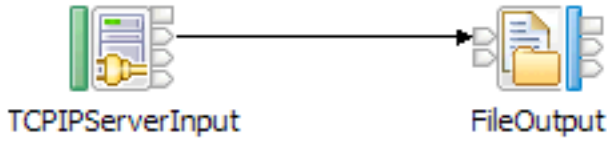
About this task

Scenario: A client application opens a TCP/IP socket and sends a binary (CWF) document. The end of the document is signaled by the closure of the client connection.

Instructions: The following steps describe how to write a message flow that can receive the binary document and write it to a flat file. Each message is written to a separate file, the name of which is based on the ID of the connection.

Procedure

1. Create a message set called Task2_MsgSet. For more information, see "Creating a message set" on page 2842.
2. Create a message flow called TCPIP_Task2 with a TCPIPServerInput node and a FileOutput node. For more information, see "Creating a message flow" on page 1431.
3. Connect the Out terminal of the TCPIPServerInput node to the In terminal of the FileOutput node.



4. Set the following properties of the TCPIPServerInput node:
   a. On the **Basic** tab, set the `Connection details` property to `14142`.
   b. On the **Input Message Parsing** tab, set the following properties:
      - Set the `Message domain` property to `MRM`.
      - Set the `Message set` property to `Task2_MsgSet`.
      - Set the `Message type` property to `Task2_MsgType`.
      - Set the `Message format` property to `Binary1`.
5. Set the following properties of the FileOutput node:
   a. On the **Basic** tab, set the following properties:
      - Set the `Directory` property to `c:\temp\task2`.
      - Set the `File name or pattern` property to `Task2.out`.
   b. On the **Request** tab, set the `Request file name property location` property to `$LocalEnvironment/TCPIP/Input/ConnectionDetails/Id`.
6. Save the message flow.

7. Create a project reference between the message flow project and the message set project:
   a. Right-click the message flow project, and click **Properties**.
   b. Click **Project References**.
   c. Select the message set project that contains your message set (Task2_MsgSet).
   d. Click **OK**.

Related concepts:

"WebSphere Broker TCP/IP Transport" on page 1735
The WebSphere Broker TCP/IP Transport is a service that connects applications that use raw TCP/IP sockets for transferring data.

"TCP/IP nodes" on page 1738
WebSphere Message Broker implements access to the TCP/IP input and output streams through a series of nodes.

"Scenarios: TCP/IP" on page 1746
Two scenarios illustrate how TCP/IP might be used as part of a business solution.

"Scenarios: Message Broker using TCP/IP" on page 1749
WebSphere Message Broker can be added to systems that use TCP/IP for transport, to generate a more flexible architecture for communication between components.

"Project references" on page 44
When a projects refers to other projects in the workspace, this is called a *project reference*. When one project references another, the files in the referenced project are available for use by the referring project. For example to configure certain nodes, to create libraries of reusable message flows, and to enable Content Assist in the ESQL editor.

Related tasks:

"Working with TCP/IP" on page 1750
You can use WebSphere Message Broker TCPIP nodes and TCP/IP configurable services to perform various tasks.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

Related reference:

"TCPIPClientInput node" on page 4854
Use the TCPIPClientInput node to create a client connection to a raw TCP/IP socket, and to receive data over that connection.

"TCPIPClientOutput node" on page 4867
Use the TCPIPClientOutput node to create a client connection to a raw TCP/IP socket, and to send data over that connection to an external application.

"TCPIPClientReceive node" on page 4877
Use the TCPIPClientReceive node to receive data over a client TCP/IP connection.

"TCPIPServerInput node" on page 4890
Use the TCPIPServerInput node to create a server connection to a raw TCPIP socket, and to receive data over that connection.

"TCPIPServerOutput node" on page 4903
Use the TCPIPServerOutput node to create a server connection to a raw TCP/IP socket, and to send data over the connection to an external application.

"TCPIPServerReceive node" on page 4913
Use the TCPIPServerReceive node to receive data over a server TCP/IP connection.

"FileOutput node" on page 4430
Use the FileOutput node to write messages to files.

Receiving data on a TCP/IP server socket and sending data back to the same connection:

Receive data on a TCP/IP server socket, then send the data to the same connection, by the use of a message flow with TCPIPServerInput and TCPIPServerOutput nodes.

About this task

Scenario: A client application opens a TCP/IP socket and sends an undefined document (of any format or size). The end of the document is signalled by the client closing the output stream (but not the connection), and waiting for the same data to be sent back.

Instructions: The following steps describe how to write a message flow that can receive the data and echo it back to the same connection:

Procedure

1. Create a message flow called TCPIP_Task3 with a TCPIPServerInput node and a TCPIPServerOutput node. For more information, see "Creating a message flow" on page 1431.
2. Connect the Out terminal of the TCPIPServerInput node to the In terminal of the TCPIPServerOutput node.

3. Set the following properties of the TCPIPServerInput node:
   a. On the **Basic** tab, set the `Connection details` property to `14143`.
   b. On the **Advanced** tab, set the `Input stream modification` property to `Reserve input stream and release at end of flow`.
4. Set the following properties of the TCPIPServerOutput node:
   a. On the **Basic** tab, set the `Connection details` property to `14143`.
   b. On the **Request** tab, set the `ID location` property to `LocalEnvironment/TCPIP/Input/ConnectionDetails/Id`.
   c. On the **Advanced** tab, set the `Close connection` property to `After data has been sent`.
5. Save the message flow.

Related concepts:

"WebSphere Broker TCP/IP Transport" on page 1735
The WebSphere Broker TCP/IP Transport is a service that connects applications that use raw TCP/IP sockets for transferring data.

"TCP/IP nodes" on page 1738
WebSphere Message Broker implements access to the TCP/IP input and output streams through a series of nodes.

"Scenarios: TCP/IP" on page 1746
Two scenarios illustrate how TCP/IP might be used as part of a business solution.

"Scenarios: Message Broker using TCP/IP" on page 1749
WebSphere Message Broker can be added to systems that use TCP/IP for transport, to generate a more flexible architecture for communication between components.

Related tasks:

"Working with TCP/IP" on page 1750
You can use WebSphere Message Broker TCPIP nodes and TCP/IP configurable services to perform various tasks.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

Related reference:

"TCPIPClientInput node" on page 4854
Use the TCPIPClientInput node to create a client connection to a raw TCP/IP socket, and to receive data over that connection.

"TCPIPClientOutput node" on page 4867
Use the TCPIPClientOutput node to create a client connection to a raw TCP/IP socket, and to send data over that connection to an external application.

"TCPIPClientReceive node" on page 4877
Use the TCPIPClientReceive node to receive data over a client TCP/IP connection.

"TCPIPServerInput node" on page 4890
Use the TCPIPServerInput node to create a server connection to a raw TCPIP socket, and to receive data over that connection.

"TCPIPServerOutput node" on page 4903
Use the TCPIPServerOutput node to create a server connection to a raw TCP/IP socket, and to send data over the connection to an external application.

"TCPIPServerReceive node" on page 4913
Use the TCPIPServerReceive node to receive data over a server TCP/IP connection.

Sending XML data from a WebSphere MQ queue to a TCP/IP client socket:

Send XML data from a WebSphere MQ queue to a TCP/IP client socket, by the use of a message flow with MQInput and TCPIPClientOutput nodes.

About this task

Scenario: A server application listens on a TCP/IP socket and waits for a TCP/IP client to connect and send data. The end of the document is signalled by the client closing the connection.

Instructions: The following steps describe how to write a message flow that can take a message from a WebSphere MQ queue, make the client connection, and send the data to the server application:

Procedure

1. Create a message flow called TCPIP_Task4 with an MQInput node and a TCPIPClientOutput node. For more information, see "Creating a message flow" on page 1431.
2. Connect the Out terminal of the MQInput node to the In terminal of the TCPIPClientOutput node.

3. Set the following properties of the MQInput node:
   a. On the **Basic** tab, set the `Queue name` property to `TCPIP.TASK4.IN1`.
   b. On the **Input message parsing** tab, set the `Message domain` property to `XMLNSC`.
4. Set the following properties of the TCPIPClientOutput node:
   a. On the **Basic** tab, set the `Connection details` property to `14144`.
   b. On the **Advanced** tab, set the `Close connection` property to `After data has been sent`.
5. Save the message flow.

Related concepts:

"WebSphere Broker TCP/IP Transport" on page 1735
The WebSphere Broker TCP/IP Transport is a service that connects applications that use raw TCP/IP sockets for transferring data.

"TCP/IP nodes" on page 1738
WebSphere Message Broker implements access to the TCP/IP input and output streams through a series of nodes.

"Scenarios: TCP/IP" on page 1746
Two scenarios illustrate how TCP/IP might be used as part of a business solution.

"Scenarios: Message Broker using TCP/IP" on page 1749
WebSphere Message Broker can be added to systems that use TCP/IP for transport, to generate a more flexible architecture for communication between components.

Related tasks:

"Working with TCP/IP" on page 1750
You can use WebSphere Message Broker TCPIP nodes and TCP/IP configurable services to perform various tasks.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

Related reference:

"TCPIPClientInput node" on page 4854
Use the TCPIPClientInput node to create a client connection to a raw TCP/IP socket, and to receive data over that connection.

"TCPIPClientOutput node" on page 4867
Use the TCPIPClientOutput node to create a client connection to a raw TCP/IP socket, and to send data over that connection to an external application.

"TCPIPClientReceive node" on page 4877
Use the TCPIPClientReceive node to receive data over a client TCP/IP connection.

"TCPIPServerInput node" on page 4890
Use the TCPIPServerInput node to create a server connection to a raw TCPIP socket, and to receive data over that connection.

"TCPIPServerOutput node" on page 4903
Use the TCPIPServerOutput node to create a server connection to a raw TCP/IP socket, and to send data over the connection to an external application.

"TCPIPServerReceive node" on page 4913
Use the TCPIPServerReceive node to receive data over a server TCP/IP connection.

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

Sending CWF data from a flat file to a TCP/IP client socket:

Send Custom Wire Format (CWF) data from a flat file to a TCP/IP client socket, by the use of a message flow with FileInput and TCPIPClientOutput nodes.

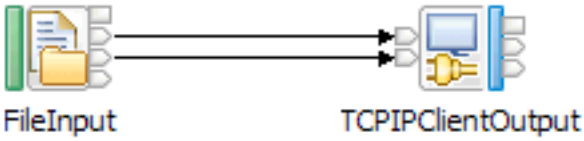
About this task

Scenario: An application writes 100–byte binary records into a flat file.

Instructions: The following steps describe how to open a new client TCP/IP connection and send the binary data with a binary termination character x'00FF'. When the whole file is finished, the client connection is closed:

Procedure

1. Create a message flow called TCPIP_Task5 with a FileInput node and a TCPIPClientOutput node. For more information, see "Creating a message flow" on page 1431.
2. Connect the Out terminal of the FileInput node to the In terminal of the TCPIPClientOutput node.
3. Connect the End of Data terminal of the FileInput node to the Close terminal of the TCPIPClientOutput node.



4. Set the following properties of the FileInput node:
   a. On the **Basic** tab, set the `Input directory` property to `c:\temp\task5`.
   b. On the **Records and elements** tab, set the following properties:
      - Set the `Record detection` property to `Fixed length`.
      - Set the `Length` property to `100`.
5. Set the following properties of the TCPIPClientOutput node:
   a. On the **Basic** tab, set the `Connection details` property to `14145`.
   b. On the **Advanced** tab, set the `Close connection` property to `After data has been sent`.
   c. On the **Records and elements** tab, set the following properties:
      - Set the `Record definition` property to `Record is delimited data`.
      - Set the `Delimiter` property to `Custom delimiter (Hexadecimal)`.
      - Set the `Custom delimiter` property to `00FF`.
6. Save the message flow.

Related concepts:

"WebSphere Broker TCP/IP Transport" on page 1735
The WebSphere Broker TCP/IP Transport is a service that connects applications that use raw TCP/IP sockets for transferring data.

"TCP/IP nodes" on page 1738
WebSphere Message Broker implements access to the TCP/IP input and output streams through a series of nodes.

"Scenarios: TCP/IP" on page 1746
Two scenarios illustrate how TCP/IP might be used as part of a business solution.

"Scenarios: Message Broker using TCP/IP" on page 1749
WebSphere Message Broker can be added to systems that use TCP/IP for transport, to generate a more flexible architecture for communication between components.

Related tasks:

"Working with TCP/IP" on page 1750
You can use WebSphere Message Broker TCPIP nodes and TCP/IP configurable services to perform various tasks.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

Related reference:

"TCPIPClientInput node" on page 4854
Use the TCPIPClientInput node to create a client connection to a raw TCP/IP socket, and to receive data over that connection.

"TCPIPClientOutput node" on page 4867
Use the TCPIPClientOutput node to create a client connection to a raw TCP/IP socket, and to send data over that connection to an external application.

"TCPIPClientReceive node" on page 4877
Use the TCPIPClientReceive node to receive data over a client TCP/IP connection.

"TCPIPServerInput node" on page 4890
Use the TCPIPServerInput node to create a server connection to a raw TCPIP socket, and to receive data over that connection.

"TCPIPServerOutput node" on page 4903
Use the TCPIPServerOutput node to create a server connection to a raw TCP/IP socket, and to send data over the connection to an external application.

"TCPIPServerReceive node" on page 4913
Use the TCPIPServerReceive node to receive data over a server TCP/IP connection.

"FileInput node" on page 4415
Use the FileInput node to process messages that are read from files.

Sending data to a TCP/IP client connection and receiving data back on the same connection (synchronous):

Send fixed-size data to a TCP/IP client connection and receive fixed-size data back on the same connection (synchronously), by the use of a message flow with MQInput, TCPIPClientOutput, TCPIPClientReceive, and MQOutput nodes.

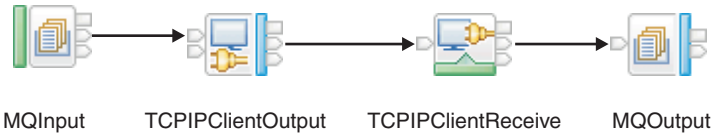
About this task

Scenario: An application sends synchronous data between TCP/IP client connections.

Instructions: The following steps describe how to create a message flow that sends data out from a client connection and waits on the same connection for a reply to be returned. The request is synchronous in the same flow, because the TCPIPClientReceive node waits for data to be returned.

Procedure

1. Create a message flow called TCPIP_Task6 with an MQInput node, a TCPIPClientOutput node, a TCPIPClientReceive node, and an MQOutput node. For more information, see "Creating a message flow" on page 1431.
2. Connect the Out terminal of the MQInput node to the In terminal of the TCPIPClientOutput node.
3. Connect the Out terminal of the TCPIPClientOutput node to the In terminal of the TCPIPClientReceive node.
4. Connect the Out terminal of the TCPIPClientReceive node to the In terminal of the MQOutput node.



5. On the MQInput node, set the `Queue name` property (on the **Basic** tab) to `TCPIP.TASK6.IN1`.
6. Set the following properties of the TCPIPClientOutput node:
   a. On the **Basic** tab, set the `Connection details` property to `14146`.
   b. On the **Records and elements** tab, set the following properties:
      - Set the `Record detection` property to `Fixed length`.
      - Set the `Length` property to `100`.
7. Set the following properties of the TCPIPClientReceive node:
   a. On the **Basic** tab, set the `Connection details` property to `14146`.
   b. On the **Advanced** tab, set the following properties:
      - Set the `Output stream modification` property to `Reserve output stream and release at end of flow`.
      - Set the `Input stream modification` property to `Reserve input stream and release at end of flow`.
   c. On the **Request** tab, set the `ID location` property to `$LocalEnvironment/WrittenDestination/TCPIP/Output/ConnectionDetails[1]/Id`.
   d. On the **Records and elements** tab, set the following properties:
      - Set the `Record detection` property to `Fixed length`.
      - Set the `Length` property to `100`.
8. On the MQOutput node, set the `Queue name` property (on the **Basic** tab) to `TCPIP.TASK6.OUT1`.
9. Save the message flow.

What to do next

See the following sample for more information:

- TCPIP Client Nodes

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Related concepts:

"WebSphere Broker TCP/IP Transport" on page 1735
The WebSphere Broker TCP/IP Transport is a service that connects applications that use raw TCP/IP sockets for transferring data.

"TCP/IP nodes" on page 1738
WebSphere Message Broker implements access to the TCP/IP input and output streams through a series of nodes.

"Scenarios: TCP/IP" on page 1746
Two scenarios illustrate how TCP/IP might be used as part of a business solution.

"Scenarios: Message Broker using TCP/IP" on page 1749
WebSphere Message Broker can be added to systems that use TCP/IP for transport, to generate a more flexible architecture for communication between components.

Related tasks:

"Working with TCP/IP" on page 1750
You can use WebSphere Message Broker TCPIP nodes and TCP/IP configurable services to perform various tasks.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

Related reference:

"TCPIPClientInput node" on page 4854
Use the TCPIPClientInput node to create a client connection to a raw TCP/IP socket, and to receive data over that connection.

"TCPIPClientOutput node" on page 4867
Use the TCPIPClientOutput node to create a client connection to a raw TCP/IP socket, and to send data over that connection to an external application.

"TCPIPClientReceive node" on page 4877
Use the TCPIPClientReceive node to receive data over a client TCP/IP connection.

"TCPIPServerInput node" on page 4890
Use the TCPIPServerInput node to create a server connection to a raw TCPIP socket, and to receive data over that connection.

"TCPIPServerOutput node" on page 4903
Use the TCPIPServerOutput node to create a server connection to a raw TCP/IP socket, and to send data over the connection to an external application.

"TCPIPServerReceive node" on page 4913
Use the TCPIPServerReceive node to receive data over a server TCP/IP connection.

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

"MQOutput node" on page 4612
Use the MQOutput node to send messages to clients that connect to the broker using the WebSphere MQ Enterprise Transport and that use the MQI and AMI application programming interfaces.

Sending data to a TCP/IP client connection and receiving data back on the same connection (asynchronous):

Send fixed-size data to a TCP/IP client connection and receive fixed-size data back on the same connection (asynchronously), by the use of a message flow with MQInput, TCPIPClientOutput, TCPIPClientInput, and MQOutput nodes.

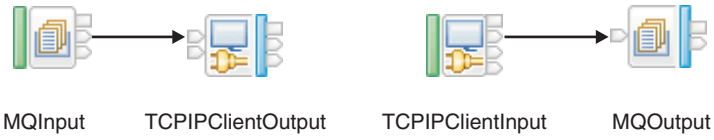
About this task

Scenario: An application sends asynchronous data between TCP/IP client connections.

Instructions: The following steps describe how to create a message flow to send data through a client connection and wait on the same connection for a reply to be returned. The request is performed asynchronously in two different flows (the TCPIPClientInput does not wait for data to be returned on this connection, but instead monitors all available connections).

Procedure

1. Create a message flow called TCPIP_Task7 with an MQInput node, a TCPIPClientOutput node, a TCPIPClientInput node, and an MQOutput node. For more information, see "Creating a message flow" on page 1431.
2. Connect the Out terminal of the MQInput node to the In terminal of the TCPIPClientOutput node.
3. Connect the Out terminal of the TCPIPClientInput node to the In terminal of the MQOutput node.



4. On the MQInput node, set the `Queue name` property (on the **Basic** tab) to `TCPIP.TASK7.IN1`.
5. Set the following properties of the TCPIPClientOutput node:
   a. On the **Basic** tab, set the `Connection details` property to `14147`.
   b. On the **Advanced** tab, set the `Output stream modification` property to `Reserve output stream`.
   c. On the **Records and elements** tab, set the following properties:
      - Set the `Record definition` property to `Fixed length`.
      - Set the `Length` property to `100`.
6. Set the following properties of the TCPIPClientInput node:
   a. On the **Basic** tab, set the `Connection details` property to `14147`.
   b. On the **Advanced** tab, set the `Output stream modification` property to `Release output stream and reset Reply ID`.
   c. On the **Records and elements** tab, set the following properties:
      - Set the `Record detection` property to `Fixed length`.
      - Set the `Length` property to `100`.
7. On the MQOutput node, set the `Queue name` property (on the **Basic** tab) to `TCPIP.TASK7.OUT1`.
8. Save the message flow.

What to do next

See the following sample for more information:

- TCPIP Client Nodes

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Related concepts:

"WebSphere Broker TCP/IP Transport" on page 1735
The WebSphere Broker TCP/IP Transport is a service that connects applications that use raw TCP/IP sockets for transferring data.

"TCP/IP nodes" on page 1738
WebSphere Message Broker implements access to the TCP/IP input and output streams through a series of nodes.

"Scenarios: TCP/IP" on page 1746
Two scenarios illustrate how TCP/IP might be used as part of a business solution.

"Scenarios: Message Broker using TCP/IP" on page 1749
WebSphere Message Broker can be added to systems that use TCP/IP for transport, to generate a more flexible architecture for communication between components.

Related tasks:

"Working with TCP/IP" on page 1750
You can use WebSphere Message Broker TCPIP nodes and TCP/IP configurable services to perform various tasks.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

Related reference:

"TCPIPClientInput node" on page 4854
Use the TCPIPClientInput node to create a client connection to a raw TCP/IP socket, and to receive data over that connection.

"TCPIPClientOutput node" on page 4867
Use the TCPIPClientOutput node to create a client connection to a raw TCP/IP socket, and to send data over that connection to an external application.

"TCPIPClientReceive node" on page 4877
Use the TCPIPClientReceive node to receive data over a client TCP/IP connection.

"TCPIPServerInput node" on page 4890
Use the TCPIPServerInput node to create a server connection to a raw TCPIP socket, and to receive data over that connection.

"TCPIPServerOutput node" on page 4903
Use the TCPIPServerOutput node to create a server connection to a raw TCP/IP socket, and to send data over the connection to an external application.

"TCPIPServerReceive node" on page 4913
Use the TCPIPServerReceive node to receive data over a server TCP/IP connection.

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

"MQOutput node" on page 4612
Use the MQOutput node to send messages to clients that connect to the broker using the WebSphere MQ Enterprise Transport and that use the MQI and AMI application programming interfaces.

Establishing a client session over a TCP/IP connection:

Configure a TCPIPClientInput node to open a session over an existing TCP/IP connection, before any data is sent or received.

About this task

You can use the Open terminal of the TCPIPClientInput node to enable processing to start when a connection is opened, rather than when data first arrives. If the Open terminal is connected, an empty message is sent to the Open terminal when a connection is created. This message has the local environment set to the connection details.

The connection associated with the message is reserved from the general connection pool until propagation to the Open terminal has finished. However, the connection can be accessed using the connectionId specified in the local environment. Each connection that is created is sent to the Open terminal, including any connections that are created mid-flow by a TCPIPClientReceive node or TCPIPClientOutput node.

If the Open terminal is not attached, open events are automatically made available in the connection pool.

The following steps show how to configure a message flow that contains a TCPIPClientInput node, with the Open terminal configured to start processing when a TCP/IP connection is created.

Procedure

1. Create a message flow containing a TCPIPClientInput node, a Compute node, and a TCPIPClientOutput node. For information about how to do this, see "Creating a message flow" on page 1431.
2. Connect the Open terminal of the TCPIPClientInput node to the In terminal of the Compute node.
3. Connect the Out terminal of the TCPIPClientInput node to the In terminal of the TCPIPClientOutput node.
4. On the TCPIPClientInput node, set the `Connection details` property (on the **Basic** tab) to `14143`.
5. On the Compute node, set the `ESQL` property (on the **Basic** tab) to:

```
CREATE COMPUTE MODULE test_Compute1
 CREATE FUNCTION Main() RETURNS BOOLEAN
 BEGIN
  -- CALL CopyMessageHeaders();
  CALL CopyEntireMessage();
  RETURN TRUE;
 END;

 CREATE PROCEDURE CopyMessageHeaders() BEGIN
  DECLARE I INTEGER 1;
  DECLARE J INTEGER;
  SET J = CARDINALITY(InputRoot.*[]);
  WHILE I < J DO
   SET OutputRoot.*[I] = InputRoot.*[I];
```

```
    SET I = I + 1;
   END WHILE;
  END;

  CREATE PROCEDURE CopyEntireMessage() BEGIN
   SET OutputRoot = InputRoot;
  END;
 END MODULE;
```

6. Set the following properties of the TCPIPClientOutput node:
   a. On the **Basic** tab, set the `Connection details` property to `14143`.
   b. On the **Request** tab, set the `ID location` property to `LocalEnvironment/TCPIP/Input/ConnectionDetails/Id`.
7. Save the message flow.

Related concepts:

"WebSphere Broker TCP/IP Transport" on page 1735
The WebSphere Broker TCP/IP Transport is a service that connects applications that use raw TCP/IP sockets for transferring data.

"TCP/IP nodes" on page 1738
WebSphere Message Broker implements access to the TCP/IP input and output streams through a series of nodes.

"Scenarios: TCP/IP" on page 1746
Two scenarios illustrate how TCP/IP might be used as part of a business solution.

"Scenarios: Message Broker using TCP/IP" on page 1749
WebSphere Message Broker can be added to systems that use TCP/IP for transport, to generate a more flexible architecture for communication between components.

Related tasks:

"Working with TCP/IP" on page 1750
You can use WebSphere Message Broker TCPIP nodes and TCP/IP configurable services to perform various tasks.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

Related reference:

"TCPIPClientInput node" on page 4854
Use the TCPIPClientInput node to create a client connection to a raw TCP/IP socket, and to receive data over that connection.

"TCPIPClientOutput node" on page 4867
Use the TCPIPClientOutput node to create a client connection to a raw TCP/IP socket, and to send data over that connection to an external application.

"TCPIPClientReceive node" on page 4877
Use the TCPIPClientReceive node to receive data over a client TCP/IP connection.

"TCPIPServerInput node" on page 4890
Use the TCPIPServerInput node to create a server connection to a raw TCPIP socket, and to receive data over that connection.

"TCPIPServerOutput node" on page 4903
Use the TCPIPServerOutput node to create a server connection to a raw TCP/IP socket, and to send data over the connection to an external application.

"TCPIPServerReceive node" on page 4913
Use the TCPIPServerReceive node to receive data over a server TCP/IP connection.

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

Broadcasting data to all currently available connections:

Broadcast data to all current connections, by the use of a message flow with MQInput and TCPIPServerOutput nodes.

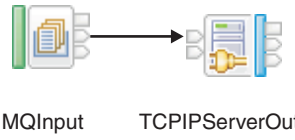
About this task

Scenario: Several applications connect into the message flow and wait to be sent data.

Instructions: The following steps describe how to send data to all the connected client applications:

Procedure

1. Create a message flow called TCPIP_Task8 with an MQInput node and a TCPIPServerOutput node. For more information, see "Creating a message flow" on page 1431.
2. Connect the Out terminal of the MQInput node to the In terminal of the TCPIPServerOutput node.



3. On the MQInput node, set the `Queue name` property (on the **Basic** tab) to `TCPIP.TASK8.IN1`.
4. Set the following properties of the TCPIPServerOutput node:
   a. On the **Basic** tab, set the `Connection details` property to `14148`.
   b. On the **Advanced** tab, set the `Send to:` property (in the **Broadcast options** group) to `All available connections`.
5. Save the message flow.

Related concepts:

"WebSphere Broker TCP/IP Transport" on page 1735
The WebSphere Broker TCP/IP Transport is a service that connects applications that use raw TCP/IP sockets for transferring data.

"TCP/IP nodes" on page 1738
WebSphere Message Broker implements access to the TCP/IP input and output streams through a series of nodes.

"Scenarios: TCP/IP" on page 1746
Two scenarios illustrate how TCP/IP might be used as part of a business solution.

"Scenarios: Message Broker using TCP/IP" on page 1749
WebSphere Message Broker can be added to systems that use TCP/IP for transport, to generate a more flexible architecture for communication between components.

Related tasks:

"Working with TCP/IP" on page 1750
You can use WebSphere Message Broker TCPIP nodes and TCP/IP configurable services to perform various tasks.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

Related reference:

"TCPIPClientInput node" on page 4854
Use the TCPIPClientInput node to create a client connection to a raw TCP/IP socket, and to receive data over that connection.

"TCPIPClientOutput node" on page 4867
Use the TCPIPClientOutput node to create a client connection to a raw TCP/IP socket, and to send data over that connection to an external application.

"TCPIPClientReceive node" on page 4877
Use the TCPIPClientReceive node to receive data over a client TCP/IP connection.

"TCPIPServerInput node" on page 4890
Use the TCPIPServerInput node to create a server connection to a raw TCPIP socket, and to receive data over that connection.

"TCPIPServerOutput node" on page 4903
Use the TCPIPServerOutput node to create a server connection to a raw TCP/IP socket, and to send data over the connection to an external application.

"TCPIPServerReceive node" on page 4913
Use the TCPIPServerReceive node to receive data over a server TCP/IP connection.

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

Configuring a server socket so that connections expire after a specified time:

Configure a TCP/IP server socket so that connections expire after a specified time, by the use of the **mqsicreateconfigurableservice** command.

About this task

Scenario: The TCPIPServer configurable service called `Task9` is configured to run on port `14149`. The connections expire when they have not been used for 5 seconds.

Instructions: Use the **mqsicreateconfigurableservice** command to set up connections that expire when they have not been used for a specified length of time. The TCP/IP node can specify either the port to be used or the name of the configurable service. For example:

```
mqsicreateconfigurableservice BRK6 -c TCPIPServer -o Task9
-n Port,ExpireConnectionSec -v 14149,5
```

Related concepts:

"WebSphere Broker TCP/IP Transport" on page 1735
The WebSphere Broker TCP/IP Transport is a service that connects applications that use raw TCP/IP sockets for transferring data.

"TCP/IP nodes" on page 1738
WebSphere Message Broker implements access to the TCP/IP input and output streams through a series of nodes.

"Scenarios: TCP/IP" on page 1746
Two scenarios illustrate how TCP/IP might be used as part of a business solution.

"Scenarios: Message Broker using TCP/IP" on page 1749
WebSphere Message Broker can be added to systems that use TCP/IP for transport, to generate a more flexible architecture for communication between components.

Related tasks:

"Working with TCP/IP" on page 1750
You can use WebSphere Message Broker TCPIP nodes and TCP/IP configurable services to perform various tasks.

Related reference:

"TCPIPClientInput node" on page 4854
Use the TCPIPClientInput node to create a client connection to a raw TCP/IP socket, and to receive data over that connection.

"TCPIPClientOutput node" on page 4867
Use the TCPIPClientOutput node to create a client connection to a raw TCP/IP socket, and to send data over that connection to an external application.

"TCPIPClientReceive node" on page 4877
Use the TCPIPClientReceive node to receive data over a client TCP/IP connection.

"TCPIPServerInput node" on page 4890
Use the TCPIPServerInput node to create a server connection to a raw TCPIP socket, and to receive data over that connection.

"TCPIPServerOutput node" on page 4903
Use the TCPIPServerOutput node to create a server connection to a raw TCP/IP socket, and to send data over the connection to an external application.

"TCPIPServerReceive node" on page 4913
Use the TCPIPServerReceive node to receive data over a server TCP/IP connection.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

Configuring a client socket to make 100 connections at deployment or startup time:

Configure a TCP/IP client socket to make 100 connections at deployment or startup time, by using the **mqsicreateconfigurableservice** command.

About this task

Use the **mqsicreateconfigurableservice** command to set up the client broker to establish 100 connections when it is created. By default, the client connections are not made until they are required by one of the TCP/IP nodes. For example:

```
mqsicreateconfigurableservice MB7BROKER -c TCPIPClient -o Task10
-n Port,MinimumConnections -v 14150,100
```

In this example, the TCPIPClient configurable service called `Task10` is configured to run on port `14150`, and 100 connections are created.

Related concepts:

"WebSphere Broker TCP/IP Transport" on page 1735
The WebSphere Broker TCP/IP Transport is a service that connects applications that use raw TCP/IP sockets for transferring data.

"TCP/IP nodes" on page 1738
WebSphere Message Broker implements access to the TCP/IP input and output streams through a series of nodes.

"Scenarios: TCP/IP" on page 1746
Two scenarios illustrate how TCP/IP might be used as part of a business solution.

"Scenarios: Message Broker using TCP/IP" on page 1749
WebSphere Message Broker can be added to systems that use TCP/IP for transport, to generate a more flexible architecture for communication between components.

Related tasks:

"Working with TCP/IP" on page 1750
You can use WebSphere Message Broker TCPIP nodes and TCP/IP configurable services to perform various tasks.

Related reference:

"TCPIPClientInput node" on page 4854
Use the TCPIPClientInput node to create a client connection to a raw TCP/IP socket, and to receive data over that connection.

"TCPIPClientOutput node" on page 4867
Use the TCPIPClientOutput node to create a client connection to a raw TCP/IP socket, and to send data over that connection to an external application.

"TCPIPClientReceive node" on page 4877
Use the TCPIPClientReceive node to receive data over a client TCP/IP connection.

"TCPIPServerInput node" on page 4890
Use the TCPIPServerInput node to create a server connection to a raw TCPIP socket, and to receive data over that connection.

"TCPIPServerOutput node" on page 4903
Use the TCPIPServerOutput node to create a server connection to a raw TCP/IP socket, and to send data over the connection to an external application.

"TCPIPServerReceive node" on page 4913
Use the TCPIPServerReceive node to receive data over a server TCP/IP connection.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

Configuring a server socket to receive XML data ending in a null character:

Configure a server TCP/IP socket to receive XML data ending in a null character, by the use of a message flow with TCPIPServerInput and MQOutput nodes.

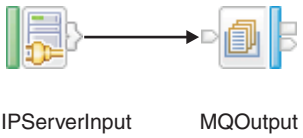
About this task

Scenario: A client application sends XML data that is delimited by a null character (hex code '00').

Instructions: The following steps describe how to break up the record based on the null character, then parse the data.

Procedure

1. Create a message flow called TCPIP_Task11 with a TCPIPServerInput node and an MQOutput node. For more information, see "Creating a message flow" on page 1431.
2. Connect the Out terminal of the TCPIPServerInput node to the In terminal of the MQOutput node.



3. Set the following properties of the TCPIPServerInput node:
   a. On the **Basic** tab, set the `Connection details` property to `14151`.
   b. On the **Input Message Parsing** tab, set the `Message domain` property to `XMLNSC`.
   c. On the **Records and elements** tab, set the following properties:
      - Set the `Record detection` property to `Delimited`.
      - Set the `Delimiter` property to `Custom delimiter`.
      - Set the `Custom delimiter` property to `00`.
4. On the MQOutput node, set the `Queue name` property (on the **Basic** tab) to `TCPIP.TASK11.IN1`.
5. Save the message flow.

Related concepts:

"WebSphere Broker TCP/IP Transport" on page 1735
The WebSphere Broker TCP/IP Transport is a service that connects applications that use raw TCP/IP sockets for transferring data.

"TCP/IP nodes" on page 1738
WebSphere Message Broker implements access to the TCP/IP input and output streams through a series of nodes.

"Scenarios: TCP/IP" on page 1746
Two scenarios illustrate how TCP/IP might be used as part of a business solution.

"Scenarios: Message Broker using TCP/IP" on page 1749
WebSphere Message Broker can be added to systems that use TCP/IP for transport, to generate a more flexible architecture for communication between

components.

Related tasks:

"Working with TCP/IP" on page 1750
You can use WebSphere Message Broker TCPIP nodes and TCP/IP configurable services to perform various tasks.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

Related reference:

"TCPIPClientInput node" on page 4854
Use the TCPIPClientInput node to create a client connection to a raw TCP/IP socket, and to receive data over that connection.

"TCPIPClientOutput node" on page 4867
Use the TCPIPClientOutput node to create a client connection to a raw TCP/IP socket, and to send data over that connection to an external application.

"TCPIPClientReceive node" on page 4877
Use the TCPIPClientReceive node to receive data over a client TCP/IP connection.

"TCPIPServerInput node" on page 4890
Use the TCPIPServerInput node to create a server connection to a raw TCPIP socket, and to receive data over that connection.

"TCPIPServerOutput node" on page 4903
Use the TCPIPServerOutput node to create a server connection to a raw TCP/IP socket, and to send data over the connection to an external application.

"TCPIPServerReceive node" on page 4913
Use the TCPIPServerReceive node to receive data over a server TCP/IP connection.

"MQOutput node" on page 4612
Use the MQOutput node to send messages to clients that connect to the broker using the WebSphere MQ Enterprise Transport and that use the MQI and AMI application programming interfaces.

Configuring a server socket to receive XML data and discover the end of a record (by using the message model):

Configure a server socket to receive XML data and use the message model to determine the end of a record, by the use of a message flow with TCPIPServerInput and MQOutput nodes.

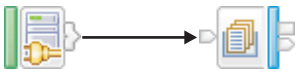
About this task

Scenario: A client application sends an XML document with no clear indication of the end of the record.

Instructions: The following steps show how to break up the record by the use of the XML parser to signal when the whole XML document as been received. The parser uses the end XML tag to signal the end of the message.

Procedure

1. Create a message flow called TCPIP_Task12 with a TCPIPServerInput node and an MQOutput node. For more information, see "Creating a message flow" on page 1431.
2. Connect the Out terminal of the TCPIPServerInput node to the In terminal of the MQOutput node.



3. Set the following properties of the TCPIPServerInput node:
   a. On the **Basic** tab, set the `Connection details` property to `14151`.
   b. On the **Input Message Parsing** tab, set the `Message domain` property to `XMLNSC`.
   c. On the **Records and elements** tab, set the `Record detection` property to `Parsed record sequence`.
4. On the MQOutput node, set the `Queue name` property (on the **Basic** tab) to `TCPIP.TASK12.IN1`.
5. Save the message flow.

Related concepts:

"WebSphere Broker TCP/IP Transport" on page 1735
The WebSphere Broker TCP/IP Transport is a service that connects applications that use raw TCP/IP sockets for transferring data.

"TCP/IP nodes" on page 1738
WebSphere Message Broker implements access to the TCP/IP input and output streams through a series of nodes.

"Scenarios: TCP/IP" on page 1746
Two scenarios illustrate how TCP/IP might be used as part of a business solution.

"Scenarios: Message Broker using TCP/IP" on page 1749
WebSphere Message Broker can be added to systems that use TCP/IP for transport, to generate a more flexible architecture for communication between components.

Related tasks:

"Working with TCP/IP" on page 1750
You can use WebSphere Message Broker TCPIP nodes and TCP/IP configurable services to perform various tasks.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

Related reference:

"TCPIPClientInput node" on page 4854
Use the TCPIPClientInput node to create a client connection to a raw TCP/IP socket, and to receive data over that connection.

"TCPIPClientOutput node" on page 4867
Use the TCPIPClientOutput node to create a client connection to a raw TCP/IP socket, and to send data over that connection to an external application.

"TCPIPClientReceive node" on page 4877
Use the TCPIPClientReceive node to receive data over a client TCP/IP connection.

"TCPIPServerInput node" on page 4890
Use the TCPIPServerInput node to create a server connection to a raw TCPIP socket, and to receive data over that connection.

"TCPIPServerOutput node" on page 4903
Use the TCPIPServerOutput node to create a server connection to a raw TCP/IP socket, and to send data over the connection to an external application.

"TCPIPServerReceive node" on page 4913
Use the TCPIPServerReceive node to receive data over a server TCP/IP connection.

"MQOutput node" on page 4612
Use the MQOutput node to send messages to clients that connect to the broker using the WebSphere MQ Enterprise Transport and that use the MQI and AMI application programming interfaces.

Configuring a server output node to close all connections:

Configure a server output node to close all connections on a specified port.

About this task

Scenario: A server output node is configured to close all connections on a specified port.

Instructions: The following steps show how to create a message flow that closes all TCP/IP connections on port 14153:

Procedure

1. Create a message flow called TCPIP_Task13 with an MQInput node and a TCPIPServerOutput node. For more information, see "Creating a message flow" on page 1431.
2. Connect the Out terminal of the MQInput node to the Close terminal of the TCPIPServerOutput node.

3. On the MQInput node, set the `Queue name` property (on the **Basic** tab) to `TCPIP.TASK13.IN1`.
4. Set the following properties of the TCPIPServerOutput node:
   a. On the **Basic** tab, set the `Connection details` property to `14153`.
   b. On the **Advanced** tab, set the `Send to:` property (in the **Broadcast options** group) to `All available connections`.
5. Save the message flow.

Related concepts:

"WebSphere Broker TCP/IP Transport" on page 1735
The WebSphere Broker TCP/IP Transport is a service that connects applications that use raw TCP/IP sockets for transferring data.

"TCP/IP nodes" on page 1738
WebSphere Message Broker implements access to the TCP/IP input and output streams through a series of nodes.

"Scenarios: TCP/IP" on page 1746
Two scenarios illustrate how TCP/IP might be used as part of a business solution.

"Scenarios: Message Broker using TCP/IP" on page 1749
WebSphere Message Broker can be added to systems that use TCP/IP for transport, to generate a more flexible architecture for communication between components.

Related tasks:

"Working with TCP/IP" on page 1750
You can use WebSphere Message Broker TCPIP nodes and TCP/IP configurable services to perform various tasks.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

Related reference:

"TCPIPClientInput node" on page 4854
Use the TCPIPClientInput node to create a client connection to a raw TCP/IP socket, and to receive data over that connection.

"TCPIPClientOutput node" on page 4867
Use the TCPIPClientOutput node to create a client connection to a raw TCP/IP socket, and to send data over that connection to an external application.

"TCPIPClientReceive node" on page 4877
Use the TCPIPClientReceive node to receive data over a client TCP/IP connection.

"TCPIPServerInput node" on page 4890
Use the TCPIPServerInput node to create a server connection to a raw TCPIP socket, and to receive data over that connection.

"TCPIPServerOutput node" on page 4903
Use the TCPIPServerOutput node to create a server connection to a raw TCP/IP socket, and to send data over the connection to an external application.

"TCPIPServerReceive node" on page 4913
Use the TCPIPServerReceive node to receive data over a server TCP/IP connection.

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

Configuring a client socket to store reply correlation details:

Configure a client TCP/IP socket to store reply correlation details, by the use of a message flow with MQInput and TCPIPClientOutput nodes.

About this task

Scenario: A client TCP/IP socket is configured to store response returned on the input stream.

Instructions: The following steps show how to set the Reply ID on a connection, which can be used when a response is returned on the input stream:

Procedure

1. Create a message flow called TCPIP_Task14 with an MQInput node and a TCPIPClientOutput node. For more information, see "Creating a message flow" on page 1431.
2. Connect the Out terminal of the MQInput node to the In terminal of the TCPIPClientOutput node.

3. On the MQInput node, set the `Queue name` property (on the **Basic** tab) to `TCPIP.TASK14.IN1`.
4. Set the following properties of the TCPIPClientOutput node:
   a. On the **Basic** tab, set the `Connection details` property to `14154`.
   b. On the **Request** tab, set the `Reply ID location` property to `$Root/MQMD/MsgId`.
5. Save the message flow.

Related concepts:

"WebSphere Broker TCP/IP Transport" on page 1735
The WebSphere Broker TCP/IP Transport is a service that connects applications that use raw TCP/IP sockets for transferring data.

"TCP/IP nodes" on page 1738
WebSphere Message Broker implements access to the TCP/IP input and output streams through a series of nodes.

"Scenarios: TCP/IP" on page 1746
Two scenarios illustrate how TCP/IP might be used as part of a business solution.

"Scenarios: Message Broker using TCP/IP" on page 1749
WebSphere Message Broker can be added to systems that use TCP/IP for transport, to generate a more flexible architecture for communication between components.

Related tasks:

"Working with TCP/IP" on page 1750
You can use WebSphere Message Broker TCPIP nodes and TCP/IP configurable services to perform various tasks.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

Related reference:

"TCPIPClientInput node" on page 4854
Use the TCPIPClientInput node to create a client connection to a raw TCP/IP socket, and to receive data over that connection.

"TCPIPClientOutput node" on page 4867
Use the TCPIPClientOutput node to create a client connection to a raw TCP/IP socket, and to send data over that connection to an external application.

"TCPIPClientReceive node" on page 4877
Use the TCPIPClientReceive node to receive data over a client TCP/IP connection.

"TCPIPServerInput node" on page 4890
Use the TCPIPServerInput node to create a server connection to a raw TCPIP socket, and to receive data over that connection.

"TCPIPServerOutput node" on page 4903
Use the TCPIPServerOutput node to create a server connection to a raw TCP/IP socket, and to send data over the connection to an external application.

"TCPIPServerReceive node" on page 4913
Use the TCPIPServerReceive node to receive data over a server TCP/IP connection.

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

Writing close connection details to a file:

Configure a message flow to write details of a connection closure to a file, by the use of TCPIPServerInput, Compute, and FileOutput or FTEOutput nodes.

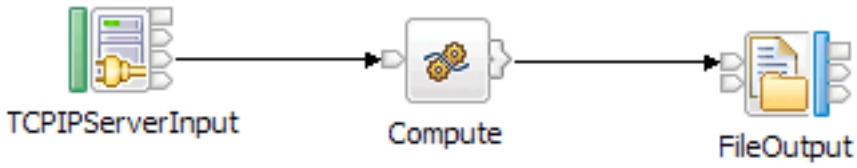
About this task

Scenario: A message flow writes close connection details to a file. The scenario uses the FileOutput node; the steps shown also apply to the FTEOutput node.

Instructions The following steps show how to configure a message flow to write details of the closure of any connection to a file:

Procedure

1. Create a message flow called TCPIP_Task15 with a TCPIPServerInput node, a Compute node, and a FileOutput node. For more information, see "Creating a message flow" on page 1431.
2. Connect the Close terminal of the TCPIPServerInput node to the In terminal of the Compute node.
3. Connect the Out terminal of the Compute node to the In terminal of the FileOutput node.



4. On the TCPIPServerInput node, set the `Connection details` property (on the **Basic** tab) to `14155`.
5. On the Compute node, set the `ESQL` property (on the **Basic** tab) to:

```
BROKER SCHEMA Tasks


CREATE COMPUTE MODULE TCPIP_Task15_Compute
 CREATE FUNCTION Main() RETURNS BOOLEAN
 BEGIN
  -- CALL CopyMessageHeaders();
  -- CALL CopyEntireMessage();
  Set OutputRoot.XMLNSC.CloseEvent = InputLocalEnvironment.TCPIP;
  RETURN TRUE;
 END;

 CREATE PROCEDURE CopyMessageHeaders() BEGIN
  DECLARE I INTEGER 1;
  DECLARE J INTEGER;
  SET J = CARDINALITY(InputRoot.*[]);
  WHILE I < J DO
   SET OutputRoot.*[I] = InputRoot.*[I];
   SET I = I + 1;
  END WHILE;
 END;
```

```
 CREATE PROCEDURE CopyEntireMessage() BEGIN
  SET OutputRoot = InputRoot;
 END;
END MODULE;
```

6. Set the following properties of the FileOutput node.
   a. On the **Basic** tab, set the following properties:
      - Set the `Directory` property to `c:\temp\Task15`.
      - Set the `File name or pattern` property to `CloseEvents.txt`.
   b. On the **Records and elements** tab, set the `Record definition` property to `Record is unmodified data`.
7. Save the message flow.

Related concepts:

"WebSphere Broker TCP/IP Transport" on page 1735
The WebSphere Broker TCP/IP Transport is a service that connects applications that use raw TCP/IP sockets for transferring data.

"TCP/IP nodes" on page 1738
WebSphere Message Broker implements access to the TCP/IP input and output streams through a series of nodes.

"Scenarios: TCP/IP" on page 1746
Two scenarios illustrate how TCP/IP might be used as part of a business solution.

"Scenarios: Message Broker using TCP/IP" on page 1749
WebSphere Message Broker can be added to systems that use TCP/IP for transport, to generate a more flexible architecture for communication between components.

Related tasks:

"Working with TCP/IP" on page 1750
You can use WebSphere Message Broker TCPIP nodes and TCP/IP configurable services to perform various tasks.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

Related reference:

"TCPIPClientInput node" on page 4854
Use the TCPIPClientInput node to create a client connection to a raw TCP/IP socket, and to receive data over that connection.

"TCPIPClientOutput node" on page 4867
Use the TCPIPClientOutput node to create a client connection to a raw TCP/IP socket, and to send data over that connection to an external application.

"TCPIPClientReceive node" on page 4877
Use the TCPIPClientReceive node to receive data over a client TCP/IP connection.

"TCPIPServerInput node" on page 4890
Use the TCPIPServerInput node to create a server connection to a raw TCPIP socket, and to receive data over that connection.

"TCPIPServerOutput node" on page 4903
Use the TCPIPServerOutput node to create a server connection to a raw TCP/IP socket, and to send data over the connection to an external application.

"TCPIPServerReceive node" on page 4913
Use the TCPIPServerReceive node to receive data over a server TCP/IP connection.

“FileOutput node” on page 4430
Use the FileOutput node to write messages to files.

“FTEOutput node” on page 4466
Use the FTEOutput node to write messages to files by using the WebSphere MQ File Transfer Edition.

“Compute node” on page 4340
Use the Compute node to construct one or more new output messages.

Configuring a client node to dynamically call a port:

Configure a client node to dynamically use a port and hostname that are set in the local environment, by the use of a message flow with MQInput, Compute, and TCPIPClientOutput nodes.

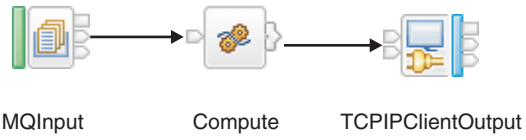
About this task

Scenario: A client node dynamically calls a port.

Instructions: The following steps show how to override the connection details specified on a client output node to dynamically use a port and hostname that are set in the local environment:

Procedure

1. Create a message flow called TCPIP_Task16 with an MQInput node, a Compute node, and a TCPIPClientOutput node. For more information, see “Creating a message flow” on page 1431.
2. Connect the Out terminal of the MQInput node to the In terminal of the Compute node.
3. Connect the Out terminal of the Compute node to the In terminal of the TCPIPClientOutput node.



4. On the MQInput node, set the `Queue name` property (on the **Basic** tab) to `TCPIP.TASK16.IN1` .
5. On the Compute node, set the `ESQL` property (on the **Basic** tab) to:

```
BROKER SCHEMA Tasks
CREATE COMPUTE MODULE TCPIP_Task16_Compute
 CREATE FUNCTION Main() RETURNS BOOLEAN
 BEGIN
  -- CALL CopyMessageHeaders();
  CALL CopyEntireMessage();
  set InputLocalEnvironment.Destination.TCPIP.Output.Hostname = 'localhost';
  set InputLocalEnvironment.Destination.TCPIP.Output.Port = 14156;
  RETURN TRUE;
 END;

 CREATE PROCEDURE CopyMessageHeaders() BEGIN
  DECLARE I INTEGER 1;
  DECLARE J INTEGER;
  SET J = CARDINALITY(InputRoot.*[]);
  WHILE I < J DO
   SET OutputRoot.*[I] = InputRoot.*[I];
   SET I = I + 1;
```

```
 END WHILE;
END;

CREATE PROCEDURE CopyEntireMessage() BEGIN
 SET OutputRoot = InputRoot;
END;
END MODULE;
```

6. On the TCPIPClientOutput node, set the `Connection details` property (on the **Basic** tab) to `9999`.
7. Save the message flow.

Related concepts:

"WebSphere Broker TCP/IP Transport" on page 1735
The WebSphere Broker TCP/IP Transport is a service that connects applications that use raw TCP/IP sockets for transferring data.

"TCP/IP nodes" on page 1738
WebSphere Message Broker implements access to the TCP/IP input and output streams through a series of nodes.

"Scenarios: TCP/IP" on page 1746
Two scenarios illustrate how TCP/IP might be used as part of a business solution.

"Scenarios: Message Broker using TCP/IP" on page 1749
WebSphere Message Broker can be added to systems that use TCP/IP for transport, to generate a more flexible architecture for communication between components.

Related tasks:

"Working with TCP/IP" on page 1750
You can use WebSphere Message Broker TCPIP nodes and TCP/IP configurable services to perform various tasks.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

Related reference:

"TCPIPClientInput node" on page 4854
Use the TCPIPClientInput node to create a client connection to a raw TCP/IP socket, and to receive data over that connection.

"TCPIPClientOutput node" on page 4867
Use the TCPIPClientOutput node to create a client connection to a raw TCP/IP socket, and to send data over that connection to an external application.

"TCPIPClientReceive node" on page 4877
Use the TCPIPClientReceive node to receive data over a client TCP/IP connection.

"TCPIPServerInput node" on page 4890
Use the TCPIPServerInput node to create a server connection to a raw TCPIP socket, and to receive data over that connection.

"TCPIPServerOutput node" on page 4903
Use the TCPIPServerOutput node to create a server connection to a raw TCP/IP socket, and to send data over the connection to an external application.

"TCPIPServerReceive node" on page 4913
Use the TCPIPServerReceive node to receive data over a server TCP/IP connection.

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

Configuring a server receive node to wait for data on a specified port:

Configure a TCPIPServerReceive node to block the message flow and wait for data to arrive on any connection.

About this task

Scenario: A server receive node is configured to wait for data on a specified port.

Instructions: The following steps show how to configure a TCPIPServerReceive node to wait for data on port 14157:

Procedure

1. Create a message flow called TCPIP_Task17 with an MQInput node and a TCPIPServerReceive node. For more information, see "Creating a message flow" on page 1431.
2. Connect the Out terminal of the MQInput node to the In terminal of the TCPIPServerReceive node.

3. On the MQInput node, set the `Queue name` property (on the **Basic** tab) to `TCPIP.TASK17.IN1`.
4. On the TCPIPServerReceive node, set the `Connection details` property (on the **Basic** tab) to `14157`.
5. Save the message flow.

Related concepts:

"WebSphere Broker TCP/IP Transport" on page 1735
The WebSphere Broker TCP/IP Transport is a service that connects applications that use raw TCP/IP sockets for transferring data.

"TCP/IP nodes" on page 1738
WebSphere Message Broker implements access to the TCP/IP input and output streams through a series of nodes.

"Scenarios: TCP/IP" on page 1746
Two scenarios illustrate how TCP/IP might be used as part of a business solution.

"Scenarios: Message Broker using TCP/IP" on page 1749
WebSphere Message Broker can be added to systems that use TCP/IP for transport, to generate a more flexible architecture for communication between components.

Related tasks:

"Working with TCP/IP" on page 1750
You can use WebSphere Message Broker TCPIP nodes and TCP/IP configurable services to perform various tasks.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

Related reference:

"TCPIPClientInput node" on page 4854
Use the TCPIPClientInput node to create a client connection to a raw TCP/IP socket, and to receive data over that connection.

"TCPIPClientOutput node" on page 4867
Use the TCPIPClientOutput node to create a client connection to a raw TCP/IP socket, and to send data over that connection to an external application.

"TCPIPClientReceive node" on page 4877
Use the TCPIPClientReceive node to receive data over a client TCP/IP connection.

"TCPIPServerInput node" on page 4890
Use the TCPIPServerInput node to create a server connection to a raw TCPIP socket, and to receive data over that connection.

"TCPIPServerOutput node" on page 4903
Use the TCPIPServerOutput node to create a server connection to a raw TCP/IP socket, and to send data over the connection to an external application.

"TCPIPServerReceive node" on page 4913
Use the TCPIPServerReceive node to receive data over a server TCP/IP connection.

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

Sending and receiving data through a TCP/IP client connection, delimiting the record by closing the output stream (asynchronous):

Send data through a TCP/IP client connection and receive data back on the same connection (asynchronously), by the use of a message flow with MQInput, TCPIPClientOutput, TCPIPClientInput, and MQOutput nodes.

About this task

Scenario: An application sends asynchronous data between TCP/IP client connections.

Instructions: The following steps describe how to create a message flow to send data through a client connection and wait on the same connection for a reply to be returned. The request is performed asynchronously in two different flows; the TCPIPClientInput node does not wait for data to be returned on this connection, but monitors all available connections. The outgoing record is delimited by closing the output stream, and the reply message is delimited by the remote server closing the input stream. The connection is then completely closed by the node.

Procedure

1. Create a message flow called TCPIP_Task18 with an MQInput node, a TCPIPClientOutput, a TCPIPClientInput node, and an MQOutput node. For more information, see "Creating a message flow" on page 1431.
2. Connect the Out terminal of the MQInput node to the In terminal of the TCPIPClientOutput node.
3. Connect the Out terminal of the TCPIPClientInput node to the In terminal of the MQOutput node.

4. On the MQInput node, set the `Queue name` property (on the **Basic** tab) to `TCPIP.TASK18.IN1`.
5. Set the following properties of the TCPIPClientOutput node:
   a. On the **Basic** tab, set the `Connection details` property to `14158`.
   b. On the **Advanced** tab, select `Close output stream after a record has been sent`.
   c. On the **Records and elements** tab, set the `Record definition` property to `Record is unmodified data`.
6. Set the following properties of the TCPIPClientInput node:
   a. On the **Basic** tab, set the `Connection details` property to `14158`.
   b. On the **Advanced** tab, set the `Close connection` property to `After data has been received`.
   c. On the **Records and elements** tab, set the `Record detection` property to `End of stream`.
7. On the MQOutput node, set the `Queue name` property (on the **Basic** tab) to `TCPIP.TASK18.OUT1`.
8. Save the message flow.

Related concepts:

"WebSphere Broker TCP/IP Transport" on page 1735
The WebSphere Broker TCP/IP Transport is a service that connects applications that use raw TCP/IP sockets for transferring data.

"TCP/IP nodes" on page 1738
WebSphere Message Broker implements access to the TCP/IP input and output streams through a series of nodes.

"Scenarios: TCP/IP" on page 1746
Two scenarios illustrate how TCP/IP might be used as part of a business solution.

"Scenarios: Message Broker using TCP/IP" on page 1749
WebSphere Message Broker can be added to systems that use TCP/IP for transport, to generate a more flexible architecture for communication between components.

Related tasks:

"Working with TCP/IP" on page 1750
You can use WebSphere Message Broker TCPIP nodes and TCP/IP configurable services to perform various tasks.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

Related reference:

"TCPIPClientInput node" on page 4854
Use the TCPIPClientInput node to create a client connection to a raw TCP/IP socket, and to receive data over that connection.

"TCPIPClientOutput node" on page 4867
Use the TCPIPClientOutput node to create a client connection to a raw TCP/IP socket, and to send data over that connection to an external application.

"TCPIPClientReceive node" on page 4877
Use the TCPIPClientReceive node to receive data over a client TCP/IP connection.

"TCPIPServerInput node" on page 4890
Use the TCPIPServerInput node to create a server connection to a raw TCPIP socket, and to receive data over that connection.

"TCPIPServerOutput node" on page 4903
Use the TCPIPServerOutput node to create a server connection to a raw TCP/IP socket, and to send data over the connection to an external application.

"TCPIPServerReceive node" on page 4913
Use the TCPIPServerReceive node to receive data over a server TCP/IP connection.

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

"MQOutput node" on page 4612
Use the MQOutput node to send messages to clients that connect to the broker using the WebSphere MQ Enterprise Transport and that use the MQI and AMI application programming interfaces.

Sending and receiving data on the same TCP/IP client connection, closing input and output streams (synchronous):

Send data through a TCP/IP client connection and wait on the same connection for a reply to be returned, by the use of a message flow with MQInput, TCPIPClientOutput, TCPIPClientReceive, and MQOutput nodes.

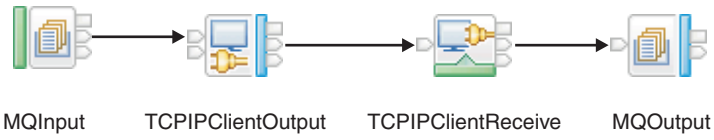
About this task

Scenario: An application sends synchronous data on the same TCP/IP client connection.

Instructions: The following steps describe how to create a message flow that sends out data through a client connection and waits on the same connection for a reply to be returned. The request is synchronous within the same flow, as a result of the TCPIPClientReceive node waiting for data to be returned. The outgoing message is delimited by closing the output stream, and the reply data is delimited by the remote application closing the input stream.

Procedure

1. Create a message flow called TCPIP_Task19 with an MQInput node, a TCPIPClientOutput node, a TCPIPClientReceive node, and an MQOutput node. For more information, see "Creating a message flow" on page 1431.
2. Connect the Out terminal of the MQInput node to the In terminal of the TCPIPClientOutput node.
3. Connect the Out terminal of the TCPIPClientOutput node to the In terminal of the TCPIPClientReceive node.
4. Connect the Out terminal of the TCPIPClientReceive node to the In terminal of the MQOutput node.



5. On the MQInput node, set the `Queue name` property (on the **Basic** tab) to `TCPIP.TASK19.IN1`.
6. Set the following properties of the TCPIPClientOutput node:
   a. On the **Basic** tab, set the `Connection details` property to `14159`.
   b. On the **Advanced** tab, set the following properties:
      - Select `Close output stream after a record has been sent`.
      - Set the `Input stream modification` property to `Reserve input stream and release at end of flow`. It is important to reserve the input stream so that it is not closed before the receive node processes the return data.
   c. On the **Records and elements** tab, set the `Record definition` property to `Record is Unmodified Data`.
7. Set the following properties of the TCPIPClientReceive node:
   a. On the **Basic** tab, set the `Connection details` property to `14159`.
   b. On the **Advanced** tab, set the `Close connection` property to `After data has been received`.
   c. On the **Request** tab, set the `ID location` property to `$LocalEnvironment/WrittenDestination/TCPIP/Output/ConnectionDetails[1]/Id`.
   d. On the **Records and elements** tab, set the `Record detection` property to `Connection closed`.
8. On the MQOutput node, set the `Queue name` property (on the **Basic** tab) to `TCPIP.TASK19.OUT1`.
9. Save the message flow.

Related concepts:

"WebSphere Broker TCP/IP Transport" on page 1735
The WebSphere Broker TCP/IP Transport is a service that connects applications that use raw TCP/IP sockets for transferring data.

"TCP/IP nodes" on page 1738
WebSphere Message Broker implements access to the TCP/IP input and output streams through a series of nodes.

"Scenarios: TCP/IP" on page 1746
Two scenarios illustrate how TCP/IP might be used as part of a business solution.

"Scenarios: Message Broker using TCP/IP" on page 1749
WebSphere Message Broker can be added to systems that use TCP/IP for transport, to generate a more flexible architecture for communication between components.

Related tasks:

"Working with TCP/IP" on page 1750
You can use WebSphere Message Broker TCPIP nodes and TCP/IP configurable services to perform various tasks.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

Related reference:

"TCPIPClientInput node" on page 4854
Use the TCPIPClientInput node to create a client connection to a raw TCP/IP socket, and to receive data over that connection.

"TCPIPClientOutput node" on page 4867
Use the TCPIPClientOutput node to create a client connection to a raw TCP/IP socket, and to send data over that connection to an external application.

"TCPIPClientReceive node" on page 4877
Use the TCPIPClientReceive node to receive data over a client TCP/IP connection.

"TCPIPServerInput node" on page 4890
Use the TCPIPServerInput node to create a server connection to a raw TCPIP socket, and to receive data over that connection.

"TCPIPServerOutput node" on page 4903
Use the TCPIPServerOutput node to create a server connection to a raw TCP/IP socket, and to send data over the connection to an external application.

"TCPIPServerReceive node" on page 4913
Use the TCPIPServerReceive node to receive data over a server TCP/IP connection.

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

"MQOutput node" on page 4612
Use the MQOutput node to send messages to clients that connect to the broker using the WebSphere MQ Enterprise Transport and that use the MQI and AMI application programming interfaces.

Processing email messages

You can configure the EmailOutput node to deliver an email from a message flow to an email server that supports Simple Mail Transfer Protocol (SMTP). You can also configure the EmailInput node to retrieve an email from an email server that supports Post Office Protocol 3 (POP3) or Internet Message Access Protocol (IMAP).

About this task

The topics in the following sections describe how the EmailOutput and the EmailInput nodes work, and how to use them to send and receive email messages.

- "Sending emails" on page 1787
- "Receiving emails" on page 1799

Sending emails

You can use the EmailOutput node to send email messages, with or without attachments, to one or more recipients. The EmailOutput node delivers an email message from your message flow to an email server that supports Simple Mail Transfer Protocol (SMTP), which you specify.

The following topics describe how the EmailOutput node works and how to use it to send email messages.

- "Sending emails" on page 1787
- "Sending an email" on page 1789
- "Sending an email with an attachment" on page 1790

- “Producing dynamic email messages” on page 1791
- “Sending a MIME message” on page 1795
- “Changing connection information for the EmailOutput node” on page 1798

Receiving emails

You can use the EmailInput node to receive email messages, with or without attachments, from one or more recipients. The EmailInput node retrieves an email message from an email server that you specify, which supports Post Office Protocol 3 (POP3) or Internet Message Access Protocol (IMAP).

The following topics describe how the EmailInput node works, and how to use it to receive email messages.

- “Receiving emails” on page 1799
- “Receiving an email” on page 1801
- “Processing responses from an EmailInput node” on page 1804
- “Changing connection information for the EmailInput node” on page 1805

Related tasks:

“Resolving problems when you use Email nodes” on page 3398
Advice for dealing with common problems that can arise when you develop message flows that contain Email nodes.

Related reference:

“EmailOutput node” on page 4400
Use the EmailOutput node to send email messages to one or more recipients.

“EmailInput node” on page 4394
Use the EmailInput node to retrieve an email, with or without attachments, from an email server that supports Post Office Protocol 3 (POP3) or Internet Message Access Protocol (IMAP).

Sending emails

You can configure WebSphere Message Broker to send an email, with or without attachments, to a static or dynamic list of recipients.

About this task

You can configure the EmailOutput node to send an email, with or without a single attachment, with a static subject and static text, to a static list of recipients. You can also construct a message flow that can produce an email where the SMTP server, list of recipients, subject, text, and multiple attachments are all determined at run time.

You can use the `SMTP Server and Port` property of the EmailOutput node to specify the host name of the SMTP server that the broker uses to send emails, or to specify an alias that refers to a configurable service that is defined on the broker. To enable the configurable service, use the **mqsicreateconfigurableservice** and **mqsichangeproperties** commands, as shown in the following example:

```
mqsicreateconfigurableservice BROKER -c SMTP -o MYSERVER
mqsichangeproperties BROKER -c SMTP -o MYSERVER -n serverName -v smtp.hursley.ibm.com:25
```

If the value of the `SMTP Server and Port` property is set to the defined alias (in this case, MYSERVER), any values that are overridden by the administrator

(smtp.hursley.ibm.com:25 set on the command line as the server name) are used in preference to any statically defined value or override value in the local environment.

The order of preference for value selection is:

1. The value that is specified in the configurable service if an alias exists with the name that was supplied in the `SMTP Server and Port` property.
2. The server name that is specified in the local environment.
3. The value of the `SMTP Server and Port` property that is specified on the node.

Any alias can be removed by using the **mqsideleteconfigurableservice** command, so that the node reverts to resolving the host name from the value that is set in the local environment or on the node. While most configurable service properties are set by the **mqsichangeproperties** command, security identities (user IDs and passwords) are typically set by using the **mqsisetdbparms** command and are referenced in a separate SecurityIdentity property, for example:

```
mqsisetdbparms BROKER -n smtp::MyIdentity -u userName -p password
mqsichangeproperties BROKER -c SMTP -o MYSERVER -n securityIdentity -v MyIdentity
```

The topics in this section describe the different ways in which you can use the EmailOutput node to send email messages.

Procedure

- To send an email with a static subject and static text to a static list of recipients, see "Sending an email" on page 1789.
- To send an email with an attachment, see "Sending an email with an attachment" on page 1790.
- To send an email where the SMTP server, list of recipients, subject, text, and multiple attachments are all determined at run time, see "Producing dynamic email messages" on page 1791.
- To send an email that is constructed from a MIME message, see "Sending a MIME message" on page 1795.
- To configure the SMTP server, port number, and security identity for the EmailOutput node as a broker external resource, see "Changing connection information for the EmailOutput node" on page 1798.

Related concepts:

"Local environment tree structure" on page 1056
The local environment tree is a part of the logical message tree in which you can store information while the message flow processes the message.

"Configurable services" on page 1296
Configurable services are typically runtime properties. You can use them to define properties that are related to external services on which the broker relies; for example, an SMTP server or a JMS provider.

Related tasks:

"Resolving problems when you use Email nodes" on page 3398
Advice for dealing with common problems that can arise when you develop message flows that contain Email nodes.

Related reference:

"EmailOutput node" on page 4400
Use the EmailOutput node to send email messages to one or more recipients.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a

broker external resource.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

Sending an email:

You can send an email with a static subject and static text to a static list of recipients.

About this task

Use the WebSphere Message Broker Toolkit to configure the properties on the EmailOutput node so that you can send an email with a statically defined subject and text, and no attachments, to a statically defined list of recipients. The same email is sent to the same recipients. This method is useful when you want to test the EmailOutput node, or when notification alone is sufficient.

Procedure

1. Switch to the Broker Application Development perspective.
2. Add an EmailOutput node to your message flow.
3. Edit the following properties of the EmailOutput node.
   a. On the **Basic** tab, add the SMTP Server and Port information (for example, *smtp.server.com:25*).
   b. On the **Email** tab, add the email addresses of recipients by using the `To Addresses`, `Cc Addresses`, and `Bcc Addresses` properties.
   c. On the **Email** tab, add the email address of the sender by using the `From Address` and `Reply-To Address` properties.
   d. On the **Email** tab, provide a subject for the email by using the `Subject of email` property.
   e. On the **Email** tab, provide the text of the email by using `Email message text` property.
4. Save the changes.
5. Add the message flow to the BAR file and deploy.

Results

When a message is passed into the deployed EmailOutput node, an email is sent to the defined set of recipients, containing the subject and text specified on the node.

Related tasks:

"Sending an email with an attachment" on page 1790
You can send an email with a fixed subject and fixed text, and an attachment, to a static list of recipients.

"Producing dynamic email messages" on page 1791
You can produce an email where the SMTP server, list of recipients, subject, text, and multiple attachments are all determined at run time.

"Sending a MIME message" on page 1795
You can send an email that is constructed from a MIME message.

"Changing connection information for the EmailOutput node" on page 1798
You can configure the SMTP server, port number, and security identity for the EmailOutput node as a broker external resource.

Sending an email with an attachment:

You can send an email with a fixed subject and fixed text, and an attachment, to a static list of recipients.

About this task

You can configure an EmailOutput node to send an email with a single attachment. To send an email with multiple attachments, see "Producing dynamic email messages" on page 1791.

Use the WebSphere Message Broker Toolkit to configure the properties on the EmailOutput node so that you can send an email with a statically defined subject and text, and an attachment, to a statically defined list of recipients. This method causes the email message to be constructed as a MIME message. The subject, text, and list of recipients remains static, but the content of the attachment is sought dynamically from the message that is passed to the EmailOutput node at run time. The location of the attachment in the message is defined statically.

Procedure

1. Switch to the Broker Application Development perspective.
2. Add an EmailOutput node to your message flow.
3. Edit the following properties of the EmailOutput node.
   a. On the **Basic** tab, add the SMTP Server and Port information (for example, *smtp.server.com:25*).
   b. On the **Email** tab, add the email addresses of recipients by using the `To Addresses`, `Cc Addresses`, and `Bcc Addresses` properties.
   c. On the **Email** tab, add the email address of the sender by using the `From Address` and `Reply-To Address` properties.
   d. On the **Email** tab, provide a subject for the email by using the `Subject of email` property.
   e. On the **Email** tab, provide the text of the email by using `Email message text` property.
   f. On the **Attachment** tab, set the `Attachment Content` property by using either an ESQL or XPath expression, referring to an element in the message tree; for example, *Body.BLOB*.
   g. On the **Attachment** tab, set the `Attachment Content Name` property with the name of the attachment as it appears in the email.
   h. On the **Attachment** tab, set the `Attachment Content Type`, `Attachment Content Encoding`, and `Multipart Content Type` properties to determine the type of attachment that is sent in the MIME message.
4. Save the changes.
5. Add the message flow to the BAR file and deploy.

Results

When a message is passed into the deployed EmailOutput node, an email is sent to the defined set of recipients, containing the subject, text, and attachment specified on the node.

Related tasks:

"Sending an email" on page 1789
You can send an email with a static subject and static text to a static list of recipients.

"Producing dynamic email messages"
You can produce an email where the SMTP server, list of recipients, subject, text, and multiple attachments are all determined at run time.

"Sending a MIME message" on page 1795
You can send an email that is constructed from a MIME message.

"Changing connection information for the EmailOutput node" on page 1798
You can configure the SMTP server, port number, and security identity for the EmailOutput node as a broker external resource.

Producing dynamic email messages:

You can produce an email where the SMTP server, list of recipients, subject, text, and multiple attachments are all determined at run time.

About this task

You can create a message flow that produces an email with multiple attachments. To configure an EmailOutput node to send an email with a single attachment, static subject, and static text to a static list of recipients, see "Sending an email with an attachment" on page 1790.

The node properties that you set when sending an email can be optional, and can be overridden at run time by values that you specify in the local environment, email output header (EmailOutputHeader), or the body of the message. To use this method, previous nodes in the message flow must construct these overrides. Where a text value is not specified in the node properties for the main body of the email, the body of the message that is passed to the EmailOutput node is used.

The following examples show how to set up the recipient, sender, subject, SMTP server, and message body information in ESQL (with a Compute node) and Java (with a JavaCompute node).

Using a Compute node:
About this task


```
CREATE FUNCTION Main() RETURNS BOOLEAN
 BEGIN
  CALL CopyMessageHeaders();

  -- Add recipient information to the EmailOutputHeader
```

```
  SET OutputRoot.EmailOutputHeader.To = '<recipient email address>';
  SET OutputRoot.EmailOutputHeader.Cc = '<recipient email address>';
  SET OutputRoot.EmailOutputHeader.Bcc = '<recipient email address>';

  -- Add sender information to EmailOutputHeader
  SET OutputRoot.EmailOutputHeader.From = '<sender email address>';
  SET OutputRoot.EmailOutputHeader."Reply-To" = '<reply email address>';

  -- Add subject to EmailOutputHeader
  SET OutputRoot.EmailOutputHeader.Subject = 'Replaced by ESQL compute node.';

  -- Add SMTP server information to the LocalEnvironment
  SET OutputLocalEnvironment.Destination.Email.SMTPServer ='<smtp.server:port>';

  -- Create a new message body, which will be sent as the main text of the email.
  SET OutputRoot.BLOB.BLOB = CAST('This is the new text for the body of the email.' AS BLOB CCSID 1208);

  RETURN TRUE;
 END;
```

To write white space characters (such as newline (NL), carriage return (CR), and line feed (LF) characters) in the text string that is produced as the email body, you can add the following lines of code.

```
DECLARE crlf CHAR CAST(X'0D0A' AS CHAR CCSID 1208);
DECLARE myEmailBodyTxt CHAR;
SET myEmailBodyTxt [equals char] 'this is the first line' || crlf ||
    'this is the second line' || crlf ||
    'this is the third line';
SET OutputRoot.BLOB.BLOB = CAST(myEmailBodyTxt AS BLOB CCSID 1208);
```

Using a JavaCompute node:

About this task


```
public void evaluate(MbMessageAssembly assembly) throws MbException {
  MbOutputTerminal out = getOutputTerminal("out");

  // Create a new assembly to propagate out of this node, as we want to update it
  MbMessage outMessage = new MbMessage();
  copyMessageHeaders(assembly.getMessage(), outMessage);
  MbMessage outLocalEnv = new MbMessage(assembly.getLocalEnvironment());
  MbMessage outExceptionList = new MbMessage(assembly.getExceptionList());
  MbMessageAssembly outAssembly = new MbMessageAssembly(assembly, outLocalEnv, outExceptionList, outMessage);
  MbElement localEnv = outAssembly.getLocalEnvironment().getRootElement();

  // Create the EmailOutputHeader parser. This is where we add recipient, sender and subject information.
  MbElement root = outMessage.getRootElement();
  MbElement SMTPOutput = root.createElementAsLastChild("EmailOutputHeader");

  // Add recipient information to EmailOutputHeader
  SMTPOutput.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "To", "<recipient email address");
  SMTPOutput.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "Cc", "<recipient email address");
  SMTPOutput.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "Bcc", "<recipient email address");

  // Add sender information to EmailOutputHeader
  SMTPOutput.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "From", "<sender email address>");
  SMTPOutput.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "Reply-To", "<reply email address>");
```

```
// Add subject information to EmailOutputHeader
SMTPOutput.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "Subject", "Replaced by Java compute node.");

// Create Destination.Email. This is where we add SMTP server information
MbElement Destination = localEnv.createElementAsLastChild(MbElement.TYPE_NAME, "Destination", null);
MbElement destinationEmail = Destination.createElementAsLastChild(MbElement.TYPE_NAME, "Email", null);
destinationEmail.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "SMTPServer", "<smtp.server:port>");

// Set last child of root (message body)
MbElement BLOB = root.createElementAsLastChild(MbBLOB.PARSER_NAME);
String text = "This is the new text for the body of the email";
BLOB.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "BLOB", text.getBytes());

outMessage.finalizeMessage(MbMessage.FINALIZE_VALIDATE);

out.propagate(outAssembly);  }
```

Using the local environment:
About this task

Use the local environment to specify overrides to the SMTP server connection information and attachments.

Local environment	Description
Destination.Email.SMTPServer	The Server:Port of the SMTP server. Port is optional; if you do not specify it, the default value is 25.
Destination.Email.SecurityIdentity	The security identity for authentication with the SMTP server, which can be the name of the userid and password pair that is defined using the **mqsisetdbparms** command, or it can reference an external resource that has a securityIdentity attribute that references a userid and password that are defined using the **mqsisetdbparms** command. In both cases, the value is appended after the string "smtp::". For example, if you use the **mqsisetdbparms** command to create a userid and password of *smtp::myUseridPassword*, the securityIdentity that is specified on the node, or indirectly in an external resource, is *myUseridPassword*.
Destination.Email.BodyContentType	Identifies that the body of the email message contains HTML rather than plain text. You can set this property to text/plain, text/html, or text/xml; text/plain is the default value.
Destination.Email.MultiPartContentType	The type of multipart, including related, mixed, and alternative. You can set any value here.
Destination.Email.Attachment.Content	Either the actual attachment (BLOB/text), or an XPath or ESQL expression that references an element; for example, an element in the message tree or LocalEnvironment. The value of the referenced element is taken as the content of the attachment. • If the element is a BLOB, it is an attachment. • If the element is text, check to see if it can be resolved to another element in the message tree or LocalEnvironment. If it can be resolved, use that element. If it cannot be resolved, add this element as the attachment.
Destination.Email.Attachment.ContentType	The type of attachment (also known as Internet Media Type), including text/plain, text/html, and text/xml. You can set any value here.
Destination.Email.Attachment.ContentName	The name of the attachment.

Local environment	Description
Destination.Email.Attachment.ContentEncoding	The encoding of the attachment: 7bit, base64, or quoted-printable. • 7bit is the default value that is used for ASCII text. • Base64 is used for non ASCII, whether non English or binary data. This format can be difficult to read. • Quoted-printable is an alternative to Base64, and is appropriate when most of the data is ASCII with some non-ASCII parts. This format is more readable; it provides a more compact encoding because the ASCII parts are not encoded.

Using the email output header:
About this task

Use the email output header to specify overrides to the SMTP server connection information and attachments. The EmailOutputHeader is a child of Root. Values that you specify in this header override equivalent properties that you set on the EmailOutput node. Use the SMTP output header to specify any of the email attributes, such as its recipients.

Location	Description
Root.EmailOutputHeader.To	A comma-separated list of email addresses.
Root.EmailOutputHeader.Cc	A comma-separated list of email addresses.
Root.EmailOutputHeader.Bcc	A comma-separated list of email addresses.
Root.EmailOutputHeader.From	A comma-separated list of email addresses.
Root.EmailOutputHeader.Reply-To	A comma-separated list of email addresses.
Root.EmailOutputHeader.Subject	The subject of the email.

Related concepts:

"Local environment tree structure" on page 1056
The local environment tree is a part of the logical message tree in which you can store information while the message flow processes the message.

Related tasks:

"Sending an email" on page 1789
You can send an email with a static subject and static text to a static list of recipients.

"Sending an email with an attachment" on page 1790
You can send an email with a fixed subject and fixed text, and an attachment, to a static list of recipients.

"Sending a MIME message" on page 1795
You can send an email that is constructed from a MIME message.

"Changing connection information for the EmailOutput node" on page 1798
You can configure the SMTP server, port number, and security identity for the EmailOutput node as a broker external resource.

Related reference:

"EmailOutput node" on page 4400
Use the EmailOutput node to send email messages to one or more recipients.

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"JavaCompute node" on page 4514
Use the JavaCompute node to work with messages by using the Java language.

Sending a MIME message:

You can send an email that is constructed from a MIME message.

About this task

You can pass a MIME message to the EmailOutput node, which uses the MIME parser to write the MIME message to a bit stream. This message is then sent to the list of recipients in the EmailOutputHeader. Local environment overrides are not considered when a MIME message is passed. You do not need to configure the EmailOutput node in the following examples.

The following examples show how to set up the recipient, sender, subject, SMTP server, and message body information in ESQL (with a Compute node) and Java (with a JavaCompute node).

Using a Compute node:
About this task


```
CREATE FUNCTION Main() RETURNS BOOLEAN
BEGIN

   CALL CopyMessageHeaders();

   -- Add recipient information to the EmailOutputHeader
   SET OutputRoot.EmailOutputHeader.To = '<recipient email address>';
   SET OutputRoot.EmailOutputHeader.Cc = '<recipient email address>';
   SET OutputRoot.EmailOutputHeader.Bcc = '<recipient email address>';

   -- Add sender information to EmailOutputHeader
   SET OutputRoot.EmailOutputHeader.From = '<sender email address>';
   SET OutputRoot.EmailOutputHeader."Reply-To" = '<reply email address>';

   -- Add subject to EmailOutputHeader
   SET OutputRoot.EmailOutputHeader.Subject = 'Dynamic MIME message in ESQL.';

   -- Add SMTP server information to the LocalEnvironment
   SET OutputLocalEnvironment.Destination.Email.SMTPServer ='<smtp.server:port>';

   -- Create a new MIME message body, which will be sent as the main text of the email,
   -- including an attachment.
   CREATE FIELD OutputRoot.MIME TYPE Name;
   DECLARE M REFERENCE TO OutputRoot.MIME;

   -- Create the Content-Type child of MIME explicitly to ensure the correct order. If we set
   -- the ContentType property instead, the field could appear as the last child of MIME.
   CREATE FIELD M."Content-Type" TYPE NameValue VALUE 'multipart/related; boundary=myBoundary';
   CREATE FIELD M."Content-ID"   TYPE NameValue VALUE 'new MIME document';

   CREATE LASTCHILD OF M TYPE Name NAME 'Parts';
   CREATE LASTCHILD OF M.Parts TYPE Name NAME 'Part';
   DECLARE P1 REFERENCE TO M.Parts.Part;
```

```
   -- First part:
   --   Create the body of the email.
   --   The body of the email has the text 'This is the main body of the email.'.
   CREATE FIELD P1."Content-Type" TYPE NameValue VALUE 'text/plain; charset=us-ascii';
   CREATE FIELD P1."Content-Transfer-Encoding" TYPE NameValue VALUE '8bit';
   CREATE LASTCHILD OF P1 TYPE Name NAME 'Data';
   CREATE LASTCHILD OF P1.Data DOMAIN('BLOB') PARSE(CAST('This is the main body of the email.'
AS BLOB CCSID 1208));

   CREATE LASTCHILD OF M.Parts TYPE Name NAME 'Part';
   DECLARE P2 REFERENCE TO M.Parts.Part[2];

   -- Second part:
   --   Create the attachment of an email.
   --   The attachment is called 'attachment.txt' and contains the text 'This is an attachment.'.
   CREATE FIELD P2."Content-Type" TYPE NameValue VALUE 'text/plain; charset=us-ascii; name=attachment.txt';
   CREATE FIELD P2."Content-Transfer-Encoding" TYPE NameValue VALUE '8bit';
   CREATE LASTCHILD OF P2 TYPE Name NAME 'Data';
   CREATE LASTCHILD OF P2.Data DOMAIN('BLOB') PARSE(CAST('This is an attachment.' AS BLOB CCSID 1208));

   RETURN TRUE;
END;
```

Using a JavaCompute node:

About this task


```
public void evaluate(MbMessageAssembly assembly) throws MbException {
   MbOutputTerminal out = getOutputTerminal("out");
   MbOutputTerminal fail = getOutputTerminal("fail");

   // Create a new assembly to propagate out of this node, as we want to
   // update it
   MbMessage outMessage = new MbMessage();
   copyMessageHeaders(assembly.getMessage(), outMessage);
   MbMessage outLocalEnv = new MbMessage(assembly.getLocalEnvironment());
   MbMessage outExceptionList = new MbMessage(assembly.getExceptionList());
   MbMessageAssembly outAssembly = new MbMessageAssembly(assembly, outLocalEnv, outExceptionList, outMessage);
   MbElement localEnv = outAssembly.getLocalEnvironment().getRootElement();

   // Create the EmailOutputHeader parser. This is where we add recipient,
   // sender and subject information.
   MbElement root = outMessage.getRootElement();
   MbElement SMTPOutput = root.createElementAsLastChild("EmailOutputHeader");

   // Add recipient information to EmailOutputHeader
   SMTPOutput.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "To", "<recipient email address>");
   SMTPOutput.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "Cc", "<recipient email address>");
   SMTPOutput.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "Bcc", "<recipient email address>");

   // Add sender information to EmailOutputHeader
   SMTPOutput.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "From", "<sender email address>");
   SMTPOutput.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "Reply-To", "<reply email address>");

   // Add subject information to EmailOutputHeader
   SMTPOutput.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "Subject", "Dynamic MIME message in Java.");

   // Create Destination.Email. This is where we add SMTP server information.
```

```
   MbElement Destination = localEnv.createElementAsLastChild(MbElement.TYPE_NAME, "Destination", null);
   MbElement destinationEmail = Destination.createElementAsLastChild(MbElement.TYPE_NAME, "Email", null);
   destinationEmail.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "SMTPServer", "<smtp.server:port>");

   // Set last child of root (message body) as MIME.
   MbElement MIME = root.createElementAsLastChild("MIME");

   // Create the Content-Type child of MIME explicitly to ensure the correct order.
   MIME.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "Content-Type", "multipart/related;
boundary=myBoundary");
   MIME.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "Content-ID", "new MIME document");
   MbElement parts = MIME.createElementAsLastChild(MbElement.TYPE_NAME, "Parts", null);
   MbElement part, data, blob;
   String text;

   // First part:
   //   Create the body of the email.
   //   The body of the email has the text 'This is the main body of the email.'.
   part = parts.createElementAsLastChild(MbElement.TYPE_NAME, "Part", null);
   part.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "Content-Type", "text/plain; charset=us-ascii");
   part.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "Content-Transfer-Encoding", "8bit");
   data = part.createElementAsLastChild(MbElement.TYPE_NAME, "Data", null);
   blob = data.createElementAsLastChild("BLOB");
   text = "This is the main body of the email.";
   try {
  blob.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "BLOB", text.getBytes("UTF8"));
   } catch (UnsupportedEncodingException e) {
  fail.propagate(outAssembly);
   }

   // Second part:
   //   Create the attachment of an email.
   //   The attachment is called 'attachment.txt' and contains the text 'This is an attachment.'.
   part = parts.createElementAsLastChild(MbElement.TYPE_NAME, "Part", null);
   part.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "Content-Type", "text/plain; charset=us-ascii;
name=attachment.txt");
   part.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "Content-Transfer-Encoding", "8bit");
   data = part.createElementAsLastChild(MbElement.TYPE_NAME, "Data", null);
   blob = data.createElementAsLastChild("BLOB");
   text = "This is an attachment.";
   try {
  blob.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "BLOB", text.getBytes("UTF8"));
   } catch (UnsupportedEncodingException e) {
  fail.propagate(outAssembly);
   }

   outMessage.finalizeMessage(MbMessage.FINALIZE_VALIDATE);
   out.propagate(outAssembly);
}
```

Related tasks:

"Sending an email" on page 1789
You can send an email with a static subject and static text to a static list of recipients.

"Sending an email with an attachment" on page 1790
You can send an email with a fixed subject and fixed text, and an attachment, to a static list of recipients.

"Producing dynamic email messages" on page 1791
You can produce an email where the SMTP server, list of recipients, subject, text, and multiple attachments are all determined at run time.

"Changing connection information for the EmailOutput node" on page 1798
You can configure the SMTP server, port number, and security identity for the EmailOutput node as a broker external resource.

Changing connection information for the EmailOutput node:

You can configure the SMTP server, port number, and security identity for the EmailOutput node as a broker external resource.

About this task

Use an alias that is specified in the `SMTP Server and Port` property on the EmailOutput node. The security identity references a user ID and password pair that is defined on the broker by using the **mqsisetdbparms** command.

Procedure

1. Use the WebSphere Message Broker Explorer; see "Using the WebSphere Message Broker Explorer to work with configurable services" on page 644 for more information, or the **mqsicreateconfigurableservice** command to create an SMTP broker external resource for the alias that is specified on the node.
2. Then use the **mqsichangeproperties** command to create an SMTPServer property with the value in the form of *server:port*. The port value is optional; if you do not specify it, the default value is 25. You can also use the **mqsichangeproperties** command to create an SMTPSecurityIdentity property with a value that is the name of a security identity that can be resolved at run time to a user ID and password for authentication with the SMTP server.

   For example:

```
mqsicreateconfigurableservice MY_BROKER –c SMTP –o SMTP_MyAlias
```

   followed by:

```
 mqsichangeproperties MY_BROKER –c SMTP –o SMTP_MyAlias –n serverName –v smtp.hursley.ibm.com:25
```

   These commands override the SMTP server and port values that are specified on any nodes that also specify an alias of SMTP_MyAlias. If the local environment contains any overrides, they take preference over the broker external resource properties. See the following example:

```
mqsichangeproperties MY_BROKER –c SMTP –o SMTP_MyAlias –n securityIdentity –v mySecurityIdentity
```

   You must also use the **mqsisetdbparms** command to define the security identity at the broker run time.

Related concepts:

"Using the WebSphere Message Broker Explorer to work with configurable services" on page 644
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view, add, modify and delete configurable services.

Related tasks:

"Sending an email" on page 1789
You can send an email with a static subject and static text to a static list of recipients.

"Sending an email with an attachment" on page 1790
You can send an email with a fixed subject and fixed text, and an attachment, to a static list of recipients.

"Producing dynamic email messages" on page 1791
You can produce an email where the SMTP server, list of recipients, subject, text, and multiple attachments are all determined at run time.

"Sending a MIME message" on page 1795
You can send an email that is constructed from a MIME message.

Related reference:

"EmailOutput node" on page 4400
Use the EmailOutput node to send email messages to one or more recipients.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

Receiving emails

You can configure the EmailInput node to receive an email, with or without an attachment, from an email server that supports Post Office Protocol 3 (POP3) or Internet Message Access Protocol (IMAP).

About this task

You can use the `Email server` property on the EmailInput node to specify the protocol, host name, and port of the email server that the broker uses to receive emails. Alternatively, you can specify a configurable service name that refers to an EmailServer configurable service that is defined on the broker. To enable the configurable service, use the **mqsicreateconfigurableservice** and **mqsichangeproperties** commands, as shown in the following examples:

```
mqsicreateconfigurableservice MB7BROKER -c EmailServer -o
myEmailConfigurableServiceName
```

```
mqsichangeproperties MB7BROKER -c EmailServer -o myEmailConfigurableServiceName
  -n serverName -v pop3://myemailserver.com:12345
```

If the value of the `Email server` EmailInput node property is set to the defined configurable service name, any values set by the administrator on the command line are used in preference to any statically defined value.

The following list details the order of preference for value selection:

1. The email server URL value that is specified in the EmailServer configurable service `serverName` property, if a configurable service exists that matches the name that is supplied in the `Email server` EmailInput node property.
2. The email server URL value of the `Email server` property that is specified directly on the EmailInput node.

A configurable service can be removed by using the **mqsideleteconfigurableservice** command, so that the EmailInput node reverts to resolving the email server URL from the value that is set on the node. For more information about creating, changing, reporting, and deleting configurable services, see "Changing connection information for the EmailInput node" on page 1805.

While most configurable service properties are set by using the **mqsichangeproperties** command, security identity support, such as an email server user ID and password pair, is typically set by using the **mqsisetdbparms** command. Security identity support can be configured by setting the `Security identity` EmailInput node property or EmailServer configurable service `securityIdentity`

property to reference a security identity object. However, the security identity object must first be created by using the **mqsisetdbparms** command, for example:

```
mqsisetdbparms MB7BROKER -n email::mySecurityIdentity -u myUserID -p myPassword
mqsichangeproperties MB7BROKER -c EmailServer -o myEmailConfigurableServiceName -n
securityIdentity -v mySecurityIdentity
```

For more information about email server security identity support, see "**mqsisetdbparms** command" on page 3954.

The topics in this section describe the different ways in which you can use the EmailInput node to receive email messages.

Procedure

- To receive an email, see "Receiving an email" on page 1801.
- To configure the email server URL and security identity for the EmailInput node as a broker external resource, see "Changing connection information for the EmailInput node" on page 1805.

Related tasks:

"Changing connection information for the EmailInput node" on page 1805
You can create a configurable service that the EmailInput node or message flow refers to at run time for email server connection information, instead of defining the connection properties on the node or the message flow. The advantage being that you can change the host name and security identity values without needing to redeploy your message flow.

"Resolving problems when you use Email nodes" on page 3398
Advice for dealing with common problems that can arise when you develop message flows that contain Email nodes.

Related reference:

"EmailInput node" on page 4394
Use the EmailInput node to retrieve an email, with or without attachments, from an email server that supports Post Office Protocol 3 (POP3) or Internet Message Access Protocol (IMAP).

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

Receiving an email:

You can receive an email, with or without attachments, from an email server that supports Post Office Protocol 3 (POP3) or Internet Message Access Protocol (IMAP).

Before you begin

Before you start:

This topic assumes that you have already created a message flow. For more information, see "Creating a message flow" on page 1431.

About this task

Use the WebSphere Message Broker Toolkit to configure the properties on the EmailInput node so that you can receive an email, with or without attachments.

Procedure

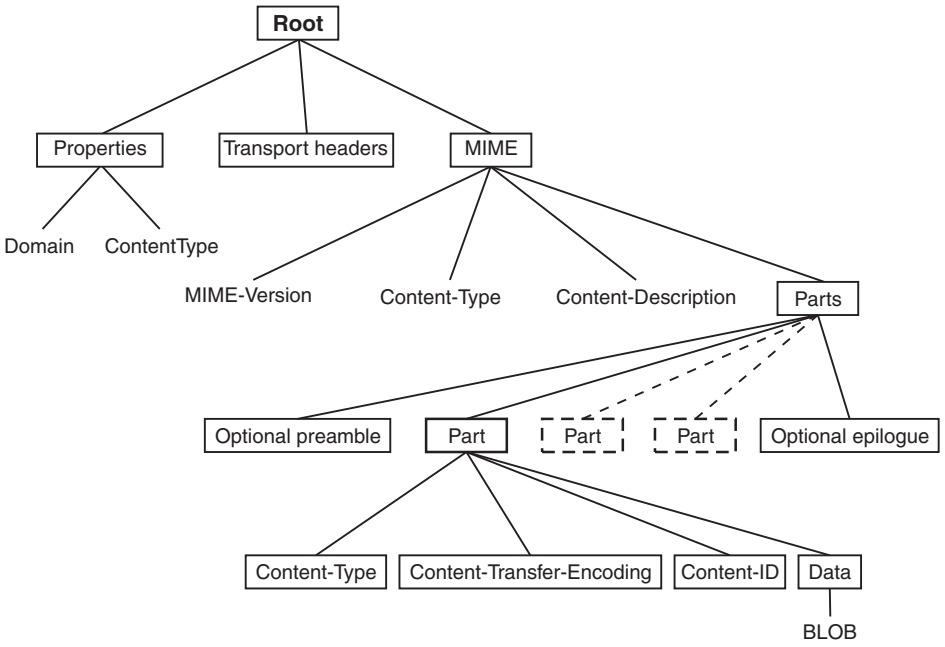
1. Add an EmailInput node to your message flow.
2. Edit the following EmailInput node properties:
   a. On the **Basic** tab, add the email server URL, or the EmailServer configurable service name as the `Email server` property value, as described in "EmailInput node" on page 4394. For example, `pop3://myemailserver.com:12345` or `imap://myemailserver.com:56789`.
   b. On the **Security** tab, add the security identity object name of the email server user ID and password pair as the `Security identity` property value. For more information about email server security identity support, see "**mqsisetdbparms** command" on page 3954.
   c. Configure the following properties on the **Retry** tab:
      - `Retry mechanism`: The Retry mechanism property defines how the EmailInput node handles a message flow failure. Valid values are `Failure`, `Short Retry`, or `Short and Long Retry`. The default value for this property is `Short and long retry`, which indicates that the email is retried until the short retry threshold is met, then long retry takes place, meaning that the email is never deleted from the email server, but also that the email is infinitely retried. Emails are deleted from the email server if the email message fails and this property value is not set to `Short and Long Retry`.
      - `Retry threshold`: The Retry threshold property is the number of times to try the message flow transaction again when the `Retry mechanism` property value is set to `Short Retry`. The default value for this property is `0`.
      - `Short retry interval (in seconds)`: The Short retry interval is the interval, in seconds, between each retry if the `Retry threshold` property value is not set to zero. The default value for this property is `0`. If the email is retried until the short retry threshold is met and the email fails, the email is routed to the failure terminal, and the email is deleted from the email server.
      - `Long retry interval (in seconds)`: The Long retry interval is the interval, in seconds, between each retry, if the `Retry mechanism` property value is `Short and Long Retry`, and the short retry threshold has been exhausted. The default value for this property is `300` seconds.

- `Action on failing email`: The Action on failing email property determines the action that the EmailInput node takes with the input data source after all attempts to process the email contents fail. The Action on failing email property is a read only property that is set to a default value of `Delete Email`, which is used with the Retry mechanism property. If the Retry mechanism property is set to `Short and Long Retry`, the message flow continues to try to retrieve the email from the email server, meaning that the email is never deleted. If the Retry mechanism property is not set to `Short and Long Retry`, the Action on failing email property value `Delete Email` is used, and the email is deleted from the email server.

3. Save the changes.
4. Add the message flow to the BAR file and deploy.

   When a message is passed into the deployed EmailInput node, an email is received from the email server and the body of the email message, and any attachments, are propagated in the Multipurpose Internet Mail Extensions (MIME) domain. All other information relating to the email is stored in the `Root.EmailInputHeader` MIME logical tree. For a complete list of the email elements that are propagated in the MIME logical tree when you use an EmailInput node, see "EmailInput node" on page 4394.

   When an email containing an attachment is received, the EmailInput node places different parts of the email body in the MIME domain, so that they are associated with the MIME parser. The MIME tree location that the EmailInput node builds to house the information is the same location that the EmailOutput node expects email data to be in when sending an email. The attachment is stored in the MIME logical tree in directory `Root.MIME.Parts.Part.Data`. Where Content-Type describes the type of data that is in the attachment.



   Viewing the `Root.Properties.ContentType` value in the MIME domain allows you to write logic to parse the attachment. For more information about the MIME logical tree, see "MIME parser and domain" on page 1117.

Results

Received emails are deleted from an email server that supports POP3 or IMAP only when the emails have been successfully propagated after being processed by the EmailInput node Failure, Out, or Catch terminals, and the message flow has successfully executed. This does not form part of a globally coordinated transaction.

Emails are deleted from the email server under the following circumstances:

- The Failure terminal is not connected.
- An exception occurs in the Failure terminal.
- The email message fails and the `Retry mechanism` property value is not set to `Short and Long Retry`.
- The `Retry threshold` is not set to `0` and the `Short retry interval` property value has been exhausted.

For more information about processing responses from an EmailInput node, and for information about rollback handling, see "Processing responses from an EmailInput node" on page 1804.

Related concepts:

"MIME parser and domain" on page 1117
Use the MIME domain if your messages use the MIME standard for multipart messages.

"Manipulating messages in the MIME domain" on page 2612
A MIME message does not need to be received over a particular transport. For example, a message can be received over HTTP by using an HTTPInput node, or over WebSphere MQ by using an MQInput node. The MIME parser is used to process a message if the message domain is set to MIME in the input node properties, or if you are using WebSphere MQ, and the MQRFH2 header has a message domain of MIME.

Related tasks:

"Processing responses from an EmailInput node" on page 1804
The EmailInput node can return different response messages that indicate the success or failure of receiving an email, with or without attachments, from an email server that supports Post Office Protocol 3 (POP3) or Internet Message Access Protocol (IMAP).

"Changing connection information for the EmailInput node" on page 1805
You can create a configurable service that the EmailInput node or message flow refers to at run time for email server connection information, instead of defining the connection properties on the node or the message flow. The advantage being that you can change the host name and security identity values without needing to redeploy your message flow.

"Resolving problems when you use Email nodes" on page 3398
Advice for dealing with common problems that can arise when you develop message flows that contain Email nodes.

Related reference:

"EmailInput node" on page 4394
Use the EmailInput node to retrieve an email, with or without attachments, from an email server that supports Post Office Protocol 3 (POP3) or Internet Message Access Protocol (IMAP).

Processing responses from an EmailInput node:

The EmailInput node can return different response messages that indicate the success or failure of receiving an email, with or without attachments, from an email server that supports Post Office Protocol 3 (POP3) or Internet Message Access Protocol (IMAP).

Before you begin

Before you start:

Ensure that you have developed a message flow with an EmailInput node, as described in "Receiving an email" on page 1801.

About this task

The EmailInput node has three output terminals:

- Failure: The output terminal to which the message is routed if an EmailInput node failure is detected when a message is propagated, or an EmailInput node fails to access the email server. Connect the Failure terminal of this node to another node in the message flow to process errors.
- Out: The output terminal to which the message is routed if it has been propagated successfully. Connect the Out terminal of this node to another node in the message flow to process the message further, or send the message to an additional destination.
- Catch: The output terminal to which a message is routed if an exception is thrown downstream and caught by this node. Exceptions are caught only if this terminal is attached.

Procedure

- **Processing successful returns**

  When an EmailInput node successfully receives an email, the resulting message is propagated to the Out terminal.
- **Processing downstream message flow exceptions**

  If a message flow exception occurs downstream of the Failure terminal in the message flow, a message is routed to the Catch terminal. If you do not have the `Retry mechanism` property value set to `Short and Long Retry` on the EmailInput node Retry tab, the current transaction is rolled back.
- **Handling failures in the node**

  Any other failures are propagated to the Failure terminal. Possible failures include:
  - A problem retrieving an email message. For example, an inability to communicate with the target email server because of a connection or authentication error. If this occurs, the connection is re attempted when the polling interval expires. Each connection failure occurrence causes an appropriate message in user trace.
  - A problem parsing an email.
  - An internal EmailInput node exception or error is detected before the message is propagated to the Out terminal. For example, a malformed email is received, causing the EmailInput to propagate the message and an exception list to the Failure terminal.
  - The `Retry mechanism` property value is set to `Failure`, causing an immediate message to be sent to the Failure terminal.

– The `Retry threshold` is not set to `0` and the `Short retry interval` property value has been exhausted, causing a message to be routed to the failure terminal, and the email being deleted from the email server.

If the failure terminal is not connected, or an exception occurs down the Failure terminal, or the email message fails and you have not set the `Retry mechanism` property value to `Short and Long Retry`, the email is deleted from the email server.

Results

You can use the EmailServer configurable service to change connection details for the EmailInput node. See "Changing connection information for the EmailInput node" for details about creating, changing, reporting, and deleting an EmailServer configurable service.

Related tasks:

"Receiving emails" on page 1799
You can configure the EmailInput node to receive an email, with or without an attachment, from an email server that supports Post Office Protocol 3 (POP3) or Internet Message Access Protocol (IMAP).

"Receiving an email" on page 1801
You can receive an email, with or without attachments, from an email server that supports Post Office Protocol 3 (POP3) or Internet Message Access Protocol (IMAP).

"Changing connection information for the CICSRequest node" on page 738
You can create a configurable service that the CICSRequest node or message flow refers to at run time for connection information, instead of defining the connection properties on the node or the message flow. The advantage being that you can change the host name and performance values without needing to redeploy your message flow.

"Resolving problems when you use Email nodes" on page 3398
Advice for dealing with common problems that can arise when you develop message flows that contain Email nodes.

Related reference:

"EmailInput node" on page 4394
Use the EmailInput node to retrieve an email, with or without attachments, from an email server that supports Post Office Protocol 3 (POP3) or Internet Message Access Protocol (IMAP).

Changing connection information for the EmailInput node:

You can create a configurable service that the EmailInput node or message flow refers to at run time for email server connection information, instead of defining the connection properties on the node or the message flow. The advantage being that you can change the host name and security identity values without needing to redeploy your message flow.

Before you begin

Before you start:

- Read "Configurable services" on page 1296 to find out more about configurable services.
- Read "Receiving emails" on page 1799 for background information.

About this task

Use the EmailServer configurable service to change the email server connection information for the EmailInput node. The properties of the EmailServer configurable service are described in "Configurable services properties" on page 3766.

Creating, changing, reporting, and deleting configurable services

Procedure

1. To create a configurable service, use the WebSphere Message Broker Explorer; see "Using the WebSphere Message Broker Explorer to work with configurable services" on page 644 for more information, or the **mqsicreateconfigurableservice** command, as shown in the following example. This example creates an EmailServer configurable service for the email server that is running at `pop3://test.email.server.ibm.com` port 12345. The security identity is identified by *mySecurityIdentityObjectName* in this example:

   ```
   mqsicreateconfigurableservice MB7BROKER -c EmailServer -o
   myEmailConfigurableServiceName -n serverName,securityIdentity -v
   pop3://test.email.server.ibm.com:12345,mySecurityIdentityObjectName
   ```

2. To change a configurable service, use the WebSphere Message Broker Explorer, or the **mqsichangeproperties** command, as shown in the following example. You must stop and start the execution group for the change of property value to take effect. This example changes the EmailInput node that is configured to use the *myEmailConfigurableServiceName* configurable service. After you run this command, the EmailInput node connects to the production system (`pop3://production.email.server.ibm.com:12345`) instead of the test system (`pop3://test.email.server.ibm.com:12345`).

   ```
   mqsichangeproperties MB7BROKER -c EmailServer -o myEmailConfigurableServiceName
     -n serverName -v pop3://production.email.server.ibm.com:12345
   ```

3. To display all EmailServer configurable services, use the WebSphere Message Broker Explorer, or the **mqsireportproperties** command, as shown in the following example:

   ```
   mqsireportproperties MB7BROKER -c EmailServer -o AllReportableEntityNames -r
   ```

4. You can delete a configurable service that you have created by using the WebSphere Message Broker Explorer, or the **mqsideleteconfigurableservice** command, as shown in the following example:

   ```
   mqsideleteconfigurableservice MB7BROKER -c EmailServer -o
   myEmailConfigurableServiceName
   ```

   You can delete only custom-named configurable services; you cannot delete IBM-defined configurable services.

Related concepts:

"Using the WebSphere Message Broker Explorer to work with configurable services" on page 644
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view, add, modify and delete configurable services.

Related tasks:

"Receiving an email" on page 1801
You can receive an email, with or without attachments, from an email server that supports Post Office Protocol 3 (POP3) or Internet Message Access Protocol (IMAP).

Related reference:

"EmailInput node" on page 4394
Use the EmailInput node to retrieve an email, with or without attachments, from an email server that supports Post Office Protocol 3 (POP3) or Internet Message Access Protocol (IMAP).

Working with files

You can use the FileInput, FileRead, and FTEInput nodes in your message flows to process data from files. You can use the FileOutput node and FTEOutput node to send data from a message flow into a file.

About this task

Using files is one of the most common methods of storing data. You can create message flows to process data in files, accepting data in files as input message data, and producing output message data for file-based destinations. The following file nodes are provided:

- FileInput node. Use this node to receive messages from files in the file system of the broker or, by using FTP or SFTP, in a remote file system. The node generates output message data that any of the output nodes can use, which means that messages can be generated for clients using any of the supported transport protocols to connect to the broker. For more information, see "Using a local file as input for your message flow" on page 1834.
- FTEInput node. Use this node to start a message flow when files are received over a WebSphere MQ File Transfer Edition network. For more information, see "Receiving a file by WebSphere MQ File Transfer Edition" on page 1845.
- CDInput node. Use this node to start a message flow when files are received over a IBM Sterling Connect:Direct network. For more information, see "Receiving a file using IBM Sterling Connect:Direct" on page 1847.
- FileRead node. Use this node to read data from a file in the middle of a message flow. For more information, see "Routing or enriching a message based on the contents of a file" on page 1837.
- FileOutput node. Use this node to write messages to a file in the file system of the broker or, by using FTP or SFTP, in a remote file system. The node can create new files and replace existing files. For more information, see "Writing a file to your local file system" on page 1853.
- FTEOutput node. Use this node to send a file to a remote destination by using a WebSphere MQ File Transfer Edition network. For more information, see "Sending a file by WebSphere MQ File Transfer Edition" on page 1859.

The FileInput and FTEInput nodes start the message flow when a new file arrives, whereas the FileRead node must be connected to another node to start the message flow transaction. The FileRead node also provides keyed access to identify a record, unlike the FileInput node, which processes all records in order.

By using these five nodes, you can also process large files without the complete message being held in memory, and you can simplify the processing of files that have large numbers of repeating entries.

If you want to work with files, read these topics:

- "WebSphere Broker File Transport" on page 1808
- "How the broker processes files" on page 1814
- "How multiple file nodes share access to files in the same directory" on page 1818
- "Using local environment variables with file nodes" on page 1820

- "File name patterns" on page 1830
- "Archiving" on page 1833
- "Reading files" on page 1834
- "Writing a file" on page 1852
- "Transferring files securely by using SFTP" on page 1864
- "Managed file transfers using WebSphere MQ File Transfer Edition" on page 1869
- "FileInput node" on page 4415
- "FTEInput node" on page 4461
- "FileRead node" on page 4444
- "FileOutput node" on page 4430
- "FTEOutput node" on page 4466
- "CDInput node" on page 4305

WebSphere Broker File Transport

WebSphere Broker File Transport is a service that connects applications to data maintained in files.

You can use the WebSphere Broker File Transport to support the following operations.

- Read messages from files in the broker file system.
- Read messages from files in a remote file system.
- Write messages to a file in the broker file system.
- Write messages to a file in a remote file system.
- Process large files without having to hold the complete message in memory.
- Simplify the processing of files that have large numbers of repeating entries.

These operations are implemented by the following built-in nodes:

- FileInput
- FileOutput
- FileRead
- FTEInput
- FTEOutput
- CDInput
- CDOutput

To use the WebSphere Broker File Transport, you must deploy a message flow that contains one of these nodes.

The FileInput node can read one or more messages from a file in a specified input directory or on a remote FTP or SFTP server, and propagate each message as a separate flow transaction. The part of a file that generates one message flow transaction is called a record. A file can be a single record, or a series of records. The node can handle messages in the following message domains:

- MRM
- XMLNSC
- DataObject
- XMLNS
- JMSMap
- JMSStream

- MIME
- BLOB
- XML
- IDOC

The FileOutput node writes one or more messages from message flow transactions to a file in the broker file system. Each message, as it is written to a file, is converted to a sequence of bytes called a record. Records are accumulated until a process is triggered that completes the file and places it either in the specified output directory or a remote FTP or SFTP server directory.

The FTEOutput node processes messages in the same way as the FileOutput node. When the file is complete, the node sends it to the remote agent.

The CDOutput node processes messages in the same way as the FileOutput node. When the file is complete, the node sends it to the IBM Sterling Connect:Direct network.

The following topics contain more information about processing files:

- "Working with files" on page 1807
- "Reading files" on page 1834
- "Writing a file" on page 1852
- "Transferring files securely by using SFTP" on page 1864
- "Managed file transfers using WebSphere MQ File Transfer Edition" on page 1869
- "Initiating a managed file transfer using IBM Sterling Connect:Direct" on page 1873

Related concepts:

"Nodes for connectivity" on page 1028
WebSphere Message Broker supports direct connections from applications, and can send direct requests to other application endpoints. WebSphere Message Broker can also connect to various subsystems including WebSphere MQ, files, and databases, to read and write existing application data.

Related tasks:

"Connecting client applications" on page 1537
Connect your client applications to the broker by using one or more of the supported protocols from other resources and software servers in your network.

Related reference:

"FileInput node" on page 4415
Use the FileInput node to process messages that are read from files.

"FileOutput node" on page 4430
Use the FileOutput node to write messages to files.

"FileRead node" on page 4444
Use the FileRead node to read one record, or the entire contents of a file, from within a message flow.

"FTEInput node" on page 4461
Use the FTEInput node to receive files using the WebSphere MQ File Transfer Edition.

"FTEOutput node" on page 4466
Use the FTEOutput node to write messages to files by using the WebSphere MQ File Transfer Edition.

"CDInput node" on page 4305
Use the CDInput node to preview files when using IBM Sterling Connect:Direct in conjunction with WebSphere Message Broker.

"CDOutput node" on page 4312
Use the CDOutput node when using IBM Sterling Connect:Direct with WebSphere Message Broker.

IBM Sterling Connect:Direct overview and concepts

An overview of IBM Sterling Connect:Direct and its terminology when used with WebSphere Message Broker.

IBM Sterling Connect:Direct is a managed file transfer product that transfers files between, and within, enterprises.

IBM Sterling Connect:Direct, in conjunction with WebSphere Message Broker uses the following terminology:

Connect:Direct server

An application that runs on a machine requiring IBM Sterling Connect:Direct functionality. Files are transferred between two Connect:Direct servers.

- The primary Connect:Direct server, also referred to as the PNODE.
- The secondary Connect:Direct server, also referred to as the SNODE, that receives the transferred file and places the file on the local file system.

Connect:Direct-S

Connect:Direct server which transfers files to other Connect:Direct servers.

Connect:Direct-R

Connect:Direct application that connects to a Connect:Direct server and requests that server does some form of processing. WebSphere Message Broker acts as a Connect:Direct-R to request transfers or receive information about transfers.

WebSphere Message Broker CDInput node

Use the WebSphere Message Broker CDInput node to receive messages that have been transferred to a given Connect:Direct server.

If two or more input nodes are deployed to listen on the same Connect:Direct server, only one receives the file and which one that is cannot be determined. There is some distribution between the nodes, and this is equivalent to having two or more MQInput nodes listening on the same queue.

CDInput nodes can be used in different flows, and in different execution groups, against the same Connect:Direct server.

You can configure the CDInput node to process only a subset of files transferred to the server, based on the directory name and the file name. This allows for multiple CDInput nodes within the same execution group to receive specific files, depending on the filters used.

See "IBM Sterling Connect:Direct concepts" on page 1811 for further information on the CDInput node, CDOutput node, and Connect:Direct server.

WebSphere Message Broker CDOutput node

Use the WebSphere Message Broker CDOutput node to serialize the message tree to a file and then transfer it to a secondary Connect:Direct server using a primary Connect:Direct server. A directory under the work path within the execution group is used as the staging area, until the file is ready to be transferred. Once the file is transferred, it is deleted from the staging area.

Initparm

The definition of the startup parameters for the primary Connect:Direct server (PNODE) and secondary Connect:Direct server (SNODE).

Netmap

The definition of a connection between the primary and secondary Connect:Direct servers.

Processname

The name given to a process that is run in IBM Sterling Connect:Direct. More than one process can be given the same name, and each process is identified uniquely using the process number.

Note that WebSphere Message Broker nodes work purely as clients connecting to the external Connect:Direct server, using the IBM Sterling Connect:Direct Java Application Interface.

By default, WebSphere Message Broker attempts to connect to a local Connect:Direct server, on the default port:

- When sending a file using CDOutput nodes, or
- To be notified of files to process using CDInput nodes.

To enable the CDOutput and CDInput nodes to connect to the local Connect:Direct server, you must set up a username and password using the "**mqsisetdbparms** command" on page 3954.

IBM Sterling Connect:Direct concepts

The CDInput node receives messages that have been transferred to a given Connect:Direct server.

The node receives both the contents of the file and meta data provided by IBM Sterling Connect:Direct on the transfer. One or more CDInput nodes can be used to receive transfers, either in the same flow, different flows, or different execution groups; for any given transfer only one CDInput node receives a message.

You can also specify which transfer a CDInput node can receive, using filters based on directory and file name of the transfer. Once the transfer has been processed there are a set of options of what to do with the transferred file; for further details see "CDInput node" on page 4305.

A simple flow that takes all transferred files from a Connect:Direct server and writes them to a WebSphere MQ queue could contain a CDInput node and an MQOutput node.

Once the CDInput node has been notified of a file to be processed, it processes the file in the same way that a normal FileInput node does. The CDInput node is

notified by the Connect:Direct server manager when a transfer occurs. The notification message is persistently stored on WebSphere MQ and survives broker and queue manager restarts.

The Connect:Direct server manager is defined and runs using a configurable service called *CDServer*. This configurable service is used for administering the access, and processing files using IBM Sterling Connect:Direct.

For more details on the *CDServer* configurable service, see "*CDServer* configurable service properties" on page 3798

Two WebSphere MQ queues are used to store transfer details from the Connect:Direct server:

- SYSTEM.BROKER.CD.STATS
- SYSTEM.BROKER.CD.TRANSFERS

For each successful transfer to the Connect:Direct server, a message is placed on the TRANSFERS queue, which is used to trigger the CDInput node to process the file.

The STATS queue contains internal data used by WebSphere Message Broker to record which transfers have been processed by WebSphere Message Broker.

Clearing both sets of these queues causes all transfers that have occurred up to the current time to be ignored; there is no need, for example, for a restart.

The input node does not poll for transfers, but is triggered instead when they arrive. However, the Connect:Direct server manager does need to poll for events to notify the CDInput node about a transfer. The Connect:Direct server manager has a polling interval of one second.

The CDInput and CDOutput nodes go through the Connect:Direct server manager to both send and receive transfers from a given server, the name of which is set on the node. The Connect:Direct server manager properties are defined using a *CDServer* configurable service.

The Connect:Direct server manager organizes:

- Monitoring the Connect:Direct server for transfers which need to be processed by the CDInput nodes.
- Sending commands created by the CDOutput node to the Connect:Direct server to perform transfers.

In both cases, the Connect:Direct server manager can be used to configure the directory structure to a shared file system, used to both send and receive files from the Connect:Direct server.

The high-level structure of the directory might be different on the two different machines and you can define properties on the Connect:Direct server manager to give the correct mapping.

See "Advanced configuration properties when using IBM Sterling Connect:Direct nodes" on page 727 for further information.

The *Default* configurable service connects to the Connect:Direct server on the local host using the default port. You can modify the default configurable service to a more complex configuration with the CD server on different ports, or a different

machine, or change the node to use a completely different configurable service which is set up to use a different port and hostname.

As well as defining connections details, you can also configure other key properties, such as the location of staging directories and mapping between directories on different machines.

See “Advanced configuration properties when using IBM Sterling Connect:Direct nodes” on page 727 for further information.

The Connect:Direct server manager is started and stopped by the CDInput and CDOutput nodes as required and is not an artifact that is deployed or seen in the deployment view; all of its function is defined by the properties in the configurable service. If transfers happen while WebSphere Message Broker is stopped (or IBM Sterling Connect:Direct flows are stopped) they are not missed and are processed as soon as the flows restart. All transfers are persistent and survive both broker and queue manager restarts.

The configurable service security identity must have adequate permission for the Connect:Direct server manager within the execution group to detect all transfers that are sent to the Connect:Direct server. For example: On UNIX systems, the user's stanza within the `userfile.cfg` in Connect:Direct must have `:cmd.selstats=A:\`. This property can either be set explicitly, or inherited from the higher-level property `:admin.auth=y:\`. If this permission is not present, then the CDInput nodes do not detect or process files that were transferred by other users. For more information about the security identifier, see "**`securityIdentifier`**" in “*CDServer* configurable service properties” on page 3798.

The broker makes a number of API connections to the local Connect:Direct server to process inbound transfers, and to request outbound transfers.

The broker creates one API connection per configurable service per execution group. This creation enables CDOutput nodes to process files sent to the local Connect:Direct server. If there are multiple configurable services that relate to the same Connect:Direct server, then multiple API connections are established.

In addition to these connections, each message flow instance that uses a CDOutput node also creates an API connection to submit transfer requests to the Connect:Direct server. Where there are multiple flows that contain CDOutput nodes, or where these flows have extra defined instances, the broker creates multiple API connections. If the number of API connections that are required exceeds the maximum that is configured in the Connect:Direct server, then the connection fails causing a BIP7951E exception, containing the following information: "`Error: Logon failed! Attempted to exceed maximum API connections`". Avoid this error by increasing the maximum client connections for the Connect:Direct server. The maximum number of connections that are accepted by a Connect:Direct server on UNIX is controlled by the **`api.max.connects`** parameter. For more information about this parameter, and the equivalent settings on other operating systems, see the IBM Sterling Connect:Direct product documentation: http://www.ibm.com/support/knowledgecenter/SS4PJT_5.2.0/cd_52_welcome.html.

Related tasks:

"Initiating a managed file transfer using IBM Sterling Connect:Direct" on page 1873
Use a CDOutput node to send a file from a specified directory on your primary Connect:Direct server (PNODE) to a filename and directory on a secondary Connect:Direct server (SNODE).

"Receiving a file using IBM Sterling Connect:Direct" on page 1847
Use the CDInput node to receive files from an IBM Sterling Connect:Direct network.

Related reference:

"CDInput node" on page 4305
Use the CDInput node to preview files when using IBM Sterling Connect:Direct in conjunction with WebSphere Message Broker.

"CDOutput node" on page 4312
Use the CDOutput node when using IBM Sterling Connect:Direct with WebSphere Message Broker.

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

"*CDServer* configurable service properties" on page 3798
Select the objects and properties that you want to change for the *CDServer* configurable service.

How the broker processes files

The broker reads files with the FileInput, FTEInput, and FileRead nodes, and writes files with the FileOutput and FTEOutput nodes.

WebSphere Message Broker can read messages from files and write messages to files in the local file system, or on a network file system that is local to the broker. The following lists the nodes that provide this capability, and the required access permissions for the directories and files they operate on:

- FileInput node: read and write permissions on the input directory and input files.
- FileOutput node: read and write permissions on the output directory.
- FileRead node: read permissions on the input directory and input files. Note: if a Finish File action is used (to move, rename or delete the file), write permissions on the input directory and input files is also required.

The FileInput, FileOutput, and FileRead nodes inherit the permissions assigned to the broker's user ID and group. See your operating system documentation for details on how to set permissions and the impact on file access.

You can use the FTEInput and FTEOutput nodes to receive or send files to a destination on a WebSphere MQ File Transfer Edition network.

You can use the CDInput and CDOutput nodes to receive or send files to a destination on an IBM Sterling Connect:Direct network.

A file, or a record within a file, is analogous to a message in a queue. The directory that contains the file is analogous to a message queue.

How the broker reads a file at the start of a flow

The FileInput node processes messages that are read from files. The FileInput node searches a specified input directory (in the file system attached to the broker) for files that match specified criteria. Optionally, files from a remote FTP or SFTP

server can be moved to the local directory whenever the directory is to be scanned. You can find the file that you require by specifying an explicit file name or a file name pattern that includes wildcard characters. If the file is locked, it is ignored during the directory scan.

The FileInput node creates an `mqsitransitin` subdirectory in the input directory. The `mqsitransitin` subdirectory holds and locks the input files while they are being processed. The broker reads the file and propagates a message, or messages, by using the contents of the file.

If an execution group that processes files in this input directory is removed, check the `mqsitransitin` subdirectory for partially processed or unprocessed files. Move any such files back into the input directory and remove the execution group UUID prefix from the file names, so that they can be processed by a different execution group. For more information about the `mqsitransitin` subdirectory, see "How multiple file nodes share access to files in the same directory" on page 1818.

In the FileInput node, you can specify how the records are derived from the file. The contents of a file can be interpreted as:

- A single record (whole file)
- Separate records, each of a fixed length (fixed-length records)
- Separate records each delimited by a specified delimiter (delimited records)
- Separate records that are recognized by a parser that you specify (parser record sequence)

After the file has been successfully processed, it is either deleted from the file system or moved to an archive subdirectory of the specified (local) directory.

When the last record of the file has been processed successfully, if the End of Data terminal of the FileInput node is connected in the message flow, an End of Data message is propagated to the End of Data terminal. The End of Data message consists of an empty BLOB message and a `LocalEnvironment.File` structure, and it allows explicit end-of-flow processing to be performed in another part of the message flow.

The message (or messages) propagated from the file can be used as input to any message flow and output node. You can create a message flow that receives messages from files and generates messages for clients that use any of the supported transports to connect to the broker.

Whenever a message is propagated from a file, the FileInput node stores information about the file in the LocalEnvironment.File message tree. This information includes the offset of the record of the message in the file that is being processed, and the record number in that file. In addition, when a wildcard is used in a file name pattern, the characters matched in the file name are placed in the WildcardMatch element of the local environment tree.

If a file is backed out, or a file is left in the `mqsitransitin` subdirectory because processing failed, remove the execution group UUID prefix from the file name and move it back into the input directory. Processing is automatically retried, beginning at the first record in the file.

The FTEInput node receives files from the WebSphere MQ File Transfer Edition network. The broker reads the file and propagates a message, or messages, by using the contents of the file. After an FTEInput node has processed a file, the file

is deleted. For details of information that the FTEInput node provides in the local environment, see “Using local environment variables with file nodes” on page 1820.

The CDInput node receives files from the IBM Sterling Connect:Direct network. The broker reads the file and propagates a message, or messages, by using the contents of the file. After a CDInput node has processed a file, the file is deleted. For details of information that the CDInput node provides in the local environment, see “Using local environment variables with file nodes” on page 1820.

The FileRead node can read a whole file, or a single, nominated record. A common use of the FileRead node is to route or enrich messages based on the contents of the file. When developing the message flow you can specify the name and location of the file to be read. You can override these values at run time based on the contents of a message. TheFileRead node reads a file in the middle of a message flow.

The FileRead node complements the existing FileInput and FileOutput nodes.

How the broker reads a file from the middle of a flow

The FileRead node locks files when they are being read and does not move files to a subdirectory to read them. Files can be read completely or separated into records, in the same way as the FileInput node. After the file has been processed by the FileRead node, it can be deleted, archived, or left unchanged.

How the broker writes a file

The FileOutput, CDOutput, and FTEOutput nodes write messages to files in the file system of the broker. When a message is received on the In terminal of the node, it creates and writes a file as a series of one or more records. One record is written to a file for every message received. The name of the file is specified either by a file name pattern in the node or an explicit file name that is either specified in the node or derived from the message.

You can specify how the records are accumulated in files:

- A single record (whole file). The file that is created consists of one record.
- Concatenated records (unmodified records). The records are not padded to any required length, and they are not separated by a delimiter.
- Uniform length records (fixed-length records). Records that are shorter than the specified length are padded to the required length.
- Separate records (delimited records). Records are either terminated or separated by a specified delimiter.

The message flow informs the output node that there are no more records to write by sending a message to its Finish File terminal. The file is then delivered to the appropriate destination:

- FileOutput node: The file is then moved to the specified output directory.

  Optionally, the file can be moved to a directory on a remote FTP or SFTP server. The server is identified by the `Remote server and port` property on the node. Alternatively, you can override the node property by setting a value in the local environment (see “Local environment overrides for the FileOutput node” on page 4443).

- FTEOutput node: The file is sent to the destination agent on the WebSphere MQ File Transfer Edition network.
- CDOutput node: The file is sent to the IBM Sterling Connect:Direct network.

If the output node produces a single record from the file (whole file), the file is moved immediately to the output directory without requiring a message to be propagated to the Finish File terminal. In this case, any message sent to the Finish File terminal of the node has no effect on any file, but the message is still propagated to a message flow attached to the End of Data terminal.

Related tasks:

"Reading files" on page 1834
Use the FileInput, CDInput, FTEInput, and FileRead nodes to read files.

"Writing a file" on page 1852
Use the FileOutput, CDOutput, and FTEOutput nodes to write files.

Related reference:

"CDInput node" on page 4305
Use the CDInput node to preview files when using IBM Sterling Connect:Direct in conjunction with WebSphere Message Broker.

"CDOutput node" on page 4312
Use the CDOutput node when using IBM Sterling Connect:Direct with WebSphere Message Broker.

"FileInput node" on page 4415
Use the FileInput node to process messages that are read from files.

"FileOutput node" on page 4430
Use the FileOutput node to write messages to files.

"FTEOutput node" on page 4466
Use the FTEOutput node to write messages to files by using the WebSphere MQ File Transfer Edition.

Recognizing file records as messages to be parsed:

Use the FileInput, FTEInput and FileRead nodes to segment your input file into messages that are to be parsed.

The node segments your input file into messages that are to be parsed by one of the following parsers:

- MRM Custom Wire Format (CWF)
- MRM Tagged Delimited String Format (TDS)
- XMLNSC

The `Message domain` property of the node specifies the parser to use; MRM or XMLNSC. Specify `Parsed Record Sequence` for the `Record detection` property so that the node splits the file into messages to be parsed by either the MRM parser or XMLNSC parser.

The MRM parser

If you select an MRM parser, ensure that the message model has a defined message boundary and does not rely on the parse being stopped when it reaches the end of the bit stream. If the final element has a *maxOccurs* value of -1, the parser continues to read bytes until the end of the bit stream or until it encounters bytes that cause a parsing exception. In either case, the parser is unable to identify the end of one message and the start of the next. If you use *Data Element Separation*

= `Use Data Pattern`, ensure that the pattern recognizes a specified number of bytes. Be aware, therefore, that a pattern of `*` identifies all available characters and so would read an entire input file.

If you use delimited separations with message group indicators and terminators, ensure that the combination of group indicator and terminator does not match a record delimiter. For example, a message might start with a left brace ({) and end with a right brace (}). If there is a delimiter of }{ within the message, the delimiter matches the boundary between multiple messages; as a result, a delimiter within the current message might be identified as a message boundary. This might cause bytes in a subsequent message to be included in the current message causing parser exceptions or unexpected content in the parse tree.

The XMLNSC parser

If you select the XMLNSC parser, the end of the root tag marks the end of the message. XML comments, XML processing instructions, and white space that appear after the end of the XML message are discarded. The start of the next XML message is marked either by the next XML root tag or the next XML prolog.

Related concepts:

"MRM Custom Wire Format" on page 1214
Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not separated by delimiters.

"How the broker processes files" on page 1814
The broker reads files with the FileInput, FTEInput, and FileRead nodes, and writes files with the FileOutput and FTEOutput nodes.

"MRM TDS format" on page 1221
The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

"Parsers" on page 1072
A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from the internal message tree representation.

Related tasks:

"Reading files" on page 1834
Use the FileInput, CDInput, FTEInput, and FileRead nodes to read files.

Related reference:

"FileInput node" on page 4415
Use the FileInput node to process messages that are read from files.

"FTEInput node" on page 4461
Use the FTEInput node to receive files using the WebSphere MQ File Transfer Edition.

How multiple file nodes share access to files in the same directory

WebSphere Message Broker controls access to files so that only one file node at a time can read or write to a file.

When a message flow uses the FileInput or FileOutput node, additional instances (threads) might be associated with the message flow, or file nodes in other message flows in the same or different execution groups might refer to files in the same directory. WebSphere Message Broker controls the way in which multiple processes

read from and write to files by moving the files to the mqsitransitin directory during processing, and locking them while they are being processed. The mqsitransitin directory is a subdirectory of the input directory specified in the FileInput node.

The broker locks the files that are being read by the FileInput node or written by the FileOutput node, to prevent other execution groups from reading or changing the files while they are being processed. The broker unlocks the file:

- When a FileInput node finishes processing the input file.
- When a FileOutput node finishes writing the file and moves it from the transit directory to the output directory.

Note: The FTEInput node does not use a transit directory. Each execution group has its own WebSphere MQ File Transfer Edition agent, and a node processes only files sent to the agent for which the node is deployed. The execution group ensures that only one node in the execution group processes each file.

Reading a file

When the FileInput node reads a file, it first moves the file into the mqsitransitin directory, where it is held during processing. A prefix (containing the UUID of the execution group) is added to the file name to indicate which execution group is processing the file. While the file is in this directory, no other execution groups can access the file. The broker maintains a lock subdirectory in the mqsitransitin directory, to ensure that files in the input directory are accessed by only one execution group at a time.

If multiple message flows or instances within an execution group are reading from the same input directory, only one instance of one message flow is allocated to reading it. Each record in the file is serially processed by this instance. Other instances of the message flow, or other message flows, can simultaneously process other files, the names of which match the pattern specified in the File name or pattern property of the node.

While a file is being processed, the file system is used to lock the file. As a result, other programs (including other execution groups) are prevented from reading, writing, or deleting the file while it is being processed by the file nodes.

While a FileInput node is reading a file, the file remains in the mqsitransitin directory until it has been fully processed (or until an unrecoverable error occurs). If the file is to be retained, it is held in a subdirectory of the mqsitransitin directory.

When the file has been processed, it is moved from the mqsitransitin directory back to the input directory. However, if the execution group stops unexpectedly while the file is in the mqsitransitin directory, you can manually restore the input file to the input directory by removing the execution group UUID prefix from the file name, and then moving it to the input directory. The input file is then processed by the next FileInput node that scans the directory.

If you use an NFS server, and have File nodes in different execution groups that access the same directory on the NFS server, ensure that you are using NFS version 4 to correctly support file locking.

Writing a file

Files that are created and written by a FileOutput node are put in the output directory when they are finished. While records are being added to a file, it is kept in the `mqsitransit` subdirectory.

Each record is written by a single message flow instance. All message flow instances that are configured to write records to a specific file can append records to that file. Because instances can run in any order, records that they write might be interleaved, which means that the sequence of records might be altered. If you require the sequence of records in the output file to be maintained, ensure that only one FileOutput node instance uses the file. To ensure that only one FileOutput node instance uses the file, configure the message flow that contains the node to use the additional instances pool with zero instances, and ensure that other message flows do not write to the same file.

While a file is being processed, the file system is used to lock the file. As a result, other programs (including other execution groups) are prevented from reading, writing, or deleting the file while it is being processed by the file nodes. This lock is retained for a short period after a FileOutput node writes to the file without finishing it, leaving it in the transit directory. If message flows that are in the same execution group use the same output file and run sufficiently quickly, the broker does not relinquish the lock before the file is finished. However, if the message flows have longer intervals between them, the broker relinquishes the lock and another process or execution group can acquire a lock on the file. To prevent this situation, ensure that output directories are not shared between execution groups.

Related concepts:

"How the broker processes files" on page 1814
The broker reads files with the FileInput, FTEInput, and FileRead nodes, and writes files with the FileOutput and FTEOutput nodes.

Related reference:

"FileInput node" on page 4415
Use the FileInput node to process messages that are read from files.

"FileOutput node" on page 4430
Use the FileOutput node to write messages to files.

"FTEInput node" on page 4461
Use the FTEInput node to receive files using the WebSphere MQ File Transfer Edition.

Using local environment variables with file nodes

You can use fields in the local environment to dynamically alter the behavior of the FileInput, FileOutput, FTEInput, and FTEOutput nodes. You can also find what values the output nodes used to process the file.

These fields are available in the following message tree structures:

- LocalEnvironment.File
- LocalEnvironment.File.Read
- LocalEnvironment.WrittenDestination.File
- LocalEnvironment.Destination.File
- LocalEnvironment.Destination.File.Remote
- LocalEnvironment.Wildcard.WildcardMatch
- LocalEnvironment.FTE
- LocalEnvironment.WrittenDestination.FTE

- LocalEnvironment.Destination.FTE
- LocalEnvironment.CD
- LocalEnvironment.CD.Transfer
- LocalEnvironment.Destination.CD
- LocalEnvironment.WrittenDestination.CD

LocalEnvironment.File fields

When you use the FileInput node, it stores information that you can access in the LocalEnvironment.File message tree. The fields in this structure are described in the following table.

Element Name	Element Data Type	Description
Directory	CHARACTER	Absolute directory path of the input directory in the form used by the file system of the broker. For example, on Windows systems, this starts with the drive letter prefix (such as `C:`).
Name	CHARACTER	File name and extension.
LastModified	TIMESTAMP	Date and time the file was last modified.
TimeStamp	CHARACTER	Date and time the input node started processing the file in the Coordinated Universal Time (UTC) zone, as a character string. This data is the string used to create archive and backout file names if a timestamp is included.
The following elements contain data about the current record:		
Offset	INTEGER	Start of the record within the file. The first record starts at offset 0. If this element is part of the End of Data message tree, this value is the length of the input file.
Record	INTEGER	Number of the record within the file. The first record is record number 1. If this element is part of the End of Data message tree, this value is the number of records.
Delimiter	CHARACTER	The characters used to separate this record from the preceding record, if `Delimited` is specified in `Record detection`. The first record has a null delimiter. If this element is part of the End of Data message tree, this value is the delimiter that follows the last record, if any.
IsEmpty	BOOLEAN	Whether the record propagated by the message flow is empty. It is set to TRUE if the current record is empty. If this element is part of the End of Data message tree, this value is always set to TRUE.

This structure is propagated with each message written to the Out terminal of the FileInput node and with the empty message written to the End of data terminal.

LocalEnvironment.File.Read fields

When the FileRead node propagates a message, it stores valid information about it in the LocalEnvironment.File.Read message tree. If the input file is empty, an empty message is propagated. The following table lists the LocalEnvironment.File.Read message tree structure.

Element Name	Element Data Type	Description
Directory	CHARACTER	Absolute directory path of the input directory in the form used by the file system of the broker. For example, on Windows systems, this starts with the drive letter prefix (such as `C:`). Alternatively this path relates to the file nodes root directory, which can be overridden with the same environment variable as used for the FileInput and FileOutput nodes.
Name	CHARACTER	File name and extension.
LastModified	TIMESTAMP	Date and time the file was last modified.
TimeStamp	CHARACTER	Date and time the FileRead node started processing the file as a character string, in the Coordinated Universal Time (UTC) zone.
The following elements contain data about the current record:		
Offset	INTEGER	The offset in the file the record starts at. The first byte in the file is offset 0.
NextRecordOffset	INTEGER	The offset in the file that the next record starts at, relative to the start of the file, and is 1 byte after the end of the current record. If the end of the file is reached, then the value is not given in the local environment.
EndOfFile	BOOLEAN	The FileRead node sets this element to TRUE when it has read the last record of the input file. It is therefore always TRUE when the detection property is `Record is Whole File`.
RecordNumber	INTEGER	The number of the record in the file relative to the offset the read node starts reading from. The value is always 1 unless the filter expression is being used, in which case it reflects the number of the record that was selected.
NoMatchReason	STRING	The reason why a message is sent to the "No match" terminal. Null if the message is sent to the Out terminal. Possible reasons: • NoFile – the file does not exist. • NoData – the file exists but has no records. • NoRecord – the file exists and contains records but none match the filter expression.
Delimiter	CHARACTER	The characters used to separate this record from the preceding record, if `Delimited` is specified in `Record detection`. The first record has a null delimiter. If this element is part of the End of Data message tree, this value is the delimiter that follows the last record, if any.
IsEmpty	BOOLEAN	Whether the record propagated by the message flow is empty. It is set to TRUE if the current record is empty.
Archive/Directory	STRING	The name of the directory where the file was archived.
Archive/Name	STRING	The name of the file where the file was archived.

This structure is propagated with each message written to the Out terminal of the FileRead node and with the empty message written to the End of data terminal.

LocalEnvironment.WrittenDestination.File fields

When you use the FileOutput node, it stores information that you can access in the LocalEnvironment.WrittenDestination.File message tree. The fields in this structure are described in the following table.

Element Name	Element Data Type	Description
Directory	CHARACTER	Absolute directory path of the output directory in the form used by the file system of the broker. For example, on Windows systems, this starts with the drive letter prefix (such as `C:`).
Name	CHARACTER	File name of the output file.
Action	CHARACTER	Possible values are: • `Replace` if an output file of the same name is replaced. • `Create` if a new output file is created. • `Append` if this value is associated with a record that is appended to an output file. • `Finish` if a Finish File message is received and no file is found to finish (for example, if `Record is Whole File` is specified and a message is sent to the Finish File terminal). • `Transmit` if the file was transferred by FTP or SFTP and the file was not retained.
Timestamp	CHARACTER	The date and time, in character string form, when the node started to process this file. This value prefixes the names of files that are archived if you specify `Time Stamp`, `Archive and Replace Existing File` in the `Output file action` property on the **Basic** tab.

LocalEnvironment.Destination.File fields

When you use the FileOutput and FileRead nodes, you can override the directory and name properties with elements in the message tree. The default location for these overrides is LocalEnvironment.Destination.File, although you can change this location by using the properties on the `Request directory property location` and `Request file name property location` on the FileOutput node. When you use the FileRead node, you can also override the length and offset properties. The fields of this structure are described in the following table.

Element Name	Element Data Type	Description
Directory	CHARACTER	This property specifies the absolute or relative directory path of the output directory in the form that is used by the file system of the broker. For example, on Windows systems, this path starts with the drive letter prefix (such as C:) and use a backslash (\) as the directory delimiter. On UNIX systems, the path includes a slash (/) as the directory delimiter.
Name	CHARACTER	This property specifies the file name of the output file. The FileOutput node does not perform wildcard replacement on the value of the element. For example, if its value is `Input*.txt`, the FileOutput node tries to write to a file with an asterisk (*) in its name. It might or might not succeed, depending on whether an asterisk is a valid character for files in the file system to which it is writing.
Length	INTEGER	This property specifies the length of the record to read from the file. The value is only used if the record detection option fixed length is being used.
Offset	INTEGER	This property specifies the offset in the file to start searching for a record. Offset 0 means start from the beginning and is the default value if no override is given.

Element Name	Element Data Type	Description
Archive/Directory	STRING	The directory where the file is archived to when using one of the file disposition archive options. By default the file is archived to 'mqsiarchive' under the file input directory. Any path is not relative to the input directory but relative to the `MQSI_FILENODES_ROOT_DIRECTORY`.
Archive/Name	STRING	The pattern to use to create an archive file name. Only one star is allowed in the file name and the star is replaced with the first star replace in the file pattern name. If `Archive with Time Stamp` is specified, then a time stamp is appended to the archive name.

LocalEnvironment.Destination.File.Remote fields

When you use the FileOutput node with the `Remote Transfer` property selected, you can override the directory name with an element in the local environment tree. The fields of this structure are described in the following table.

Element Name	Element Data Type	Description
Remote.ServerDirectory	CHARACTER	This property specifies the absolute or relative directory path of the output directory on the remote server. The property has no effect if SFTP or FTP is not enabled on the FileOutput node. Format the path according to the path syntax that is accepted by the FTP server, typically by using UNIX-style slash (/) directory delimiters.
Remote.Server	CHARACTER	This property specifies the name of an FtpServer configurable service, a server name, or a server name and port (using the syntax `serverName:port`) to use to connect to when connecting to an FTP or SFTP server from the FileOutput node. The property has no effect if SFTP or FTP is not enabled on the FileOutput node.

LocalEnvironment.Wildcard.WildcardMatch field

On the FileInput, CDInput, and FTEInput nodes, you can specify a file name pattern that contains wildcard characters. The input nodes copy the characters in the file name matched by wildcards, together with any intermediate characters, to LocalEnvironment.Wildcard.WildcardMatch.

Element Name	Element Data Type	Description
WildcardMatch	CHARACTER	The character string in the file name matched by wildcards in the file name pattern.

On the FileOutput, CDOutput, and FTEOutput nodes, you can use a wildcard character in the file name pattern. If you include the single wildcard character, '*', in the file name pattern, the node uses the value that is stored in LocalEnvironment.Wildcard.WildcardMatch. This is useful if you have a message flow where the input and output nodes are working with the same file; you can preserve the name of the input file on the output nodes. You can also use standard methods for manipulating the value of the WildcardMatch element to whatever you want; you must not use a FileInput or FTEInput node.

See “File name patterns” on page 1830 for more information.

LocalEnvironment.FTE fields

When you use the FTEInput node, it stores information that you can access in the LocalEnvironment.FTE and LocalEnvironment.FTE.Transfer message trees. The LocalEnvironment.FTE message tree stores information relating to the current record and is populated by the broker. The fields in this structure are described in the following table:

Element Name	Element Data Type	Description
TimeStamp	CHARACTER	Date and time the input node started processing the file in the Coordinated Universal Time (UTC) zone, as a character string. This data is the string used to create archive and backout file names if a timestamp is included.
Offset	INTEGER	Start of the record within the file. The first record starts at offset 0 bytes. When Offset is part of the End of Data message tree, this value is the length of the input file.
Record	INTEGER	Number of the record within the file. The first record is record number 1. When Record is part of the End of Data message tree, this value is the number of records.
Delimiter	CHARACTER	The characters used to separate this record from the preceding record, if `Delimited` is specified in `Record detection`. The first record has a null delimiter. When Delimiter is part of the End of Data message tree, this value is the delimiter that follows the last record, if any.
IsEmpty	BOOLEAN	Whether the record propagated by the message flow is empty. IsEmpty is set to TRUE if the current record is empty. When IsEmpty is part of the End of Data message tree, this value is always set to TRUE.

The LocalEnvironment.FTE.Transfer message tree contains information received from WebSphere MQ File Transfer Edition regarding the transfer or file; see WebSphere File Transfer Edition Information Center for more details. The fields in this structure are described in the following table.

Element Name	Element Data Type	Description
Directory	CHARACTER	Absolute directory path of the input directory.
JobName	CHARACTER	The name for the transfer.
Name	CHARACTER	File name and extension (per file).
LastModified	TIMESTAMP	Date and time the file was last modified (per file).
SourceAgent	CHARACTER	The name of the agent sending the file.
DestinationAgent	CHARACTER	The name of the agent to send the file to.
OriginatingHost	CHARACTER	The name of the host from which the transfer was submitted.
TransferId	CHARACTER	The unique name of the transfer.
MQMDUser	CHARACTER	The WebSphere MQ user ID in the MQMD of the transfer request message.
OriginatingUser	CHARACTER	The user ID of the user that submitted the transfer request.
TransferMode	CHARACTER	The mode of the transfer. Valid values are `Binary` or `Text`.
TransferStatus	CHARACTER	The status of the transfer of the file.
FileSize	INTEGER	The size of the file being transferred.
ChecksumMethod	CHARACTER	The only allowed value is `MD5`.

Element Name	Element Data Type	Description
Checksum	CHARACTER	If the ChecksumMethod element is set to MD5, this element is the actual checksum in hex string format.
DestinationAgentQmgr	CHARACTER	The name of the queue manager of the destination agent to send the file to.
SourceAgentQmgr	CHARACTER	The name of the queue manager of the source agent that sent the file.
OverallTransferStatus	CHARACTER	The overall status of the transfer.
TotalTransfers	INTEGER	The total number of files successfully transferred.
TransferNumber	INTEGER	The number of the current file in the transfer.

These structures are propagated with each message written to the Out terminal of the FTEInput node and with the empty message written to the End of data terminal.

LocalEnvironment.WrittenDestination.FTE fields

When you use the FTEOutput node, it stores information that you can access in the LocalEnvironment.WrittenDestination.FTE message tree. The fields in this structure are described in the following table.

Element Name	Element Data Type	Description
DestinationAgent	CHARACTER	The name of the agent to send the file to.
DestinationQmgr	CHARACTER	The name of the destination queue manager.
JobName	CHARACTER	The name for the transfer.
Directory	CHARACTER	Absolute directory path of the output directory in the form used by the file system of the broker. For example, on Windows systems, this starts with the drive letter prefix (such as C:).
Name	CHARACTER	File name of the output file.
Overwrite	BOOLEAN	Specifies whether files on the destination system can be overwritten when the destination agent moves files of the same name there. If the destination agent fails to overwrite the file, the transfer fails and the transfer logs report the failure. The FTEOutput node does not throw or log any errors.
TransferId	CHARACTER	The unique name of the transfer initiated by the FTEOutput node.

LocalEnvironment.Destination.FTE fields

When you use the FTEOutput node, you can override its Destination agent, Destination queue manager, Job name, Destination file directory, Destination file name, and Overwrite files on destination system properties with elements in the message tree. You can also call a program on the destination agent before starting the transfer, or when the transfer is finished. The default location for these overrides is LocalEnvironment.Destination.FTE. The fields of this structure are described in the following table.

Element Name	Element Data Type	Description
DestinationAgent	CHARACTER	The name of the agent to send the file to.
DestinationQmgr	CHARACTER	The name of the destination queue manager.
JobName	CHARACTER	The name for the transfer.

Element Name	Element Data Type	Description
Directory	CHARACTER	Absolute directory path of the output directory in the form used by the file system of the broker. For example, on Windows systems, this starts with the drive letter prefix (such as `C:`).
Name	CHARACTER	File name of the output file.
Overwrite	BOOLEAN	Specifies whether files on the destination system can be overwritten when the destination agent moves files of the same name there. If the destination agent fails to overwrite the file, the transfer fails and the transfer logs report the failure. The FTEOutput node does not throw or log any errors.
PreDestinationCall.Name	CHARACTER	Call a program on the destination agent before starting the transfer. This element supplies the name of an Ant script to run. You cannot call other programs that are not Ant scripts, or pass parameters to the calls. The Ant script can access all the metadata defined for the transfer, including user metadata added using the local environment override LocalEnvironment.Destination.FTE.UserDefined. See WebSphere File Transfer Edition Information Center for more details of how to use the PreDestinationCall function.
PostDestinationCall.Name	CHARACTER	Call a program on the destination agent after completing the transfer. This element supplies the name of an Ant script to run. You cannot call other programs that are not Ant scripts, or pass parameters to the calls. The Ant script can access all the metadata defined for the transfer, including user metadata added using the local environment override LocalEnvironment.Destination.FTE.UserDefined. See WebSphere File Transfer Edition Information Center for more details of how to use the PostDestinationCall function.

LocalEnvironment.CD fields

When you use the CDInput node, it stores information that you can access in the LocalEnvironment.CD and LocalEnvironment.CD.Transfer message trees. The LocalEnvironment.CD message tree stores information relating to the current record and is populated by the broker. The fields in this structure are described in the following table:

Element Name	Element Data Type	Description
Transfer	Folder	Contains meta data from the IBM Sterling Connect:Direct transfer.
Timestamp	CHAR	Timestamp of the file.
Offset	INTEGER	Start of the record within the file. The first record starts at offset 0 bytes. When Offset is part of the End of Data message tree, this value is the length of the input file.
Record	INTEGER	Number of the record within the file. The first record is record number 1. When Record is part of the End of Data message tree, this value is the number of records.
Delimiter	CHARACTER	The characters used to separate this record from the preceding record, if `Delimited` is specified in `Record detection`. The first record has a null delimiter. When Delimiter is part of the End of Data message tree, this value is the delimiter that follows the last record, if any.
IsEmpty	BOOLEAN	Whether the record propagated by the message flow is empty. IsEmpty is set to TRUE if the current record is empty. When IsEmpty is part of the End of Data message tree, this value is always set to TRUE.

These structures are propagated with each message written to the Out terminal of the CDInput node and with the empty message written to the End of data terminal.

LocalEnvironment.CD.Transfer

The LocalEnvironment.CD.Transfer message tree contains information received from IBM Sterling Connect:Direct regarding the transfer or file. The fields in this structure are described in the following table.

Element Name	Element Data Type	Description
ProcessName	CHARACTER	The process name of the script transferring the file.
StepName	CHARACTER	The step name causing the transfer to take place.
ProcessNumber	INTEGER	The number of the process running the process script.
Submitter	CHAR	The user ID submitting the process script.
Accounting	CHAR	The secondary node (SNODE) accounting details for the process script.
SourcePath	CHAR	The source path of the file on the primary node (PNODE) machine.
DestinationPath	CHAR	The destination path of the file on the secondary node (SNODE) machine
Directory	CHARACTER	The directory which the file is copied to.
Name	CHARACTER	The name of the file copied to.
PrimaryNodeName	CHARACTER	The name of the primary node from which the file was copied.
SecondaryNodeName	CHARACTER	The name of the secondary node.

LocalEnvironment.Destination.CD fields

When you use the CDOutput node, you can override various destination system properties with elements in the message tree. The default location for these overrides is LocalEnvironment.Destination.CD. The fields of this structure are described in the following table

Element Name	Element Data Type	Description
SNODE	CHARACTER	The name of the secondary Connect:Direct server (SNODE) to send the file to.
ProcessName	CHARACTER	The process name that the script uses to run.
Accounting	CHARACTER	Accounting data shown when the script is running on both the primary Connect:Direct server (PNODE) and secondary Connect:Direct server (SNODE).
Directory	CHARACTER	Absolute directory path of the output directory in the form used by the file system of the broker. For example, on Windows systems, this starts with the drive letter prefix (such as `C:`).
Name	CHARACTER	File name of the output file.
Copy.From	CHARACTER	The final part of the path name is the IBM Sterling Connect:Direct process script property you want to change. This is either a direct `<option name>` on the FROM clause, or a value in the `<SYSOPTS>` option. You must take care to ensure that the created script is valid, because any existing value created by the node is overridden.

Element Name	Element Data Type	Description
Copy.To	CHARACTER	The final part of the path name is the IBM Sterling Connect:Direct process script property you want to change. This is either a direct <option name> on the TO clause, or a value in the <SYSOPTS> option. You must take care to ensure that the created script is valid, because any existing value created by the node is overridden.

LocalEnvironment.WrittenDestination.CD fields

When you use the CDOutput node, it stores information that you can access in the LocalEnvironment.WrittenDestination.CD message tree. The fields in this structure are described in the following table.

Element Name	Element Data Type	Description
ProcessName	CHARACTER	The name of the process sending the file.
ProcessNumber	CHARACTER	The number of the process sending the file.
Directory	CHARACTER	Absolute directory path of the output directory in the form used by the file system of the broker. For example, on Windows systems, this starts with the drive letter prefix (such as C:).
Name	CHARACTER	File name of the output file.
PrimaryNodeName	CHARACTER	The name of the primary Connect:Direct server (PNODE)
PrimaryNodeOS	CHARACTER	The operating system of the primary Connect:Direct server
SecondaryNodeName	CHARACTER	The name of the secondary Connect:Direct server (SNODE)
SecondaryNodeOS	CHARACTER	The operating system of the secondary Connect:Direct server (this might not be the same as the WebSphere Message Broker operating system)

Related concepts:

"Local environment tree structure" on page 1056
The local environment tree is a part of the logical message tree in which you can store information while the message flow processes the message.

"File name patterns" on page 1830
You can specify a file name pattern, using wildcard characters, to identify a file to be read by the FileInput and FTEInput nodes. You can also specify a file name pattern, using a single wildcard character, to name the file to be created by the FileOutput and FTEOutput nodes.

"How the broker processes files" on page 1814
The broker reads files with the FileInput, FTEInput, and FileRead nodes, and writes files with the FileOutput and FTEOutput nodes.

Related tasks:

"Accessing the local environment tree" on page 2463
The local environment tree has its own correlation name, LocalEnvironment, and you must use this name in all ESQL statements that refer to or set the content of this tree.

Related reference:

"FileInput node" on page 4415
Use the FileInput node to process messages that are read from files.

"FileOutput node" on page 4430
Use the FileOutput node to write messages to files.

"FileRead node" on page 4444
Use the FileRead node to read one record, or the entire contents of a file, from within a message flow.

"FTEInput node" on page 4461
Use the FTEInput node to receive files using the WebSphere MQ File Transfer Edition.

"FTEOutput node" on page 4466
Use the FTEOutput node to write messages to files by using the WebSphere MQ File Transfer Edition.

"CDInput node" on page 4305
Use the CDInput node to preview files when using IBM Sterling Connect:Direct in conjunction with WebSphere Message Broker.

"CDOutput node" on page 4312
Use the CDOutput node when using IBM Sterling Connect:Direct with WebSphere Message Broker.

File name patterns

You can specify a file name pattern, using wildcard characters, to identify a file to be read by the FileInput and FTEInput nodes. You can also specify a file name pattern, using a single wildcard character, to name the file to be created by the FileOutput and FTEOutput nodes.

Using file name patterns with the FileInput and FTEInput nodes

The input nodes read files from a specified directory and propagate messages based on the contents of these files. Only files with names that match a pattern (the input pattern), as specified in the FileInput node's `File name or pattern` property or the FTEInput node's `File name filter` property, are read.

The match might be with a file name or a character sequence (a pattern). A pattern is a sequence containing at least one of the following wildcard characters:

Wildcard character	Description	Example
*	Any sequence of zero or more characters	`*.xml` matches all file names with an `xml` extension
?	Any single character	`f??????.csv` matches all file names consisting of the letter `f` followed by six characters, then the sequence `.csv`.

The default pattern is `*`, which matches all file names.

You cannot specify file names that contain the following characters: the asterisk ('*'), the question mark ('?'), or file name separator characters ('/' and' \').

For example:

- If you want the FileInput node to process all files that have a certain extension, such as `xml`, set its `File name or pattern` property to `*.xml` and the node will process all files in the directory that have this extension.

If you deploy the flow to a Windows server, file names match the pattern irrespective of case. However, if you deploy the flow to a Linux, UNIX, or z/OS server, file names must match the pattern character string and its case.

Pattern matching

The FileInput and FTEInput nodes set the LocalEnvironment.Wildcard.WildcardMatch element to the string matched by wildcards in the file name. The following are some examples of pattern matching with the value in this element, where the value in the FileInput node's `File name or pattern` property is `File????.from*.xml`:

- If the FileInput node finds a file with the file name `File1234.fromHQ.xml`, there is a match. The value in the LocalEnvironment.Wildcard.WildcardMatch element is set to `1234.fromHQ` and the node processes the file.
- If the file name is `File123.fromHQ.xml`, there is no match because there are insufficient characters between the `File` and `.from` elements of the file name. The FileInput node ignores this file.
- If the file name is `File2345.from.xml`, there is a match. The value in the LocalEnvironment.Wildcard.WildcardMatch element is set to `2345.from` and the node processes the file. In this example, the `*` in the character string in the `File name or pattern` property matches a string of zero characters. If you require the character string between the `from` and `.xml` elements of the file name to always have at least one character, you specify the `File name or pattern` property with a value of `File????.from?*.xml`.

Using file name patterns with the FileOutput and FTEOutput nodes

The node writes messages to files that it creates or replaces in the broker's file system. Only patterns containing a single wildcard character (the asterisk, '*') are allowed in this property. The file name to be used is determined in the following way:

- If the file name property contains no wildcard, the value of this property is the name of the file created. This value must be a valid file name on the file system that hosts the broker to which the message flow is deployed.
- If the file name property contains a single wildcard, the value of the element LocalEnvironment.Wildcard.WildcardMatch in the current message replaces the wildcard character, and the resulting value is the name of the file created. This value must be a valid file name on the file system that hosts the broker to which the message flow is deployed. If the WildcardMatch value is not found, the wildcard character is replaced by the empty string.

You cannot specify file names that contain the following characters: the asterisk ('*'), the question mark ('?'), file name separator characters ('/' and' \'). The name of the file can be overridden by values in the current message.

If the `File name or pattern` property on the FileOutput node is empty, the name must be overridden by the current message. Wildcard substitution occurs only if this property is not overridden in this way.

File names are passed to the file system to which the broker has access, and have to respect the conventions of these file systems. For example, file names on Windows systems are not case-sensitive; on UNIX systems, file names that differ by case are considered distinct.

Example: If the FileInput node has `*.out` in the `File name or pattern` property, and the incoming file is `myfile`, the name of the outgoing file is `myfile.out`.

FTP and SFTP considerations

You can use the FileInput node to transfer files from a remote FTP or SFTP server and process them. Only files with names that match the file name pattern specified in the node are read. If your broker is on an operating system that respects case sensitivity (such as UNIX), you might specify a pattern that includes a combination of uppercase and lowercase characters. If you then use this pattern to process files that are in a directory on a remote FTP or SFTP server, and this server is running on an operating system that does not respect case sensitivity (such as Windows), file name matching might fail with the result that no files are processed. This failure occurs because the file names on the remote server are not in mixed case. If your broker is on an operating system that does not respect case sensitivity, any pattern that you specify might be matched by more than one file on a remote FTP or SFTP server that is running on an operating system on which case sensitivity is significant. Each of these files is then processed sequentially.

You can use the FileOutput node to write files to a remote FTP or SFTP server. Only files with names that match the pattern specified in the node are written. If your broker is running on an operating system that respects case sensitivity (such as UNIX), you might specify a pattern that includes a combination of uppercase and lowercase characters. However, if you then use this pattern to write files to a directory on a remote FTP or SFTP server running on an operating system that does not respect case sensitivity (such as Windows), the file name is written in uppercase rather than in the way that it was specified in your pattern.

If the name of a file on a remote FTP server contains one or more characters that are not valid on the operating system on which the broker where you specified the file name pattern is running, the file is not transferred from the FTP server for processing by the FileInput node.

Related concepts:

"How the broker processes files" on page 1814
The broker reads files with the FileInput, FTEInput, and FileRead nodes, and writes files with the FileOutput and FTEOutput nodes.

"Using local environment variables with file nodes" on page 1820
You can use fields in the local environment to dynamically alter the behavior of the FileInput, FileOutput, FTEInput, and FTEOutput nodes. You can also find what values the output nodes used to process the file.

Related tasks:

"Connecting client applications" on page 1537
Connect your client applications to the broker by using one or more of the supported protocols from other resources and software servers in your network.

"Reading files" on page 1834
Use the FileInput, CDInput, FTEInput, and FileRead nodes to read files.

"Writing a file" on page 1852
Use the FileOutput, CDOutput, and FTEOutput nodes to write files.

Related reference:

"FileInput node" on page 4415
Use the FileInput node to process messages that are read from files.

"FileOutput node" on page 4430
Use the FileOutput node to write messages to files.

"FTEOutput node" on page 4466
Use the FTEOutput node to write messages to files by using the WebSphere MQ File Transfer Edition.

"FTEInput node" on page 4461
Use the FTEInput node to receive files using the WebSphere MQ File Transfer Edition.

Archiving

Files that are successfully processed by the FileInput node or FileOutput node can optionally be moved to the `mqsiarchive` subdirectory of the input or output directory.

The input directory of the FileInput node has a subdirectory called `mqsiarchive`. The output directory of the FileOutput node also has a subdirectory called `mqsiarchive`. Archiving is not enabled for the FTEOutput and FTEInput nodes.

FileInput node

Files that are processed successfully by the FileInput node are moved to the `mqsiarchive` subdirectory if the FileInput node's `Action on successful processing` property is set to `Move to Archive Subdirectory` or `Add Timestamp and Move to Archive Subdirectory`.

Select the `Replace duplicate archive files` check box to overwrite existing files in the `mqsiarchive` subdirectory. If you do not set this option, and a file with the same name already exists in the archive subdirectory, the node stops processing files. Every time that the node returns from its polling wait period, it issues a pair of messages, BIP3331 and a more specific one describing the problem. To avoid writing too many messages, duplicate messages are suppressed for increasing periods of time, until eventually they are issued only about once every hour. In this circumstance, the system administrator must stop the flow, correct the problem, then restart the flow.

Clear the `Replace duplicate archive files` check box only if you are sure either that the input files have unique names, or that some other process will remove a file from the archive directory before the FileInput node processes another of the same name. If you cannot ensure this, either specify `Add Timestamp and Move to Archive Subdirectory` in the `Action on successful processing` property so that archived files have unique names, or select the `Replace duplicate archive files` check box

FileOutput node

Files that are processed successfully by the FileOutput node are moved to the `mqsiarchive` subdirectory if the `Output file action` property of the FileOutput node is set to `Archive and Replace Existing File` or `Time Stamp, Archive and Replace Existing File`.

Select the `Replace duplicate archive files` check box to overwrite existing files in the `mqsiarchive` subdirectory. If you do not set this option, and a file with the same name already exists in the archive subdirectory, the node generates an exception when it tries to move the successfully processed file and the file remains in the transit subdirectory.

Related concepts:

"How the broker processes files" on page 1814
The broker reads files with the FileInput, FTEInput, and FileRead nodes, and writes files with the FileOutput and FTEOutput nodes.

Related tasks:

"Reading files"
Use the FileInput, CDInput, FTEInput, and FileRead nodes to read files.
"Writing a file" on page 1852
Use the FileOutput, CDOutput, and FTEOutput nodes to write files.

Related reference:

"FileInput node" on page 4415
Use the FileInput node to process messages that are read from files.
"FileOutput node" on page 4430
Use the FileOutput node to write messages to files.

Reading files

Use the FileInput, CDInput, FTEInput, and FileRead nodes to read files.

About this task

This section contains the following topics:

- "Using a local file as input for your message flow"
- "Routing or enriching a message based on the contents of a file" on page 1837
- "Reading a file on a remote FTP or SFTP directory" on page 1842
- "Receiving a file by WebSphere MQ File Transfer Edition" on page 1845
- "Controlling how files are separated into records" on page 1848
- "Receiving a file using IBM Sterling Connect:Direct" on page 1847

Related concepts:

"How the broker processes files" on page 1814
The broker reads files with the FileInput, FTEInput, and FileRead nodes, and writes files with the FileOutput and FTEOutput nodes.

Related tasks:

"Working with files" on page 1807

Related reference:

"FileInput node" on page 4415
Use the FileInput node to process messages that are read from files.
"FileRead node" on page 4444
Use the FileRead node to read one record, or the entire contents of a file, from within a message flow.
"FTEInput node" on page 4461
Use the FTEInput node to receive files using the WebSphere MQ File Transfer Edition.
"CDInput node" on page 4305
Use the CDInput node to preview files when using IBM Sterling Connect:Direct in conjunction with WebSphere Message Broker.

Using a local file as input for your message flow:

Learn how to use the FileInput node to read a file on your local file system and then propagate messages that are based on the contents of that file.

Before you begin

This example shows how one combination of values in the `Record detection`, `Delimiter`, and `Delimiter type` properties can be used to extract messages from a file. The example describes the FileInput node of a message flow and assumes that

the rest of the flow has already been developed. It is also assumed that a Windows system is being used. To complete this example task, you must first have added a FileInput node to a message flow. You also need the following resources:

- An input file. To follow this example scenario, create an input file called `test_input1.xml` with the following content:

  ```
  <Message>test1</Message>
  <Message>testtwo</Message>
  <Message>testthree</Message>
  ```

  Each line ends with a line terminator; on a Windows system, this comprises carriage return and line feed characters (X'0D0A'). Put this file into directory `C:\FileInput\TestDir`.
- A message set. This example uses a message set called `xml1` which uses the XMLNSC parser. Message set `xml1` models messages of the following form:

  ```
  <Message>...</Message>
  ```

Complete the following steps:

Procedure

1. Set the required node properties on the FileInput node. The following table summarizes the FileInput node properties that you should set, which tab they appear on and the value that you should set in order to follow this example:

Tab	Property	Value
Basic	`Input directory`	`C:\FileInput\TestDir`
	`File name or pattern`	`test_input1.xml`
	`Action on successful processing`	`Move to Archive Subdirectory`
	`Replace duplicate archive files`	Selected
Input Message Parsing	`Message domain`	`XMLNSC`
	`Message set`	`xml1`
Polling	`Polling interval`	3
Retry	`Action on failing file`	`Add Time Stamp and Move to Backout Subdirectory`
Records and Elements	`Record detection`	`Delimited`
	`Delimiter`	`DOS or UNIX Line End`
	`Delimiter type`	`Postfix`
FTP	`FTP`	Not selected

2. Deploy the message flow to the broker. See Chapter 11, “Packaging and deploying,” on page 3209.

Results

The following actions occur when you perform these steps:

1. The file is processed. In accordance with the values set in the properties on the **Records and Elements** tab, the FileInput node detects records that are separated by DOS or UNIX end-of-line characters and creates a message for each one that it finds. It propagates three messages to the flow attached to the Out terminal:

- Message 1:
  ```
  <Message>test1</Message>
  ```
- Message 2:
  ```
  <Message>testtwo</Message>
  ```
- Message 3:
  ```
  <Message>testthree</Message>
  ```

2. If a flow is attached to the End of Data terminal, the End of Data message is propagated after the last record in the file has been processed.
3. When processing is complete, the file `test_input1.xml` is moved to the `mqsiarchive` subdirectory, `C:\FileInput\TestDir\mqsiarchive\test_input1.xml`. If a file called `test_input1.xml` already exists in the `mqsiarchive` subdirectory, it is overwritten.
4. If the message flow fails, retry processing is attempted according to the values set in the properties of the FileInput node. In this example task, a time stamp is added to the file name and the file is moved to the `mqsibackout` directory. Here is an example of the path to such a file: `C:\FileInput\TestDir\mqsibackout\20070928_150234_171021_test_input1.xml`.

What to do next

To see the effects of specifying other combinations of values in the `Record detection`, `Delimiter`, and `Delimiter type` properties of the FileInput node, see "Controlling how files are separated into records" on page 1848.

The following samples also show how to use this node:

- Batch Processing
- WildcardMatch

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Related concepts:

"How the broker processes files" on page 1814
The broker reads files with the FileInput, FTEInput, and FileRead nodes, and writes files with the FileOutput and FTEOutput nodes.

Related tasks:

"Reading a file on a remote FTP or SFTP directory" on page 1842
Use a FileInput node to read a file in a directory on a remote FTP or SFTP server and then propagate messages that are based on the contents of that file.

"Writing a file to your local file system" on page 1853
Use a FileOutput node to write a file to a specified directory on your local file system.

Related reference:

"Recognizing file records as messages to be parsed" on page 1817
Use the FileInput, FTEInput and FileRead nodes to segment your input file into messages that are to be parsed.

"FileInput node" on page 4415
Use the FileInput node to process messages that are read from files.

"Controlling how files are separated into records" on page 1848
Set the `Record detection` and other properties on node **Records and Elements** tabs

to read files in different formats.

Routing or enriching a message based on the contents of a file:

The FileRead node can route or enrich messages based on the contents of the file.

When developing the message flow you can specify the name and location of the file to be read. You can override these values at run time based on the contents of a message.

The node complements the existing FileInput and FileOutput nodes. The FileRead node reads a file in the middle of a message flow.

Using the node to route messages

A message is routed by using the contents of a file colocated with WebSphere Message Broker or on a network file system. The message from the source system is routed to a target system by using an external routing file. No response is expected.

The basic flow of events is as follows:

- WebSphere Message Broker receives a message through an input node.
- The WebSphere Message Broker message flow interrogates the contents of a message to identify routing key information.
- If the file consists of more than one record, you need to determine:
  - Where the first record starts. Unless you specify an offset byte, the node starts reading the file at the first byte.
  - How each record ends (fixed-sized, delimited, or parsed.)
  - Which record to propagate. You can use any combination of information from the input message and the file in deciding this. All records from the specified start point are read until a record is found that matches the record selection expression, this record is then propagated. Examples include:

    The third record, as identified by the local environment field `$OutputLocalEnvironment/File/Read/RecordNumber=3`. In this example, the first record is fully read and the expression evaluates to false. The second record is then fully read and the expression evaluates to false. When the third record is fully read, the expression evaluates to true and the record is propagated. No further records are read.

    A key field in the input message matches a key field in the file `$InputRoot/XMLNSC/FromMQInputMessage/Record1 = $ResultRoot/XMLNSC/FromFile/Record5`. In this example, records are read from the file until the value of the Record5 element of the record matches the value of the Record1 element of the incoming message. The location of the record in the file determines how many records the node must read before successfully matching the record selection expression.
- Within the message flow you can implement a local cache of records to reduce the performance cost of reading multiple static records. For an example of how to implement a local cache, see the following sample:
  - Message Routing
- You can choose to take information from the file, and copy it to the outgoing message. The copy can be a subset of the data, and can be copied to any location in the message or the local environment.
- A target application receives the routed message.

For information about how to process messages based on the contents of an XML or CSV file, see the following sample:

- Message Routing

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related reference:

"FileRead node" on page 4444
Use the FileRead node to read one record, or the entire contents of a file, from within a message flow.

"FileInput node" on page 4415
Use the FileInput node to process messages that are read from files.

"FileOutput node" on page 4430
Use the FileOutput node to write messages to files.

Combining a WebSphere MQ message with an XML file using the contents of the message to identify which file to use:

Combine an incoming message with the contents of an XML file, using fields in the message to determine which file to use.

Before you begin

Before you start: Put a file on the file system that is local to the broker, for the FileRead node to read. Here is an example of the file contents:

```
<Data>Purchase details</Data>
```

In this example, the contents of the data in the data tag are inserted into the incoming message. Any valid XML structures can be added to this section.

Make a note of the path to the file. For example: `c:\temp\FileRead\task3.xml` or `/tmp/FileRead/task3`.

Create the following queues on the broker queue manager:

- FILEREAD.TASK3.IN1
- FILEREAD.TASK3.OUT1

Detailed information about configuring the node is given on the property panels for the node, in the WebSphere Message Broker Toolkit.

Procedure

1. Create a message flow that contains an MQInput node, a FileRead node, and an MQOutput node.
2. Wire the terminals as follows:
   a. Wire the Out terminal of the MQInput node to the In terminal of the FileRead node.

b. Wire the Out terminal of the FileRead to the In terminal of the MQOutput node.

3. Configure the MQInput node:
   a. On the Basic panel, set the Queue name to `FILEREAD.TASK3.IN1`
   b. On the Input Message Parsing panel set the domain to `XMLNSC`.
4. Configure the FileRead node.
   a. On the Basic panel, set the directory and file name to refer to the XML file. For example: `c:\temp\FileRead` and `task3.xml` or `/tmp/FileRead` and `task3`.
   b. Configure the Result panel:
      1) Set the Result data location to `$ResultRoot/XMLNSC/Data`
      2) Set the Output data location to `$InputRoot/XMLNSC/Data`
   c. Configure the Input Message Parsing panel:
      1) Set the Domain to `XMLNSC`
5. Configure the MQOutput node:
   a. On the Basic panel, set the Queue name to `FILEREAD.TASK3.OUT1`
6. Deploy the message set and message flow.
7. Change the Directory and Name fields to the correct location of the file, and then put the following XML message onto queue FILEREAD.TASK2.IN1:

```
<Invoice>
  <Directory>c:\temp\FileRead</Directory>
  <Name>task2.xml</Name>
  <Data/>
</Invoice>
```

Results

The broker routes the message to the queue FILEREAD.TASK3.OUT1 and inserts data from the file into the Data field of the output message:

```
<Invoice>
  <Directory>c:\temp\FileRead</Directory>
  <Name>task2.xml</Name>
  <Data>Purchase details</Data>
</Invoice>
```

Related concepts:

"Routing or enriching a message based on the contents of a file" on page 1837
The FileRead node can route or enrich messages based on the contents of the file.

Related reference:

"FileRead node" on page 4444
Use the FileRead node to read one record, or the entire contents of a file, from within a message flow.

Record selection expressions:

The record selection expression is used to select a record from a file to propagate to the rest of the flow.

Each record in turn is compared to the expression, and the first one to evaluate to true is propagated to the **Out** terminal. You can set the expression to any valid XPath expression that returns a Boolean value. The expression is not used when `Whole File` is selected as the `Record Detection` option.

Correlation names used in the expression

You can use any of the following correlation names in the expression:

InputRoot and InputLocalEnvironment
The Input names refer to the incoming message that enters the node through the **In** terminal

ResultRoot
The ResultRoot name refers to the message created by using the current record in the file.

OutputRoot
The OutputRoot name refers to the message that is propagated if the expression evaluates to true. This action is identical to ResultRoot, unless the Output data location or Result data location have been changed to copy the Result message found in the file to a different location in the outgoing message.

OutputLocalEnvironment
The OutputLocalEnvironment contains the normal local environment which is propagated down the **Out** terminal and contains useful information like the number of the record and its offset.

Any combination of data in these correlation names can be used, along with any valid XPath expression, to determine whether to propagate the record.

Examples

Expression is:

```
$InputRoot/XMLNSC/Invoice/AccountNumber=$ResultRoot/XMLNSC/Data/Key
```

In this example, each record is a valid XML document. The FileRead node reads each record from the file. In the incoming message the FileRead node compares the field /Data/Key to the field /Invoice/AccountNumber. If the record matches, it is propagated to the **Out** terminal.

Expression is:

```
$OutputLocalEnvironment/File/Read/RecordNumber=5
```

The FileRead node reads each record from the file and compares the record number to 5. The record is propagated when it reaches the fifth record.

Building an outgoing message by using an incoming message combined with a record from a file

The FileRead node reads a record from a file and combines it with the message coming in to the node. By default, it replaces the message with the contents of the record read from the file. However, by using properties on the Result panel, you can choose how to combine the incoming message and file record contents. The node has three logical trees:

Input The Input message assembly contains all the data in the incoming message and is the basis for the propagated record.

Result The Result message assembly contains the record read from the file.

Output
The Output message assembly is the actual object propagated from the node.

By default, the Output message assembly is constructed by copying the Input message assembly to the Output message assembly. The data part of the Output message assembly is then replaced by the contents of the Result message assembly and the OutputLocalEnvironment is updated with details of what happened in the node.

The following Result panel properties can be used to modify this behavior:

Result data location
Specifies which part of the read record is copied to the Output message. By default, the Result data location copies everything from ResultRoot but it can be changed to copy only part of the record. For example: ResultRoot.XMLNSC.Invoice.Name just copies the name field from the selected record to the output message.

Output data location
Specifies where the record is copied to in the outgoing message. By default, the Output data location copies everything to OutputRoot. The location specified can be in the data part of the message (under ResultRoot) or in any other Output tree like OutputLocalEnvironment. For example: To copy the resulting record to a field in the message body OutputRoot.XMLNSC.Invoice.Data or to copy the result to local environment OutputRoot.Variables.Invoice.data.

Copy local environment
Causes the local environment to be copied from the InputLocalEnvironment. If the Copy local environment option is not selected, the InputLocalEnvironment is used directly without copying. This option allows nodes before the FileRead node to see changes to the Local environment.

For example, the following options copy the name field from a record to the output going message. The rest of the Output message is based on the input message:

```
Result data location= ResultRoot.XMLNSC.Invoice.Name
Output data location= OutputRoot.XMLNSC.Invoice.Name
```

The following options copy the message body from a record to the output going local environment. The Output message is the same as the input message:

```
Result data location= ResultRoot.XMLNSC.Invoice
Output data location= OutputLocalEnvironment.Variables.Invoice
```

Related concepts:

"Routing or enriching a message based on the contents of a file" on page 1837
The FileRead node can route or enrich messages based on the contents of the file.

Related tasks:

"Combining a WebSphere MQ message with an XML file using the contents of the message to identify which file to use" on page 1838
Combine an incoming message with the contents of an XML file, using fields in the message to determine which file to use.

Related reference:

"FileRead node" on page 4444
Use the FileRead node to read one record, or the entire contents of a file, from

within a message flow.

Reading a file on a remote FTP or SFTP directory:

Use a FileInput node to read a file in a directory on a remote FTP or SFTP server and then propagate messages that are based on the contents of that file.

Before you begin

Before you start:

This example is an extension of the example described in "Using a local file as input for your message flow" on page 1834 and it describes how to use a FileInput node in a message flow. The instructions assume that you are using a Windows operating system and that you have created a message flow containing a FileInput node. You also require the following resources:

- An FTP or SFTP server. Ensure that an FTP or SFTP server exists, with the following settings:

  Server *ftpserver.hursley.abc.com*

  Port *21* (for FTP) or *22* (for SFTP)

  Working directory
  /ftpfileinput

  Userid
  myuserid

  Password
  mypassword

  These values are for the purposes of this example only. If you use other values, record them so that you can set the appropriate values during the task.
- A security identity. Use the **mqsisetdbparms** command to define a security identity called *myidentity* for your user and password details.

  If you want to connect to an FTP server, the security identity must have an `ftp::` prefix, to enable the file nodes to find the identity definition. For example, use the following command for a broker called *MyBroker*:

  ```
  mqsisetdbparms MyBroker -n ftp::myidentity -u myuserid -p mypassword
  ```

  If you want to connect to an SFTP server, the security identity must have an `sftp::` prefix, as shown in the following example:

  ```
  mqsisetdbparms MyBroker -n sftp::myidentity -u myuserid -p mypassword
  ```

  You can also configure a connection to an SFTP server to use public key authentication, by specifying an SSH identity file and pass phrase, instead of a password. For example:

  ```
  mqsisetdbparms MyBroker -n sftp::myidentity -u myuserid -i identity_file -r passphrase
  ```

  For more information about configuring connections to an SFTP server, see "Transferring files securely by using SFTP" on page 1864.
- An input file. To follow this example scenario, create an input file called `test_input1.xml` with the following content:

  ```
  <Message>test1</Message>
  <Message>testtwo</Message>
  <Message>testthree</Message>
  ```

Each line ends with a line terminator that is suitable for the system on which the FTP or SFTP server is found. Do not put this file in the input directory but, instead, put it in the FTP or SFTP server directory `/ftpfileinput`.

- A message set. This example uses a message set called *xml1*, which uses the XMLNSC parser. Message set *xml1* models messages of the following form:

  ```
  <Message>...</Message>
  ```

About this task

Complete the following steps:

Procedure

1. Set the required node properties on the FileInput node. The following table summarizes the FileInput node properties that you must set, the tab on which they are displayed, whether they are mandatory, and the required values.

Tab	Property	Value
Basic	`Input directory`	`C:\FileInput\TestDir` If the input directory does not exist, no files are processed, even if you are processing files over FTP or SFTP.
	`File name or pattern`	`test_input1.xml`
	`Action on successful processing`	`Move to Archive Subdirectory`
	`Replace duplicate archive files`	Selected
Input Message Parsing	`Message domain`	`XMLNSC`
	`Message set`	`xml1`
Polling	`Polling interval`	3
Retry	`Action on failing file`	`Add Time Stamp and Move to Backout Subdirectory`
Records and Elements	`Record detection`	`Delimited`
	`Delimiter`	`DOS or UNIX Line End`
	`Delimiter type`	`Postfix`
FTP	`Remote transfer`	Selected
	`Transfer protocol`	`FTP or SFTP`
	`Remote server and port`	`ftpserver.hursley.abc.com`
	`Security identity`	`myidentity`
	`Server directory`	`/ftpfileinput`
	`Transfer mode`	`ASCII` (for FTP only)
	`Scan delay`	`45`

   If you used other values for your FTP or SFTP server resource, enter those values. The settings used here are identical to those used in the example in "Using a local file as input for your message flow" on page 1834, except that the `Remote transfer` property has been selected and there are now properties on the **FTP** tab. If you clear the `Remote transfer` property, the node operates as it does in the example in "Using a local file as input for your message flow" on page 1834; the properties on the **FTP** tab remain set but are ignored.
2. Deploy the message flow to the broker. See Chapter 11, "Packaging and deploying," on page 3209.

Results

The following actions occur when you perform these steps:

1. The file `test_input1.xml` is transferred from the FTP or SFTP server directory (`/ftpfileinput`) to the local directory (`C:\FileInput\TestDir`). The file is deleted from the FTP or SFTP server directory.
2. The FileInput node detects records that end with a DOS or UNIX line end and creates a message for each one that it finds, as defined by the properties on the **Records and elements** tab. The node propagates three messages to the message flow that is attached to the Out terminal:
   - Message 1:

     ```
     <Message>test1</Message>
     ```
   - Message 2:

     ```
     <Message>testtwo</Message>
     ```
   - Message 3:

     ```
     <Message>testthree</Message>
     ```
3. If a node is attached to the End of Data terminal, the End of Data message is propagated after the last record in the file has been processed.
4. When processing is complete, the file `test_input1.xml` is moved to the `mqsiarchive` subdirectory `C:\FileInput\TestDir\mqsiarchive`. If a file called `test_input1.xml` exists in the `mqsiarchive` subdirectory, it is overwritten.
5. If the message flow fails, retry processing is attempted according to the values set in the properties of the FileInput node. In this example task, a time stamp is added to the file name and the file is moved to the `mqsibackout` directory. Here is an example of the path to such a file: `C:\FileInput\TestDir\mqsibackout\20070928_150234_171021_test_input1.xml`.

   If an error occurs on the FTP side, stating that access is denied, a 0–byte file is created and moved to the mqsibackout directory. A 0–byte file is created in the mqsibackout directory for every FTP attempt that fails.

Because the `Remote transfer` property is selected, the FTP scan delay of 45 seconds overrides the polling interval of 3 seconds.

What to do next

For more information, see "Controlling how files are separated into records" on page 1848, which shows the effects of specifying other combinations of values in the `Record detection`, `Delimiter`, and `Delimiter type` properties of the FileInput node.

The following samples also show how to use this node:
- Batch Processing
- WildcardMatch

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Related concepts:

"How the broker processes files" on page 1814
The broker reads files with the FileInput, FTEInput, and FileRead nodes, and writes files with the FileOutput and FTEOutput nodes.

"Transferring files securely by using SFTP" on page 1864
You can transfer files securely by using the Secure File Transfer Protocol (SFTP), which enables file transfer by using the Secure Shell (SSH) protocol.

Related tasks:

"Using a local file as input for your message flow" on page 1834
Learn how to use the FileInput node to read a file on your local file system and then propagate messages that are based on the contents of that file.

"Writing a file to a remote FTP or SFTP server" on page 1855
Use a FileOutput node to write a file to a directory on a remote FTP or SFTP server.

Related reference:

"Recognizing file records as messages to be parsed" on page 1817
Use the FileInput, FTEInput and FileRead nodes to segment your input file into messages that are to be parsed.

"FileInput node" on page 4415
Use the FileInput node to process messages that are read from files.

"Controlling how files are separated into records" on page 1848
Set the `Record detection` and other properties on node **Records and Elements** tabs to read files in different formats.

"FtpServer configurable service properties" on page 3794
Select the properties and values that you want to change for an existing FtpServer configurable service or to create a new service.

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

Receiving a file by WebSphere MQ File Transfer Edition:

Use the FTEInput node to receive files from an existing WebSphere MQ File Transfer Edition network.

Before you begin

Before you start:

1. Read about WebSphere MQ File Transfer Edition.
2. Prepare the environment.

About this task

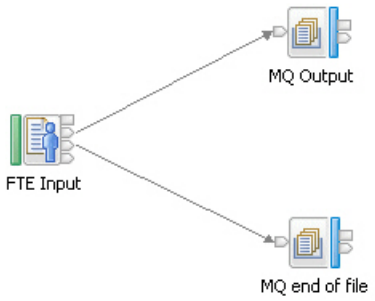
Complete the following steps to create an FTEInput node message flow using the default settings. The task includes some optional steps for additional configuration, but detailed information about configuring the FTEInput node is given on the property panels for the node, in the WebSphere Message Broker Toolkit.

Procedure

1. Drag an FTEInput node onto a message flow and wire its Out terminal to an output node of your choice.
2. Optional: To process only a defined subset of files sent to an agent, configure the Basic panel. This action allows multiple FTEInput nodes in the same execution group to receive specific files, depending on the directory or file filters specified.

   You can also specify whether, after processing, the file should be left in its directory, renamed, or deleted.

3. To change how the node handles a message flow failure, configure the Retry panel.
4. Optional: To change how records are identified in the input file, configure the `Record detection` property on the Records and Elements panel. For example, you might want to specify that a record is fixed length, and set the record length.
   a. If you set the `Record detection` property to anything other than `Whole File`, drag an additional output node to the flow, such as the MQOutput node. Wire the End of Data terminal on the FTEInput node to the In terminal of the MQOutput node, as shown in the following figure:



   The node connected to the End of Data terminal receives an empty message when the last record in the file is read.
5. Optional: To process the file based on details of the transfer, place a node such as the Route node after the FTEInput node. Details of the transfer are stored in the local environment, at `LocalEnvironment.FTE`.
6. Add the flow to a broker archive (BAR) file and deploy the BAR file.
7. Optional: Change the coordination queue manager; see "Preparing the environment for WebSphere MQ File Transfer Edition nodes" on page 740. The queue manager for the broker is the default coordination queue manager. The default might be adequate for testing; for production, consider changing it. A warning is written to the log if the coordination queue manager is not changed from the default.

Related concepts:

"Managed file transfers using WebSphere MQ File Transfer Edition" on page 1869
Transfer files, with file transfer metadata, in a timely and reliable manner.

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

"Local environment tree structure" on page 1056
The local environment tree is a part of the logical message tree in which you can store information while the message flow processes the message.

Related tasks:

"Sending a file by WebSphere MQ File Transfer Edition" on page 1859
Send files to an existing WebSphere MQ File Transfer Edition network.

"Preparing the environment for WebSphere MQ File Transfer Edition nodes" on page 740
Prepare the file system and queue managers, and determine the name of the broker agent.

Related reference:

"FTEInput node" on page 4461
Use the FTEInput node to receive files using the WebSphere MQ File Transfer Edition.

"FTEOutput node" on page 4466
Use the FTEOutput node to write messages to files by using the WebSphere MQ File Transfer Edition.

Receiving a file using IBM Sterling Connect:Direct:

Use the CDInput node to receive files from an IBM Sterling Connect:Direct network.

Before you begin

Before you start:

1. Read "IBM Sterling Connect:Direct overview and concepts" on page 1810.

About this task

MultipleCDInput nodes can be deployed to the same execution group, or to different execution groups in the same broker. Multiple CDInput nodes can read files transferred to the same directory without contention. Each file is processed only once, even if the nodes are deployed to separate execution groups, or different brokers.

On z/OS, when the CDInput node receives notification of the arrival of a dataset that it should process, the node copies that dataset into Unix System Services temporarily, before processing.

Complete the following steps to create a CDInput node message flow using the default settings. The task includes some optional steps for additional configuration, but detailed information about configuring the CDInput node is given on the property panels for the node, in the WebSphere Message Broker Toolkit. The default settings assume that you have a Connect:Direct server running on the same machine as the broker, and that you are using the default ports.

Procedure

1. Drag a CDInput node onto a message flow and wire its Out terminal to an output node of your choice.
2. Optional: To process only a defined subset of files, configure the Basic panel. This action allows multiple CDInput nodes in the same execution group to receive specific files, depending on the directory or file filters specified.

   You can also specify whether, after processing, the file is left in its directory, renamed, or deleted.
3. To change how the node handles a message flow failure, configure the Retry panel.
4. Optional: To change how records are identified in the input file, configure the `Record detection` property on the Records and Elements panel. For example, you might want to specify that a record is fixed length, and set the record length.
   a. If you set the `Record detection` property to anything other than `Whole File`, drag an additional output node to the flow, such as the MQOutput node. Wire the End of Data terminal on the CDInput node to the In terminal of

the MQOutput node. The node connected to the End of Data terminal receives an empty message when the last record in the file is read.

5. Optional: To process the file based on details of the transfer, place a node such as the Route node after the CDInput node. Details of the transfer are stored in the local environment, at `LocalEnvironment.CD`.
6. Setup the *username* and *password* that is required for the CDInput node to connect to the primary Connect:Direct server using the "**mqsisetdbparms** command" on page 3954.
7. Add the flow to a broker archive (BAR) file and deploy the BAR file.

   For information on how to use configurable services with IBM Sterling Connect:Direct, see "Advanced configuration properties when using IBM Sterling Connect:Direct nodes" on page 727.
8. Use IBM Sterling Connect:Direct to send a file to the local Connect:Direct server. The CDInput node processes this file.

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

"Local environment tree structure" on page 1056
The local environment tree is a part of the logical message tree in which you can store information while the message flow processes the message.

"IBM Sterling Connect:Direct overview and concepts" on page 1810
An overview of IBM Sterling Connect:Direct and its terminology when used with WebSphere Message Broker.

Related tasks:

"Initiating a managed file transfer using IBM Sterling Connect:Direct" on page 1873
Use a CDOutput node to send a file from a specified directory on your primary Connect:Direct server (PNODE) to a filename and directory on a secondary Connect:Direct server (SNODE).

"Advanced configuration properties when using IBM Sterling Connect:Direct nodes" on page 727
CDInput and CDOutput nodes can get connection details and staging directories in conjunction with a configurable service. To pick up new values when a configurable service is created or modified, you must reload the broker or execution group, by using the **mqsistop** and **mqsistart** commands, or the **mqsireload** command.

Related reference:

"CDInput node" on page 4305
Use the CDInput node to preview files when using IBM Sterling Connect:Direct in conjunction with WebSphere Message Broker.

"CDOutput node" on page 4312
Use the CDOutput node when using IBM Sterling Connect:Direct with WebSphere Message Broker.

Controlling how files are separated into records:

Set the `Record detection` and other properties on node **Records and Elements** tabs to read files in different formats.

The following examples are based on the examples described in "Using a local file as input for your message flow" on page 1834 and "Reading a file on a remote FTP

or SFTP directory" on page 1842. In each case, the input file to use, the property settings, and the expected results are described.

The examples, which describe using the FileInput node, can be applied to the FTEInput node, with the following provisos:

- The FTEInput node has no Basic tab.
- FTP and SFTP are not used to transmit files to an FTEInput node.

The examples can also be applied to the FileRead node, with the following provisos:

- The FileRead node propagates only one record from the file and, by default, this record is the first record in the file. The FileRead node can be configured to propagate a specific record. For more information see "Routing or enriching a message based on the contents of a file" on page 1837.
- FTP and SFTP are not used to transmit files to an FileRead node.

Example 1. Records are separated by a DOS or UNIX line end

This example is identical to the one described in "Using a local file as input for your message flow" on page 1834 or "Reading a file on a remote FTP or SFTP directory" on page 1842. Create an input file called `test_input1.xml` with the following content:

```
<Message>test1</Message>
<Message>testtwo</Message>
<Message>testthree</Message>
```

Each line ends with a line terminator.

The properties to set are:

Tab	Property	Value
Records and Elements	`Record detection`	`Delimited`
	`Delimiter`	`DOS or UNIX Line End`
	`Delimiter type`	`Postfix`

The FileInput node detects records that end with a DOS or UNIX line end and creates a message for each one that it finds.

The result is the propagation of three messages, as follows:

- Message 1:

  ```
  <Message>test1</Message>
  ```
- Message 2:

  ```
  <Message>testtwo</Message>
  ```
- Message 3:

  ```
  <Message>testthree</Message>
  ```

The DOS or UNIX line end is not part of any propagated message.

Example 2. Records are separated by a custom delimiter

Create an input file called `test_input2.xml` with the following content:

```
<Message>test01</Message>,<Message>test001</Message>,<Message>test0001</Message>
```

There must be no line terminator at the end of this file data; the XMLNSC parser ignores the line terminator if it is present.

In addition to the property settings described in "Using a local file as input for your message flow" on page 1834 or "Reading a file on a remote FTP or SFTP directory" on page 1842, set these properties:

Tab	Property	Value
Basic	File name or pattern	test_input2.xml
Records and Elements	Record detection	Delimited
	Delimiter	Custom Delimiter
	Custom delimiter	2C
	Delimiter type	Infix

The hexadecimal X'2C' represents a comma in ASCII. On other systems, you must use a different hexadecimal code.

The FileInput node detects the comma character and uses it to separate records. Because the value of the Delimiter type property is Infix, a comma is not required at the end of the file.

The result is the propagation of three messages, as follows:

- Message 1:
  <Message>test01</Message>
- Message 2:
  <Message>test001</Message>
- Message 3:
  <Message>test0001</Message>

The comma character is not part of any propagated message. There are no commas in the bodies of the message in this example; if the message bodies did contain commas, the records would be split at those points resulting in incorrectly formed messages being propagated to the rest of the flow.

Example 3. Records are separated by a fixed number of bytes

Create an input file called test_input3.xml with the following content:

```
<Message>123456789</Message><Message>abcdefghi</Message><Message>rstuvwxyz</Message>
```

There must be no line terminator at the end of this file.

In addition to the property settings described in "Using a local file as input for your message flow" on page 1834 or "Reading a file on a remote FTP or SFTP directory" on page 1842, set these properties:

Tab	Property	Value
Basic	File name or pattern	test_input3.xml
Records and Elements	Record detection	Fixed Length
	Length	28

The FileInput node splits the input file into records each 28 bytes long.

The result is the propagation of three messages, as follows:

- Message 1:

  ```
  <Message>123456789</Message>
  ```

- Message 2:

  ```
  <Message>abcdefghi</Message>
  ```

- Message 3:

  ```
  <Message>rstuvwxyz</Message>
  ```

Each message is 28 bytes long. If the file contains trailing bytes, for example a carriage return-line feed pair, a further message containing these bytes is propagated; the trailing bytes might or might not be recognized by the message domain, message set, and message type assigned to parse the message.

Example 4. Records are whole files

Create an input file called `test_input4.xml` with the following content:

```
<Message>Text string of a length decided by you, even including line
terminators, as long as it only contains this tag at the end.</Message>
```

There must be no line terminator at the end of this file; if there is one, it has no effect.

In addition to the property settings described in "Using a local file as input for your message flow" on page 1834 or "Reading a file on a remote FTP or SFTP directory" on page 1842, set these properties:

Tab	Property	Value
Basic	`File name or pattern`	`test_input4.xml`
Records and Elements	`Record detection`	`Whole File`

The FileInput node does not split the file; it propagates all of the content of the file as a single record to be parsed by the message domain, message set, and message type as specified on the node. In this example, you are using the XMLNSC parser and message set `xml1`, so the message is recognized.

The result is the propagation of one message, as follows:

- Message 1:

```
<Message>Text string of a length decided by you, even including line terminators, as long as
it only contains this tag at the end.</Message>
```

Trailing bytes (for example, line terminators) are included.

Example 5. Records are recognized as separate messages by the parser specified in the `Message domain` property

Create an input file called `test_input5.xml` with the following content:

```
<Message>Text string of a length decided by you  </Message><Message>and another</Message>
<Message>and another on a new line</Message>
```

Line terminators at the end of this file, or at the end of lines, are acceptable.

In addition to the property settings described in "Using a local file as input for your message flow" on page 1834 or "Reading a file on a remote FTP or SFTP directory" on page 1842, set these properties:

Table 13.

Tab	Property	Value
Basic	`File name or pattern`	`test_input5.xml`
Records and Elements	`Record detection`	`Parsed Record Sequence`

The FileInput node defers to the parser to determine the record boundaries. In this example, message set `xml1` in domain XMLNSC must recognize the complete `<Message>` XML format. XMLNSC absorbs trailing white space (for example, line terminators).

The result is the propagation of three messages, as follows:

- Message 1:

```
<Message>Text string of a length decided by you  </Message>
```

- Message 2:

```
<Message>and another</Message>
```

- Message 3:

```
<Message>and another on a new line</Message>
```

Trailing white space (for example, line terminators) are included in the messages.

Related concepts:

"How the broker processes files" on page 1814
The broker reads files with the FileInput, FTEInput, and FileRead nodes, and writes files with the FileOutput and FTEOutput nodes.

Related tasks:

"Using a local file as input for your message flow" on page 1834
Learn how to use the FileInput node to read a file on your local file system and then propagate messages that are based on the contents of that file.

"Reading a file on a remote FTP or SFTP directory" on page 1842
Use a FileInput node to read a file in a directory on a remote FTP or SFTP server and then propagate messages that are based on the contents of that file.

Related reference:

"Recognizing file records as messages to be parsed" on page 1817
Use the FileInput, FTEInput and FileRead nodes to segment your input file into messages that are to be parsed.

"FileInput node" on page 4415
Use the FileInput node to process messages that are read from files.

"FTEInput node" on page 4461
Use the FTEInput node to receive files using the WebSphere MQ File Transfer Edition.

Writing a file

Use the FileOutput, CDOutput, and FTEOutput nodes to write files.

About this task

This section contains the following topics:

- "Setting the `Record definition` property for the FileOutput and FTEOutput nodes" on page 1861
- "Initiating a managed file transfer using IBM Sterling Connect:Direct" on page 1873

Related concepts:

"How the broker processes files" on page 1814
The broker reads files with the FileInput, FTEInput, and FileRead nodes, and writes files with the FileOutput and FTEOutput nodes.

Related reference:

"CDOutput node" on page 4312
Use the CDOutput node when using IBM Sterling Connect:Direct with WebSphere Message Broker.

"FileOutput node" on page 4430
Use the FileOutput node to write messages to files.

Writing a file to your local file system:

Use a FileOutput node to write a file to a specified directory on your local file system.

Before you begin

This example shows you how one combination of values in the `Record definition`, `Delimiter`, and `Delimiter type` properties result in the creation of a file from multiple messages. The example describes the FileOutput node of a message flow and assumes that the rest of the flow has been developed. It is also assumed that a Windows system is being used. To complete this example task, you must first have added a FileOutput node to a message flow. You must also ensure that the following messages are produced by the flow preceding the FileOutput node:

- Three input messages, which are sent, in this order, to the In terminal of the FileOutput node:
  - Message 1:

    ```
    <Message>test1</Message>
    ```
  - Message 2:

    ```
    <Message>testtwo</Message>
    ```
  - Message 3:

    ```
    <Message>testthree</Message>
    ```

  These messages can be produced, for example, by the XMLNSC domain with a message set which recognizes XML with the following form:

  ```
  <Message>...</Message>
  ```
- A final message, which is sent to the Finish File terminal of the FileOutput node after the first three messages have been sent:

  ```
  <thiscanbe>anything</thiscanbe>
  ```

Complete the following steps:

Procedure

1. Set the required node properties on the FileOutput node. The following table summarizes the FileOutput node properties that you must set, on which tab they appear, and the value that you must set in order to follow this example:

Tab	Property	Value
Basic	Directory	C:\FileOutput\TestDir
	File name or pattern	test_output1.xml
	Output file action	Time Stamp, Archive and Replace Existing File (or Create if File does not Exist)
	Replace duplicate archive files	Selected
Records and Elements	Record definition	Record is Delimited Data
	Delimiter	Broker System Line End
	Delimiter type	Postfix
FTP	FTP	Cleared

2. Deploy the message flow to the broker. See Chapter 11, "Packaging and deploying," on page 3209.
3. Send the first three messages to the In terminal of the FileOutput node.
4. Send the final message to the Finish File terminal of the FileOutput node.

Results

The following actions occur when you perform these steps:

1. The file is processed. In accordance with the values set in the properties of the FileOutput node, the node generates one record per message with a local file system line terminator after each one. The file contains the following data, each line terminated by a carriage return (X'0D') and line feed (X'0A') pair of characters (on a Windows system):

   ```
   <Message>test1</Message>
   <Message>testtwo</Message>
   <Message>testthree</Message>
   ```

2. Records are accumulated in file test_output1.xml in the C:\FileOutput\TestDir\mqsitransit directory. When the final message is sent to the Finish File terminal, the file is moved to the output directory, C:\FileOutput\TestDir directory.
3. If a file of the same name exists in the output directory, the existing file is renamed and moved to the mqsiarchive directory. For example, the following file might be created:

   ```
   C:\FileOutput\TestDir\mqsiarchive\20081124_155346_312030_test_output1.xml
   ```

   If a file of this name exists in this archive directory, it is overwritten in accordance with the Replace duplicate archive files property selected on the FileOutput node.

What to do next

See "Setting the Record definition property for the FileOutput and FTEOutput nodes" on page 1861 to see the results of running this task with different values set in the Record definition, Delimiter, and Delimiter type properties of the FileOutput node.

The following samples also show how to use this node:

- File Output
- Batch Processing
- WildcardMatch

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Related concepts:

"How the broker processes files" on page 1814
The broker reads files with the FileInput, FTEInput, and FileRead nodes, and writes files with the FileOutput and FTEOutput nodes.

"IBM Sterling Connect:Direct overview and concepts" on page 1810
An overview of IBM Sterling Connect:Direct and its terminology when used with WebSphere Message Broker.

Related tasks:

"Using a local file as input for your message flow" on page 1834
Learn how to use the FileInput node to read a file on your local file system and then propagate messages that are based on the contents of that file.

"Writing a file to a remote FTP or SFTP server"
Use a FileOutput node to write a file to a directory on a remote FTP or SFTP server.

Related reference:

"FileOutput node" on page 4430
Use the FileOutput node to write messages to files.

"Setting the `Record definition` property for the FileOutput and FTEOutput nodes" on page 1861
Set the properties on the**Records and Elements** tab of the node to write files in different formats.

"CDOutput node" on page 4312
Use the CDOutput node when using IBM Sterling Connect:Direct with WebSphere Message Broker.

Writing a file to a remote FTP or SFTP server:

Use a FileOutput node to write a file to a directory on a remote FTP or SFTP server.

Before you begin

This example shows you how one combination of values in the `Record definition`, `Delimiter`, and `Delimiter type` properties result in the creation of a file from multiple messages. The example is an extension of the example described in "Writing a file to your local file system" on page 1853, and describes the use of a FileOutput node in a message flow.

These instructions assume that you are using a Windows system, and that you have already created a message flow containing a FileOutput node. You also require the following resources:

- An FTP or SFTP server. Ensure that an FTP or SFTP server exists with the following settings, so that you can follow this example scenario:

Server *ftpserver.hursley.abc.com*

Port *21* (for FTP) or *22* (for SFTP)

Working directory
/ftpfileoutput

User ID
myuserid

Password
mypassword

These values are for the purposes of this example only. If you use other values, record them so that you can set the appropriate values when you follow the instructions in this task.

- A security identity. Use the **mqsisetdbparms** command to define a security identity called *myidentity* for your user and password details.

  If you want to connect to an FTP server, the security identity must have an `ftp::` prefix, to enable the file nodes to find the identity definition. For example, use the following command for a broker called *MyBroker*:

  ```
  mqsisetdbparms MyBroker -n ftp::myidentity -u myuserid -p mypassword
  ```

  If you want to connect to an SFTP server, the security identity must have an `sftp::` prefix, as shown in the following example:

  ```
  mqsisetdbparms MyBroker -n sftp::myidentity -u myuserid -p mypassword
  ```

  You can also configure a connection to an SFTP server to use Public Key authentication, by specifying an SSH identity file and pass phrase, instead of a password. For example:

  ```
  mqsisetdbparms MyBroker -n sftp::myidentity -u myuserid -i identity_file -r passphrase
  ```

  For more information about configuring connections to an SFTP server, see "Transferring files securely by using SFTP" on page 1864.

- The following messages, which must be produced by the message flow preceding the FileOutput node:
  - Three input messages. These messages are sent, in this order, to the In terminal of the FileOutput node:
    - Message 1:

      ```
      <Message>test1</Message>
      ```

    - Message 2:

      ```
      <Message>testtwo</Message>
      ```

    - Message 3:

      ```
      <Message>testthree</Message>
      ```

    These messages can be produced, for example, by the XMLNSC domain with a message set that recognizes XML, in the following form:

    ```
    <Message>...</Message>
    ```

  - A final message sent to the Finish File terminal of the FileOutput node after the first three messages have been sent:

    ```
    <thiscanbe>anything</thiscanbe>
    ```

Complete the following steps:

Procedure

1. Set the required node properties on the FileOutput node. The following table summarizes the FileOutput node properties that you must set, the tabs on which they are displayed, and the values that are used in this example:

Tab	Property	Value
Basic	Directory	C:\FileOutput\TestDir
	File name or pattern	test_output1.xml
	Output file action	Time Stamp, Archive and Replace Existing File (or Create if File does not Exist)
	Replace duplicate archive files	Selected
Records and Elements	Record definition	Record is Delimited Data
	Delimiter	Broker System Line End
	Delimiter type	Postfix
FTP	Remote transfer	Selected
	Transfer protocol	FTP or SFTP
	Remote server and port	ftpserver.hursley.abc.com
	Security identity	myidentity
	Server directory	/ftpfileoutput
	Transfer mode	ASCII (for FTP only)
	Retain local file after transfer	Selected

   If you used other values for your FTP or SFTP server resource, use those values. The settings used here are identical to those settings used in the example in "Writing a file to your local file system" on page 1853 except that the Remote transfer property has been selected and there are now properties on the **FTP** tab. If you clear the Remote transfer property, the node operates as it does in the example in "Writing a file to your local file system" on page 1853; the properties on the **FTP** tab remain set but are ignored.

   You can override the Remote server and port property on the node by setting a value in the local environment. For more information, see "Local environment overrides for the FileOutput node" on page 4443.

2. Deploy the message flow to the broker. See Chapter 11, "Packaging and deploying," on page 3209.
3. Send the first three messages to the In terminal of the FileOutput node.
4. Send the final message to the Finish File terminal of the FileOutput node.

Results

The following actions occur when you perform these steps:

1. The file is processed. The FileOutput node generates one record per message with a local file system line terminator after each one. The file contains the following data, with each line terminated by a carriage return (X'0D') and line feed (X'0A') pair of characters (on a Windows system):

```
<Message>test1</Message>
<Message>testtwo</Message>
<Message>testthree</Message>
```

2. Records are accumulated in file `test_output1.xml` in the `C:\FileOutput\TestDir\mqsitransit` directory. When the final message is sent to the Finish File terminal, the file is moved to the remote FTP or SFTP server directory (because the `Remote transfer` property is selected). As a result, the file `/ftpfileoutput/test_output1.xml` is created.
3. If a file with the same name exists in the remote FTP or SFTP server directory, the existing file is overwritten.

   If the remote FTP server is not running on a Windows system and the `Transfer mode` property is set to `ASCII`, the character encoding and line terminator characters might be modified after transfer. For example, on a z/OS FTP server, the ASCII text is typically converted to EBCDIC, and the line terminator character pairs are replaced by EBCDIC new line characters (X'15'). Other FTP servers might treat ASCII transfers differently. If you are using SFTP, the `Transfer mode` property is ignored and files are sent as Binary files.
4. Because the `Retain local file after transfer` property is selected, the local file is not deleted but is moved from the `mqsitransit` subdirectory to the output directory, `C:\FileOutput\TestDir`. If a file with the same name exists in the output directory, the existing file is renamed and moved to the `mqsiarchive` directory. For example, the following file might be created:

   ```
   C:\FileOutput\TestDir\mqsiarchive\20081124_155346_312030_test_output1.xml
   ```

   However, if a file with this name exists in this archive directory, it is overwritten according to the value of the `Replace duplicate archive files` property set on the FileOutput node.

What to do next

For more information, see “Setting the `Record definition` property for the FileOutput and FTEOutput nodes” on page 1861, which shows the results of running this task with different values set in the `Record definition`, `Delimiter`, and `Delimiter type` properties of the FileOutput node.

The following samples also show how to use this node:

- File Output
- Batch Processing
- WildcardMatch

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Related concepts:

“How the broker processes files” on page 1814
The broker reads files with the FileInput, FTEInput, and FileRead nodes, and writes files with the FileOutput and FTEOutput nodes.

“Transferring files securely by using SFTP” on page 1864
You can transfer files securely by using the Secure File Transfer Protocol (SFTP), which enables file transfer by using the Secure Shell (SSH) protocol.

Related tasks:

"Reading a file on a remote FTP or SFTP directory" on page 1842
Use a FileInput node to read a file in a directory on a remote FTP or SFTP server and then propagate messages that are based on the contents of that file.

"Writing a file to your local file system" on page 1853
Use a FileOutput node to write a file to a specified directory on your local file system.

Related reference:

"FileOutput node" on page 4430
Use the FileOutput node to write messages to files.

"Setting the `Record definition` property for the FileOutput and FTEOutput nodes" on page 1861
Set the properties on the**Records and Elements** tab of the node to write files in different formats.

"FtpServer configurable service properties" on page 3794
Select the properties and values that you want to change for an existing FtpServer configurable service or to create a new service.

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

Sending a file by WebSphere MQ File Transfer Edition:

Send files to an existing WebSphere MQ File Transfer Edition network.

Before you begin

Before you start:

1. Read about WebSphere MQ File Transfer Edition.
2. Get the following information from your WebSphere MQ File Transfer Edition administrator:
   - The name of the remote WebSphere MQ File Transfer Edition agent to which the file is to be sent.
   - The name of the destination queue manager.
   - The name of the output file.
3. Create a message flow that contains an input node.

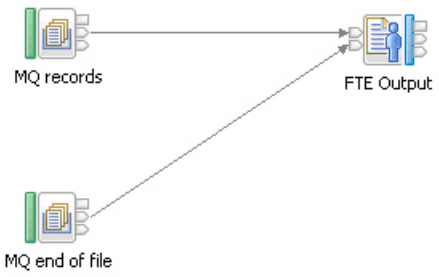
About this task

Detailed information about configuring the FTEOutput node is given on the property panels for the node, in the WebSphere Message Broker Toolkit.

Procedure

1. Drag an FTEOutput node onto the message flow, and wire its In terminal to the input node.
2. On the Basic panel, set values for the `Destination agent` and `Destination file name` properties. Configuring just these two properties is enough if you want to send all the input message tree as a single record in the output file.
3. Optional: On the Basic panel, set a value for the `Destination queue manager` property. The default destination queue manager is the queue manager for the broker.
4. Optional: To specify a location in the input message tree for the data to be sent, configure the `Data location` property on the "Request properties" panel.

5. Optional: To change how records are placed in the output file, configure the `Record definition` property on the "Record definition" panel. For example, you might want to specify that a record is fixed length, and set the record length.
   a. If you set the `Record definition` property to anything other than `Record is Whole File`, drag a node such as the MQOutput node to the flow, and wire its Out terminal to the Finish File terminal on the FTEOutput node, as shown in the following figure:



   The node connected to the Finish File terminal must have logic to determine the last record in the file.
6. Optional: To set properties for the transfer dynamically, place a node such as the Compute node or Mapping node before the FTEOutput node. You can override the following properties:
   - `Destination agent`
   - `Destination file directory`
   - `Destination file name`
   - `Destination queue manager`
   - `Job name`
   - `Overwrite files on destination`

   Configure the node to write the overrides to the `LocalEnvironment.Destination.FTE` subtree.
7. Add the flow to a broker archive (BAR) file and deploy the BAR file.
8. Optional: Change the coordination queue manager; see “Preparing the environment for WebSphere MQ File Transfer Edition nodes” on page 740. The queue manager for the broker is the default coordination queue manager. The default might be adequate for testing; for production, consider changing it. A warning is written to the log if the coordination queue manager is not changed from the default.

Related concepts:

“Managed file transfers using WebSphere MQ File Transfer Edition” on page 1869
Transfer files, with file transfer metadata, in a timely and reliable manner.

“WebSphere Message Broker Explorer” on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

Related tasks:

“Preparing the environment for WebSphere MQ File Transfer Edition nodes” on page 740
Prepare the file system and queue managers, and determine the name of the broker agent.

Related reference:

“FTEOutput node” on page 4466
Use the FTEOutput node to write messages to files by using the WebSphere MQ File Transfer Edition.

Setting the `Record definition` property for the FileOutput and FTEOutput nodes:

Set the properties on the**Records and Elements** tab of the node to write files in different formats.

The following examples are based on those described in “Writing a file to your local file system” on page 1853 and “Writing a file to a remote FTP or SFTP server” on page 1855. The FTEOutput does not archive, and files are transferred using WebSphere MQ File Transfer Edition, rather than FTP or SFTP. In all examples, it is assumed that the same messages are sent to the FileOutput node; three to the In terminal and one to the Finish File terminal:

- Three input messages which are sent, in this order, to the In terminal of the FileOutput node:
  - Message 1:

    ```
    <Message>test1</Message>
    ```
  - Message 2:

    ```
    <Message>testtwo</Message>
    ```
  - Message 3:

    ```
    <Message>testthree</Message>
    ```

  These messages can be produced, for example, by the XMLNSC domain with a message set which recognizes XML with the following form:

  ```
  <Message>...</Message>
  ```
- A final message which is sent to the Finish File terminal of the FileOutput node after the first three messages have been sent. It does not matter what this message contains.

The following examples describe the contents of the file or files produced; the disposition of the files created is as in the “Writing a file to your local file system” on page 1853 and “Writing a file to a remote FTP or SFTP server” on page 1855 topics.

Example 1. Records written are separated by a DOS or UNIX line end

This example is identical to the one described in “Writing a file to your local file system” on page 1853 or “Writing a file to a remote FTP or SFTP server” on page 1855. Specify the node's properties as described in “Writing a file to your local file system” on page 1853 or “Writing a file to a remote FTP or SFTP server” on page 1855.

These properties result in one file being written. The file contains three records each terminated by a local system line terminator. On a Windows system, this is a carriage return (X'0D') line feed (X'0A') pair of characters; on UNIX systems it is X'0A'.

```
<Message>test1</Message>
<Message>testtwo</Message>
<Message>testthree</Message>
```

Example 2. Records written are separated by a custom delimiter

In addition to the property settings described in "Writing a file to your local file system" on page 1853 or "Writing a file to a remote FTP or SFTP server" on page 1855, set these properties on the **Records and Elements** tab:

Property	Value
Record definition	Record is Delimited Data
Delimiter	Custom Delimiter
Custom delimiter	0D0A
Delimiter type	Postfix

The hexadecimal X'0D0A' represents a carriage return character followed by a line feed character. On a Windows system, this results in a file identical to the one created in Example 1. On other systems, the result might differ from the result in Example 1; Example 1 uses local system line end characters, whereas Example 2 always puts the X'0D0A' sequence at the end of each line.

Example 3. Records written are padded to a fixed length

In addition to the property settings described in "Writing a file to your local file system" on page 1853 or "Writing a file to a remote FTP or SFTP server" on page 1855, set these properties on the **Records and Elements** tab:

Property	Value
Record definition	Record is Fixed Length Data
Length (bytes)	30
Padding bytes (hexadecimal)	2A

The hexadecimal character X'2A' represents an asterisk character in ASCII.

The length of each incoming message is 24 bytes, 26 bytes, and 28 bytes respectively. The required fixed length of each record is 30 bytes. Each record is therefore padded by an extra 6 bytes, 4 bytes, and 2 bytes respectively, using the hexadecimal character X'2A'.

One file is written. It contains a single line:

```
<Message>test1</Message>******<Message>testtwo</Message>****<Message>testthree</Message>**
```

Example 4. Records written are not separated by delimiters or padding

In addition to the property settings described in "Writing a file to your local file system" on page 1853 or "Writing a file to a remote FTP or SFTP server" on page 1855, set this property on the **Records and Elements** tab:

Property	Value
Record definition	Record is Unmodified Data

The records are concatenated with no padding or delimiters.

One file is written with the following content:

```
<Message>test1</Message><Message>testtwo</Message><Message>testthree</Message>
```

There are no trailing bytes or line terminators.

Example 5. Records are written as whole files

In addition to the property settings described in "Writing a file to your local file system" on page 1853 or "Writing a file to a remote FTP or SFTP server" on page 1855, set this property on the **Records and Elements** tab:

Property	Value
Record definition	Record is Whole File

Three files are created, each containing one record:

- File 1:

  ```
  <Message>test1</Message>
  ```

- File 2:

  ```
  <Message>testtwo</Message>
  ```

- File 3:

  ```
  <Message>testthree</Message>
  ```

Each of these files is created with the same name, one by one, in the mqsitransit directory. If you are following the example in "Writing a file to a remote FTP or SFTP server" on page 1855, each file is transferred to the remote FTP server. However, because each file overwrites the previous one, only the third file remains when the task is complete.

After optional transfer, if a copy is retained, each file is moved to the output directory, `C:\FileOutput\TestDir`. In accordance with the properties on the FileOutput node as described in "Writing a file to your local file system" on page 1853 or "Writing a file to a remote FTP or SFTP server" on page 1855, the second file moved displaces the first file from the output directory which is moved to the mqsiarchive subdirectory with a time stamp added to the file name. When the third file is moved to the output directory, it displaces the second file, causing it to be moved to the mqsiarchive subdirectory and renamed. The final result is files similar to these:

```
C:\FileOutput\TestDir\mqsiarchive\20071101_165346_312030_test_output1.xml
C:\FileOutput\TestDir\mqsiarchive\20071101_165347_312030_test_output1.xml
C:\FileOutput\TestDir\test_output1.xml
```

being File 1, File 2, and File 3 respectively. If FTP processing was enabled, File 3 would also be in the remote FTP server directory and called `test_output1.xml`.

Related concepts:

"How the broker processes files" on page 1814
The broker reads files with the FileInput, FTEInput, and FileRead nodes, and writes files with the FileOutput and FTEOutput nodes.

Related tasks:

"Writing a file to your local file system" on page 1853
Use a FileOutput node to write a file to a specified directory on your local file system.

"Writing a file to a remote FTP or SFTP server" on page 1855
Use a FileOutput node to write a file to a directory on a remote FTP or SFTP server.

Related reference:

"FileOutput node" on page 4430
Use the FileOutput node to write messages to files.

Transferring files securely by using SFTP

You can transfer files securely by using the Secure File Transfer Protocol (SFTP), which enables file transfer by using the Secure Shell (SSH) protocol.

Use the properties on the FTP tab of the FileInput and FileOutput nodes to configure the secure transfer of files.

You can also use the FtpServer configurable service to configure other SFTP properties, including the cipher to be used for SFTP communication, compression level, and strict known host checking.

See the following topics for more information about configuring secure file transfer:

- "Configuring SFTP file transfer"
- "Known host checking" on page 1866

Related concepts:

"How the broker processes files" on page 1814
The broker reads files with the FileInput, FTEInput, and FileRead nodes, and writes files with the FileOutput and FTEOutput nodes.

Related tasks:

"Connecting client applications" on page 1537
Connect your client applications to the broker by using one or more of the supported protocols from other resources and software servers in your network.

"Reading a file on a remote FTP or SFTP directory" on page 1842
Use a FileInput node to read a file in a directory on a remote FTP or SFTP server and then propagate messages that are based on the contents of that file.

"Writing a file to a remote FTP or SFTP server" on page 1855
Use a FileOutput node to write a file to a directory on a remote FTP or SFTP server.

Related reference:

"FileInput node" on page 4415
Use the FileInput node to process messages that are read from files.

"FileOutput node" on page 4430
Use the FileOutput node to write messages to files.

Configuring SFTP file transfer:

Use an FtpServer configurable service to specify the Secure File Transfer Protocol (SFTP) settings for a message flow, and to override the SFTP settings that are specified on the FileInput and FileOutput nodes.

About this task

The settings that you specify by using an FtpServer configurable service are read and validated when the message flow starts, and are used to configure any SFTP connections that are made for the node. The configurable service can override any or all of the remote transfer properties on the **FTP** tab of the FileInput and FileOutput nodes. For more information about the settings that you can specify with an FtpServer configurable service, see "FtpServer configurable service properties" on page 3794.

You can configure strict host key checking and specify your own known hosts file, or you can turn off strict host key checking and use the known hosts files that are created and managed by the broker.

Multiple configurable services can specify the same host and port, even with different known hosts files. FTP defaults to port 21 and SFTP defaults to port 22, which is the SSH default port. If you set the port and specify an FTP connection to an SFTP server (or specify an SFTP connection to an FTP server) a connection error occurs and a message is added to the event log.

You can use the FtpServer configurable service to configure the following SFTP settings:

- Cipher used for SSH/SFTP communication
- Compression level
- Strict known host checking
- Protocol (FTP/SFTP) for nodes to use for remote file transfer
- Location of a known hosts file when strict known host checking is set to Yes

Procedure

1. Use the WebSphere Message Broker Explorer; see "Using the WebSphere Message Broker Explorer to work with configurable services" on page 644 for more information, or the **mqsicreateconfigurableservice** command to create an FtpServer configurable service with the required parameter values. For more information about creating the FtpServer configurable service, see "FtpServer configurable service properties" on page 3794.
2. In the FileInput and FileOutput nodes, specify the name of the FtpServer configurable service in the **Remote server and port** property on the **FTP** tab. Use the WebSphere Message Broker Explorer, or the **mqsichangeproperties** and **mqsireportproperties** commands to change or view the properties of the configurable service.

Related concepts:

"Transferring files securely by using SFTP" on page 1864
You can transfer files securely by using the Secure File Transfer Protocol (SFTP), which enables file transfer by using the Secure Shell (SSH) protocol.

"Known host checking" on page 1866
Use known host checking to control which hosts the broker can connect to, and to verify the identity of those hosts.

"How the broker processes files" on page 1814
The broker reads files with the FileInput, FTEInput, and FileRead nodes, and writes files with the FileOutput and FTEOutput nodes.

"Using the WebSphere Message Broker Explorer to work with configurable services" on page 644
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view, add, modify and delete configurable services.

Related tasks:

"Reading a file on a remote FTP or SFTP directory" on page 1842
Use a FileInput node to read a file in a directory on a remote FTP or SFTP server and then propagate messages that are based on the contents of that file.

"Writing a file to a remote FTP or SFTP server" on page 1855
Use a FileOutput node to write a file to a directory on a remote FTP or SFTP server.

Related reference:

"FileOutput node" on page 4430
Use the FileOutput node to write messages to files.

"Setting the `Record definition` property for the FileOutput and FTEOutput nodes" on page 1861
Set the properties on the**Records and Elements** tab of the node to write files in different formats.

"FtpServer configurable service properties" on page 3794
Select the properties and values that you want to change for an existing FtpServer configurable service or to create a new service.

"FileInput node" on page 4415
Use the FileInput node to process messages that are read from files.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

Known host checking:

Use known host checking to control which hosts the broker can connect to, and to verify the identity of those hosts.

Known host checking enables the broker to protect the messages in the message flow from unauthorized attempts to intercept the data (sometimes known as *man-in-the-middle attacks*). Known hosts files contain the SSH keys of the hosts to which the broker can connect. On z/OS systems, known hosts files and SSH identity files are stored in EBCDIC format, and on other operating systems they are stored in ASCII format.

Before it connects to a host to receive or transfer a file (using the FileInput or FileOutput nodes), the broker checks the host key against the keys that are stored in the known hosts file. If the host key does not match an existing entry for the host in the known hosts file, the connection fails. If the host is new (and has no entry in the known hosts file), the result depends on whether strict known host checking is turned on or off.

You can specify whether strict known host checking is turned on or off by setting the **strictHostKeyChecking** parameter of either the **mqsicreateconfigurableservice** command or the FtpServer configurable service. For more information about the settings that you can specify with an FtpServer configurable service, see "FtpServer configurable service properties" on page 3794.

Strict known host checking

When strict known host checking is turned on (by setting the **strictHostKeyChecking** parameter to `Yes`), the broker connects only to known hosts with valid SSH host keys that are stored in the known hosts file. If you use strict known host checking, you must create your own known hosts file, in OpenSSH format, containing the SSH keys of your trusted hosts. You specify the location of

your known hosts file using the **knownHostsFile** parameter of the **mqsicreateconfigurableservice** command. When the broker attempts to make a connection to a host, it checks the host key against the contents of the known hosts file. If the key is not in the known hosts file, the connection fails and a BIP3371 error occurs.

You can have multiple known hosts files and specify a different one for each node or configurable service. The known hosts files that you provide for strict known host checking are not modified by the broker.

Non-strict known host checking

When strict known host checking is turned off (by setting the **strictHostKeyChecking** parameter to `No`) the broker connects only to known hosts with valid keys or to new hosts to which it has not connected before. If the broker attempts to connect to a new host (to which it has not connected previously), the broker makes the connection, accepts the host's SSH key, and stores it in the known hosts file. When the broker tries to connect to the same host on subsequent occasions, the host key is checked against the key stored in the known hosts file, and, if the key matches, the connection is made. However, if the host key is different from the one stored in the known hosts file, the connection attempt fails and a BIP3371 error occurs.

When strict known host checking is turned off, the known hosts file is managed by the broker. One broker-managed known hosts file exists for each execution group.

BIP3371 error message

The BIP3371 error message might indicate that there has been an unauthorized attempt to intercept a message. However, the connection failure (and resulting BIP3371 error message) might be caused by a change to the host's SSH key at some time following its first connection to the broker. For example, if the SSH server is reinstalled, it is assigned a new key, which is not found in the known hosts file and is therefore not accepted by the broker. As a result, the connection fails and the BIP3371 error message is shown.

If you know that the SSH host key has changed (as a result of a recent SSH server reinstallation, for example), you can solve the connection failure by modifying the known hosts file:

1. Stop the message flow.
2. Edit the known hosts file:
   - If you have strict known host checking turned on, correct the entry for the host in the known hosts file that you specified in the **knownHostsFile** parameter of the **mqsicreateconfigurableservice** command.
   - If you have strict known host checking turned off, remove the host's entry from the broker-managed known hosts file. The known hosts file is in the `\components\`*`BROKERNAME-NAME`*`\`*`EG-UID`*`\config\known_hosts` subdirectory of the directory in which WebSphere Message Broker is installed. For example, on Windows the default directory is `C:\Documents and Settings\All Users\Application Data\IBM\MQSI\components\`*`BROKERNAME-NAME`*`\`*`EG-UID`*`\config\known_hosts`. On UNIX systems the default directory is `/var/mqsi/components/`*`BROKERNAME-NAME`*`/`*`EG-UID`*`/config/known_hosts`.

     When the broker attempts to reconnect, it adds the new host key to the known hosts file.

3. Restart the message flow.

Related concepts:

"Transferring files securely by using SFTP" on page 1864
You can transfer files securely by using the Secure File Transfer Protocol (SFTP), which enables file transfer by using the Secure Shell (SSH) protocol.

"How the broker processes files" on page 1814
The broker reads files with the FileInput, FTEInput, and FileRead nodes, and writes files with the FileOutput and FTEOutput nodes.

Related tasks:

"Configuring SFTP file transfer" on page 1864
Use an FtpServer configurable service to specify the Secure File Transfer Protocol (SFTP) settings for a message flow, and to override the SFTP settings that are specified on the FileInput and FileOutput nodes.

"Connecting client applications" on page 1537
Connect your client applications to the broker by using one or more of the supported protocols from other resources and software servers in your network.

"Reading a file on a remote FTP or SFTP directory" on page 1842
Use a FileInput node to read a file in a directory on a remote FTP or SFTP server and then propagate messages that are based on the contents of that file.

"Writing a file to a remote FTP or SFTP server" on page 1855
Use a FileOutput node to write a file to a directory on a remote FTP or SFTP server.

Related reference:

"FtpServer configurable service properties" on page 3794
Select the properties and values that you want to change for an existing FtpServer configurable service or to create a new service.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"FileInput node" on page 4415
Use the FileInput node to process messages that are read from files.

"FileOutput node" on page 4430
Use the FileOutput node to write messages to files.

Managed file transfers using WebSphere MQ File Transfer Edition

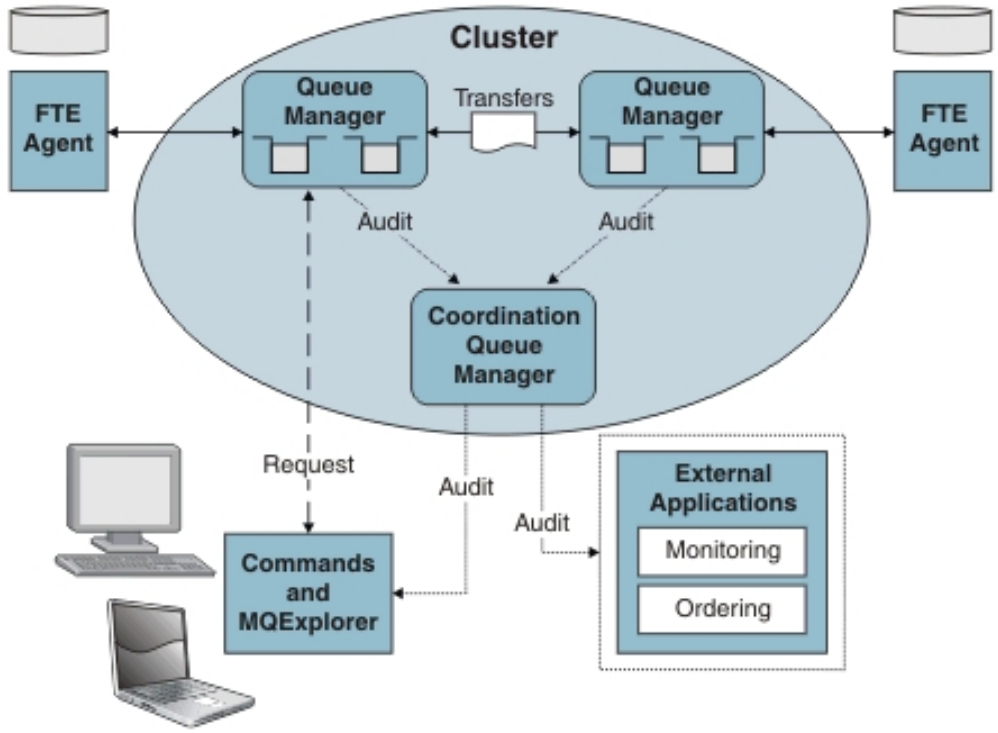
Transfer files, with file transfer metadata, in a timely and reliable manner.

- Transfer metadata is supported, providing more flexible file processing.
- Transfers are timely.
- The audit trail is complete between the broker and the remote agent.
- Resource statistics are available.

About WebSphere MQ File Transfer Edition

WebSphere MQ File Transfer Edition V7.0 delivers a reliable managed file transfer solution for moving files between IT systems without the need for programming. Files that are larger than the maximum individual WebSphere MQ message size can be moved. A log of file movements demonstrates that business data in files is transferred with integrity from source file system to destination file system.

Using the WebSphere MQ File Transfer Edition nodes offers seamless integration with your existing WebSphere MQ File Transfer Edition network. The following diagram shows a typical WebSphere MQ File Transfer Edition network:



The main components and concepts of WebSphere MQ File Transfer Edition are:

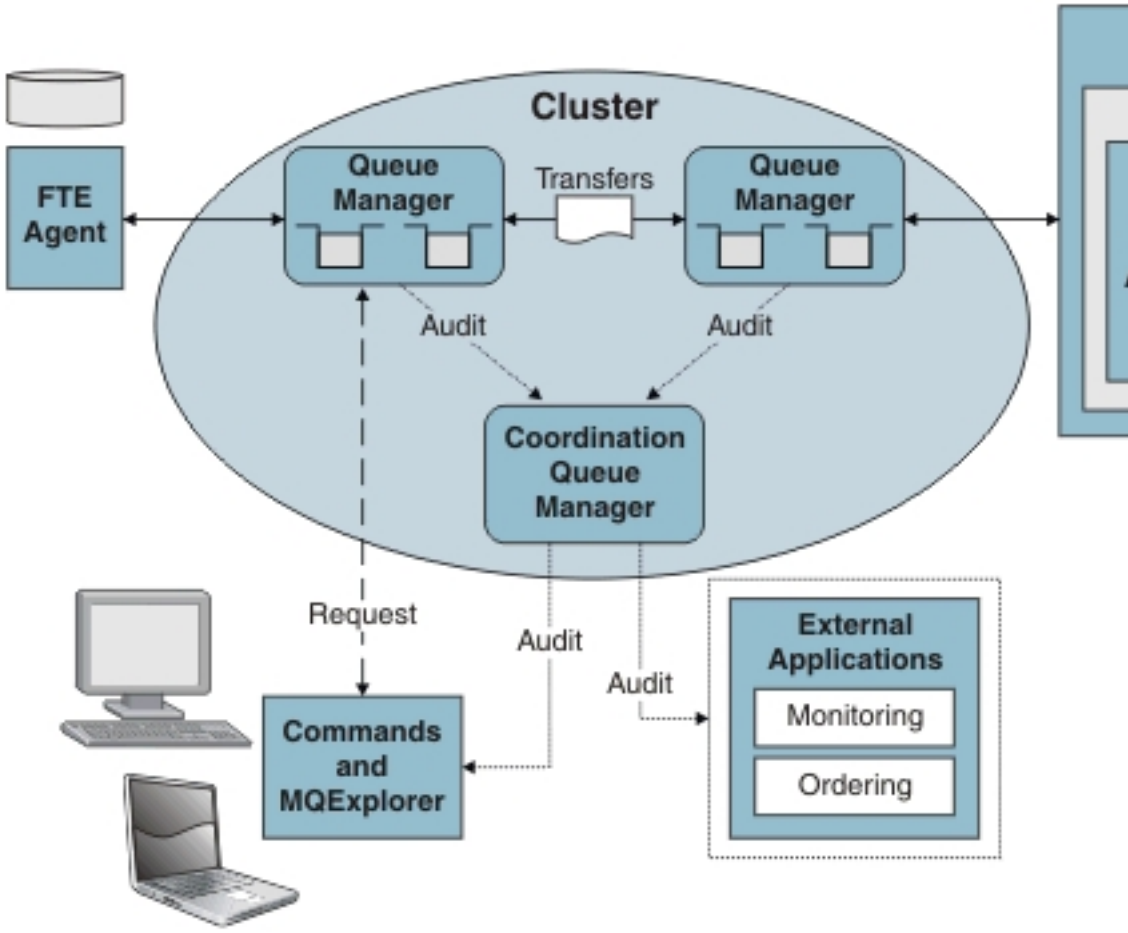
agent A process that forms the end point of a transfer (source or destination). An agent is a WebSphere MQ application, and is connected to one queue manager. Many agents can be connected to the same queue manager. Operational commands can be sent to an agent (for example, to request a transfer) by putting an XML message on a particular queue on the agent's queue manager.

coordination queue manager
One queue manager in the topology takes on the responsibility of the coordination queue manager. All agents register with the coordination queue manager and also send audit information about each transfer. The coordination queue manger is responsible for publishing that audit information to external monitoring applications.

transfer
A transfer is a movement of one or more files from one agent to another. The transfer goes direct from one agent's queue manager to the other agent's queue manager, not via the coordination queue manager. The transfer takes place even if the coordination queue manager is not running.

transfer log
The WebSphere MQ Explorer plug-in tool includes a Transfer Log view that subscribes to the coordination queue manager for the audit information. The view displays information about every transfer that occurs in a given topology.

How WebSphere Message Broker fits in



agent

An agent runs in each execution group that has deployed flows containing WebSphere MQ File Transfer Edition nodes. The agent is responsible for receiving and initiating all WebSphere MQ File Transfer Edition transfers. The agent name is derived from *Broker.ExecutionGroup*; the name is not configurable. See "Preparing the environment for WebSphere MQ File Transfer Edition nodes" on page 740 for restrictions on agent names.

You do not need to start or stop this agent; if a flow containing WebSphere MQ File Transfer Edition nodes is deployed, the agent is running. The agent is stopped only when the execution group is stopped. The broker queue manager is used as the queue manager for the agent.

coordination queue manager

The broker queue manager is the default coordination queue manager. You can specify a different coordination queue manager by using the WebSphere Message Broker Explorer, or the **mqsichangeproperties** command. You must configure your coordination queue manager as a coordination queue manager. See the WebSphere File Transfer Edition Information Center for details.

transfer

Multiple FTEOutput nodes can be deployed to the same execution group, or to different execution groups in the same broker. FTEOutput nodes can

send one file per transfer. Each file can have multiple records. Each record can have multiple elements. Transfers from the FTEOutput are non-blocking; an error occurs if another transfer is outstanding with the same file name.

Multiple FTEInput nodes can be deployed to the same execution group, or to different execution groups in the same broker. Multiple FTEInput nodes can read files transferred to the same directory without contention. Each file is processed only once, even if the nodes are deployed to separate execution groups, or different brokers.

When sending a file, you can dynamically set the following properties:

- `Destination agent`
- `Destination file directory`
- `Destination file name`
- `Destination queue manager`
- `Job name`
- `Overwrite files on destination`

Put a node such as the Compute node or Mapping node before the FTEOutput node.

When receiving files, you can apply filters. If an execution group has more than one FTEInput node, each node receives only the appropriate files. You can also determine what happens after the file has been processed (the file is left in its existing destination directory, left with a timestamp added, or deleted). See the Basic tab on the node for details.

You can combine the FTEInput and FTEOutput nodes to create a request and reply flow. See the sample for details.

Using elements in the local environment, you can call a program on the destination agent before starting the transfer, or when the transfer is finished. See "LocalEnvironment.Destination.FTE fields" on page 1826 for details.

Overview of using WebSphere MQ File Transfer Edition nodes

You do not need to configure the WebSphere MQ File Transfer Edition code that runs in the broker. Operational tools in WebSphere Message Broker Explorer are provided to create transfers. The following nodes are provided:

FTEOutput

FTEInput

Take the following steps to use the nodes to send or receive data across an existing WebSphere MQ File Transfer Edition network:

1. Create a flow that includes one of the WebSphere MQ File Transfer Edition nodes.
2. Configure the node.
3. For production purposes, change the coordination queue manager from the broker queue manager; see "Preparing the environment for WebSphere MQ File Transfer Edition nodes" on page 740.
4. Deploy the flow.

Monitoring the license usage of WebSphere MQ File Transfer Edition in WebSphere Message Broker

A message is written to the system log for every execution group that has a WebSphere MQ File Transfer Edition node deployed.

Related tasks:

"Sending a file by WebSphere MQ File Transfer Edition" on page 1859
Send files to an existing WebSphere MQ File Transfer Edition network.

"Receiving a file by WebSphere MQ File Transfer Edition" on page 1845
Use the FTEInput node to receive files from an existing WebSphere MQ File Transfer Edition network.

"Preparing the environment for WebSphere MQ File Transfer Edition nodes" on page 740
Prepare the file system and queue managers, and determine the name of the broker agent.

Related reference:

"FTEInput node" on page 4461
Use the FTEInput node to receive files using the WebSphere MQ File Transfer Edition.

"FTEOutput node" on page 4466
Use the FTEOutput node to write messages to files by using the WebSphere MQ File Transfer Edition.

"Resource statistics data" on page 6745
Learn about the measurements for which data is returned when you activate resource statistics collection.

Initiating a managed file transfer using IBM Sterling Connect:Direct

Use a CDOutput node to send a file from a specified directory on your primary Connect:Direct server (PNODE) to a filename and directory on a secondary Connect:Direct server (SNODE).

Before you begin

This topic describes use of the CDOutput node in a message flow and assumes that the rest of the flow has been developed, for example, an MQInput to a CDOutput node.

Multiple CDOutput nodes can be deployed to the same execution group, or to different execution groups in the same broker. CDOutput nodes can send one file per transfer. Each file can have multiple records; each record can have multiple elements. Transfers from the CDOutput node are non-blocking.

Complete the following steps:

Procedure

1. Set the required node properties on the CDOutput node.

   If you set the `Destination file name` only, and leave all the other options at their default values, the file is transferred:

   - From the primary Connect:Direct server (PNODE) back to itself
   - Into the default transfer directory using the default process name.

   Additionally, the file is created if it does not exist, or replaced if it already exists.

The following table summarizes the CDOutput node properties that you can set, on which tab they appear, and a value that you can select:

Tab	Property	Value
Basic	Process name	You can use any name you want for the process. Note, however, that the name must be a maximum of eight characters and cannot contain any spaces.
	SNODE	The secondary Connect:Direct server to which the file is being transferred.
	Destination file directory	TestDir on secondary Connect:Direct server (SNODE).
	Destination file name	Filename on secondary Connect:Direct server (SNODE).
	Disposition	RPL
	Transfer mode	Text mode

2. Setup the *username* and *password* that is required for the CDOutput node to connect to the primary Connect:Direct server using the "**mqsisetdbparms** command" on page 3954.
3. Deploy the message flow to the broker. See Chapter 11, "Packaging and deploying," on page 3209.
4. Send the file to the In terminal of the CDOutput node.

Results

The following actions occur when you perform these steps:

1. The file is constructed in accordance with the values set in the properties of the CDOutput node.
2. The file is staged in the local file system and then a command is sent to the Connect:Direct server to cause the transfer to occur.
3. If a file of the same name exists in the selected directory on the secondary Connect:Direct server, the processing of the existing file is determined by the value of the *Disposition* property; in this example, the file is replaced.

   Once the transfer has completed the local, staged file is deleted.

Note, that when sending a file, you can dynamically set the following properties:

- Secondary Connect:Direct server (SNODE)
- Process name
- Accounting data
- Destination file directory
- Destination file name
- Copy from options
- Copy to options

You have complete control of the Copy statements.

For example:

```
LocalEnvironment.Destination.CD.Copy.To.Option.PERMISS = '777'
```

causes IBM Sterling Connect:Direct to set the permissions on the destination file to `777` (or `RWX RWX RWX`) if the destination file is on a UNIX operating system, or within Unix System Services on z/OS.

However, if you enter an incorrect format, a syntax error is detected when the process script is submitted and this causes an error to be thrown in the node.

For example:

```
LocalEnvironment.Destination.CD.Copy.To.Option.PERMISS = '7xddd'
```

causes a syntax error because `'7xddd'` is not of the format `nnn`. When an error occurs the process script is available in the exception thrown and the user trace.

Tip: To help you with problem determination, you can view the script generated by the CDOutput node by turning on user trace.

Related concepts:

"How the broker processes files" on page 1814
The broker reads files with the FileInput, FTEInput, and FileRead nodes, and writes files with the FileOutput and FTEOutput nodes.

"IBM Sterling Connect:Direct overview and concepts" on page 1810
An overview of IBM Sterling Connect:Direct and its terminology when used with WebSphere Message Broker.

Related tasks:

"Receiving a file using IBM Sterling Connect:Direct" on page 1847
Use the CDInput node to receive files from an IBM Sterling Connect:Direct network.

"Advanced configuration properties when using IBM Sterling Connect:Direct nodes" on page 727
CDInput and CDOutput nodes can get connection details and staging directories in conjunction with a configurable service. To pick up new values when a configurable service is created or modified, you must reload the broker or execution group, by using the **mqsistop** and **mqsistart** commands, or the **mqsireload** command.

"Using a local file as input for your message flow" on page 1834
Learn how to use the FileInput node to read a file on your local file system and then propagate messages that are based on the contents of that file.

Related reference:

"CDOutput node" on page 4312
Use the CDOutput node when using IBM Sterling Connect:Direct with WebSphere Message Broker.

"CDInput node" on page 4305
Use the CDInput node to preview files when using IBM Sterling Connect:Direct in conjunction with WebSphere Message Broker.

WebSphere Service Registry and Repository

The WebSphere Service Registry and Repository (WSRR) is a central repository of entities. A wide range of entities can be stored and retrieved, including user-defined concepts and definitions related specifically to Web services, such as WSDL services, service interfaces, and associated policies.

You can configure a message flow to dynamically retrieve resources from WSRR at run time, and to use and expose those resources in the message flow. You can therefore defer the decision about which resources you want to use until run time, rather than deciding at deployment time.

WSRR has specific support for many of the document types associated with Web services, including generic XML documents, WSDL, and SCDL. For example, when you load a WSDL document into WSRR it also identifies and stores its individual logical components, such as the service and port type.

Use the WSRR nodes (the RegistryLookup and EndpointLookup nodes) to create message flows that retrieve data dynamically from WSRR. Data is retrieved according to search criteria defined by node properties, possibly supplemented or overridden by local environment definitions. The retrieved data is placed in the local environment tree, which makes the data available to subsequent nodes. The input message received by the node is propagated to the output terminal unchanged.

Use the RegistryLookup node to submit generic queries to WSRR. Entities returned by the query are stored in the ServiceRegistry output tree in the local environment. You can also specify that details of the relationships between the returned entities and other entities that they reference are represented in the ServiceRegistry output tree.

Use the EndpointLookup node to submit queries for Web service endpoints. This node is tailored to retrieve WSDL port definitions that implement a specified WSDL portType. The details of service endpoints that match the specified criteria are placed in the ServiceRegistry output tree in the local environment. If the node is configured to return a single matching service endpoint, the Web service URL destination used by the SOAP and HTTP request nodes is also overridden in the local environment. If the node is configured to return all matching service endpoints, the local environment is not set up automatically for the SOAP and HTTP request nodes. In this case the local environment tree might contain data for multiple service endpoints, and the message flow interprets and uses this information.

Set the configuration parameters for the WSRR nodes to specify how WebSphere Message Broker interfaces with your WSRR server.

- Use the **connectionTimeout** parameter to set a system-wide connection timeout for queries that are issued by the EndpointLookup and RegistryLookup nodes. If a query result is not returned from the WSRR server before the connection timeout period expires, the configured error handling is invoked.
- Use the **needCache** parameter to enable the WebSphere Message Broker WSRR cache. The cache is used to store results from queries that are issued by the EndpointLookup and RegistryLookup nodes.
- If the WebSphere Message Broker WSRR cache is enabled, use the **timeout** parameter to set a system-wide cache timeout. The cache timeout controls how long results from queries that are stored in the cache are used before the query is reissued.

If your WebSphere Message Broker application is running on Sun Solaris 10 on SPARC, you might need to increase the number of file descriptors. If there are not enough file descriptors, you might fail to get a response to a WSRR query within the configured timeout period. In addition, the service trace might contain one or

more java.lang.SecurityExceptions related to com.ibm.ws.tcp.channel.impl.ChannelSelector, and an abend file might be produced.

For information about the specific levels of WSRR that are supported with WebSphere Message Broker, see WebSphere Message Broker Requirements.

The topics in this section provide further information about working with WSRR:

- "Configuration parameters for the WebSphere Service Registry and Repository nodes"
- "Displaying the configuration parameters for the WebSphere Service Registry and Repository nodes" on page 1879
- "Changing the configuration parameters for the WebSphere Service Registry and Repository nodes" on page 1881
- "Accessing a secure WSRR repository" on page 1884
- "Caching artifacts from the WebSphere Service Registry and Repository" on page 1888
  - "Setting up cache notification" on page 1890
- "Dynamically defining the search criteria" on page 1891
- "EndpointLookup node" on page 4407
- "EndpointLookup node output" on page 1894
- "RegistryLookup node" on page 4646
- "RegistryLookup node output" on page 1897

Related information:

WebSphere Service Registry and Repository Library web page

Configuration parameters for the WebSphere Service Registry and Repository nodes

These parameters affect the broker configuration when interfacing with WebSphere Service Registry and Repository (WSRR).

The following table describes the parameters used by the **mqsichangeproperties** command to configure the `DefaultWSRR` object of the `Service Registries` configurable service.

Configuration Setting Name	Default value	Description
`endpointAddress`	For all versions of WSRR: `http://hostname:9080/WSRRCoreSDO/services/WSRRCoreSDOPort`	Endpoint of the WSRR server. Where *hostname:port* is host address and port of your WSRR server. To connect to a secure WSRR, specify `https://` instead of `http://`. For more information, see "Accessing a secure WSRR repository" on page 1884. For information about the specific levels of WSRR that are supported with WebSphere Message Broker, see WebSphere Message Broker Requirements.
`timeoutConnection`	`180`	The WSRR connection timeout period in seconds. The default value is `180` seconds. Set a value that is sufficiently long for complex queries to complete.

For more information about WSRR endpoint addresses, see WSRR Web service interface URLs.

The following table describes the parameters for Cache that are configured by using the **mqsichangeproperties** command.

Configuration Setting Name	Default value	Description
needCache	True	Enable WebSphere Message Broker WSRR cache.
timeout	100000000	The timeout value for the cache. The cache expiry time in milliseconds. (The default value of 100000000 milliseconds is approximately 27.8 hours).
predefinedCacheQueries	None	A list of semicolon-separated WSRR XPath query expressions with which to initialize the WebSphere Message Broker WSRR cache at startup. These WSRR XPath query expressions are defined by the WSRR SOAP interface query expression language, and can include an optional depth specifier extension.

The following table describes the parameters for Cache Notification that are configured by using the **mqsichangeproperties** command.

Configuration Setting Name	Default value	Description
enableCacheNotification	False	Enable WebSphere Message Broker WSRR Cache Notification.
refreshQueriesAfterNotification	True	When a notification is received from WSRR, if refreshQueriesAfterNotification is set to True, the cache is updated with the new version of the object immediately; if False, the cache is updated on the next request.
connectionFactoryName	jms/SRConnectionFactory	The name of the WSRR WebSphere Application Server JMS provider JMS connection factory for Cache Notification.
initialContextFactory	com.ibm.websphere.naming.WsnInitialContextFactory	The name of the WSRR WebSphere Application Server JMS provider JMS context factory for Cache Notification.
locationJNDIBinding	iiop://*hostname*:2809/	The URL to the WebSphere Application Server JMS provider JNDI bindings, where *hostname* is variable.
subscriptionTopic	jms/SuccessTopic	The topic name used to receive WebSphere Application Server JMS provider Cache Notification.

For information about accessing a secure WSRR server, see "Accessing a secure WSRR repository" on page 1884.

Related concepts:

"WebSphere Service Registry and Repository" on page 1875
The WebSphere Service Registry and Repository (WSRR) is a central repository of entities. A wide range of entities can be stored and retrieved, including user-defined concepts and definitions related specifically to Web services, such as WSDL services, service interfaces, and associated policies.

Related tasks:

"Displaying the configuration parameters for the WebSphere Service Registry and Repository nodes"
Use the **mqsireportproperties** command to display all of the configuration parameters of the default WebSphere Service Registry and Repository (WSRR) profile, `DefaultWSRR`.

"Changing the configuration parameters for the WebSphere Service Registry and Repository nodes" on page 1881
Use the **mqsichangeproperties** command to change the configuration parameters of the `DefaultWSRR` configurable service.

"Accessing a secure WSRR repository" on page 1884
To access a secure WebSphere Service Registry and Repository (WSRR) repository, set the configuration parameters by using the **mqsichangeproperties** command.

Related reference:

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

"EndpointLookup node" on page 4407
Use the EndpointLookup node to retrieve service endpoint information held in the WebSphere Service Registry and Repository (WSRR). The entities that match the specified search criteria are stored in the local environment. Depending on the node configuration, the Web service URL destination used by the SOAP and HTTP request nodes can also be set. The input message is not modified.

"RegistryLookup node" on page 4646
Use the RegistryLookup node to retrieve any type of entity held in the WebSphere Service Registry and Repository (WSRR). The entities that match the specified search criteria are stored in the local environment. The input message is not modified.

Displaying the configuration parameters for the WebSphere Service Registry and Repository nodes

Use the **mqsireportproperties** command to display all of the configuration parameters of the default WebSphere Service Registry and Repository (WSRR) profile, `DefaultWSRR`.

About this task

The `DefaultWSRR` is a `Service Registries` configurable service that is supplied for each broker, see "Configurable services properties" on page 3766.

For information about the specific levels of WSRR that are supported with WebSphere Message Broker, see WebSphere Message Broker Requirements.

To display the configuration parameters of the default WSRR profile `DefaultWSRR`, complete the following steps:

Procedure

1. Ensure that the broker is running. If it is not, use the **mqsistart** command to start it.
2. Enter the following command (where MB7BROKER is the name of your broker):

   ```
   mqsireportproperties MB7BROKER -c ServiceRegistries -o DefaultWSRR -r
   ```

   where:

   -c specifies the configurable service (in this case, ServiceRegistries)
   -o specifies the name of the object (in this case, DefaultWSRR)
   -r specifies that all property values of the object are displayed, including the child values if appropriate.

Results

The command produces a response similar to this:

```
ReportableEntityName=''
ServiceRegistries
  DefaultWSRR=''
    connectionFactoryName = 'jms/SRConnectionFactory'
    connectionTimeout = '120'
    endpointAddress = 'http://fill.in.your.host.here:9080/WSRRCoreSDO/services/WSRRCoreSDOPort'
    initialContextFactory = 'com.ibm.websphere.naming.WsnInitialContextFactory'
    locationJNDIBinding = 'iiop://localhost:2809/'
    needCache = 'true'
    predefinedCacheQueries = ''
    refreshQueriesAfterNotification = 'true'
    subscriptionTopic = 'jms/SuccessTopic'
    timeout = '10000000'
```

Related concepts:

"WebSphere Service Registry and Repository" on page 1875
The WebSphere Service Registry and Repository (WSRR) is a central repository of entities. A wide range of entities can be stored and retrieved, including user-defined concepts and definitions related specifically to Web services, such as WSDL services, service interfaces, and associated policies.

Related tasks:

"Changing the configuration parameters for the WebSphere Service Registry and Repository nodes" on page 1881
Use the **mqsichangeproperties** command to change the configuration parameters of the `DefaultWSRR` configurable service.

Related reference:

"Configuration parameters for the WebSphere Service Registry and Repository nodes" on page 1877
These parameters affect the broker configuration when interfacing with WebSphere Service Registry and Repository (WSRR).

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"**mqsistart** command" on page 3965
Use the **mqsistart** command to start the specified broker if all initial verification tests complete successfully.

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

"EndpointLookup node" on page 4407
Use the EndpointLookup node to retrieve service endpoint information held in the WebSphere Service Registry and Repository (WSRR). The entities that match the specified search criteria are stored in the local environment. Depending on the node configuration, the Web service URL destination used by the SOAP and HTTP request nodes can also be set. The input message is not modified.

"RegistryLookup node" on page 4646
Use the RegistryLookup node to retrieve any type of entity held in the WebSphere Service Registry and Repository (WSRR). The entities that match the specified search criteria are stored in the local environment. The input message is not modified.

Changing the configuration parameters for the WebSphere Service Registry and Repository nodes

Use the **mqsichangeproperties** command to change the configuration parameters of the `DefaultWSRR` configurable service.

About this task

`DefaultWSRR` is a configurable service object that is supplied for each broker, it defines the WebSphere Service Registry and Repository (WSRR) configuration parameters. `DefaultWSRR` has a configurable service type of `ServiceRegistries`.

For details about configuration parameters that affect WSRR use, see "Configuration parameters for the WebSphere Service Registry and Repository nodes" on page 1877.

To update the configuration parameters of the `DefaultWSRR` configurable service perform the following steps:

Procedure

1. Ensure that the broker is running. If it is not, use the **mqsistart** command to start it.
2. Enter the following command (which applies to all versions of WSRR) to change the `endpointAddress` value and point to your WebSphere Service Registry and Repository server:

   ```
   mqsichangeproperties MB7BROKER -c ServiceRegistries -o DefaultWSRR
   -n endpointAddress
   -v http://localhost:9080/WSRR6_1/services/WSRRCoreSDOPort
   ```

   where:

   -c specifies the configurable service type
   (in this case, `ServiceRegistries`)
   -o specifies the name of the configurable service object
   (in this case, `DefaultWSRR`)
   -n specifies the names of the properties to be changed
   (in this case, `endpointAddress`)
   -v specifies the values of properties defined by the **-n** parameter
   (in this case,
   `http://localhost:9080/WSRR6_1/services/WSRRCoreSDOPort`)

For information about the specific levels of WSRR that are supported with WebSphere Message Broker, see WebSphere Message Broker Requirements.

3. (Optional) Enter the following command to change the cache `timeout` value:

```
mqsichangeproperties MB7BROKER -c ServiceRegistries -o DefaultWSRR
-n timeout -v 3600000
```

where:

-c specifies the configurable service type
(in this case, `ServiceRegistries`)
-o specifies the name of the configurable service object
(in this case, `DefaultWSRR`)
-n specifies the names of the properties to be changed
(in this case, `timeout`)
-v specifies the values of properties defined by the **-n** parameter
(in this case, 3600000 milliseconds to provide WSRR cache expiry timeout of 1 hour)

4. (Optional) Enter the following command to change the `connectionTimeout` value:

```
mqsichangeproperties MB7BROKER -c ServiceRegistries -o DefaultWSRR
-n connectionTimeout -v 240
```

where:

-c specifies the configurable service type
(in this case, `ServiceRegistries`)
-o specifies the name of the configurable service object
(in this case, `DefaultWSRR`)
-n specifies the names of the properties to be changed
(in this case, `connectionTimeout`)
-v specifies the values of properties defined by the **-n** parameter
(in this case, 240 seconds to provide connection timeout for WSRR queries of 4 minutes)

5. (Optional) Enter the following command to preload the cache at broker startup with the results of specific queries:

```
mqsichangeproperties MB7BROKER -c ServiceRegistries -o DefaultWSRR
-n predefinedCacheQueries
-v "//*[@name=&apos;ConceptA1&apos;]"
```

where:

-c specifies the configurable service type
(in this case, `ServiceRegistries`)
-o specifies the name of the configurable service object
(in this case, `DefaultWSRR`)
-n specifies the names of the properties to be changed
(in this case, `predefinedCacheQueries`)
-v specifies the values of properties defined by the **-n** parameter
(in this case a simple full depth WSRR XPath query on the entity `ConceptA1`, `"//*[@name=&apos;ConceptA1&apos;]"`).

Note that single quotation marks in the WSRR query must be replaced by `&apos;)`

Multiple queries can be specified by delimiting them with ';'

For example, to perform a full depth query on the entities named `ConceptA1` and `ConceptB2` use:

```
-v "//*[@name=&apos;ConceptA1&apos;];//*[@name=&apos;ConceptB2&apos;]"
```

When a specific value for the `Depth Policy` property is used on a RegistryLookup node the same depth must be specified in the **predefinedCacheQueries** property by using an optional extension to the query expression of the form `depth=n`.
For the `Depth Policy` value of `MatchOnly` use `depth=0`.
For the `Depth Policy` value of `MatchPlusImmediate` use `depth=1`.
For the `Depth Policy` values of `MatchPlusAll` or `MatchShowReluse` `depth=-1`, which is the default value.

For example, the following query retrieves an XSD `MsgDef.xsd` with no related entities, the entity `ConceptA1` and its immediately related entities, and the entity `ServiceA2` and all entities related to it:

```
-v "//*[@name=&apos;MsgDef.xsd&apos;]{depth=0};
//*[@name=&apos;ConceptA1&apos;]{depth=1};
//*[@name=&apos;ServiceA2&apos;]"
```

Individual queries can use the full power of the WSRR query language:

```
-v "/WSRR/WSDLService/ports[binding/portType
[@name=&apos;DemoCustomer&apos;
and @namespace=&apos;http://demo.sr.eis.ibm.com&apos;]]"
```

Use the Broker User Trace to obtain the WSRR XPath query string that is issued when a RegistryLookup or EndpointLookup node is invoked.

Alternatively, use parameter `-p` instead of **-v** to specify a file from which the **mqsichangeproperties** command reads the property values:

```
mqsichangeproperties MB7BROKER -c ServiceRegistries -o DefaultWSRR
-n endpointAddress
-p config.xml
```

The following conditions apply when you use the **-p** parameter:

- The parameter is used to specify the location and name of a file from which the command reads the property values.
- The parameter can be used to set only a single property. Therefore, the **-n** parameter must specify only a single property name, not a comma-separated list of property names.
- White space characters (including line feed, carriage return, and end of file characters) are preserved when read from a file that is specified by using the **-p** parameter

See "Configurable services properties" on page 3766 for further detail.

6. Restart the broker, by using the **mqsistop** command to stop the broker, followed by the **mqsistart** command to start it.

Related concepts:

"WebSphere Service Registry and Repository" on page 1875
The WebSphere Service Registry and Repository (WSRR) is a central repository of entities. A wide range of entities can be stored and retrieved, including user-defined concepts and definitions related specifically to Web services, such as WSDL services, service interfaces, and associated policies.

Related tasks:

"Displaying the configuration parameters for the WebSphere Service Registry and Repository nodes" on page 1879
Use the **mqsireportproperties** command to display all of the configuration parameters of the default WebSphere Service Registry and Repository (WSRR) profile, `DefaultWSRR`.

Related reference:

"Configuration parameters for the WebSphere Service Registry and Repository nodes" on page 1877
These parameters affect the broker configuration when interfacing with WebSphere Service Registry and Repository (WSRR).

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsistart** command" on page 3965
Use the **mqsistart** command to start the specified broker if all initial verification tests complete successfully.

"**mqsistop** command" on page 3972
Use the **mqsistop** command to stop the specified component.

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

"EndpointLookup node" on page 4407
Use the EndpointLookup node to retrieve service endpoint information held in the WebSphere Service Registry and Repository (WSRR). The entities that match the specified search criteria are stored in the local environment. Depending on the node configuration, the Web service URL destination used by the SOAP and HTTP request nodes can also be set. The input message is not modified.

"RegistryLookup node" on page 4646
Use the RegistryLookup node to retrieve any type of entity held in the WebSphere Service Registry and Repository (WSRR). The entities that match the specified search criteria are stored in the local environment. The input message is not modified.

Accessing a secure WSRR repository

To access a secure WebSphere Service Registry and Repository (WSRR) repository, set the configuration parameters by using the **mqsichangeproperties** command.

About this task

You must connect over HTTPS, not HTTP, which is specified in the `endpointAddress` configuration parameter of the default WSRR profile, `DefaultWSRR`.

For more information about the `endpointAddress` configuration parameter, see "Configuration parameters for the WebSphere Service Registry and Repository nodes" on page 1877.

To access a secure WebSphere Service Registry and Repository, enter the following sequence of commands:

Procedure

1. Ensure that the broker is running. If it is not, use the **mqsistart** command to start it.
2. Use the ServiceRegistries configurable service to configure the broker to use HTTPS to communicate with the WSRR server. You can view the current configuration parameters for the ServiceRegistries configurable service by using the following command:

   ```
   mqsireportproperties MB7BROKER -c ServiceRegistries -o DefaultWSRR -r
   ```

where:

-c specifies the configurable service (in this case, ServiceRegistries)
-o specifies the name of the object (in this case, DefaultWSRR)
-r specifies that all property values of the object are displayed, including the child values, if appropriate.

To change the endpointAddress configuration parameter to specify HTTPS and the secure port for the DefaultWSRR of the ServiceRegistries configurable service, use the following command. The endpointAddress applies to the version of WSRR (for more information, see "Configuration parameters for the WebSphere Service Registry and Repository nodes" on page 1877).

```
mqsichangeproperties MB7BROKER -c ServiceRegistries -o DefaultWSRR
-n endpointAddress
-v https://localhost:9443/WSRR6_1/services/WSRRCoreSDOPort
```

where:

-c specifies the configurable service (in this case, ServiceRegistries)
-o specifies the name of the object (in this case, DefaultWSRR)
-n specifies the names of the properties to be changed
(in this case, endpointAddress)
-v specifies the values of properties defined by the **-n** parameter
(in this case, https://localhost:9443/WSRR6_1/services/WSRRCoreSDOPort)

For information about the specific levels of WSRR that are supported with WebSphere Message Broker, see WebSphere Message Broker Requirements.

3. Configure the broker keystore to contain your WSRR server certificate keys; for a discussion of digital certificates, see "Digital certificates" on page 356. Obtain these certificate keys from the installation of the WebSphere Application Server that hosts your WSRR server. The broker uses a single keystore, therefore, if your broker also implements WS-Security, HTTPS, or SSL-secured WebSphere MQ, you might need to merge the provided keys into an existing keystore file. The broker keystore is configured by using the **mqsichangeproperties** command to change configuration parameters for the broker. Display the current configuration parameters of the broker by using the following command:

   ```
   mqsireportproperties MB7BROKER -o BrokerRegistry -r
   ```

   where:

   -o specifies the name of the object (in this case, BrokerRegistry)
   -r specifies that all property values of the object are displayed, including the child values, if appropriate.

   To change the brokerKeystoreFile configuration parameters for the broker, use the following command:

   ```
   mqsichangeproperties MB7BROKER -o BrokerRegistry
   -n brokerKeystoreFile -v C:\WSRR\SSL\ClientKeyFile.jks
   ```

   where:

   -o specifies the name of the object (in this case, BrokerRegistry)
   -n specifies the names of the properties to be changed (in this case, brokerKeystoreFile
   -v specifies the values of properties defined by the **-n** parameter
   (in this case, C:\WSRR\SSL\ClientKeyFile.jks)

4. Configure the broker truststore to contain signer certificates for your WSRR server. As described previously for the keystore, the broker uses a single

truststore, therefore certificates might need to be merged into an existing truststore file. The broker truststore is configured by using the **mqsichangeproperties** command. To change the brokerTruststoreFile configuration parameters for the broker, use the following command:

```
mqsichangeproperties MB7BROKER -o BrokerRegistry
-n brokerTruststoreFile -v C:\WSRR\SSL\ClientTrustFile.jks
```

where:

-o specifies the name of the object
(in this case, BrokerRegistry)
-n specifies the names of the properties to be changed
(in this case, brokerTruststoreFile)
-v specifies the values of properties defined by the **-n** parameter
(in this case, C:\WSRR\SSL\ClientTrustFile.jks)

5. Stop the broker by using the **mqsistop** command. You must stop the broker to complete the following step.
6. Set the WebSphere Application Server user name and password by using the following command:

   ```
   mqsisetdbparms MB7BROKER -n DefaultWSRR::WSRR -u wasuser -p waspass
   ```

   where:

   -n specifies the name of the data source
   (in this case, DefaultWSRR::WSRR)
   -u specifies the user ID to be associated with this data source (in this case, wasuser)
   -p specifies the password to be associated with this data source (in this case, waspass)
7. Set the brokerKeystore user name and password by using the following command:

   ```
   mqsisetdbparms MB7BROKER -n brokerKeystore::password -u dummy -p WebAS
   ```

   where:

   -n specifies the name of the data source (in this case, brokerKeystore::password)
   -u specifies the user ID to be associated with this data source (in this case, dummy)
   -p specifies the password to be associated with this data source (in this case, WebAS)
8. Set the brokerTrustStore user name and password by using the following command:

   ```
   mqsisetdbparms MB7BROKER -n brokerTruststore::password -u dummy
   -p WebAS
   ```

   where:

   -n specifies the name of the data source (in this case, brokerTruststore::password)
   -u specifies the user ID to be associated with this data source (in this case, dummy)
   -p specifies the password to be associated with this data source (in this case, WebAS)
9. To use cache notification with your secure WSRR server, you must specify a valid user ID and password for the broker to use when connecting to the WSRR JMS cache notification topic. To set the user ID and password that the broker will use to make the JMS cache notification connection, use the following command:

   ```
   mqsisetdbparms MB7BROKER -n jms::DefaultWSRR@jms/SRConnectionFactory
   -u userid -p password
   ```

where:

-n specifies the name of the data source
(in this case, `jms::DefaultWSRR@jms/SRConnectionFactory`)
-u specifies the user ID to be associated with this data source
(in this case, *userid*)
-p specifies the password to be associated with this data source
(in this case, *password*)

Note: When using WebSphere Application Server Version 8.0, the default IIOP secure setting must be set to SSL-supported to enable the JMS cache notification. For more information about setting IIOP security, go to the WebSphere Application Server, Network Deployment (Distributed platforms and Windows), Version 8.0 information center, and read the topic "Common Secure Interoperability Version 2 transport inbound settings"; see WebSphere Application Server information center, Version 8.0.

10. Restart the broker by using the **mqsistart** command.

Related concepts:

"WebSphere Service Registry and Repository" on page 1875
The WebSphere Service Registry and Repository (WSRR) is a central repository of entities. A wide range of entities can be stored and retrieved, including user-defined concepts and definitions related specifically to Web services, such as WSDL services, service interfaces, and associated policies.

Related tasks:

"Displaying the configuration parameters for the WebSphere Service Registry and Repository nodes" on page 1879
Use the **mqsireportproperties** command to display all of the configuration parameters of the default WebSphere Service Registry and Repository (WSRR) profile, `DefaultWSRR`.

"Changing the configuration parameters for the WebSphere Service Registry and Repository nodes" on page 1881
Use the **mqsichangeproperties** command to change the configuration parameters of the `DefaultWSRR` configurable service.

Related reference:

"Configuration parameters for the WebSphere Service Registry and Repository nodes" on page 1877
These parameters affect the broker configuration when interfacing with WebSphere Service Registry and Repository (WSRR).

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

"**mqsistart** command" on page 3965
Use the **mqsistart** command to start the specified broker if all initial verification tests complete successfully.

"**mqsistop** command" on page 3972
Use the **mqsistop** command to stop the specified component.

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

"EndpointLookup node" on page 4407
Use the EndpointLookup node to retrieve service endpoint information held in the WebSphere Service Registry and Repository (WSRR). The entities that match the specified search criteria are stored in the local environment. Depending on the node configuration, the Web service URL destination used by the SOAP and HTTP request nodes can also be set. The input message is not modified.

"RegistryLookup node" on page 4646
Use the RegistryLookup node to retrieve any type of entity held in the WebSphere Service Registry and Repository (WSRR). The entities that match the specified search criteria are stored in the local environment. The input message is not modified.

Caching artifacts from the WebSphere Service Registry and Repository

WebSphere Message Broker saves the data it retrieves from the WebSphere Service Registry and Repository (WSRR) in a local cache.

The WSRR nodes, EndpointLookup and RegistryLookup, can retrieve data that was stored in the Broker WSRR cache by a previous query, improving performance and message throughput. The first occurrence of each query is always sent to WSRR. By default, this activity occurs when a WSRR node first issues a specific query, although it is possible to pre-populate the cache, when the broker starts, by using the queries described here.

Configuring the WSRR Cache

The cache is configured individually for each broker by using the WebSphere Message Broker Explorer or the **mqsichangeproperties** command; for details, see "Configurable services properties" on page 3766.

You can configure the cache in the following ways:

- **Disabling the cache**

  Disable the cache by setting the `needCache` parameter to `false`. By default, the cache is enabled, but the WSRR nodes can operate without the cache. If the cache is disabled, every query that is issued by the node is sent to WSRR, ensuring that the results of the query always reflect the current contents of the registry. This activity can affect performance.

- **Preloading the cache**

  Preload the cache by setting the `predefinedCacheQueries` parameter. By default, no items are preloaded in the cache and the first occurrence of every query is sent to WSRR. You can specify predefined queries that are executed when the broker starts, or when a message flow containing WSRR flows is first deployed, populating the cache for use by subsequent WSRR nodes. By specifying predefined queries, performance might be affected at startup, rather than on the first occurrence of a query at run time. The predefinedCacheQueries parameter is a list of WSRR XPath query expressions separated by semicolons, each with an optional depth specification. The user trace shows the WSRR XPath query expression generated by a WSRR node.

- **Changing the cache expiry timeout value**

  Change the cache expiry timeout value by setting the `timeout` parameter. The cached results of a query are discarded after the specified time has elapsed. The

next occurrence of the query is sent to WSRR and the new result is entered in the cache. If the contents of the registry are likely to change frequently, you can specify a shorter expiry timeout value so that changes are picked up quicker. This activity affects performance as more queries are sent to WSRR.

- **Enabling cache notification**

  Enable cache notification by setting the `enableCacheNotification` parameter to `true` and by setting the `initialContextFactory` and `locationJNDIBinding` properties appropriately for your WSRR server. By default, cache notification is disabled. Cache notification is a more flexible method than expiry timeout for refreshing cached data because it allows individual WSRR entities to be refreshed at the time they are modified in WSRR.

  If cache notification is enabled, the cache subscribes to events occurring in WSRR and is notified when an object is updated or deleted in WSRR. The object is discarded from the cache. If the `refreshQueriesAfterNotification` parameter is set to `true`, the cache is updated with the new version of the object immediately. If the `refreshQueriesAfterNotification` parameter is set to `false`, the cache is updated the next time a relevant query is issued by a WSRR node.

Related tasks:

"Setting up cache notification" on page 1890
Use the **mqsichangeproperties** command to enable cache notification, so that the cache is notified of events occurring in WebSphere Service Registry and Repository (WSRR).

Related reference:

"Configuration parameters for the WebSphere Service Registry and Repository nodes" on page 1877
These parameters affect the broker configuration when interfacing with WebSphere Service Registry and Repository (WSRR).

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

"EndpointLookup node" on page 4407
Use the EndpointLookup node to retrieve service endpoint information held in the WebSphere Service Registry and Repository (WSRR). The entities that match the specified search criteria are stored in the local environment. Depending on the node configuration, the Web service URL destination used by the SOAP and HTTP request nodes can also be set. The input message is not modified.

"RegistryLookup node" on page 4646
Use the RegistryLookup node to retrieve any type of entity held in the WebSphere Service Registry and Repository (WSRR). The entities that match the specified search criteria are stored in the local environment. The input message is not

modified.

Setting up cache notification:

Use the **mqsichangeproperties** command to enable cache notification, so that the cache is notified of events occurring in WebSphere Service Registry and Repository (WSRR).

About this task

WSRR publishes notification events by using WebSphere Application Server. Cache notification allows the cache to subscribe to these events.

To enable cache notification complete the following steps to change the relevant properties on the configurable service DefaultWSRR, and to add a user ID and password if you are connecting to a secure WebSphere Application Server:

Procedure

1. Ensure that the broker is running. If it is not, use the **mqsistart** command to start it.
2. Issue the **mqsichangeproperties** command to change the enableCacheNotification property to true. For example:

   ```
   mqsichangeproperties MB7BROKER -c ServiceRegistries -o DefaultWSRR
   -n enableCacheNotification -v true
   ```

   where:

   -c specifies the configurable service (in this case, ServiceRegistries)
   -o specifies the name of the object (in this case, DefaultWSRR)
   -n specifies the names of the properties to be changed
   (in this case, enableCacheNotification)
   -v specifies the values of properties defined by the **-n** parameter
   (in this case, true)
3. Issue the **mqsichangeproperties** command to change the locationJNDIBinding property to the value that you require for your WSRR server. For example:

   ```
   mqsichangeproperties MB7BROKER -c ServiceRegistries -o DefaultWSRR
   -n locationJNDIBinding -v iiop://localhost:2809/
   ```

   where:

   -c specifies the configurable service (in this case, ServiceRegistries)
   -o specifies the name of the object (in this case, DefaultWSRR)
   -n specifies the names of the properties to be changed
   (in this case, locationJNDIBinding)
   -v specifies the values of properties defined by the **-n** parameter
   (in this case, iiop://localhost:2809/)
4. If you are connecting to a secure WebSphere Application Server you must use a user ID and password. To set the user ID and password follow these steps:
   a. Stop the broker by using the **mqsistop** command.
   b. Issue the **mqsisetdbparms** command to set up your user ID and password. For example:

      ```
      mqsisetdbparms MB7BROKER -n jms::DefaultWSRR@jms/SRConnectionFactory
      -u userid -p password
      ```

      where:

-n specifies the name of the data source
(in this case, jms::DefaultWSRR@jms/SRConnectionFactory)
-u specifies the user ID to be associated with this data source
(in this case, *userid*)
-p specifies the password to be associated with this data source
(in this case, *password*)

c. Restart the broker by using the **mqsistart** command.

Related concepts:

"WebSphere Service Registry and Repository" on page 1875
The WebSphere Service Registry and Repository (WSRR) is a central repository of entities. A wide range of entities can be stored and retrieved, including user-defined concepts and definitions related specifically to Web services, such as WSDL services, service interfaces, and associated policies.

Related tasks:

"Accessing a secure WSRR repository" on page 1884
To access a secure WebSphere Service Registry and Repository (WSRR) repository, set the configuration parameters by using the **mqsichangeproperties** command.

"Displaying the configuration parameters for the WebSphere Service Registry and Repository nodes" on page 1879
Use the **mqsireportproperties** command to display all of the configuration parameters of the default WebSphere Service Registry and Repository (WSRR) profile, `DefaultWSRR`.

Related reference:

"Configuration parameters for the WebSphere Service Registry and Repository nodes" on page 1877
These parameters affect the broker configuration when interfacing with WebSphere Service Registry and Repository (WSRR).

"**mqsistart** command" on page 3965
Use the **mqsistart** command to start the specified broker if all initial verification tests complete successfully.

"**mqsistop** command" on page 3972
Use the **mqsistop** command to stop the specified component.

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

Dynamically defining the search criteria

You can use the RegistryLookup and EndpointLookup nodes to issue WebSphere Service Registry and Repository (WSRR) queries specified in the local environment.

The RegistryLookup and EndpointLookup nodes issue WSRR queries at run time and save the resulting data in the local environment. You can specify the queries at design time by using node properties to define the search criteria. Both nodes require at least one query property to be defined before you can deploy the message flow. However, you can specify the search criteria at run time in the local environment, either supplementing or overriding the node properties.

The following table defines the local environment overrides for WSRR queries. These fields must be set in

OutputLocalEnvironment.ServiceRegistryLookupProperties by a preceding transformation node, such as a Compute node.

Setting	Description
Name	This setting overrides the Name property on the node; for example, with an ESQL Compute node: `SET OutputLocalEnvironment.ServiceRegistryLookupProperties.Name = 'DemoCustomer';` This setting relates to the PortType Name property on the EndpointLookup node. Therefore, to set the PortType Name property, use the Name setting in the local environment.
Namespace	This setting overrides the Namespace property on the node; for example: `SET OutputLocalEnvironment.ServiceRegistryLookupProperties.Namespace = 'http://mb.sr.eis.ibm.com';` This setting relates to the PortType Namespace property on the EndpointLookup node. Therefore, to set the PortType Namespace property, use the Namespace setting in the local environment.
Version	This setting overrides the Version property on the node; for example: `SET OutputLocalEnvironment.ServiceRegistryLookupProperties.Version = '1.0';` This setting relates to the PortType Version property on the EndpointLookup node. Therefore, to set the PortType Version property, use the Version setting in the local environment.
MatchPolicy	This setting overrides the Match Policy property on the node; for example: `SET OutputLocalEnvironment.ServiceRegistryLookupProperties.MatchPolicy = 'One';` Valid values are One and All.
DepthPolicy	This setting overrides the Depth Policy property on the RegistryLookup node; for example: `SET OutputLocalEnvironment.ServiceRegistryLookupProperties.DepthPolicy = 'MatchOnly';` Valid values are: • MatchOnly for Return matched only (Depth = 0) • MatchShowRel for Return matched showing immediate relationships (For compatibility only) • MatchPlusImmediate for Return matched plus immediate related entities (Depth = 1) • MatchPlusAll for Return matched plus all related entities (Depth = -1) The MatchShowRel property provides compatibility with versions of WebSphere Message Broker before Version 6.1.0.4, by using the output format that was used in those previous versions. This option is deprecated, and should not be used if you are creating a new message flow. Consider migrating existing message flows to use one of the other options.

Setting	Description

UserProperties	This setting overrides the `User Properties` property on the node. You can specify more than one user-defined property in the local environment; for example:
	`SET OutputLocalEnvironment.ServiceRegistryLookupProperties.UserProperties.property1 = 'value1';`
	`SET OutputLocalEnvironment.ServiceRegistryLookupProperties.UserProperties.property2 = 'value2';`
	
	You can remove a user-defined property from the local environment by setting its value to NULL; for example:
	`SET OutputLocalEnvironment.ServiceRegistryLookupProperties.UserProperties.property1 = NULL;`
	
	You can use the node properties editor at design time to specify ESQL paths or XPath expressions to read the value for a user property at run time from a field in the message tree. However, the override values that you set in the local environment are the string values that are used in the query.

Classification	This setting overrides the `Classification` property on the node; for example:
	`SET OutputLocalEnvironment.ServiceRegistryLookupProperties.Classification = 'http://www.ibm.com/xmlns/prod/serviceregistry/6/0/governance/DefaultLifecycle#InitialState0';`
	
	You can specify more than one classification in the local environment. For example:
	`SET OutputLocalEnvironment.ServiceRegistryLookupProperties.Classification[1] = 'http://www.ibm.com/xmlns/prod/serviceregistry/6/0/governance/DefaultLifecycle#InitialState0';`
	`SET OutputLocalEnvironment.ServiceRegistryLookupProperties.Classification[2] = 'http://www.ibm.com.policy/GovernancePolicyDomain';`

Related concepts:

"EndpointLookup node output" on page 1894
Use an EndpointLookup node to retrieve the endpoint addresses for WSDL service definitions held in WebSphere Service Registry and Repository (WSRR).

"RegistryLookup node output" on page 1897
Use the RegistryLookup node to retrieve any type of entity held in the WebSphere Service Registry and Repository (WSRR). The entities that match the specified search criteria are stored in the local environment. The input message is not modified.

"Caching artifacts from the WebSphere Service Registry and Repository" on page 1888
WebSphere Message Broker saves the data it retrieves from the WebSphere Service Registry and Repository (WSRR) in a local cache.

Related tasks:

"Displaying the configuration parameters for the WebSphere Service Registry and Repository nodes" on page 1879
Use the **mqsireportproperties** command to display all of the configuration parameters of the default WebSphere Service Registry and Repository (WSRR) profile, `DefaultWSRR`.

"Accessing a secure WSRR repository" on page 1884
To access a secure WebSphere Service Registry and Repository (WSRR) repository, set the configuration parameters by using the **mqsichangeproperties** command.

"Changing the configuration parameters for the WebSphere Service Registry and Repository nodes" on page 1881
Use the **mqsichangeproperties** command to change the configuration parameters of the `DefaultWSRR` configurable service.

"Populating Destination in the local environment tree" on page 2467
Use the Destination subtree to set up the target destinations that are used by output nodes, the HTTPRequest node, the SOAPRequest node, the SOAPAsyncRequest node, and the RouteToLabel node. The following examples show how you can create and use an ESQL procedure to perform the task of setting up values for each of these uses.

Related reference:

"EndpointLookup node" on page 4407
Use the EndpointLookup node to retrieve service endpoint information held in the WebSphere Service Registry and Repository (WSRR). The entities that match the specified search criteria are stored in the local environment. Depending on the node configuration, the Web service URL destination used by the SOAP and HTTP request nodes can also be set. The input message is not modified.

"RegistryLookup node" on page 4646
Use the RegistryLookup node to retrieve any type of entity held in the WebSphere Service Registry and Repository (WSRR). The entities that match the specified search criteria are stored in the local environment. The input message is not modified.

"Configuration parameters for the WebSphere Service Registry and Repository nodes" on page 1877
These parameters affect the broker configuration when interfacing with WebSphere Service Registry and Repository (WSRR).

EndpointLookup node output

Use an EndpointLookup node to retrieve the endpoint addresses for WSDL service definitions held in WebSphere Service Registry and Repository (WSRR).

The input message is not changed by the EndpointLookup node. Instead, the local environment is updated to contain details of the endpoints retrieved by the query specified by the node and any local environment overrides.

You can configure the EndpointLookup node to dynamically set the service endpoint address for services that will be invoked by a subsequent SOAP or HTTP request node. The EndpointLookup node sets the destination URL in the local environment overrides for those nodes. See the following sample for an example of how to do this:

- WebSphere Service Registry and Repository Connectivity

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

EndpointLookup node output if the `Match Policy` property is set to `One`

If the `Match Policy` property of the node is set to `One`, theEndpointLookup node inserts the endpoint URL retrieved by the query into the local environment in an ITService entry in the local environment under ServiceRegistry, and sets the destination overrides for the SOAP and HTTP request nodes that can be connected directly to its output terminal. The following locations are updated:

- `LocalEnvironment.Destination.SOAP.Request.Transport.HTTP.WebServiceURL`
- `LocalEnvironment.Destination.HTTP.RequestURL`

These settings override the Web service URL property of the SOAPRequest, SOAPAsyncRequest, and HTTPRequest nodes, allowing a dynamic call to a Web service provider.

The following example shows typical output from the EndpointLookup node when the Match Policy is set to One. (Other entries might exist in the local environment depending on previous processing in the flow.)

```
<LocalEnvironment>
  <Destination>
    <SOAP>
      <Request>
        <Transport>
          <HTTP>
            <WebServiceURL>http://localhost:9081/DemoCustomerWeb/
              services/DemoCustomer</WebServiceURL>
          </HTTP>
        </Transport>
      </Request>
    </SOAP>
    <HTTP>
      <RequestURL>http://localhost:9081/DemoCustomerWeb/
        services/DemoCustomer
    </RequestURL>
   </HTTP>
  </Destination>
  <ServiceRegistry>
    <ITService>
      <Endpoint>
        <Address>http://localhost:9081/DemoCustomerWeb/
          services/DemoCustomer</Address>
        <PortType>
          <name>DemoCustomer</name>
          <namespace>http://demo.sr.eis.ibm.com</namespace>
          <version>1.0</version>
        </PortType>
        <Property>
          <name>policy</name>
          <value>RM</value>
        </Property>
        <Property>
          <name>country</name>
          <value>China</value>
        </Property>
        <Classification>http://eis.ibm.com/ServiceRegistry/
          GenericObjecttypes#Routing</Classification>
      </Endpoint>
    </ITService>
  </ServiceRegistry>
</LocalEnvironment>
```

EndpointLookup node output if the Match Policy property is set to All

If the Match Policy is set to All the EndpointLookup node writes an ITService entry in the local environment location ServiceRegistry for each endpoint retrieved by the query.

The following example shows typical output from the EndpointLookup node when the Match Policy is set to All. (Other entries might exist in the local environment depending on previous processing in the flow.)

```
<LocalEnvironment>
  <ServiceRegistry>
    <ITService>
      <Endpoint>
```

```
        <Address>http://localhost:9081/DemoCustomerWeb/
          services/DemoCustomer</Address>
        <PortType>
          <name>DemoCustomer</name>
          <namespace>http://demo.sr.eis.ibm.com</namespace>
          <version>1.0</version>
        </PortType>
        <Property>
          <name>policy</name>
          <value>RM</value>
        </Property>
        <Property>
          <name>country</name>
          <value>China</value>
        </Property>
        <Classification>http://eis.ibm.com/ServiceRegistry/
          GenericObjecttypes#Routing</Classification>
      </Endpoint>
    </ITService>
    <ITService>
      <Endpoint>
        <Address>http://localhost:9081/DemoCustomerWeb/
          services/DemoCustomer2</Address>
        <PortType>
          <name>DemoCustomer2</name>
          <namespace>http://demo.sr.eis.ibm.com</namespace>
          <version>1.0</version>
        </PortType>
      </Endpoint>
    </ITService>
  </ServiceRegistry>
</LocalEnvironment>
```

Related concepts:

"RegistryLookup node output" on page 1897
Use the RegistryLookup node to retrieve any type of entity held in the WebSphere Service Registry and Repository (WSRR). The entities that match the specified search criteria are stored in the local environment. The input message is not modified.

"Dynamically defining the search criteria" on page 1891
You can use the RegistryLookup and EndpointLookup nodes to issue WebSphere Service Registry and Repository (WSRR) queries specified in the local environment.

"Caching artifacts from the WebSphere Service Registry and Repository" on page 1888
WebSphere Message Broker saves the data it retrieves from the WebSphere Service Registry and Repository (WSRR) in a local cache.

"Local environment tree structure" on page 1056
The local environment tree is a part of the logical message tree in which you can store information while the message flow processes the message.

Related tasks:

"Populating Destination in the local environment tree" on page 2467
Use the Destination subtree to set up the target destinations that are used by output nodes, the HTTPRequest node, the SOAPRequest node, the SOAPAsyncRequest node, and the RouteToLabel node. The following examples show how you can create and use an ESQL procedure to perform the task of setting up values for each of these uses.

Related reference:

"EndpointLookup node" on page 4407
Use the EndpointLookup node to retrieve service endpoint information held in the WebSphere Service Registry and Repository (WSRR). The entities that match the specified search criteria are stored in the local environment. Depending on the node configuration, the Web service URL destination used by the SOAP and HTTP request nodes can also be set. The input message is not modified.

"RegistryLookup node" on page 4646
Use the RegistryLookup node to retrieve any type of entity held in the WebSphere Service Registry and Repository (WSRR). The entities that match the specified search criteria are stored in the local environment. The input message is not modified.

"SOAPRequest node" on page 4828
Use the SOAPRequest node to send a SOAP request to the remote Web service.

"HTTPRequest node" on page 4488
Use the HTTPRequest node to interact with a web service.

RegistryLookup node output

Use the RegistryLookup node to retrieve any type of entity held in the WebSphere Service Registry and Repository (WSRR). The entities that match the specified search criteria are stored in the local environment. The input message is not modified.

This topic contains the following sections:
- "Setting the Depth Policy property"
- "Local environment output tree" on page 1898
- "Migrating a message flow that uses the MatchShowRel option" on page 1900
- "Examples" on page 1900

Setting the `Depth Policy` property

The `Depth Policy` property on the RegistryLookup node specifies how much data is returned for each matched entity. The following table shows the valid values.

`Depth Policy` property value	**Local environment override value**	**Data returned**
`Return matched only (Depth = 0)`	`MatchOnly`	Just the matched entities
`Return matched showing immediate relationships (For compatibility only)`	`MatchShowRel`	The matched entities and additional references
`Return matched plus immediate related entities (Depth = 1)`	`MatchPlusImmediate`	The matched entities and the immediate related child entities
`Return matched plus all related entities (Depth = -1)`	`MatchPlusAll`	The matched entities and all the related child entities

Use `MatchShowRel` for compatibility with versions of WebSphere Message Broker before Version 6.1.0.4. The local environment output for `MatchShowRel` is shown in Example 1 and follows the format that was used in those previous versions. However, the `MatchShowRel` option is deprecated, and provided only for compatibility with previous versions. Do not use `MatchShowRel` if you are creating a new message flow, and consider migrating existing message flows to use one of the other options.

Use the MatchOnly option to retrieve just the individual entities matched by the search criteria. This option is efficient, however the local environment output does not contain any information about entities related to the matched entities.

Use the MatchPlusImmediate option to retrieve the entities matched by the search criteria, and the related child entities. This option offers a useful compromise, allowing you to access the immediate relations of the matched entities, while still restricting the total amount of data retrieved.

Use one of the MatchOnly or MatchPlusImmediate options when migrating a message flow that uses the deprecated MatchShowRel option. If your original message flow used the relationship information in the matched entities, use the MatchPlusImmediate option. See "Migrating a message flow that uses the MatchShowRel option" on page 1900.

Use the MatchPlusAll option to retrieve the entities matched by the search criteria, and all the related child entities. Use this option only if your message flow needs access to more than the immediate relations of the matched entities, because it retrieves considerably more data than the MatchPlusImmediate option, see "Local environment output tree."

Local environment output tree

The local environment output tree has a different format when the deprecated MatchShowRel option of the Depth Policy property is used. The following table describes the differences in the local environment output tree format.

MatchOnly, MatchPlusImmediate, and MatchPlusAll options	**MatchShowRel option**
The ServiceRegistry folder element is owned by the XMLNSC compiler.	No owning parser on the ServiceRegistry folder element, and each Entity element is owned by the XMLNS parser
The ServiceRegistry tree does not use unnecessary namespaces.	Namespaces are attached to all UserDefined folder elements, meaning that the path specified must declare and use the relevant namespace to access fields within these folders.
The ServiceRegistry tree is optimized through use of the XMLNSC parser.	The output tree contains a number of XML declaration, pcdata, and white space elements that have no business significance.
In WSRR, binary data is represented as a GenericDocument with a content attribute. The content attribute is represented in the local environment as XMLNSC type Attribute+base64Binary, (0x03000160). You can access the unencoded binary data directly as Entity.content. However, because of the special XMLNSC element type, the data is automatically base64 encoded if the tree is serialized.	Binary content is represented as a base64 encoded character string in the content attribute. The message flow must invoke a Java method to decode the value if the original binary data is required.

MatchOnly, MatchPlusImmediate, and MatchPlusAll options	**MatchShowRel option**
The retrieved entities are not modified to add any user properties.	The matched Entities appear in the local environment with a user property called `WSRREncoding`. This property has no specific significance to message flow processing. If there is a user property called `WSRREncoding` defined for the entity, the defined value is used, otherwise the `WSRREncoding` property is added with `value="DEFAULT"`.

The `MatchPlusImmediate` and `MatchPlusAll` options result in a graph of entities being returned when WSRR executes the query. Each entity in the graph can refer to other entities. There are two types of relationship that cannot be expressed directly in a hierarchical tree:

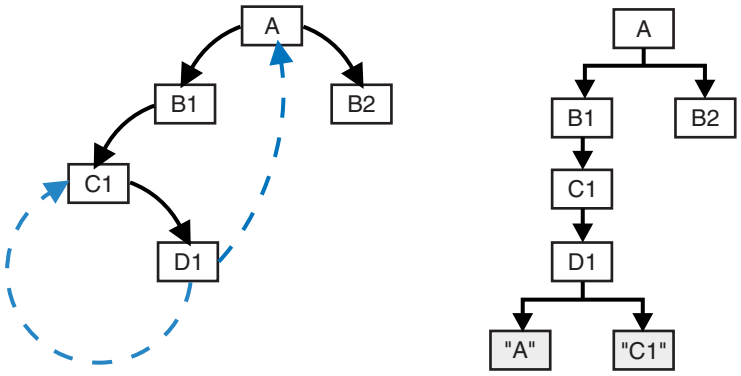
- Cyclic references - a graph can contain an entity from which you can follow relationships that arrive back at the same entity.
- Multiple references - a graph can contain an entity that is referenced by multiple entities.

The representation of these relationships in the local environment output tree is described in the following text.

The WSRR graph is represented in the local environment output tree as follows:

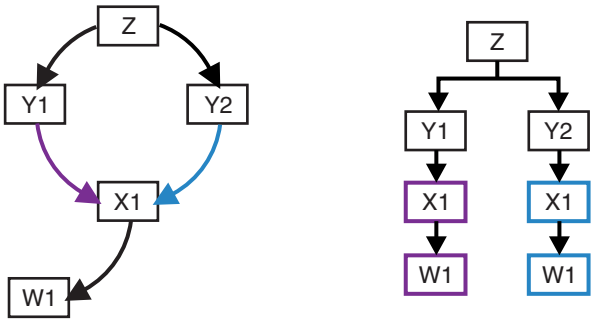
- The matched entities that are returned in the WSRR graph are represented as Entity elements as the first children of the LocalEnvironment.ServiceRegistry. The properties of the matched entities are represented as child elements.
- A matched entity can contain references to other entities. If `MatchPlusImmediate` is set, these references are represented as Entity child elements of the matched entity. If `MatchPlusAll` is set, the same rule is applied recursively to these children.
- Cyclic references - if an entity references another entity that is one of its ancestors in the WSRR graph, the referenced entity is represented as an EntityRef element. An EntityRef element does not represent the target entity directly, but provides information to identify it. This prevents the possibility of entering cyclic loops when navigating the tree.

  The following diagram shows references from entity D1 in the graph to entities C1 and A that are ancestral; therefore, Entity D1 in the message tree contains EntityRef elements for A and C1.



- Multiple references - if an entity in the WSRR SDO graph is referenced by more than one other entity, it is represented as a separate Entity element in the message tree each time it is referenced. The Entity element is cloned, along with any entities that it refers to.

  The following diagram shows that entity X1 in the graph is referenced by entities Y1 and Y2, so Entity X1 and Entity W1 that it references are modeled twice in the message tree.



Migrating a message flow that uses the `MatchShowRel` option

When you migrate an existing message flow that uses the deprecated `MatchShowRel` option of the `Depth Policy` property, the local environment output tree will have a different format. The table shown earlier describes the differences in the local environment output tree formats, and indicates what you must modify in the message flow to access the data in the updated format; see "Local environment output tree" on page 1898.

Examples

The following examples show typical output from the RegistryLookup node:

- Example 1 shows the full RegistryLookup node output in two cases for a query that returns two versions of a concept entity. In both cases the `Match Policy` property is set to `All`. In the first case the `Depth Policy` property is set to `Return matched plus immediate related entities (Depth = 1)` , and in the second case the `Depth Policy` property is set to `Return matched only, showing immediate relationships (For compatibility only)`. This example also shows example ESQL to read elements of the output. See "RegistryLookup node output: example 1" on page 1901
- Example 2 shows the structure of RegistryLookup node output for all possible values of the `Depth Policy` property for a query on a concept entity that has a number of user relationships to other concept entities. See "RegistryLookup node output: example 2" on page 1906
- Example 3 shows the structure of RegistryLookup node output for a query on an entity having metadata relationships and user-defined relationships, using a `Depth Policy` property value of `Return matched plus all related entities (Depth = -1)` . See "RegistryLookup node output: example 3" on page 1909.

Related concepts:

"EndpointLookup node output" on page 1894
Use an EndpointLookup node to retrieve the endpoint addresses for WSDL service definitions held in WebSphere Service Registry and Repository (WSRR).

"Dynamically defining the search criteria" on page 1891
You can use the RegistryLookup and EndpointLookup nodes to issue WebSphere Service Registry and Repository (WSRR) queries specified in the local environment.

"Caching artifacts from the WebSphere Service Registry and Repository" on page 1888
WebSphere Message Broker saves the data it retrieves from the WebSphere Service Registry and Repository (WSRR) in a local cache.

Related reference:

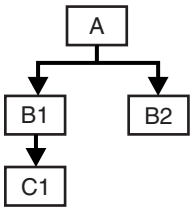
"EndpointLookup node" on page 4407
Use the EndpointLookup node to retrieve service endpoint information held in the WebSphere Service Registry and Repository (WSRR). The entities that match the specified search criteria are stored in the local environment. Depending on the node configuration, the Web service URL destination used by the SOAP and HTTP request nodes can also be set. The input message is not modified.

"RegistryLookup node" on page 4646
Use the RegistryLookup node to retrieve any type of entity held in the WebSphere Service Registry and Repository (WSRR). The entities that match the specified search criteria are stored in the local environment. The input message is not modified.

RegistryLookup node output: example 1:

Example showing the full RegistryLookup node output in two cases for a query that returns two versions of a concept entity. In both cases the `Match Policy` property is set to `All`. In the first case the `Depth Policy` property is set to `Return matched plus immediate related entities (Depth = 1)` , and in the second case the `Depth Policy` property is set to `Return matched only, showing immediate relationships (For compatibility only)`. This example also shows example ESQL to read elements of the output.

This example shows the ServiceRegistry message tree that is stored in the LocalEnvironment when an entity called ConceptA1 is retrieved from WebSphere Service Registry and Repository (WSRR). ConceptA1 is defined in WSRR with 2 versions: 1.0 which is deprecated, and 2.0 which is classified as initial state. Additional properties and relationships have been added to the 2.0 version. The following diagram shows the relationships between the elements in the message tree.



The following message trees show the different values of the `Depth Policy` property.

- `Depth Policy` property set to `Return matched plus immediate related entities (Depth = 1)`

  The following ESQL was used to create a serialization of the output of the ServiceRegistry node:

  ```
  SET OutputRoot.XMLNSC.Result.ServiceRegistry =
    InputLocalEnvironment.ServiceRegistry;
  ```

The ServiceRegistry folder element is owned by the XMLNSC parser so you can invoke a like parser tree copy to OutputRoot.XMLNSC. The following XML, which has been formatted, is produced when writing this output root tree.

```
<ServiceRegistry>
  <Entity
    bsrURI="33a9ad33-d4a4-442e.b3a6.37e62137a605"
    name="ConceptA1"
    namespace="http://www.examples.com/ConceptA1"
    version="2.0"
    description="A version 2 of the existing one"
    owner="UNAUTHENTICATED"
    lastModified="1230116323343"
    creationTimestamp="1227176757406"
    lastModifiedBy="UNAUTHENTICATED"
    primaryType="">
    <classificationURIs>
      http://www.ibm.com/xmlns/prod/serviceregistry/6/0/governance/
        DefaultLifecycle#InitialState0
    </classificationURIs>
    <classificationURIs>
      http://www.ibm.com.policy/GovernancePolicyDomain
    </classificationURIs>
    <userDefinedProperties name="property1" value="value1 for v1" />
    <userDefinedProperties name="property2" value="value1 for v2" />
    <userDefinedProperties name="property3" value="value1 for v3" />
    <userDefinedRelationships name="ContainsChildren">
      <targetEntities>
        <Entity
          bsrURI="8203cb82-8827-4757.99e1.36de6036e1af"
          name="ConceptB1"
          namespace="http://www.examples.com/ConceptB1"
          version="2.0"
          description="Next revision of this concept"
          owner="UNAUTHENTICATED"
          lastModified="1227191748250"
          creationTimestamp="1227177460156"
          lastModifiedBy="UNAUTHENTICATED"
          primaryType="" />
        <Entity
          bsrURI="a0d2bba0-f395-45bc.929e.04d143049eb5"
          name="ConceptB2"
          namespace="http://www.examples.com/ConceptB2" version="1.0"
          description="Testing"
          owner="UNAUTHENTICATED"
          lastModified="1227191700812"
          creationTimestamp="1227177334515"
          lastModifiedBy="UNAUTHENTICATED"
          primaryType="" />
      </targetEntities>
    </userDefinedRelationships>
    <userDefinedRelationships name="ReferTo">
      <targetEntities>
        <Entity
          bsrURI="81e45381-a9be-4ea4.9519.53657953196d"
          name="ConceptC1"
          namespace="http://www.examples.com/ConceptC1"
          version="1.0"
          description="Test stuff C1"
          owner="UNAUTHENTICATED"
          lastModified="1227874855140"
          creationTimestamp="1227177519609"
          lastModifiedBy="UNAUTHENTICATED" primaryType="" />
      </targetEntities>
    </userDefinedRelationships>
  </Entity>
  <Entity
```

```
    bsrURI="b68952b6-8d68-4840.8e5e.a3716da35e2e"
    name="ConceptA1"
    namespace="http://www.examples.com/ConceptA1"
    version="1.0"
    description="The original concept"
    owner="UNAUTHENTICATED"
    lastModified="1229030287593"
    creationTimestamp="1227173773250"
    lastModifiedBy="UNAUTHENTICATED"
    primaryType="">
    <classificationURIs>
      http://www.ibm.com/xmlns/prod/serviceregistry/6/0/governance/
        DefaultLifecycle#Deprecate
    </classificationURIs>
    <userDefinedProperties name="property1" value="value1" />
    <userDefinedProperties name="property2" value="value2" />
    <userDefinedRelationships name="ContainsChildren">
      <targetEntities>
        <Entity
          bsrURI="7d5fd37d-e90a-4ab0.89eb.d25b81d2ebec"
          name="ConceptB1"
          namespace="http://www.examples.com/ConceptB1" version="1.0"
          description="" owner="UNAUTHENTICATED"
          lastModified="1227874785062"
          creationTimestamp="1227177401265"
          lastModifiedBy="UNAUTHENTICATED"
          primaryType="" />
        <Entity
          bsrURI="a0d2bba0-f395-45bc.929e.04d143049eb5"
          name="ConceptB2"
          namespace="http://www.examples.com/ConceptB2" version="1.0"
          description="Testing" owner="UNAUTHENTICATED"
          lastModified="1227191700812"
          creationTimestamp="1227177334515"
          lastModifiedBy="UNAUTHENTICATED"
          primaryType="" />
      </targetEntities>
    </userDefinedRelationships>
  </Entity>
</ServiceRegistry>
```

The following ESQL shows how to retrieve the values from the ServiceRegistry message tree in the LocalEnvironment when the Depth Policy property is set to Return matched plus immediate related entities (Depth = 1).

```
DECLARE c1 CHARACTER;

-- Following sets c1 to "ConceptA1" by indexing the first entity
SET c1 = InputLocalEnvironment.ServiceRegistry.Entity[1].name;
-- Following sets c1 to "2.0" by indexing the first entity
SET c1 = InputLocalEnvironment.ServiceRegistry.Entity[1].version;
-- Following sets c1 to "property1" by indexing the first
--   userDefinedProperties within the first entity
SET c1 = InputLocalEnvironment.ServiceRegistry.Entity[1].
  userDefinedProperties[1].name;
-- Following sets c1 to "value1 for v2" by indexing the first
--   userDefinedProperties within the first entity
SET c1 = InputLocalEnvironment.ServiceRegistry.Entity[1].
  userDefinedProperties[1].value;
```

- Depth Policy property set to Return matched showing immediate relationships (For compatibility only)

  The following ESQL was used to create a serialization of the output of the ServiceRegistry node:

  ```
  DECLARE I INTEGER 1;
  DECLARE J INTEGER;
  SET J = CARDINALITY(InputLocalEnvironment.ServiceRegistry.Entity[]);
  ```

```
WHILE I < J DO
  SET OutputRoot.XMLNS.ServiceRegistry.Entity[I] =
    InputLocalEnvironment.ServiceRegistry.Entity[I];
  SET I = I + 1;
END WHILE;
```

The ServiceRegistry folder does not have an owning parser so you must navigate to the ""elements and initiate a like-parser-copy to the XMLNS owned output root tree. The following XML is produced when writing this output root tree.

```
<ServiceRegistry>
  <Entity
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:sdo="http://www.ibm.com/xmlns/prod/serviceregistry/6/0/sdo"
    xsi:type="sdo:GenericObject"
    bsrURI="33a9ad33-d4a4-442e.b3a6.37e62137a605"
    name="ConceptA1"
    namespace="http://www.examples.com/ConceptA1"
    version="2.0"
    description="A version 2 of the existing one"
    owner="UNAUTHENTICATED"
    lastModified="1229439847694"
    creationTimestamp="1227176757406"
    lastModifiedBy="UNAUTHENTICATED"
    primaryType="">
    <sdo:classificationURIs>
      http://www.ibm.com/xmlns/prod/serviceregistry/6/0/governance/
        DefaultLifecycle#InitialState0
    </sdo:classificationURIs>
    <sdo:classificationURIs>
      http://www.ibm.com.policy/GovernancePolicyDomain
    </sdo:classificationURIs>
    <sdo:userDefinedRelationships
      name="ContainsChildren"
      targets="8203cb82-8827-4757.99e1.36de6036e1af
        a0d2bba0-f395-45bc.929e.04d143049eb5" />
    <sdo:userDefinedRelationships
      name="ReferTo"
      targets="81e45381-a9be-4ea4.9519.53657953196d" />
    <sdo:userDefinedProperties name="property1" value="value1 for v2" />
    <sdo:userDefinedProperties name="property2" value="value2 for v2" />
    <sdo:userDefinedProperties name="property3" value="value3 for v2" />
    <sdo:userDefinedProperties name="WSRRencoding" value="DEFAULT" />
  </Entity>
  <Entity>
    <NS1:type
      xmlns:NS1="http://www.w3.org/2001/XMLSchema-instance">
      sdo:GenericObject
    </NS1:type>
    <bsrURI>b68952b6-8d68-4840.8e5e.a3716da35e2e</bsrURI>
    <name>ConceptA1</name>
    <namespace>http://www.examples.com/ConceptA1</namespace>
    <version>1.0</version>
    <description>The original concept</description>
    <owner>UNAUTHENTICATED</owner>
    <lastModified>1229030287593</lastModified>
    <creationTimestamp>1227173773250</creationTimestamp>
    <lastModifiedBy>UNAUTHENTICATED</lastModifiedBy>
    <primaryType></primaryType>
    <NS2:classificationURIs
      xmlns:NS2="http://www.ibm.com/xmlns/prod/serviceregistry/6/0/sdo">
      http://www.ibm.com/xmlns/prod/serviceregistry/6/0/governance/
        DefaultLifecycle#Deprecate
    </NS2:classificationURIs>
    <NS3:userDefinedRelationships
      xmlns:NS3="http://www.ibm.com/xmlns/prod/serviceregistry/6/0/sdo">
```

```
    </NS3:userDefinedRelationships>
    <NS4:userDefinedProperties
      xmlns:NS4="http://www.ibm.com/xmlns/prod/serviceregistry/6/0/sdo">
    </NS4:userDefinedProperties>
    <NS5:userDefinedProperties
      xmlns:NS5="http://www.ibm.com/xmlns/prod/serviceregistry/6/0/sdo">
    </NS5:userDefinedProperties>
    <NS6:userDefinedProperties
      xmlns:NS6="http://www.ibm.com/xmlns/prod/serviceregistry/6/0/sdo">
    </NS6:userDefinedProperties>
  </Entity>
</ServiceRegistry>
```

The following ESQL shows how to retrieve the values from the ServiceRegistry message tree in the LocalEnvironment when the `Depth Policy` property is set to `Return matched showing immediate relationships (For compatibility only)`. Note that in this case it is necessary to use the namespace qualifier.

```
DECLARE ns1 NAMESPACE 'http://www.ibm.com/xmlns/prod/serviceregistry/6/0/sdo';
DECLARE c1 CHARACTER;

-- Following sets c1 to "ConceptA1" by indexing the first entity
SET c1 = InputLocalEnvironment.ServiceRegistry.Entity[1].name;
-- Following sets c1 to "2.0" by indexing the first entity
SET c1 = InputLocalEnvironment.ServiceRegistry.Entity[1].version;
-- Following sets c1 to "property1" by indexing the first
--   userDefinedProperties within the first entity
SET c1 = InputLocalEnvironment.ServiceRegistry.Entity[1].ns1:
userDefinedProperties[1].name;
-- Following sets c1 to "value1 for v2" by indexing the first
--   userDefinedProperties within the first entity
SET c1 = InputLocalEnvironment.ServiceRegistry.Entity[1].ns1:
userDefinedProperties[1].value;
```

Related concepts:

"RegistryLookup node output: example 2" on page 1906
Example showing the structure of RegistryLookup node output for all possible values of the `Depth Policy` property for a query on a concept entity that has a number of user relationships to other concept entities.

"RegistryLookup node output: example 3" on page 1909
Example showing the structure of RegistryLookup node output for a query on an entity having metadata relationships and user-defined relationships, using a `Depth Policy` property value of `Return matched plus all related entities (Depth = -1)` .

"EndpointLookup node output" on page 1894
Use an EndpointLookup node to retrieve the endpoint addresses for WSDL service definitions held in WebSphere Service Registry and Repository (WSRR).

"Dynamically defining the search criteria" on page 1891
You can use the RegistryLookup and EndpointLookup nodes to issue WebSphere Service Registry and Repository (WSRR) queries specified in the local environment.

"Caching artifacts from the WebSphere Service Registry and Repository" on page 1888
WebSphere Message Broker saves the data it retrieves from the WebSphere Service Registry and Repository (WSRR) in a local cache.

Related reference:

"EndpointLookup node" on page 4407
Use the EndpointLookup node to retrieve service endpoint information held in the WebSphere Service Registry and Repository (WSRR). The entities that match the specified search criteria are stored in the local environment. Depending on the node configuration, the Web service URL destination used by the SOAP and HTTP

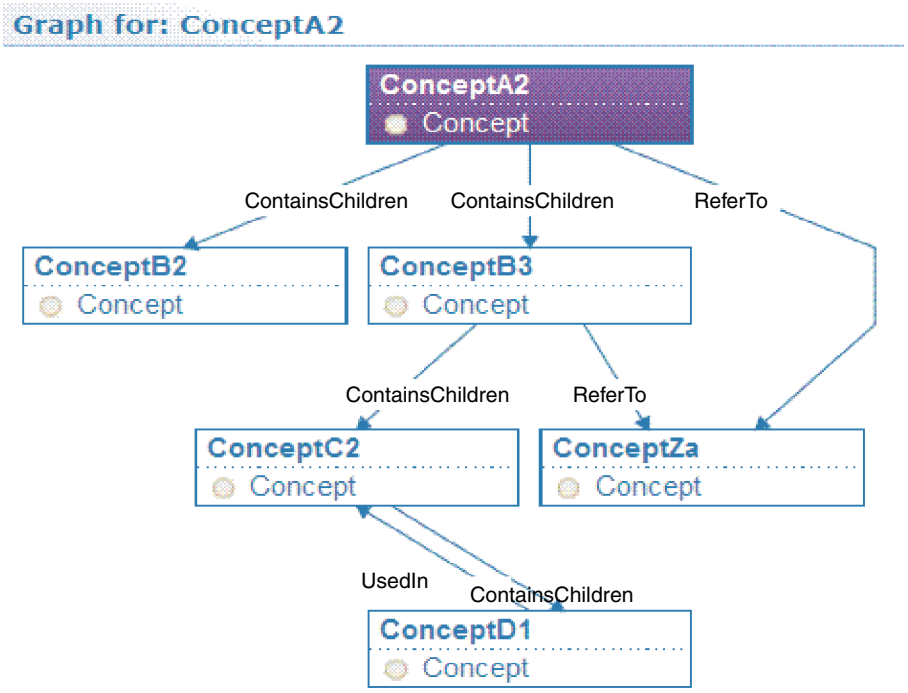
request nodes can also be set. The input message is not modified.

"RegistryLookup node" on page 4646
Use the RegistryLookup node to retrieve any type of entity held in the WebSphere Service Registry and Repository (WSRR). The entities that match the specified search criteria are stored in the local environment. The input message is not modified.

RegistryLookup node output: example 2:

Example showing the structure of RegistryLookup node output for all possible values of the `Depth Policy` property for a query on a concept entity that has a number of user relationships to other concept entities.

This example shows the ServiceRegistry message trees that are stored in the LocalEnvironment when the Concepts shown in the following WebSphere Service Registry and Repository graph are retrieved. The graph has been annotated with the relationship names to clarify the elements in the message tree.



The following ServiceRegistry message trees have some elements replaced by `...` to emphasis the structure of the tree. Likewise, the bsrURIs have been truncated.

The following shows the message trees for each possible value of the `Depth Policy` property:

- `Return matched only (Depth = 0)`

```
ServiceRegistry
  Entity
    type   = sdo:GenericObject
    bsrURI = a2e62137a605
    name   = ConceptA2
    ...
```

- `Return matched showing immediate relationships (For compatibility only)`. The entities contain elements showing the details of relationships, but only provide a list of the bsrURIs for the related child entities.

  This value of the `Depth Policy` property is deprecated, so you should use of the other options. The output tree structure produced when using this value is not compatible with those from the other values for the `Depth Policy` property. In particular, note the namespace qualifications.

```
ServiceRegistry
  Entity
    type   = sdo:GenericObject
    bsrURI = a2e62137a605
    name   = ConceptA2
    ...
    ns1:userDefinedRelationships
      name    = ContainsChildren
      targets = b2f73637f6e8 b3de6036e1af
    ns1:userDefinedRelationships
      name    = ReferTo
      targets = zac084d6b804
```

- `Return matched plus immediate related entities (Depth = 1)`. The entities contain elements showing the details of relationships, and the details of the related child entities.

```
ServiceRegistry
  Entity
    type   = GenericObject
    bsrURI = a2e62137a605
    name   = ConceptA2
    ...
    userDefinedRelationships
      name = ContainsChildren
      targetEntities
        Entity
          bsrURI = b2f73637f6e8
          name   = ConceptB2
          ...
        Entity
          bsrURI = b3de6036e1af
          name   = ConceptB3
          ...
          userDefinedRelationships
            name    = ContainsChildren
            targets = c26e43ac45a
          userDefinedRelationships
            name    = ReferTo
            targets = zac084d6b804
    userDefinedRelationships
      name = ReferTo
      targetEntities
        Entity
          bsrURI = zac084d6b804
          name   = ConceptZa
          ...
```

- `Return matched plus all related entities (Depth = -1)`. The entities contain elements showing the details of relationships, and the details of the all related child entities. ConceptD1 uses an EntityRef element to refer to its ancestor ConceptC2. ConceptZa appears twice in the tree as it is referenced by both ConceptA2 and ConceptB3.

```
ServiceRegistry
  Entity
    type   = sdo:GenericObject
    bsrURI = a2e62137a605
    name   = ConceptA2
```

```
...
userDefinedRelationships
  name = ContainsChildren
  targetEntities
    Entity
      bsrURI = b2f73637f6e8
      name   = ConceptB2
      ...
    Entity
      bsrURI = b3de6036e1af
      name   = ConceptB3
      ...
      userDefinedRelationships
        name = ContainsChildren
        targetEntities
          Entity
            bsrURI = c26e43ac45a
            name   = ConceptC2
            ...
            userDefinedRelationships
              name = ContainsChildren
              targetEntities
                Entity
                  bsrURI = d16e43ac763
                  name   = ConceptD1
                  ...
                  userDefinedRelationships
                    name = UsedIn
                    targetEntities
                      EntityRef
                        bsrURI = c26e43ac45a
                        name   = ConceptC2
      userDefinedRelationships
        name = ReferTo
        targetEntities
          Entity
            bsrURI = zac084d6b804
            name   = ConceptZa
            ...
userDefinedRelationships
  name = ReferTo
  targetEntities
    Entity
      bsrURI = zac084d6b804
      name   = ConceptZa
      ...
```

Related concepts:

"RegistryLookup node output: example 1" on page 1901
Example showing the full RegistryLookup node output in two cases for a query that returns two versions of a concept entity. In both cases the `Match Policy` property is set to `All`. In the first case the `Depth Policy` property is set to `Return matched plus immediate related entities (Depth = 1)` , and in the second case the `Depth Policy` property is set to `Return matched only, showing immediate relationships (For compatibility only)`. This example also shows example ESQL to read elements of the output.

"RegistryLookup node output: example 3" on page 1909
Example showing the structure of RegistryLookup node output for a query on an entity having metadata relationships and user-defined relationships, using a `Depth Policy` property value of `Return matched plus all related entities (Depth = -1)` .

"EndpointLookup node output" on page 1894
Use an EndpointLookup node to retrieve the endpoint addresses for WSDL service definitions held in WebSphere Service Registry and Repository (WSRR).

"Dynamically defining the search criteria" on page 1891
You can use the RegistryLookup and EndpointLookup nodes to issue WebSphere Service Registry and Repository (WSRR) queries specified in the local environment.

"Caching artifacts from the WebSphere Service Registry and Repository" on page 1888
WebSphere Message Broker saves the data it retrieves from the WebSphere Service Registry and Repository (WSRR) in a local cache.

Related reference:

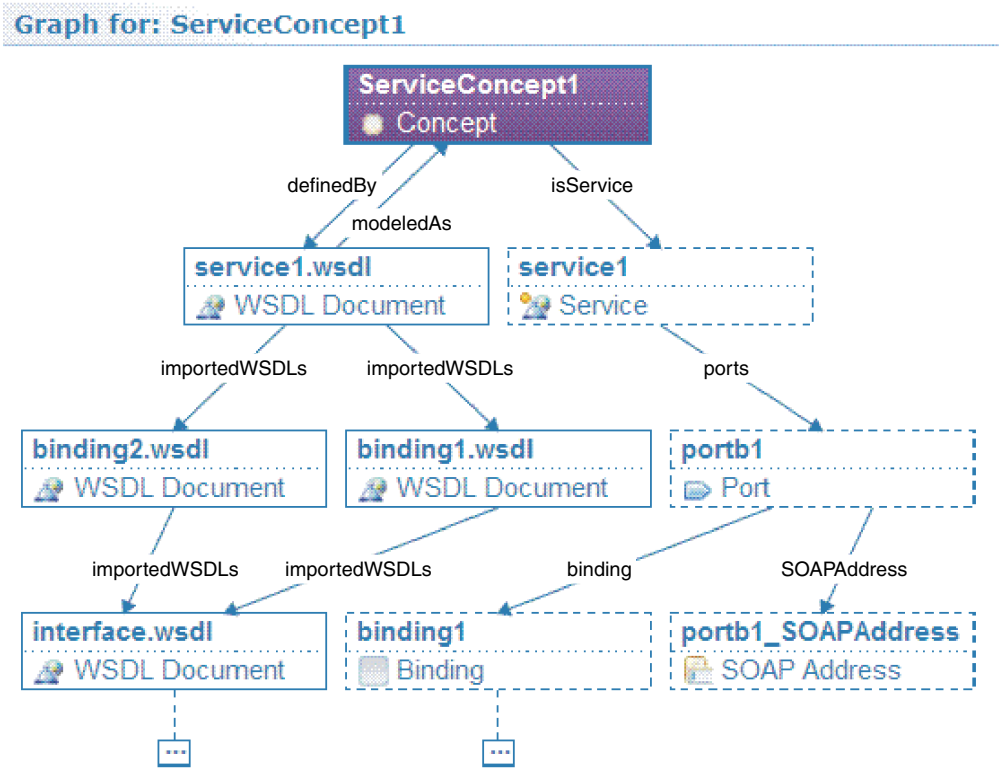
"EndpointLookup node" on page 4407
Use the EndpointLookup node to retrieve service endpoint information held in the WebSphere Service Registry and Repository (WSRR). The entities that match the specified search criteria are stored in the local environment. Depending on the node configuration, the Web service URL destination used by the SOAP and HTTP request nodes can also be set. The input message is not modified.

"RegistryLookup node" on page 4646
Use the RegistryLookup node to retrieve any type of entity held in the WebSphere Service Registry and Repository (WSRR). The entities that match the specified search criteria are stored in the local environment. The input message is not modified.

RegistryLookup node output: example 3:

Example showing the structure of RegistryLookup node output for a query on an entity having metadata relationships and user-defined relationships, using a `Depth Policy` property value of `Return matched plus all related entities (Depth = -1)` .

This example shows the ServiceRegistry message tree that is stored in the LocalEnvironment when the relationships shown in the following WebSphere Service Registry and Repository graph are retrieved using a `Depth Policy` value of `Return matched plus all related entities (Depth = -1)`. The graph has been annotated with the relationship names to clarify the elements in the message tree.



The following ServiceRegistry message tree has some elements replaced by ... to emphasis the structure of the tree.

```
ServiceRegistry
  Entity
    name = ServiceConcept1
    ...
    userDefinedRelationships
      name = modeledAs
      targetEntities
        Entity
          name = service1.wsdl
          ...
          userDefinedRelationships
            name = definedBy
            targetEntities
              EntityRef
                bsrURI = c26e43ac45a
                name = ServiceConcept1
          metaRelationships
            name = importedWSDLs
            targetEntities
              Entity
                name = binding1.wsdl
                ...
              Entity
                name = binding2.wsdl
                ...
    userDefinedRelationships
      name = isService
      targetEntities
        Entity
          name = service1
          ...
          metaRelationships
```

```
name = ports
targetEntities
  Entity
    name = portb1
    ...
    metaRelationships
      name = binding
      targetEntities
        Entity
          name = binding1
          ...
    metaRelationships
      name = SOAPAddress
      targetEntities
        Entity
          name = portb1_SOAPAddress
          ...
```

Related concepts:

"RegistryLookup node output: example 1" on page 1901
Example showing the full RegistryLookup node output in two cases for a query that returns two versions of a concept entity. In both cases the `Match Policy` property is set to `All`. In the first case the `Depth Policy` property is set to `Return matched plus immediate related entities (Depth = 1)` , and in the second case the `Depth Policy` property is set to `Return matched only, showing immediate relationships (For compatibility only)`. This example also shows example ESQL to read elements of the output.

"RegistryLookup node output: example 2" on page 1906
Example showing the structure of RegistryLookup node output for all possible values of the `Depth Policy` property for a query on a concept entity that has a number of user relationships to other concept entities.

"EndpointLookup node output" on page 1894
Use an EndpointLookup node to retrieve the endpoint addresses for WSDL service definitions held in WebSphere Service Registry and Repository (WSRR).

"Dynamically defining the search criteria" on page 1891
You can use the RegistryLookup and EndpointLookup nodes to issue WebSphere Service Registry and Repository (WSRR) queries specified in the local environment.

"Caching artifacts from the WebSphere Service Registry and Repository" on page 1888
WebSphere Message Broker saves the data it retrieves from the WebSphere Service Registry and Repository (WSRR) in a local cache.

Related reference:

"EndpointLookup node" on page 4407
Use the EndpointLookup node to retrieve service endpoint information held in the WebSphere Service Registry and Repository (WSRR). The entities that match the specified search criteria are stored in the local environment. Depending on the node configuration, the Web service URL destination used by the SOAP and HTTP request nodes can also be set. The input message is not modified.

"RegistryLookup node" on page 4646
Use the RegistryLookup node to retrieve any type of entity held in the WebSphere Service Registry and Repository (WSRR). The entities that match the specified search criteria are stored in the local environment. The input message is not modified.

Connecting to Enterprise Information Systems

Use WebSphere Adapters to communicate with Enterprise Information Systems (EIS) such as SAP, Siebel, PeopleSoft, and JD Edwards.

About this task

This section contains the following concept information:
- "WebSphere Broker Adapters Transport" on page 1913
- "WebSphere Adapters nodes" on page 1914
- "Overview of WebSphere Adapter for SAP Software" on page 1917
- "Overview of WebSphere Adapter for Siebel Business Applications" on page 2002
- "Overview of WebSphere Adapter for PeopleSoft Enterprise" on page 2013
- "Overview of WebSphere Adapter for JD Edwards EnterpriseOne" on page 2023

This section contains the following tasks:
- "Developing message flows that use WebSphere Adapters" on page 2033
- "Preparing your system to use WebSphere Adapters nodes" on page 2034
- "Activating IBM Tivoli License Manager for WebSphere Adapters" on page 2036
- "Connecting to an EIS by using the Adapter Connection wizard" on page 2037
- "Configuring EIS connections to expire after a specified time" on page 726
- "Configuring WebSphere Adapters nodes for secondary adapters" on page 2040
- "Calling new services from a WebSphere Adapters request node without changing existing deployed resources" on page 2042
- "Handling new event types from a Websphere Adapters input node without changing existing deployed resources" on page 2044
- "Interacting with an SAP application" on page 2046
- "Interacting with a Siebel application" on page 2066
- "Interacting with a PeopleSoft application" on page 2080
- "Interacting with a JD Edwards application" on page 2088

Related reference:

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPReply node" on page 4682
Use the SAPReply node to send a reply to an SAP synchronous callout. Use this node with an SAPInput node to implement a message flow that acts as a remote function call (RFC) destination.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

"SiebelInput node" on page 4740
Use the SiebelInput node to interact with a Siebel application.

"SiebelRequest node" on page 4745
Use the SiebelRequest node to interact with a Siebel application.

"PeopleSoftInput node" on page 4630
Use the PeopleSoftInput node to interact with a PeopleSoft application.

"PeopleSoftRequest node" on page 4635
Use the PeopleSoftRequest node to interact with a PeopleSoft application.

"JDEdwardsRequest node" on page 4524
Use the JDEdwardsRequest node to interact with a JD Edwards EnterpriseOne server.

"JDEdwardsInput node" on page 4519
Use the JDEdwardsInput node to interact with a JD Edwards EnterpriseOne server.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"WebSphere Adapter for Siebel properties" on page 4092
Reference information to refer to when you connect to a Siebel application.

"WebSphere Adapter for PeopleSoft properties" on page 4122
Reference information to refer to when you connect to a PeopleSoft application.

"WebSphere Adapter for JD Edwards EnterpriseOne properties" on page 4146
Reference information to refer to when you connect to a JD Edwards EnterpriseOne application.

WebSphere Broker Adapters Transport

WebSphere Broker Adapters Transport is a service that connects applications to Enterprise Information Systems (such as SAP Software, PeopleSoft Enterprise, and Siebel Business Application systems).

You can use the WebSphere Broker Adapters Transport to support the following operations:

- Accept input from an SAP, Siebel, or PeopleSoft application.
- Send requests to an SAP, Siebel, JD Edwards, or PeopleSoft application.
- Send a reply to an SAP synchronous callout.
- Route IDocs to separate message flows.

These operations are implement by the following built-in nodes:
- SAPInput node
- SAPRequest node
- SAPReply node
- SiebelInput node
- SiebelRequest node
- PeopleSoftInput node
- PeopleSoftRequest node
- JDEdwardsRequest node

Use the input nodes to accept input from an EIS, along with a matching reply node if the EIS needs a response. Use the request nodes to send requests to an EIS.

To use the WebSphere Broker Adapters Transport, you must deploy a message flow that contains one or more WebSphere Adapters nodes.

The following topics contain more information about working with WebSphere Adapters:

- "Connecting to Enterprise Information Systems" on page 1912
- "WebSphere Adapters nodes" on page 1914
- "Developing message flows that use WebSphere Adapters" on page 2033

Related concepts:

"Nodes for connectivity" on page 1028
WebSphere Message Broker supports direct connections from applications, and can send direct requests to other application endpoints. WebSphere Message Broker can also connect to various subsystems including WebSphere MQ, files, and databases, to read and write existing application data.

Related reference:

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

"SAPReply node" on page 4682
Use the SAPReply node to send a reply to an SAP synchronous callout. Use this node with an SAPInput node to implement a message flow that acts as a remote function call (RFC) destination.

"SiebelInput node" on page 4740
Use the SiebelInput node to interact with a Siebel application.

"SiebelRequest node" on page 4745
Use the SiebelRequest node to interact with a Siebel application.

"PeopleSoftInput node" on page 4630
Use the PeopleSoftInput node to interact with a PeopleSoft application.

"PeopleSoftRequest node" on page 4635
Use the PeopleSoftRequest node to interact with a PeopleSoft application.

"JDEdwardsRequest node" on page 4524
Use the JDEdwardsRequest node to interact with a JD Edwards EnterpriseOne server.

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

WebSphere Adapters nodes

A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

The following terms are associated with WebSphere Adapters:

EIS Enterprise information system. This term is used to describe the applications that form an enterprise's existing system for handling company-wide information. An enterprise information system offers a well-defined set of services that are exposed as local or remote interfaces or both. Enterprise Resource Planning (ERP) and Customer Relationship Management (CRM) are typical enterprise information systems.

EMD Enterprise Metadata Discovery. A specification that you can use to examine an EIS and get details of business object data structures and APIs. An EMD stores definitions as XML schemas by default, and builds components that can access the EIS. In WebSphere Message Broker you use the Adapter Connection wizard to examine an EIS.

Business object
In a development or production environment, a set of XML schema attributes that represents a business entity (such as an invoice) and the definition of actions that can be performed on those attributes (such as the create and update operations).

The WebSphere Adapters support two modes of communication:

- **Inbound**: An event is generated on the EIS and the adapter responds to the event by sending a message to the message broker. The WebSphere Adapters input nodes support inbound communication. When the EIS sends an event to the adapter, a message is propagated from the WebSphere Adapters input node. For example, use an SAPInput node to accept input from an SAP application.
- **Outbound**: The message broker uses the adapter to send a request to the EIS. The WebSphere Adapters request nodes support outbound communication.

When a message is propagated to the WebSphere Adapters request node, the adapter sends a request to the EIS. For example, use an SAPRequest node to send requests to an SAP application.

The WebSphere Adapters nodes need an adapter component to access the EIS. The input nodes need an inbound adapter component, which allows the EIS to invoke the message flow when an event occurs. The request nodes need an outbound adapter component, which is used by the message flow to invoke a service in the EIS.

The WebSphere Adapters nodes also need a message set to ensure that the WebSphere Message Broker messages that are propagated to and from the nodes reflect the logical structure of the data in the EIS.

SAP, Siebel, JD Edwards, and PeopleSoft adapters are supported by the following message flow nodes in WebSphere Message Broker:

- SAPInput node
- SAPRequest node
- SAPReply node
- SiebelInput node
- SiebelRequest node
- PeopleSoftInput node
- PeopleSoftRequest node
- JDEdwardsRequest node
- JDEdwardsInput node

The TwineballInput and TwineballRequest nodes are sample nodes with their own sample EIS. You can use the Twineball nodes to see how adapters nodes work. You cannot use the Twineball nodes to connect to the external SAP, Siebel, and PeopleSoft EIS systems.

You can configure the Websphere Adapters nodes by using properties on the nodes, or by using a configurable service. For example, you can use a configurable service to specify the connection details for the EIS. For more information, see “Configurable services properties” on page 3766.

To effectively maintain the pool of connections to the EIS, you can set a connection timeout value on a configurable service. For more information, see “Configuring EIS connections to expire after a specified time” on page 726.

The SAPRequest node can also use an identity that is present on an input message, and propagate it to SAP, by using the `Propagate` property on the security profile that is defined on the node. For more information, see “Identity and security token propagation” on page 426.

The mode in which your broker is working can affect the number of execution groups and message flows that you can deploy, and the type of node that you can use. For example, in Remote Adapter Deployment mode, only adapter-related features are enabled, and the types of node that you can use, and the number of execution groups that you can create, are limited. For more information about the available modes of operation, see “Operation modes” on page 48.

For more information about support for adapters on different operating systems, see WebSphere Message Broker Requirements.

The following topics provide an overview of the WebSphere Adapters:

- "Overview of WebSphere Adapter for SAP Software" on page 1917
- "Overview of WebSphere Adapter for Siebel Business Applications" on page 2002
- "Overview of WebSphere Adapter for PeopleSoft Enterprise" on page 2013
- "Overview of WebSphere Adapter for JD Edwards EnterpriseOne" on page 2023

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

Related reference:

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"WebSphere Adapter for Siebel properties" on page 4092
Reference information to refer to when you connect to a Siebel application.

"WebSphere Adapter for PeopleSoft properties" on page 4122
Reference information to refer to when you connect to a PeopleSoft application.

"WebSphere Adapter for JD Edwards EnterpriseOne properties" on page 4146
Reference information to refer to when you connect to a JD Edwards EnterpriseOne application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

"SAPReply node" on page 4682
Use the SAPReply node to send a reply to an SAP synchronous callout. Use this node with an SAPInput node to implement a message flow that acts as a remote function call (RFC) destination.

"SiebelInput node" on page 4740
Use the SiebelInput node to interact with a Siebel application.

"SiebelRequest node" on page 4745
Use the SiebelRequest node to interact with a Siebel application.

"PeopleSoftInput node" on page 4630
Use the PeopleSoftInput node to interact with a PeopleSoft application.

"PeopleSoftRequest node" on page 4635
Use the PeopleSoftRequest node to interact with a PeopleSoft application.

"JDEdwardsRequest node" on page 4524
Use the JDEdwardsRequest node to interact with a JD Edwards EnterpriseOne server.

"JDEdwardsInput node" on page 4519
Use the JDEdwardsInput node to interact with a JD Edwards EnterpriseOne server.

"TwineballInput node" on page 4951
Use the TwineballInput node to discover how the WebSphere Adapters nodes

work.

"TwineballRequest node" on page 4955
Use the TwineballRequest node to discover out how WebSphere Adapters nodes work.

Overview of WebSphere Adapter for SAP Software:

With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

By using the adapter, an application component (the program or piece of code that performs a specific business function) can send requests to the SAP server (for example, to query a customer record in an SAP table or to update an order document) or receive events from the server (for example, to be notified that a customer record has been updated). The adapter creates a standard interface to the applications and data on the SAP server so that the developer of the application component does not have to understand the lower-level details (the implementation of the application or the data structures) on the SAP server.

WebSphere Adapter for SAP Software complies with the Java Connector Architecture (JCA) 1.5, which standardizes the way in which application components, application servers, and Enterprise Information Systems (EIS), such as an SAP server, interact with each other.

The adapter, which you generate with the Adapter Connection wizard, uses a standard interface and standard data objects. The adapter takes the standard data object sent by the application component and calls the SAP function. The adapter then returns a standard data object to the application component. The application component does not have to deal directly with the SAP function; it is the SAP adapter that calls the function and returns the results.

For example, the application component that requested the list of customers sends a standard business object with the range of customer IDs to the SAP adapter. The application component receives, in return, the results (the list of customers) in the form of a standard business object. The adapter completes all the interactions directly with the SAP function.

Similarly, the message flow might want to know about a change to the data on the SAP server (for example, a change to a particular customer). You can generate an adapter component that listens for such events on the SAP server and notifies message flows with the update. In this case, the interaction begins at the SAP server.

For more information, see "Technical overview of Adapter for SAP Software" on page 1918.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System

(EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

Technical overview of Adapter for SAP Software:

WebSphere Adapter for SAP Software provides multiple ways to interact with applications and data on SAP servers. Outbound processing (from an application to the adapter to the SAP server) and inbound processing (from the SAP server to the adapter to an application) are supported.

WebSphere Adapter for SAP Software connects to SAP systems running on SAP Web application servers. The adapter supports Advanced Event Processing (AEP) and Application Link Enabling (ALE) for inbound processing, and the Business Application Programming Interface (BAPI), AEP, ALE, and Query Interface for SAP Systems (QISS) for outbound processing. You set up the adapter to perform outbound and inbound processing by using the Adapter Connection wizard to generate business objects based on the services it discovers on the SAP server.

For outbound processing, the adapter client invokes the adapter operation to create, update, or delete data on the SAP server or to retrieve data from the SAP server.

For inbound processing, an event that occurs on the SAP server is sent from the SAP server to the adapter. The ALE inbound and BAPI inbound interfaces start event listeners that detect the events. Conversely, the Advanced event processing interface polls the SAP server for events. The adapter then delivers the event to an endpoint, which is an application or other consumer of the event from the SAP server.

You configure the adapter to perform outbound and inbound processing by using the Adapter Connection wizard to create a message set project that includes the interface to the SAP application as well as business objects based on the functions or tables that it discovers on the SAP server.

Overview of the outbound processing interfaces

WebSphere Adapter for SAP Software provides multiple interfaces to the SAP server for outbound processing.

- Through its BAPI interfaces, the adapter issues remote function calls (RFCs) to RFC-enabled functions, such as a Business Application Programming Interface (BAPI) function. These remote function calls create, update, or retrieve data on an SAP server.
  - The BAPI interface works with individual BAPIs (simple BAPIs). For example, you might want to check to see whether specific customer information exists in an SAP database.
  - The BAPI work unit interface works with ordered sets of BAPIs. For example, you might want to update an employee record. To do so, you use three BAPIs:
    1. To lock the record (to prevent any other changes to the record)
    2. To update the record
    3. To have the record approved
  - The BAPI result set interface uses two BAPIs to select multiple rows of data from an SAP database.

  BAPI calls are useful when you need to perform data retrieval or manipulation and a BAPI or RFC function that performs the task already exists.

  Simple BAPIs can be sent through the synchronous RFC, asynchronous transactional RFC, or asynchronous queued RFC protocol.
  - With synchronous RFC, both the adapter and the SAP server must be available when the call is made from the adapter to the SAP server. The adapter sends a request to the SAP server and waits for a response.
  - With asynchronous transactional RFC, a transaction ID is associated with the call from the adapter to the SAP server. The adapter does not wait for a response from the SAP server. Only the transaction ID is returned to the message flow.
  - With asynchronous queued RFC, the call from the adapter is delivered to a predefined queue on the SAP server. As with asynchronous RFC, a transaction ID is associated with the call, and the adapter does not wait for a response from the SAP server.

    This interface is useful when the event sequence must be preserved.
- The Query interface for SAP Software retrieves data from specific SAP application tables. It can return the data or check for the existence of the data. You can use this type of interaction with SAP if you need to retrieve data from an SAP table without using an RFC function or a BAPI.

- With the Application Link Enabling (ALE) interface, you exchange data using SAP Intermediate Data structures (IDocs). For outbound processing, you send an IDoc or a packet of IDocs to the SAP server.

  The ALE interface, which is particularly useful for batch processing of IDocs, provides asynchronous exchange. You can use the queued transactional (qRFC) protocol to send the IDocs to a queue on the SAP server. The qRFC protocol ensures the order in which the IDocs are received. It is often used for system replications or system-to-system transfers.
- With the ALE pass-through IDoc interface, the adapter sends the IDoc to the SAP server with no conversion of the IDoc. The message tree contains a BLOB field that represents the IDoc.
- With the Advanced event processing interface, you send data to the SAP server. The data is then processed by an ABAP handler on the SAP server.

Overview of the inbound processing interfaces

WebSphere Adapter for SAP Software provides the following interfaces to the SAP server for inbound processing.

- Through its BAPI inbound interface, the adapter listens for events and receives notifications of RFC-enabled function calls from the SAP server.
  - With synchronous RFC, both the adapter and the SAP server must be available when the call is made from the SAP server to the adapter. The adapter sends the request to a predefined application and returns the response to the SAP server.
  - With asynchronous transactional RFC, the event is delivered to the adapter even if the adapter is not available when the call is made. The SAP server stores the event on a list of functions to be invoked and continues to attempt to deliver it until the adapter is available.

    You also use asynchronous transaction RFC if you want to deliver the functions from a predefined queue on the SAP server. Delivering the files from a queue ensures the order in which the functions are sent.

    If you select assured once-only delivery, the adapter uses a data source to persist the event data received from the SAP server. Event recovery is provided to track and recover events in case a problem occurs when the adapter attempts to deliver the event to the endpoint.
- With the ALE inbound processing interface, the adapter listens for events and receives one or more IDocs from the SAP server. As with ALE outbound processing, ALE inbound processing provides asynchronous exchange.

  You can use the qRFC interface to receive the IDocs from a queue on the SAP server, which ensures the order in which the IDocs are received.

  If you select assured once-only delivery, the adapter uses a data source to persist the event data, and event recovery is provided to track and recover events in case a problem occurs when the adapter attempts to deliver the event to the endpoint.
- With the ALE pass-through IDoc interface, the SAP server sends the IDoc through the adapter to the endpoint with no conversion of the IDoc. The message tree contains a BLOB field that represents the IDoc.
- The Advanced event processing interface polls the SAP server for events. It discovers events waiting to be processed. It then processes the events and sends them to the endpoint. For more information, see "The Advanced event processing interface" on page 1987.

How the adapter interacts with the SAP server

The adapter uses the SAP Java Connector (SAP JCo) API to communicate with SAP applications. An application sends a request to the adapter, which uses the SAP JCo API to convert the request into a BAPI function call. The SAP system processes the request and sends the results to the adapter. The adapter sends the results in a response message to the calling application.

For more information, see the following topics.

- "The Adapter Connection wizard (SAP)" on page 1922
- "The BAPI interfaces" on page 1923
- "The ALE interfaces" on page 1959
- "Query interface for SAP Software" on page 1982
- "The Advanced event processing interface" on page 1987

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

The Adapter Connection wizard (SAP):

The Adapter Connection wizard is a tool that you use to create services. The Adapter Connection wizard establishes a connection to the SAP server, discovers services (based on search criteria that you provide), and generates business objects, interfaces, and import or export files, based on the services that are discovered.

By using WebSphere Message Broker, you establish a connection to the SAP server to browse the metadata repository on the SAP server. The SAP metadata repository, which is a database of the SAP data, provides a consistent and reliable means of access to that data.

You specify connection information (such as the user name and password needed to access the server), and you specify the interface that you want to use (for example, BAPI). The service metadata that is associated with that interface is displayed. You can then provide search criteria and select the information (for example, you can list all BAPIs that relate to "CUSTOMER" by using the search filter with "BAPI_CUSTOMER*", then select one or more BAPIs).

The result of running the Adapter Connection wizard is an adapter connection project and a message set project that contain the interfaces and business objects as well as the adapter.

The Adapter Connection wizard also produces an import file (for outbound processing) or an export file (for inbound processing).

- The import file contains the managed connection factory property settings that you provide in the wizard.
- The export file contains the activation specification property settings you provide in the wizard.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

The BAPI interfaces:

The WebSphere Adapter for SAP Software supports outbound processing for simple BAPIs, BAPI units of work, and BAPI result sets. In outbound processing, message flows call BAPIs and other RFC-enabled functions on the SAP server. The adapter supports inbound processing for simple BAPIs only. In inbound processing, the SAP server sends an RFC-enabled function (such as a BAPI function) through the adapter to an endpoint.

For example, you want to build a service that creates a new customer on the SAP server. You run the Adapter Connection wizard to discover the BAPI_CUSTOMER_CREATEFROMDATA function, and the wizard generates the business-object definition for BAPI_CUSTOMER_CREATEFROMDATA, as well as other Service Component Architecture (SCA) service resources. During BAPI outbound processing, the adapter receives the service request and converts the data into a BAPI invocation.

BAPI interface (simple BAPIs)

A simple BAPI performs a single operation, such as retrieving a list of customers. The adapter supports simple BAPI calls by representing each with a single business object schema.

Simple BAPIs can be used for outbound or inbound processing. You can specify synchronous RFC processing or asynchronous transactional RFC (tRFC) processing when you configure a module for a simple BAPI. In addition, for outbound processing, you can specify asynchronous queued RFC (qRFC) processing, in which BAPIs are delivered to a predefined queue on the SAP server.

- In synchronous RFC processing, the SAP server and the adapter must be available during processing.
  - In outbound processing, the message flow sends a request, then waits for a response from the SAP server.
  - In inbound processing, the SAP server sends a request through the adapter to an endpoint and waits for a response from the adapter.
- In asynchronous tRFC outbound processing, the adapter associates a transaction ID with the function call to the SAP server. The adapter does not wait for a response from the SAP server. If the delivery is unsuccessful, the message flow can use the SAP transaction ID (TID) to make the request again. The TID is a field in your message.
- In asynchronous tRFC inbound processing, the adapter does not have to be available when the SAP server runs the function call. The function call is placed on a list of functions to be invoked, and the call is attempted until it is successful.

  To send function calls from a user-defined outbound queue on the SAP server, you also specify asynchronous tRFC inbound processing.
- In asynchronous qRFC outbound processing, the process is similar to asynchronous tRFC outbound processing. A TID is associated with the function call, and the adapter does not wait for a response from the SAP server. In addition, the BAPIs are delivered to a predefined queue on the SAP server. By sending BAPIs to the predefined queue, you can ensure the order in which they are delivered.

BAPI work unit interface

A BAPI work unit consists of a set of BAPIs that are processed in sequence to complete a task. For example, to update an employee record in the SAP system, the record needs to be locked before being updated. This task is accomplished by calling three BAPIs, in sequence, in the same work unit. The following three BAPIs illustrate the kind of sequence that forms such a unit of work:
- BAPI_ADDRESSEMP_REQUEST
- BAPI_ADDRESSEMP_CHANGE
- BAPI_ADDRESSEMP_APPROVE

The first BAPI locks the employee record, the second updates the record, and the third approves the update. The advantage of using a BAPI unit of work is that the message flow can request the employee record change with a single call, even though the work unit consists of three separate functions. In addition, if SAP requires that the BAPIs be processed in a specific sequence for the business flow to complete correctly, the work unit supports this sequence.

BAPI result set interface

BAPI result sets use the GetList and GetDetail functions to retrieve an array of data from the SAP server. The information that is returned from the GetList function is used as input to the GetDetail function.

For example, if you want to retrieve information on a set of customers, you use BAPI_CUSTOMER_GETLIST, which acts as the query BAPI, and BAPI_CUSTOMER_GETDETAIL, which acts as the result BAPI. The BAPIs perform the following steps:

1. The BAPI_CUSTOMER_GETLIST call returns a list of keys (for example, CustomerNumber).

2. Each key is mapped dynamically to the business object for BAPI_CUSTOMER_GETDETAIL.
3. BAPI_CUSTOMER_GETDETAIL is processed multiple times, so that an array of customer information is returned.

You use the Adapter Connection wizard to discover the BAPI_CUSTOMER_GETLIST and BAPI_CUSTOMER_GETDETAIL functions and build the key relationship between the two BAPIs. The wizard then generates business object definitions for these BAPIs as well as other SCA service resources. At run time, the client sets the values in the BAPI_CUSTOMER_GETLIST business object, and the adapter returns the corresponding set of customer detail records from the SAP server.

For more information, see the following topics.

- "Outbound processing for the BAPI interface" on page 1926
- "Business objects for the BAPI interface" on page 1952

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

Outbound processing for the BAPI interface:

In BAPI outbound processing, a message flow sends a request to the SAP server. For BAPI units of work and BAPI result sets, processing is handled synchronously (the message flow waits for a response from the SAP server). For simple BAPIs, you can request that processing be handled synchronously or asynchronously (the message flow does not wait for a response from the SAP server).

For BAPI units of work and BAPI result sets, the processing is handled as described in "Synchronous RFC." For simple BAPIs, you make a selection, during configuration, about the type of remote RFC call you want to make.

Synchronous RFC

If you select **Synchronous RFC** (the default) during configuration for a simple BAPI, or if you are using BAPI units of work or BAPI result sets, the following processing steps occur:

1. The adapter receives a request from a message flow in the form of a BAPI business object.
2. The adapter converts the BAPI business object to an SAP JCo function call.
3. The adapter uses the Remote Function Call (RFC) interface to process the BAPI or RFC function call in the SAP application.
4. After passing the data to the SAP server, the adapter handles the response from SAP and converts it back into the business object format required by the message flow.
5. The adapter then sends the response back to the message flow.

Asynchronous transactional RFC

If you select **Asynchronous transactional RFC** during configuration, the following processing steps occur:

1. The adapter receives a request from a message flow in the form of a BAPI business object.
2. The adapter checks the business object to see whether the SAP transaction ID attribute has a value assigned. (The SAP transaction ID (TID) is a field in your message.)
   - If the SAP transaction ID attribute has a value, the adapter uses that value during processing.
   - If the attribute does not have a value, the adapter makes a call to the SAP server and gets a transaction ID from the SAP server.
3. The adapter converts the BAPI business object to an SAP JCo function call.
4. The adapter uses the transactional Remote Function Call (tRFC) protocol to make the call to the SAP server.

The adapter does not wait for a response from the SAP server.

5. After the function data is passed to the SAP application, control returns to the adapter.
   - If the call to the SAP server fails, the SAP server throws an ABAPException.
   - If the call to the SAP server succeeds but contains invalid data, no exception is returned to the adapter. For example, if the adapter sends a request that contains an invalid customer number, the adapter does not respond with an exception indicating that no such customer exists.
6. The request node builds a message tree that contains the transaction ID as one of the fields.

Asynchronous queued RFC

If you select **Asynchronous queued RFC** during configuration, the following processing steps occur:

1. The adapter receives a request from a message flow in the form of a BAPI business object.
2. The adapter checks the business object to see whether the SAP transaction ID attribute has a value assigned. (The SAP transaction ID (TID) is a field in your message.)
   - If the SAP transaction ID attribute has a value, the adapter uses that value during processing.
   - If the attribute does not have a value, the adapter makes a call to the SAP server and gets a transaction ID from the SAP server.
3. The adapter converts the BAPI business object to an SAP JCo function call.
4. The adapter uses the tRFC protocol to make the call to the specified queue on the SAP server.

   The adapter does not wait for a response from the SAP server.
5. After the function data is passed to the SAP application, control returns to the adapter.
   - If the call to the SAP server fails, the SAP server throws an ABAPException.
   - If the call to the SAP server succeeds but contains invalid data, no exception is returned to the adapter. For example, if the adapter sends a request that contains an invalid customer number, the adapter does not respond with an exception indicating that no such customer exists.
6. The request node builds a message tree that contains the transaction ID as one of the fields.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

"SAP BAPI transaction commit" on page 1928
When the SAP adapter is used with the BAPI interface, you must consider certain factors when you design transactional flows.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

SAP BAPI transaction commit:

When the SAP adapter is used with the BAPI interface, you must consider certain factors when you design transactional flows.

You can configure message flows to be transactional so that updates to resources such as databases can be coordinated; changes are committed or rolled back together within the same transaction. This transactional coordination can be extended to external system updates, such as SAP databases, when you use the BAPI interface with SAPRequest nodes.

The SAP adapter can control whether it waits for SAP to commit the updates synchronously, or issues a commit and returns while the SAP commit happens asynchronously. You can determine this behavior by using the **Use wait parameter before calling BAPI commit** parameter on the Configure Objects pane of the Adapter Connection wizard. The adapter relies on the transactionality setting of the message flow to determine whether to issue the commit call.

BAPIs with implicit commit

In earlier releases of SAP, some BAPIs were coded with a commit. From SAP Release 4.0A onwards, it is more effective for BAPIs to issue a separate

BAPI_TRANSACTION_COMMIT to force the update, instead of doing commit work. By using this method, BAPI calls can be made before the work is committed as a batched unit of work. To find out if a BAPI is coded with a commit, see the documentation for the BAPI.

Message flow transactionality

When the `Transaction mode` property on the SAPRequest node is set to `Yes`, the adapter is instructed to issue the SAP commit on completion of the message flow in line with other database commits. You can set the **`Use wait parameter before calling BAPI commit`** parameter in the Adapter Connection wizard that determines whether the commit is synchronous or asynchronous.

If the `Transaction mode` property on the SAPRequest node is set to `No`, the adapter does not issue an SAP commit and the parameter that you set on the Adapter Connection wizard has no relevance. However, the commit can still be issued as part of a BAPI work unit COMMIT verb (to which the property on the wizard does apply) or a call to the BAPI_TRANSACTION_COMMIT (to which the property on the wizard does not apply).

The following rules apply when you set the `Transaction mode` property on the SAPRequest node.

- Set `Transaction mode` to `No` if the following conditions apply:
  - The BAPIs already have commits
  - A BAPI_TRANSACTION_COMMIT is called by an SAPRequest node
  - A BAPI work unit includes a BAPI_TRANSACTION_COMMIT or the COMMIT verb is added on the Configure Objects pane of the Adapter Connection wizard

  If the BAPIs are coded with commits and you set `Transaction mode` to `Yes`, the BAPI is called as part of the same transaction as those from other SAPRequest nodes in the same flow and using the same adapter. Therefore, any BAPIs that were called previously in this message flow are committed.
- Set `Transaction mode` to `Yes` if the following conditions apply:
  - The BAPI needs to be committed (that is, the BAPI is not coded with a commit)
  - The BAPI work unit needs to be committed and does not include a BAPI_TRANSACTION_COMMIT or the COMMIT verb

  If you set `Transaction mode` to `No`, the BAPI is not committed now or at the end of the message flow; it is not guaranteed ever to be committed.

The following scenarios illustrate the visibility of the updates made to an SAP system, and show how to use the adapter to avoid uncertainty when data is being committed by an external system.

- Scenario 1: Business partner and relationship processing in a single flow
- Scenario 2: Order create and query application processing with two flows

Related concepts:

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

Scenario 1: Business partner and relationship processing in a single flow:

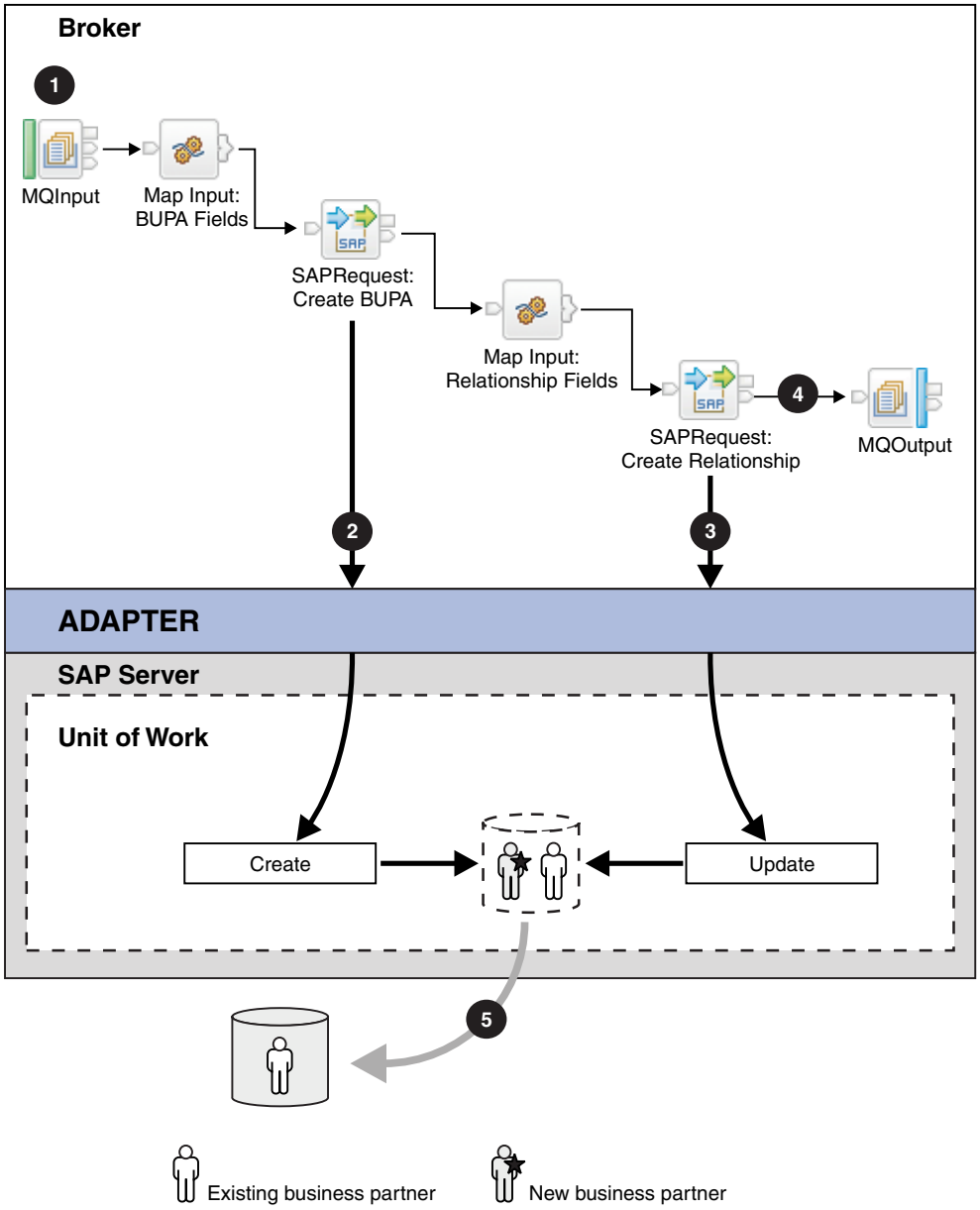
You must set the `Transaction mode` property appropriately on an SAPRequest node when you are processing in a single message flow.

This scenario is one of two examples that illustrate the concepts that are described in "SAP BAPI transaction commit" on page 1928; see also "Scenario 2: Order create and query application processing with two flows" on page 1933.

In this scenario, a message flow is used to create a business partner, and a new relationship with an existing partner by using two BAPI calls:

```
BAPI_BUPA_CREATE_FROM_DATA
BAPI_BUPR_RELATIONSHIP_CREATE
```

The message flow consists of two SAPRequest nodes with the `Transaction mode` property set to `Yes` on both nodes to allow commit or rollback in case of exceptions. When the `Transaction mode` property is set to `Yes`, the message flow's final commit takes place at the end of the flow when the adapter requests SAP to commit the order.



1. An application triggers the transactional flow that creates the business partner.
2. The SAPRequest node submits a BUPA creation and returns the business partner number. The commit happens when the message flow completes because the node participates in a message flow level transaction.
3. The second SAPRequest node attempts to create a relationship between an existing business partner and the new business partner; however, SAP has not yet committed the creation of the new business partner to the database.

   If the same adapter is used for both BAPIs, the adapter ensures a single connection to SAP because both nodes have to participate in the same logical work unit. The single connection means that the BUPA creation is visible to the relationship update call (3 in the diagram), even though the flow transactionality has yet to initiate the commit.

   If the `Transaction mode` property were set to `Yes` on the create BUPA call, but `No` on the create relationship call, the adapter would need to use two different connections to SAP; that is, the transactional properties of the connections

would be different. The create relationship call would therefore fail because the new business partner would not be visible until the message flow and the transactional commit have completed.

4. The MQOutput node puts an MQ message on the output queue pending transactional commit.
5. The message flow completes and the broker begins to commit all the resources involved in that flow, including SAP (5 in the diagram). The updates are committed in SAP.

This scenario illustrates the ability of the broker to use its transactional control of the message flow to provide the necessary information to the SAPRequest nodes to carry out related processing, even though the external SAP system is committing the work asynchronously.

Related concepts:

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

"SAP BAPI transaction commit" on page 1928
When the SAP adapter is used with the BAPI interface, you must consider certain factors when you design transactional flows.

"Scenario 2: Order create and query application processing with two flows" on page 1933
You must set the `Transaction mode` property appropriately on an SAPRequest node when you are processing by using separate message flows.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

Managed connection factory properties (SAP)
The adapter uses the managed connection factory properties at run time to create an outbound connection instance with the SAP server.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

Scenario 2: Order create and query application processing with two flows:

You must set the `Transaction mode` property appropriately on an SAPRequest node when you are processing by using separate message flows.

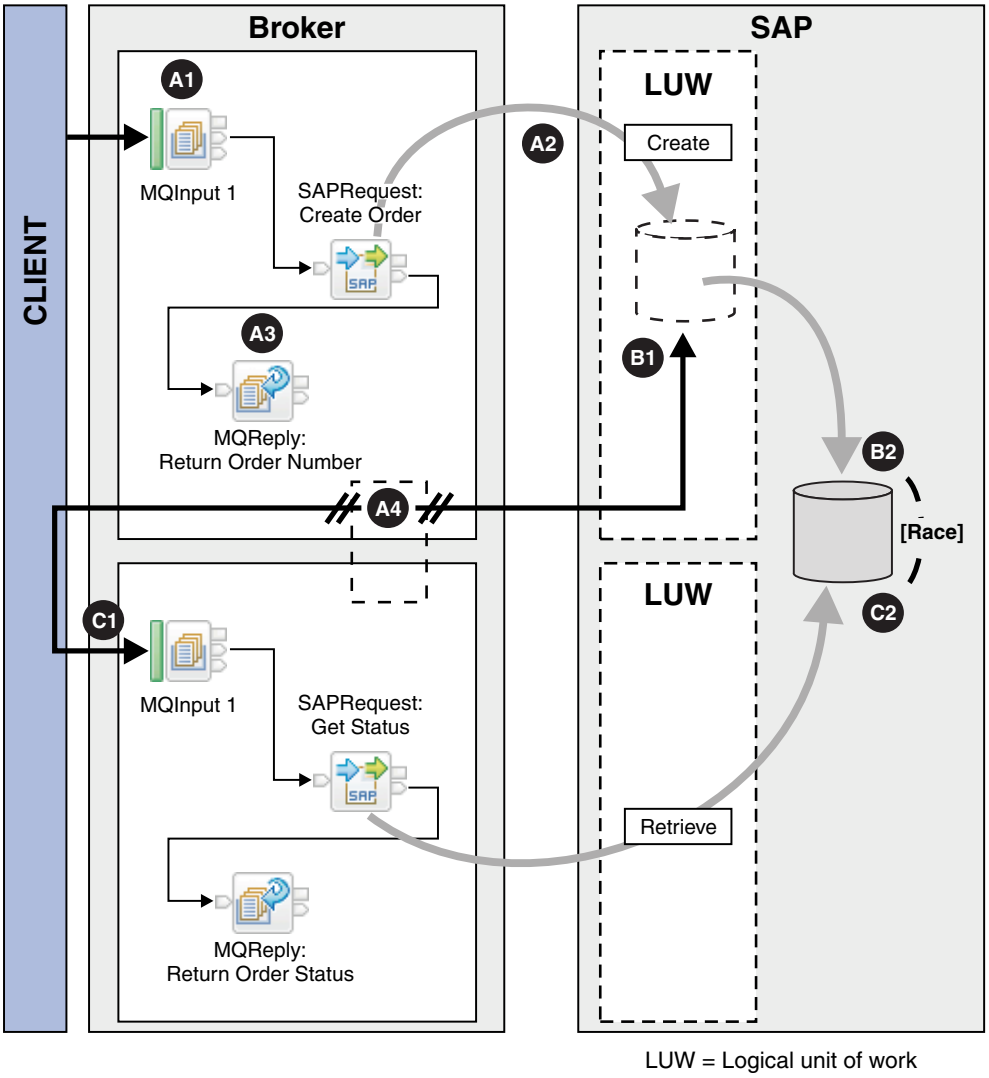
This scenario is one of two examples that illustrate the concepts that are described in "SAP BAPI transaction commit" on page 1928; see also "Scenario 1: Business partner and relationship processing in a single flow" on page 1930.

In this scenario, two message flows are used to issue a sales order create and a subsequent sales order check by using two BAPI calls:

```
BAPI_SALESORDER_CREATEFROMDAT2
BAPI_SALESORDER_GETSTATUS
```

For example, a user queries an order after a purchase has been made by using a Web-based application. The result of the query is closely linked to the asynchronous or synchronous behavior of the order creation steps carried out by the external SAP server.

In the following asynchronous example (the default behavior), the query might fail and the user receives a negative acknowledgment for the order that has been created.



BAPI Create Order message flow

A1. An application triggers the transactional message flow that creates a sales order.

A2. The SAPRequest node submits an order creation and returns the order registration number. The commit happens when the message flow completes because the node participates in a message flow level transaction.

A3. The MQReply node puts an MQ message on the output queue pending transactional commit.

A4. The message flow completes and the broker begins to commit all the resources involved in that flow, including SAP and the MQReply node call. The order number is available to the user application.

The following two processes occur simultaneously, and effectively race each other to complete.

SAP Commits processing asynchronously	**BAPI Get Order Status message flow**

BAPI Create Order message flow

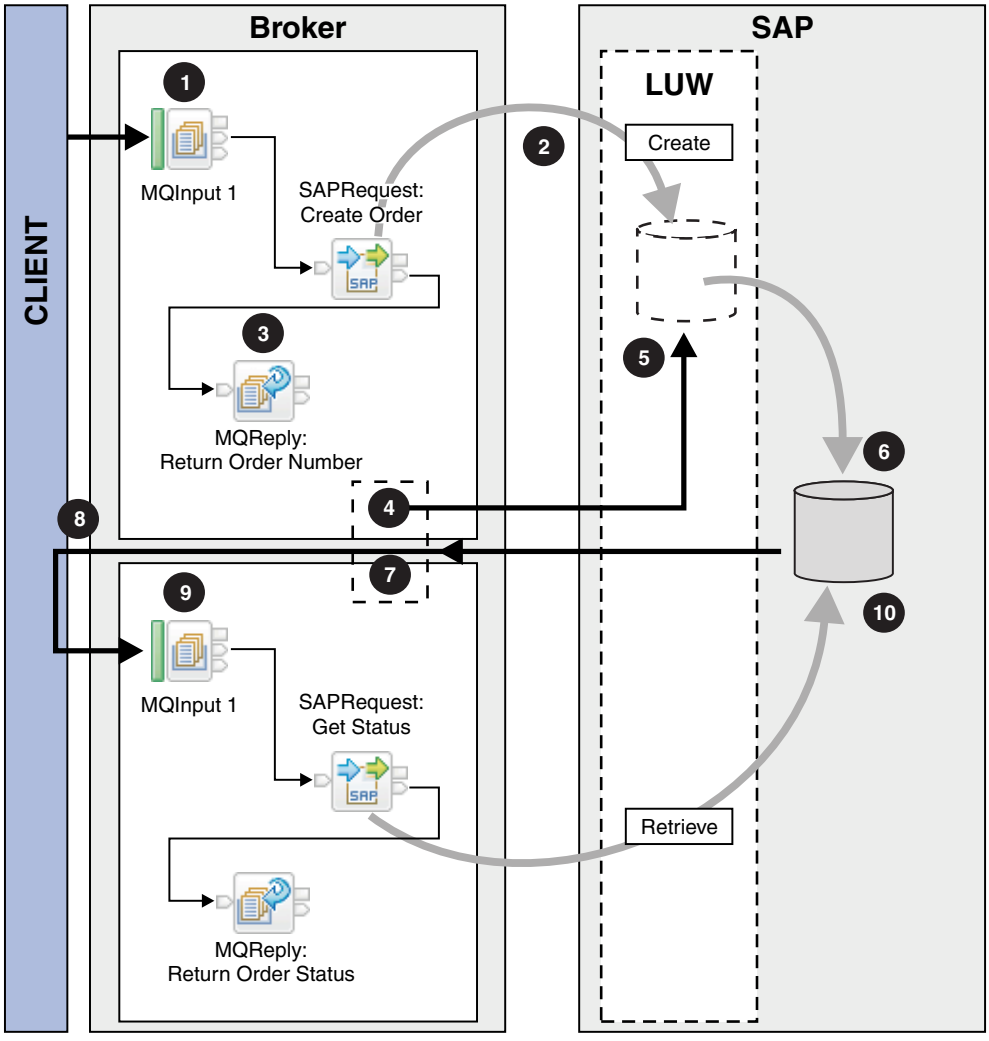
B1. The SAP commit begins.
B2. The SAP commit completes.
C1. A request for an order status query is made.
C2. The SAPRequest node requests the order.

Because of the asynchronous commit, two outcomes are possible when the order is queried:

- The order is not found because SAP has not completed the order commit.
- The order is found but only if SAP has committed the order before the query is made.

You can avoid this uncertainty by configuring the adapter to perform the commit synchronously; set the **Use wait parameter before calling BAPI commit** parameter on the adapter connection wizard to `True`, and set the `Transaction mode` property on the SAPRequest node to `Yes`.

In the following synchronous example, the query is successful and the user receives a positive acknowledgment for the order that has been created.



BAPI Create Order message flow

1. An application triggers the transactional message flow that creates the sales order.
2. The SAPRequest node submits an order creation and returns the order registration number. The commit happens when the message flow completes because the node participates in the a message flow level transaction.
3. The MQReply node puts an MQ message on the output queue pending transactional commit.
4. The message flow completes and the broker begins to commit all the resources involved in that flow, including SAP.

SAP Commits processing synchronously

5. The SAP commit begins.
6. The SAP commit completes.
7. The adapter hands control back to the broker.
8. The MQReply node call is committed, therefore the order number is available to the user application.

BAPI Get Order Status message flow

9. A request for an order status query is made.
10. The SAPRequest node requests the order.

SAP has completed the order commit; therefore, the order query is successful.

Related concepts:

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

"SAP BAPI transaction commit" on page 1928
When the SAP adapter is used with the BAPI interface, you must consider certain factors when you design transactional flows.

"Scenario 1: Business partner and relationship processing in a single flow" on page 1930
You must set the `Transaction mode` property appropriately on an SAPRequest node when you are processing in a single message flow.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

Managed connection factory properties (SAP)
The adapter uses the managed connection factory properties at run time to create an outbound connection instance with the SAP server.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

Inbound processing for the BAPI interface:

The adapter supports inbound processing (from the SAP server to the adapter) for simple BAPIs.

A client application on the SAP server invokes a function through the adapter to an end point.

For more information, see the following topics:
- "Synchronous and asynchronous RFC" on page 1938
- "Event recovery for the BAPI interface" on page 1941
- "Passing parameters and reporting errors" on page 1942
- "BAPI inbound scenarios" on page 1943
- "SAP adapter scalability and performance" on page 1949

Related concepts:

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the

WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Tuning the SAP adapter for scalability and performance" on page 3278
You can configure the number of listeners on the adapter and the number of additional instances on the message flow to prevent delays when processing synchronous calls from SAP.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

"SAPReply node" on page 4682
Use the SAPReply node to send a reply to an SAP synchronous callout. Use this node with an SAPInput node to implement a message flow that acts as a remote function call (RFC) destination.

Synchronous and asynchronous RFC:

For BAPI inbound and outbound processing, you can specify that the processing be handled synchronously (in which both the message flow and the adapter must be available during processing) or asynchronously (in which the adapter does not have to be available when the message flow makes the function call). In synchronous processing, the message flow waits for a response from the adapter. In asynchronous processing, the SAP application does not wait for a response and the adapter does not have to be available when the SAP application makes the function call.

For diagrams that illustrate synchronous and asynchronous RFC, see "BAPI inbound scenarios" on page 1943.

The BAPI interface has two sets of activation specification properties (one for synchronous RFC and one for asynchronous RFC), which you use to set up inbound processing. You specify values for the properties with the Adapter Connection wizard.

The sequence of processing actions that result from an inbound request differ, depending on the selection you make during configuration from the **SAP Remote Function Call (RFC) type** list.

Synchronous RFC

If you select **Synchronous RFC** (the default) during configuration, the following processing steps occur:

1. The adapter starts event listeners, which listen for an RFC-enabled function event (which you specified with the RFCProgramID property) on the SAP server.

2. The RFC-enabled function event is pushed to the adapter by way of the event listeners.
3. The adapter resolves the operation and business object name using the received RFC-enabled function name.
4. The adapter sends the business object to an endpoint in a synchronous manner.
5. The adapter receives the response business object from the endpoint.
6. The adapter maps the response business object to an RFC-enabled function and returns it to the SAP server.

The adapter does not listen for events until the endpoint is active and available.

Asynchronous transactional RFC

If you select **Asynchronous Transactional/Queued RFC** during configuration, the following processing steps occur:

1. A client on the SAP server invokes an RFC-enabled function call on the adapter.

   Note: To send the RFC-enabled functions from a queue on the SAP server, the client program on the SAP server delivers the events to a user-defined outbound queue.

   A transaction ID is associated with the call.

   The calling program on the SAP server does not wait to see whether the call to the adapter was successful, and no data is returned to the calling program.
2. The RFC-function call is placed on a list of functions to be delivered.

   You can see the event list by entering transaction code SM58 on the SAP server
3. The RFC-function call is invoked on the adapter. If the adapter is not available, the call remains in the list on the SAP server, and the call is repeated at regular intervals until it can be processed by the adapter.

   When the SAP server successfully delivers the call event, it removes the function from the list.
4. If you selected **Ensure once-only event delivery**, the adapter sets the transaction ID in the event persistent table.

   This is to ensure the event is not processed more than once.
5. The adapter resolves the operation and business object name using the received RFC-enabled function name.
6. The adapter sends the business object to an endpoint.

   If you are sending functions from a user-defined queue on the SAP server, the functions are delivered in the order in which they exist on the queue. You can see the contents of the queue by entering transaction code SMQ1 on the SAP server.
7. If the delivery is successful, and if you selected **Ensure once-only event delivery**, the adapter removes the transaction ID from the event persistent table.

   If a failure occurs when the adapter attempts to deliver the business object, the transaction ID remains in the event table. When another event is received from the SAP server, the following processing occurs:
   a. The adapter checks the transaction ID.
   b. If the event matches an ID in the table, the adapter processes the failed event once; it does not send the event with the duplicate ID, thereby ensuring that the event is processed only once.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

"Passing parameters and reporting errors" on page 1942
The BAPI interface is defined by its input parameters (IMPORT), output parameters (EXPORT), and tables.

"BAPI inbound scenarios" on page 1943
In SAP, you can call functions in other applications or SAP systems that are registered with SAP as remote function call (RFC) servers. In WebSphere Message Broker, you can register the SAP adapter with SAP as an RFC server so that it accepts synchronous and asynchronous calls from SAP.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

"SAPReply node" on page 4682
Use the SAPReply node to send a reply to an SAP synchronous callout. Use this node with an SAPInput node to implement a message flow that acts as a remote

function call (RFC) destination.

Event recovery for the BAPI interface:

You can configure the adapter for BAPI inbound processing so that it supports event recovery in case a failure occurs while the event is being delivered from the adapter to the endpoint. When event recovery is specified, the adapter persists the event state in an event recovery table that resides on a data source.

Event recovery is not the default; you must specify it by enabling once-only delivery of events during adapter configuration.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

"Passing parameters and reporting errors" on page 1942
The BAPI interface is defined by its input parameters (IMPORT), output parameters (EXPORT), and tables.

"BAPI inbound scenarios" on page 1943
In SAP, you can call functions in other applications or SAP systems that are registered with SAP as remote function call (RFC) servers. In WebSphere Message Broker, you can register the SAP adapter with SAP as an RFC server so that it accepts synchronous and asynchronous calls from SAP.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection

wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

"SAPReply node" on page 4682
Use the SAPReply node to send a reply to an SAP synchronous callout. Use this node with an SAPInput node to implement a message flow that acts as a remote function call (RFC) destination.

Passing parameters and reporting errors:

The BAPI interface is defined by its input parameters (IMPORT), output parameters (EXPORT), and tables.

After the Adapter Connection wizard has discovered the BAPI interface, the wizard creates a message set that contains an element and a type for the definition of that interface. Each of the import or export parameters has a corresponding field, which has an associated type that can be simple or complex. Tables are represented in the message type definition as repeating complex structures (maxOccurs = -1).

BAPIs are also defined by the messages (error, warning, or success) that they can return. These messages are typically returned in an export parameter (for example, BAPIRETURN). Most BAPIs have this parameter in common, although the type of the parameter can vary. For example, its type can be:

- BAPIRETURN
- BAPIRET1
- BAPIRET2

These structures are similar, except for a few fields that have been added in later versions.

To send an error back to the calling SAP program, use the BAPIReturn structure that was specified as one of the export parameters when the BAPI was defined. Messages are returned in the Return export parameter. You can use the transaction code SE91 for message maintenance.

For more information, see Return Parameters (Error Handling) in the BAPI Programming Guide Reference on the SAP Help Portal.

As well as the application level errors, which can be reported by the Return export parameter, system or communication failures also exist, which indicate to SAP that the function could not be called or did not complete.

Related concepts:

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

"BAPI inbound scenarios"
In SAP, you can call functions in other applications or SAP systems that are registered with SAP as remote function call (RFC) servers. In WebSphere Message Broker, you can register the SAP adapter with SAP as an RFC server so that it accepts synchronous and asynchronous calls from SAP.

"SAP adapter scalability and performance" on page 1949
You can improve performance by configuring the number of listeners on the adapter and the number of additional instances on the message flow to prevent delays when processing synchronous calls from SAP.

"DataObject parser and domain" on page 1114
Use the DataObject domain to parse and write messages for WebSphere Adapters and CORBA applications.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

"Tuning the SAP adapter for scalability and performance" on page 3278
You can configure the number of listeners on the adapter and the number of additional instances on the message flow to prevent delays when processing synchronous calls from SAP.

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

Related reference:

"SAPReply node" on page 4682
Use the SAPReply node to send a reply to an SAP synchronous callout. Use this node with an SAPInput node to implement a message flow that acts as a remote function call (RFC) destination.

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

BAPI inbound scenarios:

In SAP, you can call functions in other applications or SAP systems that are registered with SAP as remote function call (RFC) servers. In WebSphere Message

Broker, you can register the SAP adapter with SAP as an RFC server so that it accepts synchronous and asynchronous calls from SAP.

Integrating SAP with a system with a synchronous interface:
In SAP, a BAPI interface is a function call. If a system is connected to SAP as an RFC server, you can use the BAPI interface to define the interface where a program running in SAP can call an external system. The external system is identified in SAP by its RFCDestination value, which is bound to a program ID in SAP administration. The program ID is specified by the external system when it first connects to SAP.

By using the WebSphere Adapter for SAP, a message flow can connect to SAP as an RFC Server by configuring the adapter with the relevant program ID and deploying the message flow to the broker. After deployment, the message flow can receive synchronous function calls by using the BAPI interface.

The reply identifier

The BAPI import parameters are received by the adapter and propagated from an SAPInput node as a message tree structure. The BAPI export parameters are propagated to an SAPReply node as a message tree structure. The adapter then sends back the export parameters to the calling SAP program. In this case, the SAPReply node is typically in the same flow as the SAPInput node. The SAPInput node provides a unique ID (the reply identifier) for each BAPI call. The reply identifier is propagated to the SAPReply node (in the local environment) to indicate to which BAPI call it is replying.

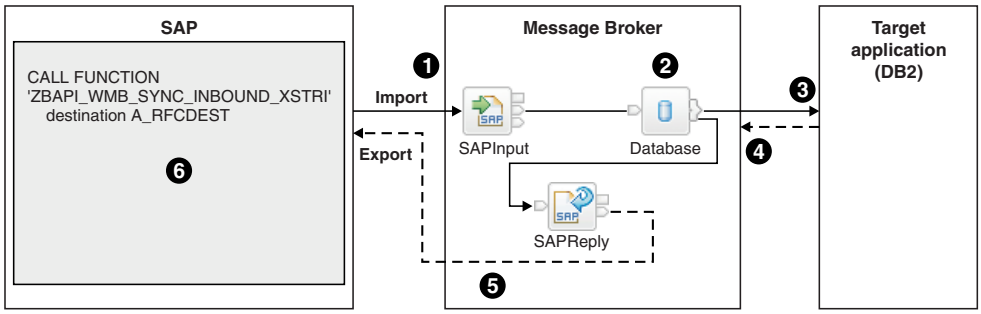
If no reply identifier exists in the local environment, the SAPReply node automatically uses the reply identifier from the SAPInput node that triggered the current execution of the message flow. If the flow is triggered by anything but an SAPInput node, or if a break occurs in the flow, an error is issued. An unhandled exception in the message flow causes a system failure in SAP.

Even if the BAPI does not expect any output parameters, the (empty) message must be propagated to the SAPReply node.

You can process two concurrent callouts from SAP by configuring the message flow with an additional instance.

Scenario 1

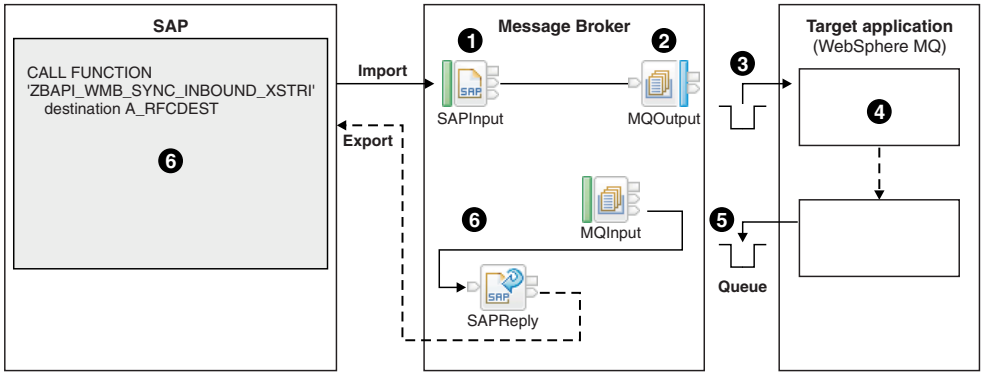
The following diagram represents a message flow where WebSphere Message Broker provides a link between SAP and a target application (in this case, DB2) The SAP program requires reply data, therefore it blocks further processing until the call completes.



1. The SAP program makes a BAPI call to WebSphere Message Broker.
2. WebSphere Message Broker converts the call to an SQL call.
3. WebSphere Message Broker passes the call on to DB2.
4. DB2 processes the SQL and returns the result to WebSphere Message Broker.
5. WebSphere Message Broker converts the SQL result to a BAPI reply and sends the reply to SAP.
6. The SAP program processes the next line of code.

Scenario 2

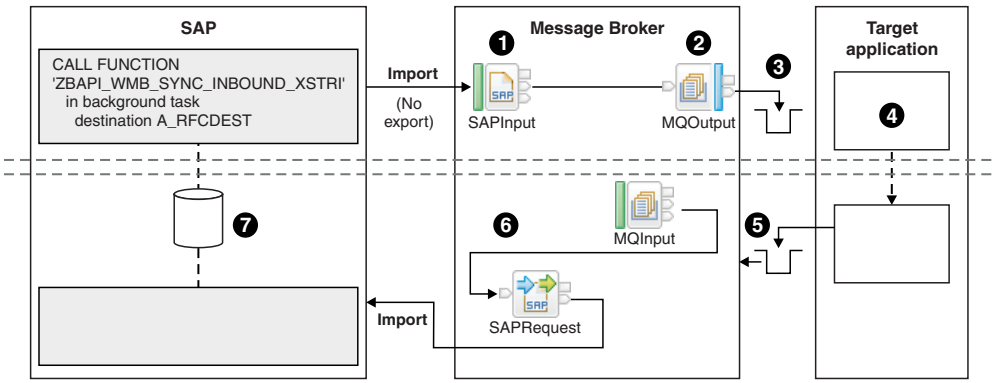
The following diagram also represents a message flow where the SAP program requires reply data, but in this scenario, calls between WebSphere Message Broker and the target application (in this case, WebSphere MQ) are asynchronous. The SAP system blocks further processing until the call completes. When you use the SAPReply node in a different flow from the SAPInput node, deploy the SAPReply node in the same execution group as the SAPInput node.



1. The SAP program makes a BAPI call to WebSphere Message Broker.
2. WebSphere Message Broker converts the import parameters into a message format that is understood by the target application.
3. WebSphere Message Broker puts that message on a request queue.
4. The target application gets the request message from the queue, processes it, and puts a reply message on the reply-to queue.
5. WebSphere Message Broker gets the reply message from the queue.
6. WebSphere Message Broker converts the reply message to BAPI export parameters and sends the reply to SAP.

Scenario 3

The following diagram represents an asynchronous call from SAP to WebSphere Message Broker, and an asynchronous call from WebSphere Message Broker to a target application. This scenario shows how you can combine the inbound processing that is described in this topic with outbound processing to achieve the same result as in scenarios 1 and 2. For more information about outbound processing, see “Outbound processing for the BAPI interface” on page 1926.



1. The SAP program makes a BAPI call to WebSphere Message Broker, stores relevant information in a database table, and continues to process the next line of code.
2. WebSphere Message Broker converts the import parameters into a message format that is understood by the target application.
3. WebSphere Message Broker puts that message on a request queue.
4. The target application gets the request message from the queue and processes it.
5. WebSphere Message Broker gets the reply message from the queue.
6. The SAPRequest node sends the message to the SAP program, requesting an update in SAP.
7. The SAP program refers to the information that is stored in the database table and makes the requested update.

Errors and warnings

- If an SAPReply node is deployed but no SAPInput node is deployed to that execution group, a warning is written to syslog or the Windows Event Viewer.
- If the SAPReply node is provided with a reply identifier that does not correspond to any BAPI call in this execution group, an error is issued.
- If the same reply identifier is sent to two SAPReply nodes, the second node receives an error message.
- If an SAP program tries to call a BAPI that is using the RFCDestination value of the broker, but that BAPI was not discovered for that adapter, an error is written to syslog or the Windows Event Viewer, and a failure message is sent back to the calling SAP program.

Related concepts:

“Overview of WebSphere Adapter for SAP Software” on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

“Passing parameters and reporting errors” on page 1942
The BAPI interface is defined by its input parameters (IMPORT), output

parameters (EXPORT), and tables.

"SAP adapter scalability and performance" on page 1949
You can improve performance by configuring the number of listeners on the adapter and the number of additional instances on the message flow to prevent delays when processing synchronous calls from SAP.

"DataObject parser and domain" on page 1114
Use the DataObject domain to parse and write messages for WebSphere Adapters and CORBA applications.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

"Tuning the SAP adapter for scalability and performance" on page 3278
You can configure the number of listeners on the adapter and the number of additional instances on the message flow to prevent delays when processing synchronous calls from SAP.

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

Related reference:

"SAPReply node" on page 4682
Use the SAPReply node to send a reply to an SAP synchronous callout. Use this node with an SAPInput node to implement a message flow that acts as a remote function call (RFC) destination.

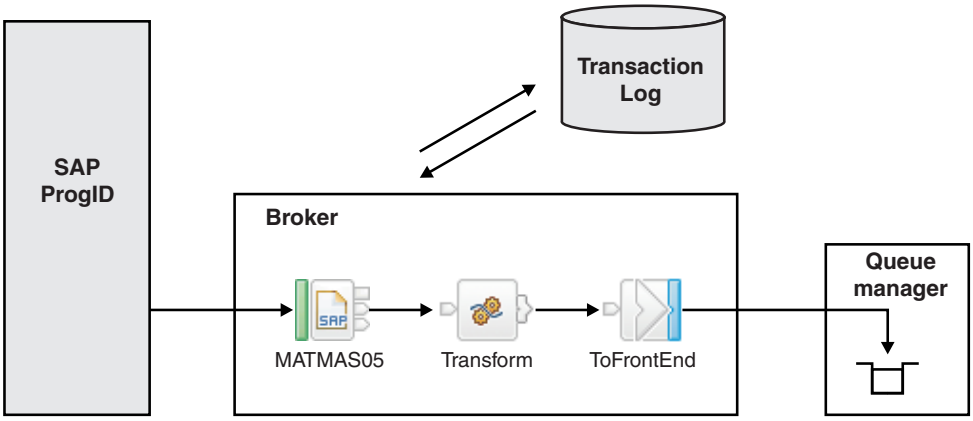
"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

SAP high availability:

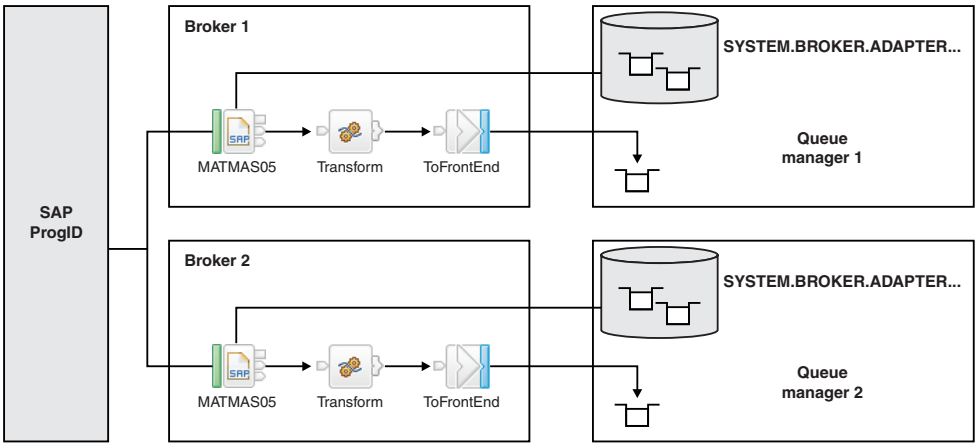
You can configure WebSphere Message Broker to withstand software or hardware failures when working with SAP, so that WebSphere Message Broker is available for as much of the time as possible.

In WebSphere Message Broker , the tRFC protocol between SAP and WebSphere Message Broker (acting as the RFC server) ensures that IDocs and tRFC BAPI calls are delivered exactly once. This behavior is possible because each delivery has an

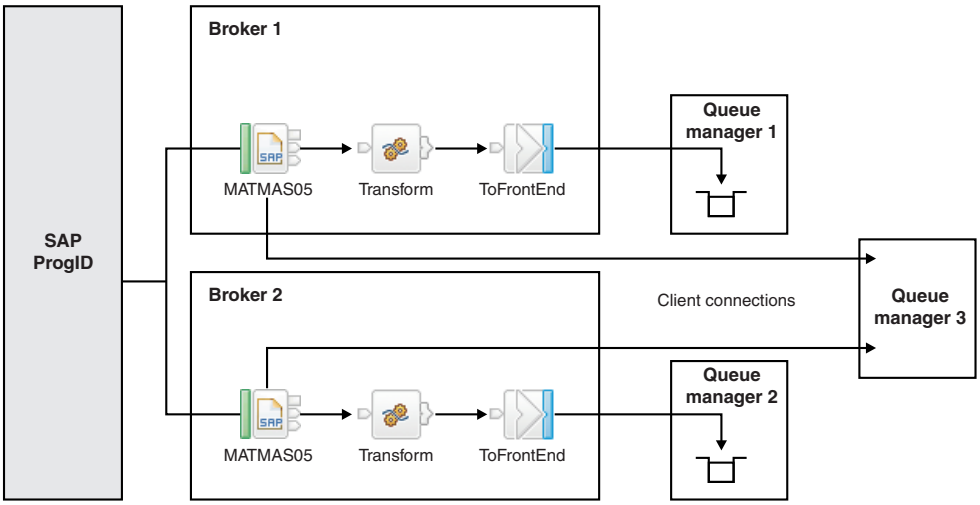
associated transaction ID (TID). WebSphere Message Broker monitors the progress of a delivery until SAP confirms a successful delivery. If the connection is lost or WebSphere Message Broker fails before that confirmation is issued, SAP attempts to redeliver the message. By keeping a persistent record (in the TID store or transaction log), the broker can ensure integrity and avoid a duplicate delivery.



When two `.inadapter` components, with the same RFC program ID, are deployed to two brokers, two connections to the same RFC server are visible to SAP. If the connection is lost to one of the brokers, SAP might attempt to redeliver to the other broker. The brokers have separate TID stores, therefore, the second broker accepts the redelivery, even though the first broker might have processed some (or all) of the IDocs in the packet.



In WebSphere Message Broker Version 7.0, you can move the TID store to a remote queue manager that can be shared between two brokers. To avoid a single point of failure, make this third queue manager a WebSphere MQ high availability, multi-instance queue manager. For instructions, see “Setting up SAP for high availability” on page 2057.



Related concepts:

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Setting up SAP for high availability" on page 2057
You can configure WebSphere Message Broker to withstand software or hardware failures when working with SAP by moving the transaction ID (TID) store to a remote queue manager that can be shared between two brokers. To avoid a single point of failure, make this queue manager a WebSphere MQ high availability, multi-instance queue manager.

"Defining WebSphere MQ resources" on page 1558
An application client can run on a computer anywhere in the WebSphere MQ network. If your applications use WebSphere MQ facilities to connect to the broker, and to interact with it (by using the MQI and AMI), you must set up the WebSphere MQ resources that they require.

"Using WebSphere MQ cluster queues for input and output" on page 1544
Design your broker network to use WebSphere MQ queues, if appropriate for your business needs.

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

Related reference:

"WebSphere MQ resources for the broker" on page 585
Each broker depends on a number of WebSphere MQ resources: some are required, others are optional, and depend on your environment and requirements. Some of these resources are created for you, but others you must define for yourself.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

SAP adapter scalability and performance:

You can improve performance by configuring the number of listeners on the adapter and the number of additional instances on the message flow to prevent delays when processing synchronous calls from SAP.

The SAP inbound adapter receives synchronous calls from SAP. The adapter has a property called `Number of listeners`, which controls the maximum number of concurrent calls by configuring the adapter to have a particular number of threads listening to the SAP program ID. The listeners send the input parameters of these calls to the SAPInput node for processing, and the output parameters are sent to the SAPReply node.

When the listener receives a call from SAP, it blocks processing until a message flow instance that contains the SAPInput node is available. When a message flow instance has become available, and has started to process the import parameters, the listener again blocks processing until a message that contains the export parameters is propagated to an SAPReply node.

The amount of time that elapses between the SAPInput node sending the message and the SAPReply node receiving the reply message can be affected by the additional instances property on the message flow. To avoid delays in processing, tune the maximum number of listeners and the number of additional instances for the message flow that contains the SAPInput and SAPReply nodes. You can configure the number of additional instances at message flow level or on the SAPInput node itself.

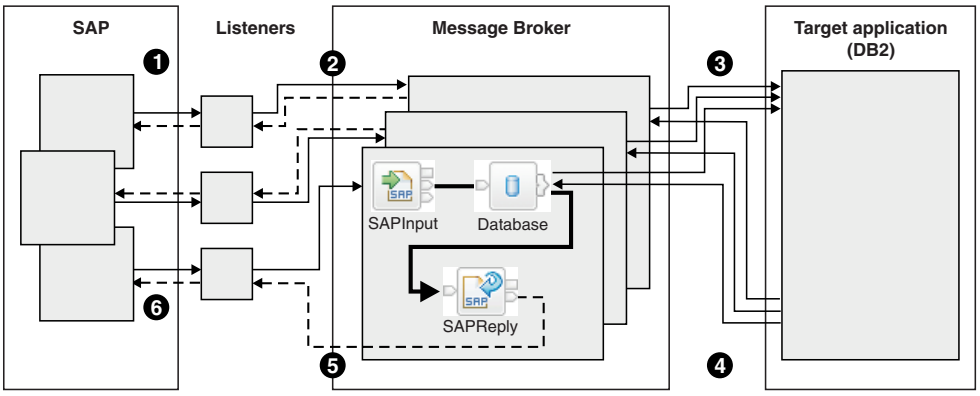
Restrictions

The number of listeners is limited by the SAP configuration. In SAP transaction SMQS, you can view and change the maximum connections property for each RFC destination. Configuring a number of listeners greater than the maximum number of connections has no effect.

For each additional instance of the message flow, extra resources are used by each node in the flow, depending on the types of node in the flow.

Scenarios

The following diagram illustrates a basic scenario, where the number of listeners is equal to the number of additional instances of the message flow; in this case, the scenario has been configured with three listeners and three message flow instances.



1. SAP makes three calls, each of which is received by a listener.
2. Each of the listeners sends the input parameters of each call to the SAPInput node in one of three instances of the message flow.
3. Each message flow instance sends its message to the target application.

4. The target application processes the messages and returns replies to the message flow instances.
5. The SAPReply node in each message flow instance sends the reply message to the listener that received the original call.
6. Each listener returns the reply message to the appropriate SAP program.

The SAPReply node can be in the same message flow as the SAPInput node, as illustrated in the previous example. However, in the following scenario, the SAPReply node is in a different flow from the SAPInput node. The SAPReply node must be deployed in the same execution group as the SAPInput node.

1. SAP makes three calls, each of which is received by a different listener.
2. Each of the listeners sends the input parameters of each call to a message flow that contains an SAPInput node.
3. The message flow puts the message onto a queue for the target application.
4. The target application gets the messages from the queue and processes them.
5. The target application puts the messages onto a queue.
6. A second message flow that contains an SAPReply node gets the messages from the queue and processes them.
7. The SAPReply node sends the reply messages to the appropriate listeners.
8. Each listener returns a reply message to the appropriate SAP program.

In this scenario, the message flow has low latency compared to the time taken by the entire scenario. Therefore, you can limit the resources that are used by the message flow containing the SAPInput node by configuring fewer additional instances for this message flow. One instance of the message flow, as in the example, can service many listeners because the message flow propagates the import parameters quickly for processing.

The following scenarios are also possible.

- A single message flow contains an SAPReply node but it is used to reply to several SAPInput nodes. After the message has been propagated to the SAPReply node, the listener sends the reply back to SAP and is therefore ready to receive another call from SAP. However, the current instance of the message flow is still busy processing the nodes downstream from the SAPReply node. In this case, increase the number of instances on the message flow that contains the SAPReply node.
- After propagating to the SAPReply node, other nodes in the message flow perform extra processing. In this case, increase the number of instances on the message flow that contains the SAPReply node, even when it is the same flow that contains the SAPInput node.

  This scenario might have an undesirable effect on the integrity of your data. If an exception occurs after the SAPReply node, resources that are updated by the message flow (for example, database tables or WebSphere MQ queues) might be rolled back. However, the reply has been sent back to SAP already and cannot be rolled back. If this behavior is not appropriate, you can improve data integrity by ensuring that the SAPReply node is the final node in the message flow.

For information about how to tune the SAP adapter, see “Tuning the SAP adapter for scalability and performance” on page 3278.

Related concepts:

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Tuning the SAP adapter for scalability and performance" on page 3278
You can configure the number of listeners on the adapter and the number of additional instances on the message flow to prevent delays when processing synchronous calls from SAP.

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

"Starting user trace" on page 3197
Use user trace for debugging your applications; you can trace brokers, execution groups, and deployed message flows. Start user trace facilities using the **mqsichangetrace** command or the WebSphere Message Broker Explorer.

"Starting to collect message flow accounting and statistics data" on page 3288
You can start collecting message flow accounting and statistics data for an active broker at any time. You can specify what type of statistics you want to collect, and where to send the data.

Related reference:

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"Configurable message flow properties" on page 4020
When you add a message flow to a broker archive (BAR) file in preparation for deploying it to a broker, you can set additional properties that influence its run time operation. These properties are available for review and update when you select the **Manage and Configure** tab for the broker archive file.

Business objects for the BAPI interface:

A business object is a structure that consists of data, the action to be performed on the data, and additional instructions for processing the data.

For outbound processing, the broker uses business objects to send data to SAP or obtain data (through the adapter) from SAP. The broker sends a business object to the adapter, and the adapter converts the data in the business object to a format that is compatible with an SAP API call. The adapter then runs the SAP API with this data.

For inbound processing, the SAP server sends a BAPI function call through the adapter to an endpoint. The adapter converts the BAPI function call into a business object.

The adapter uses the BAPI metadata that is generated by the Adapter Connection wizard to construct a business-object definition. This metadata contains BAPI-related information such as the operation of the business object, import parameters, export parameters, table parameters, transaction information, and dependent or grouped BAPIs.

The BAPI business-object definition that is generated by the Adapter Connection wizard is modeled on the BAPI function interface in SAP. The business-object definition represents a BAPI function, such as a BAPI_CUSTOMER_GETLIST function call.

If you change the BAPI interface, you must run the Adapter Connection wizard again to rebuild the business object definition.

How business-object definitions are created

You create business-object definitions by using the Adapter Connection wizard. The wizard connects to the application, discovers data structures in the application, and generates business-object definitions to represent them. It also generates other resources that are needed by the adapter, such as the interface information that indicates the input and output parameters.

Business object structure

The structure of a BAPI business object depends on the interface type (simple BAPI, BAPI work unit, or BAPI result set).

For more information, see the following topics.

- "Business object structure for a simple BAPI" on page 1954
- "Business object structure for a nested BAPI" on page 1955
- "Business object structure for a BAPI work unit" on page 1956
- "Business object structure for a BAPI result set" on page 1958

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

Business object structure for a simple BAPI:

A business object for a simple BAPI call reflects a BAPI method or function call in SAP. Each business object property maps to a BAPI parameter. The metadata of each business-object property indicates the corresponding BAPI parameter. The operation metadata determines the correct BAPI to call.

For a simple BAPI that performs Create, Update, Retrieve, and Delete operations, each operation is represented by a business object, with the business objects being grouped together in a wrapper.

The business object wrapper can be associated with multiple operations, but for a simple BAPI, each business object is associated with only one operation. For example, while a wrapper business object can contain BAPIs for Create and Delete operations, BAPI_CUSTOMER_CREATE is associated with the Create operation, not the Delete operation.

The BAPI business objects are children of the business object wrapper, and, depending on the operation to be performed, only one child object in this wrapper needs to be populated at run time in order to process the simple BAPI call. Only one BAPI, the one that is associated with the operation to be performed, is called at a time.

If you select **Asynchronous Transactional RFC** (for outbound or inbound processing) or **Asynchronous Queued RFC** (for outbound processing) , the BAPI wrapper business object also contains a transaction ID. The transaction ID is used to resend the BAPI call if the receiving system is not available at the time of the initial call.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

Business object structure for a nested BAPI:

A nested BAPI business object contains structure parameters that can contain one or more other structures as components.

A BAPI business object can contain both simple parameters and structure parameters. A business object that contains structure parameters can in turn contain other structures, such as simple parameters and a business object.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

Business object structure for a BAPI work unit:

A business object that represents a BAPI work unit (also known as a BAPI transaction) is a wrapper object that contains multiple child BAPI objects. Each child BAPI object in the wrapper object represents a simple BAPI.

The adapter supports a BAPI work unit by using a top-level wrapper business object that consists of multiple child BAPIs, each one representing a simple BAPI

in the sequence. The BAPI wrapper object represents the complete work unit, while the child BAPI objects contained in the BAPI wrapper object represent the individual operations that make up the work unit.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

Business object structure for a BAPI result set:

The top-level business object for a result set is a wrapper that contains a GetDetail business object. The GetDetail business object contains the results of a query for SAP data. The GetDetail business object also contains, as a child object, the query business object. The query business object represents a GetList BAPI. These two BAPIs work together to retrieve information from the SAP server.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

The ALE interfaces:

The SAP Application Link Enabling (ALE) interface and ALE pass-through IDoc interface enable business process integration and asynchronous data communication between two or more SAP systems or between SAP and external systems. The data is exchanged in the form of Intermediate Documents (IDocs).

The adapter supports outbound and inbound processing by enabling the exchange of data in the form of business objects.

- For inbound processing, SAP pushes the data in IDocs to the SAP adapter. The adapter converts the IDocs to business objects and delivers them to the endpoint.
- For outbound processing, the SAP adapter converts the business object to an IDoc and delivers it to SAP.

To use the ALE interface or ALE pass-through IDoc interface for inbound processing, make sure that your SAP server is properly configured (for example, you must set up a partner profile and register an SAP RCF program ID to listen for events).

Application systems are loosely coupled in an ALE integrated system, and the data is exchanged asynchronously.

IDocs

Intermediate Documents (IDocs) are containers for exchanging data in a predefined (structured ASCII) format across system boundaries. The IDoc type indicates the SAP format that is to be used to transfer the data. An IDoc type can transfer several message types (the logical messages that correspond to different business processes). IDocs can be used for outbound and inbound processing. The SAP adapter supports the basic and extension type of IDocs.

For example, if an application developer wants to be notified when a sales order is created on the SAP server, the developer runs the Adapter Connection wizard to discover the ORDERS05 IDoc on the SAP server. The wizard then generates the business object definition for ORDERS05. The wizard also generates other resources, such as an EIS export component and Service Component Architecture (SCA) interfaces.

IDocs are exchanged for inbound and outbound events, and IDocs can be exchanged either as individual documents or in packets.

The processing of IDoc data depends on whether you are using the ALE interface or the ALE pass-through IDoc interface.

- **ALE interface**

  For outbound processing, the adapter uses the IDoc business object to populate the appropriate RFC-enabled function call made to the SAP server.

  For inbound processing, IDocs can be sent from the SAP server as parsed or unparsed documents

  – For parsed documents, the adapter parses the IDoc and creates a business object that reflects the internal structure of the IDoc.
  – For unparsed IDocs, the adapter processes the IDoc but does not convert the data portion of the IDoc.
- **ALE pass-through IDoc interface**

For both outbound and inbound processing, the adapter does no conversion of the IDoc, which is useful when the client wants to perform the IDoc parsing.

Transactional RFC processing

The adapter uses transactional RFC (tRFC) to assure delivery and to ensure that each IDoc is exchanged only once with SAP. The tRFC component stores the called RFC function in the database of the SAP system with a unique transaction identifier (TID). The TID is a field in your message.

The message flow must determine how to store the SAP transaction ID and how to relate the SAP transaction ID to the data being sent to the adapter. When the events are successful, the message flow should not resubmit the event associated with this TID again to prevent the processing of duplicate events.

- If the message flow does not send an SAP transaction ID with the business object, the adapter returns one after running the transaction.
- If the message flow has an SAP transaction ID, it must populate the SAP transaction ID property in the business object with that value before running the transaction.

The SAP transaction ID can be used for cross-referencing with a global unique ID that is created for an outbound event. You can create the global unique ID for managing integration scenarios.

Queued RFC processing

The adapter uses qRFC (queued transactional RFC) to ensure that IDocs are delivered in sequence to a queue on the SAP server or are received in sequence from the SAP server. Additional threads can increase the throughput of a message flow but you should consider the potential effect on message order. To maintain message order, ensure that your message flow is single threaded.

For more information about ALE interfaces, see the following topics:

- "Outbound processing for the ALE interface" on page 1961
- "Inbound processing for the ALE interface" on page 1963
- "Pass-through support for IDocs, and MQSeries link for R/3 link migration" on page 1973
- "ALE business objects" on page 1978

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere

Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

Outbound processing for the ALE interface:

The adapter supports outbound processing (from the adapter to the SAP server) for the ALE interface and the ALE pass-through IDoc interface. ALE uses IDocs for data exchange, and the adapter uses business objects to represent the IDocs.

The following list describes the sequence of processing actions that result from an outbound request that uses the ALE interface and ALE pass-through IDoc interface.

The message flow that makes the request uses the interface information that was generated by the Adapter Connection wizard.

1. The adapter receives a request, which includes an IDoc business object, from a message flow.

   For pass-through IDocs, the Message tree contains a BLOB field that represents the IDoc. No separate IDoc business object exists for pass-through IDocs.
2. The adapter uses the IDoc business object to populate the appropriate RFC-enabled function call used by the ALE interface.
3. The adapter establishes an RFC connection to the ALE interface and passes the IDoc data to the SAP system. If you are using the qRFC protocol, the adapter passes the IDoc data in the order specified in the wrapper business object to the specified queue on the SAP server.

4. After passing the data to SAP, the adapter performs one of the following steps:
   - If the call is not managed by a local transaction that uses the broker's Local Transaction Manager, the adapter releases the connection to SAP and does not return any data to the caller. When no exceptions are raised, the outbound transaction is considered successful. You can verify whether the data is incorporated into the SAP application by inspecting the IDocs that have been generated in SAP.
   - If the call is managed by a local transaction that uses the broker's Local Transaction Manager, the adapter returns the transaction ID.

     The adapter uses the tRFC protocol to support J2C local transactions.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.
"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

Inbound processing for the ALE interface:

The adapter supports inbound processing (from the SAP server to the adapter) for the ALE interface and the ALE pass-through IDoc interface.

When you are configuring a module for the ALE interface or the ALE pass-through interface, you indicate whether the IDocs are sent as a packet and, for the ALE interface, you can specify whether they are sent parsed or unparsed. You make these selections in the Adapter Connection wizard. When you use the ALE pass-through IDoc interface, the Message tree contains a BLOB field that represents the IDoc. No separate IDoc business object exists for pass-through IDocs.

The following list describes the sequence of processing actions that result from an inbound request using the ALE interface.
1. The adapter starts event listeners to the SAP server.
2. Whenever an event occurs in SAP, the event is sent to the adapter through the event listeners.
3. The adapter converts the event into a business object before sending it to the endpoint.

The adapter uses the event recovery mechanism to track and recover events in case of abrupt termination. The event recovery mechanism uses a data source for persisting the event state.

The following table provides an overview of the differences between the ALE interface and the ALE pass-through IDoc interface for inbound processing.

Interface	When to use	SplitIDoc = true	SplitIDoc = false	Parsed IDoc = true
ALE inbound	This interface converts the raw incoming IDocs to business objects, which are readily consumable by the client at the endpoint.	On receiving the IDoc packet from SAP, the adapter converts the IDocs to business objects, one by one, before sending each one to the endpoint.	On receiving the IDoc packet from SAP, the adapter converts the IDocs in the packet as one business object before sending it to the endpoint.	The incoming IDoc is only partially parsed (the control record of the IDoc is parsed but the data record is not). The client at the endpoint is responsible for parsing the data record.
ALE pass-through IDoc	This interface wraps the raw incoming IDoc in a business object before delivering it to the client at the endpoint. The client is responsible for parsing the raw IDoc.	On receiving the IDoc packet from SAP, the adapter wraps each raw IDoc in a business object before sending the objects, one by one, to the endpoint.	On receiving the IDoc packet from SAP, the adapter wraps the raw IDoc packet in a business object before sending it to the endpoint.	This attribute is not applicable to the ALE pass- through IDoc interface. (Neither the control record nor the data record of the IDoc is parsed.)

For more information, see the following topics.
- "Event error handling" on page 1965

- "Event recovery for the ALE interface" on page 1966
- "Event processing for parsed IDoc packets" on page 1967
- "Event processing for unparsed IDocs" on page 1969
- "IDoc status updates" on page 1971

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

Event error handling:

WebSphere Adapter for SAP Software provides error handling for inbound ALE events by logging the errors and attempting to restart the event listener.

When the adapter detects an error condition, it performs the following actions:

1. The adapter logs the error information in the syslog (on Linux and UNIX systems), Windows Event Viewer, or user trace log.
2. The adapter attempts to restart the existing event listeners.

   The adapter uses the activation specification values for RetryLimit and RetryInterval.

   - If the SAP application is not active, the adapter attempts to restart the listeners for the number of times configured in the RetryLimit property.
   - The adapter waits for the time specified in the RetryInterval parameter before attempting to restart the event listeners.
3. If the attempt to restart the event listeners fails, the adapter performs the following actions:
   - The adapter logs the error condition in the syslog (on Linux and UNIX systems), Windows Event Viewer, or user trace log.
   - The adapter cleans up the existing ALE event listeners.
   - The adapter starts new event listeners.

   The adapter uses the values of the RetryLimit and RetryInterval properties when starting the new event listeners.
4. If all the retry attempts fail, the adapter logs the relevant message and CEI events and stops trying to recover the ALE event listener.

   You must restart the adapter or Service Component Architecture (SCA) application in this case.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system

administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

Event recovery for the ALE interface:

You can configure the adapter for ALE inbound processing so that it supports event recovery in case of abrupt termination.

When event recovery is specified, the adapter persists the event state in an event recovery table that resides on a data source. Event recovery is not the default behavior; you must specify it by enabling once-only delivery of events during adapter configuration.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

Event processing for parsed IDoc packets:

An inbound event can contain a single IDoc or multiple IDocs, with each IDoc corresponding to a single business object. The multiple IDocs are sent by the SAP server to the adapter in the form of an IDoc packet. You can specify, during adapter configuration, whether the packet can be split into individual IDocs or whether it must be sent as one object (non-split).

Event processing begins when the SAP server sends a transaction ID to the adapter. The following sequence occurs.

1. The adapter checks the status of the event and takes one of the following actions:
   - If this is a new event, the adapter stores an EVNTID (which corresponds to the transaction ID) along with a status of `0` (Created) in the event recovery table.
   - If the event status is `-1` (Rollback), the adapter updates the status to `0` (Created).
   - If the event status is `1` (Executed), the adapter returns an indication of success to the SAP system.
2. The SAP system sends the IDoc to the adapter.
3. The adapter converts the IDoc to a business object and sends it to the endpoint.

   For single IDocs and non-split IDoc packets, the adapter can deliver objects to endpoints that support transactions as well as to endpoints that do not support transactions.

- For endpoints that support transactions, the adapter delivers the object as part of a unique XA transaction. When the endpoint processes the event and the transaction is committed, the status of the event is updated to 1 (Executed).

  The endpoint must be configured to support XA transactions.
- For endpoints that do not support transactions, the adapter delivers the object to the endpoint and updates the status of the event to 1 (Executed). The adapter delivers the business object without the quality of service (QOS) that guarantees once-only delivery.

4. For split packets only, the adapter performs the following tasks:
   a. The adapter updates the BQTOTAL column (or table field) in the event recovery table to the number of IDocs in the packet. This number is used for audit and recovery purposes.
   b. The adapter sends the business objects to the message endpoint, one after the other, and updates the BQPROC property to the sequence number of the IDoc it is working on. The adapter delivers the objects to the appropriate endpoint as part of a unique XA transaction (a two-phase commit transaction) controlled by the application server.
   c. When the endpoint receives the event and the transaction is committed, the adapter increments the number in the BQPROC property.

      The message endpoint must be configured to support XA transactions.If the adapter encounters an error while processing a split IDoc packet, it can behave in one of two ways, depending on the IgnoreIDocPacketErrors configuration property:
      - If the IgnoreIDocPacketErrors property is set to `false`, the adapter stops processing any additional IDocs in the packet and reports errors to the SAP system.
      - If the IgnoreIDocPacketErrors property is set to `true`, the adapter logs an error and continues processing the rest of the IDocs in the packet. The status of the transaction is marked 3 (InProgress). In this case, the adapter log shows the IDoc numbers that failed, and you must resubmit those individual IDocs separately. You must also manually maintain these records in the event recovery table.

      This property is not used for single IDocs and for non-split IDoc packets.
   d. The SAP system sends a COMMIT call to the adapter.
   e. After the adapter delivers all the business objects in the IDoc packet to the message endpoint, it updates the event status to 1 (Executed).
   f. In case of abrupt interruptions during IDoc packet processing, the adapter resumes processing the IDocs from the current sequence number. The adapter continues updating the BQPROC property, even if IgnoreIDocPacketErrors is set to `true`. The adapter continues the processing in case you terminate the adapter manually while the adapter is processing an IDoc packet.
5. If an exception occurs either while the adapter is processing the event or if the endpoint generates an exception, the event status is updated to `-1` (Rollback).
6. If no exception occurs, the SAP server sends a CONFIRM call to the adapter.
7. The adapter deletes the records with a 1 (Executed) status and logs a common event infrastructure (CEI) event that can be used for tracking and auditing purposes.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

Event processing for unparsed IDocs:

Unparsed IDocs are passed through, with no conversion of the data (that is, the adapter does not parse the data part of the IDoc). The direct exchange of IDocs in the adapter enables high-performance, asynchronous interaction with SAP, because the parsing and serializing of the IDoc occurs outside the adapter. The consumer of the IDoc parses the IDoc.

The adapter processes the data based on whether the packet IDoc is split or non-split and whether the data needs to be parsed.

- The adapter can process packet IDocs as a packet or as individual IDocs. When an IDoc is received by the adapter from SAP as a packet IDoc, it is either split and processed as individual IDocs, or it is processed as a packet. The value of the SplitIDocPacket metadata at the business-object level determines how the IDoc is processed.

  For split IDocs, the wrapper contains only a single, unparsed IDoc object.
- The Type metadata specifies whether the data should be parsed. For unparsed IDocs, this value is `UNPARSEDIDOC`; for parsed IDocs, the value is `IDOC`. This value is set by the Adapter Connection wizard.

Unparsed data format

In the fixed-width format of an unparsed IDoc, the segment data of the IDoc is set in the IDocData field of the business object. It is a byte array of fixed-length data.

The entire segment length might not be used. The adapter pads spaces to the fields that have data; the rest of the fields are ignored, and an end of segment is set. The end of segment is denoted by a null value.

The following figure shows a segment with fields demarcated by the '|' symbol for reference.

Figure 2. Example of a segment before processing

When the adapter processes this segment into unparsed data, it takes into account only those fields that have data in them. It maintains the field width for each segment field. When it finds the last field with data, it appends a null value to mark the end of segment.

Figure 3. Example of a segment after processing

The next segment data processed as unparsed data would be appended after the null value.

Limitations

The unparsed event feature introduces certain limitations on the enterprise application for a particular IDoc type.

- The enterprise application supports either parsed or unparsed business-object format for an IDoc type or message type.
- For an IDoc type, if you select unparsed business-object format for inbound, you cannot have inbound and outbound interfaces in the same EAR file, because outbound is based on parsed business objects.
- The DummyKey feature is not supported for unparsed IDocs.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

IDoc status updates:

To monitor IDoc processing, you can configure the adapter to update the IDoc status.

When the adapter configuration property UpdateStatus is set to `true` (indicating that an audit trail is required for all message types), the adapter updates the IDoc status of ALE business objects that are retrieved from the SAP server. After the event is sent to the message endpoint, the adapter updates the status of the IDoc in SAP to indicate whether the processing succeeded or failed. Monitoring of IDocs applies only to inbound processing (when the IDoc is sent from the SAP server to the adapter).

The adapter updates a status IDoc (ALEAUD) and sends it to the SAP server.

An IDoc that is not successfully sent to the endpoint is considered a failure, and the IDoc status is updated by the adapter. Similarly, an IDoc that reaches the endpoint is considered successfully processed, and the status of the IDoc is updated.

The status codes and their associated text are configurable properties of the adapter, as specified in the activation specification properties and shown in the following list:

- SuccessCode
- FailureCode
- SuccessText
- FailureText

Perform the following tasks to ensure that the adapter updates the standard SAP status code after it retrieves an IDoc:

- Set the UpdateStatus configuration property to `true` and set values for the SuccessCode and FailureCode configuration properties.
- Configure the inbound parameters of the partner profile of the logical system in SAP to receive the ALEAUD message type. Set the following properties to the specified values:

Table 14. Inbound properties of the logical system partner profile

SAP property	**Value**
Basic Type	ALEAUD01
Logical Message Type	ALEAUD
Function module	IDOC_INPUT_ALEAUD
Process Code	AUD1

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

Pass-through support for IDocs, and MQSeries link for R/3 link migration:

Both the inbound and outbound SAP adapters support a pass-through mode for IDocs.

In this mode, the bit stream for the IDoc is provided without any form of parsing. The bit stream can then be used directly in a message flow, and parsed by other parsers, or sent unchanged over transports.

Use the Adapter Connection wizard to select pass-through support: on the Configure settings for adapter pane, select `ALE pass-through IDoc` as the interface type.

A business object is created that contains one field, which is the bit stream of the IDoc. You can transform this business object in a Compute node to an MQSeries link for R/3 format message, as shown in the following example.

```
DECLARE ns NAMESPACE
'http://www.ibm.com/xmlns/prod/websphere/j2ca/sap/sapmatmas05';

CREATE COMPUTE MODULE test4_Compute
 CREATE FUNCTION Main() RETURNS BOOLEAN
 BEGIN
```

```
  CALL CopyMessageHeaders();
  -- CALL CopyEntireMessage();
  set OutputRoot.MQSAPH.SystemNumber = '00';
  set OutputRoot.BLOB.BLOB =
InputRoot.DataObject.ns:SapMatmas05.IDocStreamData;
  RETURN TRUE;
 END;

 CREATE PROCEDURE CopyMessageHeaders() BEGIN
  DECLARE I INTEGER 1;
  DECLARE J INTEGER;
  SET J = CARDINALITY(InputRoot.*[]);
  WHILE I < J DO
   SET OutputRoot.*[I] = InputRoot.*[I];
   SET I = I + 1;
  END WHILE;
 END;

 CREATE PROCEDURE CopyEntireMessage() BEGIN
  SET OutputRoot = InputRoot;
 END;
END MODULE;
```

You can also create a request business object from an MQSeries link for R/3 message, as shown in the following example.

```
CREATE COMPUTE MODULE test4_Compute
 CREATE FUNCTION Main() RETURNS BOOLEAN
 BEGIN
  set
OutputRoot.DataObject.ns:SapMatmas05.IDocStreamData =
InputRoot.BLOB.BLOB;
  RETURN TRUE;
 END;
END MODULE;
```

The name of the SapMatmas05 element depends on selections that you make when you run the Adapter Connection wizard.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

ALE pass-through IDoc business object structure:

During ALE processing, the adapter exchanges business objects with the SAP application.

The message tree contains a BLOB field that represents the IDoc. The BLOB field is modelled as a HexBinary type in the XSD file. The message tree contains the stream data (the BLOB field), which is the entire data of the IDoc. The transaction ID, iDoc type, and queue name fields contain information about that data and how it has been transferred.

The transaction ID (SAPTransactionID) is used to ensure once-only delivery of business objects, and the queue name (qRFCQueueName) specifies the name of the queue on the SAP server to which the IDocs should be delivered. If you are not using transaction IDs or queues, these properties are blank.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

Generic IDoc routing:

By using the SAPInput node in passthrough mode, WebSphere Message Broker can receive any IDoc and route it according to IDoc type.

The message set for a generic IDoc in ALE passthrough mode contains four fields. Three of these fields are control information:

- The transaction ID
- The queue name (if qRFC is used)
- The name of the IDoc type

The fourth field is a hexBinary field that contains the entire IDoc in a raw, unparsed format. You can use the DataObject parser to parse this raw format to a full logical structure of a specific IDoc if you provide a message set that contains the definition for that IDoc type. To parse raw IDocs, you must use an MQInput node with the `Message domain` property set to `DataObject`, and the `Message format` property set to `SAP ALE IDoc`.

When passthrough mode is not used, the message set contains a fully parsed IDoc structure. These structures are specific to each IDoc, therefore the message set must have a type defined for every IDoc that could be received by the `.inadapter` component. This message set is determined by the RFC Program ID that is configured on the adapter and the ALE or RFC configuration on the SAP side. This behavior can affect how message models are managed.

Whenever you need to develop message flows in isolation, where each flow is to handle a different type of IDoc, the ALE parsed mode is not appropriate because all flows have a common denominator (the message set), which needs to be changed whenever a new IDoc type is added.

By using the generic passthrough mode, you can create a routing message flow that uses the IDoc type field of the generic IDoc model to separate WebSphere MQ queues, for example. You can create message flows that deal with each different IDoc type. If the set of discovered IDocs is extended, you can create a message flow and a message set (which contains just the new IDoc), then deploy them, without the need to change existing message flows or message sets.

By using this method, you can also use a single RFC program ID to receive all IDoc types, while still allowing individual IDoc processing.

For instructions about how to route generic IDocs, see "Routing IDocs to separate message flows" on page 2055.

For more information about the pattern that is used in the task, see Data distribution SAP to WebSphere MQ: one-way (for IDoc). You can view patterns in the information center by using the links only when you use the information center that is integrated with the WebSphere Message Broker Toolkit, or when you use the online information center.

Related concepts:

"The ALE interfaces" on page 1959
The SAP Application Link Enabling (ALE) interface and ALE pass-through IDoc interface enable business process integration and asynchronous data communication between two or more SAP systems or between SAP and external systems. The data is exchanged in the form of Intermediate Documents (IDocs).

"Inbound processing for the ALE interface" on page 1963
The adapter supports inbound processing (from the SAP server to the adapter) for the ALE interface and the ALE pass-through IDoc interface.

"DataObject parser and domain" on page 1114
Use the DataObject domain to parse and write messages for WebSphere Adapters and CORBA applications.

Related tasks:

"Routing IDocs to separate message flows" on page 2055
You can use a pattern to process IDocs of various kinds with a single RFC program ID without having to redeploy or rediscover existing message sets and adapters, even when adding new types of IDoc.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

Related reference:

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

ALE business objects:

A business object is a structure that consists of data, the action to be performed on the data, and additional instructions for processing the data. The adapter client uses business objects to send data to SAP or to obtain data (through the adapter) from SAP.

The adapter uses the IDoc metadata that is generated by the Adapter Connection wizard to construct a business-object definition. This metadata contains ALE-related information such as segment information, field names, and an indication of whether the business object handles a single IDoc or an IDoc packet.

The Message tree contains a BLOB field that represents the IDoc.

How business-object definitions are created

You create business-object definitions by using the Adapter Connection wizard. The wizard connects to the application, discovers data structures in the application, and generates business-object definitions to represent them. It also generates other resources that are needed by the adapter, such as the interface information that indicates the input and output parameters.

For more information, see the following topics.

- "ALE business object structure" on page 1979
- "Transaction ID support" on page 1981

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

ALE business object structure:

During ALE processing, the adapter exchanges business objects with the SAP application. The business object represents an individual IDoc or an IDoc packet. This business object is a top-level wrapper object that contains one or more IDoc child objects, each one corresponding to a single IDoc. The same business object format is used for inbound and outbound processing.

Wrapper business object

The wrapper business object contains a transaction ID, a queue name, and one or more IDoc business objects. The transaction ID (SAPTransactionID) is used to ensure once-only delivery of business objects, and the queue name (qRFCQueueName) specifies the name of the queue on the SAP server to which the IDocs should be delivered. If you are not using transaction IDs or queues, these properties are blank.

For individual IDocs, the wrapper business object contains only one instance of an IDoc business object. For IDoc packets, the wrapper business object contains multiple instances of an IDoc business object.

IDoc business object

The IDoc business object contains the following objects:

- The control record business object contains the metadata required by the adapter to process the business object.
- The data record business object contains the business object data to be processed by the SAP application and the metadata required for the adapter to convert it to an IDoc structure for the RFC call.

Dummy keys are not used in WebSphere Message Broker.

Unparsed IDocs

For an unparsed IDoc, in which the data part of the IDoc is not parsed by the adapter, the IDoc business object contains a dummy key, a control record, and the IDoc data. The IDoc data is of hexBinary type and represents the whole data record containing segments in binary content.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

Transaction ID support:

An SAP transaction ID (TID) is contained in the ALE wrapper business object and is therefore available as a field in the message tree. You can use transaction ID support to ensure once-only delivery of ALE objects.

The most common reason for using transaction ID support is to ensure once and only once delivery of data. The SAP transaction ID property is always generated by the Adapter Connection wizard.

The message flow must determine how to store the SAP transaction ID and how to relate the SAP transaction ID to the data being sent to the adapter. When the events are successful, the message flow should not resubmit the event associated with this TID again to prevent the processing of duplicate events.

- If the message flow does not send an SAP transaction ID with the business object, the adapter returns one after executing the transaction.
- If the message flow has an SAP transaction ID, it needs to populate the SAP transaction ID property with that value before executing the transaction.

The SAP transaction ID can be used for cross-referencing with a global unique ID that is created for an outbound event. The global unique ID is something you can create for managing integration scenarios.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

Query interface for SAP Software:

The Query interface for SAP Software (QISS) provides you with the means to retrieve data from application tables on an SAP server or to query SAP application tables for the existence of data. The adapter can perform hierarchical data retrieval from the SAP application tables.

Query interface for SAP Software supports outbound interactions for read operations (RetrieveAll and Exists) only. You can use this interface in local transactions to look up records before write operations (Create, Update, or Delete). For example, you can use the interface as part of a local transaction to do an existence check on a customer before creating a sales order. You can also use the interface in non-transaction scenarios.

Query interface for SAP Software supports data retrieval from SAP application tables, including hierarchical data retrieval from multiple tables. The interface supports static as well as dynamic specification of `where` clauses for the queries.

The Adapter Connection wizard finds the application data tables in SAP, interprets the hierarchical relationship between tables, and constructs a representation of the tables and their relationship in the form of a business object. The wizard also builds a default `where` clause for the query.

You can control the depth of the data retrieval, as well as the amount of information, by using the maxRow and rowsSkip properties.

Query interface for SAP software (QISS) supports hierarchical data retrieval from the SAP application tables. By using this interface, the adapter can determine the

existence of data in SAP application tables, or retrieve all the data in SAP application tables. For example, to check if customer Bob exists in SAP system, the Adapter Connection wizard can be run to discover the SAP application table KNA1. The wizard then generates the business object for KNA1 along with other SCA service artifacts. At run time, the SAPRequest node passes the KNA1 business object to the adapter to call the QISS interface, the adapter retrieves the table data from SAP, and returns the result to the node.

For more information, see the following topics.

- "Outbound processing for the query interface for SAP Software" on page 1984
- "Business objects for the query interface for SAP Software" on page 1985

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

Outbound processing for the query interface for SAP Software:

You use the Query interface for SAP Software for outbound processing only.

The message flow that makes the request uses the interface information that was generated by the Adapter Connection wizard.

The following list describes the sequence of processing actions that result from an outbound request that uses the query interface for SAP Software.

1. The adapter receives a request, which includes a table object, from a message flow.

   The query business object can be in a container business object, or it can be received as a table business object.
2. The adapter determines, from the table object sent with the query, the name of the table to examine.
3. The adapter determines the columns to retrieve or examine.
4. The adapter determines the rows to retrieve or examine.
5. The adapter responds.
   - For a RetreiveAll operation, the adapter returns a result set in the form of a container of query business objects, which represent the data for each row retrieved from the table. If the query is received as a table business object (not inside a container), the rows are returned one at a time, as they are retrieved.
   - For the Exists operation, the adapter returns an indication of whether the data exists in the SAP table.
   - If no data exists, the adapter generates an exception.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

Business objects for the query interface for SAP Software:

A business object is a structure that consists of data, the action to be performed on the data, and additional instructions, if any, for processing the data. The input to the Query interface for SAP Software is a table business object. The table business object represents the columns in a table on the SAP server. The adapter uses the table business object to obtain data from tables on the SAP server.

How data is represented in business objects

The adapter uses metadata that is generated by the Adapter Connection wizard to construct a business-object definition.

The data in the business object represents the columns of the associated table in SAP.

How business objects are created

You create business-object definitions by using the Adapter Connection wizard. The wizard connects to the application, discovers data structures in the application, and generates business-object definitions to represent them. It also generates other resources that are needed by the adapter, such as the interface information that indicates the input and output parameters.

Business object structure

The table business object can be part of a container.

The table business object contains columns selected from the specified SAP table.

In addition to column information, the table business object also contains a query business object as the last parameter.

The properties of the query business object are sapWhereClause, sapRowsSkip, and sapMaxRows:

- The sapWhereClause property retrieves information from SAP tables. The default value is populated by the Adapter Connection wizard. The space character is used as the delimiter to parse the sapWhereClause.
- The sapMaxRows property is the maximum number of rows to be returned. The default value is `100`.
- The sapRowsSkip property is the number of rows to skip before retrieving data. The default value is `0`.

The tables can be modeled as hierarchical business objects. You specify the parent-child relationship of the tables in the Adapter Connection wizard.

Tables are linked by a foreign key to form parent-child relationships. The child table business object has a foreign key that references a property in the parent query business object.

In the KNA1 business object, notice the reference to SapAdrc, a child business object. The SapAdrc table object has a column named AddressNumber. This column has an associated property (ForeignKey) that contains a reference to the parent business object.

The return from the Query interface for SAP Software call for a RetrieveAll operation is a container of table objects.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

The Advanced event processing interface:

The Advanced event processing interface of the WebSphere Adapter for SAP Software is used for both inbound and outbound processing.

For inbound processing, it polls for events in SAP, converts them into business objects, and sends the event data as business objects to WebSphere Message Broker.

For outbound processing, the adapter processes events sent from an application to retrieve data from or update data in the SAP server.

For more information, see the following topics.

- "Outbound processing for the AEP interface" on page 1988
- "Inbound processing for the AEP interface" on page 1995
- "Business objects for the Advanced Event Processing (AEP) interface" on page 2001

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

Outbound processing for the AEP interface:

During outbound processing, business object data is converted into an ABAP handler function, which is called on the SAP server. When the data is returned by the ABAP handler function, the data is converted to a business object, and the business object is returned as a response.

The following list describes the sequence of processing actions that result from an outbound request that uses the Advanced event processing interface.

1. The adapter receives the Advanced event processing record, which contains business data along with the metadata.
2. The Advanced event processing interface of the adapter uses the metadata of the business object to obtain the type of IDoc specified and to reformat the business object data into the structure of that IDoc.
3. After it reformats the data, the adapter passes the business object data to an object-specific ABAP handler (based on the operation), which handles the integration with an SAP native API.

4. After the object-specific ABAP handler finishes processing the business object data, it returns the response data in IDoc format to the adapter, which converts it to the business object.
5. The adapter returns the results to the caller.

For more information, see the following topics.

- "ABAP handler overview" on page 1990
- "ABAP handler creation" on page 1992
- "Call Transaction Recorder wizard" on page 1994

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.
"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

ABAP handler overview:

An ABAP handler is a function module that gets data into and out of the SAP application database. For each business object definition that you develop, you must support it by developing a custom ABAP handler.

ABAP handlers are in the SAP application as ABAP function modules. ABAP handlers are responsible for adding business-object data into the SAP application database (for Create, Update, and Delete operations) or for using the business-object data as the keys to retrieving data from the SAP application database (for the Retrieve operation).

You must develop operation-specific ABAP handlers for each hierarchical business object that must be supported. If you change the business object definition, you must also change the ABAP handler.

An ABAP handler can use any of the SAP native APIs for handling the data. The following list contains some of the native APIs.

- Call Transaction

  Call Transaction is the SAP-provided mechanism for entering data into an SAP system. Call Transaction guarantees that the data adheres to the SAP data model by using the same screens an online user sees in a transaction. This process is commonly referred to as *screen scraping*.
- Batch data communication (BDC)

  Batch Data Communication (BDC) is an instruction set that SAP can follow to process a transaction without user intervention. The instructions specify the sequence in which the screens in a transaction are processed and which fields are populated with data on which screens. All of the elements of an SAP transaction that are exposed to an online user have identifications that can be used in a BDC.
- ABAP SQL

  ABAP SQL is the SAP proprietary version of SQL. It is database-independent and platform-independent, so that whatever SQL code you write, you can run it on any database and platform combination that SAP supports. ABAP SQL is similar in syntax to other versions of SQL and supports all of the basic database table commands such as update, insert, modify, select, and delete. For a complete description of ABAP SQL, see your SAP documentation.

  By using ABAP SQL, an ABAP handler can modify SAP database tables with business-object data for create, update, and delete operations. It can also use the business-object data in the where clause of an ABAP select statement as the keys.

  Do not use ABAP SQL to modify SAP tables, because it might corrupt the integrity of the database. Use ABAP SQL only to retrieve data.
- ABAP Function Modules and Subroutines

  From the ABAP handler, you can call ABAP function modules or subroutines that implement the required function.

The adapter provides the following tools to help in the development process:

- The adapter includes the Call Transaction Recorder wizard to assist you in developing the ABAP handlers that use call transactions or BDC sessions.
- The Adapter Connection wizard generates the required business objects and other resources for Advanced event processing. The business objects are based on IDocs, which can be custom or standard.
- The adapter provides samples that you can refer to for an understanding of the Advanced event processing implementation.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

ABAP handler creation:

For each IDoc object definition that you develop, you must support it by developing a custom ABAP handler.

You can use either standard IDocs or custom IDocs for the Advanced event processing interface. After defining the custom IDoc for an integration scenario, create an ABAP handler (function module) for each operation of the business object that must be supported.

Each function should have the following interface to ensure that adapter can call it:

```
*" IMPORTING
*" VALUE(OBJECT_KEY_IN) LIKE /CWLD/LOG_HEADER-OBJ_KEY OPTIONAL
*" VALUE(INPUT_METHOD) LIKE BDWFAP_PAR-INPUTMETHD OPTIONAL
*" VALUE(LOG_NUMBER) LIKE /CWLD/LOG_HEADER-LOG_NR OPTIONAL
*" EXPORTING
*" VALUE(OBJECT_KEY_OUT) LIKE /CWLD/LOG_HEADER-OBJ_KEY
*" VALUE(RETURN_CODE) LIKE /CWLD/RFCRC_STRU-RFCRC
*" VALUE(RETURN_TEXT) LIKE /CWLD/LOG_HEADER-OBJ_KEY
*" TABLES
*" IDOC_DATA STRUCTURE EDID4
*" LOG_INFO STRUCTURE /CWLD/EVENT_INFO
```

The following table provides information about the parameters:

Table 15. Interface parameters

Parameter	Description
OBJECT_KEY_IN	Should be no value.
INPUT_METHOD	Indicates whether the IDoc should be processed in a dialog (that is, through Call Transaction). Possible values are: " " - Background (no dialog) "A" - Show all screens "E" - Start the dialog on the screen where the error occurred "N" Default
LOG_NUMBER	Log Number.
OBJECT_KEY_OUT	Customer ID returned from the calling transaction.
RETURN_CODE	0 - Successful. 1 - Failed to retrieve. 2 - Failed to create, update, or delete.
RETURN_TEXT	Message describing the return code.
IDOC_DATA	Table containing one entry for each IDoc data segment. The following fields are relevant to the inbound function module: Docnum - The IDoc number. Segnam - The segment name. Sdata - The segment data.

Table 15. Interface parameters (continued)

Parameter	Description
LOG_INFO	Table containing details regarding events processed with either a success or error message.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

Call Transaction Recorder wizard:

The adapter provides the Call Transaction Recorder wizard to assist you in developing the ABAP handlers that use call transactions or BDC sessions.

The Call Transaction Recorder wizard enables you to generate sample code for call transactions to facilitate development. It generates sample code stubs for each screen that is modified during the recording phase.

To access this wizard, enter the /CWLD/HOME transaction in the SAP GUI.

The following example is sample code that is generated by the wizard. You can adopt this code in the ABAP Handler.

```
* Customer master: request screen chnge/displ cent.
perform dynpro_new using 'SAPMF02D' '0101' .

* Customer account number
perform dynpro_set using 'RF02D-KUNNR' '1' .

* Function Command
perform dynpro_set using 'BDC_OKCODE' '/00' .

* Function Command
perform dynpro_set using 'BDC_OKCODE' '/00' .

* Customer master: General data, CAM address, communication
perform dynpro_new using 'SAPMF02D' '0111' .

* Title
perform dynpro_set using 'SZA1_D0100-TITLE_MEDI' 'Mr.' .

* Function Command
perform dynpro_set using 'BDC_OKCODE' '=UPDA' .

* Call Transaction
Call Transaction 'XD02' using bdcdata
    mode input_mode
   update 'S'
  messages into bdc_messages.
```

The wizard does not generate the required business object. You use the Adapter Connection wizard to generate the business object.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere

Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

Inbound processing for the AEP interface:

The adapter uses the Advanced event processing interface to poll for events on the SAP server, to process the events, and to send them to an endpoint.

The following list describes the sequence of processing actions that result from an inbound request that uses the Advanced event processing interface.

1. A triggered event enters the event table with an initial status of prequeued.
2. When the adapter polls for events, the status of the event changes from prequeued to queued if there are no database locks for the combination of the user who created the event and the event key.
3. After the event is retrieved from the event table, the status of the event is updated to InProgress.

   If locks exist, the status of the event is set to locked and the event is requeued into the queue. Every event with a prequeued or locked status is updated with every poll. You can configure the polling frequency by using the Poll Frequency property.
4. After preprocessing all prequeued events, the adapter selects the events.

   The property Poll Quantity determines the maximum number of events returned for a single poll call.
5. For each event, the adapter uses the remote function specified for the Retrieve operation to retrieve the data and send it to the endpoint.

If the AssuredOnceDelivery property is set to true, an XID value is set for each event in the event store. After each event is picked up for processing, the XID value for that event is updated in the event table.

If, before the event is delivered to the endpoint, the SAP connection is lost or the application is stopped, and the event is consequently not processed completely, the XID column ensures that the event is reprocessed and sent to the endpoint. After the SAP connection is reestablished or the adapter starts up again, it first checks for events in the event table that have a value in the XID column. It then processes these events first and then polls the other events during the poll cycles.

6. After each event is processed, it is updated or archived in the SAP application.

   When the event is processed successfully, it is archived and then deleted from the event table.

   The adapter can also filter the events to be processed by business object type. The filter is set in the Event Filter Type property. This property has a comma-delimited list of business object types, and only the types specified in the property are picked for processing. If no value is specified for the property, no filter is applied and all the events are picked up for processing.

For more information, see the following topics.

- "Event detection" on page 1997
- "Event triggers" on page 1999

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

Event detection:

Event detection refers to the collection of processes that notify the adapter of SAP application object events. Notification includes, but is not limited to, the type of the event (object and operation) and the data key required for the external system to retrieve the associated data.

Event detection is the process of identifying that an event was generated in the SAP application. Typically, adapters use database triggers to detect an event. However, because the SAP application is tightly integrated with the SAP database, SAP allows very limited access for direct modifications to its database. Therefore, the event-detection mechanisms are implemented in the application transaction layer above the database.

Adapter-supported event detection mechanisms

The four event-detection mechanisms that are supported by the adapter are described in the following list:

- Custom Triggers, which are implemented for a business process (normally a single SAP transaction) by inserting event detection code at an appropriate point in the SAP transaction
- Batch programs, which involve developing an ABAP program containing the criteria for detecting an event
- Business workflows, which use the object-oriented event detection capabilities of SAP
- Change pointers, a variation of business workflows, which use the concept of change documents to detect changes for a business process

All these event-detection mechanisms support real-time triggering and retrieval of objects. In addition, custom triggers and batch programs provide the ability to delay the retrieval of events. An event whose retrieval is delayed is called a future event.

Each event detection mechanism has advantages and disadvantages that need to be considered when designing and developing a business object trigger. Keep in mind that these are only a few examples of event detection mechanisms. There are many different ways to detect events.

After you determine the business process to support (for example, sales quotes or sales orders) and determine the preferred event-detection mechanism, implement the mechanism for your business process.

When implementing an event detection mechanism, it is a good idea to support all of the functions for a business process in one mechanism. This limits the effect in the SAP application and makes event detection easier to manage.

Event table

Events that are detected are stored in an SAP application table. This event table is delivered as part of the ABAP component. The event table structure is as follows.

Table 16. Event table fields

Name	Type	Description
event_id	NUMBER	Unique event ID that is a primary key for the table.
object_name	STRING	Business object name.
object_key	STRING	Delimited string that contains the keys for the business object.
object_function	STRING	Operation corresponding to the event (Delete, Create, or Update).
event_priority	NUMBER	Any positive integer to denote the priority of the event.
event_time	DATE	Date and time when the event was generated.
event_status	NUMBER	Event processing status. Possible values are: 0 - Ready for poll 1 - Event delivered 2 - Event prequeued 3 - Event in progress 4 - Event locked -1 - Event failed
Xid	STRING	Unique XID (transaction ID) value for assured-once delivery.
event_user	STRING	User who created the event.
event_comment	STRING	Description of the event.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

Event triggers:

After an event is identified by one of the event-detection mechanisms, it is triggered by one of the adapter-delivered event triggers. Event triggers can cause events to be processed immediately or in the future.

The function modules that trigger events are described in the following list.

- /CWLD/ADD_TO_QUEUE

  This function module triggers events to the current event table for immediate processing.
- /CWLD/ADD_TO_QUEUE_IN_FUTURE

  This function module triggers events to the future event table to be processed at a later time.

Both functions are for real-time triggering.

Current event table

If the event will be triggered in real-time, /CWLD/ADD_TO_QUEUE_AEP commits the event to the current event table (/CWLD/EVT_CUR_AEP).

Specifically, it adds a row of data for the object name, verb, and key that represents the event.

Future event table

If an event needs to be processed at a future date, the processing that is described in the following list occurs.

1. A custom ABAP handler calls /CWLD/ADD_TO_QUEUE_IN_FUTURE_AEP with the event.
2. The /CWLD/ADD_TO_QUEUE_IN_FUTURE_AEP module commits the event to the future event table (/CWLD/EVT_FUT_AEP). Specifically, it adds a row of data for the object name, verb, and key that represents the event. In addition, it adds a Date row
3. The adapter-delivered batch program /CWLD/SUBMIT_FUTURE_EVENTS_AEP reads the future event table.
4. If scheduled to do so, the batch program retrieves events from the future event table.
5. After it retrieves an event, the batch program calls /CWLD/ADD_TO_QUEUE_AEP.
6. The /CWLD/ADD_TO_QUEUE_AEP module triggers the event to the current event table.

/CWLD/ADD_TO_QUEUE_IN_FUTURE_AEP uses the system date as the current date when it populates the Date row of the future event table.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the

WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

Business objects for the Advanced Event Processing (AEP) interface:

During advanced event processing, the adapter exchanges business objects with the SAP application.

How data is represented in business objects

Advanced event processing business objects are based on custom IDocs, standard IDocs, or extension IDocs available in the SAP system.

How business-object definitions are created

You create business-object definitions by using the Adapter Connection wizard. The wizard connects to the application, discovers data structures in the application, and generates business-object definitions to represent them. It also generates other resources that are needed by the adapter, such as the interface information that indicates the input and output parameters.

For custom interfaces that you want to support, you must first define the custom IDoc in the SAP system. You can then use the Adapter Connection wizard to discover this custom IDoc and build the required resources, including the business-object definition.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System

(EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

Overview of WebSphere Adapter for Siebel Business Applications:

With WebSphere Adapter for Siebel Business Applications, you can create integrated processes that exchange information with a Siebel application, without special coding.

Use the WebSphere Adapter for Siebel Business Applications to create integrated processes that exchange information with a Siebel application. With the adapter, an application can send requests to the Siebel Business Applications server or receive notifications of changes from the server.

The adapter creates a standard interface to the applications and data on the Siebel Business Applications server, so that the application developer does not need to understand the lower-level details (the implementation of the application or the data structures) on the Siebel Business Applications server. An application, for example, can send a request to the Siebel Business Applications server, to query or update an Account record, represented by a Siebel business component instance. It can also receive events from the server; for example, to be notified that a customer record has been updated. This behavior provides you with improved business workflow and processes to help manage your customer relations.

WebSphere Adapter for Siebel Business Applications complies with the Java Connector Architecture (JCA). JCA standardizes the way application components, application servers, and Siebel applications, such as Siebel Business Applications server, interact with each other.

The adapter configuration, which you generate with the Adapter Connection wizard, uses a standard interface and standard data objects. The adapter takes the standard data object sent by the application component and calls the Siebel Business Applications function. The adapter then returns a standard data object to the application component. The application component does not have to deal directly with the Siebel Business Applications function; it is the Siebel Business Applications adapter that calls the function and returns the results.

For example, the application component that needs the list of customers sends a standard business object with the range of customer IDs to Adapter for Siebel Business Applications. In return, the application component receives the results (the list of customers) in the form of a standard business object. The application component does not need to know how the function worked or how the data was structured. The adapter completes all the interactions with the Siebel Business Applications function.

Similarly, the message flow might want to know about a change to the data on the Siebel Business Applications server (for example, a change to a particular customer). You can generate an adapter component that polls for such events on the Siebel Business Applications server and notifies message flows of the update. In this case, the interaction begins at the Siebel Business Applications server.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for Siebel properties" on page 4092
Reference information to refer to when you connect to a Siebel application.

"SiebelInput node" on page 4740
Use the SiebelInput node to interact with a Siebel application.

"SiebelRequest node" on page 4745
Use the SiebelRequest node to interact with a Siebel application.

Technical overview of the Adapter for Siebel Business Applications:

WebSphere Adapter for Siebel Business Applications supports the exchange of information between your existing applications and Siebel Business Applications. The adapter supports Siebel entities, including business objects, business components, and business services. This enables you to create business processes that exchange data.

The adapter supports outbound processing (requests for data or services from an application to the Siebel application) and inbound processing (event notification from a Siebel application server to an application).

With Adapter for Siebel Business Applications, you can use existing or newly-created applications that run in a supported runtime environment to send requests for data and services to Siebel Business Applications.

You can also add event-generation triggers to Siebel business objects to have notifications of events, such as the creation, update, and deletion of a record, sent to one or more of your applications.

For more information, see the following topics.

- "Outbound processing for Siebel Business Applications" on page 2005
- "Inbound processing for Siebel Business Applications" on page 2006
- "Business objects for Siebel Business Applications" on page 2011
- "Adapter Connection wizard (Siebel)" on page 2012

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection

wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for Siebel properties" on page 4092
Reference information to refer to when you connect to a Siebel application.

"SiebelInput node" on page 4740
Use the SiebelInput node to interact with a Siebel application.

"SiebelRequest node" on page 4745
Use the SiebelRequest node to interact with a Siebel application.

Outbound processing for Siebel Business Applications:

WebSphere Adapter for Siebel Business Applications supports synchronous outbound processing. This means that when the component sends a request in the form of a WebSphere business object hierarchy to the adapter, the adapter processes the request and returns a WebSphere business object hierarchy that represents the result of the operation.

When the adapter receives a WebSphere business object hierarchy, the adapter processes it as follows:

1. The adapter extracts metadata from the WebSphere business object hierarchy.
2. It identifies the appropriate Siebel objects to access (for example, Siebel business objects and business components, or Siebel business services, integrations objects, and integration components) depending on the objects against which the artifacts were generated.
3. The adapter extracts the outbound operation to perform from the WebSphere business object hierarchy.
4. After accessing the required Siebel objects, the adapter retrieves, updates, deletes, or creates a Siebel business component hierarchy or performs the corresponding business service method on the integration component hierarchy.
5. If there are updates (Create, Update, Delete), the adapter populates that Siebel object (business or integration component hierarchy) with data from the hierarchy of WebSphere business objects.

Supported Outbound Operations

WebSphere Adapter for Siebel Business Applications supports the outbound operations shown in the following table.

Table 17. Supported outbound operations

Operation	Description
Create	Creates the business component.
Delete	Deletes the business component and its children.
Exists	Checks for the existence of incoming business objects. The output business object, "ExistsResult" will be returned with the boolean value populated.
Retrieve	Specifies the value of the business component.
RetrieveAll	Retrieves multiple instances of the same business component and populates it as the container object.

Table 17. Supported outbound operations (continued)

Operation	Description
Update	Updates the Siebel application with the incoming business object.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for Siebel Business Applications" on page 2002
With WebSphere Adapter for Siebel Business Applications, you can create integrated processes that exchange information with a Siebel application, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for Siebel properties" on page 4092
Reference information to refer to when you connect to a Siebel application.

"SiebelInput node" on page 4740
Use the SiebelInput node to interact with a Siebel application.

"SiebelRequest node" on page 4745
Use the SiebelRequest node to interact with a Siebel application.

Inbound processing for Siebel Business Applications:

WebSphere Adapter for Siebel Business Applications supports asynchronous inbound processing, which means that the adapter polls the Siebel Business Applications at specified intervals for events. When the adapter detects an event, it converts the event data into a business object and sends it to the component.

Before inbound processing can occur, a Siebel event business component must be created in the Siebel application (IBM2 for Siebel version 7.x and IBM_EVENT for Siebel version 8) and its name specified against the corresponding property in the adapter activation specification.

When the adapter detects an event for Siebel event business components or integration components, it processes the event by retrieving the updated data for the Siebel event business component or integration component and converting it into a business object. The adapter then sends the business object to the event business component. For example, if an event business component (an account) is updated, an event trigger adds an event record to the event business component. The adapter polls the event business component, retrieves the event record, and processes it.

When the adapter finds an event for a Siebel event business component, it processes the event in the following way:

1. The adapter retrieves the event information from the Siebel event business component.
2. The adapter retrieves the corresponding event business component instance hierarchy.
3. The adapter populates the associated WebSphere business object or business graph (if it was generated) with the values that it retrieves from the event business component.
4. The adapter sends a notification to each registered application.

If an inbound event in the event table fails or is invalid, the event status is updated to -1, which indicates an error in processing the event, and a resource exception message is issued that explains the reason for the error.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for Siebel Business Applications" on page 2002
With WebSphere Adapter for Siebel Business Applications, you can create integrated processes that exchange information with a Siebel application, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection

wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for Siebel properties" on page 4092
Reference information to refer to when you connect to a Siebel application.

"SiebelInput node" on page 4740
Use the SiebelInput node to interact with a Siebel application.

"SiebelRequest node" on page 4745
Use the SiebelRequest node to interact with a Siebel application.

Event store for Siebel Business Applications:

The event store is a persistent cache where event records are saved until the polling adapter can process them. To keep track of inbound events as they make their way through the system, the adapter uses an event store.

The creation, update, or deletion of an event record in the Siebel business application is an 'event'. Each time a business object is created, updated, or deleted, the adapter updates the status of the event in an event store.

For example, if you have a customer component and a new customer has just been added to it, this signals an update. If the adapter is configured to receive the events about the new update, there are triggers attached to the Siebel end and connected to the customer component. The triggers add a record to the event business component. The record contains information about the new customer, such as the customer ID. This information is stored in the object key. The object key is the unique identifier that provides the key name and value of the event business component that was updated (for example, `Id=1-20RT`). The object name is the WebSphere business-object name that represents the customer component (for example, `Account`). The adapter retrieves this event and the new customer information that relates to it. It then processes the event and delivers it to the export component.

During inbound processing, the adapter polls the event business components from the event store at regular intervals. Each time it polls, a number of events are processed by the adapter. Events are processed in ascending order of priority and ascending order of the event time stamp. In each poll cycle, new events are picked up. The adapter retrieves the value set in the object key field for the event and loads the business object that corresponds to it. The business object is created from the retrieved information and is delivered to the export components.

If you set the activation specification property AssuredOnceDelivery to `true`, a transaction ID (XID) value is set for each event in the event store. After the event is retrieved for processing, the XID value for it is updated in the event store and displayed in the XID column in the event business component. The event is then delivered to its corresponding export component, and the status is updated to show that the event has been successfully delivered. If the application is stopped or the event is not processed completely, the XID column is filled with a value. This ensures that the event is reprocessed and sent to the export component. After the connection is reestablished or the adapter starts again, the adapter checks for events in the event store that have a value in the XID column. The adapter processes these events first, then polls the other events during the poll cycles.

The adapter can either process all events or process events filtered by business-object type. You set the filter through the activation specification property, EventTypeFilter. This property contains a comma-delimited list of business-object types. Only the types specified in the property are processed. If the EventTypeFilter property is not set, all of the events are processed. If the FilterFutureEvents property is set to `true`, the adapter filters events based on the time stamp. The adapter compares the system time in each poll cycle to the time stamp on each event. If an event is set to occur in the future, it is not processed until that time.

After an event is successfully posted and delivered to the export component, the entry is deleted from the event store. Failed events (posting and delivery to the export component is unsuccessful), remain in the event store and are marked `-1`. This prevents duplicate processing.

Event store structure for Siebel business objects and business components

The `IBM2` event business component stores information about the event. The information stored is used by the resource adapter during event subscription to build the corresponding business object and send it to the registered export components. The information that is stored, and the structure of the event store used by the adapter, are shown in the following table.

Table 18. Event store structure for `IBM2` Siebel event business objects and business components

Field	Description	Example
Description	Any comment associated with the event.	Account Create Event
Event ID	The ID of the event row.	Automatically generated unique ID in Siebel (for example: `1-XYZ`)
Event timestamp	The time stamp for the event. The format is in *mm/dd/yyyy hh:mm:ss*	*02/24/2007 11:37:56*
Event type	The type of event.	Create, Update, or Delete
Object key	A unique identifier of the business-object row for which the event was created. It is a name-value pair consisting of the name of the property (key name) and value.	ID=1-20RT
Object name	The name of the business object for which the event was detected.	`IOAccountPRMANIICAccount`
Priority	The event priority.	1
Status	The event status. This is initially set to the value for a new event and updated by the adapter as it processes the event. The status can have one of the following values: • 0: Identifies a new event. • 1: Identifies an event that has been delivered to an export component. • -1: An error occurred while processing the event. This column cannot contain a null value.	0

Table 18. Event store structure for IBM2 Siebel event business objects and business components (continued)

Field	Description	Example
XID	The transaction ID. This is to ensure assured once-only delivery.	None

Event store structure for Siebel business services

The event is retrieved from the IBM2 event business component and the information is used to retrieve the event business component.

Table 19. Event store structure for IBM2 Siebel business services

Field	Description	Example
Description	Any comment associated with the event.	Account PRM ANI Event
Event ID	The ID of the event row.	Automatically generated unique ID in Siebel (for example: 1-XYZ)
Event timestamp	The time stamp for the event. The format is in *mm/dd/yyyy hh:mm:ss*	*02/24/2007 11:37:56*
Event type	The type of event.	Create, Update, or Delete
Object key	A unique identifier of the business-object row for which the event was created. It is a name value pair consisting of the name of the property (key name) and value.	Name=TestName;Location=BGM, where 'Name' and 'Location' are the keys in the integration component. 'TestName' and 'BGM' are the values specified, and ; is the event key delimiter.
Object name	The name of the business object for which the event was detected.	IOAccountPRMANIICAccount
Priority	The event priority.	1
Status	The event status. This is initially set to the value for a new event and updated by the adapter as it processes the event. The status can have one of the following values: • 0: Identifies a new event. • 1: Identifies an event that has been delivered to an export component. • -1: An error occurred while processing the event. This column cannot contain a null value.	0
XID	The transaction ID. This is to ensure 'assured once delivery'.	None

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for Siebel Business Applications" on page 2002
With WebSphere Adapter for Siebel Business Applications, you can create integrated processes that exchange information with a Siebel application, without

special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for Siebel properties" on page 4092
Reference information to refer to when you connect to a Siebel application.

"SiebelInput node" on page 4740
Use the SiebelInput node to interact with a Siebel application.

"SiebelRequest node" on page 4745
Use the SiebelRequest node to interact with a Siebel application.

Business objects for Siebel Business Applications:

To send data or obtain data from Siebel Business Applications, the adapter uses business objects. A business object is a structure that consists of data, the action to be performed on the data, and additional instructions, if any, for processing the data. The data can represent either a business entity, such as an invoice or an employee record, or unstructured text.

How business objects are created

You create business objects by using the Adapter Connection wizard, which connects to the application, discovers data structures in the application, and generates business objects to represent them. It also generates other resources that are needed by the adapter.

The Siebel business objects are created with long names by default. To generate business objects with shorter names, select **Generate business objects with shorter names** on the Configure Objects screen of the Adapter Connection wizard. For more information, see "Naming conventions for business objects representing Siebel business services" on page 4095.

Business object structure

The adapter supports business objects that are hierarchically structured. The top-level business object must have a one-to-one correspondence with the Siebel business component, and collections that occur in the top-level object are children of it. Information about the processed object is stored in the application-specific information for the object and each of its attributes.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for Siebel Business Applications" on page 2002
With WebSphere Adapter for Siebel Business Applications, you can create integrated processes that exchange information with a Siebel application, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for Siebel properties" on page 4092
Reference information to refer to when you connect to a Siebel application.

"SiebelInput node" on page 4740
Use the SiebelInput node to interact with a Siebel application.

"SiebelRequest node" on page 4745
Use the SiebelRequest node to interact with a Siebel application.

Adapter Connection wizard (Siebel):

The Adapter Connection wizard is a tool that you use to configure your adapter. The wizard establishes a connection to the Siebel server, discovers business objects and services (based on search criteria you provide), and generates business objects based on the services discovered.

By using the Adapter Connection wizard, you establish a connection to the Siebel server to browse the metadata repository on the Siebel server. You also specify connection information, such as the Connection URL, user name, and password needed to access the server.

The result of running the wizard is a message set project that contains the Siebel business objects and services along with the adapter.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for Siebel Business Applications" on page 2002
With WebSphere Adapter for Siebel Business Applications, you can create integrated processes that exchange information with a Siebel application, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for Siebel properties" on page 4092
Reference information to refer to when you connect to a Siebel application.

"SiebelInput node" on page 4740
Use the SiebelInput node to interact with a Siebel application.

"SiebelRequest node" on page 4745
Use the SiebelRequest node to interact with a Siebel application.

Overview of WebSphere Adapter for PeopleSoft Enterprise:

With the adapter for PeopleSoft Enterprise you can create integrated processes that exchange information with PeopleSoft Enterprise through a standard interface. This interface shields the message flow developer from having to understand the lower level details about implementation of the application or data structures used on the PeopleSoft Enterprise server.

With the adapter, a message flow can send a request, for example to a PeopleSoft Enterprise database to query a record in an HR table, or it can receive events from the server, such as notification that an employee record has been updated.

WebSphere Adapter for PeopleSoft Enterprise complies with the Java Connector Architecture (JCA), which standardizes the way application components, application servers, and enterprise information systems, such as a PeopleSoft Enterprise server, interact with each other.

The adapter component, which you generate with the Adapter Connection wizard uses a standard interface and standard data objects. The adapter component takes the standard data object sent by the message flow and calls the PeopleSoft function. It then returns a standard data object to the message flow. The message flow does not have to deal directly with the PeopleSoft function; it is the adapter component that calls the function and returns the results.

For example, the message flow that requested the list of employees sends a standard business object with the range of skill codes to the PeopleSoft adapter component. The message flow receives, in return, the results (the list of employees) in the form of a standard business object. The adapter component completes all the interactions directly with the PeopleSoft function.

Similarly, the message flow might need to be notified about a change to the data on the PeopleSoft Enterprise server (for example, a change to the skills set of a particular employee). You can generate an adapter component that listens for such events on the PeopleSoft Enterprise server and notifies message flows with the update. In this case, the interaction begins at the PeopleSoft Enterprise server.

For more information, see “Technical overview of the WebSphere Adapter for PeopleSoft Enterprise” on page 2015.

Related concepts:

“WebSphere Adapters nodes” on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

Related tasks:

“Preparing the environment for WebSphere Adapters nodes” on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

“Developing message flows that use WebSphere Adapters” on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

“Creating a custom event project in PeopleTools” on page 2083
The WebSphere Adapter requires an event project in PeopleSoft to perform asynchronous inbound event processing. Use PeopleTools to create the custom event project.

“Connecting to an EIS by using the Adapter Connection wizard” on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

“Debugging message flows that contain WebSphere Adapters nodes” on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for PeopleSoft properties" on page 4122
Reference information to refer to when you connect to a PeopleSoft application.

"PeopleSoftInput node" on page 4630
Use the PeopleSoftInput node to interact with a PeopleSoft application.

"PeopleSoftRequest node" on page 4635
Use the PeopleSoftRequest node to interact with a PeopleSoft application.

Technical overview of the WebSphere Adapter for PeopleSoft Enterprise:

The adapter supports the exchange of business data between the PeopleSoft Enterprise server and WebSphere Message Broker. It does so by connecting to two layers of PeopleTools application programming interface classes that reveal the underlying business data for integration.

The Adapter for PeopleSoft Enterprise establishes bidirectional connectivity with the PeopleSoft Enterprise server by connecting to two PeopleTools application programming interfaces as follows:

1. The adapter accesses the primary API layer to create a session instance and to connect to the application server through the Jolt port.
2. The adapter then accesses the PeopleSoft Component Interface API, which reveals underlying business objects, logic, and functions.

In PeopleSoft, a component is a set of pages grouped together for a business purpose (such as an employee profile), and a component interface is an API that provides synchronous access to a component from an external application. After the adapter connects to the component interface, the following entities are exposed to the adapter and available for integration:

- All business objects in the component interface definition
- PeopleCode methods associated with the underlying components
- Records, except searches and menu-specific processing options

For more information, see the following topics.

- "Outbound processing for PeopleSoft Enterprise" on page 2016
- "Inbound processing for PeopleSoft Enterprise" on page 2018
- "Business objects for PeopleSoft Enterprise" on page 2021

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for PeopleSoft Enterprise" on page 2013
With the adapter for PeopleSoft Enterprise you can create integrated processes that exchange information with PeopleSoft Enterprise through a standard interface. This interface shields the message flow developer from having to understand the lower level details about implementation of the application or data structures used on the

PeopleSoft Enterprise server.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Creating a custom event project in PeopleTools" on page 2083
The WebSphere Adapter requires an event project in PeopleSoft to perform asynchronous inbound event processing. Use PeopleTools to create the custom event project.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for PeopleSoft properties" on page 4122
Reference information to refer to when you connect to a PeopleSoft application.

"PeopleSoftInput node" on page 4630
Use the PeopleSoftInput node to interact with a PeopleSoft application.

"PeopleSoftRequest node" on page 4635
Use the PeopleSoftRequest node to interact with a PeopleSoft application.

Outbound processing for PeopleSoft Enterprise:

The Adapter for PeopleSoft Enterprise supports synchronous outbound request processing. Synchronous outbound processing means that when the message flow sends a request in the form of a business object to the adapter, the adapter processes the request and returns a business object representing the result of the operation to the message flow.

When the adapter receives a WebSphere business object hierarchy, adapter processes it as follows:

1. The adapter extracts metadata from the WebSphere business object hierarchy that identifies the appropriate PeopleSoft component interface to access.
2. The adapter extracts the outbound operation to perform from the WebSphere business object hierarchy.
3. After the adapter accesses the component interface, it sets the keys from values specified in the business objects. If key values are not generated, for example with a create operation, the PeopleSoft application generates key fields.
4. After it retrieves the PeopleSoft objects, the adapter instantiates an existing component interface to delete, retrieve, update, or create a component interface.

5. If there are updates (Create, Update), the adapter populates the component interface with data from the WebSphere business object hierarchy. If there are Deletes, the adapter populates the component interface only with StatusColumnName and value information.

For Create and Update operations, the adapter processes attributes in the order defined in the business object. For example, if there is a complex attribute between two simple attributes, the adapter processes the simple attribute at the first position, the complex attribute followed by the simple attribute. After the changes are made, the component interface is saved to commit the data to the PeopleSoft database.

Supported outbound operations

WebSphere Adapter for PeopleSoft Enterprise supports the following outbound operations:

Table 20. Supported outbound operations

Operation	Description
Create	Creates the business object.
Delete	Deletes the business object and its children. Because the adapter supports only logical deletes, objects are marked as deleted but not removed.
Exists	Checks for the existence of incoming business objects.
Retrieve	Retrieves the PeopleSoft component, and maps component data onto the business object hierarchy.
RetrieveAll	Retrieves multiple instances of the PeopleSoft component, and maps component data onto the business object hierarchy.
Update	Updates the corresponding PeopleSoft component with the incoming business object.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for PeopleSoft Enterprise" on page 2013
With the adapter for PeopleSoft Enterprise you can create integrated processes that exchange information with PeopleSoft Enterprise through a standard interface. This interface shields the message flow developer from having to understand the lower level details about implementation of the application or data structures used on the PeopleSoft Enterprise server.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Creating a custom event project in PeopleTools" on page 2083
The WebSphere Adapter requires an event project in PeopleSoft to perform asynchronous inbound event processing. Use PeopleTools to create the custom event project.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for PeopleSoft properties" on page 4122
Reference information to refer to when you connect to a PeopleSoft application.

"PeopleSoftInput node" on page 4630
Use the PeopleSoftInput node to interact with a PeopleSoft application.

"PeopleSoftRequest node" on page 4635
Use the PeopleSoftRequest node to interact with a PeopleSoft application.

Inbound processing for PeopleSoft Enterprise:

The WebSphere Adapter for PeopleSoft Enterprise supports inbound event processing.

Inbound event processing means that the adapter polls the PeopleSoft Enterprise server at specified intervals for events. When the adapter detects an event, it converts the event data into a business object and sends it to the message flow.

To use inbound event processing, you must create a custom event project in PeopleSoft, as described in "Creating a custom event project in PeopleTools" on page 2083.

For more information, see "Event store for PeopleSoft Enterprise" on page 2019.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for PeopleSoft Enterprise" on page 2013
With the adapter for PeopleSoft Enterprise you can create integrated processes that exchange information with PeopleSoft Enterprise through a standard interface. This interface shields the message flow developer from having to understand the lower level details about implementation of the application or data structures used on the PeopleSoft Enterprise server.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Creating a custom event project in PeopleTools" on page 2083
The WebSphere Adapter requires an event project in PeopleSoft to perform asynchronous inbound event processing. Use PeopleTools to create the custom event project.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for PeopleSoft properties" on page 4122
Reference information to refer to when you connect to a PeopleSoft application.

"PeopleSoftInput node" on page 4630
Use the PeopleSoftInput node to interact with a PeopleSoft application.

"PeopleSoftRequest node" on page 4635
Use the PeopleSoftRequest node to interact with a PeopleSoft application.

Event store for PeopleSoft Enterprise:

The event store is a table that holds events that represent data changes until the polling adapter can process them. The adapter uses the event store to keep track of event entities.

To use inbound processing, you must use PeopleTools Application Designer to create a custom project for event notification. The custom project uses two PeopleCode functions that determine the way future events are processed, and the custom project creates the event store the adapter needs for inbound processing. Each time a business object is created, updated, or deleted, the PeopleCode function used in the project and then added to the component interface inserts a new record in the event store, with the appropriate object name, keys, and status value.

With inbound processing, the adapter polls the event entities from the event store at configured poll intervals. In each poll call, a configured number of events are processed by the adapter. The order of event processing is based on the ascending order of priority and the ascending order of the event time stamp. The events with the status, Ready for poll (0), are picked up for polling in each poll cycle. The adapter uses the object name and object key to retrieve the corresponding business object.

If you set the activation specification property AssuredOnceDelivery to `true`, an XID (transaction ID) value is set for each event in the event store, and it is used to ensure that an event is delivered only once to the target application. After an event is obtained for processing, the XID value for that event is updated in the event store. The event is then delivered to its corresponding export component, and its status is updated to show that event delivery has been completed. If the application is stopped before the event can be delivered to the export component

or if delivery has failed, the event might not be processed completely. In this case, the XID value represents in-progress status, and the XID column ensures that the event is reprocessed and sent to the export component. Once the database connection is re-established or the adapter starts again, the adapter checks for events in the event table that have a value in the XID column of Ready for Poll (0). The adapter processes these events first, and then polls the other events during the poll cycles.

The adapter uses special processing for events that have status code (99), which indicates that they will occur in the future. During a poll cycle, when the adapter retrieves events with a future status, the adapter compares the system time with the time stamp on each event. If the event time is earlier than or equal to the system time, the adapter processes the event and changes the event status to Ready for Poll (0).

If you want to the adapter to process future status events in the present time, use the function `IBM_PUBLISH_EVENT` instead of `IBM_FUTURE_PUBLISH_EVENT`. Doing so means that the event is identified as Ready to Poll (0) instead of Future (99).

As events are retrieved and processed from the event store, the status of the event changes to reflect the cycle, as shown in the following table:

Table 21. Event status values

Status short name	Description	Event table value
Error processing event	An error occurred during event processing.	-1
Ready for poll	The event has not yet been picked up by the adapter. The event is ready to be picked up.	0
Success	The event has been delivered to the event manager.	1
Deleted	The event has been processed successfully and is removed from the event store.	4
Future Events	These events should be processed at a future date.	99

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for PeopleSoft Enterprise" on page 2013
With the adapter for PeopleSoft Enterprise you can create integrated processes that exchange information with PeopleSoft Enterprise through a standard interface. This interface shields the message flow developer from having to understand the lower level details about implementation of the application or data structures used on the PeopleSoft Enterprise server.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System

(EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Creating a custom event project in PeopleTools" on page 2083
The WebSphere Adapter requires an event project in PeopleSoft to perform asynchronous inbound event processing. Use PeopleTools to create the custom event project.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for PeopleSoft properties" on page 4122
Reference information to refer to when you connect to a PeopleSoft application.

"PeopleSoftInput node" on page 4630
Use the PeopleSoftInput node to interact with a PeopleSoft application.

"PeopleSoftRequest node" on page 4635
Use the PeopleSoftRequest node to interact with a PeopleSoft application.

Business objects for PeopleSoft Enterprise:

To send data or obtain data from PeopleSoft Enterprise, the adapter uses business objects. A business object is a structure that consists of data, the action to be performed on the data, and additional instructions, if any, for processing the data. The data can represent either a business entity, such as an invoice or an employee record, or unstructured text.

How business objects are created

You create business objects by using the Adapter Connection wizard. The wizard connects to the application, discovers data structures in the application, and generates business objects to represent them. It also generates other resources that are needed by the adapter.

Business object structure

The adapter supports business objects that are hierarchically structured. The top-level business object must have a one-to-one correspondence with the PeopleSoft component interface, and collections that occur in the top-level object are children of it. Information about the processed object is stored in the application-specific information for the object and each of its attributes.

The following table describes the attributes that form a business object.

Attribute property	Description
Name	Indicates the name of the business object attribute.
Type	Indicates the type of the Business Object attribute. The adapter uses character mapping between PeopleSoft component property types and the generated business object attribute types. PeopleSoft component property types map to generated attribute types in the following manner: CHAR maps to attribute type String NUMBER maps to attribute type BigDecimal LONG maps to attribute type Long SIGN maps to attribute type BigDecimal DATE maps to attribute type Date TIME maps to attribute type Time DTTM maps to attribute type DateTime
Key	Child business objects have their own keys that have the primary key application-specific information. They also inherit keys from their parent business object.
Cardinality	Single cardinality for simple attributes; multiple cardinality for container attributes.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for PeopleSoft Enterprise" on page 2013
With the adapter for PeopleSoft Enterprise you can create integrated processes that exchange information with PeopleSoft Enterprise through a standard interface. This interface shields the message flow developer from having to understand the lower level details about implementation of the application or data structures used on the PeopleSoft Enterprise server.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Creating a custom event project in PeopleTools" on page 2083
The WebSphere Adapter requires an event project in PeopleSoft to perform asynchronous inbound event processing. Use PeopleTools to create the custom event project.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for PeopleSoft properties" on page 4122
Reference information to refer to when you connect to a PeopleSoft application.

"PeopleSoftInput node" on page 4630
Use the PeopleSoftInput node to interact with a PeopleSoft application.

"PeopleSoftRequest node" on page 4635
Use the PeopleSoftRequest node to interact with a PeopleSoft application.

Overview of WebSphere Adapter for JD Edwards EnterpriseOne:

With the WebSphere Adapter for JD Edwards EnterpriseOne, you can create integrated processes that include the exchange of information with a JD Edwards EnterpriseOne server, without special coding.

The adapter provides a standard interface that eliminates the need for the component to understand the lower-level implementation details or data structures of the application. By using the adapter, a component (the program or piece of code that performs a specific business function) can send requests to the JD Edwards EnterpriseOne server (for example, to query a customer record in a table or to update an order document).

WebSphere Adapter for JD Edwards EnterpriseOne complies with the Java 2 Platform, Enterprise Edition (J2EE) Connector Architecture (JCA). JCA standardizes the way in which application components, application servers, and Enterprise Information Systems (EIS), such as a JD Edwards EnterpriseOne server, interact with each other. WebSphere Adapter for JD Edwards EnterpriseOne makes it possible for JCA-compliant application servers to connect to and interact with the JD Edwards EnterpriseOne server. Clients running on the JCA-compliant server can then communicate with the JD Edwards EnterpriseOne server in a standard way (by using business objects or JavaBeans).

Suppose a medium-sized retail company uses JD Edwards EnterpriseOne to coordinate most of its business operations. JD Edwards EnterpriseOne includes a business function that can return a real-time list of inventory items for its 100 stores located across the United States. An application component might be able to use this business function as part of an overall business process. For example, an employee of a retail company can access the real-time list of available inventory items, therefore providing correct, real-time information to a customer.

For more information, see "Technical overview of the WebSphere Adapter for JD Edwards EnterpriseOne" on page 2024.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Technical overview of the WebSphere Adapter for JD Edwards EnterpriseOne" on page 2024
The IBM WebSphere Adapter for JD Edwards EnterpriseOne provides a way for applications to interact with data on JD Edwards EnterpriseOne applications.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for JD Edwards EnterpriseOne properties" on page 4146
Reference information to refer to when you connect to a JD Edwards EnterpriseOne application.

Technical overview of the WebSphere Adapter for JD Edwards EnterpriseOne:

The IBM WebSphere Adapter for JD Edwards EnterpriseOne provides a way for applications to interact with data on JD Edwards EnterpriseOne applications.

The adapter processes requests using one of two types of business objects: business functions and XML Lists. A business function is a business object container that can contain one or many business objects that can be processed as a single transaction. An XML List is a single business object that can query a table and return multiple records.

You create business objects by using the Adapter Connection wizard. The business objects that are generated by the Adapter Connection wizard have predefined business object definitions.

For more information, see the following topics:

- "Outbound processing (JD Edwards)" on page 2025
- "Inbound processing for JD Edwards EnterpriseOne" on page 2027
- "Business objects for JD Edwards EnterpriseOne" on page 2031
- "The Adapter Connection wizard (JD Edwards)" on page 2032

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for JD Edwards EnterpriseOne" on page 2023
With the WebSphere Adapter for JD Edwards EnterpriseOne, you can create integrated processes that include the exchange of information with a JD Edwards EnterpriseOne server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for JD Edwards EnterpriseOne properties" on page 4146
Reference information to refer to when you connect to a JD Edwards EnterpriseOne application.

Outbound processing (JD Edwards):

The WebSphere Adapter for JD Edwards EnterpriseOne supports synchronous outbound request processing. When the adapter receives a request from the module, in the form of a business object, the adapter processes the request and returns the result, when applicable, in a business object.

The adapter processes requests by using one of two types of business object: business functions and XML Lists. A business function is a business object container that can contain one or many business objects, which can be processed as a single transaction. An XML List is a single business object that can query a table and return multiple records. The component sends a request in the form of a WebSphere business object hierarchy to the adapter, the adapter processes the request, and returns a WebSphere business object hierarchy that represents the result of the operation.

When the adapter receives a WebSphere business object hierarchy, the adapter processes it in the following way.

1. The adapter extracts metadata from the WebSphere business object hierarchy.
2. The adapter identifies the appropriate JD Edwards EnterpriseOne objects to access (for example, JD Edwards EnterpriseOne business objects and business components, or JD Edwards EnterpriseOne business function, or JD Edwards EnterpriseOne XML List) depending on the objects against which the artifacts were generated.
3. The adapter extracts the outbound operation to complete from the WebSphere business object hierarchy.

4. After accessing the required JD Edwards EnterpriseOne objects, the adapter retrieves the data for XML List, or completes the corresponding business function method.
5. If updates are involved (Create, Update, Delete), the adapter populates that JD Edwards EnterpriseOne object (business function or XML List hierarchy) with data from the hierarchy of WebSphere business objects.

Supported outbound operations

When the adapter receives a request, it processes the request by using the JD Edwards EnterpriseOne Dynamic Java connector to call either a business function or an XML List.

Business functions support the following types of operation:

- Create
- Delete
- Execute
- Retrieve
- Update

XML Lists support the retrieveAll operation

Operations	Description
Create	Creates the business object.
Delete	Deletes the business object and its children. The adapter supports only logical deletes, therefore objects are marked as deleted but not removed.
Exists	Checks for the existence of incoming business objects.
Retrieve	Retrieves the JD Edwards EnterpriseOne component, and maps component data onto the business object.
RetrieveAll	Retrieves multiple instances of the JD Edwards EnterpriseOne component, and maps component data onto the business object.
Update	Updates the corresponding JD Edwards EnterpriseOne component with the incoming business object.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for JD Edwards EnterpriseOne" on page 2023
With the WebSphere Adapter for JD Edwards EnterpriseOne, you can create integrated processes that include the exchange of information with a JD Edwards EnterpriseOne server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for JD Edwards EnterpriseOne properties" on page 4146
Reference information to refer to when you connect to a JD Edwards EnterpriseOne application.

Inbound processing for JD Edwards EnterpriseOne:

The WebSphere Adapter for JD Edwards EnterpriseOne supports asynchronous inbound processing. The adapter polls the JD Edwards EnterpriseOne server at specified intervals for events. When the adapter receives an event, it converts the event data into a business object and sends the business object to the component.

The WebSphere Adapter for JD Edwards EnterpriseOne supports real-time events. A real-time event is a business transaction that provides information from the JD Edwards EnterpriseOne server that can be used to interoperate with a third-party system. Real-time events can be generated wherever business functions run, such as HTML, WIN32, and enterprise servers. Real-time events are useful for producing notifications in real time. The adapter supports both single and container real-time events.

When the adapter gets a real-time event from the JD Edwards EnterpriseOne transaction server by calling the JD Edwards EnterpriseOne Dynamic Java Connector API, it parses the content of this real-time event and converts it into a business object. The adapter then sends the business object to the event endpoints. For example, if a company is updated, the JD Edwards EnterpriseOne server captures this change immediately, and one real-time event is generated by the JD Edwards EnterpriseOne transaction server. The adapter then communicates with the JD Edwards EnterpriseOne transaction server, retrieves this real-time event, and processes it. After converting it into a business object, the adapter delivers this business object to the event endpoint.

WebSphere Adapter for JD Edwards EnterpriseOne processes events in the following way.

1. The adapter calls the JD Edwards EnterpriseOne Dynamic Java Connector API to get a real-time event.
2. The adapter parses the content of this real-time event.
3. The adapter populates the associated business object with the values that it retrieves from the payload of this real-time event.
4. The adapter sends the generated business object to each registered application.

Before inbound processing can occur, the JD Edwards EnterpriseOne server must be configured to support real-time events.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for JD Edwards EnterpriseOne" on page 2023
With the WebSphere Adapter for JD Edwards EnterpriseOne, you can create integrated processes that include the exchange of information with a JD Edwards EnterpriseOne server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for JD Edwards EnterpriseOne properties" on page 4146
Reference information to refer to when you connect to a JD Edwards EnterpriseOne application.

Event persistence (JD Edwards):

The WebSphere Adapter for JD Edwards EnterpriseOne supports event persistence for inbound processing in case of abrupt termination. Event persistence (or assured-once delivery) is a way to make sure that events are delivered once, and only once, to the endpoint in the case of a failure.

During event processing, the adapter persists the event state in an event store located on the data source. You must set up this data source before you can create the event store. To use the recovery feature, set the `AssuredOnceDelivery` property in the activation specification to `True`. This recovery feature is set to `True` by default.

When a failed event occurs, the adapter tries to deliver the event once more only. If the event fails again, its status is marked as FAILED and the event remains in the event store until it is either deleted manually or has its status changed. If the adapter fails or stops before the event is marked as FAILED, when it recovers, the

adapter retrieves the event from the event store with the stored data and attempts to redeliver the event, marking its status as FAILED only after the default retry limit of one has been exceeded again.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for JD Edwards EnterpriseOne" on page 2023
With the WebSphere Adapter for JD Edwards EnterpriseOne, you can create integrated processes that include the exchange of information with a JD Edwards EnterpriseOne server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for JD Edwards EnterpriseOne properties" on page 4146
Reference information to refer to when you connect to a JD Edwards EnterpriseOne application.

Event store (JD Edwards):

The event store is a persistent cache where event records are saved until the polling adapter can process them. The adapter uses event stores to record inbound events as they make their way through the system.

Each time a real-time event is received, the adapter updates the status of the event in an event store. The status of each event is continually updated by the adapter for recovery purposes until the events are delivered to the endpoint. If the adapter detects that no event store exists for the inbound module in WebSphere Message Broker, it creates one automatically when the application is deployed to the run time. Each event store that is created by the adapter is associated with a specific inbound module. The adapter does not support multiple adapter modules pointing to the same event store.

When the adapter polls the JD Edwards EnterpriseOne transaction server and receives a real-time event, it creates an entry in the event store for each event that matches the search criteria specified in the activation specification properties. The adapter records the status of each new entry as NEW.

If a real-time event is successfully delivered, the corresponding event store entries are deleted. For failed events, the entries remain in the event store.

Assured once delivery

The JD Edwards EnterpriseOne transaction server provides guaranteed event delivery quality of service. All the real-time events to which the adapter subscribes are delivered to the adapter without any loss. The JD Edwards EnterpriseOne transaction server can send duplicate real-time events to the adapter, therefore the adapter provides assured once event delivery; each event is delivered once and only once. To enable assured once delivery, you must set the `AssuredOnceDelivery` activation specification property to `True`.

When you set the `AssuredOnceDelivery` activation specification property to `True`, the `AutoAcknowledge` activation specification property is set to `False` automatically. You can also set assured once delivery by using the JDEdwardsConnection configurable service; for more information, see "Configurable services properties" on page 3766.

When a real-time event is obtained, it is processed in the following way.

1. The event is delivered to its corresponding endpoint.
2. The event entry is deleted from the event store.
3. An acknowledgment is issued to the JD Edwards EnterpriseOne transaction server.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for JD Edwards EnterpriseOne" on page 2023
With the WebSphere Adapter for JD Edwards EnterpriseOne, you can create integrated processes that include the exchange of information with a JD Edwards EnterpriseOne server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for JD Edwards EnterpriseOne properties" on page 4146
Reference information to refer to when you connect to a JD Edwards EnterpriseOne application.

Business objects for JD Edwards EnterpriseOne:

A business object is a structure that consists of data, the action to be performed on the data, and additional instructions, if any, for processing the data. The data can represent either a business entity, such as an invoice or an employee record, or unstructured text. The adapter uses business objects to send data to or obtain data from the JD Edwards EnterpriseOne server.

How the adapter uses business objects

The WebSphere Adapter for JD Edwards EnterpriseOne uses the JD Edwards EnterpriseOne Dynamic Java Connector APIs to communicate with the JD Edwards EnterpriseOne server. The adapter exchanges information with JD Edwards EnterpriseOne through business functions, XML List calls, and the real-time event mechanism.

How business objects are created

You create business objects by using the Adapter Connection wizard, which connects to the application, discovers data structures in the application, and generates business objects to represent them. It also generates other resources needed by the adapter.

Business object structure

The adapter supports processing of hierarchical business objects. A container business object representing a JD Edwards EnterpriseOne operation is a wrapper object that contains single or multiple child business function objects, also called simple business function objects. Each business function object represents a specific function call in the JD Edwards EnterpriseOne application.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for JD Edwards EnterpriseOne" on page 2023
With the WebSphere Adapter for JD Edwards EnterpriseOne, you can create integrated processes that include the exchange of information with a JD Edwards EnterpriseOne server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System

(EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for JD Edwards EnterpriseOne properties" on page 4146
Reference information to refer to when you connect to a JD Edwards EnterpriseOne application.

The Adapter Connection wizard (JD Edwards):

The Adapter Connection wizard is a tool that you use to create services. The Adapter Connection wizard establishes a connection to the JD Edwards EnterpriseOne server, discovers services (based on search criteria that you provide), and generates business objects, interfaces, and import or export files, based on the services that are discovered.

See the WebSphere Adapter for JD Edwards EnterpriseOne Information Center for further information.

By using WebSphere Message Broker, you establish a connection to the JD Edwards EnterpriseOne server to browse the metadata repository on the JD Edwards EnterpriseOne server.

You specify connection information (such as the user name and password needed to access the server), and you specify the method of operation that you want to use for an outbound or inbound request:

- For an outbound request, you specify either a business function or an XML List.
- For an inbound request, you specify real-time events.

You can then provide search criteria and select the information (for example, by using the search filter with libraries for business functions, or by selecting tables for XML Lists).

The result of running the Adapter Connection wizard is an adapter connection project and a message set project that contain the interfaces and business objects as well as the adapter.

The Adapter Connection wizard also produces an import file (for outbound processing) or an export file (for inbound processing).

- The import file contains the managed connection factory property settings that you provide in the wizard.

- The export file contains the activation specification property settings you provide in the wizard.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for JD Edwards EnterpriseOne" on page 2023
With the WebSphere Adapter for JD Edwards EnterpriseOne, you can create integrated processes that include the exchange of information with a JD Edwards EnterpriseOne server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters"
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for JD Edwards EnterpriseOne properties" on page 4146
Reference information to refer to when you connect to a JD Edwards EnterpriseOne application.

Developing message flows that use WebSphere Adapters

For information about how to develop message flows that use WebSphere Adapters, see the following topics.

About this task

- "Preparing your system to use WebSphere Adapters nodes" on page 2034
- "Activating IBM Tivoli License Manager for WebSphere Adapters" on page 2036
- "Connecting to an EIS by using the Adapter Connection wizard" on page 2037
- "Configuring EIS connections to expire after a specified time" on page 726
- "Configuring WebSphere Adapters nodes for secondary adapters" on page 2040
- "Calling new services from a WebSphere Adapters request node without changing existing deployed resources" on page 2042
- "Handling new event types from a Websphere Adapters input node without changing existing deployed resources" on page 2044
- "Interacting with an SAP application" on page 2046

- "Interacting with a Siebel application" on page 2066
- "Interacting with a PeopleSoft application" on page 2080
- "Interacting with a JD Edwards application" on page 2088

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

"Tuning the SAP adapter for scalability and performance" on page 3278
You can configure the number of listeners on the adapter and the number of additional instances on the message flow to prevent delays when processing synchronous calls from SAP.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

WebSphere Adapters technotes

WebSphere Adapters Library web page

Preparing your system to use WebSphere Adapters nodes:

Before you can connect to an Enterprise Information System (EIS), you must prepare your system by adding external software dependencies and configuring the EIS to work with the WebSphere Adapter.

Before you begin

Before you start:

- For general background information, read "WebSphere Adapters nodes" on page 1914
- Check for the latest information about WebSphere Adapters at WebSphere Adapters technotes.
- Check for the latest information about support for adapters on different operating systems at WebSphere Message Broker Requirements.
- Check the mode of your broker, because it can affect the number of execution groups and message flows that you can deploy, and the types of node that you can use. For more information, see "Restrictions that apply in each operation mode" on page 3657 and "Checking the operation mode of your broker" on page 657.
- Enable the WebSphere Adapters nodes in the broker runtime environment; see "Preparing the environment for WebSphere Adapters nodes" on page 717.

- If you want to use the IBM Tivoli License Manager (ITLM), perform the steps in "Activating IBM Tivoli License Manager for WebSphere Adapters" on page 2036.
- To see how the WebSphere Adapters work, look at the following samples:
  - Twineball Example EIS Adapter
  - SAP Connectivity

  You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

About this task

Perform the following steps, in the order shown, to prepare your system to use WebSphere Adapter nodes.

Procedure

- **SAP**
  1. Follow the instructions in "Adding external software dependencies for SAP" on page 2048.
  2. Follow the instructions in "Configuring the SAP server to work with the adapter" on page 2050.
- **Siebel**
  1. Follow the instructions in "Adding external software dependencies for Siebel" on page 2068.
  2. Follow the instructions in "Creating the event store manually" on page 2072.
- **PeopleSoft**
  1. Follow the instructions in "Adding external software dependencies for PeopleSoft" on page 2081.
  2. Follow the instructions in "Creating a custom event project in PeopleTools" on page 2083.
- **JD Edwards**
  1. Follow the instructions in "Adding external software dependencies for JD Edwards EnterpriseOne" on page 2090.

What to do next

After you have prepared your system, connect to an EIS by following the instructions in "Connecting to an EIS by using the Adapter Connection wizard" on page 2037.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Creating a custom event project in PeopleTools" on page 2083
The WebSphere Adapter requires an event project in PeopleSoft to perform asynchronous inbound event processing. Use PeopleTools to create the custom event project.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

WebSphere Adapters technotes

Activating IBM Tivoli License Manager for WebSphere Adapters:

If you want to use IBM Tivoli License Manager (ITLM), you must activate it for each of the WebSphere Adapters.

About this task

ITLM enables you to monitor the usage of IBM (and other) software products. For more information, see the IBM Tivoli License Manager Information Center or the IBM Tivoli License Manager website.

The following steps describe how to activate the ITLM file for each of the adapters.

Procedure

1. Locate the ITLM directory for the adapter.
   - For SAP: *install_dir*/itlm/SAP
   - For Siebel: *install_dir*/itlm/Siebel
   - For PeopleSoft: *install_dir*/itlm/PeopleSoft
2. Remove the inactive file extension from the file in the ITLM directory so that it ends with .sys2.

Results

After you have performed these steps, when you run ITLM, the adapter is visible.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and

PeopleSoft.

Related tasks:

"Installing Tivoli License Manager" on page 301
IBM Tivoli License Manager (ITLM) enables you to monitor the use of IBM (and other) software products. WebSphere Message Broker includes support for ITLM Version 2.1.

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

Related reference:

"Directory structures after installation" on page 3633
When you install WebSphere Message Broker components, the installation program creates a structure of subdirectories under the directory that you specified as the installation directory. The exact structure depends on the platform on which you have installed WebSphere Message Broker, and the components that you have installed. If you install the WebSphere Message Broker Toolkit, other directories might be created for IBM Installation Manager, the Shared Resources Directory, and the package group in which you install the toolkit.

Connecting to an EIS by using the Adapter Connection wizard:

Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

Before you begin

Before you start:

- Complete the preparatory tasks listed in "Developing message flows that use WebSphere Adapters" on page 2033.
- Before you run the Adapter Connection wizard, gather the information that you need from the system administrator. You need information to connect to the EIS, and information to discover objects. For a list of the information that you might need to gather for each EIS, see the appropriate topic:
  - "Interacting with an SAP application" on page 2046
  - "Interacting with a Siebel application" on page 2066
  - "Interacting with a PeopleSoft application" on page 2080
  - "Interacting with a JD Edwards application" on page 2088

About this task

A message flow that uses one of the WebSphere Adapters requires the following resources:

- One or more message flows that contain one or more WebSphere Adapters nodes
- A message set that contains the XML Schema Definitions (XSD) for each business object in the Enterprise Information System (EIS)
- An adapter component file for the WebSphere Adapter that is being used

The Adapter Connection wizard creates these resources automatically.

You can take an adapter component that was created by using the Adapter Connection wizard, and update it with newly discovered objects from the EIS by running the Adapter Connection - Iterative Discovery wizard. This facility is known as *iterative discovery*. You can either add the new objects without modifying existing objects, or replace existing objects. For more information, see "Enhancing existing adapters with newly discovered objects" on page 2063.

The following steps describe how to connect to an EIS.

Procedure

1. Click **File** > **New** > **Adapter Connection**. The Adapter Connection wizard opens.
2. Follow the instructions in the wizard. To see a description of each field within the wizard, hold the mouse pointer over the field.

   Ensure that inbound and outbound SAP IDocs have different names if they are stored in the same message set. For more information, see "An error is issued when you use the message set that is generated by the Adapter Connection wizard" on page 3431.

   When you have completed the steps in the wizard, the specified message set project contains a message set with a message type for each business object that is to be used, and the specified message flow project references the message set project.

   When the Adapter Connection wizard completes, the WebSphere Message Broker Toolkit displays a message that prompts you to drag the adapter component onto the message flow canvas.
3. Ensure that a message flow is open in the Message Flow editor so that the message flow canvas is available.
4. In the Broker Development view, expand the folders beneath the message set until you see the adapter component, which has a suffix of inadapter or outadapter.
5. Drag the adapter component onto the message flow canvas. The component appears as a message flow node.
6. Configure the node, as described in .
7. When you have developed and saved your message flow, deploy it by following the instructions in "Deploying a message flow that uses WebSphere Adapters" on page 3240.

What to do next

Using configurable services for WebSphere Adapters nodes

WebSphere Adapters nodes can get connection details from either the adapter component or a configurable service. By using configurable services, you can change the connection details for adapters without the need to redeploy the adapters. For more details about creating, changing, reporting, and deleting the configurable services for each EIS, see the appropriate topic:

- "Changing connection details for SAP adapters" on page 719
- "Changing connection details for Siebel adapters" on page 720
- "Changing connection details for PeopleSoft adapters" on page 722
- "Changing connection details for JD Edwards adapters" on page 724

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Enhancing existing adapters with newly discovered objects" on page 2063
In WebSphere Message Broker Version 7.0, you can take an adapter component that was created by using the Adapter Connection wizard, and update it with newly discovered objects from the Enterprise Information System (EIS). This facility is known as *iterative discovery*. You can either add the new objects without modifying existing objects, or replace existing objects.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Resolving problems when developing message flows with WebSphere Adapters nodes" on page 3428
Advice for dealing with common problems that can arise when you develop message flows that contain WebSphere Adapters nodes.

Related reference:

"SAP options for rediscovery" on page 4090
In WebSphere Message Broker Version 7.0, you can add newly discovered objects into an existing adapter component without modifying any existing objects. This facility is known as *iterative discovery*. You can rediscover certain objects for an inbound or outbound adapter.

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

Configuring EIS connections to expire after a specified time:

You can configure connections to SAP, Siebel, and PeopleSoft to expire after a specified time by using a configurable service.

About this task

You can use the connectionIdleTimeout property on a configurable service to control the number of connections to SAP, Siebel and PeopleSoft by closing connections that have not been used for a specified time.

Procedure

Use the **mqsicreateconfigurableservice** command to set up connections that expire when they have not been used for a specified length of time.
In the following example, the SAPConnection configurable service is configured to close connections when they have not been used for 120 seconds.

```
mqsicreateconfigurableservice MB7BROKER -c SAPConnection -o mySAPAdapter.outadapter
-n connectionIdleTimeout -v 120
```

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

Related tasks:

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

Related reference:

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

Configuring WebSphere Adapters nodes for secondary adapters:

You can deploy the resources that are required to support new methods in an Enterprise Information System (EIS), without affecting any resources that are already deployed, by using primary and secondary adapters and message sets. You must configure the WebSphere Adapters nodes in your message flows to use the secondary adapters.

Before you begin

Before you start:

- Read the concept topic, "WebSphere Adapters deployment" on page 3219.
- Complete the steps in "Preparing your system to use WebSphere Adapters nodes" on page 2034.

About this task

The primary adapter for a WebSphere Adapters node contains its connection information and part of its interface; the secondary adapters contain the rest of the interface. The primary message set contains message model metadata for the parts of the interfaces that are supported by the node; the secondary message sets define the rest of the model.

Complete the following steps to configure a message flow so that it can be extended by secondary adapters and message sets.

Procedure

1. To create a primary adapter and message set, run the Adapter Connection wizard by following the instructions in "Connecting to an EIS by using the Adapter Connection wizard" on page 2037.
2. When the wizard has finished, create a message flow that contains one or more WebSphere Adapters nodes.
3. In the Broker Development view, expand the folders beneath the message set until you see the adapter component, which has a suffix of `.inadapter` or `.outadapter`.
4. Drag the adapter component onto the WebSphere Adapters node in your message flow.
5. On the Basic properties tab of the WebSphere Adapters node, set the `Secondary adapter mode` to `All adapters in execution group`.
6. Save the message flow.

What to do next

Next:

Add the message flow, adapter, and message set to a BAR file, then deploy the BAR file to the execution group. For more information, see "Deploying a message flow that uses WebSphere Adapters" on page 3240.

You have created a message flow that you can enhance in the future when you need to call new services or handle new event types in an EIS. For instructions about how to enhance your message flow, see the following topics:

- "Calling new services from a WebSphere Adapters request node without changing existing deployed resources" on page 2042
- "Handling new event types from a Websphere Adapters input node without changing existing deployed resources" on page 2044

Related concepts:

"WebSphere Adapters deployment" on page 3219
When you have run the Adapter Connection wizard and created a message flow, you must deploy the resources that are generated by adding them to a broker archive (BAR) file.

Related tasks:

"Preparing your system to use WebSphere Adapters nodes" on page 2034
Before you can connect to an Enterprise Information System (EIS), you must prepare your system by adding external software dependencies and configuring the EIS to work with the WebSphere Adapter.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Calling new services from a WebSphere Adapters request node without changing existing deployed resources" on page 2042
If your message flow acts as a gateway to an Enterprise Information System (EIS), you can use it to call new services that did not exist when you developed the flow. Therefore, if a new service is provided by the EIS, you do not have to modify and retest the message flow.

"Handling new event types from a Websphere Adapters input node without changing existing deployed resources" on page 2044
You can create an event handler to an Enterprise Information System (EIS) to handle new event types that did not exist when you first developed your message flow. Therefore, if a new event is provided by the EIS, you do not have to modify and retest the message flow.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

"Connecting to Enterprise Information Systems" on page 1912
Use WebSphere Adapters to communicate with Enterprise Information Systems (EIS) such as SAP, Siebel, PeopleSoft, and JD Edwards.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

Calling new services from a WebSphere Adapters request node without changing existing deployed resources:

If your message flow acts as a gateway to an Enterprise Information System (EIS), you can use it to call new services that did not exist when you developed the flow. Therefore, if a new service is provided by the EIS, you do not have to modify and retest the message flow.

Before you begin

Before you start:

- Read the concept topic "WebSphere Adapters deployment" on page 3219.
- Ensure that you have created and deployed a message flow that is configured to use secondary adapters, as described in "Configuring WebSphere Adapters nodes for secondary adapters" on page 2040.

About this task

To use iterative deployment, you can configure an SAPRequest, SiebelRequest, PeopleSoftRequest, or JDEdwardsRequest node to look for a specified operation in all relevant `.outadapter` components that are deployed to the execution group.

Procedure

1. Run the Adapter Connection wizard to create an `.outadapter` component and a message set for the new EIS services.

   You do not need to rediscover existing services. For more information about running the Adapter Connection wizard, see "Connecting to an EIS by using the Adapter Connection wizard" on page 2037.
2. Ensure that method names are unique across all primary and secondary adapters that are used by the request node. If they are not unique, edit them by clicking **Edit Operations** on the Service Generation and Deployment Configuration panel of the Adapter Connection wizard.

   The method names correspond to the Service Operation names, which are configured by the Adapter Connection wizard. In most cases, the names are based on the name of the service that is being discovered (for example, the BAPI name in SAP, or the business object and operation name in Siebel). However, in some cases, you must edit them to avoid a clash.

You can set the method name dynamically in the message flow by setting the local environment field `$LocalEnvironment/Adapter/MethodName`.

3. Avoid duplicate method names by using user trace in the following way.
   a. Start user trace by following the instructions in "Starting user trace" on page 3197.
   b. Stop and restart the message flow.
   c. Read user trace by using the **mqsireadlog** command. Message BIP3432 identifies which methods are already defined by currently deployed adapters.

   Alternatively, you can identify the methods that are defined by the adapter by looking at the `Default method` property of the request node. If you have a `.outadapter` component in your workspace, you can drop it onto a message flow to create a request node for the `.outadapter` component. The request node has that adapter set as its primary adapter, and the list of methods that are defined by that adapter are visible in the `Default method` property of the request node.
4. Ensure that the message set that is created does not contain any types that share the same name and namespace of existing message sets. You can change the namespaces of the types on the Adapter Connection wizard by using the Business Object Namespace control.

   Use different namespaces for different message sets. The use of different namespaces is important when working with BAPIs because the BAPI return field typically has the same name for all BAPIs, and its type definition can change depending on the age of the BAPI.

Related concepts:

"WebSphere Adapters deployment" on page 3219
When you have run the Adapter Connection wizard and created a message flow, you must deploy the resources that are generated by adding them to a broker archive (BAR) file.

Related tasks:

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Configuring WebSphere Adapters nodes for secondary adapters" on page 2040
You can deploy the resources that are required to support new methods in an Enterprise Information System (EIS), without affecting any resources that are already deployed, by using primary and secondary adapters and message sets. You must configure the WebSphere Adapters nodes in your message flows to use the secondary adapters.

"Handling new event types from a Websphere Adapters input node without changing existing deployed resources" on page 2044
You can create an event handler to an Enterprise Information System (EIS) to handle new event types that did not exist when you first developed your message flow. Therefore, if a new event is provided by the EIS, you do not have to modify and retest the message flow.

"Starting user trace" on page 3197
Use user trace for debugging your applications; you can trace brokers, execution groups, and deployed message flows. Start user trace facilities using the **mqsichangetrace** command or the WebSphere Message Broker Explorer.

"Stopping a message flow using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 952
Use the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer

to stop a message flow.

"Starting a message flow by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 951
Use the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer to start a message flow.

"Connecting to Enterprise Information Systems" on page 1912
Use WebSphere Adapters to communicate with Enterprise Information Systems (EIS) such as SAP, Siebel, PeopleSoft, and JD Edwards.

Related reference:

"**mqsireadlog** command" on page 3905
Use the **mqsireadlog** command to retrieve trace records for the specified component.

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

Handling new event types from a Websphere Adapters input node without changing existing deployed resources:

You can create an event handler to an Enterprise Information System (EIS) to handle new event types that did not exist when you first developed your message flow. Therefore, if a new event is provided by the EIS, you do not have to modify and retest the message flow.

Before you begin

Before you start:

- Read the concept topic "WebSphere Adapters deployment" on page 3219.
- Ensure that you have created and deployed a message flow that is configured to use secondary adapters, as described in "Configuring WebSphere Adapters nodes for secondary adapters" on page 2040.

About this task

To implement an event handler that you can enhance iteratively with new event types (that is, deploy new event types to the event handler), configure an SAPInput node to use secondary adapters. The set of secondary adapters can be extended at operational time without affecting any existing adapters.

Procedure

1. Run the Adapter Connection wizard to create a new `.inadapter` component and a new message set for the new EIS events.

   You do not need to rediscover existing events. For more information about running the Adapter Connection wizard, see "Connecting to an EIS by using the Adapter Connection wizard" on page 2037.
2. Ensure that the method names are unique.

   The method names correspond to the Service Operation names. Therefore, when you are choosing or editing the Service Operation names, make sure that they do not clash with those names defined in other adapters that are being used by that node.
3. Avoid duplicate method names by using user trace in the following way.
   a. Start user trace by following the instructions in "Starting user trace" on page 3197.

b. Stop and restart the message flow.
c. Read user trace by using the **mqsireadlog** command. Message BIP3432 identifies which methods are already defined by currently deployed adapters.

4. Ensure that the message set that is created does not contain any types that share the same name and namespace of existing message sets. You can change the namespaces of the types on the Adapter Connection wizard by using the Business Object Namespace control.

   Use different namespaces for different message sets. The use of different namespaces is important when working with BAPIs because the BAPI return field typically has the same name for all BAPIs, and its type definition can change depending on the age of the BAPI.

Related concepts:

"WebSphere Adapters deployment" on page 3219
When you have run the Adapter Connection wizard and created a message flow, you must deploy the resources that are generated by adding them to a broker archive (BAR) file.

Related tasks:

"Preparing your system to use WebSphere Adapters nodes" on page 2034
Before you can connect to an Enterprise Information System (EIS), you must prepare your system by adding external software dependencies and configuring the EIS to work with the WebSphere Adapter.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Configuring WebSphere Adapters nodes for secondary adapters" on page 2040
You can deploy the resources that are required to support new methods in an Enterprise Information System (EIS), without affecting any resources that are already deployed, by using primary and secondary adapters and message sets. You must configure the WebSphere Adapters nodes in your message flows to use the secondary adapters.

"Calling new services from a WebSphere Adapters request node without changing existing deployed resources" on page 2042
If your message flow acts as a gateway to an Enterprise Information System (EIS), you can use it to call new services that did not exist when you developed the flow. Therefore, if a new service is provided by the EIS, you do not have to modify and retest the message flow.

"Resolving problems that occur during deployment of message flows" on page 3440
Use the advice given here to help you to resolve problems that can arise during deployment of message flows or message sets.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

"Connecting to Enterprise Information Systems" on page 1912
Use WebSphere Adapters to communicate with Enterprise Information Systems (EIS) such as SAP, Siebel, PeopleSoft, and JD Edwards.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

Interacting with an SAP application:

To interact with an SAP application, obtain external software dependencies, run the Adapter Connection wizard, develop a message flow, then deploy the relevant resources.

About this task

To connect to an SAP application, the SAP adapter requires certain files and libraries. You must store these files so that they are accessible to the Adapter Connection wizard. The wizard creates various resources, such as an adapter component and message flow. After you have completed the wizard, you can develop a message flow to define the interaction with the SAP application, then deploy the relevant resources.

Procedure

1. To obtain software dependencies, follow the instructions in "Adding external software dependencies for SAP" on page 2048.
2. Optional: If you are using Application Link Enabling (ALE) processing, register a Remote Function Call (RFC) destination on the SAP server and configure the SAP server, as described in "Configuring the SAP server to work with the adapter" on page 2050.
3. Optional: To set up SAP for high availability, see "Setting up SAP for high availability" on page 2057.
4. Before you run the Adapter Connection wizard, gather the following information from your SAP administrator:
   - SAP system user name
   - SAP system password
   - SAP host name or IP address
   - SAP Client ID (for example, 001)
   - SAP system number (for example, 00)
   - Language code (for example, EN)

   You might also need to gather some specific information from your SAP administrator about the items that you are discovering. For example, if you are discovering IDocs, you need the appropriate information to complete the following fields:
   - IDoc name
   - Version
   - SAP outbound
   - Verbs (create, update, delete)
   - Partner number
   - Message type

   For more information, see "SAP connection properties for the Adapter Connection wizard" on page 4044.
5. To connect to SAP by using the Adapter Connection wizard, and create a message flow, follow the instructions in "Connecting to an EIS by using the Adapter Connection wizard" on page 2037.
6. Develop your message flow to define the interaction with the SAP application.

   For example, to route IDocs to separate messages, follow the instructions in "Routing IDocs to separate message flows" on page 2055.

7. Optional: To propagate security credentials to an SAP request, see "Propagating security credentials to an SAP request" on page 2065.
8. Deploy the appropriate resources, as described in "Deploying a message flow that uses WebSphere Adapters" on page 3240.
9. Optional: To change connection details for SAP adapters, see "Changing connection details for SAP adapters" on page 719.
10. Optional: To enhance existing adapters with newly discovered objects, see "Enhancing existing adapters with newly discovered objects" on page 2063.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Activating IBM Tivoli License Manager for WebSphere Adapters" on page 2036
If you want to use IBM Tivoli License Manager (ITLM), you must activate it for each of the WebSphere Adapters.

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

Adding external software dependencies for SAP:

Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

Before you begin

Before you start:

Ensure that you have the relevant prerequisite files for your SAP system. (For information about the versions of files that WebSphere Message Broker supports, see WebSphere Message Broker Requirements.)

- `sapjco3.jar`
- Windows On Windows:
  - `sapjco3.dll`
- z/OS Linux UNIX On z/OS, Linux, and UNIX:
  - `libsapjco3.so`

Download these files for your operating system from the external SAP Web site, SAP Service Marketplace, and save them to a directory, such as `C:\SAP_LIB`. (On Windows, the directory cannot be a mapped network drive on a remote Windows computer; the directory must be local or on a Storage Area Network (SAN) disk.) You must have an SAPNet account to be able to access this Web site.

- Windows On Windows, download the `.dll` files that come with the SAP JCo download.
- z/OS Linux UNIX On z/OS, Linux, and UNIX, download the `.so` and `.o` files that come with the SAP JCo download.

About this task

Locating the SAP support files in the runtime environment

To add the SAP prerequisite files to the runtime environment, take the following steps.

Procedure

- Windows Linux UNIX On Windows, Linux and UNIX:
  1. Ensure that the broker has started.
  2. Either open the Command Console, or open a Windows command prompt and enter **`mqsiprofile`** to initialize the environment.
  3. Enter the following command to display the locations of the prerequisite JAR files and native libraries:

     ```
     mqsireportproperties MB7BROKER -c EISProviders -o AllReportableEntityNames  -r
     ```

     The following example shows what typically is displayed when you run this command:

     ```
     ReportableEntityName=''
      EISProviders
       JDEdwards=''
        jarsURL='default_Path'
        nativeLibs='default_Path'
     ```

```
PeopleSoft=''
 jarsURL='default_Path'
 nativeLibs='default_Path'
SAP=''
 jarsURL='default_Path'
 nativeLibs='default_Path'
Siebel=''
 jarsURL='default_Path'
 nativeLibs='default_Path'
 siebelPropertiesURL=''
Twineball=''
 jarsURL='default_Path'
 nativeLibs='default_Path'
```

4. Set the location of the SAP prerequisite files using the following command:

```
mqsichangeproperties MB7BROKER -c EISProviders -o SAP  -n jarsURL -v C:\SAP_LIB
mqsichangeproperties MB7BROKER -c EISProviders -o SAP  -n nativeLibs -v C:\SAP_LIB
```

5. To check that the values have been set correctly, run the following command:

   ```
   mqsireportproperties  MB7BROKER -c EISProviders -o SAP -r
   ```

   The following example shows what is displayed by the **mqsireportproperties** command.

   ```
   ReportableEntityName=' '
   EISProviders
    SAP=' '
     jarsURL='C:\SAP_LIB'
     nativeLibs='C:\SAP_LIB'

   BIP8071I: Successful command completion.
   ```

6. Restart the broker.

- z/OS On z/OS: Run the **mqsireportproperties** command by customizing and submitting the BIPCHPR utility. For more information about this utility, see "Contents of the broker PDSE" on page 3991.

About this task

Locating the SAP support files through the WebSphere Message Broker Toolkit

To add the SAP prerequisite files through the WebSphere Message Broker Toolkit, take the following steps.

Procedure

- Windows On Windows: When you run the Adapter Connection wizard, you are prompted to specify the paths to the required libraries.
- Linux On Linux: Append the directory that contains the SAP libraries to the LD_LIBRARY_PATH environment variable:

  ```
  LD_LIBRARY_PATH=DIRECTORY_CONTAINING_SAP_LIBRARIES:${LD_LIBRARY_PATH}
  export LD_LIBRARY_PATH
  ```

  You must set this variable either for the whole system, or in the same shell or environment from which the WebSphere Message Broker Toolkit is launched.

What to do next

Next: configure the SAP system to work with the adapter

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Activating IBM Tivoli License Manager for WebSphere Adapters" on page 2036
If you want to use IBM Tivoli License Manager (ITLM), you must activate it for each of the WebSphere Adapters.

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Configuring the SAP server to work with the adapter"
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

Configuring the SAP server to work with the adapter:

Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

Before you begin

Before you start:

Add the required external software dependencies for SAP.

About this task

Complete the following steps on the SAP server by using the SAP graphical user interface.

Procedure

1. Register an RFC program ID:
   a. Open transaction **SM59** (Display and Maintain RFC Destinations).
   b. Click **Create**.
   c. Type a name for the RFC destination.
   d. In the **Connection Type** field, select **T**.
   e. In the **Activation Type** field, select **Registered Server Program**.
   f. Type a Program ID.

      You use this program ID when you configure the adapter. This value indicates to the SAP gateway which RFC-enabled functions the program ID listens for.
   g. Enter a description in **Description 1**, such as `RFC for Test Sample`.
   h. Enter a description in **Description 2**, such as your name.
   i. Click **MDMP & Unicode**, and set the RFC destination to **Unicode** or **non-Unicode**, depending on the communication type of the target system.

      To avoid errors when you use multiple language settings, set the RFC destination to **Unicode**.
   j. Save your entry.
2. Set up a receiver port:
   a. Open transaction **WE21** (Ports in IDoc processing).
   b. Select **Transactional RFC**, click **Ports**, and click the **Create** icon.
   c. Type a name for the port and click **OK**.
   d. Type the name of the destination that you created in the previous task (or select it from the list).
   e. Save your entry.
3. Specify a logical system:
   a. Open transaction **BD54** (Change View Logical Systems).
   b. Click **New Entries**.
   c. Type a name for the logical system and click the **Save** icon.
   d. If you see the Prompts for Workbench request, click the **New Request** icon. Then enter a short description and click **Save**.
   e. Click the **Continue** icon.
4. Configure a distribution model:
   a. Open transaction **BD64** (Maintenance of Distribution Model).
   b. Click **Distribution Model** > **Switch processing model**.
   c. Click **Create model view**.
   d. Type a name for the model view and click the **Continue** icon.

e. Select the distribution model that you created and click **Add message type**.
f. For outbound processing, type the logical system name that you created in the previous task as **Sender**, and type the logical name of the SAP server as **Receiver**, then select a message type (for example, **MATMAS**) and click the **Continue** icon.
g. Select the distribution model again and click **Add message type**.
h. For inbound processing, type the logical name of the SAP server as **Sender**, and the logical system name that you created in the previous task as **Receiver**, then select a message type (for example, **MATMAS**) and click the **Continue** icon.
i. Save your entry.

5. Set up a partner profile:
   a. Open transaction **WE20** (Partner Profiles).
   b. Click the **Create** icon.
   c. Type the name of the logical system that you created in the earlier task and, for **Partner Type**, select **LS**.
   d. For **Post Processing: permitted agent**, type `US` and your user ID.
   e. Click the **Save** icon.
   f. In the Outbound parameters section, click the **Create outbound parameter** icon.
   g. In the Outbound parameters window, type a message type (for example, `MATMAS05`), select the receiver port that you created in the earlier task, and select **Transfer IDoc immed**.
   h. Click the **Save** icon.
   i. Press **F3** to return to the Partner Profiles view.
   j. In the Inbound parameters section, click the **Create inbound parameter** icon.
   k. In the Inbound parameters window, type a message type (for example, `MATMAS`), and a process code (for example, `MATM`).
   l. Click the **Save** icon.
   m. Press **F3** to return to the Partner Profiles view.
   n. In the Inbound parameters section, click the **Create inbound parameter** icon.
   o. In the Inbound parameters window, type the following values: `ALEAUD` for **Message Type**, and `AUD1` for **Process Code**.
   p. Click the **Save** icon.
   q. Press **F3** to return to the Partner Profiles view.
   r. Click the **Save** icon.

What to do next

Next: connect to an EIS using the Adapter Connection wizard.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special

coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

Changing connection details for SAP adapters:

SAP nodes can get SAP connection details from either the adapter component or a configurable service. By using configurable services, you can change the connection details for adapters without the need to redeploy the adapters. To pick up new values when a configurable service is created or modified, you must reload the broker or execution group to which the adapter was deployed, by using the **mqsistop** and **mqsistart** commands, or the **mqsireload** command.

Before you begin

Before you start:

- Read "WebSphere Adapters nodes" on page 1914 and "Configurable services" on page 1296 for background information.

About this task

Use the SAPConnection configurable service to change connection details for an SAP adapter. The SAP node reads all connection properties from the adapter component that it is configured to use. If a configurable service exists that has the same name as the node's adapter component, the node uses the values that are

defined in that configurable service to override the corresponding properties from the adapter. If any properties on the configurable service are set to an empty string, the values that are configured in the `.inadapter` or `.outadapter` file are used. The properties of the SAP configurable services are described in "Configurable services properties" on page 3766.

Creating, changing, reporting, and deleting configurable services

Procedure

- To create a configurable service, use the WebSphere Message Broker Explorer; see "Using the WebSphere Message Broker Explorer to work with configurable services" on page 644 for more information, or the **mqsicreateconfigurableservice** command, as shown in the following example. This example creates an SAPConnection configurable service for the SAP adapter *mySAPAdapter.outadapter* that connects to the SAP host `test.sap.ibm.com`, and uses client *001* for the connections into that server:

  ```
  mqsicreateconfigurableservice MB7BROKER -c SAPConnection
  -o mySAPAdapter.outadapter -n applicationServerHost,client
  -v test.sap.ibm.com,001
  ```

  To pick up the new values in the configurable service, restart the execution group and message flow.
- To change a configurable service, use the WebSphere Message Broker Explorer, or the **mqsichangeproperties** command, as shown in the following example. This example changes the connections that are used by the adapter *mySAPAdapter.outadapter*. After you run this command, all adapters connect to the production system (`production.sap.ibm.com`) instead of the test system (`test.sap.ibm.com`):

  ```
  mqsichangeproperties MB7BROKER -c SAPConnection -o mySAPAdapter.outadapter
  -n applicationServerHost -v production.sap.ibm.com
  ```

  To pick up the updated values in the configurable service, restart the execution group and message flow.
- To display all SAPConnection configurable services, use the WebSphere Message Broker Explorer, or the **mqsireportproperties** command, as shown in the following example:

  ```
  mqsireportproperties MB7BROKER -c SAPConnection -o AllReportableEntityNames -r
  ```
- You can delete a configurable service that you have created by using the WebSphere Message Broker Explorer, or the **mqsideleteconfigurableservice** command, as shown in the following example:

  ```
  mqsideleteconfigurableservice MB7BROKER -c SAPConnection
  -o mySAPAdapter.outadapter
  ```

  You can delete only custom-named configurable services; you cannot delete IBM-defined configurable services.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Configurable services" on page 1296
Configurable services are typically runtime properties. You can use them to define properties that are related to external services on which the broker relies; for example, an SMTP server or a JMS provider.

"Using the WebSphere Message Broker Explorer to work with configurable services" on page 644
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view, add, modify and delete configurable services.

Related reference:

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

Routing IDocs to separate message flows:

You can use a pattern to process IDocs of various kinds with a single RFC program ID without having to redeploy or rediscover existing message sets and adapters, even when adding new types of IDoc.

Before you begin

Before you start:

Read the concept topic about "Generic IDoc routing" on page 1976.

About this task

You can create a routing message flow that contains an SAPInput node. You can use this flow to route the IDocs, based on their type, to separate message flows that deal with each different IDoc type. If the set of discovered IDocs is extended, you can create a message flow and message set, then deploy them, without the need to change existing message flows or message sets.

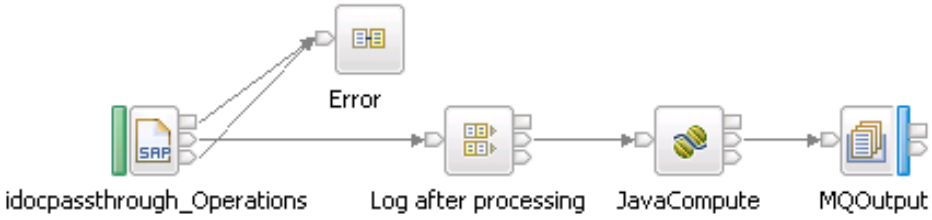
For more information about the pattern that is used in this task, see Data distribution SAP to WebSphere MQ: one-way (for IDoc). You can view patterns in the information center by using the links only when you use the information center that is integrated with the WebSphere Message Broker Toolkit, or when you use the online information center.

The following instructions describe how to use a pattern to create the resources that you need.

Procedure

1. Open the Patterns Explorer by following the instructions in "Choosing a pattern" on page 1313.
2. Expand the Application Integration category, and the SAP category, then click **MQ one-way (IDoc)** to open the pattern. The pattern specification describes the pattern and how to use it.
3. Click **Create New Instance**.
4. Enter a name and provide the required configuration parameters for the pattern instance, then click **Generate**.

   When the pattern has been generated, the Broker Development view lists the resources that have been generated:
   - A Java project
   - A routing message flow:



   - A message set
   - An `.inadapter` component
5. Add these resources to a BAR file then deploy the BAR file.
6. Use the **mqsisetdbparms** command to set your user name and password for the SAP system. For more information about how to use this command, see "**mqsisetdbparms** command" on page 3954.
7. The pattern also creates a configurable service called `idocpassthrough.inadapter.configurableservice`. Deploy the configurable service by dragging it onto the broker in WebSphere Message Broker Explorer.

Results

Result: The SAP adapter is connected by using the program ID that is specified in the configurable service. The message flow receives an IDoc from SAP and writes it to WebSphere MQ. The ESQL code in the Compute node specifies that the queue name to which the message is written is the same as the IDoc type. You can create separate message flows to process the different types of IDoc.

What to do next

Next: The SAPInput node can route the IDocs according to their type, but it does not parse the binary data BLOB messages that they contain. You can create a message flow that contains an MQInput node, which uses an adapter message set in the DataObject domain to parse the binary data BLOB message into a structured record that WebSphere Message Broker can manipulate. To use the DataObject parser to parse IDocs, you must set the `Message domain` property of the MQInput node to `DataObject`, and set the `Message format` property to `SAP ALE IDoc`. To create the message set, run the Adapter Connection wizard.

Related concepts:

"Generic IDoc routing" on page 1976
By using the SAPInput node in passthrough mode, WebSphere Message Broker can receive any IDoc and route it according to IDoc type.

"The ALE interfaces" on page 1959
The SAP Application Link Enabling (ALE) interface and ALE pass-through IDoc interface enable business process integration and asynchronous data communication between two or more SAP systems or between SAP and external systems. The data is exchanged in the form of Intermediate Documents (IDocs).

"Inbound processing for the ALE interface" on page 1963
The adapter supports inbound processing (from the SAP server to the adapter) for the ALE interface and the ALE pass-through IDoc interface.

"DataObject parser and domain" on page 1114
Use the DataObject domain to parse and write messages for WebSphere Adapters and CORBA applications.

"Patterns" on page 1310
A *pattern* is a reusable solution that encapsulates a tested approach to solving a common architecture, design, or deployment task in a particular context.

Related tasks:

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Deploying a broker archive file" on page 3235
After you have created and populated a broker archive (BAR) file, deploy the file to an execution group on a broker, so that the file contents can be used in the broker.

Related reference:

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

Setting up SAP for high availability:

You can configure WebSphere Message Broker to withstand software or hardware failures when working with SAP by moving the transaction ID (TID) store to a remote queue manager that can be shared between two brokers. To avoid a single point of failure, make this queue manager a WebSphere MQ high availability, multi-instance queue manager.

Before you begin

Before you start:

Read the concept topic about "SAP high availability" on page 1947.

About this task

The following topics describe how to set up a WebSphere MQ shared queue on distributed systems or z/OS.

Procedure

- "Setting up a shared queue on distributed systems for the SAP adapter event store"
- "Setting up a shared queue on z/OS for the SAP adapter event store" on page 2061

Related concepts:

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Defining WebSphere MQ resources" on page 1558
An application client can run on a computer anywhere in the WebSphere MQ network. If your applications use WebSphere MQ facilities to connect to the broker, and to interact with it (by using the MQI and AMI), you must set up the WebSphere MQ resources that they require.

"Using WebSphere MQ cluster queues for input and output" on page 1544
Design your broker network to use WebSphere MQ queues, if appropriate for your business needs.

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

Related reference:

"WebSphere MQ resources for the broker" on page 585
Each broker depends on a number of WebSphere MQ resources: some are required, others are optional, and depend on your environment and requirements. Some of these resources are created for you, but others you must define for yourself.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

Setting up a shared queue on distributed systems for the SAP adapter event store:

To achieve high availability when processing SAP messages on distributed systems, you can set up a shared queue for the SAP adapter event store.

About this task

On distributed systems, you can configure the broker to use a remote queue manager to persist the transaction ID (TID) store for SAP transactional RFC (tRFC) data. By using this configuration, two adapters that are deployed to two brokers can share the same TID, and can therefore operate as a single RFC server. This configuration is essential if the adapters have been configured with the same RFC Program ID.

Before the broker can use a remote queue manager as the TID store, you must complete some administration tasks on that queue manager. You must first create the queue for the broker to use as the TID store. Then you must define two

channels for the broker to use to connect to that queue manager: define the server channel on the queue manager, and define a client channel in a file, which you must make available to the broker.

You can use the Run WebSphere MQ Commands (runmqsc) command or WebSphere MQ Explorer to create this client channel definition file on any queue manager. After you have created the file, you must move it from the queue manager to a file system that is accessible to the broker.

To set up a shared queue for the SAP adapter event store, complete the following steps.

Procedure

1. Create the queue SYSTEM.BROKER.ADAPTER.PROCESSED.

   You can create the queue either by using WebSphere MQ Explorer or by running the following command in WebSphere MQ **runmqsc**:

   ```
   DEFINE QLOCAL('SYSTEM.BROKER.ADAPTER.PROCESSED')
   ```

2. Create the server channel.
   a. On the computer that hosts the shared queue manager, create a server channel by running the WebSphere MQ **runmqsc** command and entering the following parameters, where *WSADAPTERS.SAP* is an example of the channel name.

      ```
      DEFINE CHANNEL ('WSADAPTERS.SAP') CHLTYPE (SVRCONN) TRPTYPE(TCP)
      ```

   b. After you have created the channel, start it by using the following command in **runmqsc**:

      ```
      START CHANNEL ('WSADAPTERS.SAP')
      ```

3. Create the client definition file.

   On any queue manager, create a client definition file by running the **runmqsc** command and entering the following parameters, where the following criteria apply:

   - *WSADAPTERS.SAP* is an example of the channel name.
   - *myhost.ibm.com* is an example of the host name of the computer where the queue manager is running. This queue manager is the one on which you created the queue in step 1 and the server channel in step 2. This name is used by the broker to connect to the queue manager, therefore do not use *localhost*.
   - *1414* is an example of the port number where the listener is running on that queue manager.
   - *QMGR* is an example of the queue manager name.
   - You can use any name for the server and client channels, but they must match.
   - Set the CONNAME parameter to the host name or IP address of the computer that hosts the shared queue manager, and the port number of the MQ Listener that is running on that computer. 1414 is the default listener port.

   ```
   DEFINE CHANNEL ('WSADAPTERS.SAP') CHLTYPE (CLNTCONN) CONNAME('myhost.ibm.com(1414)') TRPTYPE(T
   ```

   If the queue manager is running in multi-instance mode, define two channels, one for each instance of the queue manager

4. Move the client definition file to a file system that is accessible by the broker.

- Linux UNIX On Linux and UNIX, you can find this file at `/var/mqm/qmgrs/QMGR_NAME/@ipcc/AMQCLCHL.TAB`.
- Windows On Windows, you can find this file at `C:\Program Files\IBM\WebSphere MQ\Qmgrs\QMGR_NAME\@ipcc\AMQCLCHL.TAB`.

5. Configure the broker.
   a. Create an SAPConnection configurable service for the `.inadapter` component by using the **mqsicreateconfigurableservice** command.
   b. Set the sharedTidStoreClientDefinitionFile parameter of the configurable service to the path of the file that you created in step 3 and moved in step 4.
   c. Set the sharedTidStoreQueueManger parameter to the name of the queue manager for which you created the queue in step 1 and the server channel in step 2.

      For more information about these parameters, see "SAPConnection configurable service" on page 3784.
6. Verify the setup.

   After you have completed these steps, you can verify that the broker is using the queue on the remote queue manager by inspecting user trace. If the setup is successful, message BIP3470 is issued, specifying which queue manager the broker is using as the TID store.

Related concepts:

"SAP high availability" on page 1947
You can configure WebSphere Message Broker to withstand software or hardware failures when working with SAP, so that WebSphere Message Broker is available for as much of the time as possible.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Setting up a shared queue on z/OS for the SAP adapter event store" on page 2061
To achieve high availability when processing SAP messages on z/OS, you can set up a shared queue for the SAP adapter event store.

"Defining WebSphere MQ resources" on page 1558
An application client can run on a computer anywhere in the WebSphere MQ network. If your applications use WebSphere MQ facilities to connect to the broker, and to interact with it (by using the MQI and AMI), you must set up the WebSphere MQ resources that they require.

"Using WebSphere MQ cluster queues for input and output" on page 1544
Design your broker network to use WebSphere MQ queues, if appropriate for your business needs.

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

Related reference:

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"WebSphere MQ resources for the broker" on page 585
Each broker depends on a number of WebSphere MQ resources: some are required, others are optional, and depend on your environment and requirements. Some of these resources are created for you, but others you must define for yourself.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

Setting up a shared queue on z/OS for the SAP adapter event store:

To achieve high availability when processing SAP messages on z/OS, you can set up a shared queue for the SAP adapter event store.

Before you begin

Before you start:

Ensure that the queue-sharing group name that you are planning to use does not exist in SYS1.PARMLIB(IEFSSNxX). If the name does exist, choose a different name for the new group.

About this task

You can find the data set members, which are referred to in the following steps, as samples in the `thlqual.SCSQPROC` library, unless stated otherwise. The library is supplied with WebSphere MQ for z/OS.

Procedure

1. Review your DB2 data-sharing requirements. The queue-sharing group is backed by a DB2 data sharing group. Add entries to DB2 for WebSphere MQ by completing the following steps.
   a. Customize the CSQ4INSS data set member with the structure names and ensure that it is included in the CSQINP2 DD concatenation for one queue manager in the queue-sharing group.
   b. To amend the QSGDATA system parameter, amend and rerun the CSQ4ZPRM sample in the format (*Qsgname*,*Dsgname*,*Db2name*,*Db2servers*).
   c. Copy, customize, and run the following sample JCLs. These jobs need to be run only once for each DB2 subsystem or data-sharing group.
      1) Customize and run sample JCL CSQ45CSG to create the storage group that is to be used for the WebSphere MQ database, table spaces, and tables.
      2) Customize and run sample JCL CSQ45CDB to create the database to be used by all queue managers that connect to this DB2 data-sharing group.
      3) Customize and run sample JCL CSQ45CTS to create the table spaces to contain the queue manager and channel initiator tables that are used for queue-sharing groups (created in step i).
      4) Customize and run sample JCL CSQ45CTB to create the twelve DB2 tables and associated indexes. Do not change any of the row names or attributes.
      5) Customize and run sample JCL CSQ45BPL to bind the DB2 plans for the queue manager, utilities, and channel initiator.
      6) Customize and run sample JCL CSQ45GEX to grant execute authority to the respective plans for the user IDs that will be used by the queue

manager, utilities, and channel initiator. The user IDs for the queue manager and channel initiator are the user IDs under which their started task procedures run. The user IDs for the utilities are the user IDs under which the batch jobs can be submitted. For more information about the names of the appropriate plans, see the WebSphere MQ Version 7 Information Center online.

2. Create a queue-sharing group by completing the following steps.
   a. Define the Coupling Facility structures that are used by the queue managers in the queue-sharing group.

      Define the administration and application structures in the Coupling Facility Resource Management (CFRM) policy data set for the z/OS Sysplex that will host the queue sharing group, as shown in the following example, where *MQPU* is the queue-sharing name.

      ```
      STRUCTURE NAME(MQPUCSQ_ADMIN)
      SIZE(20000)
      INITSIZE(10000)
      PREFLIST(PUCF01)

      STRUCTURE NAME(MQPUAPPLICATION1)
      SIZE(20000)
      INITSIZE(10000)
      PREFLIST(PUCF01)
      ```

      For more information about setting up the Coupling Facility, see the WebSphere MQ Version 7 Information Center online.
   b. Define each of the queue-sharing groups to DB2 by using the ADD QSG function of the queue-sharing group utility (CSQ5PQSG). Sample CSQ45AQS is provided in the SCSQPROC library. Run the sample once for each queue-sharing group.
   c. Define each of the queue managers to DB2 by using the ADD QMGR function of the queue-sharing group utility CSQ5PQSG. Sample CSQ45AQM is provided in the SCSQPROC library. Run the sample for every member of a queue-sharing group.
   d. To activate the changes that you have made, stop and start all the queue managers in the group.
3. Define the SAP event store queue in the queue sharing group.

   Typically, creating an event store queue is the most effective method. If you are migrating from a previous broker, ensure that any outstanding SAP requests have been resolved before you move to the new topology.
   a. To create a shared queue, run the following command on one of the queue managers only:

      ```
      DEFINE QL(SYSTEM.BROKER.ADAPTER.SHARED)
      LIKE(SYSTEM.BROKER.ADAPTER.PROCESSED) CFSTRUCT(NEW) QSGDISP(SHARED) SHARE DEFSOPT(SHARED)
      ```
   b. To allow other queue managers that have been added to the group to use the shared queue, delete their locally defined queues by running the following command on each of those queues:

      ```
      DELETE QLOCAL(SYSTEM.BROKER.ADAPTER.PROCESSED) QSGDISP(QMGR)
      ```
4. Configure the broker.
   a. Create an SAPConnection configurable service for the `.inadapter` component by using the **mqsicreateconfigurableservice** command.
   b. Set the sharedTidStoreQueueManger parameter to the name of the queue manager of the broker.

For more information about this parameter, see "SAPConnection configurable service" on page 3784.

5. Verify the setup.

   After you have completed these steps, you can verify that the broker is using the queue as a shared TID store by inspecting user trace. If the setup is successful, message BIP3470 is issued, specifying which queue manager the broker is using as the TID store.

Results

WebSphere Message Broker first attempts to open the SYSTEM.BROKER.ADAPTER.SHARED queue. If that queue is unavailable, WebSphere Message Broker opens the SYSTEM.BROKER.ADAPTER.PROCESSED queue instead. The availability of the SYSTEM.BROKER.ADAPTER.SHARED queue indicates that you are operating in a z/OS high availability environment.

Related concepts:

"SAP high availability" on page 1947
You can configure WebSphere Message Broker to withstand software or hardware failures when working with SAP, so that WebSphere Message Broker is available for as much of the time as possible.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Setting up a shared queue on distributed systems for the SAP adapter event store" on page 2058
To achieve high availability when processing SAP messages on distributed systems, you can set up a shared queue for the SAP adapter event store.

"Defining WebSphere MQ resources" on page 1558
An application client can run on a computer anywhere in the WebSphere MQ network. If your applications use WebSphere MQ facilities to connect to the broker, and to interact with it (by using the MQI and AMI), you must set up the WebSphere MQ resources that they require.

"Using WebSphere MQ cluster queues for input and output" on page 1544
Design your broker network to use WebSphere MQ queues, if appropriate for your business needs.

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

Related reference:

"WebSphere MQ resources for the broker" on page 585
Each broker depends on a number of WebSphere MQ resources: some are required, others are optional, and depend on your environment and requirements. Some of these resources are created for you, but others you must define for yourself.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

Enhancing existing adapters with newly discovered objects:

In WebSphere Message Broker Version 7.0, you can take an adapter component that was created by using the Adapter Connection wizard, and update it with newly discovered objects from the Enterprise Information System (EIS). This facility is

known as *iterative discovery*. You can either add the new objects without modifying existing objects, or replace existing objects.

Before you begin

Before you start:

The following instructions assume that you have already run the Adapter Connection wizard to discover a set of objects, as described in "Connecting to an EIS by using the Adapter Connection wizard" on page 2037. You can run iterative discovery only on projects that were created in WebSphere Message Broker Version 7.0 or later versions.

About this task

Procedure

1. From the Broker Development view, expand the appropriate message set project folders until you see the .inadapter or .outadapter component.
2. Right-click the adapter component, then click **Iterative discovery**.

   The Adapter Connection - Iterative Discovery wizard opens. The fields in the wizard are populated with the values that you specified when you ran it previously. You can modify these values if anything has changed, such as a password.
3. On the Find and Discover Services page, the objects that you previously discovered and selected for import are shown in the Objects to be imported pane. You can discover and select new objects to import from the Objects discovered by query pane.
4. On the final page of the Adapter Connection - Iterative Discovery wizard, choose from the following options.
   - To add only the new objects that you discovered, select **Add XSDs for new objects and replace .wsdl, .import/.export**. When you click **Finish**, the following results occur:
     - XSD files for the additional objects are added to the adapter component. These files are shown in the New files pane.
     - Updated .wsdl and .import or .export files replace the files already in the adapter component.
     - The XSD files that are being added to the adapter component are also imported into the message set that is associated with the adapter component.

     If you have discovered objects by using the SAP Wrapper or SAP Work Unit options, the XSD files that correspond to the SAP wrapper objects are replaced in the adapter component. These XSD files are also imported into the message set, replacing any files that have the same name. This behavior is because the wrapper objects contain references to the new objects.
   - To replace all existing objects with the objects that you have discovered, select **Replace the contents of .in/.outAdapter with the newly discovered files**. When you click **Finish**, the following results occur:
     - The contents of the .inadapter or .outadapter file in the adapter component are replaced.
     - All discovered XSD files are imported into the message set, replacing any files that have the same name.

   Note:

Select the **Replace the contents of .in/.outAdapter with the newly discovered files** option if you have changed any of the configuration options during rediscovery, or if the interfaces or data structure of objects that you are discovering have changed since the previous discovery.

If you have discovered objects by using the BAPI result set option, the **Add XSDs for new objects and replace .wsdl, .import/.export** option is not available and you see the message: `Incremental discovery is not applicable for the selected option`.

For options that are used in rediscovering SAP objects, and the corresponding restrictions, see "SAP options for rediscovery" on page 4090.

5. Click **Finish**.

Related concepts:

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

"WebSphere Adapters deployment" on page 3219
When you have run the Adapter Connection wizard and created a message flow, you must deploy the resources that are generated by adding them to a broker archive (BAR) file.

Related tasks:

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Calling new services from a WebSphere Adapters request node without changing existing deployed resources" on page 2042
If your message flow acts as a gateway to an Enterprise Information System (EIS), you can use it to call new services that did not exist when you developed the flow. Therefore, if a new service is provided by the EIS, you do not have to modify and retest the message flow.

"Handling new event types from a Websphere Adapters input node without changing existing deployed resources" on page 2044
You can create an event handler to an Enterprise Information System (EIS) to handle new event types that did not exist when you first developed your message flow. Therefore, if a new event is provided by the EIS, you do not have to modify and retest the message flow.

Related reference:

"SAP options for rediscovery" on page 4090
In WebSphere Message Broker Version 7.0, you can add newly discovered objects into an existing adapter component without modifying any existing objects. This facility is known as *iterative discovery*. You can rediscover certain objects for an inbound or outbound adapter.

"Configuration properties for the WebSphere Adapter for SAP Software" on page 4043
The WebSphere Adapter for SAP Software has several categories of configuration properties, which you set with the Adapter Connection wizard when you generate or create objects and services.

Propagating security credentials to an SAP request:

The SAPRequest node can use an identity that is present in the Properties folder of the message tree structure for the security credentials in a request, by using the `Propagate` property on the security profile that is defined on the node.

About this task

If an SAPRequest node is configured with a security profile, it extracts security tokens from the input message at run time, and propagates an identity to SAP.

Procedure

To propagate an identity to be used for the SAP request security credentials, complete the following steps.

1. Ensure that an appropriate security profile exists for the SAPRequest node, or create a security profile, by following the instructions in "Creating a security profile" on page 433.
2. Use the Broker Archive editor to select a security profile for the SAPRequest node that has identity propagation enabled. For detailed instructions, see "Configuring for identity propagation" on page 492.

Related concepts:

"Identity and security token propagation" on page 426
Identity and security token propagation enables the identity and security tokens (associated with each message) to be propagated throughout a message flow, and on to target applications through output or request nodes.

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

Related tasks:

"Configuring for identity propagation" on page 492
To enable a message flow to perform identity propagation, the input nodes must extract the identity and the output node must propagate it.

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

Related reference:

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

Interacting with a Siebel application:

To interact with a Siebel application, obtain external software dependencies, run the Adapter Connection wizard, develop a message flow, then deploy the relevant resources.

About this task

To connect to a Siebel application, the Siebel adapter requires certain files and libraries. You must store these files so that they are accessible to the Adapter Connection wizard. The wizard creates various resources, such as an adapter component and message flow. After you have completed the wizard, you can develop a message flow to define the interaction with the Siebel application, then deploy the relevant resources.

Procedure

1. To obtain software dependencies, follow the instructions in "Adding external software dependencies for Siebel" on page 2068.
2. To configure the Siebel application to work with the adapter, create an event table and a Siebel business object, as described in "Configuring the Siebel application to work with the adapter" on page 2070.
3. Optional: To connect to a Siebel server in a clustered environment, see "Connecting to a clustered Siebel environment" on page 2078.
4. Optional: To connect to different versions of Siebel, see "Connecting to different versions of Siebel" on page 2077.
5. Before you run the Adapter Connection wizard, gather the following information from your Siebel administrator:
   - Siebel user name
   - Siebel password
   - Siebel host name or IP address
   - Language code

   For more information, see "Siebel connection properties for the Adapter Connection wizard" on page 4099.
6. To connect to Siebel by using the Adapter Connection wizard, and create a message flow, follow the instructions in "Connecting to an EIS by using the Adapter Connection wizard" on page 2037.
7. Develop your message flow to define the interaction with the Siebel application.
8. Deploy the appropriate resources, as described in "Deploying a message flow that uses WebSphere Adapters" on page 3240.
9. Optional: To change connection details for Siebel adapters, see "Changing connection details for Siebel adapters" on page 720.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for Siebel Business Applications" on page 2002
With WebSphere Adapter for Siebel Business Applications, you can create integrated processes that exchange information with a Siebel application, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Activating IBM Tivoli License Manager for WebSphere Adapters" on page 2036
If you want to use IBM Tivoli License Manager (ITLM), you must activate it for each of the WebSphere Adapters.

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for Siebel properties" on page 4092
Reference information to refer to when you connect to a Siebel application.

"SiebelInput node" on page 4740
Use the SiebelInput node to interact with a Siebel application.

"SiebelRequest node" on page 4745
Use the SiebelRequest node to interact with a Siebel application.

Adding external software dependencies for Siebel:

Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

Before you begin

Before you start:

Ensure that you have the relevant prerequisite files for your Siebel system.

- Siebel Business Applications Versions 7.5 and earlier
  - `SiebelJI_`*`language code`*`.jar` (for example, `SiebelJI_enu.jar`)
  - `SiebelJI_Common.jar`
- Siebel Business Applications Versions 7.7x, 7.8x, and 8.0
  - `Siebel.jar`
  - `SiebelJI_`*`language code`*`.jar` (for example, `SiebelJI_enu.jar`)

Download these files from the Siebel application, and save them to a directory, such as `C:\Siebel_LIB.` (On Windows, the directory cannot be a mapped network drive on a remote Windows computer; the directory must be local or on a Storage Area Network (SAN) disk.)

The sample resources that you need to set up the Siebel system so that it can communicate with the broker are in the following directory: *`install_dir`*`\ResourceAdapters\Siebel_7.0.0\samples`.

About this task

Locating the Siebel support files in the runtime environment on Windows

To add the Siebel prerequisite files to the runtime environment, take the following steps.

Procedure

1. Ensure that the broker has started.
2. Either open the Command Console, or open a Windows command prompt and enter **`mqsiprofile`** to initialize the environment.

3. Enter the following command to display the locations of the prerequisite JAR files and native libraries:

   ```
   mqsireportproperties MB7BROKER -c AllTypes -o AllReportableEntityNames  -r
   ```

   The following example shows what typically is displayed when you run this command:

   ```
   ReportableEntityName=''
    EISProviders
     JDEdwards=''
      jarsURL='default_Path'
      nativeLibs='default_Path'
     PeopleSoft=''
      jarsURL='default_Path'
      nativeLibs='default_Path'
     SAP=''
      jarsURL='default_Path'
      nativeLibs='default_Path'
     Siebel=''
      jarsURL='default_Path'
      nativeLibs='default_Path'
      siebelPropertiesURL=''
     Twineball=''
      jarsURL='default_Path'
      nativeLibs='default_Path'
   ```

4. Set the location of the Siebel prerequisite files by using the **mqsichangeproperties** command; for example:

```
mqsichangeproperties MB7BROKER -c EISProviders -o Siebel -n jarsURL -v C:\Siebel_LIB
mqsichangeproperties MB7BROKER -c EISProviders -o Siebel -n nativeLibs -v C:\Siebel_LIB
```

5. Optional: To connect to a server in a clustered environment, complete the following steps.

   a. If you do not already have a `siebel.properties` file, create one with a `siebel.conmgr.virtualhosts` property that lists the Siebel servers in the cluster. The `siebel.conmgr.virtualhosts` property is the only property in the `siebel.properties` file that WebSphere Message Broker supports. The server names are represented by a comma separated list in the format *hostname:port*.

   b. Set the location of the `siebel.properties` file by using the **mqsichangeproperties** command; for example:

```
mqsichangeproperties MB7BROKER -c EISProviders -o Siebel -n siebelPropertiesURL
-v C:\siebel.properties
```

   Information message BIP3427 indicates if a `siebel.properties` file is being used, and error message BIP3428 indicates if a problem occurs when attempting to access the `siebel.properties` file.

6. Optional: To connect to different versions of Siebel, follow the instructions in "Connecting to different versions of Siebel" on page 2077.

7. To check that the values have been set correctly, run the following command:

   ```
   mqsireportproperties MB7BROKER -c EISProviders -o Siebel -r
   ```

   The following example shows what is displayed by the **mqsireportproperties** command.

   ```
   ReportableEntityName=' '
   EISProviders
    Siebel=' '
     jarsURL='C:\Siebel_LIB'
   ```

```
    nativeLibs='C:\Siebel_LIB'
    siebelPropertiesURL=''

BIP8071I: Successful command completion.
```

8. Restart the broker.

What to do next

Next: configure the Siebel application to work with the adapter.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for Siebel Business Applications" on page 2002
With WebSphere Adapter for Siebel Business Applications, you can create integrated processes that exchange information with a Siebel application, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Activating IBM Tivoli License Manager for WebSphere Adapters" on page 2036
If you want to use IBM Tivoli License Manager (ITLM), you must activate it for each of the WebSphere Adapters.

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for Siebel properties" on page 4092
Reference information to refer to when you connect to a Siebel application.

"SiebelInput node" on page 4740
Use the SiebelInput node to interact with a Siebel application.

"SiebelRequest node" on page 4745
Use the SiebelRequest node to interact with a Siebel application.

Configuring the Siebel application to work with the adapter:

To configure the Siebel application, create an event table and a Siebel business object.

Before you begin

Before you start:

1. Add the required external software dependencies for Siebel.
2. Before you configure the Siebel application to work with WebSphere Adapter for Siebel Business Applications, you must create a user name and password so that the Adapter Connection wizard can connect to Siebel Business Applications to perform outbound operations, and retrieve Siebel business objects and services.

   You perform this task on the Siebel server, therefore ensure that you are familiar with the Siebel tools that are required to complete it. For information about using Siebel tools, refer to the Siebel tools documentation.

   To open Siebel Sales Enterprise on your local database, you must have administrative privileges.

About this task

To configure the Siebel application, you must create an event table and a Siebel business object. WebSphere Message Broker contains resources that help you to create the event components and triggers. This topic describes how to use those resources. You can also create the event table and Siebel business object manually; for more information, see "Creating the event store manually" on page 2072.

Procedure

1. Locate the scripts folder at *install_dir*/WMBT700/ResourceAdapters/Siebel_7.0.0.3_IF08/scripts.

   The scripts folder contains two folders: Siebel7.x.x and Siebel8.0. Each version has an Event_pkg folder, which contains a `.sif` file and a number of `.js` scripts. You use the `.sif` file to create the event components; it can add business objects, views, and all other Siebel objects to the Siebel repository. The `.js` scripts help you to create Siebel triggers.
2. To use the `.sif` file:
   a. Open Siebel tools and click **Tools** > **Import**.
   b. Import the `.sif` file.
   c. Merge the differences between the `.sif` file and the Siebel repository.
   d. Recompile the repository into a Siebel repository file (`.srf` file).
3. Use the `.js` scripts to create Siebel triggers. The provided samples show how to create entries in the inbound table when new Account objects are created.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for Siebel Business Applications" on page 2002
With WebSphere Adapter for Siebel Business Applications, you can create integrated processes that exchange information with a Siebel application, without special coding.

Related tasks:

"Creating the event store manually" on page 2072
To configure the Siebel application, create an event table and a Siebel business object.

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for Siebel properties" on page 4092
Reference information to refer to when you connect to a Siebel application.

"SiebelInput node" on page 4740
Use the SiebelInput node to interact with a Siebel application.

"SiebelRequest node" on page 4745
Use the SiebelRequest node to interact with a Siebel application.

Creating the event store manually:

To configure the Siebel application, create an event table and a Siebel business object.

About this task

"Configuring the Siebel application to work with the adapter" on page 2070 describes how to use the samples that are supplied with WebSphere Message Broker to configure the Siebel application. This topic describes how to create the event store manually.

The following steps describe how to create the event store to be used for inbound operations in the Siebel application. You can substitute all references to Siebel Sales Enterprise with the name of the Siebel application that you are using.

Procedure

1. Create a project called IBM, and lock the project with Siebel tools.
2. Using the object wizard, create an event table called CX_IBM_EVENT in which to store the events.
   a. In the event table, create the columns that are shown in the following table.

Column Name	Type	Length	Data Type	Required	Nullable	Status
DESCRIPTION	Data (public)	255	Varchar	No	Yes	Active

Column Name	Type	Length	Data Type	Required	Nullable	Status
EVENT_ID	Data (public)	30	Varchar	Yes	No	Active
EVENT_TYPE	Data (public)	20	Varchar	Yes	No	Active
OBJECT_KEY	Data (public)	255	Varchar	Yes	No	Active
OBJECT_NAME	Data (public)	255	Varchar	Yes	No	Active
PRIORITY	Data (public)	10	Varchar	No	Yes	Active
STATUS	Data (public)	20	Varchar	Yes	No	Active
XID	Data (public)	255	Varchar	Yes	No	Active

b. Create a new business component called IBM Event.
c. Create a new time stamp called Field Event, and map it to the CREATED column from CX_IBM_EVENT. Make the Type of this field DTYPE_UTCDATETIME.
d. Create a new business object called IBM Event.
e. Associate the IBM event business component to the IBM Event business object.
f. Create an applet called IBM Event List Applet, and base it on the IBM Event business component that you have created.
g. Create a view called IBM Event List View, and base it on the IBM Event business object that you have created.
h. Create a screen called IBM Event Screen, and associate it with the IBM Event List View in the Siebel tools.

3. Create a page tab.
   a. Click **Start Application** > **Siebel Sales Enterprise**.
   b. Right-click the **Page** tab, and click **New Record**.
   c. Specify IBM Event as the screen name, and IBM Event for the **Text - String Override** field.
   d. Leave the **Inactive** field blank.
4. Create a new business object called Schema Version for your IBM project and associate it with the Schema Version business component.
   a. Apply the physical schema for the new tables to your local database by querying for the new table, CX_IBM_EVENT_Q and selecting the current query to create a physical schema. Leave the table space and index space blank.
   b. Click **Activate** to activate the new schema.
5. Add or modify the Siebel VB or e-scripts for the business component that corresponds to the business objects that are used at your site. Siebel scripts trigger event notification for business objects. Samples are located in the Samples folder in your adapter installation.
6. Create a new Siebel repository file by compiling the updated and locked projects on your local database. The new repository file has an extension of `.srf`.
7. Create and populate a new responsibility.
   a. Open Siebel Sales Enterprise on your local database.
   b. Create a new responsibility called IBM Responsibility for IBM Event List View.
   c. Add the employees or teams who are responsible for reviewing events to the newly created IBM Responsibility.

d. Create a user name called IBMCONN (or another user name to be used by the adapter later). Add the user name to the newly created IBM Responsibility and also to the Administrative Responsibility.

8. Test the application in your local environment to ensure that you can see the IBM Event List View. An event is generated in the view after you create a record in the supported object. As part of the test, create a new Account business component instance in Siebel. Confirm that a new Account event is shown in the IBM Event List View (assuming that you have added the e-script trigger to the Account business component). If a new Account event is not displayed in the view, check for an error and fix it. For more information on the errors that might be generated, check either the Siebel support site or Siebel documentation.
9. When the test that you perform in Step 8 is successful, add your new and updated projects to your development server.
10. Activate the new table in the development server.
11. Compile a new Siebel repository (`.srf`) file on the server.
12. Back up the original repository file on the server.
13. Stop the Siebel server and replace the original repository file with the newly created one.
14. Restart the Siebel server.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for Siebel Business Applications" on page 2002
With WebSphere Adapter for Siebel Business Applications, you can create integrated processes that exchange information with a Siebel application, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Configuring the Siebel application to work with the adapter" on page 2070
To configure the Siebel application, create an event table and a Siebel business object.

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for Siebel properties" on page 4092
Reference information to refer to when you connect to a Siebel application.

"SiebelInput node" on page 4740
Use the SiebelInput node to interact with a Siebel application.

"SiebelRequest node" on page 4745
Use the SiebelRequest node to interact with a Siebel application.

Changing connection details for Siebel adapters:

Siebel nodes can get Siebel connection details from either the adapter component or a configurable service. By using configurable services, you can change the connection details for adapters without the need to redeploy the adapters. To pick up new values when a configurable service is created or modified, you must reload the broker or execution group to which the adapter was deployed, by using the **mqsistop** and **mqsistart** commands, or the **mqsireload** command.

Before you begin

Before you start:

- Read "WebSphere Adapters nodes" on page 1914 and "Configurable services" on page 1296 for background information.

About this task

Use the SiebelConnection configurable service to change connection details for a Siebel adapter. The Siebel node reads all connection properties from the adapter component that it is configured to use. If a configurable service exists that has the same name as the adapter component of the node, the node uses the values that are defined in that configurable service to override the corresponding properties from the adapter. If a configurable service is being used, all properties that are exposed by the configurable service are taken from the configurable service. The only properties that are taken from the adapter are those that you cannot set on the configurable service. The properties of the Siebel configurable service are described in "Configurable services properties" on page 3766.

You can also connect to different versions of Siebel by creating a custom EISProviders configurable service and setting the location of the appropriate library files. For more information, see "Connecting to different versions of Siebel" on page 2077.

Creating, changing, reporting, and deleting configurable services

Procedure

- To create a configurable service, use the WebSphere Message Broker Explorer; see "Using the WebSphere Message Broker Explorer to work with configurable services" on page 644 for more information, or the **mqsicreateconfigurableservice** command, as shown in the following example. This example creates a SiebelConnection configurable service for the Siebel instance that is running on *my.siebel.qa.com*:

```
mqsicreateconfigurableservice MB7BROKER -c SiebelConnection -o mySiebelAdapter.outadapter
-n connectString -v "siebel://my.siebel.qa.com/SBA_80/SSEObjMgr_enu"
```

To pick up the new values in the configurable service, restart the execution group and message flow.

- To change a configurable service, use the WebSphere Message Broker Explorer, or the **mqsichangeproperties** command, as shown in the following example. This example changes the connections that are used by the adapter *mySiebelAdapter.outadapter*. After you run this command, all adapters connect to the production system (`my.siebel.production.com`) instead of the test system (`my.siebel.qa.com`):

```
mqsichangeproperties MB7BROKER -c SiebelConnection -o mySiebelAdapter.outadapter -n connectString
-v "siebel://my.siebel.production.com/SBA_80/SSEObjMgr_enu"
```

To pick up the updated values in the configurable service, restart the execution group and message flow.

- To display all SiebelConnection configurable services, use the WebSphere Message Broker Explorer, or the **mqsireportproperties** command, as shown in the following example:

```
mqsireportproperties MB7BROKER -c SiebelConnection -o AllReportableEntityNames -r
```

- You can delete a configurable service that you have created by using the WebSphere Message Broker Explorer, or the **mqsideleteconfigurableservice** command, as shown in the following example:

```
mqsideleteconfigurableservice MB7BROKER -c SiebelConnection -o mySiebelAdapter.outadapter
```

You can delete only custom-named configurable services; you cannot delete IBM-defined configurable services.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Configurable services" on page 1296
Configurable services are typically runtime properties. You can use them to define properties that are related to external services on which the broker relies; for example, an SMTP server or a JMS provider.

"Using the WebSphere Message Broker Explorer to work with configurable services" on page 644
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view, add, modify and delete configurable services.

Related reference:

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"SiebelInput node" on page 4740
Use the SiebelInput node to interact with a Siebel application.

"SiebelRequest node" on page 4745
Use the SiebelRequest node to interact with a Siebel application.

Connecting to different versions of Siebel:

You can access multiple versions of Siebel from a single broker by using different versions of the client libraries.

About this task

You can connect to different versions of Siebel by creating a custom EISProviders configurable service and setting the location of the appropriate library files. You can then configure an execution group so that all Siebel nodes use this custom configurable service. Therefore, you can connect to different versions of a Siebel server from two different execution groups in the same broker.

You can complete this task by using the command line, the WebSphere Message Broker Explorer, or the Administration API for WebSphere Message Broker (CMP API). For more information about using the WebSphere Message Broker Explorer, see "Using the WebSphere Message Broker Explorer to work with configurable services" on page 644. For more information about using the CMP API, see "Developing applications that use the Administration API" on page 956. This topic describes how to connect to different versions of Siebel by using the command line.

Procedure

1. For each version of Siebel to which you want to connect, create a custom EISProviders configurable service by using the **mqsicreateconfigurableservice** command, as shown in the following example.

```
mqsicreateconfigurableservice MB7BROKER -c EISProviders -o EISProvidersName -n jarsURL -v "c:\siebelclient\"
```

   The **-o** parameter specifies the name of the EIS provider, and the **-n** and **-v** parameters are used to specify the file location of the EIS provider JAR files.

2. Configure your execution group to use the custom EISProviders configurable service by using the **mqsichangeproperties** command, as shown in the following example.

```
mqsichangeproperties MB7BROKER –e ExecutionGroupLabel –o ComIbmSiebelManager –n EISProviders –v EISProvidersName
```

3. Restart the execution group by using the **mqsistartmsgflow** command, or in the WebSphere Message Broker Toolkit by right-clicking the execution group and clicking **Start**.
4. Deploy message flows to the appropriate execution group.

Related concepts:

"Configurable services" on page 1296
Configurable services are typically runtime properties. You can use them to define properties that are related to external services on which the broker relies; for example, an SMTP server or a JMS provider.

"Using the WebSphere Message Broker Explorer to work with configurable services" on page 644
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view, add, modify and delete configurable services.

Related tasks:

"Interacting with a Siebel application" on page 2066
To interact with a Siebel application, obtain external software dependencies, run the Adapter Connection wizard, develop a message flow, then deploy the relevant resources.

"Developing applications that use the Administration API" on page 956
Develop Java applications that use the Administration API (also known as the CMP API) to communicate with, deploy to, and manage brokers and their associated resources.

Related reference:

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"WebSphere Adapter for Siebel properties" on page 4092
Reference information to refer to when you connect to a Siebel application.

Connecting to a clustered Siebel environment:

To connect to a Siebel server in a clustered environment, use a `siebel.properties` file.

Before you begin

Before you start:

Ensure the external software dependencies for Siebel have been set up, as described in "Adding external software dependencies for Siebel" on page 2068.

About this task

To connect to a Siebel server in a clustered environment, use the `siebel.conmgr.virtualhosts` property in the `siebel.properties` file to list the Siebel servers in the cluster.

The `siebel.conmgr.virtualhosts` property is used to list groups of servers with the same function. An incoming call attempts to connect to each server in a group in turn.

The only property in the `siebel.properties` file that WebSphere Message Broker supports is `siebel.conmgr.virtualhosts`. Information message BIP3427 indicates if a `siebel.properties` file is being used, and error message BIP3428 indicates if a problem occurs when attempting to access the `siebel.properties` file.

When the list of servers has been set up in the `siebel.properties` file, you specify the location of the file by using a configurable service, as described in the following steps.

Procedure

1. If you do not already have a `siebel.properties` file, create one with a `siebel.conmgr.virtualhosts` property that lists the Siebel servers in the cluster. Servers are listed by using a comma separated list in the format *hostname:port*.
2. Set the location of the `siebel.properties` file by using the **mqsichangeproperties** command; for example:

```
mqsichangeproperties MYBROKER -c EISProviders -o Siebel -n siebelPropertiesURL
-v C:\siebel.properties
```

Related concepts:

"Overview of WebSphere Adapter for Siebel Business Applications" on page 2002
With WebSphere Adapter for Siebel Business Applications, you can create integrated processes that exchange information with a Siebel application, without special coding.

Related tasks:

"Adding external software dependencies for Siebel" on page 2068
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapter for Siebel properties" on page 4092
Reference information to refer to when you connect to a Siebel application.

"SiebelInput node" on page 4740
Use the SiebelInput node to interact with a Siebel application.

"SiebelRequest node" on page 4745
Use the SiebelRequest node to interact with a Siebel application.

Interacting with a PeopleSoft application:

To interact with a PeopleSoft application, obtain external software dependencies, run the Adapter Connection wizard, develop a message flow, then deploy the relevant resources.

About this task

To connect to a PeopleSoft application, the PeopleSoft adapter requires certain files and libraries. You must store these files so that they are accessible to the Adapter Connection wizard. The wizard creates various resources, such as an adapter component and message flow. After you have completed the wizard, you can develop a message flow to define the interaction with the PeopleSoft application, then deploy the relevant resources.

Procedure

1. To obtain software dependencies, follow the instructions in "Adding external software dependencies for PeopleSoft" on page 2081.
2. To configure the PeopleSoft application to work with the adapter, create a custom event project, as described in "Creating a custom event project in PeopleTools" on page 2083.
3. Before you run the Adapter Connection wizard, gather the following information from your PeopleSoft administrator:
   - PeopleSoft user name
   - PeopleSoft password
   - PeopleSoft host name or IP address
   - Port number (for example, 9000)
   - Language code (for example, ENG)

   For more information, see "PeopleSoft connection properties for the Adapter Connection wizard" on page 4132.
4. To connect to PeopleSoft by using the Adapter Connection wizard, and create a message flow, follow the instructions in "Connecting to an EIS by using the Adapter Connection wizard" on page 2037.
5. Develop your message flow to define the interaction with the PeopleSoft application.
6. Deploy the appropriate resources, as described in "Deploying a message flow that uses WebSphere Adapters" on page 3240.
7. Optional: To change connection details for PeopleSoft adapters, see "Changing connection details for PeopleSoft adapters" on page 722.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for PeopleSoft Enterprise" on page 2013
With the adapter for PeopleSoft Enterprise you can create integrated processes that exchange information with PeopleSoft Enterprise through a standard interface. This interface shields the message flow developer from having to understand the lower level details about implementation of the application or data structures used on the

PeopleSoft Enterprise server.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Activating IBM Tivoli License Manager for WebSphere Adapters" on page 2036
If you want to use IBM Tivoli License Manager (ITLM), you must activate it for each of the WebSphere Adapters.

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for PeopleSoft properties" on page 4122
Reference information to refer to when you connect to a PeopleSoft application.

"PeopleSoftInput node" on page 4630
Use the PeopleSoftInput node to interact with a PeopleSoft application.

"PeopleSoftRequest node" on page 4635
Use the PeopleSoftRequest node to interact with a PeopleSoft application.

Adding external software dependencies for PeopleSoft:

Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

Before you begin

Before you start:

Ensure that you have the relevant prerequisite files for your PeopleSoft system.

- `psjoa.jar`
- A JAR file that contains the component interface API classes

Save both support files to a directory such as `C:\PeopleSoft_LIB.` (On Windows, the directory cannot be a mapped network drive on a remote Windows computer; the directory must be local or on a Storage Area Network (SAN) disk.) You can find the `psjoa.jar` file in the following location on the PeopleSoft Application Server: *`peopleTools_installation_directory`*`\web\PSJOA\psjoa.jar`. Use PeopleTools to generate the component interface JAR file for your business objects.

The sample resources that you need to set up the PeopleSoft system so that it can communicate with the broker are in *install_dir*\ResourceAdapters\PeopleSoft_7.0.0\samples.

About this task

Locating the PeopleSoft support files in the run time on Windows

To add the PeopleSoft prerequisite files to the run time, complete the following steps.

Procedure

1. Ensure that the broker has started.
2. Either open the Command Console, or open a Windows command prompt and enter **mqsiprofile** to initialize the environment.
3. Enter the following command to display the locations of the prerequisite JAR files and native libraries:

   ```
   mqsireportproperties MB7BROKER -c AllTypes -o AllReportableEntityNames  -r
   ```

   The following example shows what typically is displayed when you run this command:

   ```
   ReportableEntityName=''
    EISProviders
     JDEdwards=''
      jarsURL='default_Path'
      nativeLibs='default_Path'
     PeopleSoft=''
      jarsURL='default_Path'
      nativeLibs='default_Path'
     SAP=''
      jarsURL='default_Path'
      nativeLibs='default_Path'
     Siebel=''
      jarsURL='default_Path'
      nativeLibs='default_Path'
      siebelPropertiesURL=''
     Twineball=''
      jarsURL='default_Path'
      nativeLibs='default_Path'
   ```

4. Set the location of the PeopleSoft prerequisite files by using the following command:

```
mqsichangeproperties MB7BROKER -c EISProviders -o PeopleSoft -n jarsURL -v C:\PeopleSoft_LIB
```

5. To check that the values have been set correctly, run the following command:

   ```
   mqsireportproperties MB7BROKER -c EISProviders -o PeopleSoft -r
   ```

   The following example shows what is displayed by the **mqsireportproperties** command.

   ```
   ReportableEntityName=' '
   EISProviders
    PeopleSoft=' '
     jarsURL='C:\PeopleSoft_LIB'

   BIP8071I: Successful command completion.
   ```

6. Restart the broker.

What to do next

Next: create a custom event project in PeopleTools.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for PeopleSoft Enterprise" on page 2013
With the adapter for PeopleSoft Enterprise you can create integrated processes that exchange information with PeopleSoft Enterprise through a standard interface. This interface shields the message flow developer from having to understand the lower level details about implementation of the application or data structures used on the PeopleSoft Enterprise server.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Activating IBM Tivoli License Manager for WebSphere Adapters" on page 2036
If you want to use IBM Tivoli License Manager (ITLM), you must activate it for each of the WebSphere Adapters.

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for PeopleSoft properties" on page 4122
Reference information to refer to when you connect to a PeopleSoft application.

"PeopleSoftInput node" on page 4630
Use the PeopleSoftInput node to interact with a PeopleSoft application.

"PeopleSoftRequest node" on page 4635
Use the PeopleSoftRequest node to interact with a PeopleSoft application.

Creating a custom event project in PeopleTools:

The WebSphere Adapter requires an event project in PeopleSoft to perform asynchronous inbound event processing. Use PeopleTools to create the custom event project.

Before you begin

Before you start:

Add the required external software dependencies for PeopleSoft.

About this task

If your environment requires inbound event support, you must use a custom event project in PeopleSoft. A sample event project, IBM_EVENT_V600, is provided with the adapter. You can modify and use and the sample project, or you can create your own project by using PeopleTools. If you create your own project, make sure that you complete the following steps.

Procedure

1. Use PeopleTools Application Designer to create and name a new project.
2. Create the fields for the new project as described in the following table:

Field name	Field description
IBM_EVENT_ID	A numeric value that is retrieved from IBM_FETCH_ID record. This value is a unique ID for the event.
IBM_OBJECT_NAME	The name of the corresponding business graph.
IBM_OBJECT_KEYS	The get key property names in the Component Interface, followed by the key values in name-value pairs. This information is used for the component's retrieval from the EIS.
IBM_EVENT_STATUS	If the event is ready to be polled, the status is set to 0 and the IBMPublishEvent function is called.
IBM_OBJECT_VERB	The verb that is set on the business object graph that contains the retrieved business object.
IBM_EVENT_DTTM	The date on which the event is created. For a future dated event, this is the effective date.
IBM_NEXT_EVENT_ID	The field that has the latest event ID under the record IBM_FETCH_ ID. This field is incremented for each event that is added to the IBM_EVENT_TBL, and it populates the IBM_EVENT_ID field in that table.
IBM_XID	The transaction ID that is needed to provide assured event delivery.

3. Create a record named IBM_EVENT_TBL and add to it all the fields that you have just created, except IBM_NEXT_EVENT_ID.
4. Create a record named IBM_FETCH_ID and add to it only the IBM_NEXT_EVENT_ID field.
5. Open the IBM_FETCH_ID record, select the IBM_NEXT_EVENT_ID field, view the PeopleCode, and select **fieldformula.**
6. Copy the PeopleCode for a custom event project from "PeopleCode for a custom event project" on page 4125 to the project that you are creating.

7. Create a page under your project that contains the fields of the IBM_EVENT_TBL record at level 0. The page can have any name.
8. Create a component under your project that contains the page that you have just created. The component can have any name.
9. Create a Component Interface against this component and give it any name. Confirm that you want to default the properties that are based on the underlying component definition.
10. Build the entire project, selecting all create options.
11. Test and confirm that the Component Interface works, by using the Component Interface tester.
12. Generate the Java APIs for the Component Interface, then add the generated classes to the adapter classpath. For complete information about building a PeopleTools project and testing the PeopleSoft Component Interface, refer to PeopleSoft documentation.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for PeopleSoft Enterprise" on page 2013
With the adapter for PeopleSoft Enterprise you can create integrated processes that exchange information with PeopleSoft Enterprise through a standard interface. This interface shields the message flow developer from having to understand the lower level details about implementation of the application or data structures used on the PeopleSoft Enterprise server.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for PeopleSoft properties" on page 4122
Reference information to refer to when you connect to a PeopleSoft application.

"PeopleSoftInput node" on page 4630
Use the PeopleSoftInput node to interact with a PeopleSoft application.

"PeopleSoftRequest node" on page 4635
Use the PeopleSoftRequest node to interact with a PeopleSoft application.

Changing connection details for PeopleSoft adapters:

PeopleSoft nodes can get PeopleSoft connection details from either the adapter component or a configurable service. By using configurable services, you can change the connection details for adapters without the need to redeploy the adapters. To pick up new values when a configurable service is created or modified, you must reload the broker or execution group to which the adapter was deployed, by using the **mqsistop** and **mqsistart** commands, or the **mqsireload** command.

Before you begin

Before you start:

- Read "WebSphere Adapters nodes" on page 1914 and "Configurable services" on page 1296 for background information.

About this task

Use the PeopleSoftConnection configurable service to change connection details for a PeopleSoft adapter. The PeopleSoft node reads all connection properties from the adapter component that it is configured to use. If a configurable service exists that has the same name as the node's adapter component, the node uses the values that are defined in that configurable service to override the corresponding properties from the adapter. If a configurable service is being used, all properties that are exposed by the configurable service are taken from the configurable service. The only properties that are taken from the adapter are those that you cannot set on the configurable service. The properties of the PeopleSoft configurable service are described in "Configurable services properties" on page 3766.

Creating, changing, reporting, and deleting configurable services

Procedure

- To create a configurable service, use the WebSphere Message Broker Explorer; see "Using the WebSphere Message Broker Explorer to work with configurable services" on page 644 for more information, or the **mqsicreateconfigurableservice** command, as shown in the following example. This example creates a PeopleSoftConnection configurable service for the PeopleSoft instance that is running on *my.peoplesoft.qa.com*:

```
mqsicreateconfigurableservice MB7BROKER -c PeopleSoftConnection -o myPeopleSoftAdapter.outadapter
-n hostName,port -v "my.peoplesoft.qa.com",9000
```

  To pick up the new values in the configurable service, restart the execution group and message flow.
- To change a configurable service, use the WebSphere Message Broker Explorer, or the **mqsichangeproperties** command, as shown in the following example. This example changes the connections that are used by the adapter *myPeopleSoftAdapter.outadapter*. After you run this command, all adapters connect to the production system (`my.peoplesoft.production.com`) instead of the test system (`my.peoplesoft.qa.com`):

```
mqsichangeproperties MB7BROKER -c PeopleSoftConnection -o myPeopleSoftAdapter.outadapter -n hostName
-v "my.peoplesoft.production.com"
```

To pick up the updated values in the configurable service, restart the execution group and message flow.

- To display all PeopleSoftConnection configurable services, use the WebSphere Message Broker Explorer, or the **mqsireportproperties** command, as shown in the following example:

```
mqsireportproperties MB7BROKER -c PeopleSoftConnection -o AllReportableEntityNames -r
```

- You can delete a configurable service that you have created by using the WebSphere Message Broker Explorer, or the **mqsideleteconfigurableservice** command, as shown in the following example:

```
mqsideleteconfigurableservice MB7BROKER -c PeopleSoftConnection -o myPeopleSoftAdapter.outadapter
```

You can delete only custom-named configurable services; you cannot delete IBM-defined configurable services.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Configurable services" on page 1296
Configurable services are typically runtime properties. You can use them to define properties that are related to external services on which the broker relies; for example, an SMTP server or a JMS provider.

"Using the WebSphere Message Broker Explorer to work with configurable services" on page 644
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view, add, modify and delete configurable services.

Related reference:

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"PeopleSoftInput node" on page 4630
Use the PeopleSoftInput node to interact with a PeopleSoft application.

"PeopleSoftRequest node" on page 4635
Use the PeopleSoftRequest node to interact with a PeopleSoft application.

Interacting with a JD Edwards application:

To interact with a JD Edwards application, obtain external software dependencies, run the Adapter Connection wizard, develop a message flow, then deploy the relevant resources.

Before you begin

Before you start:

Gather the following information from your JD Edwards administrator; you will need this information when you run the Adapter Connection wizard.

- JD Edwards EnterpriseOne environment name
- JD Edwards user name
- JD Edwards password
- The role that is associated with the user name

To find objects in the JD Edwards system, you use the Adapter Connection wizard to run a query to discover business functions or XML lists. Ask your JD Edwards administrator about the available business function libraries or XML list tables. To complete the wizard, you need to know the table type and the attributes to specify in the query. For more information, see JD Edwards connection properties for the Adapter Connection wizard.

You need the following information to complete the system connection information in the Adapter Connection wizard. You can find this information in the `jdbj.ini` file that is included with the external JD Edwards resource files.

```
[JDBj-BOOTSTRAP SESSION]
user=user
password=***
environment=JDEenv
role=*ALL
```

Connecting to a DB2 database

If you are connecting to a DB2 database, the JDBC driver file that is required for DB2 can be found in the JDBC driver files table (see "Adding external software dependencies for JD Edwards EnterpriseOne" on page 2090). The `jdbj.ini` file contains the following example entries, which have been configured by the JD Edwards administrator:

```
[JDBj-BOOTSTRAP DATA SOURCE]

databaseType=W

[JDBj-JDBC DRIVERS]

UDB=COM.ibm.db2.jdbc.app.DB2Driver
```

Connecting to an Oracle database

If you are Connecting to an Oracle database, the JDBC driver files that are required for Oracle can be found in the JDBC driver files table (see "Adding external software dependencies for JD Edwards EnterpriseOne" on page 2090). The `jdbj.ini` file contains the following example entries, which have been configured by the JD Edwards administrator:

```
[JDBj-JDBC DRIVERS]
ORACLE=oracle.jdbc.driver.OracleDriver

[JDBj-ORACLE]
tns=tnsnames.ora
```

About this task

To connect to a JD Edwards application, the JD Edwards adapter requires certain files and libraries. You must store these files so that they are accessible to the Adapter Connection wizard. The wizard creates various resources, such as an adapter component and message flow. After you have completed the wizard, you can develop a message flow to define the interaction with the JD Edwards application, then deploy the relevant resources.

Procedure

1. To obtain software dependencies, follow the instructions in "Adding external software dependencies for JD Edwards EnterpriseOne" on page 2090.
2. To connect to JD Edwards by using the Adapter Connection wizard, and create a message flow, follow the instructions in "Connecting to an EIS by using the Adapter Connection wizard" on page 2037.
3. Develop your message flow to define the interaction with the JD Edwards application.
4. Deploy the appropriate resources, as described in "Deploying a message flow that uses WebSphere Adapters" on page 3240.
5. Optional: To change connection details for JD Edwards adapters, see "Changing connection details for JD Edwards adapters" on page 724.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for JD Edwards EnterpriseOne" on page 2023
With the WebSphere Adapter for JD Edwards EnterpriseOne, you can create integrated processes that include the exchange of information with a JD Edwards EnterpriseOne server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Activating IBM Tivoli License Manager for WebSphere Adapters" on page 2036
If you want to use IBM Tivoli License Manager (ITLM), you must activate it for each of the WebSphere Adapters.

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapter for JD Edwards EnterpriseOne properties" on page 4146
Reference information to refer to when you connect to a JD Edwards EnterpriseOne application.

"JDEdwardsRequest node" on page 4524
Use the JDEdwardsRequest node to interact with a JD Edwards EnterpriseOne server.

Adding external software dependencies for JD Edwards EnterpriseOne:

Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

Before you begin

Before you start:

Obtain the relevant prerequisite files for your JD Edwards EnterpriseOne system from your JD Edwards EnterpriseOne administrator. The necessary files are listed in the following table. The software dependencies differ, depending on which version of JD Edwards EnterpriseOne Tools you use.

JD Edwards EnterpriseOne Tools, version 8.9 (SP1, SP2), 8.93	**JD Edwards EnterpriseOne Tools, version 8.94**	**JD Edwards EnterpriseOne Tools, version 8.95, 8.96**	**JD Edwards EnterpriseOne Tools, version 8.97, 8.98**
JDBC driver files (For more information, see the following table.)	JDBC driver files (For more information, see the following table.)	JDBC driver files (For more information, see the following table.)	JDBC driver files (For more information, see the following table.)
connector.jar	Common_Jar.jar	ApplicationAPIs_JAR.jar	ApplicationAPIs_JAR.jar
database.jar	Connector.jar	ApplicationLogic_JAR.jar	ApplicationLogic_JAR.jar
jdeinterop.ini	database.jar	Base_JAR.jar	Base_JAR.jar
jdeLog.properties	EventProcessor_EJB.jar	BizLogicContainer_JAR.jar	BizLogicContainer_JAR.jar
kernel.jar	jdeutil.jar	BizLogicContainerClient_JAR.jar	BizLogicContainerClient_JAR.jar
log4j.jar	jdbj.ini	bootstrap.jar	BusinessLogicServices_JAR.jar
owra.jar	jdeinterop.ini	castor.jar	castor.jar
xalan.jar	jdelog.properties	Connector.jar	commons-httpclient-3.0.jar
xerces.jar	kernel.jar	ecutils.jar	commons-logging.jar
	log4j.jar	EventProcessor_JAR.jar	Connector.jar
	xalan.jar	EventProcessor_EJB.jar	EventProcessor_EJB.jar
	xerces.jar	jdbj.ini	EventProcessor_JAR.jar
		JdbjBase_JAR.jar	Generator_JAR.jar
		JdbjInterfaces_JAR.jar	jdbj.ini
		jdeinterop.ini	JdbjBase_JAR.jar
		jdelog.properties	JdbjInterfaces_JAR.jar
		JdeNet_JAR.jar	jdeinterop.ini

JD Edwards EnterpriseOne Tools, version 8.9 (SP1, SP2), 8.93	JD Edwards EnterpriseOne Tools, version 8.94	JD Edwards EnterpriseOne Tools, version 8.95, 8.96	JD Edwards EnterpriseOne Tools, version 8.97, 8.98
		lmproxy.jar	jdelog.properties
		log4j.jar	JdeNet_JAR.jar
		messagingClient.jar	jmxremote.jar
		naming.jar	jmxremote_optional.jar
		PMApi_JAR.jar	jmxri.jar
		Spec_JAR.jar	log4j.jar
		System_JAR.jar	ManagementAgent_JAR.jar
		urlprotocols.jar	Metadata.jar
		xalan.jar	MetadataInterface.jar
		xerces.jar	PMApi_JAR.jar
			Spec_JAR.jar
			System_JAR.jar
			SystemInterfaces_JAR.jar
			xalan.jar
			xerces.jar
			xmlparserv2.jar

JDBC driver files

Database	JDBC driver files	Implementation class
Oracle	tsnames.ora classes12.zip	oracle.jdbc.driver.OracleDriver
SQLServer	sqljdbc.jar	com.ibm.microsoft.sqlserver.jdbc.SQLServerDriver
AS/400	jt400.jar	com.ibm.as400.access.AS400JDBCDriver
DB2 Type-2 (JDK 1.4/1.5)	db2java.zip	com.ibm.db2.jdbc.app.DB2Driver
DB2 Type-4 (JDK 1.4/1.5)	db2jcc.jar db2jcc_license_cu.jar	com.ibm.db2.jcc.DB2Driver
DB2 Type-4 (JDK 1.6)	db2jcc4.jar	com.ibm.db2.jcc.DB2Driver

Copy the external dependency files to a temporary location. For example, copy them to `C:\temp\JDE_dependencies\`.

Tip: The Adapter Connection wizard can browse the JDBC driver files more easily if they are stored in their own folder. For example, if you are using an Oracle database server, you put the `tnsnames.ora` and `classes12.zip` files in the following location: `C:\temp\JDE_dependencies\`.

About this task

Locating the JD Edwards support files in the runtime environment on Windows

To add the JD Edwards prerequisite files to the runtime environment, complete the following steps.

Procedure

1. Ensure that the broker has started.
2. Either open the Command Console, or open a Windows command prompt and enter **mqsiprofile** to initialize the environment.
3. Enter the following command to display the locations of the prerequisite JAR files:

   ```
   mqsireportproperties MB7BROKER -c AllTypes -o AllReportableEntityNames  -r
   ```

   The following example shows what typically is displayed when you run this command:

   ```
   ReportableEntityName=''
    EISProviders
     JDEdwards=''
      jarsURL='default_Path'
      nativeLibs='default_Path'
     PeopleSoft=''
      jarsURL='default_Path'
      nativeLibs='default_Path'
     SAP=''
      jarsURL='default_Path'
      nativeLibs='default_Path'
     Siebel=''
      jarsURL='default_Path'
      nativeLibs='default_Path'
     Twineball=''
      jarsURL='default_Path'
      nativeLibs='default_Path'
   ```

4. Set the location of the JD Edwards prerequisite files by using the following command:

```
mqsichangeproperties MB7BROKER -c EISProviders -o JDEdwards -n jarsURL
-v C:\temp\JDE_dependencies\
mqsichangeproperties MB7BROKER -c EISProviders -o JDEdwards -n nativeLibs
-v C:\temp\JDE_dependencies\
```

5. To check that the values have been set correctly, run the following command:

   ```
   mqsireportproperties MB7BROKER -c EISProviders -o JDEdwards -r
   ```

   The following example shows what is displayed by the **mqsireportproperties** command.

   ```
   ReportableEntityName=' '
   EISProviders
    JDEdwards=' '
     jarsURL='C:\temp\JDE_dependencies'
     nativeLibs='default_Path'

   BIP8071I: Successful command completion.
   ```

6. Restart the broker.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for JD Edwards EnterpriseOne" on page 2023
With the WebSphere Adapter for JD Edwards EnterpriseOne, you can create integrated processes that include the exchange of information with a JD Edwards EnterpriseOne server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Activating IBM Tivoli License Manager for WebSphere Adapters" on page 2036
If you want to use IBM Tivoli License Manager (ITLM), you must activate it for each of the WebSphere Adapters.

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapter for JD Edwards EnterpriseOne properties" on page 4146
Reference information to refer to when you connect to a JD Edwards EnterpriseOne application.

"JDEdwardsRequest node" on page 4524
Use the JDEdwardsRequest node to interact with a JD Edwards EnterpriseOne server.

Changing connection details for JD Edwards adapters:

JD Edwards nodes can get JD Edwards EnterpriseOne connection details from either the adapter component or a configurable service. By using configurable services, you can change the connection details for adapters without the need to redeploy the adapters. To pick up new values when a configurable service is created or modified, you must reload the broker or execution group to which the adapter was deployed, by using the **mqsistop** and **mqsistart** commands, or the **mqsireload** command.

Before you begin

Before you start:

- Read "WebSphere Adapters nodes" on page 1914 and "Configurable services" on page 1296 for background information.

About this task

Use the JDEdwardsConnection configurable service to change connection details for a JD Edwards adapter. The JD Edwards node reads all connection properties from the adapter component that it is configured to use. If a configurable service exists that has the same name as the nodes adapter component, the node uses the values that are defined in that configurable service to override the corresponding properties from the adapter. If a configurable service is being used, all properties that are exposed by the configurable service are taken from the configurable service. The only properties that are taken from the adapter are the ones that you

cannot set on the configurable service. The properties of the JD Edwards configurable service are described in "Configurable services properties" on page 3766.

Creating, changing, reporting, and deleting configurable services

Procedure

- To create a configurable service, use the WebSphere Message Broker Explorer; see "Using the WebSphere Message Broker Explorer to work with configurable services" on page 644 for more information, or the **mqsicreateconfigurableservice** command, as shown in the following example. This example creates a JDEdwardsConnection configurable service to connect to the DV7333 development Environment for a user with the Role of administrator.

```
mqsicreateconfigurableservice MYBROKER -c JDEdwardsConnection -o myJdedwardsAdapter.outadapter
-n Environment,Role -v "dv7333,administrator"
```

  To pick up the new values in the configurable service, restart the execution group and message flow.
- To change a configurable service, use the WebSphere Message Broker Explorer, or the **mqsichangeproperties** command, as shown in the following example. This example changes the connection that is used by the adapter *myJdedwardsAdapter.outadapter* to connect to the PD7333 production Environment. After you run this command, all adapters connect to the production Environment (PD7333) instead of the development system (DV7333).

```
mqsichangeproperties MYBROKER -c JDEdwardsConnection -o myJdedwardsAdapter.outadapter -n Environment,Role
-v PD7333,administrator
```

  To pick up the updated values in the configurable service, restart the execution group and message flow.
- To display all JDEdwardsConnection configurable services, use the WebSphere Message Broker Explorer, or the **mqsireportproperties** command, as shown in the following example:

```
mqsireportproperties MYBROKER -c JDEdwardsConnection -o AllReportableEntityNames -r
```

- You can delete a configurable service that has been created by using the WebSphere Message Broker Explorer, or the **mqsideleteconfigurableservice** command, as shown in the following example:

```
mqsideleteconfigurableservice MYBROKER -c JDEdwardsConnection -o myJdedwardsAdapter.outadapter
```

  You can delete only custom-named configurable services; you cannot delete IBM-defined configurable services.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Configurable services" on page 1296
Configurable services are typically runtime properties. You can use them to define properties that are related to external services on which the broker relies; for example, an SMTP server or a JMS provider.

"Using the WebSphere Message Broker Explorer to work with configurable services" on page 644
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view, add, modify and delete configurable services.

Related reference:

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"JDEdwardsRequest node" on page 4524
Use the JDEdwardsRequest node to interact with a JD Edwards EnterpriseOne server.

Working with WebSphere Process Server

WebSphere Process Server is a business integration server that uses Service Component Architecture (SCA) to present all business transactions in a service-oriented way in its runtime environment. WebSphere Message Broker can accept requests from WebSphere Process Server, and can also call service components on WebSphere Process Server.

About this task

The topics in this section describe what SCA is, and how you can configure your message flows to use SCA to interoperate with WebSphere Process Server.

- "Working with Service Component Architecture (SCA)"
- "Service Component Architecture (SCA) overview" on page 2096
- "Interoperability with WebSphere Process Server" on page 2097
- "SCA nodes" on page 2101
- "Using Broker SCA definitions to configure message flows" on page 2102
- "Developing SCA applications for non-XML data" on page 2103
- "Message flows for Service Component Architecture (SCA)" on page 2105

Working with Service Component Architecture (SCA)

Start here to find out how you can use SCA to allow interoperability with WebSphere Process Server Version 6.2.

SCA is a specification that describes a model for building applications and systems using a service-oriented architecture (SOA).

WebSphere Message Broker provides built-in message flow nodes that allow interoperability with WebSphere Process Server. These nodes support inbound and outbound scenarios with WebSphere Process Server. In an inbound scenario, a service component in WebSphere Process Server can send a request to WebSphere

Message Broker. In an outbound scenario, a message flow in WebSphere Message Broker can send a synchronous or asynchronous request to WebSphere Process Server.

The nodes are configured by a Broker SCA definition. You can create a Broker SCA definition in two ways:

- From an SCA import or SCA export component that is defined in WebSphere Integration Developer Version 6.2 and imported into WebSphere Message Broker.
- From a WebSphere Message Broker message set.

The SCA nodes cannot be used without a Broker SCA definition to configure them.

If you want to call services available on WebSphere Message Broker, you must either:

- Import an SCA import component from WebSphere Integration Developer and create a message flow that uses an SCAInput node, or
- Generate an inbound Broker SCA definition and export the `.insca` file to WebSphere Integration Developer

If you want to call a service in WebSphere Process Server, you must either:

- Import an SCA export component from WebSphere Integration Developer and configure a message flow using the Broker SCA definition that is created, or
- Generate an outbound Broker SCA definition and export the `.outsca` file to WebSphere Integration Developer

When designing message flows with SCA, you must select a suitable transport (also called a *binding*). The SCA nodes support these transports:

- WebSphere MQ Enterprise transport
- WebSphere Broker HTTP transport with SOAP

Important: The SCA nodes do not support the SCA binding in WebSphere Integration Developer, which is used only for SCA invocations between modules running within WebSphere Process Server.

The following topics describe how to work with SCA:

- "Service Component Architecture (SCA) overview"
- "Message flows for Service Component Architecture (SCA)" on page 2105

Service Component Architecture (SCA) overview

Service Component Architecture (SCA) is a specification that describes a model for building applications and systems using a service-oriented architecture (SOA).

SCA simplifies the creation and integration of business applications that are built using an SOA by separating business logic from its implementation so that you can focus on assembling an integrated application without knowing details of its implementation.

SCA divides the steps of building a service-oriented application into two major parts:

- The implementation of components that provide services and use other services.
- The assembly of these service components to build the business application.

Service components can be assembled graphically using WebSphere Integration Developer, and the implementation can be added later.

For more information about SCA, see SCA.

WebSphere Process Server is a business integration server that supports solutions that are based on SOA; see "Interoperability with WebSphere Process Server."

Related concepts:

"Interoperability with WebSphere Process Server"
WebSphere Process Server is a business integration server that supports solutions based on service-oriented architecture (SOA). It uses Service Component Architecture (SCA) to present all business transactions in a service-oriented way in its runtime environment. WebSphere Message Broker can accept requests from WebSphere Process Server, and can also call service components on WebSphere Process Server.

"SCA nodes" on page 2101
SCA nodes support interoperability between WebSphere Message Broker and WebSphere Process Server.

"Using Broker SCA definitions to configure message flows" on page 2102
Use Broker SCA definitions to create and configure SCA nodes in a message flow.

"Message flows for Service Component Architecture (SCA)" on page 2105
Use the SCA nodes to develop message flows so that WebSphere Message Broker can interoperate with WebSphere Process Server. These nodes support scenarios in which service components that run on WebSphere Process Server either are called from WebSphere Message Broker or call a WebSphere Message Broker message flow.

"Working with Service Component Architecture (SCA)" on page 2095
Start here to find out how you can use SCA to allow interoperability with WebSphere Process Server Version 6.2.

"Service Component Architecture (SCA) overview" on page 2096
Service Component Architecture (SCA) is a specification that describes a model for building applications and systems using a service-oriented architecture (SOA).

Interoperability with WebSphere Process Server

WebSphere Process Server is a business integration server that supports solutions based on service-oriented architecture (SOA). It uses Service Component Architecture (SCA) to present all business transactions in a service-oriented way in its runtime environment. WebSphere Message Broker can accept requests from WebSphere Process Server, and can also call service components on WebSphere Process Server.

WebSphere Process Server provides components that can use, or be used by, the services in WebSphere Message Broker. This interoperability is based on SCA, and the SCA nodes provide the facilities that permit operation between the two products.

WebSphere Integration Developer is the development environment for WebSphere Process Server. It is the tool for building and deploying SOA-based integration solutions on WebSphere Process Server.

When designing a solution, the Integration Developer can choose either of the following approaches:

"Starting from WebSphere Integration Developer" on page 2099
Design the applications in WebSphere Integration Developer, export your project as a Project Interchange (PI) file, then import it into WebSphere

Message Broker. For more details, see “Importing and exporting resources in a Project Interchange file” on page 1452.

“Starting from WebSphere Message Broker” on page 2100
Design the applications in WebSphere Message Broker, export the Broker SCA definitions, then import them into WebSphere Integration Developer.

Differences in WSDL definition files

When you compare messages on the wire from WebSphere Message Broker and WebSphere Integration Developer, the root element might be different. WSDL generated by WebSphere Message Broker creates messages whose root element is the element name of the message. In most cases, this element name is the same as the message definition name that is used to generate the WSDL. However in WebSphere Integration Developer, messages are wrapped with an additional element whose name is the same as the name of the operation; consequently, the root element has the name of the operation.

Related tasks:

“Exporting an SCA Import or Export from a Broker SCA definition” on page 2945
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to export SCA import or SCA export components. You must export an SCA import or SCA export if the Broker SCA definition is to be used by WebSphere Integration Developer.

“Generating a Broker SCA definition from a message set” on page 2967
WebSphere Message Broker creates a Broker SCA definition from existing message definitions.

“Importing SCA import or SCA export components” on page 2943
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to import SCA import or SCA export components from WebSphere Integration Developer. You must import an SCA import or SCA export into the workspace to provide a broker SCA definition for use in configuring the SCA nodes.

“Developing SCA applications for non-XML data” on page 2103
The SCA nodes allow non-XML data to be sent and received from WebSphere Process Server by using the WebSphere MQ binding. For example, create a message model from a COBOL copybook, and use that message model to parse messages received from WebSphere Process Server.

Related reference:

“SCAInput node” on page 4707
Use the SCAInput node with the SCAReply node to process messages from WebSphere Process Server.

“SCAReply node” on page 4726
Use the SCAReply node to send a message from the broker to the originating client in response to a message received by a SCAInput node.

“SCAAsyncRequest node” on page 4690
Use the SCAAsyncRequest node with the SCAAsyncResponse node to construct a pair of message flows that invoke a WebSphere Process Server service component asynchronously.

“SCAAsyncResponse node” on page 4698
Use the SCAAsyncResponse node with the SCAAsyncRequest node to construct a pair of message flows that start a component asynchronously.

Starting from WebSphere Integration Developer:

You can design SOA applications in WebSphere Integration Developer (IBM Integration Designer) that need to call services in WebSphere Message Broker, or be called by services in WebSphere Message Broker

- If you are developing a service on WebSphere Integration Developer and you want WebSphere Message Broker to be able to send requests to it, you create an export component in your WebSphere Integration Developer application.
- If you want to call a WebSphere Message Broker service from your WebSphere Integration Developer application, you create an import component in your application.

After you have created your WebSphere Integration Developer application, you export a PI file, which you can then import into WebSphere Message Broker.

Use the New Message Definition File wizard or the Start from SCA Import or Export Quick Start wizard to import WebSphere Integration Developer export and import components. See "Importing SCA import or SCA export components" on page 2943. Ensure that the import or export components that you import from WebSphere Integration Developer use SOAP 1.1 bindings; a validation error occurs if you attempt to import SCA import components or SCA export components that have been generated with SOAP 1.2 bindings.

Importing the PI file creates the message definition file and the Broker SCA definition files.

If you import a WebSphere Integration Developer import component and create a Broker SCA definition from it, you create a Broker SCA definition with a file extension of `.insca`. The Broker SCA definition contains:

- binding information (in a file with an extension of `.import`)
- WSDL and XSD files that describe the message format

Broker SCA definitions that are created from imported WebSphere Integration Developer import components provide values for properties on the SCAInput node which can then receive requests from WebSphere Process Server.

If you import a WebSphere Integration Developer export component and create a Broker SCA definition from it, you create a Broker SCA definition with a file extension of `.outsca`. This file contains:

- binding information (in a file with an extension of `.export`)
- WSDL and XSD files that describe the message format

Broker SCA definitions that are created from imported WebSphere Integration Developer export components provide values for properties on the SCARequest node, or the SCAAsyncRequest and SCARequest nodes, which can then send requests to WebSphere Process Server.

Related concepts:

"Interoperability with WebSphere Process Server" on page 2097
WebSphere Process Server is a business integration server that supports solutions based on service-oriented architecture (SOA). It uses Service Component Architecture (SCA) to present all business transactions in a service-oriented way in its runtime environment. WebSphere Message Broker can accept requests from WebSphere Process Server, and can also call service components on WebSphere Process Server.

"Starting from WebSphere Message Broker"
You can design SOA applications in WebSphere Message Broker that need to call services in WebSphere Process Server or be called by services in WebSphere Process Server.

Starting from WebSphere Message Broker:

You can design SOA applications in WebSphere Message Broker that need to call services in WebSphere Process Server or be called by services in WebSphere Process Server.

- If you are developing a service on WebSphere Message Broker and you want a WebSphere Integration Developer application to be able to send requests to it, you generate an `.insca` Broker SCA Definition file. You use the Broker SCA Definition file to create an SCAInput node, which can receive requests from WebSphere Process Server. You also export this file from WebSphere Message Broker to the file system or workspace. You can import the exported artifacts into WebSphere Integration Developer. Importing the file into WebSphere Integration Developer creates an Import component in your WebSphere Integration Developer application; to access a service provided by WebSphere Message Broker, connect this Import component in the assembly diagram.
- If you are developing a service on WebSphere Message Broker and you want to access a service on WebSphere Process Server, you generate an `.outsca` Broker SCA Definition file. You use the Broker SCA Definition file to create either:
  - A pair of SCAAsyncRequest and SCAAsyncResponse nodes, for asynchronous operations.
  - A SCARequest node, for operations that are synchronous or one-way.

  The SCAAsyncRequest or SCARequest node sends requests to WebSphere Process Server. You also export this file from WebSphere Message Broker to the file system or workspace. You can import the exported artifacts into WebSphere Integration Developer. Importing the artifacts into WebSphere Integration Developer creates an Export component in your WebSphere Integration Developer application; to receive requests from a broker application, connect this Export component in the assembly diagram.

Use the Generate Broker SCA Definition wizard to generate Broker SCA definitions; see "Generating a Broker SCA definition from a message set" on page 2967.

Use the Export an SCA Import or Export from a Broker SCA definition wizard to export a Broker SCA definition after you have generated it; see "Exporting an SCA Import or Export from a Broker SCA definition" on page 2945.

Related concepts:

"Interoperability with WebSphere Process Server" on page 2097
WebSphere Process Server is a business integration server that supports solutions based on service-oriented architecture (SOA). It uses Service Component Architecture (SCA) to present all business transactions in a service-oriented way in its runtime environment. WebSphere Message Broker can accept requests from WebSphere Process Server, and can also call service components on WebSphere Process Server.

"Starting from WebSphere Integration Developer" on page 2099
You can design SOA applications in WebSphere Integration Developer (IBM Integration Designer) that need to call services in WebSphere Message Broker, or be called by services in WebSphere Message Broker

SCA nodes

SCA nodes support interoperability between WebSphere Message Broker and WebSphere Process Server.

- The SCAInput and SCAReply nodes are analogous to the SOAPInput and SOAPReply nodes, and are used in a message flow that provides an SCA endpoint. These SCA nodes are used to construct a message flow which implements SCA.

  The SCAInput node listens for SCA inbound requests (from WebSphere Process Server to WebSphere Message Broker).

  The SCAReply node sends a reply back to the client that originated the SCA request. See "SCAInput node" on page 4707 and "SCAReply node" on page 4726.

- The SCAAsyncRequest and SCAAsyncResponse nodes are used to construct a pair of message flows that call a WebSphere Process Server service component asynchronously.

  The SCAAsyncRequest node sends an SCA outbound request to a service component that runs in WebSphere Process Server.

  The SCAAsyncResponse node receives the response from a business process that is running in WebSphere Process Server and to which the previous asynchronous request was made. The SCAAsyncResponse node can be in the same message flow or in a separate message flow.

  Calling a WebSphere Process Server service component asynchronously means that the SCAAsyncRequest node sends a request but does not wait for the associated response to be received, although it might wait for an acknowledgment of the request.

  The nodes are used as a pair, and correlate responses and requests. See "SCAAsyncRequest node" on page 4690 and "SCAAsyncResponse node" on page 4698.

- The SCARequest node is used to send a request to WebSphere Process Server. The node is configured using a Broker SCA Definition (`.outsca`) file; depending on the contents of the `.outsca` file, requests are either:
  - Two-way, synchronous; the node sends the request, then blocks until it receives a response, or the timeout period is exceeded.
  - One-way; the node sends a request only.

Related concepts:

"Service Component Architecture (SCA) overview" on page 2096
Service Component Architecture (SCA) is a specification that describes a model for building applications and systems using a service-oriented architecture (SOA).

"Interoperability with WebSphere Process Server" on page 2097
WebSphere Process Server is a business integration server that supports solutions based on service-oriented architecture (SOA). It uses Service Component Architecture (SCA) to present all business transactions in a service-oriented way in its runtime environment. WebSphere Message Broker can accept requests from WebSphere Process Server, and can also call service components on WebSphere Process Server.

"Message flows for Service Component Architecture (SCA)" on page 2105
Use the SCA nodes to develop message flows so that WebSphere Message Broker can interoperate with WebSphere Process Server. These nodes support scenarios in which service components that run on WebSphere Process Server either are called from WebSphere Message Broker or call a WebSphere Message Broker message flow.

Related reference:

"SCAAsyncRequest node" on page 4690
Use the SCAAsyncRequest node with the SCAAsyncResponse node to construct a pair of message flows that invoke a WebSphere Process Server service component asynchronously.

"SCAAsyncResponse node" on page 4698
Use the SCAAsyncResponse node with the SCAAsyncRequest node to construct a pair of message flows that start a component asynchronously.

"SCAInput node" on page 4707
Use the SCAInput node with the SCAReply node to process messages from WebSphere Process Server.

"SCARequest node" on page 4719
Use the SCARequest node to send a request to WebSphere Process Server. The node is a synchronous request and response node, and blocks after sending the request until the response is received. The node can also send one-way requests.

"SCAReply node" on page 4726
Use the SCAReply node to send a message from the broker to the originating client in response to a message received by a SCAInput node.

Using Broker SCA definitions to configure message flows

Use Broker SCA definitions to create and configure SCA nodes in a message flow.

A Broker SCA definition is a broker artifact that has a `.insca` or `.outsca` extension. The artifact contains the following information:

- Binding type and interface information which is present in SCA import or SCA export components
- WSDL that defines the interface
- XSD files that describe the message format

The Broker SCA definition files appear in a folder, named `Broker SCA Definitions`, in the message set project.

A `.insca` file contains an SCA Import component, and is used to configure SCAInput and SCAReply nodes.

A `.outsca` file contains an SCA Export component, and is used to configure SCAAsyncRequest, SCAAsyncResponse, and SCARequest nodes.

To create and configure SCA nodes, drag Broker SCA definitions, which are generated from SCA import or export files, into the message flow editor. This action automatically creates the appropriate nodes, and completes many of the configurable properties of the nodes. The nodes cannot be used if there is no Broker SCA definition with which to configure the nodes.

For Web Services binding, the data defines the service and the port name for the service within the WSDL that is invoked. It also defines an SCA endpoint address, extracted from the WSDL.

For MQ binding, the data defines the WebSphere MQ connection properties, the message binding format, the replyTo destination, and transaction and security information.

Related concepts:

"Working with Service Component Architecture (SCA)" on page 2095
Start here to find out how you can use SCA to allow interoperability with WebSphere Process Server Version 6.2.

Developing SCA applications for non-XML data

The SCA nodes allow non-XML data to be sent and received from WebSphere Process Server by using the WebSphere MQ binding. For example, create a message model from a COBOL copybook, and use that message model to parse messages received from WebSphere Process Server.

About this task

Both WebSphere Message Broker and WebSphere Integration Developer use XML schema to describe the *logical* format of the message to be parsed. Extra information in the form of XML schema annotations is used to describe any *physical* formats that you define for the messages. The physical format describes the precise appearance of the message bit stream during transmission.

In WebSphere Message Broker, using the MRM domain, you can model different physical representations using specific physical formats. These physical formats include *Custom Wire Format* (CWF) and *Tagged Delimited String Format* (TDS). Using the CWF physical format, you can model a message directly from a COBOL copybook.

"Interoperability with WebSphere Process Server" on page 2097 describes how the schema describing the logical format can be exchanged between WebSphere Message Broker and WebSphere Integration Developer. But the XML schema annotations that describe the physical format is not exchanged in those steps. This topic shows how to model the same message in both WebSphere Message Broker

and WebSphere Integration Developer. You create applications in WebSphere Message Broker and WebSphere Process Server that use a message model derived from a COBOL copybook. You can use a similar method for other message models derived from other non-XML data structures such as C Header files.

Starting from WebSphere Integration Developer:
Procedure

1. Import the COBOL copybook into your WebSphere Integration Developer module. See *Creating a business object from a source file* in the WebSphere Integration Developer Information Center.
2. Add the SCA Import or SCA Export to the assembly diagram that has an interface containing operations using message data types that were created from the COBOL copybook.
3. Export the Project Interchange (PI) file from WebSphere Integration Developer.
4. Import the PI file into WebSphere Message Broker using the Importer wizard.
5. Create the Inbound or Outbound SCA message flow from the SCA Definition file that the importer created.
6. Create a second message set and import the **same** COBOL copybook that was used to import into WebSphere Integration Developer.
7. Update the project references for your message flow project so that it also references the second message set.
8. In the message flow, select the **Input message parsing** tab in the SCAInput node, or **Response message parsing** tab in the SCARequest or SCAAsyncRequest node. Change the message domain from BLOB to MRM. Select the second message set that you created in the message set property. Select the relevant message type and message format.
9. Continue developing the rest of the application in WebSphere Message Broker.

Starting from WebSphere Message Broker:
Procedure

1. Create a message set from the COBOL copybook.
2. Generate the Broker SCA definition.
3. Export the SCA component.
4. In WebSphere Integration Developer, import the SCA import or export component and WSDL that have been exported from WebSphere Message Broker. Do not select the XSD files. See *Importing WSDL files* in WebSphere Integration Developer Information Center.
5. In WebSphere Integration Developer, select the import or export component and select the binding tab. For the input data format and output data format, click **Select** and step through the Data Format Transformation wizard; the information required by this wizard is not exported from WebSphere Message Broker.
6. In WebSphere Integration Developer, create the data types in the same module by importing the COBOL copybook. See *Creating a business object from a source file* in WebSphere Integration Developer Information Center.
7. Update the operations in the interface used by the SCA Import or Export so that they use the data types that you created.
8. Continue developing the rest of the application in WebSphere Integration Developer.

Related tasks:

"Importing SCA import or SCA export components" on page 2943
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to import SCA import or SCA export components from WebSphere Integration Developer. You must import an SCA import or SCA export into the workspace to provide a broker SCA definition for use in configuring the SCA nodes.

"Importing from COBOL copybooks" on page 2937
This topic describes how to create a new message definition from a COBOL data structure using the New Message Definition File wizard in the WebSphere Message Broker Toolkit.

"Generating a Broker SCA definition from a message set" on page 2967
WebSphere Message Broker creates a Broker SCA definition from existing message definitions.

"Exporting an SCA Import or Export from a Broker SCA definition" on page 2945
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to export SCA import or SCA export components. You must export an SCA import or SCA export if the Broker SCA definition is to be used by WebSphere Integration Developer.

Message flows for Service Component Architecture (SCA)

Use the SCA nodes to develop message flows so that WebSphere Message Broker can interoperate with WebSphere Process Server. These nodes support scenarios in which service components that run on WebSphere Process Server either are called from WebSphere Message Broker or call a WebSphere Message Broker message flow.

Message flows that are called by service components in WebSphere Process Server are SCA inbound message flows. Message flows that call service components in WebSphere Process Server are SCA outbound message flows.

Inbound flows use either a SCAInput node, or a pairing of the SCAInput node and SCAReply node to process messages from WebSphere Process Server.

Outbound flows use either a SCARequest node, or a pairing of the SCAAsyncRequest node and SCAAsyncResponse node, to call a service component in WebSphere Process Server.

Related concepts:

"SCA inbound message flows"
SCA inbound message flows are message flows that can be called by a service component running on WebSphere Process Server. By using SCAInput and SCAReply nodes in your message flow, you can process messages from WebSphere Process Server.

"SCA outbound message flows" on page 2107
SCA outbound message flows are message flows that call a service component on WebSphere Process Server. By using an SCARequest node, or a pair of SCAAsyncRequest and SCAAsyncResponse nodes, you can call a service component in WebSphere Process Server.

SCA inbound message flows:

SCA inbound message flows are message flows that can be called by a service component running on WebSphere Process Server. By using SCAInput and SCAReply nodes in your message flow, you can process messages from WebSphere Process Server.

When you want WebSphere Process Server to call services that are provided by WebSphere Message Broker, the SCAInput node acts as an SCA endpoint for the WebSphere Process Server service component. It accepts requests from WebSphere Process Server. The SCAReply node sends replies back to WebSphere Process Server.

Values for many of the properties of the SCAInput node that relate to the WebSphere Process Server binding are provided in the Broker SCA definition. You can generate this in two ways:

- From a message set.
- From an SCA import component imported from WebSphere Integration Developer.

You can put an instance of the SCAInput node into a message flow in either of the following ways:

- Drag a Broker SCA definition with an extension of `.insca` from a message set onto the message flow editor canvas. If the `.insca` file contains only one-way operations, dragging a `.insca` file onto the canvas creates a SCAInput node. Otherwise, a pair of SCAInput and SCAReply nodes is created. If you use this method, many of the values for the properties of the node or nodes are supplied by the Broker SCA definition.
- Drag an instance of the node from the node palette onto the canvas. You then configure the node by dragging a Broker SCA definition with an extension of `.insca` onto the node.

The values for properties that relate to the binding supported by the specific SCA component are automatically supplied to the node from the Broker SCA definition; you do not have to manually provide them.

Related concepts:

"Using Broker SCA definitions to configure message flows" on page 2102
Use Broker SCA definitions to create and configure SCA nodes in a message flow.

"SCA outbound message flows" on page 2107
SCA outbound message flows are message flows that call a service component on WebSphere Process Server. By using an SCARequest node, or a pair of SCAAsyncRequest and SCAAsyncResponse nodes, you can call a service component in WebSphere Process Server.

"Message flows for Service Component Architecture (SCA)" on page 2105
Use the SCA nodes to develop message flows so that WebSphere Message Broker can interoperate with WebSphere Process Server. These nodes support scenarios in which service components that run on WebSphere Process Server either are called from WebSphere Message Broker or call a WebSphere Message Broker message flow.

Related tasks:

"Generating a Broker SCA definition from a message set" on page 2967
WebSphere Message Broker creates a Broker SCA definition from existing message definitions.

"Importing SCA import or SCA export components" on page 2943
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to import SCA import or SCA export components from WebSphere Integration Developer. You must import an SCA import or SCA export into the workspace to provide a broker SCA definition for use in configuring the SCA nodes.

Related reference:

"SCAInput node" on page 4707
Use the SCAInput node with the SCAReply node to process messages from WebSphere Process Server.

"SCAReply node" on page 4726
Use the SCAReply node to send a message from the broker to the originating client in response to a message received by a SCAInput node.

"SCAAsyncRequest node" on page 4690
Use the SCAAsyncRequest node with the SCAAsyncResponse node to construct a pair of message flows that invoke a WebSphere Process Server service component asynchronously.

"SCAAsyncResponse node" on page 4698
Use the SCAAsyncResponse node with the SCAAsyncRequest node to construct a pair of message flows that start a component asynchronously.

"SCARequest node" on page 4719
Use the SCARequest node to send a request to WebSphere Process Server. The node is a synchronous request and response node, and blocks after sending the request until the response is received. The node can also send one-way requests.

SCA outbound message flows:

SCA outbound message flows are message flows that call a service component on WebSphere Process Server. By using an SCARequest node, or a pair of SCAAsyncRequest and SCAAsyncResponse nodes, you can call a service component in WebSphere Process Server.

Many of the properties of the SCAAsyncRequest and SCAAsyncResponse nodes are provided in the Broker SCA definition. You can generate this in two ways:

- From a message set.
- From an SCA export component imported from WebSphere Integration Developer.

The Broker SCA definition contains specific data that relates to the binding supported by the specific SCA component.

You create the appropriate nodes by dragging an outbound Broker SCA definition (`.outsca` file) onto the message flow editor from a message set project.

- If the `.outsca` file contains either a single request-response operation, or more than one operation (one-way or request-response), you are prompted to select the operation. If the chosen operation is request-response, you can also choose whether to invoke the service synchronously or asynchronously. Synchronous invocation is the default value, and creates a SCARequest node. Choosing the asynchronous option creates a pair of SCAAsyncRequest and SCAAsyncResponse nodes.
- If the chosen operation is one-way, synchronous invocation is the only option, and a SCARequest node is created.

Synchronous requests:

Use a SCARequest node when:

- WebSphere Message Broker needs to synchronously invoke a request-response operation.
- WebSphere Message Broker needs to invoke a one-way operation in an application running on WebSphere Process Server.

If the request is request-response, the node sends the request, then blocks until it receives a response, or the timeout period is exceeded. If the timeout period is exceeded, the message received on the input terminal of the SCARequest is propagated to the Failure terminal.

If the request is one-way, the node sends a request only. The message received on the input terminal of the SCARequest node is propagated to the Out terminal.

Asynchronous requests: When WebSphere Message Broker needs to make an asynchronous call to a service component that is provided by WebSphere Process Server, use a message flow, or flows, that contains a pair of SCAAsyncRequest and SCAAsyncResponse nodes. The SCAAsyncRequest node sends a request to a service component running on WebSphere Process Server. The SCAAsyncResponse node receives the response from WebSphere Process Server to a previously made asynchronous request from an SCAAsyncRequest node. Responses are correlated against the original requests.

The SCAAsyncRequest node requests sends a request to a service component running on WebSphere Process Server. The SCAAsyncResponse node receives the response from WebSphere Process Server to a previously made asynchronous request from an SCAAsyncRequest node. Responses are correlated against the original requests.

The SCAAsyncResponse node can be in the same message flow as the SCAAsyncRequest node which makes the request, or it can be in a separate message flow; it must, however, be in the same execution group as the SCAAsyncRequest node.

Related concepts:

"Using Broker SCA definitions to configure message flows" on page 2102
Use Broker SCA definitions to create and configure SCA nodes in a message flow.

"Message flows for Service Component Architecture (SCA)" on page 2105
Use the SCA nodes to develop message flows so that WebSphere Message Broker can interoperate with WebSphere Process Server. These nodes support scenarios in which service components that run on WebSphere Process Server either are called from WebSphere Message Broker or call a WebSphere Message Broker message flow.

"SCA inbound message flows" on page 2105
SCA inbound message flows are message flows that can be called by a service component running on WebSphere Process Server. By using SCAInput and SCAReply nodes in your message flow, you can process messages from WebSphere Process Server.

Related tasks:

"Generating a Broker SCA definition from a message set" on page 2967
WebSphere Message Broker creates a Broker SCA definition from existing message definitions.

"Importing SCA import or SCA export components" on page 2943
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to import SCA import or SCA export components from WebSphere Integration Developer. You must import an SCA import or SCA export into the workspace to provide a broker SCA definition for use in configuring the SCA nodes.

Related reference:

"SCAAsyncRequest node" on page 4690
Use the SCAAsyncRequest node with the SCAAsyncResponse node to construct a pair of message flows that invoke a WebSphere Process Server service component asynchronously.

"SCAAsyncResponse node" on page 4698
Use the SCAAsyncResponse node with the SCAAsyncRequest node to construct a pair of message flows that start a component asynchronously.

Working with databases

Create and configure databases to use with your message flow applications.

About this task

- "Databases overview"
- "Accessing databases from message flows" on page 2112
- "Accessing databases from ESQL" on page 2115
- "Extended database support" on page 2117
- "Event-based database integration" on page 2118
- "Configuring a DatabaseInput node" on page 2120

Related concepts:

"Developing user-defined extensions" on page 2970
A user-defined extension is a component that you design and implement to extend the function of WebSphere Message Broker.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Routing messages" on page 2209
Route messages through your message flow or the broker by using one or more of the techniques described in this section.

"Transforming and enriching messages" on page 2227
Transform and enrich messages by using one or more of the techniques described in this section.

Databases overview

Brokers can access *user databases* to manipulate your business data. If you have user databases, you must configure them before you can access them from your message flow.

WebSphere Message Broker supports the databases that are listed in "Supported databases" on page 3591 for user databases. If you configure your message flows to access user databases, you cannot access some of the data types that are supported by these databases. The supported data types are defined in "Data types of values from external databases" on page 5288.

User databases

User databases are the databases in which you store the business data that is processed by message flow applications. Additional local and remote database managers might also be supported for your computer. For more information, see "Supported databases" on page 3591 and "Database locations" on page 3595.

You must set up connections to the user databases so that the broker can access the databases on behalf of its deployed message flows. Both ODBC and JDBC

connections are supported; some restrictions apply on some platforms, as described in the topics in this section. ODBC drivers are supplied and installed with WebSphere Message Broker. JDBC drivers are not supplied by WebSphere Message Broker; you must obtain these files from your database vendor. Supported drivers are listed in "Supported databases" on page 3591.

You can use the **mqsicvp** command as an ODBC test tool; see "Enabling ODBC connections to the databases" on page 668 for further information.

For information about connections to user databases, see "User database connections."

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

"User database connections"
User databases contain business data that is written and accessed by deployed message flows. You must create connections from the broker to the user database by using ODBC or JDBC.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Creating the user databases" on page 661
If your message flows create, update, retrieve, or delete application and business data in one or more user databases, create the databases before you deploy the message flows to a broker.

"Enabling ODBC connections to the databases" on page 668
Set up the resources and environment that the broker requires for Open Database Connectivity (ODBC) connections to user databases on distributed systems.

"Enabling JDBC connections to the databases" on page 683
Configure connections to a user database through a JDBCProvider service.

Related reference:

"Supported databases" on page 3591
You can optionally configure databases to contain data that is accessed by your message flows. Databases from IBM and other suppliers are supported at specific versions on supported operating systems.

"Data types of values from external databases" on page 5288
How database data types are implicitly cast to ESQL data types.

User database connections:

User databases contain business data that is written and accessed by deployed message flows. You must create connections from the broker to the user database by using ODBC or JDBC.

ODBC connections to databases are managed internally by the broker, and therefore any configurable connection pooling options that are available on the ODBC driver should not be used.

The broker requires a database connection for each data source name (DSN) that is referenced in the message flow, even if different DSNs resolve to the same physical database.

The number of connections to a user database that a broker requires depends on the actions of the message flows that access the database. For each message flow thread, a broker that accesses a user database makes one connection for each data source name (DSN). If a different node on the same thread uses the same DSN, the same connection is used, unless a different transaction mode is used, in which case another connection is required. For further information about transactions, see "Database connections for coordinated message flows" on page 4235.

When you start a broker, and while it is running, it opens connections to WebSphere MQ queues and to databases. The broker makes the connections when it needs to use them, and they remain open until one of the following events occurs:

The message flow has been idle for 1 minute

The message flow is stopped

The broker is stopped

If the message flow contains a DatabaseInput node, at least one database connection remains open while the message flow is running.

Database connections from message flows that are not globally coordinated are released when a flow has no work. For example, a connection is released if the message flow input queue has no messages, and the database has not been accessed for 1 minute.

Linux UNIX Windows On Linux, UNIX, and Windows systems, to avoid breaking global coordination, database connections are released only for message flows that are not globally coordinated.

z/OS On z/OS, database connections for globally coordinated message flows are also released if the database has not been accessed for 1 minute.

To change the default time of 1 minute after which a database connection for an idle message flow is released, use the following command:

```
mqsichangeproperties myBroker -e myExecutionGroup -o ComIbmDatabaseConnectionManager -n maxConnect
```

or the following command, to change the default time for all execution groups:

```
mqsichangeproperties myBroker -o ComIbmDatabaseConnectionManager -n maxConnectionAge -v newValue
```

where *maxConnectionAge* is specified in seconds. If *maxConnectionAge* is set to option -1, database connections are never released until the execution group or broker is stopped.

If you are using DB2 for your database, the default action is to limit the number of concurrent connections to a database to the value of the **maxappls** configuration parameter. The default for **maxappls** is 40. If you believe that the connections that the broker might require exceeds the value for **maxappls**, increase this parameter and the associated **maxagents** parameter to new values based on your calculations.

For z/OS, the number of connections does not change when you use ODBC CAF (Call Attachment Facility) connections or RRSAF (Recoverable Resource Services Attachment Facility). For more information on the number of connections required, see You do not know how many database connections a broker requires.

If you are using another database, check the database documentation for information about connections and the limits or restrictions that might apply.

When a message flow is idle, the execution group periodically releases database connections. Therefore, connections held by the broker reflect its current use of these resources. This situation allows the broker to respond when a database quiesces, if the database manager supports quiescing. Not all databases support the quiesce function, and not all databases quiesce in the same way. Check your database documentation for information about database quiescing.

Related tasks:

"Enabling ODBC connections to the databases" on page 668
Set up the resources and environment that the broker requires for Open Database Connectivity (ODBC) connections to user databases on distributed systems.

"Accessing databases from message flows"
Create and configure message flows to access user databases.

"Configuring transactionality for message flows" on page 1290
A message flow runs in a single transaction, which is started when data is received by an input node, and can be committed or rolled back when all processing has completed.

Related reference:

"Supported databases" on page 3591
You can optionally configure databases to contain data that is accessed by your message flows. Databases from IBM and other suppliers are supported at specific versions on supported operating systems.

"Support for Unicode and DBCS data in databases" on page 3668
You can manipulate Unicode Standard version 3.0 data, in suitably configured databases, using ESQL, in nodes that access databases by ODBC. The broker does not support DBCS-only columns in tables that are defined in databases.

"Database connections for coordinated message flows" on page 4235
When you configure a message flow to access a database, the broker establishes a connection to that database based on the ODBC DSN.

"Listing database connections that the broker holds" on page 1002
The broker does not provide an interface that you can use to list the connections that it has to a database. You must use the facilities of the database suppliers to list connections.

"Quiescing a database" on page 1002
The broker and database exhibit specific behaviors when you quiesce the database.

"WebSphere MQ connections" on page 4222
The number of WebSphere MQ connections a broker requires to its queue manager depends on the actions of the message flows that access the WebSphere MQ resource.

Accessing databases from message flows

Create and configure message flows to access user databases.

Before you begin

Before you start:

Complete the following tasks:

- "Creating a message flow" on page 1431
- "Working with databases" on page 2109

Read the following concept topic:
- "Message flow nodes" on page 1024

Check which databases are supported on which platform, and if any restrictions apply:
- "Supported databases" on page 3591

About this task

You can access a database from a message flow in two ways:

- You can design a message flow that responds to events generated by the database.
- After a flow has already started, you can access the database to read or update information in it. Information from the database can be used to enhance or influence the operation of the message flow.

You can access a database from a message flow by using the following nodes:
- Compute
- Database
- DatabaseInput
- DatabaseRetrieve
- DatabaseRoute
- DataDelete
- DataInsert
- DataUpdate
- Filter
- JavaCompute
- Mapping
- Warehouse

For more details about these nodes, and how to configure them in message flows, see "Built-in nodes" on page 4293.

If you want the actions that the message flow takes against the database to be coordinated with other actions, configure the message flow to support global coordination of transactions. For information about how to complete this task, see "Configuring transactionality for message flows" on page 1290.

To access a database from a message flow:

Procedure

1. Identify the database that you want to access. You can access an existing database, or create a new database for this purpose. See "Data sources on z/OS" on page 4014 for more information about what to call a z/OS user database.

   Create the database you want, or ask your database administrator to create it for you. If you are using DB2 or Oracle databases, some of the samples programs include basic instructions for creating databases, which you can use as a guide. For information about which samples include databases, see "Creating the user databases" on page 661.
2. Define a connection to the data source name (DSN) to enable a connection to the database, if one does not exist:

- Define a JDBC connection if you want to interact with a database directly from a Java application. You can code Java in both a JavaCompute node and in a Java user-defined node.

  For more information, see “Enabling JDBC connections to the databases” on page 683.
- Define an ODBC connection if you want to interact with a database in a node that supports ESQL, including a JavaCompute node in which you use the MbSQLStatement interface.

  For more information, see “Enabling ODBC connections to the databases” on page 668.

3. Authorize the broker to access the database.

   Access to a user database from within a message flow is controlled by user ID and password.

   - Linux UNIX Windows Use the **mqsisetdbparms** command to specify a user ID and password for a specific database, or to set up a default user ID and password.
   - z/OS Use the BIPSDBP JCL in the customization data set `<hlq>.SBIPPROC` to customize the **mqsisetdbparms** command to specify a user ID and password for a specific database, or to set up a default user ID and password.

What to do next

The following samples access databases from message flows:

- Message Routing
- Data Warehouse
- Error Handler
- Airline Reservations

Message Routing and Data Warehouse use Compute nodes to access the database. Error Handler uses Database nodes to access the database, and Airline Reservations uses both Compute and Database nodes.

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Related tasks:

“Working with databases” on page 2109
Create and configure databases to use with your message flow applications.

“Enabling JDBC connections to the databases” on page 683
Configure connections to a user database through a JDBCProvider service.

“Enabling ODBC connections to the databases” on page 668
Set up the resources and environment that the broker requires for Open Database Connectivity (ODBC) connections to user databases on distributed systems.

“Interaction with databases using ESQL” on page 2487
Use ESQL statements and functions to read from, write to, and modify databases from your message flows.

“Interacting with databases by using the JavaCompute node” on page 2661
Access databases from Java code included in the JavaCompute node.

"Extending the capability of a Java message processing or output node" on page 3069
Within a message processing or output node, you can add extended functions to your Java node.

"Adding database definitions to the WebSphere Message Broker Toolkit" on page 2278
Use the New Database Definition File wizard to add database definitions to the WebSphere Message Broker Toolkit.

"Configuring transactionality for message flows" on page 1290
A message flow runs in a single transaction, which is started when data is received by an input node, and can be committed or rolled back when all processing has completed.

Related reference:

"Supported databases" on page 3591
You can optionally configure databases to contain data that is accessed by your message flows. Databases from IBM and other suppliers are supported at specific versions on supported operating systems.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"Support for Unicode and DBCS data in databases" on page 3668
You can manipulate Unicode Standard version 3.0 data, in suitably configured databases, using ESQL, in nodes that access databases by ODBC. The broker does not support DBCS-only columns in tables that are defined in databases.

"Data sources on z/OS" on page 4014

Accessing databases from ESQL

Configure your broker and your database to support connections from message flows.

Before you begin

Before you start:

- Create the broker.

You must configure both your broker and your databases to support read, write, and update operations in your message flows.

About this task

- Set the `Data Source` property of each node to the name (that is, the ODBC DSN) of the database that you want to access.
- Configure the broker to be able to connect to the database:
  - Create ODBC data source connections on the system on which the broker is running.
  - Define a user ID and password to be used by the broker to connect to the database:

- If you have used the **mqsisetdbparms** command, or submitted the JCL member BIPSDBP in the customization data set <hlq>.SBIPPROC on z/OS, to set a user ID and password for a particular database, the broker uses these values to connect to the database.
- If you have not set up a specific user ID and password for this database, the broker users the default ID and password set by using the **mqsisetdbparms** command, or the JCL file BIPSDBP.
- If you have not set up a default user ID and password:
  - On Windows, the service user ID and password are used to connect to the database.
  - On z/OS, the broker started task ID is used. The schema used is the one defined for a specific DSN or a default DSN set up by using the **mqsisetdbparms** command. If neither exist, the value of CURRENTSQLID in the BIPDSNAO file is used. If CURRENTSQLID is not set, the schema defaults to the started task user ID for the broker.
  - On other platforms, connection to the database fails.

- Set up the authorization for the user ID to access the database by using the administration facilities provided by the database vendor. If you do not do so, the broker generates an error when the message flow runs.
- All databases accessed from the same node must have the same ODBC functions as the database specified on the Data Source property on that node. This requirement is always satisfied if the databases are of the same type (for example, DB2 or Oracle), at the same release level (for example, release 9.1), and on the same platform. Other database combinations might have the same ODBC functions. If a node tries to access a database that does not have the same ODBC functions as the database specified on the Data Source property on that node, the broker generates an error message.

  You can use the **mqsicvp** command as an ODBC test tool. This test tool can be run against two databases simultaneously, and tells you whether those two datasources are eligible to be used together in the same node.

  The test tool is also useful in displaying any limitations there might be in your datasource, prior to constructing your ESQL; see "Enabling ODBC connections to the databases" on page 668 for further information.
- With a single SELECT FROM clause, you can access only tables that exist in a single database.
- If you access database columns that have names composed of only numeric characters, you must enclose the names in double quotation marks; for example, "0001". Because of this restriction, you cannot use a SELECT * statement, which returns the names without quotation marks; the names are therefore invalid and the broker raises an exception.

For details of the ESQL statements and functions that you can use to access databases, see "Interaction with databases using ESQL" on page 2487.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"User database connections" on page 2110
User databases contain business data that is written and accessed by deployed message flows. You must create connections from the broker to the user database by using ODBC or JDBC.

Related tasks:

"Enabling ODBC connections to the databases" on page 668
Set up the resources and environment that the broker requires for Open Database Connectivity (ODBC) connections to user databases on distributed systems.

"Working with databases" on page 2109
Create and configure databases to use with your message flow applications.

"Configuring transactionality for message flows" on page 1290
A message flow runs in a single transaction, which is started when data is received by an input node, and can be committed or rolled back when all processing has completed.

Related reference:

"Supported databases" on page 3591
You can optionally configure databases to contain data that is accessed by your message flows. Databases from IBM and other suppliers are supported at specific versions on supported operating systems.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"Support for Unicode and DBCS data in databases" on page 3668
You can manipulate Unicode Standard version 3.0 data, in suitably configured databases, using ESQL, in nodes that access databases by ODBC. The broker does not support DBCS-only columns in tables that are defined in databases.

Extended database support

Use WebSphere Message Broker ODBC Database Extender (IE02) to allow WebSphere Message Broker to support two ODBC database driver managers in the same process space.

Database supported:

solidDB
: solidDB is an in-memory database product from IBM. solidDB V6.5 is supported; for more information see IBM solidDB product family.

See "Supported databases" on page 3591 for a full list of supported operating systems on which the WebSphere Message Broker supports solidDB.

Related tasks:

"Connecting to a database from Linux and UNIX systems using the WebSphere Message Broker ODBC Database Extender (IE02)" on page 682
WebSphere Message Broker ODBC Database Extender encapsulates the unixODBC driver manager and this topic describes how you set up and configure the broker to use it.

Related reference:

"Supported databases" on page 3591
You can optionally configure databases to contain data that is accessed by your message flows. Databases from IBM and other suppliers are supported at specific versions on supported operating systems.

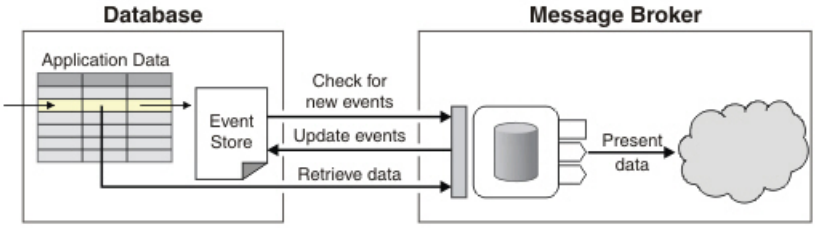
"Sample WebSphere Message Broker ODBC Database Extender (IE02) configuration files" on page 3596
How you use WebSphere Message Broker ODBC Database Extender (IE02) to load the correct data source driver.

"**mqsicvp** command" on page 3857
Use the **mqsicvp** command to perform verification tests on a broker, or to verify ODBC connections.

Event-based database integration

Use a DatabaseInput node to respond to events in a database. For example, the broker can keep an external system synchronized with a database by sending updates to the target system whenever data is changed in the database.

The database must record the fact that data has changed in an *event store*, which is typically a database table. The event store is not the same as the application data. The following diagram shows the interaction between the database, event store, and the broker.

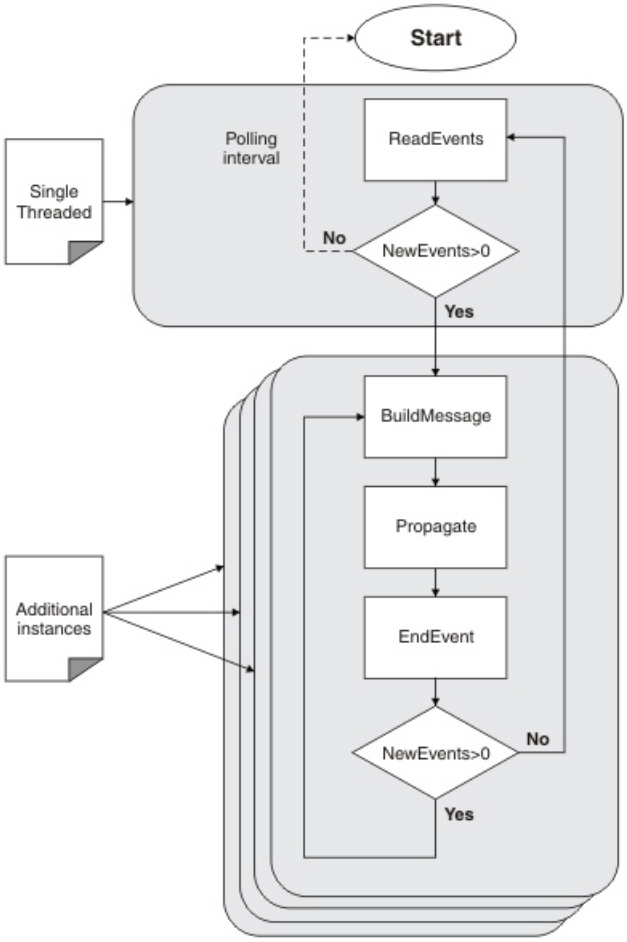


1. A database application changes a database table.
2. The database management system (DBMS) records the change in the event store.
3. The broker polls the event store after the interval specified in the `Polling interval` configuration setting.
4. WebSphere Message Broker retrieves the new or changed data, and updates the event store, so that the data is processed only once.
5. WebSphere Message Broker processes the data, and ultimately presents it to the target application, for example SAP, web services, or CICS Transaction Server for z/OS. The data can be presented in a different logical and physical format, if required.

Implementing: Implementing this scenario involves the following steps:

1. Configuring the database to record events.
2. Determining the format in which the target system must receive the data from these new events.
3. Configuring the broker to detect these events by using the DatabaseInput node. To configure the DatabaseInput node, see "Configuring a DatabaseInput node" on page 2120.
4. Configuring the rest of the message flow to present the data to your target system in the correct format.

The DatabaseInput node: The following diagram shows how the DatabaseInput node works.



When the process starts, ReadEvents checks the event store for new events, which are then used by BuildMessage to build the message. This message is propagated to the message flow and then EndEvent updates the event store to ensure that the event cannot be processed again. When all events have been processed, the broker calls ReadEvents to retrieve any events that have been added since the previous check. If the event store is empty, the broker waits until the polling interval has expired, and then calls ReadEvents again. To avoid contention, the check of the event store is single-threaded.

For each event that is read by ReadEvents, BuildMessage builds the message that is propagated to the message flow. Building the message typically uses the event data to look up data in the application table. The data from the application table is then used to construct the message. When BuildMessage ends, the message is automatically propagated to the message flow. When the message is propagated, the broker starts any downstream nodes that are required to process the message.

After the message has been propagated to the message flow, EndEvent updates the event store to ensure that the event that has just been processed cannot be processed again.

The detailed operation of ReadEvents, BuildMessage, and EndEvent are controlled by ESQL code. The DatabaseInput node contains an ESQL module with sample

code and comments, which you must modify to suit your requirements. For information about modifying the ESQL, see "Configuring a DatabaseInput node."

Transactions and Scaling: The processes that are completed by the DatabaseInput node are split across separate transactions. A new transaction is started when ReadEvents starts. When ReadEvents ends, this transaction is committed and new events are marked for processing. By committing this transaction, any locks put on the database by ESQL code that is run from ReadEvents are released. Then, for each event received by BuildMessage, a new transaction is started. This new transaction is committed after EndEvent finishes.

To scale a DatabaseInput node for many events, change the `Additional instances` property on the **Instances** tab from its default value of `0` to the number of instances that you require. If you are using additional instances, the database must be configured so that multiple applications can read different rows from the application table at the same time. ReadEvents always runs in single-threaded mode to avoid database contention, even if you use additional instances. To improve performance, ReadEvents can read multiple events each time it runs, and these events can be processed at the same time by multiple instances of BuildMessage. The event store must have a primary key, which ReadEvents uses to identify events that are currently being processed. You do not have to write the ESQL in ReadEvents to filter out events that are currently being processed by the message flow.

Related concepts:

"Event tables" on page 2126
An event table stores information about changes to application tables.

"Databases overview" on page 2109
Brokers can access *user databases* to manipulate your business data. If you have user databases, you must configure them before you can access them from your message flow.

Related tasks:

"Responding to database updates" on page 2123
Implement a message flow that responds to database updates, and presents the data to another application.

"Working with databases" on page 2109
Create and configure databases to use with your message flow applications.

"Configuring a DatabaseInput node"
Create and configure message flows that respond to events in a database.

Related reference:

"DatabaseInput node" on page 4360
Use the DatabaseInput node to detect events recorded in a database, and to retrieve the data affected by those events.

"New message definition file wizard: Create a new message definition file from a database definition" on page 6365
You can create a new message definition from a database definition file (`.dbm`) by using the New Message Definition File wizard in the WebSphere Message Broker Toolkit.

Configuring a DatabaseInput node:

Create and configure message flows that respond to events in a database.

Before you begin

Before you start:

Read the following topics:
- "Event-based database integration" on page 2118
- "Responding to database updates" on page 2123

Check which databases are supported on which platform, and if any restrictions apply:
- "Supported databases" on page 3591

Ensure that your database is configured to record events (uses an event table), and that you know how to query those events. For information about event tables and triggers, see the DatabaseInput Node sample. You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

z/OS If you use DB2 on z/OS, your user ID (or your user group) requires permission to perform a SELECT on SYSIBM.SYSJAROBJECTS.

Complete the following tasks:
- Add a database definition to the toolkit

About this task

When you drag a DatabaseInput node onto the canvas, WebSphere Message Broker creates an ESQL module that contains boilerplate text. To configure the DatabaseInput node, modify the statements in that module to suit your requirements.

When you double-click the node to modify the ESQL code, the editor displays the Database Event Design tab for the module. Complete the mandatory fields and then click **Generate query**. To view or modify the code, click the Source tab. Code that has been generated is clearly marked by color-coded `--@!{` and `--@!}` comments. Any changes that you make within these comments are lost if you regenerate the code.

Procedure

1. In the WebSphere Message Broker Toolkit, drag a DatabaseInput node onto the canvas, and double-click the node. The Database Event Design tab is displayed. Ensure that the correct module is selected.
2. Complete the Event Table section.
   a. Optional: Complete the `Database schema` property. Leave it blank to use the default runtime schema.
   b. Complete the `Table` property. This property represents the database table used as your event store.
   c. Complete the `Primary key` property. This property represents the primary key of the database table used as the event store.
   d. Complete the `Foreign key to application table` property. This property represents the column in the event table that references the row in the

application table containing the changed data to be processed by the DatabaseInput node. This is typically the primary key of the application table.

e. Optional: Complete the `Status column` property. This property represents the name of a column, if you update a column in the event table to indicate that the event has been processed. Leave blank if you delete events from the event table after processing.

f. Optional: Complete the `New event status value` property. This property represents the value written to the status column when the event is first added. Enclose character values in single quotation marks, for example 'Y'. Enter numbers without quotation marks. For a null value, enter NULL. Check the trigger setting in your database for appropriate values.

g. Optional: Complete the `Processed event status value` property. This property represents the value written to the status column after the event has been processed. Enclose character values in single quotation marks, for example 'Y'. Enter numbers without quotation marks. For a null value, enter NULL. Check the trigger setting in your database for appropriate values.

3. Complete the Application Table section.
   a. Complete the `Table` property. This property represents the table that includes the changed data to be processed by the DatabaseInput node.
   b. Complete the `Primary key` property. This property represents the primary key of the database table used as the application table.
   c. Complete the `Output message element` property. This property represents the output message that will be propagated to the flow.
4. Click **Generate query**.
5. Optional: Click the Source tab to view the code, or add customized code.
6. On the Basic tab of the DatabaseInput node, specify the data source. This data source is the ODBC data source name of the database that contains the tables that you refer to in the ESQL module. See "Enabling ODBC connections to the databases" on page 668.
7. On the Basic tab, ensure that the `ESQL module` property refers to the correct module.
8. Optionally, change values on the other tabs of the node.
9. Configure the rest of the flow to use the message from this node.

What to do next

Configure your target system to receive the message.

Changing the default color of auto-generated text:
Procedure

1. Click **Window** > **Preferences**.
2. In the tree on the left, navigate to **Broker Development** > **ESQL** > **ESQL Editor**.
3. On the Colors tab, select **Auto-generated**, and select the color.

Related concepts:

"Event tables" on page 2126
An event table stores information about changes to application tables.

Related tasks:

"Responding to database updates" on page 2123
Implement a message flow that responds to database updates, and presents the

data to another application.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

"Working with databases" on page 2109
Create and configure databases to use with your message flow applications.

"Interaction with databases using ESQL" on page 2487
Use ESQL statements and functions to read from, write to, and modify databases from your message flows.

"Selecting data from database columns" on page 2491
You can configure a Compute, Filter, or Database node to select data from database columns and include it in an output message.

"Accessing multiple database tables" on page 2496
You can refer to multiple tables that you have created in the same database. Use the FROM clause on the SELECT statement to join the data from the two tables.

"Creating a message definition file from an existing resource" on page 2867
You must create a message definition file before you can create the message model objects. The message definition file contains the logical and physical model definitions of the objects in XML schema form.

"Adding database definitions to the WebSphere Message Broker Toolkit" on page 2278
Use the New Database Definition File wizard to add database definitions to the WebSphere Message Broker Toolkit.

Related reference:

"Supported databases" on page 3591
You can optionally configure databases to contain data that is accessed by your message flows. Databases from IBM and other suppliers are supported at specific versions on supported operating systems.

"DatabaseInput node" on page 4360
Use the DatabaseInput node to detect events recorded in a database, and to retrieve the data affected by those events.

Responding to database updates:

Implement a message flow that responds to database updates, and presents the data to another application.

Before you begin

Before you start:

- Create an event table (a database table that serves as a transient store for event data).
- Create a trigger on the application data table. The trigger inserts a row into the event store whenever the application data is changed.
- Configure the broker runtime to connect to the database; see "Enabling ODBC connections to the databases" on page 668.

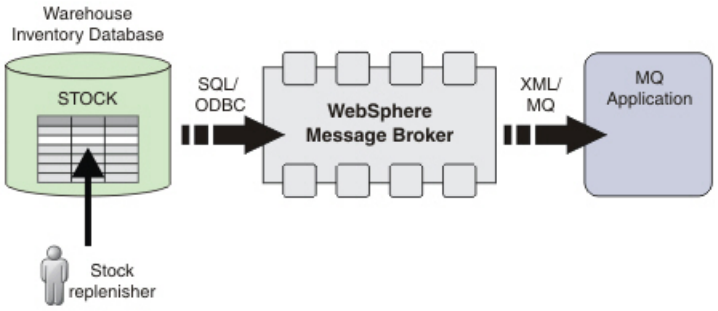
For information about event tables and triggers, see the DatabaseInput Node sample. You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

z/OS If you use DB2 on z/OS, your user ID (or your user group) requires permission to perform a SELECT on SYSIBM.SYSJAROBJECTS.

You do not need experience of ESQL to complete this task.

About this task

Scenario: A retail company uses a relational database to manage its stock inventory. Since a recent acquisition, a new set of applications based on XML and WebSphere MQ are added to the environment. The applications notify interested parties of any changes to the stock levels. The applications have a predefined XSD schema model that describes the input message.



WebSphere Message Broker is used to respond to database updates, and to notify the WebSphere MQ application of these changes.

1. A DatabaseInput node retrieves the data.
2. A transformation node, such as a Compute node or a Mapping node, transforms the data to the target format.
3. An output or request node, such as an MQOutput node, sends the transformed message to the target system.

You will complete the following actions:

1. "Discover the database model"
2. "Create a new message model for the database input" on page 2125
3. "Create the message flow" on page 2125
4. "Test the flow" on page 2126

Discover the database model:

Create a `.dbm` file that you will use to create the message model. You create a data design project, and use a wizard to give WebSphere Message Broker details of your database event store and data table.

Procedure

1. Click **File** > **New** > **DataBase Definition**.
2. Click **New** to create a new data design project, or select an existing data design project from the drop down list.
3. Select the appropriate database type and version, and then click **Next**.
4. Select an existing JDBC connection, or create a connection to your database. If you create a connection, test the connection.
5. Select the database schema that you will use to create the message definition.

6. Select the database elements that you need for the model. You require `Tables` and `Triggers`. The data model is created, and you can see details of the database tables that are described in the chosen schema.

Create a new message model for the database input:

Create a new message schema model file from the discovered data definition if you require a model of the data structure that the database input will present. You need a model if you want to graphically map this input. The model also enables content assistance auto-completion of paths in the ESQL editor if you are transforming the data in ESQL.

Procedure

1. Click **File** > **New** > **Message model**
2. In the **Other** section, select `Database record`, and then click **Next**.
3. Select `Create an XML schema file from a database definition`, and then click **Next**.
4. Navigate to and select the Database definition `.dbm` file that you created during discovery, and then click **Next**.
5. Ensure that the database tables that will be used are selected.
6. Click **Next** and then **Finish**.

Results

The New Message model wizard creates an XML schema message model file in your selected location.

Create the message flow:

Create and configure a flow that consists of a DatabaseInput node, a Mapping node, and an MQOutput node.

You will use the schema file that describes the input message to create a message definition file.

Procedure

1. Create a message flow project that references both the data design project, and the message set project, that you created earlier.
2. Create a message flow, and drag a DatabaseInput node onto the canvas.
3. Configure the node as follows:
   a. Set the **`Data source`** to the ODBC connection that you created earlier.
   b. Follow the instructions in "Configuring a DatabaseInput node" on page 2120 to configure the ESQL procedures to provide details of the event store and application data.
4. To enable graphical mapping from the database data to the output message format, in addition to Creating the new message model for the database input as above, you also require a message model for the target message. This can be a DFDL or XML schema message model as appropriate to your scenario.
5. Drag a Mapping node onto the flow, and configure it.
   a. Set the map source to the message that you defined in "Create a new message model for the database input."
   b. Set the map target to the `TARGET` message that you just defined.

You can use other methods to transform the message; see "Transforming and enriching messages" on page 2227.

6. Drag an MQOutput node onto the canvas, and set the queue manager name and queue name.

Test the flow:

Use the debugger to test the flow.

Procedure

1. Start the debugger, and then add breakpoints to the flow.
2. Deploy the flow.
3. Change the data source, for example by adding a new row. You can change data from within the WebSphere Message Broker Toolkit. Under **Data Source Explorer**, right-click the table and choose **Data** > **Edit**.
4. Use the debugger, and WebSphere Message Broker Explorer, to check that the flow is working correctly.

Related concepts:

"Event-based database integration" on page 2118
Use a DatabaseInput node to respond to events in a database. For example, the broker can keep an external system synchronized with a database by sending updates to the target system whenever data is changed in the database.

"Event tables"
An event table stores information about changes to application tables.

Related tasks:

"Starting the flow debugger" on page 3160
To start the flow debugger, you must attach it to an execution group. When the flow debugger is started, you can introduce test messages to your message flow.

Event tables:

An event table stores information about changes to application tables.

The event table is a database table created by the user, generally within the same schema as the application table for which it stores events. The event table describes the type of change made to an application table, and also contains an identifier for the changed row.

To populate an event table, one or more *triggers* must be created. A trigger is a database construct that can run an SQL script when a predefined action occurs. For example, a trigger can insert a row in the event table when an update in the application table occurs.

For examples of triggers and event tables, see the DatabaseInput Node sample. You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

The following table shows some typical columns in an event table, and the reasons for including them.

Column name	Column function	Example value
EVENT_ID	Required. The primary key, which identifies the event being processed.	1
OBJECT_KEY	Required. The identifying element of the changed row in the application table, typically the element of the row in the primary key column.	cust1
OBJECT_VERB	Optional. The change performed, typically one of CREATE, UPDATE, or DELETE. This event is used to distinguish a DELETE event, where the application table contains no row to retrieve when the message for the flow is built.	CREATE
OBJECT_NAME	Optional. The name of the application table that has changed. This column is required if the DatabaseInput node is being used to support updates to more than one application table.	customer
EVENT_PRIORITY	Optional. The priority of the event. For example, you can ensure that high value transactions are computed first.	1
EVENT_TIME	Optional. The time at which the operation was performed. Generally used for logging or performance monitoring of the flow.	2010-10-19T17:10:00
EVENT_STATUS	Optional. Used to determine if the event has already been processed. Required if the events are not to be deleted or archived after processing.	0
EVENT_COMMENT	Optional. Free-form field, for example, it can be used to store the outcome of the message processing if the event was not deleted after processing.	Processed with exceptions

The column names are examples only. You can use other names. If you have a high-throughput application table, a single row might be changed multiple times between retrieving events from the event table. In this case, only the details of the latest change are picked up by the flow. If a record of intermediate changes is required, include more details in the event table. Also ensure that your event table has enough information about events generated by DELETE operations. Here, because the row in the application table no longer exists, all information required to successfully process the event must be present in the event table.

For example, if a new customer with primary key `cust1` is created in the application table, a row is added to the event table:

EVENT_ID	OBJECT_KEY	OBJECT_VERB
1	cust1	Create

The DatabaseInput node responds to the change, and processes the new row in a message flow.

Processing options on completion

When the message flow has processed an event, the flow can handle the event in the following ways:

- Delete the event. Use this option if you do not want to store the event for future reference.
- Update the status column. Use this option if you want to keep a record of processed events. Your event table must have a status column.
- Archive the event to a separate event table. Use this option if you want to keep a record of events while keeping the event table to a minimal size.

Related concepts:

"Event-based database integration" on page 2118
Use a DatabaseInput node to respond to events in a database. For example, the broker can keep an external system synchronized with a database by sending updates to the target system whenever data is changed in the database.

Related tasks:

"Configuring a DatabaseInput node" on page 2120
Create and configure message flows that respond to events in a database.

"Responding to database updates" on page 2123
Implement a message flow that responds to database updates, and presents the data to another application.

Related reference:

"DatabaseInput node" on page 4360
Use the DatabaseInput node to detect events recorded in a database, and to retrieve the data affected by those events.

Working with IMS

You can use the IMSRequest node to connect to IMS, a message-based transaction manager and hierarchical-database manager for z/OS.

About this task

This section contains the following concept information:

- "IBM Information Management System (IMS)" on page 2129
- "IMS nodes" on page 2130
- "IMS transactions and programs" on page 2132
- "Response models" on page 2133
- "IMS message structure" on page 2135
- "IMS connections" on page 2137

This section contains the following tasks:

- "Preparing the environment for IMS nodes" on page 731
- "Securing the connection to IMS by using SSL" on page 549
- "Propagating security credentials to IMS" on page 2144
- "Changing connection information for the IMSRequest node" on page 732

Related tasks:

"Resolving problems when you use IMS nodes" on page 3419
Advice for dealing with common problems that can arise when you develop message flows that contain IMS nodes.

Related reference:

"IMSRequest node" on page 4504
Use the IMSRequest node to send a request to run a transaction on a local or remote IMS system, and wait for a response. IMS Connect must be configured and running on the IMS system.

IBM Information Management System (IMS)

IMS is a message-based transaction manager and hierarchical-database manager for z/OS. External applications can use transactions to interact with applications that run inside IMS.

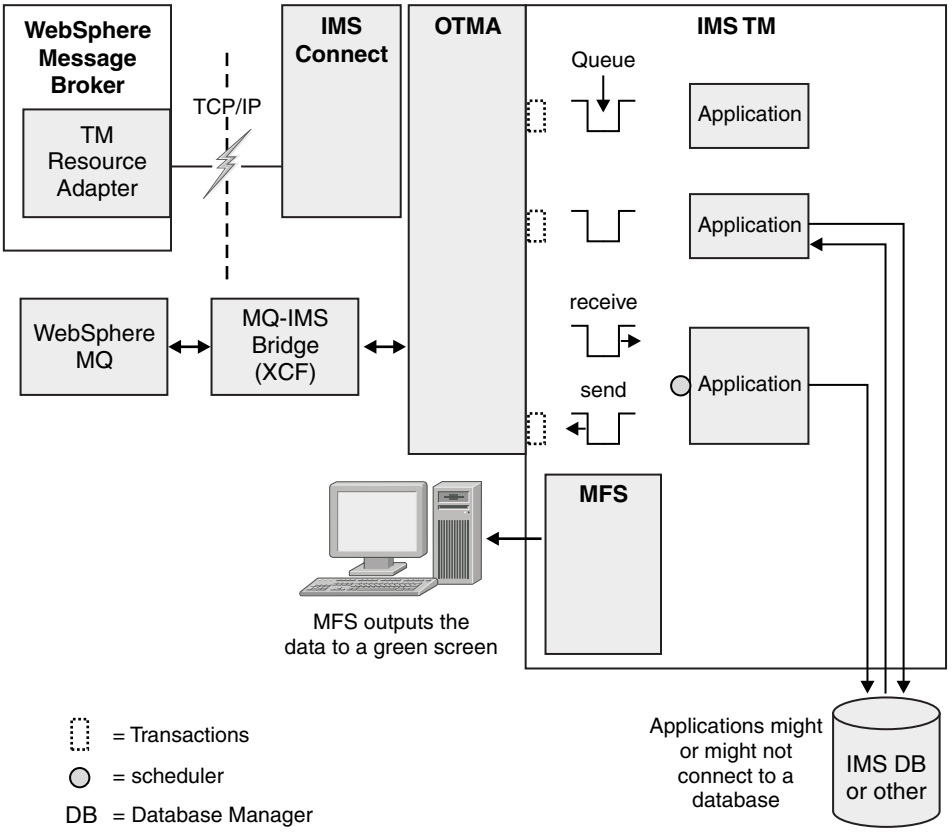
IMS includes two components:

IMS Database Manager (IMS DB)
A database management system for defining database structure, organizing business data, performing queries against the data, and performing database transactions.

IMS Transaction Manager (IMS TM)
A message-based transaction manager for processing input and output messages. IMS TM manages message queuing, security, scheduling, formatting, logging, and recovery.

In addition to these components, IMS Connect manages communications for IMS, connecting one or more clients with one or more IMS systems. IMS Connect also handles workload balancing, and supports the IBM supplied client, the IMS TM resource adapter.

The following diagram shows the layers of communication between WebSphere Message Broker and IMS.



You can also use the following methods to connect to an IMS system:

- **The WebSphere MQ-IMS bridge**

The WebSphere MQ-IMS bridge is a component of WebSphere MQ for z/OS. You can use it to access applications on your IMS system from WebSphere MQ applications. For more information about the WebSphere MQ-IMS bridge, see WebSphere MQ Version 7 Information Center online.

- **The IMS SOAP Gateway**

  The IMS SOAP Gateway is a Web service solution that integrates IMS assets in a Service-Oriented Architecture (SOA) environment. For more information, see IMS SOAP Gateway web page.

For more details about how IMS works with WebSphere Message Broker, see "IMS nodes."

For further information about IMS and its components, see the IBM Information Management Software Library web page, which contains links to the IMS information center and associated IBM Redbooks publications.

Related concepts:

"IMS nodes"
WebSphere Message Broker message flows use IMS nodes to call programs that are running in IMS.

Related tasks:

"Resolving problems when you use IMS nodes" on page 3419
Advice for dealing with common problems that can arise when you develop message flows that contain IMS nodes.

Related reference:

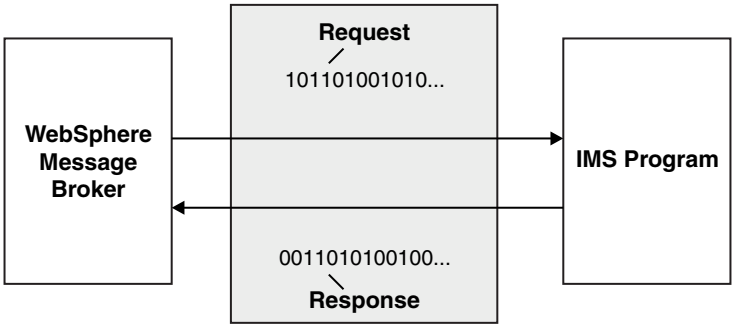
"IMSRequest node" on page 4504
Use the IMSRequest node to send a request to run a transaction on a local or remote IMS system, and wait for a response. IMS Connect must be configured and running on the IMS system.

IBM Information Management Software Library web page

IMS nodes:

WebSphere Message Broker message flows use IMS nodes to call programs that are running in IMS.

The IMS node sends a bit stream to IMS, which schedules one of its programs to process the message. The program generates a message, which IMS sends back to the IMS node, as illustrated in the following diagram.



The bit stream contains the routing information that IMS needs so that it can schedule a program to receive that bit stream. The structure of the bit stream varies

depending on whether it is a request or response bit stream. The structure of the different bit streams are described in the following sections.

Request bit stream

The structure of the request bit stream is illustrated by the following diagram.

- *LLZZ* is a four-byte field. The first two bytes indicate the length of the bit stream, and the other two bytes are reserved for use by IMS.
- The transaction code can contain up to eight characters. If the code contains less than eight characters, the transaction code must be delimited by a space. When the transaction code is less than eight bytes, IMS reads only the transaction code and one space. The response segments do not need to have the transaction name, but an IMS program can add it.
- The rest of the bit stream comprises the data that the IMS program needs.

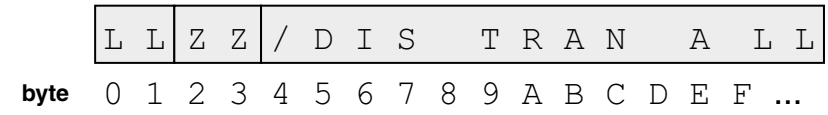
IMS reads the first twelve bytes of the bit stream, but it passes the entire bit stream to the IMS program.

Response bit stream

The structure of the response bit stream is illustrated by the following diagram.

Commands

You can also use bit streams to run commands. The structure of the response bit stream is illustrated by the following diagram.



The first character after LLZZ is the slash (/) character, which is followed by the command verb and any arguments. For commands, the response bit stream has the same format as the response bit stream for transactions: LLZZ is followed by the response data.

For more information about IMS concepts, see the following topics:
- "IMS transactions and programs" on page 2132
- "Response models" on page 2133
- "IMS message structure" on page 2135
- "IMS connections" on page 2137

Related concepts:

"IBM Information Management System (IMS)" on page 2129
IMS is a message-based transaction manager and hierarchical-database manager for z/OS. External applications can use transactions to interact with applications that run inside IMS.

Related tasks:

"Preparing the environment for IMS nodes" on page 731
Before you can use the IMS nodes, you must set up the broker runtime environment so that you can access the IMS system.

"Changing connection information for the IMSRequest node" on page 732
You can create a configurable service that the IMSRequest node or message flow refers to at run time for connection information, instead of defining the connection properties on the node or the message flow. The advantage being that you can change the host name, performance, and security values without needing to redeploy your message flow.

"Resolving problems when you use IMS nodes" on page 3419
Advice for dealing with common problems that can arise when you develop message flows that contain IMS nodes.

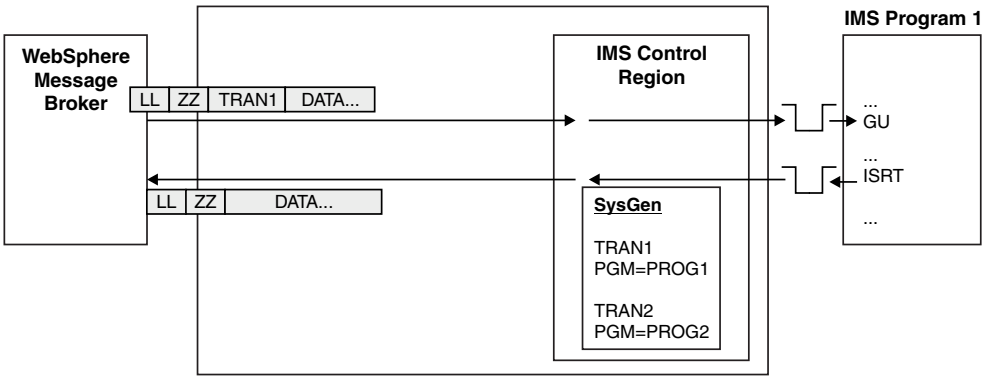
Related reference:

"IMSRequest node" on page 4504
Use the IMSRequest node to send a request to run a transaction on a local or remote IMS system, and wait for a response. IMS Connect must be configured and running on the IMS system.

IBM Information Management Software Library web page

IMS transactions and programs:

The IMS system administrator defines the transactions. For each transaction that is defined, a program name is specified. When you invoke a transaction by using an IMS node, the IMS Control Region determines which program is configured for that transaction, and queues the data for retrieval by that program.



After the program has prepared the response data for the IMS node in the message flow, it inserts that data onto another queue. This output queue is tied to the socket to which WebSphere Message Broker is connected. Therefore, multiple concurrent message flows that are calling the same transaction each have a separate queue to receive the responses.

The IMS program gets messages by issuing a GU (GetUnique) call and it produces messages by issuing an ISRT (Insert) call. These calls are known as *DL/1 calls*. DL/1 is the programming interface to IMS. Other common DL/1 calls are PURG (purge) and GN (GetNext).

Related concepts:

"IMS nodes" on page 2130
WebSphere Message Broker message flows use IMS nodes to call programs that are running in IMS.

"Response models"
The synchronous request and response model is associated with IMS.

"IMS message structure" on page 2135
Each message that is sent to and from IMS can consist of one or more segments. IMS messages often contain multiple segments.

"IMS connections" on page 2137
Open Transaction Manager Access (OTMA) is used to provide access to IMS from WebSphere Message Broker.

"IBM Information Management System (IMS)" on page 2129
IMS is a message-based transaction manager and hierarchical-database manager for z/OS. External applications can use transactions to interact with applications that run inside IMS.

Related tasks:

"Resolving problems when you use IMS nodes" on page 3419
Advice for dealing with common problems that can arise when you develop message flows that contain IMS nodes.

Related reference:

"IMSRequest node" on page 4504
Use the IMSRequest node to send a request to run a transaction on a local or remote IMS system, and wait for a response. IMS Connect must be configured and running on the IMS system.

IBM Information Management Software Library web page

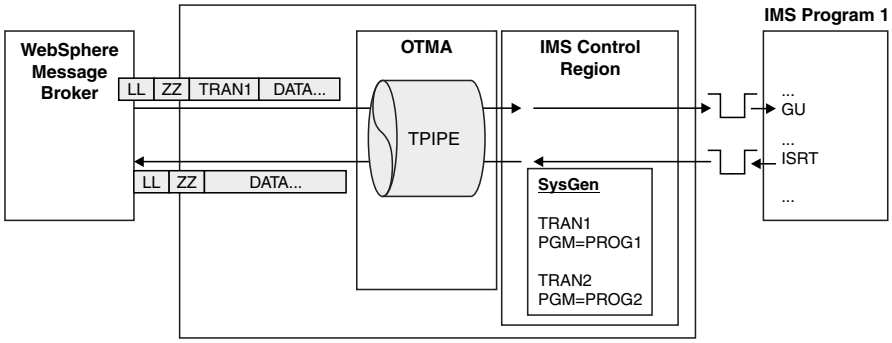
Response models:

The synchronous request and response model is associated with IMS.

Synchronous request and response

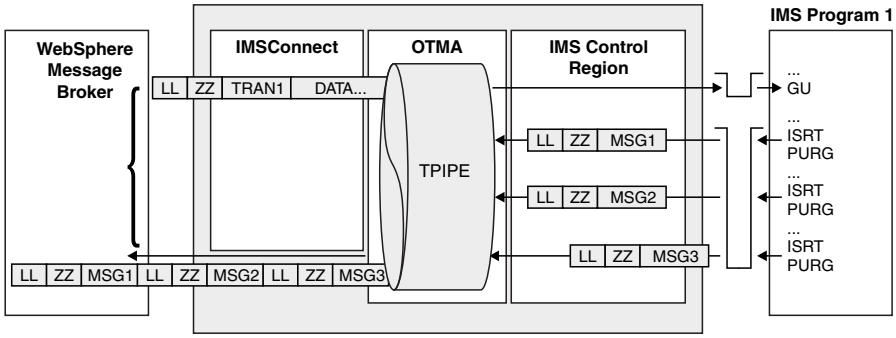
In the synchronous request and response model, the entire transmission is returned to the IMS node synchronously.

After the IMS program has prepared the response data for the WebSphere Message Broker message flow, it inserts that data onto a queue. WebSphere Message Broker uses Open Transaction Manager Access (OTMA) to communicate with the IMS program. OTMA helps to correlate the input to the IMS program with the output from the IMS program by using a transaction pipe (TPIPE). Therefore, multiple concurrent message flows that are calling the same transaction each receive the relevant response.



The TPIPE is created automatically. It is not necessary for the name of the TPIPE to be known to WebSphere Message Broker because it uses a mode of operation known as *sharable persistent sockets*, whereby a TPIPE is created automatically for every TCP/IP connection.

The output of a transaction can contain multiple messages. The IMS program inserts then purges each message, as shown in the following diagram. (For multi-segment output, the IMS program inserts multiple messages without purging them.) If the IMS program inserts multiple messages in a single sync point, OTMA correlates all of those messages to the input message and returns them all to the message flow as a single transmission.



Related concepts:

"IMS nodes" on page 2130
WebSphere Message Broker message flows use IMS nodes to call programs that are running in IMS.

"IMS transactions and programs" on page 2132
The IMS system administrator defines the transactions. For each transaction that is defined, a program name is specified. When you invoke a transaction by using an IMS node, the IMS Control Region determines which program is configured for that transaction, and queues the data for retrieval by that program.

"IMS message structure" on page 2135
Each message that is sent to and from IMS can consist of one or more segments. IMS messages often contain multiple segments.

"IMS connections" on page 2137
Open Transaction Manager Access (OTMA) is used to provide access to IMS from WebSphere Message Broker.

"IBM Information Management System (IMS)" on page 2129
IMS is a message-based transaction manager and hierarchical-database manager for z/OS. External applications can use transactions to interact with applications that run inside IMS.

Related tasks:

"Resolving problems when you use IMS nodes" on page 3419
Advice for dealing with common problems that can arise when you develop message flows that contain IMS nodes.

Related reference:

"IMSRequest node" on page 4504
Use the IMSRequest node to send a request to run a transaction on a local or remote IMS system, and wait for a response. IMS Connect must be configured and running on the IMS system.

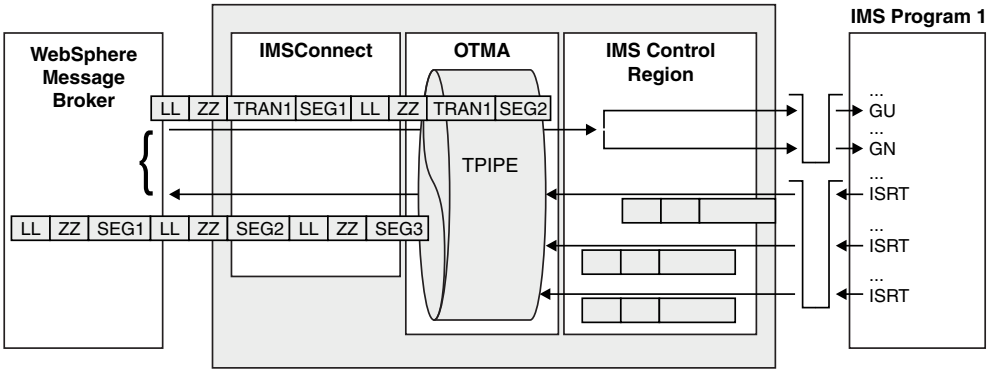
IBM Information Management Software Library web page

IMS message structure:

Each message that is sent to and from IMS can consist of one or more segments. IMS messages often contain multiple segments.

The bit stream that flows between WebSphere Message Broker and the IMS program (also known as the *transmission*) can contain multiple segments. Each segment begins with the LLZZ and Transaction code fields that are described in "IMS nodes" on page 2130. The transmission can contain multiple messages, each one containing multiple segments. The IMS program gets the segments one at a time and typically inserts the output data onto the queue one segment at a time. The IMS program purges the end of a message before it sends the first segment of the next message.

For input messages, each segment includes the LLZZ field. Only the first segment contains the transaction code (Trancode) field. For output messages, each segment includes the LLZZ field. The IMS program gets the segments one at a time. It makes a GetUnique (GU) call to read the first segment of the next message, and a GetNext (GN) call to read the next segment of the current message. The IMS program typically inserts the output data to the queue one segment at a time, and purges the end of a message before it sends the first segment of the next message, as shown in the following diagram.



A COBOL IMS program typically includes a copybook with the data structure definition of each segment. The program logic indicates the order in which the

segments are retrieved and emitted by the program. The WebSphere Message Broker application has two ways to implement this information:
- Model the entire message in MRM.
- Model each segment in MRM but configure the message flow to reflect the application logic in determining the order of these segments.

An IMS transaction's response can have various structures:
- An MRM-CWF structure, such as a message definition that is derived from a COBOL copybook
- An MRM-TDS structure when the output is 3270–based, such as a list of name=value strings
- Another structure, such as XML output from a Java program that is running in an IMS Java Processing Region (JPR)

If the message definition is derived from a COBOL copybook, a message is a sequence of segments, each of which has a model built by importing its copybook. If the output is 3270-based, each segment is a line of output with an MRM-TDS model built by understanding the IMS transaction program's output.

IMS presents program output as one or more messages (typically, one output message per input message), each of which comprises one or more segments. The IMSRequest node presents the message as a single BLOB. You can parse the message into segments and use Filter or Compute nodes to test the shape of the response to determine how to re-parse the segments with ResetContentDescriptor nodes.

You must set the LL and ZZ values on output. The LL value is the entire length of the segment, including the four-byte LLZZ prefix. Therefore, the message flow typically requires an ESQL expression to calculate the LL value. The LLZZ field must use a big-endian encoding 785.

Related concepts:

"IBM Information Management System (IMS)" on page 2129
IMS is a message-based transaction manager and hierarchical-database manager for z/OS. External applications can use transactions to interact with applications that run inside IMS.

"IMS nodes" on page 2130
WebSphere Message Broker message flows use IMS nodes to call programs that are running in IMS.

"IMS transactions and programs" on page 2132
The IMS system administrator defines the transactions. For each transaction that is defined, a program name is specified. When you invoke a transaction by using an IMS node, the IMS Control Region determines which program is configured for that transaction, and queues the data for retrieval by that program.

"Response models" on page 2133
The synchronous request and response model is associated with IMS.

"IMS connections" on page 2137
Open Transaction Manager Access (OTMA) is used to provide access to IMS from WebSphere Message Broker.

Related tasks:

"Preparing the environment for IMS nodes" on page 731
Before you can use the IMS nodes, you must set up the broker runtime environment so that you can access the IMS system.

"Changing connection information for the IMSRequest node" on page 732
You can create a configurable service that the IMSRequest node or message flow refers to at run time for connection information, instead of defining the connection properties on the node or the message flow. The advantage being that you can change the host name, performance, and security values without needing to redeploy your message flow.

"Resolving problems when you use IMS nodes" on page 3419
Advice for dealing with common problems that can arise when you develop message flows that contain IMS nodes.

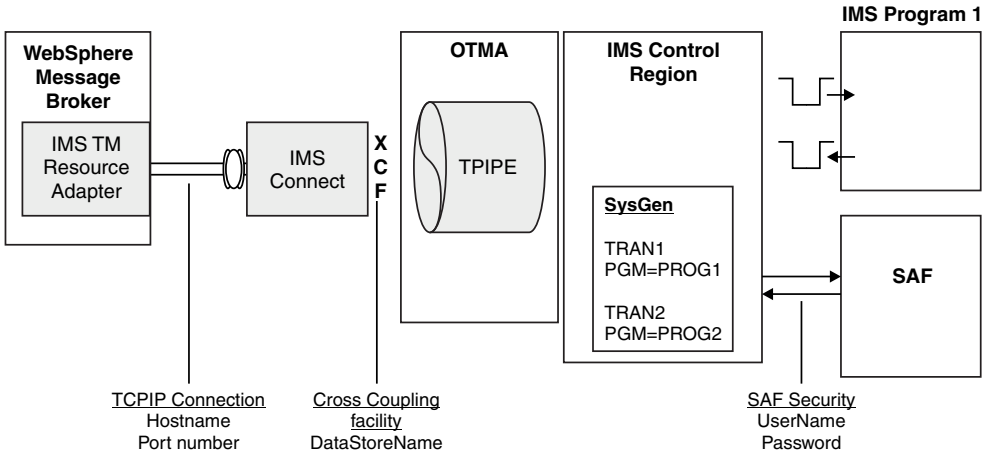
Related reference:

"IMSRequest node" on page 4504
Use the IMSRequest node to send a request to run a transaction on a local or remote IMS system, and wait for a response. IMS Connect must be configured and running on the IMS system.

IBM Information Management Software Library web page

IMS connections:

Open Transaction Manager Access (OTMA) is used to provide access to IMS from WebSphere Message Broker.



OTMA uses the z/OS Cross Coupling Facility (XCF) to provide access to IMS from an OTMA client. To access a service through XCF, you must specify a data store name on the IMS node.

IMS Connect is an OTMA client that exposes a TCP/IP interface. WebSphere Message Broker uses the IMS TM Resource Adapter, which connects to IMS through IMS Connect. To connect to IMS Connect, you must specify a host name and port number. If the IMS system is configured to authenticate users by using a System Authorization Facility (SAF) product, such as RACF, you must specify a user ID and password. You can use the **mqsisetdbparms** command to set a user ID and password for an IMSConnect configurable service. For detailed information about how to configure IMS Connect for security, see the *IMS Connect Security Support* topic in the IBM Information Management Software for z/OS Solutions Information Center.

The IMSRequest node can use an identity that is present on an input message, and propagate it to IMS, by using the `Propagate` property on the security profile that is defined for the node. For more information, see “Propagating security credentials to IMS” on page 2144.

Some restrictions exist for the OTMA environment that affect the range of IMS applications with which WebSphere Message Broker can interact. For example, OTMA cannot update the IMS main storage database (MSDB) because it has read only access to the database. For detailed information about the restrictions for the OTMA environment, see the *OTMA restrictions* topic in the IBM Information Management Software for z/OS Solutions Information Center.

Using configurable services for IMS nodes

You can configure IMS nodes to get connection details from a configurable service. For details about creating, changing, reporting, and deleting the configurable services, see “Changing connection information for the IMSRequest node” on page 732.

You can also use the IMSConnect configurable service to configure the IMSRequest node to use Secure Sockets Layer (SSL) protocol. For more information, see “Securing the connection to IMS by using SSL” on page 549.

Related concepts:

“IBM Information Management System (IMS)” on page 2129
IMS is a message-based transaction manager and hierarchical-database manager for z/OS. External applications can use transactions to interact with applications that run inside IMS.

“IMS nodes” on page 2130
WebSphere Message Broker message flows use IMS nodes to call programs that are running in IMS.

“IMS transactions and programs” on page 2132
The IMS system administrator defines the transactions. For each transaction that is defined, a program name is specified. When you invoke a transaction by using an IMS node, the IMS Control Region determines which program is configured for that transaction, and queues the data for retrieval by that program.

“Response models” on page 2133
The synchronous request and response model is associated with IMS.

“IMS message structure” on page 2135
Each message that is sent to and from IMS can consist of one or more segments. IMS messages often contain multiple segments.

Related tasks:

“Changing connection information for the IMSRequest node” on page 732
You can create a configurable service that the IMSRequest node or message flow refers to at run time for connection information, instead of defining the connection properties on the node or the message flow. The advantage being that you can change the host name, performance, and security values without needing to redeploy your message flow.

“Securing the connection to IMS by using SSL” on page 549
Configure the IMSRequest node to communicate with IMS over the Secure Sockets Layer (SSL) protocol by creating a keystore file, and configuring the broker to use SSL.

“Preparing the environment for IMS nodes” on page 731
Before you can use the IMS nodes, you must set up the broker runtime

environment so that you can access the IMS system.

"Resolving problems when you use IMS nodes" on page 3419
Advice for dealing with common problems that can arise when you develop message flows that contain IMS nodes.

Related reference:

"IMSRequest node" on page 4504
Use the IMSRequest node to send a request to run a transaction on a local or remote IMS system, and wait for a response. IMS Connect must be configured and running on the IMS system.

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

IBM Information Management Software Library web page

Changing connection information for the IMSRequest node:

You can create a configurable service that the IMSRequest node or message flow refers to at run time for connection information, instead of defining the connection properties on the node or the message flow. The advantage being that you can change the host name, performance, and security values without needing to redeploy your message flow.

Before you begin

Before you start:

- Read "Configurable services" on page 1296 to find out more about configurable services.
- Read "IBM Information Management System (IMS)" on page 2129 for background information.

About this task

Use the IMSConnect configurable service to change the connection information for the IMSRequest node. Two configurable services can connect to the same instance of IMS Connect. The properties of the IMSConnect configurable service are described in "Configurable services properties" on page 3766.

You can use the IMSConnect configurable service to configure the IMSRequest node to use Secure Sockets Layer (SSL) protocol. For more information, see "Securing the connection to IMS by using SSL" on page 549.

Creating, changing, reporting, and deleting configurable services

Procedure

- To create a configurable service, use the WebSphere Message Broker Explorer; see "Using the WebSphere Message Broker Explorer to work with configurable services" on page 644 for more information, or the **mqsicreateconfigurableservice** command, as shown in the following example. This example creates an IMSConnect configurable service for the IMS instance IMSA that is running on `test.ims.ibm.com` port 9999:

  ```
  mqsicreateconfigurableservice MB7BROKER -c IMSConnect -o myIMSConnectService
  -n Hostname,PortNumber,DataStoreName -v test.ims.ibm.com,9999,IMSA
  ```

- To change a configurable service, use the WebSphere Message Broker Explorer, or the **mqsichangeproperties** command, as shown in the following example. This

example changes all the nodes that are configured to use the *myIMSConnectService* configurable service. After you run this command, the IMSRequest node connects to the production system (`production.ims.ibm.com`) instead of the test system (`test.ims.ibm.com`). The command also changes to coded character set identifier (CCSID) to 37.

```
mqsichangeproperties MB7BROKER -c IMSConnect -o myIMSConnectService -n Hostname,CodedCharSetID
-v production.ims.ibm.com,37
```

See "Securing the connection to IMS by using SSL" on page 549 for information about how to turn on SSL support in the broker by setting the `UseSSL` and `SSLEncryptionType` IMSConnect configurable service properties.

- To display all IMSConnect configurable services, use the WebSphere Message Broker Explorer, or the **mqsireportproperties** command, as shown in the following example:

  ```
  mqsireportproperties MB7BROKER -c IMSConnect -o AllReportableEntityNames -r
  ```

- You can delete a configurable service that you have created by using the WebSphere Message Broker Explorer, or the **mqsideleteconfigurableservice** command, as shown in the following example:

  ```
  mqsideleteconfigurableservice MB7BROKER -c IMSConnect -o myIMSconnectService
  ```

  You can delete only custom-named configurable services; you cannot delete IBM-defined configurable services.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Configurable services" on page 1296
Configurable services are typically runtime properties. You can use them to define properties that are related to external services on which the broker relies; for example, an SMTP server or a JMS provider.

"Using the WebSphere Message Broker Explorer to work with configurable services" on page 644
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view, add, modify and delete configurable services.

Related tasks:

"Securing the connection to IMS by using SSL" on page 549
Configure the IMSRequest node to communicate with IMS over the Secure Sockets Layer (SSL) protocol by creating a keystore file, and configuring the broker to use SSL.

"Resolving problems when you use IMS nodes" on page 3419
Advice for dealing with common problems that can arise when you develop message flows that contain IMS nodes.

Related reference:

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"IMSRequest node" on page 4504
Use the IMSRequest node to send a request to run a transaction on a local or remote IMS system, and wait for a response. IMS Connect must be configured and running on the IMS system.

Preparing the environment for IMS nodes

Before you can use the IMS nodes, you must set up the broker runtime environment so that you can access the IMS system.

Before you begin

Before you start:

Read "IBM Information Management System (IMS)" on page 2129.

About this task

Complete the following steps to ensure that WebSphere Message Broker can connect to the IMS system.

Procedure

1. Ensure that IMS Connect is installed and started on the IMS system.
2. If you do not want to configure IMS connection properties directly on the IMSRequest node, define a configurable service for each IMS system to which you want to connect.

   For example, to create an IMSConnect configurable service for the IMS instance *IMSA* that is running on *test.ims.ibm.com*, port *9999*, run the **mqsicreateconfigurableservice** command as shown:

   ```
   mqsicreateconfigurableservice MB7BROKER -c IMSConnect -o myIMSConnectService
   -n Hostname,PortNumber,DataStoreName -v test.ims.ibm.com,9999,IMSA
   ```

   For details about how to create, change, and report configurable services, see "Changing connection information for the IMSRequest node" on page 732. You can use the IMSConnect configurable service to configure the IMSRequest node to use Secure Sockets Layer (SSL) protocol. For more information, see "Securing the connection to IMS by using SSL" on page 549.
3. Use the **mqsisetdbparms** command to set security details in the broker store.

   For example, to associate a user ID and password pair with an IMS Connect connection, run the **mqsisetdbparms** command as shown:

   ```
   mqsisetdbparms MB7BROKER -n ims::mySecurityIdentity -u myuserid -p mypassword
   ```

Related concepts:

"IBM Information Management System (IMS)" on page 2129
IMS is a message-based transaction manager and hierarchical-database manager for z/OS. External applications can use transactions to interact with applications that run inside IMS.

Related tasks:

"Changing connection information for the IMSRequest node" on page 732
You can create a configurable service that the IMSRequest node or message flow refers to at run time for connection information, instead of defining the connection properties on the node or the message flow. The advantage being that you can change the host name, performance, and security values without needing to redeploy your message flow.

"Securing the connection to IMS by using SSL" on page 549
Configure the IMSRequest node to communicate with IMS over the Secure Sockets Layer (SSL) protocol by creating a keystore file, and configuring the broker to use SSL.

Related reference:

"IMSRequest node" on page 4504
Use the IMSRequest node to send a request to run a transaction on a local or remote IMS system, and wait for a response. IMS Connect must be configured and running on the IMS system.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

Securing the connection to IMS by using SSL

Configure the IMSRequest node to communicate with IMS over the Secure Sockets Layer (SSL) protocol by creating a keystore file, and configuring the broker to use SSL.

Before you begin

Before you start:

Set up a public key infrastructure (PKI) at broker level by following the instructions in "Setting up a public key infrastructure" on page 504.

About this task

To configure the IMSRequest node to use SSL, complete the following steps.

Procedure

1. Turn on SSL support in the broker by setting the `UseSSL` and `SSLEncryptionType` properties on the IMSConnect configurable service, as shown in the following example.

   This example changes the IMSRequest node that is configured to use the *myIMSConnectService* configurable service. After you run this command, the IMSRequest node connects to IMS over SSL.

   ```
   mqsichangeproperties MB7BROKER -c IMSConnect -o myIMSConnectService -n
   UseSSL,SSLEncryptionType -v True,Weak
   ```

2. Optional: Develop a message flow that contains a IMSRequest node.
3. Test your configuration.

Related concepts:

"Security overview" on page 351
When you are designing a WebSphere Message Broker application it is important to consider the security measures that are needed to protect the information in the system.

"IMS connections" on page 2137
Open Transaction Manager Access (OTMA) is used to provide access to IMS from WebSphere Message Broker.

"Using the WebSphere Message Broker Explorer to work with configurable services" on page 644
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view, add, modify and delete configurable services.

Related tasks:

"Implementing SSL authentication" on page 504
Use SSL authentication to enhance security in your broker environment.

"Changing connection information for the IMSRequest node" on page 732
You can create a configurable service that the IMSRequest node or message flow refers to at run time for connection information, instead of defining the connection properties on the node or the message flow. The advantage being that you can change the host name, performance, and security values without needing to redeploy your message flow.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

Related reference:

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"IMSRequest node" on page 4504
Use the IMSRequest node to send a request to run a transaction on a local or remote IMS system, and wait for a response. IMS Connect must be configured and running on the IMS system.

Propagating security credentials to IMS

The IMSRequest node can use an identity that is present in the Properties folder of the message tree structure for the security credentials in a request, by using the `Propagate` property on the security profile that is defined for the node.

About this task

If an IMSRequest node is configured with a security profile, it extracts security tokens from the input message at run time, and propagates an identity to IMS.

Procedure

To propagate an identity to be used for the IMS request security credentials, complete the following steps.

1. Ensure that an appropriate security profile exists for the IMSRequest node, or create a security profile, by following the instructions in "Creating a security profile" on page 433.
2. Use the Broker Archive editor to select a security profile for the IMSRequest node that has identity propagation enabled. For detailed instructions, see "Configuring for identity propagation" on page 492.

Related concepts:

"Identity and security token propagation" on page 426
Identity and security token propagation enables the identity and security tokens (associated with each message) to be propagated throughout a message flow, and on to target applications through output or request nodes.

"IMS nodes" on page 2130
WebSphere Message Broker message flows use IMS nodes to call programs that are running in IMS.

Related tasks:

"Configuring for identity propagation" on page 492
To enable a message flow to perform identity propagation, the input nodes must extract the identity and the output node must propagate it.

Related reference:

"IMSRequest node" on page 4504
Use the IMSRequest node to send a request to run a transaction on a local or remote IMS system, and wait for a response. IMS Connect must be configured and running on the IMS system.

Working with CORBA

Use CORBA nodes to connect to CORBA Internet Inter-Orb Protocol (IIOP) applications.

About this task

This section contains the following concept information:

- "Common Object Request Broker Architecture (CORBA)" on page 2145
- "CORBA nodes" on page 2147
- "CORBA support" on page 2149
- "IDL data types" on page 2150
- "CORBA naming service" on page 2154
- "CORBA operation parameters" on page 2156

This section contains the following tasks:

- "Connecting to an external CORBA application" on page 2159
- "Developing a message flow with a CORBARequest node" on page 2161
- "Building a message for the CORBARequest node" on page 2164
- "Processing responses from a CORBARequest node" on page 2167
- "Defining where the CORBARequest node gets the object reference" on page 734

Related tasks:

"Importing an IDL file" on page 2952
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to create a message definition from an IDL file.

"Resolving problems when you use CORBA nodes" on page 3396
Advice for dealing with common problems that can arise when you develop message flows that contain CORBA nodes.

Related reference:

"CORBARequest node" on page 4349
Use the CORBARequest node to call an external CORBA application over Internet Inter-Orb Protocol (IIOP).

Common Object Request Broker Architecture (CORBA)

The Common Object Request Broker Architecture (CORBA) is a standard defined by the Object Management Group (OMG) that enables software components written in multiple computer languages and running on multiple computers to work together.

CORBA is a standard for distributing objects across networks so that operations on those objects can be called remotely. CORBA is not associated with a particular programming language, and any language with a CORBA binding can be used to call and implement CORBA objects. Objects are described in a syntax called Interface Definition Language (IDL).

CORBA includes four components:

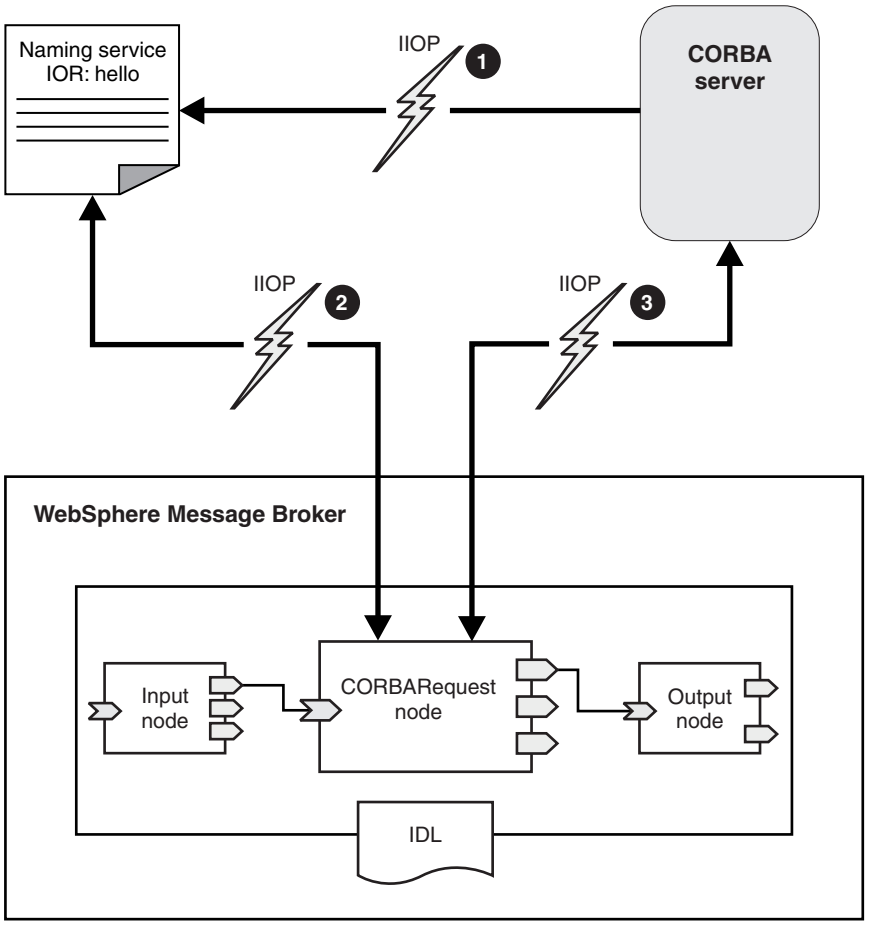
Object Request Broker (ORB)
: The Object Request Broker (ORB) handles the communication, marshaling, and unmarshaling of parameters so that the parameter handling is transparent for a CORBA server and client applications.

CORBA server
: The CORBA server creates CORBA objects and initializes them with an ORB. The server places references to the CORBA objects inside a naming service so that clients can access them.

Naming service
: The naming service holds references to CORBA objects.

CORBARequest node
: The CORBARequest node acts as a CORBA client.

The following diagram shows the layers of communication between WebSphere Message Broker and CORBA.



The diagram illustrates the following steps.

1. CORBA server applications create CORBA objects and put object references in a naming service so that clients can call them.
2. At deployment time, the node contacts a naming service to get an object reference.
3. When a message arrives, the node uses the object reference to call an operation on an object in the CORBA server.

For more details about how CORBA works with WebSphere Message Broker, see "CORBA nodes" on page 2147.

Related concepts:

"Common Object Request Broker Architecture (CORBA)" on page 2145
The Common Object Request Broker Architecture (CORBA) is a standard defined by the Object Management Group (OMG) that enables software components written in multiple computer languages and running on multiple computers to work together.

"CORBA nodes" on page 2147
Use CORBA nodes to connect WebSphere Message Broker with CORBA Internet Inter-Orb Protocol (IIOP) applications.

"IDL data types" on page 2150
When you use the DataObject domain with CORBA, you need to know how XML schema and ESQL types correspond to the types in the IDL file.

"CORBA naming service" on page 2154
A CORBA naming service holds CORBA object references.

"CORBA support" on page 2149
The CORBA nodes in WebSphere Message Broker support a set of types and operations in imported IDL files.

"DataObject parser and domain" on page 1114
Use the DataObject domain to parse and write messages for WebSphere Adapters and CORBA applications.

Related tasks:

"Importing an IDL file" on page 2952
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to create a message definition from an IDL file.

"Connecting to an external CORBA application" on page 2159
Connecting to an external CORBA application involves importing an IDL file, creating a message flow, building a message, and processing the response from the CORBARequest node.

"Building a message for the CORBARequest node" on page 2164
You can use an XML message for the CORBARequest node, or you can build a message by using another message flow node.

"Processing responses from a CORBARequest node" on page 2167
Configure the CORBARequest node to define the location to which responses are sent.

"Resolving problems when you use CORBA nodes" on page 3396
Advice for dealing with common problems that can arise when you develop message flows that contain CORBA nodes.

Related reference:

"CORBARequest node" on page 4349
Use the CORBARequest node to call an external CORBA application over Internet Inter-Orb Protocol (IIOP).

CORBA nodes:

Use CORBA nodes to connect WebSphere Message Broker with CORBA Internet Inter-Orb Protocol (IIOP) applications.

CORBA is a standard for distributing objects across networks so that operations on those objects can be called remotely. CORBA objects are described in Interface Definition Language (IDL) files, and these IDL files are used to configure the CORBA message flow nodes. The IDL file is stored in a message set project, in a folder called CORBA IDLs.

An IDL importer imports the IDL file into the message set project and creates the message definition file (`.mxsd`) in the message set. This message definition file is used for mid-flow validation, ESQL content assist, and the Mapping node.

For each IDL file, a single message definition is created. In the message definition, two messages are created for each operation in the IDL file: one message for the request, and one for the response. The request has a child element for each in and inout parameter; the response has a child element for each inout and out parameter, and a child element named "_return" for the return type of the operation.

The name of these elements is based on the interface name and operation name; for example, for the operation *sayHello* in the Interface *Hello*, the request element is

called *Hello.sayHello*, and the response element is called *Hello.sayHelloResponse*. If the interface is contained in a module, the request and response element names are qualified with the names of the modules. For example, if the operation *sayHello* in the Interface *Hello* is contained in *ModuleB*, which in turn is contained in ModuleA, the response element would be called *ModuleA.ModuleB.Hello.sayHelloResponse*.

When you add a message flow that contains CORBA nodes to a BAR file, all the IDL files that are used by the nodes are added to the BAR file automatically.

The main scenario for connecting WebSphere Message Broker with CORBA applications is described in the following section.

WebSphere Message Broker calls a CORBA server

By using a message flow that includes a CORBARequest node, you can give existing CORBA applications a new external interface; for example, a SOAP interface. The message flow uses the IDL file to configure which operation is called on which interface.

After you have deployed the BAR file, you can start the message flow. A CORBA request is sent by using values from the message tree in the DataObject domain as input parameters. If a response is received, the return type and output parameters are propagated to the Out terminal of the CORBARequest node. You can use the data that is returned from the CORBA application to verify the result from the CORBARequest node.

Related concepts:

"Common Object Request Broker Architecture (CORBA)" on page 2145
The Common Object Request Broker Architecture (CORBA) is a standard defined by the Object Management Group (OMG) that enables software components written in multiple computer languages and running on multiple computers to work together.

"CORBA nodes" on page 2147
Use CORBA nodes to connect WebSphere Message Broker with CORBA Internet Inter-Orb Protocol (IIOP) applications.

"IDL data types" on page 2150
When you use the DataObject domain with CORBA, you need to know how XML schema and ESQL types correspond to the types in the IDL file.

"CORBA naming service" on page 2154
A CORBA naming service holds CORBA object references.

"CORBA support" on page 2149
The CORBA nodes in WebSphere Message Broker support a set of types and operations in imported IDL files.

"DataObject parser and domain" on page 1114
Use the DataObject domain to parse and write messages for WebSphere Adapters and CORBA applications.

Related tasks:

"Importing an IDL file" on page 2952
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to create a message definition from an IDL file.

"Connecting to an external CORBA application" on page 2159
Connecting to an external CORBA application involves importing an IDL file, creating a message flow, building a message, and processing the response from the CORBARequest node.

"Building a message for the CORBARequest node" on page 2164
You can use an XML message for the CORBARequest node, or you can build a message by using another message flow node.

"Processing responses from a CORBARequest node" on page 2167
Configure the CORBARequest node to define the location to which responses are sent.

"Resolving problems when you use CORBA nodes" on page 3396
Advice for dealing with common problems that can arise when you develop message flows that contain CORBA nodes.

Related reference:

"CORBARequest node" on page 4349
Use the CORBARequest node to call an external CORBA application over Internet Inter-Orb Protocol (IIOP).

CORBA support:

The CORBA nodes in WebSphere Message Broker support a set of types and operations in imported IDL files.

WebSphere Message Broker supports the CORBA 2.3.1 specification and uses the Java 6 JRE ORB, and is therefore compatible with any CORBA vendor that is compatible with the JRE ORB.

WebSphere Message Broker currently supports the following CORBA types and operations.

- All primitive types, except bounded strings
- Two-way operations with in, inout, and out parameters
- User-defined exceptions
- Enums
- Modules
- Sequences (sequences must have associated typedefs; anonymous sequences are not supported)
- Structs
- Typedefs
- Comments
- The following preprocessing tokens:
  - #ifndef
  - #endif
  - #define
  - #include

  Other preprocessing tokens are ignored.

When you import an IDL file, supported and unsupported operations are listed. You can import and deploy IDL files that contain unsupported types and operations, but if you try to call one of these unsupported operations, you see an error message.

You cannot import an IDL file that is not valid. During the importing process, if you select an IDL file that is not valid, you see an error message and cannot complete the wizard.

Abstract interfaces and interfaces that contain inheritance are not supported. IDL pragma directives are not supported, but you can include the following pragmas in your IDL file (these pragmas are ignored):

- cpponly
- ID
- init
- localonly
- localonly abstract
- Prefix
- version

Related concepts:

"IDL data types"
When you use the DataObject domain with CORBA, you need to know how XML schema and ESQL types correspond to the types in the IDL file.

"CORBA nodes" on page 2147
Use CORBA nodes to connect WebSphere Message Broker with CORBA Internet Inter-Orb Protocol (IIOP) applications.

"CORBA operation parameters" on page 2156
CORBA operations can have parameters that can be modified by the server.

Related tasks:

"Importing an IDL file" on page 2952
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to create a message definition from an IDL file.

"Connecting to an external CORBA application" on page 2159
Connecting to an external CORBA application involves importing an IDL file, creating a message flow, building a message, and processing the response from the CORBARequest node.

"Resolving problems when you use CORBA nodes" on page 3396
Advice for dealing with common problems that can arise when you develop message flows that contain CORBA nodes.

Related reference:

"CORBARequest node" on page 4349
Use the CORBARequest node to call an external CORBA application over Internet Inter-Orb Protocol (IIOP).

IDL data types:

When you use the DataObject domain with CORBA, you need to know how XML schema and ESQL types correspond to the types in the IDL file.

Primitive IDL types

The following table shows the mapping between IDL types, XML schema simple types, and ESQL types.

IDL	XML schema	ESQL
boolean	xsd:boolean	BOOLEAN
char	<xsd:simpleType name="char"> <xsd:restriction base="xsd:string"> <xsd:length value="1" fixed="true/> </xsd:restriction> </simpleType>	CHARACTER

IDL	XML schema	ESQL
wchar	<xsd:simpleType name= “wchar”> <xsd:restriction base=”xsd:string”/> </xsd:simpleType>	CHARACTER
double	xsd:double	FLOAT
float	xsd:float	FLOAT
octet	xsd:unsignedByte	INTEGER
long	xsd:int	INTEGER
Long long	xsd:long	INTEGER
short	xsd:short	INTEGER
string	xsd:string	CHARACTER
wstring	xsd:string	CHARACTER
Unsigned short	xsd:unsignedShort	INTEGER
Unsigned long	xsd:unsignedInt	INTEGER
Unsigned long long	xsd:unsignedLong	DECIMAL

Complex IDL types

WebSphere Message Broker supports the following complex IDL types:

- Enums
- Typedefs
- Sequences
- Structures

Each complex type is supported in the following places:

- Return types for operations
- In parameters
- Inout parameters
- Out parameters
- Inside exceptions
- Inside structures
- Inside sequences
- Inside typedefs

The following examples show the mapping between IDL types, XML schema, and XML.

Enums

IDL Enums are mapped to enumerations in XML schema. Enums inside the tree are of type string.

Here is an example IDL file:

```
enum myEnum {A, B, C};
interface example {
 void myoperation(in myEnum input1);
};
```

Here is an example XML schema:

```
<xsd:simpleType name="myEnum">
 <xsd:restriction base="xsd:string">
  <xsd:enumeration value="A"/>
  <xsd:enumeration value="B"/>
  <xsd:enumeration value="C"/>
 </xsd:restriction>
</xsd:simpleType>
```

Here is an XML example:

```
<example.myoperation>
 <input1>A</input1>
</example.myoperation>
```

Sequences and typedefs

IDL typedefs are mapped to XML schema type restrictions. IDL sequences are mapped to XML schema sequence complex types. Sequences can be used only within typedefs.

Here is an example IDL file:

```
Typedef long myLong;
typedef sequence<long> longSeq;
interface example {
 void myoperation(in longSeq input1, inout myLong input2);
};
```

Here is an example XML schema:

```
<xsd:complexType name="longSeq">
 <xsd:sequence>
  <xsd:element name="item" minOccurs="0" maxOccurs="unbounded" type="xsd:int"/>
 </xsd:sequence>
</xsd.complexType>
```

A sequence can be bounded with the syntax sequence<long, 10>, which puts a bound in the XSD file.

Here is an XML example:

```
<example.myoperation>
 <input1>
  <item>10</item>
  <item>11</item>
  <item>12</item>
 </input1>
</example.myoperation>
```

Structures

IDL structures are mapped to XML schema complexType definitions.

Here is an example IDL file:

```
struct myStruct {
 char c;
 string str;
 octet o;
 short s;
 unsigned long long ull;
 float f;
 double d;
};
interface example {
 void myoperation(in myStruct input1);
};
```

Here is an example XML schema:

```
<xsd:complexType name="myStruct">
 <xsd:sequence>
  <xsd:element name="c" type="xsd:string" maxOccurs="1" minOccurs="1"/>
  <xsd:element name="str" type="xsd:string" nillable="true" maxOccurs="1" minOccurs="1"/>
  <xsd:element name="o" type="xsd:byte" maxOccurs="1" minOccurs="1"/>
  <xsd:element name="s" type="xsd:short" maxOccurs="1" minOccurs="1"/>
  <xsd:element name="ull" type="xsd:unsignedLong" maxOccurs="1" minOccurs="1"/>
  <xsd:element name="f" type="xsd:float" maxOccurs="1" minOccurs="1"/>
  <xsd:element name="d" type="xsd:double" maxOccurs="1" minOccurs="1"/>
 </xsd:sequence>
</xsd:complexType>
```

Here is an XML example:

```
<example.myoperation>
 <input1>
  <c>c</c>
  <str>hello</str>
  <o>12</o>
  <s>10</s>
  <ull>110</ull>
  <f>12.0</f>
  <d>12.1</d>
 </input1>
</example.myoperation>
```

Modules

In CORBA, modules provide scope. If an interface is contained in a module in the IDL file, the interface name is qualified with the module name in the following format:

ModuleName.InterfaceName.OperationName

The following example shows a module in an IDL file.

```
Module one {
 Interface OneAInterface {
 };
};
```

The fully qualified name of the interface called OneAInterface is `one.OneAInterface`. In an IDL file, modules can be nested in other modules. In this case, the fully qualified name of the interface can include more than one module name, starting from the root module; for example:

ModuleNameA.ModuleNameB.InterfaceName.OperationName

An IDL file can contain more than one operation with the same name provided that the operations are in different modules.

Related concepts:

"Common Object Request Broker Architecture (CORBA)" on page 2145
The Common Object Request Broker Architecture (CORBA) is a standard defined by the Object Management Group (OMG) that enables software components written in multiple computer languages and running on multiple computers to work together.

"CORBA nodes" on page 2147
Use CORBA nodes to connect WebSphere Message Broker with CORBA Internet Inter-Orb Protocol (IIOP) applications.

"IDL data types" on page 2150
When you use the DataObject domain with CORBA, you need to know how XML

schema and ESQL types correspond to the types in the IDL file.

"CORBA naming service"
A CORBA naming service holds CORBA object references.

"CORBA support" on page 2149
The CORBA nodes in WebSphere Message Broker support a set of types and operations in imported IDL files.

"DataObject parser and domain" on page 1114
Use the DataObject domain to parse and write messages for WebSphere Adapters and CORBA applications.

Related tasks:

"Importing an IDL file" on page 2952
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to create a message definition from an IDL file.

"Connecting to an external CORBA application" on page 2159
Connecting to an external CORBA application involves importing an IDL file, creating a message flow, building a message, and processing the response from the CORBARequest node.

"Building a message for the CORBARequest node" on page 2164
You can use an XML message for the CORBARequest node, or you can build a message by using another message flow node.

"Processing responses from a CORBARequest node" on page 2167
Configure the CORBARequest node to define the location to which responses are sent.

"Resolving problems when you use CORBA nodes" on page 3396
Advice for dealing with common problems that can arise when you develop message flows that contain CORBA nodes.

Related reference:

"CORBARequest node" on page 4349
Use the CORBARequest node to call an external CORBA application over Internet Inter-Orb Protocol (IIOP).

CORBA naming service:

A CORBA naming service holds CORBA object references.

A CORBA server puts references to CORBA objects inside a naming service so that clients can query the naming service and obtain the object reference, then call operations on the CORBA objects. Typically, a client queries the naming service once, then caches the object reference.

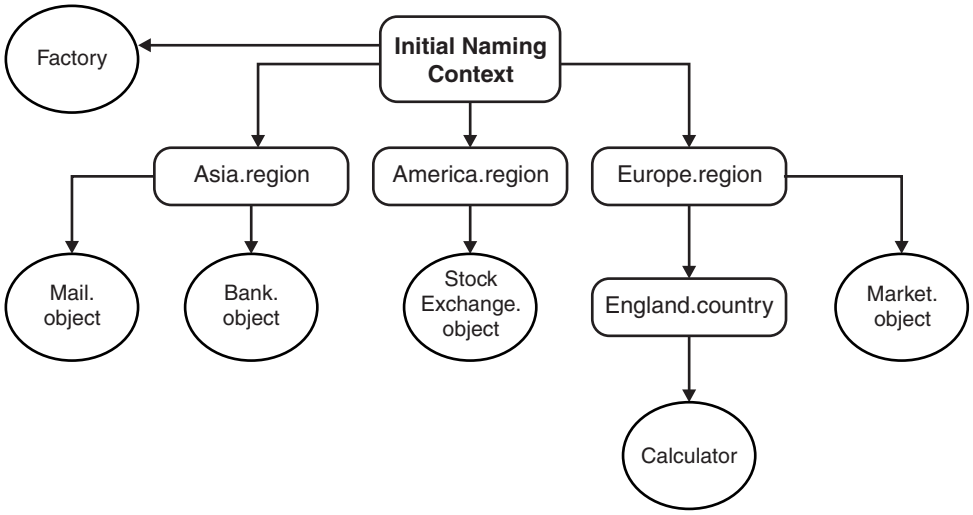
A CORBARequest node is a CORBA client; therefore, when it is deployed, the node contacts a naming service to obtain an object reference. If the object reference is not in the naming service at deployment time, or the naming service that is configured on the node is unavailable, the CORBARequest node issues a warning, and attempts to contact the naming service to get the object reference when it receives a message. If an object reference cannot be acquired from the naming service when the node receives a message, an error is issued. You can specify the location of an object reference by using the properties on the CORBARequest node, or by using the CORBA configurable service. For more information, see "CORBARequest node" on page 4349 and "Defining where the CORBARequest node gets the object reference" on page 734.

Identifying an object reference in a naming service

Each object in a naming service has a unique name. You must use this name when you configure the `Object reference name` property on the CORBARequest node.

Naming services are typically arranged in a hierarchy so that names can be given context or scope. The initial naming context is at the top of the hierarchy. Object references can be added to the initial naming context, and additional contexts can exist below it. The number of levels in the hierarchy is unlimited.

Object references and contexts can be assigned a *kind* to facilitate grouping. The kind is appended to the context in the format *context.kind*. If you are using WebSphere Message Broker to access an external CORBA application, you need to know the location of the naming service and the name of the object reference in the naming service. The following example shows how to determine the exact string representation of the name.



In the diagram, contexts are represented by squares, and object references are represented by circles.

- An object called Factory is directly attached to the initial naming context.
- Three contexts, with kind *region*, are also attached to the initial naming context.
- These three contexts each have one or more object references attached to them.
- The Europe context has an England context attached to it, of kind *country*, which has an object attached to it (*Calculator*).

The name that you specify when you configure the `Object reference name` property on the CORBARequest node reflects the position of the object in the hierarchy. The following table shows how to refer to the specific objects in the diagram.

Object	Object reference name
Factory	Factory
Bank	Asia.region/Bank.object
Mail	Asia.region/Mail.object
StockExchange	America.region/StrockExchange.object
Market	Europe.region/Market.object

Object	Object reference name
Calculator	Europe.region/England.country/Calculator

All objects in the naming service can be connected directly to the initial naming context; in which case, their names would be in the same format as the Factory object in this example.

Related concepts:

"Common Object Request Broker Architecture (CORBA)" on page 2145
The Common Object Request Broker Architecture (CORBA) is a standard defined by the Object Management Group (OMG) that enables software components written in multiple computer languages and running on multiple computers to work together.

"CORBA nodes" on page 2147
Use CORBA nodes to connect WebSphere Message Broker with CORBA Internet Inter-Orb Protocol (IIOP) applications.

"IDL data types" on page 2150
When you use the DataObject domain with CORBA, you need to know how XML schema and ESQL types correspond to the types in the IDL file.

"CORBA naming service" on page 2154
A CORBA naming service holds CORBA object references.

"CORBA support" on page 2149
The CORBA nodes in WebSphere Message Broker support a set of types and operations in imported IDL files.

"DataObject parser and domain" on page 1114
Use the DataObject domain to parse and write messages for WebSphere Adapters and CORBA applications.

Related tasks:

"Importing an IDL file" on page 2952
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to create a message definition from an IDL file.

"Connecting to an external CORBA application" on page 2159
Connecting to an external CORBA application involves importing an IDL file, creating a message flow, building a message, and processing the response from the CORBARequest node.

"Building a message for the CORBARequest node" on page 2164
You can use an XML message for the CORBARequest node, or you can build a message by using another message flow node.

"Processing responses from a CORBARequest node" on page 2167
Configure the CORBARequest node to define the location to which responses are sent.

"Resolving problems when you use CORBA nodes" on page 3396
Advice for dealing with common problems that can arise when you develop message flows that contain CORBA nodes.

Related reference:

"CORBARequest node" on page 4349
Use the CORBARequest node to call an external CORBA application over Internet Inter-Orb Protocol (IIOP).

CORBA operation parameters:

CORBA operations can have parameters that can be modified by the server.

CORBA operations can have in, out, and inout parameters. In and inout parameters dictate the appearance of the tree under the DataObject domain when going into a CORBARequest node. The return type, inout and out parameters dictate the appearance of the tree when leaving a CORBARequest node.

The inbound message is contained in the element *interfaceName.operationName* and needs an element for every in or inout parameter. If the interface is contained in a module, the name is qualified with the name of the module. If the module is nested in other modules, all module names are stated; for example: *moduleNameA.moduleNameB.interfaceName.operationName*. Out parameters are not required because the client does not send a value for out parameters. These elements must be in the same order as the parameters in the IDL file.

The output message from the CORBARequest node is contained in the element *interfaceName.operationName*Response. If the interface is contained in a module, the name is qualified with the module name. The outbound message has an element for the return type, named *_return*, and an element for every inout and out parameter.

Here is an example of an IDL file:

```
interface exampleInterface {
 string outsideModuleOperation(in string one, out string two, inout string three);

};
```

The input message might look like the following example:

```
<exampleInterface.outsideModuleOperation>
 <one>something</one>
 <three>something</three>
</exampleInterface.outsideModuleOperation>
```

The output message might look like the following example:

```
<exampleInterface.outsideModuleOperationResponse>
 <_return>something</_return>
 <two>something</two>
 <three>something</three>
</exampleInterface.outsideModuleOperationResponse>
```

The input message requires all in and inout parameters, therefore one and three are specified. The output has the following elements:

- A *_return* element (because the operation has a return type of string)
- An element called *two* (because *two* is an out parameter)
- An element called *three* (because *three* is an inout parameter)

User-defined exceptions

Exceptions are propagated to the Error terminal under the DataObject domain; the structure of the message depends on the exception.

The following example shows how a user-defined exception is defined in an IDL file.

```
exception BadRecord {
    string why;
  };
```

```
interface SomeInterface {
  long bar(in float pi) raises (BadRecord);
};
```

The operation *bar* can issue the exception *BadRecord*. If this exception is issued, the following message is propagated to the Error terminal.

```
<BadRecord>
 <why>Reason text</why>
</BadRecord>
```

Edge cases

Two identified edge cases exist for reading a tree and producing a tree:

- No input parameters exist.
- A void function has no inout or out parameters

The following sample IDL file illustrates two examples.

```
interface exampleInterface {
 string exampleOne();
void exampleTwo(in string one);
};
```

- **Example 1 - no input parameters exist**

  Input message: The input message is irrelevant because the CORBARequest node does not look at the body of the message.

  Output message:

  ```
  <exampleInterface.exampleOneResponse>
   <_return>something</_return>
  </exampleInterface.exampleOneResponse>
  ```

- **Example 2 - A void function has no inout or out parameters**

  Input message:

  ```
  <exampleInterface.exampleTwo>
   <one>something</one>
  </exampleInterface.exampleTwo>
  ```

  Output message:

  ```
  <exampleInterface.exampleTwoResponse>
  </exampleInterface.exampleTwoResponse>
  ```

Related concepts:

"Common Object Request Broker Architecture (CORBA)" on page 2145
The Common Object Request Broker Architecture (CORBA) is a standard defined by the Object Management Group (OMG) that enables software components written in multiple computer languages and running on multiple computers to work together.

"CORBA nodes" on page 2147
Use CORBA nodes to connect WebSphere Message Broker with CORBA Internet Inter-Orb Protocol (IIOP) applications.

"IDL data types" on page 2150
When you use the DataObject domain with CORBA, you need to know how XML schema and ESQL types correspond to the types in the IDL file.

"CORBA naming service" on page 2154
A CORBA naming service holds CORBA object references.

"CORBA support" on page 2149
The CORBA nodes in WebSphere Message Broker support a set of types and operations in imported IDL files.

"DataObject parser and domain" on page 1114
Use the DataObject domain to parse and write messages for WebSphere Adapters and CORBA applications.

Related tasks:

"Importing an IDL file" on page 2952
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to create a message definition from an IDL file.

"Connecting to an external CORBA application"
Connecting to an external CORBA application involves importing an IDL file, creating a message flow, building a message, and processing the response from the CORBARequest node.

"Developing a message flow with a CORBARequest node" on page 2161
To connect to an external CORBA application, create a message flow that contains a CORBARequest node.

"Building a message for the CORBARequest node" on page 2164
You can use an XML message for the CORBARequest node, or you can build a message by using another message flow node.

"Processing responses from a CORBARequest node" on page 2167
Configure the CORBARequest node to define the location to which responses are sent.

"Resolving problems when you use CORBA nodes" on page 3396
Advice for dealing with common problems that can arise when you develop message flows that contain CORBA nodes.

Related reference:

"CORBARequest node" on page 4349
Use the CORBARequest node to call an external CORBA application over Internet Inter-Orb Protocol (IIOP).

Connecting to an external CORBA application

Connecting to an external CORBA application involves importing an IDL file, creating a message flow, building a message, and processing the response from the CORBARequest node.

Before you begin

Before you start:

An IDL file is used to configure the CORBARequest node. Ensure that you have a valid IDL file that contains elements that are supported by WebSphere Message Broker. The CORBA IDL file must contain at least one interface that has one operation. For more information, see "CORBA support" on page 2149.

About this task

To connect to an external CORBA application, complete the following steps.

Procedure

1. Create a message flow project and a message set project.
2. Import an IDL file into the WebSphere Message Broker Toolkit, as described in "Importing an IDL file" on page 2952.
3. Develop a message flow that contains a CORBARequest node, as described in "Developing a message flow with a CORBARequest node" on page 2161.

4. Build a message to send to the CORBARequest node, as described in "Building a message for the CORBARequest node" on page 2164.
5. Process the response from the CORBARequest node, as described in "Processing responses from a CORBARequest node" on page 2167.

What to do next

After you have deployed a message flow that contains a CORBARequest node, you can change the location of the object reference by using a configurable service. For more information, see "Defining where the CORBARequest node gets the object reference" on page 734.

Related concepts:

"Common Object Request Broker Architecture (CORBA)" on page 2145
The Common Object Request Broker Architecture (CORBA) is a standard defined by the Object Management Group (OMG) that enables software components written in multiple computer languages and running on multiple computers to work together.

"CORBA nodes" on page 2147
Use CORBA nodes to connect WebSphere Message Broker with CORBA Internet Inter-Orb Protocol (IIOP) applications.

"IDL data types" on page 2150
When you use the DataObject domain with CORBA, you need to know how XML schema and ESQL types correspond to the types in the IDL file.

"CORBA naming service" on page 2154
A CORBA naming service holds CORBA object references.

"CORBA support" on page 2149
The CORBA nodes in WebSphere Message Broker support a set of types and operations in imported IDL files.

"DataObject parser and domain" on page 1114
Use the DataObject domain to parse and write messages for WebSphere Adapters and CORBA applications.

Related tasks:

"Importing an IDL file" on page 2952
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to create a message definition from an IDL file.

"Building a message for the CORBARequest node" on page 2164
You can use an XML message for the CORBARequest node, or you can build a message by using another message flow node.

"Processing responses from a CORBARequest node" on page 2167
Configure the CORBARequest node to define the location to which responses are sent.

"Creating a message flow project" on page 1425
A message flow project is a container for message flows; you must create a project before you can create a message flow.

"Resolving problems when you use CORBA nodes" on page 3396
Advice for dealing with common problems that can arise when you develop message flows that contain CORBA nodes.

Related reference:

"CORBARequest node" on page 4349
Use the CORBARequest node to call an external CORBA application over Internet Inter-Orb Protocol (IIOP).

Developing a message flow with a CORBARequest node:

To connect to an external CORBA application, create a message flow that contains a CORBARequest node.

Before you begin

Before you start:

Ensure that you have created a message flow project and message set project, and that you have imported an IDL file, as described in "Connecting to an external CORBA application" on page 2159.

About this task

You can create and configure a message flow manually, or you can create a message flow by dragging an imported IDL file onto the canvas.

Creating a message flow from an imported IDL file:
Procedure

1. Drag an IDL file from the `CORBA IDLs` folder in the Broker Development view to an empty canvas. (If you have imported an IDL file that contains includes, drag the top-level IDL file onto the canvas.)

   A CORBARequest node is created. The `IDL file`, `Interface name`, and `Operation name` properties are set according to the IDL file.
2. If the IDL file contains more than one interface or operation, select an interface and operation from the dialog box.
3. Configure the following properties on the CORBARequest node:
   - `Naming service`: Specify the host name and port of the naming service.

     The format of this value is host:port, where port is optional; for example, localhost:2809. You can obtain this value from the administrator of the CORBA application that you are calling.
   - `Object reference name`: Specify the name of the object reference in the naming service.

     You can obtain this value from the CORBA server that you are calling. For more information about how to specify the object reference name, see "CORBA naming service" on page 2154.
4. Add to the message flow other nodes that build the incoming and outgoing messages.

   You can use an XML message for the CORBARequest node, or you can build a message by using a Compute, JavaCompute, or PHPCompute node. If the incoming message has a message model, you can use a Mapping node to build the message that is sent to the CORBARequest node.

   If the message that the CORBARequest node produces has a message model, you can use a Mapping node to build the outgoing message.
5. Build a message for the CORBARequest node by using the examples in "Building a message for the CORBARequest node" on page 2164.
6. Save the message flow.
7. Deploy the message flow. (If you have used an IDL file that contains includes, ensure that all the IDL files are deployed with the message flow.)

What to do next

You can also drag an IDL file onto an existing CORBARequest node. The existing `IDL file`, `Interface name`, and `Operation name` properties are replaced with values from the new IDL file, and the `Naming service` and `Object reference name` properties are cleared. If the IDL file contains more than one interface or operation, the `Interface name` property is set to the first interface in the IDL file, and the `Operation name` property is set to the first operation in that interface.

Creating a message flow manually:
Procedure

1. Create a message flow.
2. Add a CORBARequest node to the message flow.
3. Configure the following properties on the CORBARequest node:
   - `Naming service`: Specify the host name and port of the naming service.

     The format of this value is host:port, where port is optional; for example, localhost:2809. You can obtain this value from the administrator of the CORBA application that you are calling.

     You can also use a configurable service to specify a naming service; for more information, see "Defining where the CORBARequest node gets the object reference" on page 734.
   - `Object reference name`: Specify the name of the object reference in the naming service.

     You can obtain this value from the CORBA server that you are calling. For more information about how to specify the object reference name, see "CORBA naming service" on page 2154.

     You can also use a configurable service to specify an object reference name; for more information, see "Defining where the CORBARequest node gets the object reference" on page 734.
   - `IDL file`: Click **Browse** and select the IDL file from the message set project. If you have imported an IDL file that contains includes, select the top-level IDL file.
   - `Interface name`: Specify the name of the interface in the IDL file that the node calls.
   - `Operation name`: Specify the name of the operation from the interface that you select in the IDL file.

     You can override this property in the local environment by specifying a value in the following location:

     ```
     $LocalEnvironment/Destination/CORBA/Request/OperationName
     ```
4. Add to the message flow other nodes that build the incoming and outgoing messages.

   You can use an XML message for the CORBARequest node, or you can build a message by using a Compute, JavaCompute, or PHPCompute node. If the incoming message has a message model, you can use a Mapping node to build the message that is sent to the CORBARequest node.

   If the message that the CORBARequest node produces has a message model, you can use a Mapping node to build the outgoing message.
5. Build a message for the CORBARequest node by using the examples in "Building a message for the CORBARequest node" on page 2164.
6. Save the message flow.

7. Deploy the message flow. (If you have used an IDL file that contains includes, ensure that all the IDL files are deployed with the message flow.)

What to do next

Next: After you have deployed the message flow, learn how calls are processed by the CORBARequest node; see "Processing responses from a CORBARequest node" on page 2167.

Related concepts:

"Common Object Request Broker Architecture (CORBA)" on page 2145
The Common Object Request Broker Architecture (CORBA) is a standard defined by the Object Management Group (OMG) that enables software components written in multiple computer languages and running on multiple computers to work together.

"CORBA nodes" on page 2147
Use CORBA nodes to connect WebSphere Message Broker with CORBA Internet Inter-Orb Protocol (IIOP) applications.

"IDL data types" on page 2150
When you use the DataObject domain with CORBA, you need to know how XML schema and ESQL types correspond to the types in the IDL file.

"CORBA naming service" on page 2154
A CORBA naming service holds CORBA object references.

"CORBA support" on page 2149
The CORBA nodes in WebSphere Message Broker support a set of types and operations in imported IDL files.

"DataObject parser and domain" on page 1114
Use the DataObject domain to parse and write messages for WebSphere Adapters and CORBA applications.

Related tasks:

"Importing an IDL file" on page 2952
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to create a message definition from an IDL file.

"Connecting to an external CORBA application" on page 2159
Connecting to an external CORBA application involves importing an IDL file, creating a message flow, building a message, and processing the response from the CORBARequest node.

"Building a message for the CORBARequest node" on page 2164
You can use an XML message for the CORBARequest node, or you can build a message by using another message flow node.

"Processing responses from a CORBARequest node" on page 2167
Configure the CORBARequest node to define the location to which responses are sent.

"Resolving problems when you use CORBA nodes" on page 3396
Advice for dealing with common problems that can arise when you develop message flows that contain CORBA nodes.

Related reference:

"CORBARequest node" on page 4349
Use the CORBARequest node to call an external CORBA application over Internet Inter-Orb Protocol (IIOP).

Building a message for the CORBARequest node:

You can use an XML message for the CORBARequest node, or you can build a message by using another message flow node.

Before you begin

Before you start:

Ensure that you have created and configured a message flow with a CORBARequest node, as described in "Developing a message flow with a CORBARequest node" on page 2161.

About this task

The CORBARequest node requires an input message. The node can use an XML message from another node, or you can build a message by using a Compute, JavaCompute, or PHPCompute node. If the incoming message has a message model, you can use a Mapping node to build the message to send to the CORBARequest node. The incoming message tree must be in the DataObject domain, and the elements in the tree must match the IDL interface that you are calling. The physical representation of a message in the DataObject domain is XML; therefore, your application constructs message bodies like the following examples.

You can specify the location in the incoming message tree from which data is retrieved to form the request that is sent by the CORBARequest node. Specify this location by using the `Data location` property on the Request tab. The default value is `$Body`.

Procedure

1. Find out what data needs to be in the body of the message to send to the CORBARequest node.
2. Specify the top-level element (interface.operationName) for the message that you need to send to the CORBARequest node.
3. Specify a child for each in and inout parameter. You do not have to pass in a value for out parameters.

Example

The following examples show the XML and ESQL that could be received by a message flow.

A mixture of in inout and out parameters

Here is an example IDL file:

```
interface

ExampleOne {
 enum completion{YES, NO, MAYBE};
  typedef sequence<string> stringlist;
 struct stringobject {string member;};

 string exampleOneOperation(in string inparamA, inout string inoutparamA, out string outparamA);
 completion exampleOneOperationB(in stringlist inparamB, inout stringobject inoutparamB, out complet
}
```

This IDL file contains an interface with two operations, which have an in parameter, an inout parameter, and an out parameter.

For the first operation, *exampleOneOperationA*, you must pass in the parameters *inparamA* and *inoutparamA* under the top-level type *ExampleOne.exampleOneOperationA*. You do not need to pass in the *outparamA* parameter.

Here is an XML example:

```
<ExampleOne.exampleOneOperationA>
 <inparamA>your value</inparamA>
 <inoutparamA>your value</inoutparamA>
</ExampleOne.exampleOneOperationA>
```

Here is an ESQL example:

```
SET
OutputRoot.DataObject."ExampleOne.exampleOneOperationA".inparamA = 'yourvalue';

SET
OutputRoot.DataObject."ExampleOne.exampleOneOperationA".inoutparamA = 'yourvalue';
```

For the second operation, *exampleOneOperationB*, you must pass in the parameters *inparamB* and *inoutparamB* under the top-level type *ExampleOne.exampleOneOperationB*. You do not need to pass in the *outparamB* parameter.

Here is an XML example:

```
<ExampleOne.exampleOneOperationB>
 <inparamB><item>your value</item></inparamB>
 <inoutparamB><member>your value</member></inoutparamB>
</ExampleOne.exampleOneOperationB>
```

Here is an ESQL example:

```
SET
OutputRoot.DataObject."ExampleOne.exampleOneOperationB".inparamB.item = 'your value';

SET
OutputRoot.DataObject."ExampleOne.exampleOneOperationB".inoutparamB.member = 'your value'
```

In and inout parameters only

Here is an example IDL file:

```
interface

ExampleTwo {
 enum completion{YES, NO, MAYBE};
 typedef sequence<string> stringlist;
 struct stringobject {string member;};

 string exampleTwoOperationA(in string inparamA, inout string inoutparamA);
 completion exampleTwoOperationB(in stringlist inparamB, inout stringobject inoutparamB);

}
```

This IDL file contains two operations with an in and inout parameter only. The removal of the out parameter and the existence of only in and inout parameters does not change the parameters that you need to pass in.

To call the first operation, *exampleTwoOperationA*, pass in *inparamA* and *inoutparamA* under the top-level type *ExampleTwo.exampleTwoOperationA*.

Here is an XML example:

```
<ExampleTwo.exampleTwoOperationA>
 <inparamA>your value</inparamA>
 <inoutparamA>your value</inoutparamA>
</ExampleTwo.exampleTwoOperationA>
```

Here is an ESQL example:

```
SET
OutputRoot.DataObject."ExampleTwo.exampleTwoOperationA".inparamA = 'yourvalue';

SET
OutputRoot.DataObject."ExampleTwo.exampleTwoOperationA".inoutparamA = 'yourvalue';
```

To call the second operation, *exampleTwoOperationB*, pass in *inparamB* and *inoutparamB* under the top-level type *ExampleTwo.exampleTwoOperationB*.

Here is an XML example:

```
<ExampleTwo.exampleTwoOperationB>
 <inparamB><item>your value</item></inparamB>
 <inoutparamB><member>your value</member></inoutparamB>
</ExampleTwo.exampleTwoOperationB>
```

Here is an ESQL example:

```
SET
OutputRoot.DataObject."ExampleTwo.exampleTwoOperationB".inparamB.item = 'your value';

SET
OutputRoot.DataObject."ExampleTwo.exampleTwoOperationB".inoutparamB.member = 'your value';
```

No parameters or out parameters only

If the operation contains no parameters, or out parameters only, you do not need to put anything in the body of the message. The CORBARequest node does not look at the incoming message.

What to do next

Next: Process the responses from the CORBARequest, as described in "Processing responses from a CORBARequest node" on page 2167.

Related concepts:

"Common Object Request Broker Architecture (CORBA)" on page 2145
The Common Object Request Broker Architecture (CORBA) is a standard defined by the Object Management Group (OMG) that enables software components written in multiple computer languages and running on multiple computers to work together.

"CORBA nodes" on page 2147
Use CORBA nodes to connect WebSphere Message Broker with CORBA Internet Inter-Orb Protocol (IIOP) applications.

"IDL data types" on page 2150
When you use the DataObject domain with CORBA, you need to know how XML schema and ESQL types correspond to the types in the IDL file.

"CORBA naming service" on page 2154
A CORBA naming service holds CORBA object references.

"CORBA support" on page 2149
The CORBA nodes in WebSphere Message Broker support a set of types and operations in imported IDL files.

"DataObject parser and domain" on page 1114
Use the DataObject domain to parse and write messages for WebSphere Adapters and CORBA applications.

Related tasks:

"Importing an IDL file" on page 2952
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to create a message definition from an IDL file.

"Connecting to an external CORBA application" on page 2159
Connecting to an external CORBA application involves importing an IDL file, creating a message flow, building a message, and processing the response from the CORBARequest node.

"Developing a message flow with a CORBARequest node" on page 2161
To connect to an external CORBA application, create a message flow that contains a CORBARequest node.

"Processing responses from a CORBARequest node"
Configure the CORBARequest node to define the location to which responses are sent.

"Resolving problems when you use CORBA nodes" on page 3396
Advice for dealing with common problems that can arise when you develop message flows that contain CORBA nodes.

Related reference:

"CORBARequest node" on page 4349
Use the CORBARequest node to call an external CORBA application over Internet Inter-Orb Protocol (IIOP).

Processing responses from a CORBARequest node:

Configure the CORBARequest node to define the location to which responses are sent.

Before you begin

Before you start:

Complete the tasks that are described in the following topics:

- "Developing a message flow with a CORBARequest node" on page 2161
- "Building a message for the CORBARequest node" on page 2164

About this task

The CORBARequest node has three output terminals:

- Out: For a successful invocation of the operation.
- Error: If a CORBA system exception or user-defined exception is issued.
- Failure: If a failure occurs in the node; for example, if the node cannot communicate with the naming service or it receives an incorrect message.

You can select the location to which to send the response by configuring the `Output data location` property on the **Result** tab. This property specifies the message tree location to which the CORBARequest node puts output. The output is put under the DataObject domain.

Procedure

- **Processing successful calls**

  When a call is successful, the resulting message is propagated to the Out terminal. Inside the tree is a top-level element named *InterfaceName.OperationName*Response. Under this type is the return type of the operation (named _return) and every inout or out parameter. In parameters are not propagated because they do not change.

  A mixture of in inout and out parameters

  Here is an example IDL file:

```
interface

ExampleOne {
   string exampleOneOperation(in string inparam, out string outparam, inout string inoutparam);

}
```

  You receive the parameters *outparam* and *inoutparam* under the top-level element `ExampleOne.exampleOneOperationResponse`.

  Here is an XML example:

```
<ExampleOne.exampleOneOperationResponse>
 <_return>The operation return value</_return>
 <outparam>value from corba app</outparam>
 </inoutparam>your value changed by the corba app<inoutparam>
</ExampleOne.exampleOneOperationResponse>
```

  Out and inout parameters

  Here is an example IDL file:

```
interface

ExampleTwo {
   string exampleTwoOperation(out string outparam, inout string inoutparam);

}
```

  The removal of the in parameter makes no difference to the message that is propagated from a CORBARequest node because in parameters are not propagated.

  Here is an XML example:

```
<ExampleTwo.exampleTwoOperationResponse>
 <_return>The operation return value</_return>
 <outparam>value from corba app</outparam>
 </inoutparam>your value changed by the corba app<inoutparam>
</ExampleTwo.exampleTwoOperationResponse>
```

  In parameters only or no parameters

  If the operation contains no parameters, or in parameters only, you receive the _return value under the top-level element.

  Here is an example IDL file:

```
interface

ExampleThree {
   string exampleThreeOperation(in string inparam);

}
```

  Here is an XML example:

```
<ExampleThree.exampleThreeOperationResponse>
 <_return>The operation return value</_return>
</ExampleThree.exampleThreeOperationResponse>
```

No return type (void) and no inout or out parameters

If the operation that you are calling has no return type (void) and no inout or out parameters, no values are returned from the CORBA application to put in the tree. In this case, only the top-level element is created.

Here is an example IDL file:

```
interface

ExampleFour {
   void exampleFourOperation(in string inparam);

}
```

Here is an XML example:

```
<ExampleFour.exampleFourOperationResponse>
</ExampleFour.exampleFourOperationResponse>
```

- **Processing user-defined exceptions and CORBA system exceptions**

  When a CORBA user-defined exception or a CORBA system exception occurs, a message is propagated to the Error terminal. CORBA system exceptions have a standard shape and look like this:

  In the IDL file:

```
exception SystemException { // descriptive of error

 unsigned long;      // more detail about error
 CompletionStatus;   // yes, no, maybe
}
```

  In the tree:

```
<SystemException>
 <minor>10</minor>
 <completed>maybe</completed>
</SystemException>
```

  The top-level element is the name of the CORBA system exception that is issued. The structure of a user-defined exception is based on the IDL for the exception. Here is an example:

  In the IDL file:

```
  exception BadRecord {
 string why;

};
```

  In the message:

```
<BadRecord>
 <why>reason text</why>
</BadRecord>
```

  As you can see from this example, the structure of the message is based on the exception, and is not qualified by the operation that was being called.

- **Handling failures in the node**

  Any failures are propagated to the Failure terminal. Possible failures include:

  – Inability to communicate with the CORBA server
  – Inability to communicate with the CORBA naming service
  – Invalid body (including the wrong top-level element or missing parameters)

Related concepts:

"Common Object Request Broker Architecture (CORBA)" on page 2145
The Common Object Request Broker Architecture (CORBA) is a standard defined by the Object Management Group (OMG) that enables software components written in multiple computer languages and running on multiple computers to work together.

"CORBA nodes" on page 2147
Use CORBA nodes to connect WebSphere Message Broker with CORBA Internet Inter-Orb Protocol (IIOP) applications.

"IDL data types" on page 2150
When you use the DataObject domain with CORBA, you need to know how XML schema and ESQL types correspond to the types in the IDL file.

"CORBA naming service" on page 2154
A CORBA naming service holds CORBA object references.

"CORBA support" on page 2149
The CORBA nodes in WebSphere Message Broker support a set of types and operations in imported IDL files.

"DataObject parser and domain" on page 1114
Use the DataObject domain to parse and write messages for WebSphere Adapters and CORBA applications.

Related tasks:

"Importing an IDL file" on page 2952
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to create a message definition from an IDL file.

"Connecting to an external CORBA application" on page 2159
Connecting to an external CORBA application involves importing an IDL file, creating a message flow, building a message, and processing the response from the CORBARequest node.

"Developing a message flow with a CORBARequest node" on page 2161
To connect to an external CORBA application, create a message flow that contains a CORBARequest node.

"Building a message for the CORBARequest node" on page 2164
You can use an XML message for the CORBARequest node, or you can build a message by using another message flow node.

"Resolving problems when you use CORBA nodes" on page 3396
Advice for dealing with common problems that can arise when you develop message flows that contain CORBA nodes.

Related reference:

"CORBARequest node" on page 4349
Use the CORBARequest node to call an external CORBA application over Internet Inter-Orb Protocol (IIOP).

Defining where the CORBARequest node gets the object reference:

You can specify an object reference name either on the CORBARequest node or by using a configurable service.

Before you begin

Before you start:

- Read "Common Object Request Broker Architecture (CORBA)" on page 2145 and "Configurable services" on page 1296 for background information.

About this task

By using configurable services, you can specify the location from which the CORBARequest node gets the object reference without the need to redeploy the message flow. You can also use the configurable service to specify this location for multiple CORBARequest nodes. To pick up new values when a configurable service is created or modified, you must reload the broker or execution group to which the message flow was deployed, by using the **mqsistop** and **mqsistart** commands, or the **mqsireload** command.

The properties of the CORBA configurable services are described in "Configurable services properties" on page 3766.

Creating, changing, reporting, and deleting configurable services

Procedure

- To create a configurable service, use the WebSphere Message Broker Explorer (as described in "Using the WebSphere Message Broker Explorer to work with configurable services" on page 644) or the **mqsicreateconfigurableservice** command shown in the following example.

  This example creates a CORBA configurable service called "myCORBAService" that connects to the CORBA host on the local host on port 2809, and looks for the object reference called "Europe.region/Market.object". (For more information about how to specify the object reference, see "CORBA naming service" on page 2154.)

  ```
  mqsicreateconfigurableservice MB7BROKER -c CORBA -o myCORBAService
  -n namingService,objectReferenceName -v localhost:2809,Europe.region/Market.object
  ```

  To pick up the new values in the configurable service, restart the execution group and message flow.
- To change a configurable service, use the WebSphere Message Broker Explorer or the **mqsichangeproperties** command shown in the following example.

  This example changes the location of the object reference.

  ```
  mqsichangeproperties MB7BROKER -c CORBA -o myCORBAService -n namingService,objectReferenceName
  ```

  To pick up the updated values in the configurable service, restart the execution group and message flow.
- To display all CORBA configurable services, use the WebSphere Message Broker Explorer or the **mqsireportproperties** command shown in the following example.

  ```
  mqsireportproperties MB7BROKER -c CORBA -o AllReportableEntityNames -r
  ```
- You can delete a configurable service that you have created by using the WebSphere Message Broker Explorer or the **mqsideleteconfigurableservice** command shown in the following example.

  ```
  mqsideleteconfigurableservice MB7BROKER -c CORBA -o myCORBAService
  ```

  You can delete only custom-named configurable services; you cannot delete IBM-defined configurable services.

Related concepts:

"Configurable services" on page 1296
Configurable services are typically runtime properties. You can use them to define properties that are related to external services on which the broker relies; for example, an SMTP server or a JMS provider.

"Using the WebSphere Message Broker Explorer to work with configurable services" on page 644
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view, add, modify and delete configurable services.

Related tasks:

"Working with CORBA" on page 2144
Use CORBA nodes to connect to CORBA Internet Inter-Orb Protocol (IIOP) applications.

Related reference:

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"CORBARequest node" on page 4349
Use the CORBARequest node to call an external CORBA application over Internet Inter-Orb Protocol (IIOP).

Working with CICS Transaction Server for z/OS

Use the CICSRequest node to connect to CICS Transaction Server for z/OS applications.

About this task

This section contains the following concept information:

- "CICS Transaction Server for z/OS overview" on page 2173
- "CICS Transaction Server for z/OS connectivity" on page 2174
- "CICS Transaction Server for z/OS two-tier connectivity" on page 2177
- "CICS Transaction Server for z/OS three-tier connectivity" on page 2181
- "COMMAREA or channel data structures" on page 2183
- "CICS Transaction Server for z/OS mirror transactions" on page 2189
- "Local environment overrides for the CICSRequest node" on page 2191

This section contains the following tasks:

- "Connecting to a CICS Transaction Server for z/OS application" on page 2192
- "Defining a CICS Transaction Server for z/OS data structure" on page 2193
- "Preparing the environment for the CICSRequest node" on page 736

- "Securing the connection to CICS Transaction Server for z/OS by using SSL" on page 547
- "Developing a message flow with a CICSRequest node" on page 2199
- "Building a message for the CICSRequest node" on page 2202
- "Processing responses from a CICSRequest node" on page 2204
- "Changing connection information for the CICSRequest node" on page 738
- "Propagating security credentials to CICS Transaction Server for z/OS" on page 2208

Related reference:

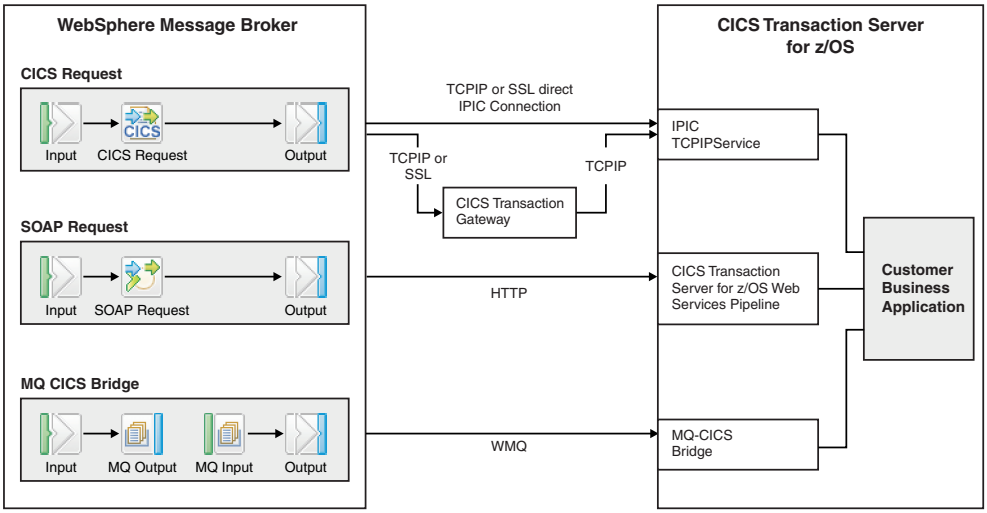
"CICSRequest node" on page 4321
Use the CICSRequest node to call CICS Transaction Server for z/OS programs over TCP/IP-based IP InterCommunications (IPIC) protocol.

CICS Transaction Server for z/OS overview

CICS Transaction Server for z/OS provides general-purpose transaction processing software for z/OS. CICS is a powerful application server that meets the transaction-processing needs of both large and small enterprises.

By using the CICS support that is provided in WebSphere Message Broker you can deploy CICS applications into a service-oriented architecture (SOA). This support keeps your business logic intact and requires little or no change to CICS and CICS applications.

The following diagram shows the layers of communication between WebSphere Message Broker and CICS.



You can use the following methods to connect to CICS:

- **CICSRequest node support**

  WebSphere Message Broker includes a CICSRequest node that supports connectivity to CICS by using the IP InterCommunications (IPIC) protocol, which is available in CICS Transaction Server for z/OS Version 3.2 and later.

  Support includes the following connection methods:

  – A direct connection to CICS (two-tier).
  – A connection to CICS through CICS Transaction Gateway for Multiplatforms (three-tier).

- **CICS Transaction Gateway for Multiplatforms**

  CICS Transaction Gateway for Multiplatforms is a high-performing, secure, and scalable solution that provides workload management and high-availability options for access to CICS. By using standards-based interfaces, CICS Transaction Gateway for Multiplatforms delivers access to new and existing CICS applications.

The CICSRequest node is non-transactional. For further information about the CICSRequest node, see "CICSRequest node" on page 4321.

- **CICS web services support**

  You can use the web services support in WebSphere Message Broker to connect to CICS applications. For further information about web Services, see "Processing Web service messages" on page 1601.

- **CICS-WebSphere MQ Integration**

  CICS has several components that support integration with WebSphere MQ. These components are:

  – The CICS-WebSphere MQ adapter, also known as the CICS-MQ attach (including the CKTI trigger monitor or task initiator). The term CICS-MQ adapter is used to refer to the CICS-WebSphere MQ adapter.

  – The CICS-WebSphere MQ bridge, also known as the CICS 3270 and DPL bridges. The term CICS-MQ bridge is used to refer to the CICS-WebSphere MQ bridge.

For further information about how to use the CICSRequest node to connect WebSphere Message Broker to CICS applications, see "CICS Transaction Server for z/OS connectivity."

For further information about CICS, see the CICS Library web page, which contains links to the CICS Information Center and associated IBM Redbooks publications.

Related concepts:

"CICS Transaction Server for z/OS connectivity"
Use the CICSRequest node to connect WebSphere Message Broker with CICS Transaction Server for z/OS applications.

Related reference:

"CICSRequest node" on page 4321
Use the CICSRequest node to call CICS Transaction Server for z/OS programs over TCP/IP-based IP InterCommunications (IPIC) protocol.

CICS Library web page

CICS Transaction Server for z/OS connectivity:

Use the CICSRequest node to connect WebSphere Message Broker with CICS Transaction Server for z/OS applications.

The CICSRequest node that is available in WebSphere Message Broker provides connectivity to CICS applications by using IP InterCommunications (IPIC) protocol. IPIC is part of a CICS multi-version initiative to provide communications support over TCP/IP as an alternative to that provided over intersystem communication (ISC) and multiregion operation (MRO).

IPIC supports Distributed Program Link (DPL) requests over TCP/IP. The CICSRequest node communicates with CICS by sending Distributed Program Link

(DPL) requests over TCP/IP-based IPIC. IPIC provides a multiplexed single socket connection to CICS Transaction Server for z/OS Version 3.2 and later.

WebSphere Message Broker message flows can use the CICSRequest node to call programs that are running externally in a targeted CICS region. The CICSRequest node can be used by a message flow deployed to any broker platform. The CICSRequest node is non-transactional.

The CICSRequest node supports the following capabilities:

- Commareas.
- Channels and containers.
- Mirror transactions.
- A direct connection to CICS (two-tier).
- A connection to CICS through CICS Transaction Gateway for Multiplatforms (three-tier).
- Local environment overrides for some properties.
- Request timeout.
- A broker APPLID (client APPLID and qualifier) that can identify the application that is connecting to CICS, therefore identifying that WebSphere Message Broker is connected.
- CICSConnection configurable services.
- Secure Sockets Layer (SSL) protocol connection to CICS.
- Security identity.
- User name, or user name and password identity propagation.
- Resource statistics.

You can specify either a COMMAREA data structure or a channel data structure on the CICSRequest node to use as input for linking to CICS programs. The data structure that is specified as input returns the same data structure as output. Channels are an alternative for COMMAREAs, providing relief from the COMMAREA maximum size of `32766` bytes, and allowing greater flexibility in input/output data structures. For more information about using a COMMAREA or channel data structure, see “COMMAREA or channel data structures” on page 2183.

CICS channels hold a number of structures called containers. In WebSphere Message Broker, a CICS channel is represented as a message collection structure. A message collection can hold child messages, each treated as a container by the CICSRequest node. For information about using ESQL to create a message collection, see “Creating a message collection by using ESQL” on page 2758.

If a single container is required for input only, a message collection does not need to be constructed. Instead a regular message can be used, provided the 16-character maximum alphanumeric channel name and the single 16-character maximum alphanumeric container name are specified in the local environment. For more information about using single message mode, see “COMMAREA or channel data structures” on page 2183.

Because it is not possible to know how many containers are in the response, a message collection is always produced as output. However, the CICSRequest node `Result data location` property can be used to reduce the result tree down to a

single message folder, or down to a single field or subtree for output. For information about the `Result data location` property, see “CICSRequest node” on page 4321.

You can add name-value attributes to a message collection to create CICS containers. Name-value attributes in the message collection, apart from *`CollectionName`*, can be used in lieu of full message-folders for simple data. For example, a name-value string attribute can be set in the message collection and used directly by the CICSRequest node without needing to create a message set for the element. For more information about attributes, see “COMMAREA or channel data structures” on page 2183.

Name-value attributes can be produced from containers on output, as well as accepted for input. For information about creating an attribute instead of a message folder from a container, see “CICSRequest node” on page 4321.

You can specify a mirror transaction name on the CICSRequest node for CICS tasks and programs to run under. This grouping greatly assists stat collection, accounting, and aids decision making about task priority. For more information about mirror transactions, see “CICS Transaction Server for z/OS mirror transactions” on page 2189.

The CICSRequest node support in WebSphere Message Broker provides direct communication with CICS (two-tier connection) by sending Distributed Program Link (DPL) requests over TCP/IP-based IPIC, or communication with CICS through CICS Transaction Gateway for Multiplatforms (three-tier connection). For more information about the two-tier and three-tier connection models, see “CICS Transaction Server for z/OS overview” on page 2173 for a high-level overview, or “CICS Transaction Server for z/OS two-tier connectivity” on page 2177 and “CICS Transaction Server for z/OS three-tier connectivity” on page 2181 for detailed conceptual information.

For information about configuring the CICSRequest node to get connection details from a CICSConnection configurable service, see “Changing connection information for the CICSRequest node” on page 738.

You can configure the CICSRequest node or a CICSConnection configurable service to use SSL protocol. For more information, see “Securing the connection to CICS Transaction Server for z/OS by using SSL” on page 547.

You can use the **`mqsisetdbparms`** command to set a user ID and password for the CICSRequest node or CICSConnection configurable service. For detailed information about how to configure CICS security identity support, see “**`mqsisetdbparms`** command” on page 3954.

The CICSRequest node can use an identity that is present on an input message, and propagate it to CICS, by using the `Propagate` property on the security profile that is defined for the node. For more information, see “Propagating security credentials to CICS Transaction Server for z/OS” on page 2208 and “Identity and security token propagation” on page 426.

You can use the CICSRequest node to connect to a CICS application by using a synchronous style of message flow. For details about how to use this node in a message flow, see “CICSRequest node” on page 4321.

Related concepts:

"CICS Transaction Server for z/OS overview" on page 2173
CICS Transaction Server for z/OS provides general-purpose transaction processing software for z/OS. CICS is a powerful application server that meets the transaction-processing needs of both large and small enterprises.

"COMMAREA or channel data structures" on page 2183
CICS Transaction Server for z/OS programs can be linked to by using either a COMMAREA data structure or a channel data structure as input, which return the same data structure as output. The CICSRequest node supports interaction with CICS through COMMAREA or channel data structures.

Related tasks:

"Connecting to a CICS Transaction Server for z/OS application" on page 2192
Connecting to a CICS Transaction Server for z/OS application involves creating a message flow, building a message, and processing the response from the CICSRequest node.

"Changing connection information for the CICSRequest node" on page 738
You can create a configurable service that the CICSRequest node or message flow refers to at run time for connection information, instead of defining the connection properties on the node or the message flow. The advantage being that you can change the host name and performance values without needing to redeploy your message flow.

"Securing the connection to CICS Transaction Server for z/OS by using SSL" on page 547
Configure the CICSRequest node to communicate with CICS Transaction Server for z/OS over the Secure Sockets Layer (SSL) protocol by updating a CICSConnection configurable service or the CICSRequest node to use SSL.

Related reference:

"CICSRequest node" on page 4321
Use the CICSRequest node to call CICS Transaction Server for z/OS programs over TCP/IP-based IP InterCommunications (IPIC) protocol.

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

CICS Transaction Server for z/OS two-tier connectivity:

The CICSRequest node support in WebSphere Message Broker provides direct communication with CICS Transaction Server for z/OS (two-tier connection) by sending Distributed Program Link (DPL) requests over TCP/IP-based IP InterCommunications (IPIC) protocol.

The CICSRequest node also supports communication with CICS through CICS Transaction Gateway for Multiplatforms (three-tier connection). For more information about three-tier connections, see "CICS Transaction Server for z/OS three-tier connectivity" on page 2181.

A direct two-tier connection from WebSphere Message Broker to CICS can be made by using the CICSConnection configurable service or by setting the properties directly on the CICSRequest node.

CICSConnection configurable service connections:

A CICS connection from WebSphere Message Broker is made to a listening TCPIPSERVICE resource in CICS. When that connection is established, the active connection between WebSphere Message Broker and CICS is represented by an IPCONN resource.

Each CICSConnection configurable service results in a separate connection to CICS, so for every configurable service that is being used, there is an IPCONN resource in CICS. The properties of the IPCONN resource determine the properties of the link between WebSphere Message Broker and CICS.

The IPCONN resource that represents a WebSphere Message Broker to CICS connection can be created in two different ways; autoinstall or pre-defined.

Autoinstall:

Autoinstalling a connection means that when WebSphere Message Broker connects, the resource is created, and when WebSphere Message Broker disconnects, the resource is discarded. In this setup, the IPCONN is created from a template IPCONN that is named by a user-replaceable-module (URM), which is named in the TCPIPSERVICE resource. Properties of the IPCONN are based on that template resource.

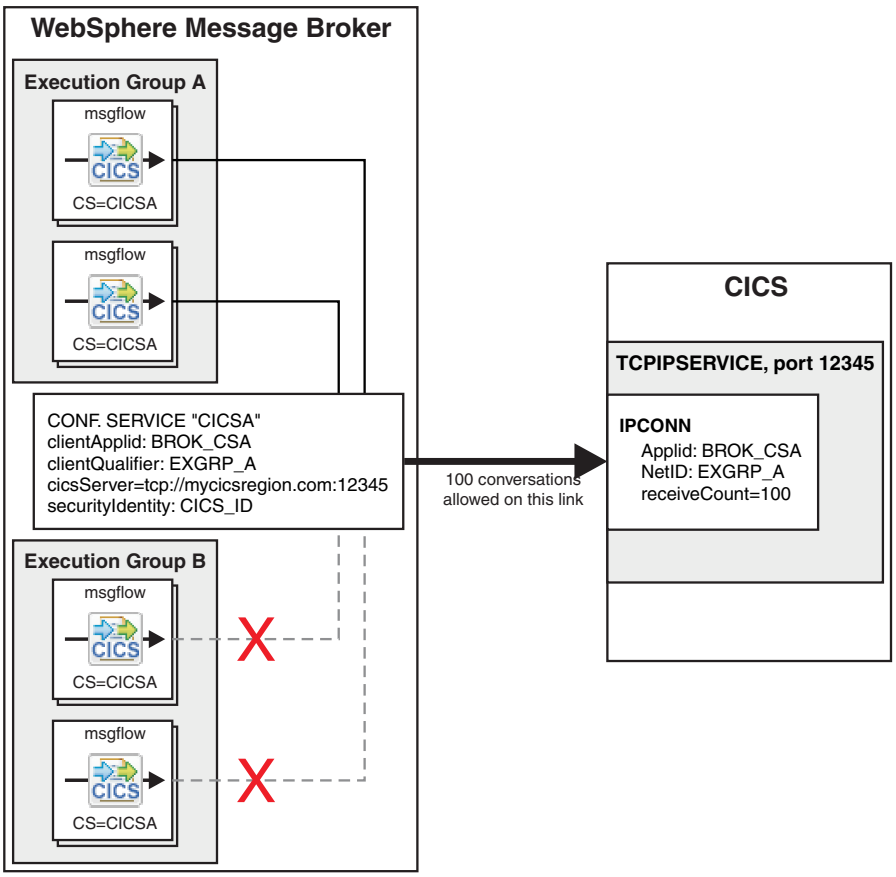
Pre-defined:

Alternatively, the IPCONN can be pre-defined by using standard CICS resource definition mechanisms, such as CICS Explorer, CEDA, or CICSPlex® Systems Manager (CICSPLEX SM). If the IPCONN definition is created in advance, it is matched to the incoming connection by using the IPCONN APPLID and Network ID properties, which correlate to the `clientApplid` and `clientQualifier` properties that can be set on a CICSConnection configurable service.

The advantage of pre-specifying the IPCONN is that you can tightly control the properties of incoming connections, including the security properties and the number of simultaneous requests. However the following rules apply:

- Do not configure different execution groups to use the same CICSConnection `clientApplid` and `clientQualifier` combination to connect to the same CICS region. An IPCONN is tied to WebSphere Message Broker through the CICSConnection configurable service properties `clientApplid` and `clientQualifier`. If this is attempted, only the first configurable service successfully connects.
- Do not specify a host name and port when defining the IPCONN resource in CICS. These fields are used for connections between CICS regions only, they must not be set for WebSphere Message Broker connections.

The following diagram shows how WebSphere Message Broker can directly connect to CICS by using a CICSConnection configurable service.



The two-tier direct CICS connection model is based on the following rules:

- Each configurable service name results in a separate connection to CICS.
- The CICSConnection configurable service must only be used from one execution group, because any further execution groups attempting to use the same configurable service thereafter fail to connect.
- The `clientApplid` and `clientQualifier` properties in the configurable service are used to find the IPCONN resource in CICS. A chosen `clientApplid` and `clientQualifier` combination must be unique to the CICS region. Only one IPCONN resource can exist with that combination.
- More than one message flow instance can use the CICS connection, however each request that goes through a CICSRequest node uses a conversation on the connection for the duration of the request.

When defining an IPCONN resource in CICS, consider the following properties:

- **CICS APPLID and Network ID**

  The CICS APPLID and Network ID properties must match the CICSConnection configurable service `clientApplid` and `clientQualifier` properties.
- **CICS host name and Port number**

  The CICS host name and port properties must be used for connections between CICS regions only, they must not be set for WebSphere Message Broker connections.

- **CICS TCPIPSERVICE**

  IPCONNs are owned by a parent TCPIPSERVICE resource in CICS.
- **CICS Receivecount**

  The CICS Receivecount property controls the number of simultaneous requests that can be performed over the connection. The number of simultaneous requests defaults to 100 for autoinstalled connections.
- **CICS Sendcount**

  The Sendcount property must be set to 0 because the Sendcount property is used for CICS connections only, and must not be used for WebSphere Message Broker connections.
- **CICS LINKAUTH**

  The CICS LINKAUTH property controls how the link security is managed. To use a resource in CICS, two security checks are performed; the "flowed" user, which checks the security credentials that are sent from WebSphere Message Broker, and the "link" user, which must also have permission for the resource. Both user IDs must have permission to use the resource before the request is granted. The link user ID is given low privileges, which means that even if the flowed user has many permissions, the link user ID can be used to cap the privilege of the connection. If LINKAUTH is set to SECUSER, the SECURITYNAME field is used to specify the link user ID. If set to CERTUSER, the link user is determined from an SSL client certificate that is mapped by RACF.
- **CICS USERAUTH**

  The CICS USERAUTH property determines how the flowed user security is configured. If USERAUTH is set to "LOCAL" or "DEFAULTUSER", no user ID or password is to be sent to CICS on a request. This means that all requests use the CICS region ID. If USERAUTH is set to "IDENTIFY", user IDs are flowed without a password. If USERAUTH is set to "VERIFY", user IDs and passwords are required.

Each CICSRequest node in a message flow acts as a request on one of the connections to CICS. Which connection is used is determined by the configurable service that is used.

For more information about configuring the CICSRequest node to get connection details from a CICSConnection configurable service, see “Changing connection information for the CICSRequest node” on page 738.

You can configure the CICSRequest node or a CICSConnection configurable service to use SSL protocol. For more information, see “Securing the connection to CICS Transaction Server for z/OS by using SSL” on page 547.

CICSRequest node connections:

If a CICSConnection configurable service is not specified on the CICSRequest node, and a host name is used directly in the `CICS server` property, the request shares a connection with other resources that have specified the same `CICS server` URL. The first CICSRequest node to be used opens the connection to CICS, regardless of whether a URL or a configurable service is specified in the `CICS server` property.

Related concepts:

"CICS Transaction Server for z/OS three-tier connectivity"
The CICSRequest node support in WebSphere Message Broker can provide communication with CICS Transaction Server for z/OS through CICS Transaction Gateway for Multiplatforms (three-tier connection).

"CICS Transaction Server for z/OS connectivity" on page 2174
Use the CICSRequest node to connect WebSphere Message Broker with CICS Transaction Server for z/OS applications.

Related tasks:

"Connecting to a CICS Transaction Server for z/OS application" on page 2192
Connecting to a CICS Transaction Server for z/OS application involves creating a message flow, building a message, and processing the response from the CICSRequest node.

"Changing connection information for the CICSRequest node" on page 738
You can create a configurable service that the CICSRequest node or message flow refers to at run time for connection information, instead of defining the connection properties on the node or the message flow. The advantage being that you can change the host name and performance values without needing to redeploy your message flow.

"Securing the connection to CICS Transaction Server for z/OS by using SSL" on page 547
Configure the CICSRequest node to communicate with CICS Transaction Server for z/OS over the Secure Sockets Layer (SSL) protocol by updating a CICSConnection configurable service or the CICSRequest node to use SSL.

Related reference:

"CICSRequest node" on page 4321
Use the CICSRequest node to call CICS Transaction Server for z/OS programs over TCP/IP-based IP InterCommunications (IPIC) protocol.

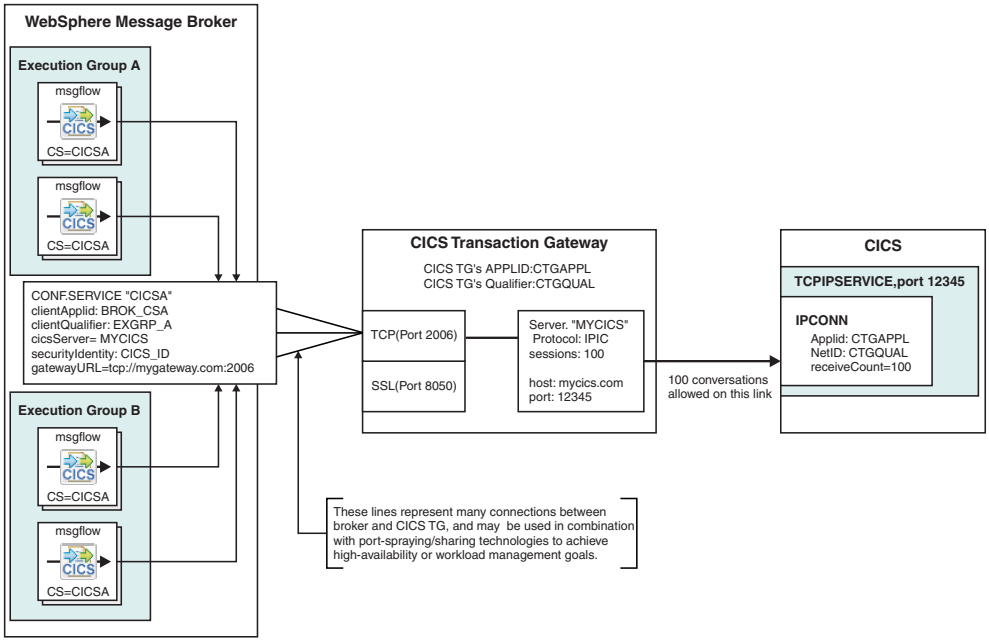
CICS Transaction Server for z/OS three-tier connectivity:

The CICSRequest node support in WebSphere Message Broker can provide communication with CICS Transaction Server for z/OS through CICS Transaction Gateway for Multiplatforms (three-tier connection).

The CICSRequest node also supports direct communication with CICS (two-tier connection) by sending Distributed Program Link (DPL) requests over TCP/IP-based IP InterCommunications (IPIC) protocol. For more information about two-tier connections, see "CICS Transaction Server for z/OS two-tier connectivity" on page 2177.

A three-tier connection from WebSphere Message Broker to CICS through CICS Transaction Gateway for Multiplatforms can be made by configuring the `cicsServer` and `gatewayURL` CICSConnection configurable service properties.

The following diagram shows how WebSphere Message Broker can make a connection to CICS through CICS Transaction Gateway for Multiplatforms by using the CICSConnection configurable service.



For more information about configuring the `cicsServer` and `gatewayURL` CICSConnection configurable service properties to make a three-tier connection, see "Configurable services properties" on page 3766.

The three-tier connection to CICS through CICS Transaction Gateway for Multiplatforms connection model is based on the following rules:

- The number of connections that can be made to CICS Transaction Gateway for Multiplatforms is determined by the maximum number of simultaneous CICS requests in progress. Ensure that the Connection Manager and Worker Thread resources in your CICS Transaction Gateway for Multiplatforms deployment have sufficient capacity to handle the number of required WebSphere Message Broker connections.
- The `clientApplid` and `clientQualifier` properties in the CICSConnection configurable service can be used to identify the broker connection within CICS Transaction Gateway for Multiplatforms. In addition, any CICS tasks that are started through the configurable service each contain point of origin information, including the specified client APPLID and qualifier, which you can find in the CICS task association data.
- Unlike two-tier connections, there is no restriction about sharing client APPLIDs between execution groups because the connection is made to CICS Transaction Gateway for Multiplatforms, and not to CICS directly.

For more information about configuring the CICSRequest node to get connection details from a CICSConnection configurable service, see "Changing connection information for the CICSRequest node" on page 738.

Related concepts:

"CICS Transaction Server for z/OS two-tier connectivity" on page 2177
The CICSRequest node support in WebSphere Message Broker provides direct communication with CICS Transaction Server for z/OS (two-tier connection) by sending Distributed Program Link (DPL) requests over TCP/IP-based IP InterCommunications (IPIC) protocol.

"CICS Transaction Server for z/OS connectivity" on page 2174
Use the CICSRequest node to connect WebSphere Message Broker with CICS Transaction Server for z/OS applications.

Related tasks:

"Connecting to a CICS Transaction Server for z/OS application" on page 2192
Connecting to a CICS Transaction Server for z/OS application involves creating a message flow, building a message, and processing the response from the CICSRequest node.

"Changing connection information for the CICSRequest node" on page 738
You can create a configurable service that the CICSRequest node or message flow refers to at run time for connection information, instead of defining the connection properties on the node or the message flow. The advantage being that you can change the host name and performance values without needing to redeploy your message flow.

"Securing the connection to CICS Transaction Server for z/OS by using SSL" on page 547
Configure the CICSRequest node to communicate with CICS Transaction Server for z/OS over the Secure Sockets Layer (SSL) protocol by updating a CICSConnection configurable service or the CICSRequest node to use SSL.

Related reference:

"CICSRequest node" on page 4321
Use the CICSRequest node to call CICS Transaction Server for z/OS programs over TCP/IP-based IP InterCommunications (IPIC) protocol.

COMMAREA or channel data structures:

CICS Transaction Server for z/OS programs can be linked to by using either a COMMAREA data structure or a channel data structure as input, which return the same data structure as output. The CICSRequest node supports interaction with CICS through COMMAREA or channel data structures.

Channels are a modern alternative for COMMAREAs, providing relief from the COMMAREA maximum size of `32766` bytes, and allowing greater flexibility in input/output data structures.

- "COMMAREAs"
- "Channels" on page 2184

COMMAREAs: If using a COMMAREA as the input data structure for communicating with CICS, the CICSRequest node takes a portion of the Input Body, as defined in the CICSRequest node Request properties, and sends it to CICS as the COMMAREA.

The returning COMMAREA is then put into the Output tree and replaces the existing Body at the location that is defined in the CICSRequest node Result properties. The COMMAREA can then be configured for parsing by using the CICSRequest node Response Message Parsing properties.

When defining a COMMAREA data structure as input, you must ensure that the CICSRequest node `Commarea length` property value is large enough to contain the input request data, or the output response data, but that it does not exceed the maximum value of `32767` bytes. If the `Commarea length` value is not large enough to be used for the response data, or the request data, a memory leak occurs in CICS. The size of the COMMAREA cannot be changed by the CICS program. If the

serialized request data is larger than the Commarea length, the data is truncated to the Commarea length. You can obtain the Commarea length value from the CICS administrator or developer.

The default value for the CICSRequest node data structure basic property is Commarea.

For more information about using a COMMAREA data structure as input, see "Defining a CICS Transaction Server for z/OS data structure" on page 2193, "Developing a message flow with a CICSRequest node" on page 2199, and "Building a message for the CICSRequest node" on page 2202.

Channels: CICS channels hold a number of structures called containers. Containers hold the business information that is accessed by the target CICS program. Each container can hold up to 2 GB of data, and channels can have as many containers within them as required, which provides flexibility in terms of the size and layout of data. Each container has a 16-character maximum alphanumeric container name, which is unique within the channel, and is used as the mechanism to retrieve the contents of the container from the channel.

There are two types of container; character or binary. The type of container can affect the data conversion between WebSphere Message Broker and CICS, however the container type does not have any impact on the format of the information that can be put into the container.

Character containers
Character containers are more likely to be individual strings or discrete data items, but can also be mapped structures, however it is important to remember that data conversion is applied to the data in the container. When a character container is constructed, the source coded character set ID (CCSID) information about the container is sent to CICS as metadata. The CICS program uses the GET CONTAINER application programming interface (API) call to convert the metadata to the default CCSID of the region, unless another CCSID is provided. The CICS program then places the container back into the channel, and the data is converted ready for the WebSphere Message Broker application to retrieve and use.

Binary containers
Binary containers can be mapped by using a COBOL copybook structure or they might be discrete values. Data conversion is not applied to the data in a binary container, therefore the data in a binary container is sent to CICS and retrieved from CICS in original form only.

Unlike COMMAREA structures, the size of the response channel does not need to resemble the request, whereas COMMAREAs must allow for the size of the response in the request.

Channels and containers in CICS

In the following example diagram, the CICS channel has two containers; CustomerName and Order.

Channel Name:		MyChannel		
CustomerName	CHAR	'Joe Bloggs'		
Order	BIN	100	Apples	0.39
...	...	...		

`CustomerName` is a character (CHAR) container that contains a single character string; `Joe Bloggs`. Because `CustomerName` is a character container, data conversion can be applied to the data in the container. `Order` is a binary (BIN) container that might be created by using a COBOL copybook structure or C header file, which you can then import to populate your message set with message definitions. The following example copybook describes the binary layout of the data that the CICS program expects to receive:

```
01 ORDER_STRUCTURE.
          03 QTY                 COMP-1.
          03 ITEM                PIC X(10).
          03 PRICE               PIC S9(9).
```

The target CICS program can retrieve both of these containers from the channel by providing the name of the container when using the `GET CONTAINER` API. When the data is provided to the CICS program, the program processes the data however it chooses. For example, the program can place other containers into the channel to provide a response to the called container by using the `PUT CONTAINER` API.

Channels and containers in WebSphere Message Broker

In WebSphere Message Broker, a CICS channel is represented as a message collection structure. A message collection can hold child messages, each treated as a container by the CICSRequest node. A message collection structure is used as both input and output to the CICSRequest node when using a channel data structure. For more information about message collections, see “Message collections” on page 2755, and for information about creating a message collection, see “Creating a message collection by using ESQL” on page 2758.

The message collection name is used to name the channel. The name of the child message in the message collection is used as the name of the container in the channel, and must be unique. If the child message name in the message collection is not unique, the request is rejected in CICS.

The following table shows the channel and container to message collection and child message mapping:

CICS	WebSphere Message Broker
Channel name	Message collection name
Container name (must be unique to the parent channel)	Child message name (must be unique to the message collection)

Name-value attributes

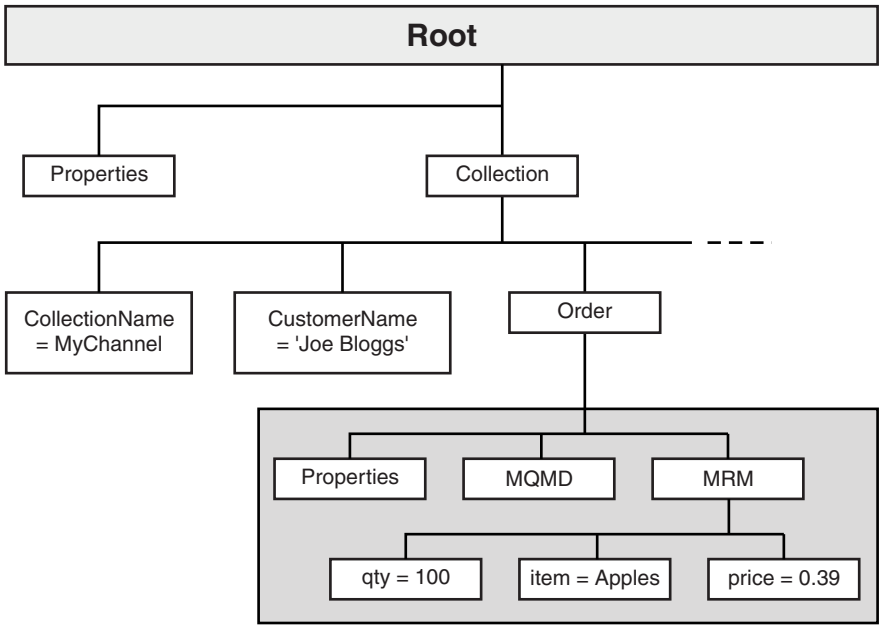
WebSphere Message Broker supports adding name-value attributes to a message collection to create a container. A message collection can have zero or more

attributes. The name of an attribute must be unique within a message collection. A standard attribute for the message collection is an attribute called *CollectionName*.

You can add name-value attributes to a message collection to create CICS containers. Name-value attributes in the message collection, apart from *CollectionName*, can be used in lieu of full message-folders for simple data. For example, a name-value string attribute can be set in the message collection and used directly by the CICSRequest node without needing to create a message set for the element.

Name-value attributes can be produced from containers on output, as well as accepted for input. For information about creating an attribute instead of a message folder from a container, see "CICSRequest node" on page 4321.

In the following example diagram, the CICS channel is represented by a message collection named `Collection`. `Collection` holds two containers that are represented by child messages named `CustomerName` and `Order`. *CollectionName* and `CustomerName` are both name-value attributes, however the *CollectionName* attribute is not treated as a container by the CICSRequest node, and is therefore not sent to CICS.



If the `CustomerName` attribute is to be treated as a character container by the CICSRequest node, the LocalEnvironment must reflect this.

LocalEnvironment

Each child message in a message collection is treated as having a default type of binary, which determines whether the data is converted to the CCSID in the CICS region. However, you can dynamically override this value to character, on a per message basis, in the local environment. For example, you can set the following value under `LocalEnvironment.Destination.CICS.RequestChannel.Containers`:

```
SET OutputLocalEnvironment.Destination.CICS.RequestChannel.Containers.<myContainerName> = CHARACTER;
```

When a message collection is emitted from the CICSRequest node following the request, the LocalEnvironment contains the returned type information for the containers. For example, when the response channel comes back from CICS, the LocalEnvironment shows the types of the containers that have come back in the following location:
`LocalEnvironment.CICS.ResponseChannel.Containers.<myContainerName> = CHARACTER`

The channel name, which has a 16-character alphanumeric limit, can be overridden as follows:

```
SET OutputLocalEnvironment.Destination.CICS.RequestChannel.ChannelName = <myNewChannelName>;
```

If a single container is required for input only, a message collection does not need to be constructed. Instead a regular message can be used, however, you must set the 16-character maximum alphanumeric `ChannelName` in the LocalEnvironment. The 16-character maximum alphanumeric `SingleMessageContainerName` that needs to be created must also be provided in the following location:

```
SET OutputLocalEnvironment.Destination.CICS.RequestChannel.SingleMessageContainerName = <mySingleM
```

Because a message collection allows each container in the channel to be modeled as a separate message, each message has its own structure and parsing options. For example, one container might be XML and another might be based on a copybook, which can be represented by using XMLNSC and MRM messages within a message collection.

Each child message in the message collection contains message domain, set, type, format, CCSID, and encoding information in the `Properties` folder that is associated with the child message, which is serialized into a byte stream and sent to CICS. Each child message folder within the message collection that is being sent to CICS is serialized at the level of the last child of the message property domain. Not all CICS containers require a message set to represent them.

In the previous example, the `Order` container can be represented as MRM, and a message set can be created from the copybook ORDER_STRUCTURE to represent it. The returning channel is converted into a message collection, where every child message in the message collection represents a container from the channel. Child messages in the message collection are mapped to a list of message domain, set, type, format, CCSID, and encoding information by using the child message name, however CCSID, and encoding information are ignored for character messages. If a mapping cannot be found in the message, a default mapping can be provided.

Because it is not possible to know how many containers are in the response, a message collection is always produced as output.

You can use the CICSRequest node Response Message Parsing properties to map a returning container to a message domain, set, type, format, CCSID, and encoding information. In particular, the `Result data location` property can be used to reduce the result tree down to a single message folder, or down to a single field or subtree for output. For information about the `Result data location` property, see "CICSRequest node" on page 4321.

The following details must be specified for the message to arrive at the CICSRequest node input terminal and be processed into a series of containers, and for those containers in the response channel to be placed back in the message as it leaves the node:

- The channel must be given a 16 character maximum alphanumeric name.

  Because the channel is represented by a message collection in WebSphere Message Broker, you can create the channel name by setting the message collection name. Message collection names are set by using the *CollectionName* attribute. For more information about creating a message collection and setting the message collection name, see "Creating a message collection by using ESQL" on page 2758.
- The following details must be specified for each container in the channel:
  - A 16 character maximum alphanumeric name.

    Because a container is represented by a child message in WebSphere Message Broker, you can create a container name by setting the child message name. For more information about creating a message collection and setting the child message name, see "Creating a message collection by using ESQL" on page 2758.
  - A container type; for example, `binary` or `character`.
  - A directory to use to place the response data into.

A sample is provided that demonstrates how to call a channel-based CICS program by creating and populating a message collection for the CICSRequest node, and how to process the collection after the call. For more information, see CICS Transaction Server for z/OS Channel Connectivity.

Related concepts:

"CICS Transaction Server for z/OS connectivity" on page 2174
Use the CICSRequest node to connect WebSphere Message Broker with CICS Transaction Server for z/OS applications.

"Local environment tree structure" on page 1056
The local environment tree is a part of the logical message tree in which you can store information while the message flow processes the message.

"Message collections" on page 2755
A message collection is a single message that contains multiple messages derived from one or more sources.

Related tasks:

"Connecting to a CICS Transaction Server for z/OS application" on page 2192
Connecting to a CICS Transaction Server for z/OS application involves creating a message flow, building a message, and processing the response from the CICSRequest node.

"Accessing the local environment tree" on page 2463
The local environment tree has its own correlation name, LocalEnvironment, and you must use this name in all ESQL statements that refer to or set the content of this tree.

"Using message collections" on page 2755
A message collection is a single message that contains multiple messages derived from one or more sources. You can use a Collector node to group together messages from one or more sources into a message collection, so that they can be processed together by downstream nodes.

"Creating a message collection by using ESQL" on page 2758
A message collection can be constructed by using ESQL. Using a message collection is useful if messages must be grouped together for parsing, or if the message collection must be constructed to represent a particular data structure, such as a CICS Transaction Server for z/OS channel data structure.

"Setting the collection name" on page 2774
You can set a default name, or use a correlation string, for the name of your message collections, by using the `Collection name` property on the Collector node.

Related reference:

"CICSRequest node" on page 4321
Use the CICSRequest node to call CICS Transaction Server for z/OS programs over TCP/IP-based IP InterCommunications (IPIC) protocol.

CICS Transaction Server for z/OS mirror transactions:

You can use a mirror transaction to group CICS Transaction Server for z/OS tasks and programs together. This grouping greatly assists stat collection, accounting, and aids decision making about task priority.

You can specify a mirror transaction name on the CICSRequest node for CICS tasks and programs to run under. For example, if each department in a business has a different mirror transaction name, work can be tracked back to the correct source, and decisions about task priority and quality of service (QoS) can be made in CICS. Potentially, different security privileges might be available depending on the transaction name chosen. Or the transaction name might be used as a way of indicating the origin of the task. Alternatively, a mirror transaction can be used to denote whether code page translation of the commarea data is required.

There are two ways that a mirror transaction can be specified:

Specifying a mirror transaction name that corresponds with a defined TRANSACTION resource in CICS:

You can specify a mirror transaction name by configuring the `Mirror transaction ID` property on the CICSRequest node Basic tab, however the `Mirror transaction ID` property value that you specify must correspond to a defined TRANSACTION resource in CICS. For example, if you have a defined TRANSACTION resource of `ATRN` in CICS, and you want tasks and programs to run under that transaction, you must configure `ATRN` as the `Mirror transaction ID` property value.

When the message flow containing the configured CICSRequest node is deployed, any CICS programs that are started thereafter appear in CICS as running under the specified mirror transaction.

Specifying a weaker form of mirror transaction that does not require a TRANSACTION resource to be defined in CICS:

You can use a weaker form of mirror transaction that does not change the TRANSACTION resource, but instead sets a variable called `EIBTRNID`, which is available to the called program. You can configure the `EIBTRNID` variable to tell the program what TRANSACTION resource it is running under, without the TRANSACTION resource being defined in CICS.

For example, you can specify this weaker form of mirror transaction by configuring the `Mirror transaction ID` property with the name of the required TRANSACTION resource; for example `ATRN`, and by selecting the `Set EIBTRNID only` property on the CICSRequest node Basic tab.

When the message flow containing the configured CICSRequest node is deployed, any CICS programs that are started thereafter appear in CICS as running under the specified mirror transaction.

If the value of the CICSRequest node `Mirror transaction ID` property is not set, the mirror transaction name defaults to `CPMI` if called by a distributed platform, or `CSMI` if called by a z/OS system.

The following table describes the mirror transaction handling that is applied depending on the configuration of the `Mirror transaction ID` and `Set EIBTRNID only` CICSRequest node properties. Where `ATRN` is an example of a user-defined transaction name:

Mirror transaction ID property value	**Set EIBTRNID only property value**	**Task and programs run under defined TRANSACTION resource:**	**EIBTRNID is:**
Blank	Cleared	`CPMI` if called by a distributed platform, or `CSMI` if called by a z/OS system	`CPMI` if called by a distributed platform, or `CSMI` if called by a z/OS system
`ATRN`	Cleared	`ATRN`	`ATRN`
`ATRN`	Selected	`CPMI` if called by a distributed platform, or `CSMI` if called by a z/OS system	`ATRN`

If you are considering whether to use a mirror transaction as a way of finding the point of origin of your data, using the CICS task association data might be a better alternative. All tasks that are initiated in CICS over IP InterCommunications (IPIC) protocol contain origin information, including source Internet Protocol (IP) and APPLID information.

The CICS mirror transaction properties can be changed by configuring the properties directly on the CICSRequest node, by using the **mqsiapplybaroverride** command, or by dynamically overriding these property values with elements in the message tree, on a per message basis, in the local environment. For more information about dynamically overriding CICSRequest node values, see "Local environment overrides for the CICSRequest node" on page 2191 and for information about the **mqsiapplybaroverride** command, see "Configurable properties" on page 3687.

Related concepts:

"CICS Transaction Server for z/OS connectivity" on page 2174
Use the CICSRequest node to connect WebSphere Message Broker with CICS Transaction Server for z/OS applications.

"Local environment tree structure" on page 1056
The local environment tree is a part of the logical message tree in which you can store information while the message flow processes the message.

Related tasks:

"Connecting to a CICS Transaction Server for z/OS application" on page 2192
Connecting to a CICS Transaction Server for z/OS application involves creating a message flow, building a message, and processing the response from the CICSRequest node.

"Accessing the local environment tree" on page 2463
The local environment tree has its own correlation name, LocalEnvironment, and you must use this name in all ESQL statements that refer to or set the content of this tree.

Related reference:

"CICSRequest node" on page 4321
Use the CICSRequest node to call CICS Transaction Server for z/OS programs over TCP/IP-based IP InterCommunications (IPIC) protocol.

"Configurable properties" on page 3687
Some properties of message flow nodes are configurable and can be changed by using the **mqsiapplybaroverride** command. The following table maps the message flow node properties to the corresponding properties of the **mqsiapplybaroverride** command.

Local environment overrides for the CICSRequest node:

When you use the CICSRequest node in a message flow, you can override some of its properties with elements in the message tree.

You can dynamically override values, on a per message basis, in the local environment. You can override the CICS Transaction Server for z/OS program that you are calling, the COMMAREA length, mirror transactions, and the processing of the response message.

You can set the following values under `LocalEnvironment.Destination.CICS`.

Setting	Description
CICSProgramName	Overrides the `Program name` property on the node; for example: `SET OutputLocalEnvironment.Destination.CICS.CICSProgramName = 'progx';` You can override the location of this value by using a property on the CICSRequest node.
CICSCommareaLen	Overrides the `Commarea length` property on the node; for example: `SET OutputLocalEnvironment.Destination.CICS.CICSCommareaLen = 100;`
mirrorTran	Overrides the `Mirror transaction ID` property on the node; for example: `SET OutputLocalEnvironment.Destination.CICS.mirrorTran = 'ATRN';`
eibtrnidOnly	Overrides the `Set EIBTRNID only` property on the node; for example: `SET OutputLocalEnvironment.Destination.CICS.eibtrnidOnly = 'TRUE';` Valid values are `TRUE`, `FALSE`, `YES`, or `NO`. Where `TRUE` or `YES` represents a selected check box for the `Set EIBTRNID only` property in the WebSphere Message Broker Toolkit. Where `FALSE` or `NO` represents a cleared check box in the WebSphere Message Broker Toolkit.

You can set the following values under `LocalEnvironment.Destination.CICS.Response`.

Setting	Description
messageDomain	Overrides the `Message domain` property on the node; for example: `SET OutputLocalEnvironment.Destination.CICS.Response.messageDomain = 'MRM';`
messageSet	Overrides the `Message set` property on the node; for example: `SET OutputLocalEnvironment.Destination.CICS.Response.messageSet = '{com.test}:MyMessageSet';`
messageType	Overrides the `Message type` property on the node; for example: `SET OutputLocalEnvironment.Destination.CICS.Response.messageType = 'messageA';`
messageFormat	Overrides the `Message format` property on the node; for example: `SET OutputLocalEnvironment.Destination.CICS.Response.messageFormat = 'XML1';`

Setting	Description
messageCCSID	Overrides the `Message coded character set ID` property on the node; for example: `SET OutputLocalEnvironment.Destination.CICS.Response.messageCCSID = 500;`
messageEncoding	Overrides the `Message encoding` property on the node; for example: `SET OutputLocalEnvironment.Destination.CICS.Response.messageEncoding = 785;`

You can also set LocalEnvironment values for CICS channels and containers. For more information, see "COMMAREA or channel data structures" on page 2183.

Related concepts:

"Local environment tree structure" on page 1056
The local environment tree is a part of the logical message tree in which you can store information while the message flow processes the message.

"CICS Transaction Server for z/OS overview" on page 2173
CICS Transaction Server for z/OS provides general-purpose transaction processing software for z/OS. CICS is a powerful application server that meets the transaction-processing needs of both large and small enterprises.

Related tasks:

"Accessing the local environment tree" on page 2463
The local environment tree has its own correlation name, LocalEnvironment, and you must use this name in all ESQL statements that refer to or set the content of this tree.

"Connecting to a CICS Transaction Server for z/OS application"
Connecting to a CICS Transaction Server for z/OS application involves creating a message flow, building a message, and processing the response from the CICSRequest node.

Related reference:

"CICSRequest node" on page 4321
Use the CICSRequest node to call CICS Transaction Server for z/OS programs over TCP/IP-based IP InterCommunications (IPIC) protocol.

Connecting to a CICS Transaction Server for z/OS application

Connecting to a CICS Transaction Server for z/OS application involves creating a message flow, building a message, and processing the response from the CICSRequest node.

About this task

To connect to a CICS application, complete the following steps.

Procedure

1. Create a message flow project and a message set project. See "Creating a message flow project" on page 1425 and "Working with a message set project" on page 2838 for more information.
2. Define the COMMAREA data structure as a message set, as described in "Defining a CICS Transaction Server for z/OS data structure" on page 2193.
3. Configure IP InterCommunications (IPIC) protocol on CICS, as described in "Preparing the environment for the CICSRequest node" on page 736.
4. Optional: Configure the CICSRequest node to communicate with CICS Transaction Server for z/OS over the Secure Sockets Layer (SSL) protocol, as described in "Securing the connection to CICS Transaction Server for z/OS by using SSL" on page 547.

5. Develop a message flow that contains a CICSRequest node, as described in "Developing a message flow with a CICSRequest node" on page 2199.
6. Build a message to send to the CICSRequest node, as described in "Building a message for the CICSRequest node" on page 2202.
7. Process the response from the CICSRequest node, as described in "Processing responses from a CICSRequest node" on page 2204.

What to do next

You can change the connection details for the CICSRequest node by using the CICSConnection configurable service, as described in "Changing connection information for the CICSRequest node" on page 738.

The CICSRequest node can also use an identity that is present on an input message, and propagate it to CICS, by using the `Propagate` property on the security profile that is defined for the node. For more information, see "Propagating security credentials to CICS Transaction Server for z/OS" on page 2208

Related concepts:

"CICS Transaction Server for z/OS overview" on page 2173
CICS Transaction Server for z/OS provides general-purpose transaction processing software for z/OS. CICS is a powerful application server that meets the transaction-processing needs of both large and small enterprises.

"CICS Transaction Server for z/OS connectivity" on page 2174
Use the CICSRequest node to connect WebSphere Message Broker with CICS Transaction Server for z/OS applications.

Related tasks:

"Creating a message flow project" on page 1425
A message flow project is a container for message flows; you must create a project before you can create a message flow.

"Creating a message set" on page 2842
Use the New Message Set wizard to create a message set.

Related reference:

"CICSRequest node" on page 4321
Use the CICSRequest node to call CICS Transaction Server for z/OS programs over TCP/IP-based IP InterCommunications (IPIC) protocol.

Defining a CICS Transaction Server for z/OS data structure:

Mapped data structures, such as COBOL copybooks or C structures, can be used to define an MRM message definition for a CICS Transaction Server for z/OS COMMAREA or a channel container.

Before you begin

Before you start:

- Create a message flow project and a message set project. See "Creating a message flow project" on page 1425 and "Working with a message set project" on page 2838 for more information.
- Optional: If you are using a COMMAREA data structure, read the COMMAREA concept information that is defined in the CICS Transaction Server for z/OS Version 4.1 Information Center online.

- Read the COMMAREA and channel concept information that is defined in "COMMAREA or channel data structures" on page 2183.
- Create a message definition file by using the New Message Definition File wizard in the WebSphere Message Broker Toolkit.
  - If you are using a C header file, see "Importing from C" on page 2934.
  - If you are using a COBOL data structure, see "Importing from COBOL copybooks" on page 2937.
  - For information about other data structures, see "Working with data structures" on page 2930.

About this task

After you have created the message definition, complete the following steps.

Procedure

1. Select the `Data structure` property value on the **Basic** tab of the CICSRequest node in accordance with the targeted CICS program. For example, if the target program is channel-based, select `Channel` as the data structure.
2. Ensure that the properties on the **Response Message Parsing** tab of the CICSRequest node specify the output COMMAREA or container structure.
3. Check that the upstream nodes are configured to provide the correct input structure.
4. Optional: If you are using a COMMAREA data structure, ensure that the CICSRequest node `Commarea length` property is large enough to hold the serialized input or output structures to avoid a memory leak in the CICS application.

What to do next

Next: If you are using a channel data structure, create a message collection to represent the channel data structure, as described in "Creating a message collection by using ESQL" on page 2758.

Related concepts:

"CICS Transaction Server for z/OS overview" on page 2173
CICS Transaction Server for z/OS provides general-purpose transaction processing software for z/OS. CICS is a powerful application server that meets the transaction-processing needs of both large and small enterprises.

"CICS Transaction Server for z/OS connectivity" on page 2174
Use the CICSRequest node to connect WebSphere Message Broker with CICS Transaction Server for z/OS applications.

"COMMAREA or channel data structures" on page 2183
CICS Transaction Server for z/OS programs can be linked to by using either a COMMAREA data structure or a channel data structure as input, which return the same data structure as output. The CICSRequest node supports interaction with CICS through COMMAREA or channel data structures.

Related tasks:

"Connecting to a CICS Transaction Server for z/OS application" on page 2192
Connecting to a CICS Transaction Server for z/OS application involves creating a message flow, building a message, and processing the response from the CICSRequest node.

"Preparing the environment for the CICSRequest node" on page 736
Before you can use the CICSRequest node, you must configure IP InterCommunications (IPIC) protocol on the target CICS Transaction Server for z/OS.

"Developing a message flow with a CICSRequest node" on page 2199
To connect to a CICS Transaction Server for z/OS application, create a message flow that contains a CICSRequest node.

Related reference:

"CICSRequest node" on page 4321
Use the CICSRequest node to call CICS Transaction Server for z/OS programs over TCP/IP-based IP InterCommunications (IPIC) protocol.

Preparing the environment for the CICSRequest node:

Before you can use the CICSRequest node, you must configure IP InterCommunications (IPIC) protocol on the target CICS Transaction Server for z/OS.

Before you begin

Before you start:

Read "CICS Transaction Server for z/OS connectivity" on page 2174 for background information.

About this task

The CICSRequest node can send IPIC requests over TCP/IP to CICS Transaction Server for z/OS Version 3.2 and later. Complete the following steps on CICS to perform this configuration:

Procedure

1. Set the System Initialization (SIT) parameter **TCPIP=YES**.
2. Define the TCP/IP address and host name for CICS. By default, they are defined in the PROFILE.TCPIP and TCPIP.DATA data sets.
3. Add a TCP/IP listener to CICS by using the following **CEDA** command to define a TCPIPSERVICE resource in a group:

   ```
   CEDA DEF TCPIPSERVICE(service-name) GROUP(group-name)
   ```

   Ensure that the group in which you define the service is in the GRPLIST system initialization parameter, so that the listener starts when CICS is started. Key fields are explained as follows:

   POrtnumber:
   The port on which the TCP/IP service listens.

   PROtocol:
   The protocol of the service is IPIC.

   TRansaction:
   The transaction that CICS runs to handle incoming IPIC requests. Set the field to CISS, which is the default.

   Backlog:
   The Backlog field is the number of TCP/IP connection requests are sent to CICS, which are placed in a TCP/IP queue to be assigned an IPCONN connection to CICS. The default value is 1. Do not use a value

of 0; a value of 0 indicates that no TCP/IP connection requests are to be assigned an IPCONN connection to CICS, which disables incoming connection requests.

Ipaddress:
The IP address, in dotted decimal form, on which the TCPIPSERVICE resource listens. For configurations with more than one IP stack, specify ANY to make the TCPIPSERVICE resource listen on all addresses.

SOcketclose:
Whether CICS waits before closing the socket after issuing a receive for incoming data on that socket. To ensure that the connection from the CICSRequest node always remains open, set SOcketclose to NO for IPIC connections.

SSl: Whether the CICS TCP/IP service is to use Secure Sockets Layer (SSL) protocol for encryption and authentication. Valid values are NO, YES, or CLIENTAUTH. Where:

- NO indicates that SSL is not to be used.
- YES indicates that the personal certificate of the CICS region must be trusted by WebSphere Message Broker.
- CLIENTAUTH indicates that the personal certificate of the CICS region must be trusted by WebSphere Message Broker, and the WebSphere Message Broker personal certificate must be trusted by CICS.

  The CICSRequest node does not support a separate truststore, so the keystore file must provide both personal and signer certificates. For more information, see "Securing the connection to CICS Transaction Server for z/OS by using SSL" on page 547.

4. Use the following command to install the TCPIPSERVICE definition:
   ```
   CEDA INS TCPIPSERVICE(service-name) GROUP(group-name)
   ```

Related concepts:

"CICS Transaction Server for z/OS overview" on page 2173
CICS Transaction Server for z/OS provides general-purpose transaction processing software for z/OS. CICS is a powerful application server that meets the transaction-processing needs of both large and small enterprises.

Related tasks:

"Changing connection information for the CICSRequest node" on page 738
You can create a configurable service that the CICSRequest node or message flow refers to at run time for connection information, instead of defining the connection properties on the node or the message flow. The advantage being that you can change the host name and performance values without needing to redeploy your message flow.

Related reference:

"CICSRequest node" on page 4321
Use the CICSRequest node to call CICS Transaction Server for z/OS programs over TCP/IP-based IP InterCommunications (IPIC) protocol.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

Securing the connection to CICS Transaction Server for z/OS by using SSL:

Configure the CICSRequest node to communicate with CICS Transaction Server for z/OS over the Secure Sockets Layer (SSL) protocol by updating a CICSConnection configurable service or the CICSRequest node to use SSL.

Before you begin

Before you start:

Ensure that you have completed the following tasks:

1. The CICSRequest node does not support a separate truststore, so the keystore file must provide both personal and signer certificates. If client-authentication (CLIENTAUTH) is enabled in the TCPIPSERVICE in CICS, the broker keystore file must also contain a personal certificate that is trusted by CICS. To set up a public key infrastructure (PKI) at broker or execution group level, follow the instructions in "Setting up a public key infrastructure" on page 504.
2. Create a message flow project and message set project, as described in "Creating a message flow project" on page 1425 and "Working with a message set project" on page 2838.
3. Define the COMMAREA data structure as a message set, as described in "Defining a CICS Transaction Server for z/OS data structure" on page 2193.
4. Configure IP InterCommunications (IPIC) protocol on CICS, as described in "Preparing the environment for the CICSRequest node" on page 736.

About this task

To configure the CICSRequest node to use SSL, complete the following steps:

Procedure

1. For client-authenticated (CLIENTAUTH) SSL connections, CICS expects the SSL client certificate to be mapped to a RACF user ID. Therefore the SSL client certificate must be mapped to a RACF user ID before attempting to establish the SSL connection to CICS. If the client certificate is not mapped to a RACF user ID, the broker might display a `ECI_ERR_NO_CICS` response. You can map a client certificate to a RACF user ID by using the RACF command **RACDCERT**, which stores the client certificate in the RACF database and associates a user ID with it, or by using RACF certificate name filtering. Client certificates can be mapped one-to-one with a user ID, or a mapping from one to the other can be provided to allow a many-to-one mapping. You can achieve this mapping by using one of the following methods:
   - **Associating a client certificate with a RACF user ID**

a. Copy the certificate that you want to process into an MVS sequential file. The file must have variable length, blocked records (RECFM=VB), and be accessible from TSO.

b. Run the **RACDCERT** command in TSO by using the following syntax:

```
RACDCERT ADD('datasetname') TRUST [ ID(userid) ]
```

Where:

– *datasetname* is the name of the data set containing the client certificate.

– *userid* is the user ID to be associated with the certificate. This parameter is optional. If omitted, the certificate is associated with the user issuing the **RACDCERT** command.

When you issue the **RACDCERT** command, RACF creates a profile in the DIGTCERT class. This profile associates the certificate with the user ID. You can then use the profile to translate a certificate to a user ID without giving a password. For full details of RACF commands, see z/OS Security Server RACF Command Language Reference.

- **RACF certificate name filtering**

  With certificate name filtering, client certificates are not stored in the RACF database. The association between one or more certificates and a RACF user ID is achieved by defining a filter rule that matches the distinguished name of the certificate owner or issuer (CA). A sample filter rule might look like the following example:

```
RACDCERT ID(DEPT3USR) MAP SDNFILTER
(OU=DEPT1.OU=DEPT2.O=IBM.L=LOC.SP=NY.C=US)
```

  This sample filter rule would associate user ID DEPT3USR with all certificates when the distinguished name of the certificate owner contains the organizational unit DEPT1 and DEPT2, the organization IBM, the locality LOC, the state/province NY, and the country US.

2. Turn on SSL support in the broker by setting the `cicsServer` property on the CICSConnection configurable service, as shown in the following example. This example changes the CICSRequest node that is configured to use the *myCICSConnection* configurable service for the CICS instance that is running at `mycicsregion.com` port 56789. After you run this command, the CICSRequest node connects to CICS over SSL.

```
mqsichangeproperties MB7BROKER -c CICSConnection -o myCICSConnection -n
cicsServer -v ssl://mycicsregion.com:56789
```

   Alternatively you can configure the `CICS server` property directly on the CICSRequest node.

What to do next

Next: When you have configured the broker or the CICSRequest node to use SSL, develop a message flow that contains a CICSRequest node by following the steps in "Developing a message flow with a CICSRequest node" on page 2199.

Related concepts:

"Security overview" on page 351
When you are designing a WebSphere Message Broker application it is important to consider the security measures that are needed to protect the information in the system.

"CICS Transaction Server for z/OS connectivity" on page 2174
Use the CICSRequest node to connect WebSphere Message Broker with CICS Transaction Server for z/OS applications.

"Using the WebSphere Message Broker Explorer to work with configurable services" on page 644
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view, add, modify and delete configurable services.

Related tasks:

"Implementing SSL authentication" on page 504
Use SSL authentication to enhance security in your broker environment.

"Changing connection information for the CICSRequest node" on page 738
You can create a configurable service that the CICSRequest node or message flow refers to at run time for connection information, instead of defining the connection properties on the node or the message flow. The advantage being that you can change the host name and performance values without needing to redeploy your message flow.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

Related reference:

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

"CICSRequest node" on page 4321
Use the CICSRequest node to call CICS Transaction Server for z/OS programs over TCP/IP-based IP InterCommunications (IPIC) protocol.

Developing a message flow with a CICSRequest node:

To connect to a CICS Transaction Server for z/OS application, create a message flow that contains a CICSRequest node.

Before you begin

Before you start:

Ensure that you have completed the following tasks:

1. Create a message flow project and message set project, as described in "Creating a message flow project" on page 1425 and "Working with a message set project" on page 2838.
2. Define the COMMAREA or channel data structure, as described in "Defining a CICS Transaction Server for z/OS data structure" on page 2193.
3. Optional: If you are using a channel data structure, create a message collection to represent the channel data structure, as described in "Creating a message collection by using ESQL" on page 2758.

4. Configure IP InterCommunications (IPIC) protocol on CICS, as described in "Preparing the environment for the CICSRequest node" on page 736.
5. Optional: Configure the CICSRequest node to communicate with CICS Transaction Server for z/OS over the Secure Sockets Layer (SSL) protocol, as described in "Securing the connection to CICS Transaction Server for z/OS by using SSL" on page 547.

About this task

Complete the following steps to develop a message flow with a CICSRequest node.

Procedure

1. Create a message flow.
2. Add a CICSRequest node to the message flow.
3. Configure the following properties on the CICSRequest node.
   - `CICS server`: The `CICS server` property can be defined either as a configurable service name, for example *myCICSConnection*, or as a URL.

     You can either connect to CICS by using the two-tier connection model; for example, by making a direct connection from WebSphere Message Broker to CICS, or by using the three-tier connection model; for example, by connecting to CICS through CICS Transaction Gateway for Multiplatforms. For more information about the two-tier and three-tier connection models, see "CICS Transaction Server for z/OS overview" on page 2173 for a high-level overview, or "CICS Transaction Server for z/OS two-tier connectivity" on page 2177 and "CICS Transaction Server for z/OS three-tier connectivity" on page 2181 for detailed conceptual information.

     Using the two-tier connection model:
     If you are making a direct two-tier connection from WebSphere Message Broker to CICS, you can define the `CICS server` property either as a configurable service name, for example *myCICSConnection*, or as a URL.

     For more information about defining this property as a configurable service, see "Changing connection information for the CICSRequest node" on page 738.

     To define a URL, specify the protocol and the CICS host name and port number. The format of this value is *protocol*`://`*hostname*`:`*port*. Where:

     - *protocol* can be `tcp` or `ssl`.
     - *hostname* is the TCP/IP address of the CICS host.
     - *port* is the port number of the TCPIPSERVICE listener in CICS that is listening for IPIC requests over TCP/IP or SSL.

     For example: `tcp://mycicsregion.com:12345` or `ssl://mycicsregion.com:56789`. You can obtain the *hostname* and *port* values from the IPIC TCPIPSERVICE definition in the target CICS region.

     Using the three-tier connection model:
     If you are making a three-tier connection to CICS through CICS Transaction Gateway for Multiplatforms, the `CICS server` CICSRequest node property must be defined as a configurable service name, for example *myCICSConnection*.

For more information about defining this property as a configurable service, see "Changing connection information for the CICSRequest node" on page 738.

To make a three-tier connection to CICS through CICS Transaction Gateway for Multiplatforms, you must configure the `cicsServer` and `gatewayURL` CICSConnection configurable service properties. For more information about configuring the `cicsServer` and `gatewayURL` CICSConnection configurable service properties to make a three-tier connection, see "Configurable services properties" on page 3766.

- `Program name`: Specify the name of the program that you want to run in the target CICS region.

  You can override this property in the local environment by specifying a value in the following location:

  `$LocalEnvironment/Destination/CICS/CICSProgramName`

- `Data structure`: Specify whether to use a COMMAREA or a channel data structure. The default for this property is `Commarea`. The decision depends on the targeted CICS program, for example; whether the target program is channel-based or not.

- `Commarea length`:

  This property is not configurable if a value of `Channel` is selected for the `Data structure` property.

  The `Commarea length` property is the size, in bytes, of the COMMAREA that is used by the CICS program. The byte size value is sent to CICS, and before the program is started, an area of memory is created to match that number. For example, if you send a `Commarea length` value of `100`, 100 bytes are allocated. The program accesses this area as the `DFHCOMMAREA`.

  Ensure that the `Commarea length` property value is large enough to contain the input request data, or the output response data, but that it does not exceed the maximum value of `32767` bytes. If the `Commarea length` value is not large enough to be used for the response data, or the request data, a memory leak occurs in CICS.

  The size of the COMMAREA cannot be changed by the CICS program.

  If the serialized request data is larger than the `Commarea length`, the data is truncated to the `Commarea length`.

  You can obtain the `Commarea length` value from the CICS administrator or developer.

  You can override this property in the local environment by specifying a value in the following location:

  `$LocalEnvironment/Destination/CICS/CICSCommareaLen`

4. Save the message flow.
5. Deploy the message flow.

What to do next

Next: When you have created and configured the message flow, build a message by following the steps in "Building a message for the CICSRequest node" on page 2202.

Related concepts:

"CICS Transaction Server for z/OS overview" on page 2173
CICS Transaction Server for z/OS provides general-purpose transaction processing software for z/OS. CICS is a powerful application server that meets the

transaction-processing needs of both large and small enterprises.

"CICS Transaction Server for z/OS connectivity" on page 2174
Use the CICSRequest node to connect WebSphere Message Broker with CICS Transaction Server for z/OS applications.

Related tasks:

"Connecting to a CICS Transaction Server for z/OS application" on page 2192
Connecting to a CICS Transaction Server for z/OS application involves creating a message flow, building a message, and processing the response from the CICSRequest node.

"Building a message for the CICSRequest node"
Create a message definition from a data structure and build a message by using another message flow node.

"Processing responses from a CICSRequest node" on page 2204
The CICSRequest node can return different response messages that indicate the success or failure of the request sent to CICS Transaction Server for z/OS.

"Changing connection information for the CICSRequest node" on page 738
You can create a configurable service that the CICSRequest node or message flow refers to at run time for connection information, instead of defining the connection properties on the node or the message flow. The advantage being that you can change the host name and performance values without needing to redeploy your message flow.

Related reference:

"CICSRequest node" on page 4321
Use the CICSRequest node to call CICS Transaction Server for z/OS programs over TCP/IP-based IP InterCommunications (IPIC) protocol.

Building a message for the CICSRequest node:

Create a message definition from a data structure and build a message by using another message flow node.

About this task

You can specify the location in the incoming message tree from which data is retrieved to form the request that is sent by the CICSRequest node. Specify this location by using the `Data location` Request property on the CICSRequest node.

Procedure

1. Find out what data needs to be in the body of the message to send to the CICSRequest node. The message body data must match the input structure that is required for your commarea, as defined in a message set definition based on the language structure. For example; COBOL or C copybook.
2. Ensure that the `Commarea length` Basic property value that is configured in the CICSRequest node is large enough to contain the input request data, or the output response data, but that it does not exceed the maximum value of `32767` bytes. If the `Commarea length` value is not large enough to be used for the response data, or the request data, a memory leak occurs in CICS. The size of the commarea cannot be changed by the CICS program. If the serialized request data is larger than the `Commarea length`, the data is truncated to the `Commarea length`. You can obtain the `Commarea length` value from the CICS administrator or developer.

Example

View the following sample to see how to use the CICSRequest node:

- CICS Transaction Server for z/OS Connectivity

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

The following example shows a message that has been modeled in the MRM domain, which can be received by a CICSRequest node and sent to CICS.

COBOL copybook

This example shows the structure of the data that CICS is expecting. The copybook describes the binary layout of the data that the CICS program expects to receive.

```
01 DFHAXCS-REQUEST.
   10 AXCS-COMMAND                  PIC S9(9) COMP.
   10 AXCS-FILE                    PIC X(8).
   10 AXCS-RIFLD                   PIC X(6) VALUE SPACES.
   10 AXCS-DATA.
      15 AXCS-STAT                 PIC X(1)  VALUE SPACES.
      15 AXCS-RECID                PIC X(6)  VALUE SPACES.
      15 AXCS-NAME                 PIC X(20) VALUE SPACES.
      15 AXCS-ADDRESS              PIC X(20) VALUE SPACES.
      15 AXCS-PHONE                PIC X(8)  VALUE SPACES.
      15 AXCS-DATE                 PIC X(8)  VALUE SPACES.
      15 AXCS-AMOUNT               PIC X(8)  VALUE SPACES.
      15 AXCS-COMMENT              PIC X(9)  VALUE SPACES.
```

The example copybook can be used to create a binary structure that requires 98 bytes of COMMAREA or memory space, as shown in the following example:

Table 22.

Name	**Type and Size**
AXCS-COMMAND	4 byte integer (fullword)
AXCS-FILE	8 byte character string
AXCS-RIFLD	6 byte character string
AXCS-STAT	1 byte character string
AXCS-RECID	6 byte character string
AXCS-NAME	20 byte character string
AXCS-ADDRESS	20 byte character string
AXCS-PHONE	8 byte character string
AXCS-DATE	8 byte character string
AXCS-AMOUNT	8 byte character string
AXCS-COMMENT	9 byte character string
Total	**98 bytes**

The COBOL copybook structure must be imported as a message definition, see “Importing from COBOL copybooks” on page 2937 for further information, and a message containing such a structure must be passed to the CICSRequest node. A second copybook might be required to map the returning COMMAREA.

What to do next

Next: Process the responses from the CICSRequest, as described in "Processing responses from a CICSRequest node."

Related concepts:

"CICS Transaction Server for z/OS overview" on page 2173
CICS Transaction Server for z/OS provides general-purpose transaction processing software for z/OS. CICS is a powerful application server that meets the transaction-processing needs of both large and small enterprises.

"CICS Transaction Server for z/OS connectivity" on page 2174
Use the CICSRequest node to connect WebSphere Message Broker with CICS Transaction Server for z/OS applications.

"Samples" on page 98
The WebSphere Message Broker Toolkit provides samples that show the features that are available in WebSphere Message Broker, and how to use them. This topic provides links to the information about the individual samples.

Related tasks:

"Connecting to a CICS Transaction Server for z/OS application" on page 2192
Connecting to a CICS Transaction Server for z/OS application involves creating a message flow, building a message, and processing the response from the CICSRequest node.

"Working with data structures" on page 2930
You can create a message definition file in a message set by importing from XML Schema, XML DTD, SCA import or export components, IBM supplied messages, WSDL definitions, IDL files, C header files, and COBOL copybooks. This topic area describes how to import from these data structures using the command line or the WebSphere Message Broker Toolkit.

"Processing responses from a CICSRequest node"
The CICSRequest node can return different response messages that indicate the success or failure of the request sent to CICS Transaction Server for z/OS.

"Changing connection information for the CICSRequest node" on page 738
You can create a configurable service that the CICSRequest node or message flow refers to at run time for connection information, instead of defining the connection properties on the node or the message flow. The advantage being that you can change the host name and performance values without needing to redeploy your message flow.

Related reference:

"CICSRequest node" on page 4321
Use the CICSRequest node to call CICS Transaction Server for z/OS programs over TCP/IP-based IP InterCommunications (IPIC) protocol.

Processing responses from a CICSRequest node:

The CICSRequest node can return different response messages that indicate the success or failure of the request sent to CICS Transaction Server for z/OS.

Before you begin

Before you start:

Ensure that you have built a message for the CICSRequest node, as described in "Building a message for the CICSRequest node" on page 2202.

About this task

The CICSRequest node has four output terminals:

- Out: The output terminal from which the message tree is propagated, including the data returned from CICS.
- Failure: The output terminal to which a message is routed if a CICSRequest node exception is detected, or a CICSRequest node to CICS connection failure occurs.
- Error: The output terminal to which a message is propagated if a CICS error (abend) occurs.
- Timeout: The output terminal to which the message is propagated if a per-request timeout occurs when an individual request is sent to CICS, but the request takes too long.

You can select the location to which to send the response by configuring the `Output data location` Result property on the CICSRequest node. This property specifies the location in the message tree to which the CICSRequest node places the output.

Procedure

- **Processing successful calls**

  When a CICSRequest node successfully calls a CICS application, the resulting message is propagated to the Out terminal.
- **Processing CICS abends**

  An abend in CICS causes a message to be propagated from the Error terminal. The input message is propagated with a CICS\AbendCode field in the LocalEnvironment. If the Error terminal is not connected, the abend is lost.
- **Processing request timeouts**

  If a CICSRequest node is configured with a `Request timeout` Basic property, and a particular message takes longer than the specified time in seconds to be processed, the request fails and the message is propagated from the Timeout terminal. The output message contains the input message body and a timeout exception in the ExceptionList. If the Timeout terminal is not connected and a timeout occurs, the request timeout exception is routed to the Failure terminal. If the Failure terminal is not connected, the broker throws an exception and returns control to the closest upstream node that can process it. The default behavior is that the message is returned to the input node.
- **Handling failures in the node**

  Any other failures are propagated to the Failure terminal. Possible failures include:

  – An inability to communicate with the target CICS region.
  – An internal CICSRequest node exception is detected.
  – An invalid message body. For example, a parsing error or input message is creating a structure that is larger than the `Commarea length` Basic property that is configured in the CICSRequest node. A similar situation for the returning COMMAREA cannot be detected. For example, if the returning data is larger than the value that is defined in the `Commarea length` property, a memory leak occurs in CICS. Therefore the `Commarea length` Basic property must be correctly configured.

Results

You can use the CICSConnection configurable service to change connection details for the CICSRequest node. See "Changing connection information for the CICSRequest node" on page 738 for details about creating, changing, reporting, and deleting the CICSConnection configurable service.

Related concepts:

"CICS Transaction Server for z/OS overview" on page 2173
CICS Transaction Server for z/OS provides general-purpose transaction processing software for z/OS. CICS is a powerful application server that meets the transaction-processing needs of both large and small enterprises.

"CICS Transaction Server for z/OS connectivity" on page 2174
Use the CICSRequest node to connect WebSphere Message Broker with CICS Transaction Server for z/OS applications.

Related tasks:

"Connecting to a CICS Transaction Server for z/OS application" on page 2192
Connecting to a CICS Transaction Server for z/OS application involves creating a message flow, building a message, and processing the response from the CICSRequest node.

"Changing connection information for the CICSRequest node" on page 738
You can create a configurable service that the CICSRequest node or message flow refers to at run time for connection information, instead of defining the connection properties on the node or the message flow. The advantage being that you can change the host name and performance values without needing to redeploy your message flow.

Related reference:

"CICSRequest node" on page 4321
Use the CICSRequest node to call CICS Transaction Server for z/OS programs over TCP/IP-based IP InterCommunications (IPIC) protocol.

Changing connection information for the CICSRequest node:

You can create a configurable service that the CICSRequest node or message flow refers to at run time for connection information, instead of defining the connection properties on the node or the message flow. The advantage being that you can change the host name and performance values without needing to redeploy your message flow.

Before you begin

Before you start:

- Read "Configurable services" on page 1296 to find out more about configurable services.
- Read "CICS Transaction Server for z/OS overview" on page 2173 for background information.

About this task

Use the CICSConnection configurable service to change the CICS Transaction Server for z/OS connection information for the CICSRequest node. The advantage being that you can change the host name, performance, and security identity values without needing to redeploy your message flow. The properties of the CICSConnection configurable service are described in "Configurable services properties" on page 3766.

You can use the CICSConnection configurable service to configure the CICSRequest node to use Secure Sockets Layer (SSL) protocol. For more information, see "Securing the connection to CICS Transaction Server for z/OS by using SSL" on page 547.

Creating, changing, reporting, and deleting configurable services

Procedure

- To create a configurable service, use the WebSphere Message Broker Explorer; see "Using the WebSphere Message Broker Explorer to work with configurable services" on page 644 for more information, or the **mqsicreateconfigurableservice** command, as shown in the following example. This example creates a CICSConnection configurable service for the CICS instance that is running at `test.cics.ibm.com` port 12345. The broker is identified by APPLID *BRKApp* and qualifier *BRKQual*. The connection timeout is 10 seconds, the request timeout is 5 seconds, and the security identity is identified by *mySecurityIdentity* in this example:

  ```
  mqsicreateconfigurableservice MB7BROKER -c CICSConnection -o myCICSConnectionService
  -n cicsServer,clientApplid,clientQualifier,connectionTimeoutSecs,requestTimeoutSecs,
  securityIdentity
  -v tcp://test.cics.ibm.com:12345,BRKApp,BRKQual,10,5,mySecurityIdentity
  ```

- To change a configurable service, use the WebSphere Message Broker Explorer, or the **mqsichangeproperties** command, as shown in the following example. You must stop and start the execution group for the change of property value to take effect. This example changes the CICSRequest node that is configured to use the *myCICSConnectionService* configurable service. After you run this command, the CICSRequest node connects to the production system (`tcp://production.cics.ibm.com:12345`) instead of the test system (`tcp://test.cics.ibm.com:12345`).

  ```
  mqsichangeproperties MB7BROKER -c CICSConnection -o myCICSConnectionService
    -n cicsServer -v tcp://production.cics.ibm.com:12345
  ```

  See "Securing the connection to CICS Transaction Server for z/OS by using SSL" on page 547 for information about how to turn on SSL support in the broker by setting the `cicsServer` CICSConnection configurable service property.

- To display all CICSConnection configurable services, use the WebSphere Message Broker Explorer, or the **mqsireportproperties** command, as shown in the following example:

  ```
  mqsireportproperties MB7BROKER -c CICSConnection -o AllReportableEntityNames -r
  ```

- You can delete a configurable service that you have created by using the WebSphere Message Broker Explorer, or the **mqsideleteconfigurableservice** command, as shown in the following example:

  ```
  mqsideleteconfigurableservice MB7BROKER -c CICSConnection -o myCICSConnectionService
  ```

  You can delete only custom-named configurable services; you cannot delete IBM-defined configurable services.

Related concepts:

"CICS Transaction Server for z/OS connectivity" on page 2174
Use the CICSRequest node to connect WebSphere Message Broker with CICS Transaction Server for z/OS applications.

"Configurable services" on page 1296
Configurable services are typically runtime properties. You can use them to define properties that are related to external services on which the broker relies; for example, an SMTP server or a JMS provider.

"Using the WebSphere Message Broker Explorer to work with configurable services" on page 644
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view, add, modify and delete configurable services.

Related reference:

"CICSRequest node" on page 4321
Use the CICSRequest node to call CICS Transaction Server for z/OS programs over TCP/IP-based IP InterCommunications (IPIC) protocol.

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

Propagating security credentials to CICS Transaction Server for z/OS:

The CICSRequest node can use an identity that is present in the Properties folder of the message tree structure for the security credentials in a request, by using the `Propagate` property on the security profile that is defined on the node.

About this task

If a CICSRequest node is configured with a security profile, it extracts security tokens from the input message at run time, and propagates an identity to CICS.

Procedure

To propagate an identity to be used for the CICS request security credentials, complete the following steps.

1. Ensure that an appropriate security profile exists for the CICSRequest node, or create a security profile, by following the instructions in "Creating a security profile" on page 433.
2. Use the Broker Archive editor to select a security profile for the CICSRequest node that has identity propagation enabled. For detailed instructions, see "Configuring for identity propagation" on page 492.

Related concepts:

"Identity and security token propagation" on page 426
Identity and security token propagation enables the identity and security tokens (associated with each message) to be propagated throughout a message flow, and on to target applications through output or request nodes.

"CICS Transaction Server for z/OS connectivity" on page 2174
Use the CICSRequest node to connect WebSphere Message Broker with CICS Transaction Server for z/OS applications.

Related tasks:

"Configuring for identity propagation" on page 492
To enable a message flow to perform identity propagation, the input nodes must extract the identity and the output node must propagate it.

"Connecting to a CICS Transaction Server for z/OS application" on page 2192
Connecting to a CICS Transaction Server for z/OS application involves creating a message flow, building a message, and processing the response from the CICSRequest node.

Related reference:

"CICSRequest node" on page 4321
Use the CICSRequest node to call CICS Transaction Server for z/OS programs over TCP/IP-based IP InterCommunications (IPIC) protocol.

Routing messages

Route messages through your message flow or the broker by using one or more of the techniques described in this section.

About this task

- "Using nodes for decision making"
- "Routing using publish/subscribe applications" on page 2215

Some nodes use fields in your messages to make routing decisions. You might need to define a model of your messages to enable access to the fields in the message for these nodes. For information about how to develop a message model, and why you might want to develop a message model, see "Constructing message models" on page 2838.

Related tasks:

"Connecting client applications" on page 1537
Connect your client applications to the broker by using one or more of the supported protocols from other resources and software servers in your network.

"Processing events" on page 2717
Using event driven message processing you can control the flow of messages through your message flows.

"Transforming and enriching messages" on page 2227
Transform and enrich messages by using one or more of the techniques described in this section.

Using nodes for decision making

You can use several built-in nodes in different ways to control the path that a message takes through the message flow.

Before you begin

Before you start:

Read the concept topic about message flow nodes.

About this task

These nodes let you decide how messages are processed by specifying the route that each message takes through the message flow based on dynamic values such as message structure and content.

For more information, see the following topics:
- "Testing the message structure (Validate node)" on page 2211
- "Controlling the order of processing in a message flow" on page 2212
- "Using the destination list to route messages (RouteToLabel and Label nodes)" on page 2214

You can use one of the following nodes to determine the path taken by a message through the message flow based on its content:
- "Route node" on page 4669
- Filter node (ESQL)
- JavaCompute node
- PHPCompute node

Use the DatabaseRoute node to route messages using information from a database. For more information, see "DatabaseRoute node" on page 4373.

Related concepts:

"Local environment tree structure" on page 1056
The local environment tree is a part of the logical message tree in which you can store information while the message flow processes the message.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"DatabaseRoute node" on page 4373
Use the DatabaseRoute node to route messages using information from a database in conjunction with XPath expressions.

"Filter node" on page 4452
Use the Filter node to route a message according to message content.

"FlowOrder node" on page 4458
Use the FlowOrder node to control the order in which a message is processed by a message flow.

"JavaCompute node" on page 4514
Use the JavaCompute node to work with messages by using the Java language.

"Label node" on page 4569
Use the Label node to process a message that is propagated by a RouteToLabel node to dynamically determine the route that the message takes through the

message flow.

"PHPCompute node" on page 4639
Use the PHPCompute node to route and transform an incoming message, using the PHP scripting language.

"ResetContentDescriptor node" on page 4663
Use the ResetContentDescriptor node to request that the message is reparsed by a different parser.

"Route node" on page 4669
Use the Route node to direct messages that meet certain criteria down different paths of a message flow.

"RouteToLabel node" on page 4673
Use the RouteToLabel node in combination with one or more Label nodes to dynamically determine the route that a message takes through the message flow, based on its content.

"Validate node" on page 4959
Use the Validate node to check that the message that arrives on its input terminal is as expected. You can use this node to check that the message has the expected message template properties, and to check that the content of the message is correct by selecting message validation.

Testing the message structure (Validate node)

Use the Validate node to test the characteristics of the message structure.

About this task

If you set the Validate node properties appropriately, you can request that one or all of the message domain, message set, and message type are compared to a specific value. If the message matches those values for which you have requested the check, it is routed through the match terminal and is processed by the sequence of nodes that you have connected to that terminal.

If the message does not match any one of those values for which you have requested the check, it is routed through the failure terminal and is processed by the sequence of nodes that you have connected to that terminal.

For example, you might design a message flow that provides additional processing for all messages that are in the MRM domain. You can include a Validate node that tests just that characteristic of the message, and passes it to a sequence of nodes that provide the specialized processing. If the message is not in the MRM domain, the extra nodes are bypassed, and the failure terminal is wired up directly to the node that follows the sequence required for MRM messages only.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Using nodes for decision making" on page 2209
You can use several built-in nodes in different ways to control the path that a message takes through the message flow.

"Controlling the order of processing in a message flow" on page 2212
Use the FlowOrder node to control the order of processing in a message flow.

"Testing the message content (Filter node)" on page 2213
You can use the Filter node to determine the path taken by a message through the

message flow based on its content.

"Using the destination list to route messages (RouteToLabel and Label nodes)" on page 2214
You can determine the path that a message takes through the message flow by using the RouteToLabel and Label nodes.

Related reference:

"Validate node" on page 4959
Use the Validate node to check that the message that arrives on its input terminal is as expected. You can use this node to check that the message has the expected message template properties, and to check that the content of the message is correct by selecting message validation.

Controlling the order of processing in a message flow

Use the FlowOrder node to control the order of processing in a message flow.

About this task

When you connect message flow nodes together, the broker determines the way in which the different connections are processed. This includes the order in which they are processed. If you have connected more than one node or sequence of nodes to a single output terminal, you cannot predict whether one sequence is processed before another for a message.

If the order of processing is important in your message flow, use the FlowOrder node to force a prescribed order of processing of the messages that are propagated by this node.

The FlowOrder node has two output terminals that you can connect to control the order in which subsequent nodes process the message. The output terminals, named First and Second, are always processed in that order.

When you connect a node or sequence of nodes to the First terminal, the input message is passed to the next node, and all processing defined by all subsequent nodes in this sequence is completed before control returns to the FlowOrder node.

The input message is then propagated to the next node in the sequence of nodes connected to the Second terminal.

The message passed to both sequences of nodes, from the First terminal and the Second terminal, is identical. It is always the message that the FlowOrder node receives as input. The message that the FlowOrder node propagates to the Second terminal is in no way affected by the processing of the message that has been performed by the sequence of nodes connected to the First terminal.

The FlowOrder node provides no other processing on the input message; it is used only for imposing order on subsequent processing.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Using nodes for decision making" on page 2209
You can use several built-in nodes in different ways to control the path that a message takes through the message flow.

"Testing the message structure (Validate node)" on page 2211
Use the Validate node to test the characteristics of the message structure.

"Testing the message content (Filter node)"
You can use the Filter node to determine the path taken by a message through the message flow based on its content.

"Using the destination list to route messages (RouteToLabel and Label nodes)" on page 2214
You can determine the path that a message takes through the message flow by using the RouteToLabel and Label nodes.

Related reference:

"FlowOrder node" on page 4458
Use the FlowOrder node to control the order in which a message is processed by a message flow.

Testing the message content (Filter node)

You can use the Filter node to determine the path taken by a message through the message flow based on its content.

About this task

You can customize the Filter node by using ESQL statements to determine if the message content meets some condition. The condition tested must yield a Boolean result, that is it must be true or false (or unknown). You can create the test to reference information from a database, if applicable.

You can connect nodes following the Filter node to the corresponding terminals of the Filter node, and process the message according to its content.

Look at the following samples to see how to use the Filter node:
- Airline Reservations
- Error Handler

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Using nodes for decision making" on page 2209
You can use several built-in nodes in different ways to control the path that a message takes through the message flow.

"Testing the message structure (Validate node)" on page 2211
Use the Validate node to test the characteristics of the message structure.

"Controlling the order of processing in a message flow" on page 2212
Use the FlowOrder node to control the order of processing in a message flow.

"Using the destination list to route messages (RouteToLabel and Label nodes)" on page 2214
You can determine the path that a message takes through the message flow by using the RouteToLabel and Label nodes.

Related reference:

"Filter node" on page 4452
Use the Filter node to route a message according to message content.

Using the destination list to route messages (RouteToLabel and Label nodes)

You can determine the path that a message takes through the message flow by using the RouteToLabel and Label nodes.

About this task

These nodes provide a more flexible way to process messages than the Filter node, which depends on the Boolean result of an ESQL expression for its logic.

When you use RouteToLabel and Label nodes, you must include a Compute node that determines, by using some combination of message content, database content, and ESQL logic, how messages are to be processed next. Configure the Compute node to create a destination list (in the DestinationList folder in the local environment subtree) that contains the destination for each message, specified as the LabelName of a Label node. The Compute node passes the message to the RouteToLabel node, which reads the destination list and propagates the message to either the first or last item on the destination list, according to the value that is specified for the RouteToLabel node's `Mode` property. Although there is no limit to the number of destinations that the Compute node writes in the destination list, the RouteToLabel node propagates the message only to a single label node. This use of the destination list is in contrast to its use to define the final recipients of the output messages. For more information about the procedure for creating a destination list, see "Creating destination lists" on page 1477.

If you intend to derive destination values from the message itself, or from a database, you might also need to cast values from one type to another. For more information about the local environment, see "Local environment tree structure" on page 1056. For more information about casting, see "Supported casts" on page 5273.

Look at the following sample to see how to use these nodes:
- Airline Reservations

The XML_PassengerQuery message flow in the previous sample demonstrates how you can use the destination list in the local environment to route messages based on the information in the message itself.

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Local environment tree structure" on page 1056
The local environment tree is a part of the logical message tree in which you can store information while the message flow processes the message.

Related tasks:

"Using nodes for decision making" on page 2209
You can use several built-in nodes in different ways to control the path that a

message takes through the message flow.

"Testing the message structure (Validate node)" on page 2211
Use the Validate node to test the characteristics of the message structure.

"Controlling the order of processing in a message flow" on page 2212
Use the FlowOrder node to control the order of processing in a message flow.

"Testing the message content (Filter node)" on page 2213
You can use the Filter node to determine the path taken by a message through the message flow based on its content.

Related reference:

"Label node" on page 4569
Use the Label node to process a message that is propagated by a RouteToLabel node to dynamically determine the route that the message takes through the message flow.

"RouteToLabel node" on page 4673
Use the RouteToLabel node in combination with one or more Label nodes to dynamically determine the route that a message takes through the message flow, based on its content.

"ResetContentDescriptor node" on page 4663
Use the ResetContentDescriptor node to request that the message is reparsed by a different parser.

Routing using publish/subscribe applications

You can route your messages to applications using the publish/subscribe method of messaging.

There are two situations where you can use publish/subscribe in WebSphere Message Broker. These situations are where you want to:

- Provide additional transformation or routing function, or both, at publication time.
- Filter messages based on the content of the body of the message.

Developing publish/subscribe applications

For background information about the publish/subscribe method of messaging, see the following topics:

- "Publish/Subscribe"
- "Publishers" on page 2218
- "Publications" on page 2219
- "Subscribers" on page 2220
- "Filters" on page 2221
- "Subscription points" on page 2222

Publish/Subscribe in WebSphere Message Broker Version 7.0

In WebSphere Message Broker Version 7.0, publish/subscribe uses the function available in WebSphere MQ Version 7.0.

Publish/Subscribe

Publish/subscribe is a style of messaging application in which the providers of information (publishers) are decoupled from the consumers of that information (subscribers).

In a publish/subscribe system, a publisher does not need to know who uses the information (publication) that it provides, and a subscriber does not need to know who provides the information that it receives as the result of a subscription.

Compare this with a point-to-point style of messaging application, in which the application that sends messages needs to know the destinations of the messages that it sends.

Queue managers make sure that messages arrive at the correct destinations, and are transformed to the format required at each destination.

A typical publish/subscribe system has more than one publisher and more than one subscriber, and often more than one queue manager. An application can be both a publisher and a subscriber.

The publisher generates a message that it wants to publish and defines the topic of the message.

A subscriber registers a request for a publication by specifying one of the following items:

- The topic, or topics, of the published messages that it is interested in.
- The subscription point from which it wants to receive publications.
- The content filter that should be applied to the published message.
- The name of the queue (known as the subscriber queue) on which publications that match the criteria selected are placed. This queue can be a cluster queue, so that publications can be distributed to clustered subscribers.

Related concepts:

"Publishers" on page 2218
A *publisher* is an application that makes information about a specified topic available to a queue manager in a publish/subscribe system.

"Publications" on page 2219
A *publication* is a piece of information about a specified topic that is available to a queue manager in a publish/subscribe system.

"Subscription points" on page 2222
A *subscription point* is the name that a subscriber uses to request publications from a particular set of Publication nodes. It is the property of a Publication node that differentiates that Publication node from other Publication nodes in the same message flow.

"Subscribers" on page 2220
A *subscriber* is an application that requests information about a specified topic from a publish/subscribe queue manager.

"Filters" on page 2221
A *filter* is an expression, which might include wildcard characters, that is applied to the content of a publication message to determine whether it matches a subscription.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Routing using publish/subscribe applications" on page 2215
You can route your messages to applications using the publish/subscribe method of messaging.

Related reference:

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

"Publication node" on page 4643
Use the Publication node to filter output messages from a message flow and transmit them to subscribers who have registered an interest in a particular set of topics.

"Publish/subscribe" on page 6395
Use the reference information in this section to help you develop publish/subscribe applications.

Changes to nodes in WebSphere Message Broker Version 7.0:

There are changes to the nodes provided in this version of the product. These changes include removal of the SCADA and RealTime nodes.

The Publication node has two additional terminals, Out and No Match; see "Publication node" on page 4643 for further information.

WebSphere MQ Telemetry Transport is supported from WebSphere MQ; see the WebSphere MQ Telemetry Transport documentation for further information.

WebSphere MQ Real-time Transport and WebSphere MQ Telemetry Transport are no longer supported. As a result of this change, the following nodes have been removed:

Real-timeInput node
Flows containing Real-timeInput nodes that have been migrated to WebSphere Message Broker Version 7.0 will not start until these nodes have been removed from the flow and the flow has been redeployed.

Real-timeOptimizedFlow node
Flows containing Real-timeOptimizedFlow nodes that have been migrated to WebSphere Message Broker Version 7.0 will not start until these nodes have been removed from the flow and the flow has been redeployed.

If your message flows contain only a Real-timeOptimizedFlow node, no WebSphere Message Broker flow is required after you have migrated to WebSphere Message Broker Version 7.0.

SCADAInput node
Flows containing SCADAInput nodes that have been migrated to WebSphere Message Broker Version 7.0 will not start until these nodes have been removed from the flow and the flow has been redeployed.

SCADAOutput node
Flows containing SCADAOutput nodes that have been migrated to WebSphere Message Broker Version 7.0 will fail if processing reaches the node and you receive a `BIP` warning message.

MQeInput
The MQeInput node has been removed. This node was deprecated in an earlier version of the product.

MQeOutput
The MQeOutput node has been removed. This node was deprecated in an earlier version of the product.

Contact your account representative for more information about support for WebSphere MQ Real-time Transport and WebSphere MQ Telemetry Transport.

Additionally, although the MQOptimizedFlow node still works, this node is redundant.

Related reference:

"Publication node" on page 4643
Use the Publication node to filter output messages from a message flow and transmit them to subscribers who have registered an interest in a particular set of topics.

"Real-timeInput node" on page 4646
The Real-timeInput node, available in earlier versions of WebSphere Message Broker, is not supported in WebSphere Message Broker Version 7.0. See for information about migrating your message flows from WebSphere Message Broker to WebSphere Message Broker Version 7.0.

"Real-timeOptimizedFlow node" on page 4646
The Real-timeOptimizedFlow node, available in earlier versions of WebSphere Message Broker, is not supported in WebSphere Message Broker Version 7.0. See for information about migrating your message flows from WebSphere Message Broker to WebSphere Message Broker Version 7.0.

"SCADAInput node" on page 4706
The SCADAInput node, available in earlier versions of WebSphere Message Broker, is not supported in WebSphere Message Broker Version 7.0. See for information about migrating your message flows from WebSphere Message Broker to WebSphere Message Broker Version 7.0.

"SCADAOutput node" on page 4706
The SCADAOutput node, available in earlier versions of WebSphere Message Broker, is not supported in WebSphere Message Broker Version 7.0. See for information about migrating your message flows from WebSphere Message Broker to WebSphere Message Broker Version 7.0.

Publishers:

A *publisher* is an application that makes information about a specified topic available to a queue manager in a publish/subscribe system.

In a publish/subscribe system, an application, known as the publisher, can send a message to a message queue or port that is associated with an input node in a message flow that contains a Publication node.

Another application, known as the subscriber, can send a subscription request to the queue manager, which then sends relevant publication messages to the message queue or port of the subscriber.

A published message can be requested by more than one subscriber, and a subscriber can request messages, on the same or different topics, from more than one publisher.

Related concepts:

"Publications" on page 2219
A *publication* is a piece of information about a specified topic that is available to a queue manager in a publish/subscribe system.

"Subscribers" on page 2220
A *subscriber* is an application that requests information about a specified topic from a publish/subscribe queue manager.

Related reference:

"Publish message" on page 6405

"Real-timeInput node" on page 4646
The Real-timeInput node, available in earlier versions of WebSphere Message Broker, is not supported in WebSphere Message Broker Version 7.0. See for information about migrating your message flows from WebSphere Message Broker to WebSphere Message Broker Version 7.0.

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

"Publication node" on page 4643
Use the Publication node to filter output messages from a message flow and transmit them to subscribers who have registered an interest in a particular set of topics.

"SCADAInput node" on page 4706
The SCADAInput node, available in earlier versions of WebSphere Message Broker, is not supported in WebSphere Message Broker Version 7.0. See for information about migrating your message flows from WebSphere Message Broker to WebSphere Message Broker Version 7.0.

Publications:

A *publication* is a piece of information about a specified topic that is available to a queue manager in a publish/subscribe system.

Typically, a queue manager distributes a publication that it receives to all applications that are connected to it and that have registered a subscription for the publication. The queue manager also distributes the publication to all other queue managers connected to it, either directly or through a network of queue managers that have subscribers for the publication.

Local publications: Publishers can restrict access to their publications to only those subscribers that are registered to the same queue manager as the publisher. This publication is known as a *local publication*. Local publications are not forwarded to other queue managers.

Global publications: A publication whose distribution is not restricted to only those subscribers that are registered to the same queue manager as the publisher is known as *global publication*. A global publication is forwarded to all queue managers, connected either directly or through a network of queue managers, that have one or more subscribers for the publication.

Retained publications: Typically, a queue manager discards a publication after it has been sent. However, a publisher can specify (in the case of the *Publish* message, by specifying the *RetainPub* option) that it wants the queue manager to keep a copy of the publication, which is then called a *retained publication*.

If a retained publication has been published, new subscribers to that publication receive the publication without having to wait for it to be published again.

For example, a subscriber that registers a subscription for a stock price receives the latest published stock price immediately, and does not have to wait for the stock price to be republished.

A queue manager retains only one publication for each combination of topic and subscription point.

State and event information: Information being published can be categorized either as *state* information or as *event* information.

State information is information about the current state of something. The current price of stock or the current score in a soccer match are both examples of state information.

Event information is information about an individual event that occurs. A change in the price of stock or the scoring of a particular goal in a soccer match are both examples of event information.

When an event occurs, the current state information is no longer required and is superseded by new state information.

If a publication contains state information, it is often published as a retained publication. A new subscriber typically wants the current information immediately; the subscriber does not want to wait for an event that causes the information to be republished.

Related concepts:

"Subscription points" on page 2222
A *subscription point* is the name that a subscriber uses to request publications from a particular set of Publication nodes. It is the property of a Publication node that differentiates that Publication node from other Publication nodes in the same message flow.

Related reference:

"Publication node" on page 4643
Use the Publication node to filter output messages from a message flow and transmit them to subscribers who have registered an interest in a particular set of topics.

"Publish message" on page 6405

Subscribers:

A *subscriber* is an application that requests information about a specified topic from a publish/subscribe queue manager.

The subscribing application might be a WebSphere MQ or WebSphere Message Broker application.

The subscriber sends a subscription request to a queue manager, specifying which publications it wants to receive. The request defines the topic, the filter, and the subscription point of each publication.

Messages that are published by a publisher can be received by more than one subscriber, and a subscriber can receive messages, on the same or different topics, from more than one publisher.

Related concepts:

"Publish/Subscribe" on page 2215
Publish/subscribe is a style of messaging application in which the providers of information (publishers) are decoupled from the consumers of that information (subscribers).

"Publishers" on page 2218
A *publisher* is an application that makes information about a specified topic available to a queue manager in a publish/subscribe system.

"Publications" on page 2219
A *publication* is a piece of information about a specified topic that is available to a queue manager in a publish/subscribe system.

"Filters"
A *filter* is an expression, which might include wildcard characters, that is applied to the content of a publication message to determine whether it matches a subscription.

"Subscription points" on page 2222
A *subscription point* is the name that a subscriber uses to request publications from a particular set of Publication nodes. It is the property of a Publication node that differentiates that Publication node from other Publication nodes in the same message flow.

Filters:

A *filter* is an expression, which might include wildcard characters, that is applied to the content of a publication message to determine whether it matches a subscription.

When you register a subscription, in addition to specifying a topic and subscription point, you can specify a filter to select publications according to their contents. WebSphere Message Broker must know how to parse the contents of the message correctly, which can be achieved in a number of ways:

- The message is a self-defining XML message.
- The message template is defined in the MQRFH2 header.

  If the message has an MQRFH header, the message set and type are taken from that header. Otherwise, the message is assumed to be as defined in the properties (domain, set, type, and format) of the input node.

The filter itself is entered as an ESQL expression; for example:

```
Body.Name LIKE 'Smit%'
```

This example means that the contents of a field called Name in the body of a publication message are extracted and compared with the string given in the expression. If the string in the message starts with the characters "Smit", the expression evaluates to TRUE and the publication is sent to the subscriber.

If you want to select publications using filters only, without specifying a topic, you can register a subscription with the required filter and a topic of "#" (all topics). You then receive publications on only those topics for which you have access authority.

This subscription results in all publications from all connected brokers being sent to the broker that is local to the subscriber. Therefore, for performance reasons, if you have set up a network of brokers, you are advised to not use this technique.

Related concepts:

"Content based filtering using ESQL" on page 2223
WebSphere MQ supports message selection based on `Root.MQMD` and message properties; WebSphere Message Broker Version 7.0 provides extra options.

Related reference:

“ESQL reference” on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

Subscription points:

A *subscription point* is the name that a subscriber uses to request publications from a particular set of Publication nodes. It is the property of a Publication node that differentiates that Publication node from other Publication nodes in the same message flow.

A subscriber that registers a subscription without specifying a subscription point receives publications from any unnamed Publication node in the message flow, if there is a match with the topic and filter specified by the subscriber.

This behavior applies to all message flows running in all brokers connected in the same network, unless the local has been specified when registering the subscription.

Using subscription points: If you have more than one Publication node in a message flow, you can differentiate between them by specifying subscription points. Choose values that indicate the type of message that is routed to each Publication node.

Example: Consider an application that publishes stock prices. The prices that are available from the first Publication node in the message flow are in dollars. This Publication node uses the default subscription point.

You can define a second path through the message flow that takes the price in dollars, and converts this using a defined conversion value, to produce the same message but with the stock price in pounds. These messages are published at a second Publication node that has its subscription point property set to "Pounds".

You might have another message flow, in the same broker or a connected broker, that publishes stock prices in pounds on the same topic. Make sure that it uses the "Pounds" subscription point, and that all other message flows that publish their stock prices in dollars use the default subscription point.

Subscribers specifying the relevant topic (for example, "stock") can then choose whether to receive the information in dollars or in pounds, by using the default subscription point or the "Pounds" subscription point when they subscribe.

Related concepts:

“Message flows overview” on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

“Publishers” on page 2218
A *publisher* is an application that makes information about a specified topic available to a queue manager in a publish/subscribe system.

“Publications” on page 2219
A *publication* is a piece of information about a specified topic that is available to a queue manager in a publish/subscribe system.

“Subscribers” on page 2220
A *subscriber* is an application that requests information about a specified topic from a publish/subscribe queue manager.

"Filters" on page 2221
A *filter* is an expression, which might include wildcard characters, that is applied to the content of a publication message to determine whether it matches a subscription.

Related reference:

"Publication node" on page 4643
Use the Publication node to filter output messages from a message flow and transmit them to subscribers who have registered an interest in a particular set of topics.

"MQRFH2 structure" on page 6397

"Publish message" on page 6405

Content based filtering using ESQL

WebSphere MQ supports message selection based on `Root.MQMD` and message properties; WebSphere Message Broker Version 7.0 provides extra options.

Content based filters in WebSphere Message Broker Version 7.0 extend WebSphere MQ message selectors to allow the use of ESQL expressions to filter on the entire message.

You can use the following:

Correlation names
: In content based filters, the message can be accessed using the `Root`, `Body`, and `Properties` correlation names.

  Any other correlation name is interpreted as a WebSphere MQ message property, apart from the following list of excluded reserved names:

  - `Database`
  - `DestinationList`
  - `Environment`
  - `LocalEnvironment`
  - `ExceptionList`
  - `LocalEnvironment`

  Any other top level identifier is assumed to be a WebSphere MQ message property, and is accepted.

Trees
: The content based filtering engine does not update the message tree. It accesses the message tree as read-only. Any statements that attempt to modify the tree are rejected.

  By supporting ESQL, content based filtering in WebSphere Message Broker Version 7.0 supports fully-qualified Namespaces, as documented in "ESQL reference" on page 5019.

ESQL expressions
: Content based filtering accepts any ESQL expression, with the exclusion of database access, that returns a Boolean value.

Related concepts:

"Content based filtering" on page 6409
Content based filtering allows a subscriber to filter messages based on their content.

Related tasks:

"Enabling content-based filtering with publish/subscribe"
It is possible to subscribe to messages, based on a selection of message payload content, in WebSphere Message Broker Version 7.0. You must enable content based filtering explicitly.

"Setting content-based filtering options using the WebSphere Message Broker Explorer" on page 2226
You can configure content based filtering using the WebSphere Message Broker Explorer.

Related reference:

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

Enabling content-based filtering with publish/subscribe:

It is possible to subscribe to messages, based on a selection of message payload content, in WebSphere Message Broker Version 7.0. You must enable content based filtering explicitly.

Before you begin

Before you start

- Read about Selectors on WebSphere MQ.
- Read the following overview of how WebSphere MQ selects content of messages.

  When an application publishes on a topic string, where one or more subscribers have a selection string selecting on the content of the message, WebSphere MQ requests that the extended message selection provider parse the publication and inform WebSphere MQ whether the publication matches the selection criteria specified by each subscriber with a content filter.

  If the extended message selection provider determines that the publication matches the subscriber's selection string, the message continues to be delivered to the subscriber.

  If the extended message selection provider determines that the publication does not match, the message is not delivered to the subscriber. This might cause the WebSphere MQ MQPUT or MQPUT1 call to fail with reason code MQRC_PUBLICATION_FAILURE. If the extended message selection provider is unable to parse the publication, reason code MQRC_CONTENT_ERROR is returned and the MQPUT or MQPUT1 call fails.

  If the extended message selection provider is unavailable or is unable to determine whether the subscriber should receive the publication, reason code MQRC_SELECTION_NOT_AVAILABLE is returned and theWebSphere MQ MQPUT or MQPUT1 call fails.

  When a subscription is being created with a content filter and the extended message selection provider is not available, the WebSphere MQ MQSUB call fails with reason code MQRC_SELECTION_NOT_AVAILABLE. If a subscription with a content filter is being resumed and the extended message selection provider is not available, the WebSphere MQ MQSUB call returns a warning of MQRC_SELECTION_NOT_AVAILABLE, but the subscription is allowed to be resumed.

- z/OS For z/OS systems, verify that the message broker started task ID has `UPDATE` access permission to profile `<MQ_QMNAME>.BATCH` of class `MQCONN`.

- Linux UNIX Windows On Linux, UNIX and Windows systems, grant authorization for the system to the queue manager: `setmqaut -t <qmgr> +system -p <brokerUserId>`

About this task

WebSphere Message Broker extends the message selection support provided by WebSphere MQ. WebSphere Message Broker does this by allowing ESQL statements rather than SQL92 statements, and filtering based on message content. See "Content based filtering using ESQL" on page 2223 for details of the scope and exclusions of supported ESQL.

The main external differences in the current implementation of content-based filtering are:

- Content based filtering is no longer limited to WebSphere MQ MQRFH2 subscribers. WebSphere Message Broker Version 7.0 provides content filtering services for the following WebSphere MQ subscribers:

  MQRFH2

  MQSUB

  If you are performing content filtering based on the *NameValueData* field, within an MQRFH2 header, on z/OS, the data can be present either in the first or second MQRFH2 header. For example, the filter:

  ```
  Root.MQRFH2.mcd.Msd='XML'
  ```

  might not work as you expect on z/OS. Use the following syntax to search all MQRFH2 headers:

  ```
  FOR ANY Root.MQRFH2[] AS I (I.mcd.Msd='XML')
  ```
- WebSphere MQ message properties are supported as part of the filter expression.
- If the publication does not contain an `mcd` folder, the payload is assumed to be XMLNSC.

Read the following steps to see how to enable content-based filtering on WebSphere Message Broker Version 7.0.

Procedure

1. Set the `cbfEnabled` property of the `ContentBasedFiltering` object for the execution group in which you want content-based filtering to run.

   You set the `cbfEnabled` property from the command line using either the:

   - **`mqsichangeproperties`** command; see "Content based filtering component parameter values" on page 3805 for more information, or
   - WebSphere Message Broker Explorer; see "Setting content-based filtering options using the WebSphere Message Broker Explorer" on page 2226 for more information.

   Note that you must explicitly enable the `cbfEnabled` property for content based filtering to work; the default setting is for content based filtering to be off.
2. Restart the execution group for the change to take effect.

   If you enable content based filtering in multiple execution groups on z/OS, content based filtering is active in only one execution group at any time. Subsequent execution groups for which content based filtering is enabled,

propagate the following messages to the syslog on start up (for each content based filtering thread) and then every 30 minutes, as they fail to connect to the queue manager:

```
BIP2111E MQ04BRK jheg1 15 MESSAGE BROKER INTERNAL ERROR: DIAGNOSTIC INFORMATION
 'Error occurred in Content Based Filtering Thread'. : ImbCbfWorker(909)
BIP2624E MQ04BRK jheg1 14 UNABLE TO CONNECT TO QUEUE MANAGER 'MQ04': MQCC=2;
 MQRC=2002; MESSAGE FLOW NODE 'ContentBasedFiltering'. : ImbCbfWorker(214)
```

If you stop the execution group that is currently providing content based filtering services, another execution group for which content based filtering is enabled connects to the queue manager and provides content based filtering services.

Both the `evaluationThreads` and `validationThreads` properties default to one if content-based filtering is enabled.

Evaluation threads are used to validate content filters against a given publication at publication time. A network with a high number of publications might require the `evaluationThreads` property to be increased (up to a maximum of 32) to handle the workload at publication time.

Validation threads are used to validate syntax of content filters at subscription time. A publish/subscribe network with a high number of subscribers, especially if dynamic subscribers, might require the `validationThreads` property to be increased (up to a maximum of 32) to handle the high throughput of subscription requests.

It is possible to enable this function in multiple execution groups, but you must ensure that any message sets required to parse any published message (and referenced in the `mcd` folder of that message) are deployed to all the execution groups that have been enabled for content-based filtering.

Any errors encountered parsing the message within the evaluation thread cause WebSphere MQ to return MQRC_CONTENT_ERROR to the publishing application. The parsing error appears as well in the event log as a WebSphere Message Broker exception.

Example of a subscription `<psc>` folder processing ESQL and message properties in the filter:

```
<psc>
 <Command>RegSub</Command><Topic>topic</Topic>
   <Filter>
   SUBSTRING(Root.XMLNSC.Name.FirstName FROM 1 FOR 1) = 'J' and usr.flag = 'yes'
   </Filter>
</psc>
```

Related concepts:

"Content based filtering using ESQL" on page 2223
WebSphere MQ supports message selection based on `Root.MQMD` and message properties; WebSphere Message Broker Version 7.0 provides extra options.

Related reference:

"Content based filtering component parameter values" on page 3805
Select the objects and properties associated with the content based filtering component that you want to change.

Setting content-based filtering options using the WebSphere Message Broker Explorer:

You can configure content based filtering using the WebSphere Message Broker Explorer.

About this task

To configure content based filtering:

Procedure

1. Open the WebSphere Message Broker Explorer.
2. Right-click the execution group to select the Properties dialog.
3. Select the ContentBasedFiltering tab.
4. Select **Enabled** from the drop down list to enable content based filtering.
5. Select the number of Validation (Subscription) Threads and Evaluation (Publication) Threads you require. Valid values for each of these options is a number in the range 1 - 32.

   Entering any other value causes an error to appear in the Properties dialog.

   If you have not enabled content based filtering, the fields controlling the thread values are grayed out.
6. Restart the execution group for the changes to take effect.

Results

You have configured content based filtering.

Related concepts:

"Content based filtering using ESQL" on page 2223
WebSphere MQ supports message selection based on `Root.MQMD` and message properties; WebSphere Message Broker Version 7.0 provides extra options.

Related tasks:

"Enabling content-based filtering with publish/subscribe" on page 2224
It is possible to subscribe to messages, based on a selection of message payload content, in WebSphere Message Broker Version 7.0. You must enable content based filtering explicitly.

Transforming and enriching messages

Transform and enrich messages by using one or more of the techniques described in this section.

About this task

Use one or more of following five options for transforming and enriching the messages in your message flows:

- Mappings
- ESQL
- Java
- XSL style sheets
- PHP

For more information about these options, and why you might choose one in preference to another, see "Client application programming interfaces" on page 1038. To use some of these options for transforming and enriching messages you might need to create a model of the messages that you want to transform. For information about how to develop a message model, and why you might want to develop a message model, see "Constructing message models" on page 2838.

For details of the tasks associated with these options, use the instructions provided in the following sections:
- "Using message mappings"
- "Developing ESQL" on page 2370
- "Developing Java" on page 2628
- "Using XSL Transform" on page 2669
- "Using PHP" on page 2670

Using message mappings

Message mappings define the blueprint for creating a message.

About this task

The following topics provide information about mapping:

Concept topics:
- "Message mapping overview" on page 2229
- "Advanced schema structures" on page 2230

Task topics:
- "Creating message mappings" on page 2232
- "Creating a message map file in the Broker Development view" on page 2233
- "Creating a message mapping file from a Mapping node" on page 2236
- "Configuring message mappings" on page 2237
- "Mapping from source: by selection" on page 2240
- "Mapping from source: by name" on page 2241
- "Mapping a target element from source message elements" on page 2254
- "Setting the value of a target element to a constant" on page 2257
- "Setting the value of a target element to a WebSphere MQ constant" on page 2258
- "Setting the value of a target element using an expression or function" on page 2261
- "Deleting a source or target element" on page 2264
- "Configuring conditional mappings" on page 2265
- "Configuring mappings for repeating elements" on page 2266
- "Populating a message map" on page 2269
- "Configuring the LocalEnvironment" on page 2271
- "Mapping headers and folders" on page 2271
- "Adding messages or message components to the source or target" on page 2273
- "Adding a database as a source or target" on page 2274
- "Modifying databases using message mappings" on page 2277
- "Creating and calling submaps and subroutines" on page 2298
- "Transforming a SOAP request message" on page 2311
- "Editing a default-generated map manually" on page 2312
- "Message mapping tips and restrictions" on page 2314
- "Message mapping scenarios" on page 2318

Reference topics:
- "Message mappings" on page 4981

- "Message Mapping editor" on page 4981
- "Mapping node" on page 4994

Message mapping overview

Message mappings define the blueprint for creating a message, where the created message is known as the target message.

Input and output objects are defined by reference to message models, which provide a definition of the message structure and type through the following components:

- Simple elements and attributes, which define the type, range, and default values
- Complex elements, which build the structure of the message
- Repeating simple or complex elements
- Other (embedded) messages

Messages can contain protocol-specific headers, which might have to be manipulated by WebSphere Message Broker. Dynamic setting of a message destination (routing) within the WebSphere Message Broker might also be required.

Values for target message elements can be derived from:

- Input message elements (the input message is also known as the source message)
- Database tables
- Constant values
- WebSphere MQ constants
- Functions supplied by the Mapping node
- User-defined functions

The logic to derive values can be simple or complex; conditional statements might be needed, as might loops, summations, and other functions. All of the mappings shown earlier can be achieved using a Mapping node.

You can also create a reusable form of map known as a submap. Submaps allow you to use a set of mapping functions in multiple maps to transform a common set of elements in the input object to the output object.

The Mapping node supports the following message domains:

MRM
XMLNSC
XMLNS
MIME
SOAP
DataObject
JMSMap
JMSStream
XML
BLOB
IDOC

The Mapping node does not support the JSON message domain, because no JSON message modeling is supported.

Related concepts:

"Client application programming interfaces" on page 1038
You can configure the nodes in your message flows to customize the behavior of those nodes by using one or more of the supported programming interfaces.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Using message mappings" on page 2228
Message mappings define the blueprint for creating a message.

Related reference:

"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.

"Mapping node" on page 4571
Use the Mapping node to construct one or more new messages and populate them with various types of information.

Advanced schema structures:

You can use several advanced schema structures in mappings.

This section contains information about the following subjects:

- "Substitution groups"
- "Wildcards"
- "Derived types"
- "List types" on page 2231
- "Union types" on page 2231

Substitution groups: A substitution group is an XML Schema feature that provides a means of substituting one element for another in an XML message. The element that can be substituted is called the *head* element, and the substitution group is the list of elements that can be used in its place.

The head element and any mapped substitutions are shown by default in the Source and Target panes of the Message Mapping editor. The mapped substitutions are listed beneath the head element. You can show and hide the substituting elements displayed in the Source and Target panes by selecting **Show Substituting Elements**. You create mappings to or from members of substitution groups in the same way as you would map other elements.

An abstract head element of a substitution group is not displayed and when substitution is blocked, the substitution group folder is not displayed.

Wildcards: A mapping that you perform to or from a wildcard results in a submap call. Specify the wildcard replacement when you choose the parameter of a submap call.

A wildcard element or attribute can be instantiated only with another element or attribute. The Message Mapping editor allows only a global element or attribute as a wildcard replacement.

Derived types: For an element of a given type, the base type and the mapped derived types are shown by default in the Source and Target panes of the Message

Mapping editor. All attributes and elements of the base and derived types are listed under each type respectively. You can show and hide the derived types displayed in the Source and Target panes by selecting **Show Derived Types**.

You create mappings to or from a derived type and its contents in the same way that you would map any type or type content. When you map a derived type element, the Message Mapping editor generates ESQL code with the appropriate xsi:type attribute.

List types: A list type is a way of rendering a repeating simple value. The notation is more compact than the notation for a repeating element and provides a way to have multi-valued attributes.

You map list type attributes or elements in the same way that you would map any other simple type attribute or element. Mapping between two list type elements is the same as mapping between any two simple type elements.

To transform between a list type and a non-list type, such as a repeating element, write an ESQL function, then package the function as a map. The Message Mapping editor automatically selects this submap as the default transformation for the list type.

Union types: A union type is the same as a union of two or more other simple types and it allows a value to conform to any one of several different simple types.

Use the Message Mapping editor to create mappings to or from union type attributes or elements in the same way as you would map atomic simple type attributes or elements, as shown in the following diagram:

```
<xsd:simpleType name="zipUnion">
 <xsd:union memberTypes="USState listOfMyIntType"/>
</xsd:simpleType>
<xsd:element name=zip type=zipUnion/>
```

Related concepts:

"Substitution groups in the message model" on page 1199
Substitution groups are an XML Schema feature that provides a way of substituting one element for another in an XML message.

"Message model objects: Wildcard attributes" on page 1187
For XML messages, a *wildcard attribute* enables unmodeled attributes to be present in a message.

"Message model objects: simple types" on page 1180
A *simple type* is an abstract definition of an item of data such as a number, a string, or a date.

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

"Client application programming interfaces" on page 1038
You can configure the nodes in your message flows to customize the behavior of those nodes by using one or more of the supported programming interfaces.

Related tasks:

"Creating a new submap for a wildcard source" on page 2300
You can map a wildcard value in the source to a wildcard value in the target.

"Creating and calling submaps and subroutines" on page 2298
Use submaps, ESQL subroutines, or both, to map source elements to target elements.

Related reference:

"Local element logical properties" on page 5442
The logical properties of a local element include properties that specify the number of occurrences and value of the local element.

"Wildcard element properties" on page 6064
Different types of properties are available for a wildcard element.

"Wildcard attribute properties" on page 6060
Different types of properties are available for a wildcard attribute.

Creating message mappings

You can create message mappings either in the Broker Development view of the Toolkit, or from a Mapping node.

The topics in this section describe how to create message mappings. Most actions can be achieved either by using the menu bar, or by right-clicking and choosing an action from a drop-down menu. For consistency, the following topics describe the menu bar method:

- "Creating a message map file in the Broker Development view" on page 2233
- "Creating a message mapping file from a Mapping node" on page 2236
- "Configuring message mappings" on page 2237
- "Mapping from source: by selection" on page 2240
- "Mapping from source: by name" on page 2241
- "Mapping a target element from source message elements" on page 2254
- "Setting the value of a target element to a constant" on page 2257
- "Setting the value of a target element to a WebSphere MQ constant" on page 2258
- "Setting the value of a target element using an expression or function" on page 2261
- "Creating a BLOB output message using a message map" on page 2262
- "Mapping from a BLOB message to an output message" on page 2263
- "Deleting a source or target element" on page 2264
- "Configuring conditional mappings" on page 2265
- "Configuring mappings for repeating elements" on page 2266
- "Populating a message map" on page 2269
- "Configuring the LocalEnvironment" on page 2271
- "Mapping headers and folders" on page 2271
- "Adding messages or message components to the source or target" on page 2273
- "Adding a database as a source or target" on page 2274
- "Modifying databases using message mappings" on page 2277
- "Creating and calling submaps and subroutines" on page 2298
- "Transforming a SOAP request message" on page 2311
- "Editing a default-generated map manually" on page 2312

Related concepts:

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

"Message mapping tips and restrictions" on page 2314
Information to help you use message mapping.

Related tasks:

"Message mapping scenarios" on page 2318

"Debugging mappings" on page 3185
When message flow processing has paused at a breakpoint that you have set in source code within a node that contains mappings, you can view the mapping routines and modify user-defined variables in the Flow Debugger.

Related reference:

"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.

For a basic introduction to mapping, see the IBM Redbooks publication WebSphere Message Broker Basics.

Creating a message map file in the Broker Development view:

Before you begin

You can create a message map file for use in your message flows in the Broker Development view. If you want to add a database to your message map file, you must have created a database definition for the database.

About this task

To create a message map (.msgmap) file in the Broker Development view:

Procedure

1. From the Broker Application Development perspective, click **File** > **New** > **Message Map**.

   Alternatively, in the Broker Development view, right-click the message flow project that you want to create the message map in and click **New** > **Message Map**.

   The New Message Map wizard opens.
2. Specify the Project, Name and Schema for the message map. The project list is filtered to show only projects in the active working set. Click **Next**.
3. Follow the on-screen instructions to complete the New Message Map wizard:
   a. On the **Select map kind and its source and target** pane, select the type of map you want to create.
      - If you select the option **Message map called by a message flow node**, a message map is created that can be accessed from a node. Properties are associated with any source or target messages, and you can select to include message headers and the LocalEnvironment with the message body.
      - If you select the option **Submap called by another map**, a message map is created that can be referenced from another message map. This is known as a submap and can contain components of a message body such as global elements, global attributes, and global types. A submap does not contain Properties, message headers, or the LocalEnvironment.

   b. Expand the Messages, Message Components, and Data Sources folders and their children, to display all the available options. Select the Messages, Message Components, Database Selects, Stored Procedures, and User Defined Functions that you want to use for your map from **Select Map Sources**. The stored procedure list, if any, shows the stored procedure signature, such as the procedure name, its parameter mode (IN/OUT/INOUT), parameter name, and the data type.

c. Expand the Messages and Data Targets folders and their children, to display all the available options. Select the Messages, Table Inserts, Table Updates and Table Deletes that you want to use as targets for your map from **Select Map Targets**.

The information contained inside a square bracket is the data source name, schema name, and the database project which contains the database definition.

Messages, data sources and data targets are filtered to show only resources from the active working set. If you cannot find the Messages, Message Components, Data Sources or Data Targets that you expect, clear the **Apply working set filtering to artifact selection(s) on the page** check box.

4. Select **Finish** to create the new message map. The "Message Mapping editor" on page 4981 opens with the selected sources and targets.

What to do next

After you have created a message map file, configure the message mappings. You must also configure the **Mapping routine** property on your mapping node to match the name of your new mapping file.

Related concepts:

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

Related tasks:

"Editing a default-generated map manually" on page 2312
A map that is generated by the Message Mapping editor might not do everything that you want; you can change maps manually to enhance its operation.

"Creating and calling submaps and subroutines" on page 2298
Use submaps, ESQL subroutines, or both, to map source elements to target elements.

"Configuring message mappings" on page 2237
Use the Message Mapping editor to configure a message mapping.

Related reference:

"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.

"Mapping node" on page 4571
Use the Mapping node to construct one or more new messages and populate them with various types of information.

"DataDelete node" on page 4382
Use the DataDelete node to interact with a database in the specified ODBC data source.

Analyzing the impact of changes to message maps:

Use impact analysis to analyze the effect of renaming or moving message maps.

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message flow project" on page 1425
- "Creating a message map file in the Broker Development view" on page 2233
- "Enabling and disabling indexing" on page 1454

About this task

This information covers the following tasks:

- "Renaming message maps"
- "Moving a message map"

Renaming message maps:
Procedure

1. In the Broker Development view, right-click the object that you want to rename, then click **Impact Analysis** > **Rename**.
2. In the Impact Analysis - Rename Artifact window, type the new name of the object, then click **Analyze Impact**.

   The Rename Artifact dialog box shows the results of impact analysis, listing primary and secondary impacts.

   You can view the results of impact analysis in the Impact Analysis dialog box, or the Impact Analysis view of the WebSphere Message Broker Toolkit. For more information about how to view information about selected resources, mark changes as complete, copy results to an external application, and view previous results, see "Impact Analysis view" on page 6801.

Moving a message map:
Procedure

1. In the Broker Development view, right-click the file, then click **Impact Analysis** > **Move**.
2. In the bottom panel, select the new folder for the object, then click **Analyze Impact**.

   The Impact Analysis - Move Artifact dialog box shows the results of impact analysis, listing primary and secondary impacts.

   You can view the results of impact analysis in the Impact Analysis dialog box, or the Impact Analysis view of the WebSphere Message Broker Toolkit. For more information about how to view information about selected resources, mark changes as complete, copy results to an external application, and view previous results, see "Impact Analysis view" on page 6801.

:

Related concepts:

"Impact analysis: analyzing the effects of planned changes to your applications" on page 1150
When developing an application, you might want to change the names of some artifacts, or move them. Impact analysis reports the artifacts that are likely to be affected by a particular change.

Related tasks:

"Enabling and disabling indexing" on page 1454
Enable indexing to support impact analysis.

"Analyzing planned changes to message model objects" on page 2897
Use impact analysis to analyze the effect of renaming message model objects.

"Analyzing planned changes to ESQL objects" on page 2403
Use impact analysis to analyze the effect of renaming or moving ESQL objects, or moving or changing the broker schema of an ESQL file.

"Analyzing planned changes to message flows" on page 1436
Use impact analysis to analyze the effect of renaming or moving message flows, including subflows.

"Analyzing planned changes to message set resources" on page 1165
Use impact analysis to analyze the effect of renaming message definition or deployable WSDL files.

Related reference:

"Impact analysis: reference" on page 4174
Some artifacts are excluded from secondary analysis.

Creating a message mapping file from a Mapping node:

You can use a Mapping node to create a message map with messages and databases as both sources and targets.

Before you begin

Before you create a mapping file, ensure that you complete the following tasks:

1. "Creating a message flow project" on page 1425.
2. "Creating a message flow" on page 1431.
3. Define message flow content that includes a Mapping node; for more information, see "Defining message flow content" on page 1488.

About this task

To create a message map (`.msgmap`) file from a Mapping node:

Procedure

1. Open your message flow from the Broker Application Development perspective.
2. Double-click the Mapping node, or right-click the Mapping node and click **Open Map**. The New Message Map wizard opens.
3. Select the combination of Messages, Data Sources, or both, that you want to use as sources for your map from Select map sources. Select the combination of Messages, Data Targets, or both, that you want to use as targets for your map from Select map targets.

   If you cannot find the Messages, Data Sources or Data Targets that you expect, clear the **Apply working set filtering to artifact selection(s) on the page** check box.
4. Click **OK** to create the new message map. The Message Mapping editor opens with the selected sources and targets, for more information see "Message Mapping editor" on page 4981.

What to do next

After you have created a message map file, you can configure the message mappings, see "Configuring message mappings" on page 2237.

Related concepts:

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

Related tasks:

"Editing a default-generated map manually" on page 2312
A map that is generated by the Message Mapping editor might not do everything that you want; you can change maps manually to enhance its operation.

"Creating and calling submaps and subroutines" on page 2298
Use submaps, ESQL subroutines, or both, to map source elements to target elements.

"Configuring message mappings"
Use the Message Mapping editor to configure a message mapping.

Related reference:

"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.

"Mapping node" on page 4571
Use the Mapping node to construct one or more new messages and populate them with various types of information.

"DataDelete node" on page 4382
Use the DataDelete node to interact with a database in the specified ODBC data source.

Configuring message mappings:

Use the Message Mapping editor to configure a message mapping.

About this task

When you have created a map, you can edit it using the Message Mapping editor.

The editor provides the ability to set values for:

- Message destination
- Message content
- Message headers

See the "Mapping node" on page 4571 topic for more information about how to set the properties of a Mapping node.

Wizards and dialog boxes are provided for tasks such as adding mappable elements and working with submaps. Mappings that are created with the Message Mapping editor are validated and compiled automatically, ready to be added to a broker archive (BAR) file, and for subsequent deployment to WebSphere Message Broker.

Use the Message Mapping editor to perform the following tasks:

Common tasks:

- "Mapping from source: by selection" on page 2240
- "Mapping from source: by name" on page 2241
- "Mapping a target element from source message elements" on page 2254
- "Mapping a target element from database tables" on page 2288
- "Mapping a target element from database stored procedures" on page 2290
- "Mapping a target element from database user-defined functions" on page 2292
- "Setting the value of a target element to a constant" on page 2257
- "Setting the value of a target element to a WebSphere MQ constant" on page 2258
- "Setting the value of a target element to an ESQL constant" on page 2260
- "Setting the value of a target element using an expression or function" on page 2261

- “Configuring conditional mappings” on page 2265
- “Configuring mappings for repeating elements” on page 2266

Message destination tasks: You might map a destination so that the destination can be set dynamically, by setting values in LocalEnvironment.Destination. You can also retrieve information after a message has been sent, by accessing information in LocalEnvironment.WrittenDestination.

- “Configuring the LocalEnvironment” on page 2271
- “Mapping headers and folders” on page 2271

Message content tasks:

- “Adding messages or message components to the source or target” on page 2273
- “Adding a database as a source or target” on page 2274
- “Showing or hiding substituting elements in the Message Mapping editor”
- “Showing or hiding derived types in the Message Mapping editor” on page 2239

Message header tasks:

- “Configuring message headers” on page 2271
- “Mapping headers and folders” on page 2271

Related concepts:

“Message Mapping editor” on page 4981
You use the Message Mapping editor to create and edit message mappings.

Related tasks:

“Editing a default-generated map manually” on page 2312
A map that is generated by the Message Mapping editor might not do everything that you want; you can change maps manually to enhance its operation.

Related reference:

“Message mappings” on page 4981
Edit and configure message maps using the Message Mapping editor.

Showing or hiding substituting elements in the Message Mapping editor:

You can use the **Show Substituting Elements** dialog box to show and hide substituting elements in the Message Mapping editor.

About this task

The head element and any mapped substitutions are shown by default in the Mapping editor. You can use the **Show Substituting Elements** dialog to show and hide the substituting elements in the Mapping editor:

Procedure

1. In either the Source or the Target pane, right-click the element that you want to show or hide the substituted elements for. If an element has substituted elements, it is displayed as a substitutions folder in the Source or the Target pane.
2. Click **Show Substituting Elements** on the pop-up menu. The Show Substituting Elements dialog box is displayed.

3. In the Show Substituting Elements dialog box, select elements to show them in the Message Mapping editor, or clear them to hide them in the Message Mapping editor. You cannot hide any element that is already used in a mapping.
4. Click **OK**.

Results

The elements that you have chosen to show or to hide are stored as preferences in your workspace.

Related concepts:

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

"WebSphere Message Broker Toolkit" on page 31
The WebSphere Message Broker Toolkit is an integrated development environment and graphical user interface based on the Eclipse platform.

Related tasks:

"Configuring message mappings" on page 2237
Use the Message Mapping editor to configure a message mapping.

"Adding a database as a source or target" on page 2274
Add a database as a source, and database tables as targets, to message maps that support database mappings.

Related reference:

"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.

"Mapping node" on page 4571
Use the Mapping node to construct one or more new messages and populate them with various types of information.

Showing or hiding derived types in the Message Mapping editor:

You can use the **Show Derived Types** dialog box to show and hide derived types in the Message Mapping editor.

About this task

For an element of a given type, only the base type and any mapped derived types are shown in the Message Mapping editor. You can use the **Show Derived Types** dialog to show and hide derived types:

Procedure

1. In either the Source or the Target pane, right-click the element that you want to show or hide the derived types for. If an element has derived types it is displayed as a specializations folder in the Source or the Target pane.
2. Click **Show Derived Types** on the pop-up menu. The Show Derived Types dialog box is displayed.
3. In the Show Derived Types dialog box, select elements to show them in the Message Mapping editor, or clear them to hide them in the Message Mapping editor. You cannot hide any element that is already used in a mapping.
4. Click **OK**.

Results

The elements that you have chosen to show or to hide are stored as preferences in your workspace.

Related concepts:

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

"WebSphere Message Broker Toolkit" on page 31
The WebSphere Message Broker Toolkit is an integrated development environment and graphical user interface based on the Eclipse platform.

Related tasks:

"Configuring message mappings" on page 2237
Use the Message Mapping editor to configure a message mapping.

"Adding a database as a source or target" on page 2274
Add a database as a source, and database tables as targets, to message maps that support database mappings.

Related reference:

"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.

"Mapping node" on page 4571
Use the Mapping node to construct one or more new messages and populate them with various types of information.

Mapping from source: by selection:

You can map from source fields by using Map from Source, or by using the drag-and-drop method.

About this task

Using Map from Source

Procedure

1. Select the source and target elements that you want to map by clicking them. (Ctrl+click to select multiple source or target elements.)
2. Click **Map** > **Map from Source**.

   There are four possible scenarios that result in mapping by selection using Map from Source.

   - If more than one mappable source element is selected, the selected sources are mapped to the selected target.
   - If more than one mappable target element is selected, the selected source is mapped to the selected targets.
   - If one mappable source and one mappable target are selected, and neither element has any children, the selected source is mapped to the selected target.
   - If one mappable source and one mappable target are selected, where both the elements have children and the same type definition, the selected source is mapped to the selected target.
   - If one mappable source and one mappable target are selected, and one of them is an element with mixed content and the other does not have any children, the mixed content is mapped with the other selected item.

What to do next

Using the drag-and-drop method

Drag the appropriate source element or elements onto the target element or elements (Ctrl+click to select multiple source or target elements.)

When you use the drag-and-drop method to map from source, mapping by selection is always performed. You can use the drag-and-drop method in the following scenarios:

- More than one mappable source element is selected. In this case, the selected sources are mapped to the selected target.
- More than one mappable target element is selected. In this case, the selected source is mapped to the selected targets.
- One mappable source and one mappable target are selected, and neither element has any children. In this case, the selected source is mapped to the selected target.
- One mappable source and one mappable target are selected, and one of them is an element with mixed content and the other does not have any children. In this case, the selected mixed content is mapped to the other selected item.
- The selected source and target elements have the same type definition, or the source type is derived from the target type. In this case the entire structure below the element is copied.

In other scenarios, when a mapping by selection is not possible, the Map by Name wizard opens to enable a Map by Name mapping to be performed instead.

Related concepts:

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

"Creating message mappings" on page 2232
You can create message mappings either in the Broker Development view of the Toolkit, or from a Mapping node.

Related tasks:

"Mapping from source: by name"
The Map by Name wizard is used to map complex types by examining the names of source elements and target elements to create mappings.

"Mapping a target element from source message elements" on page 2254

"Mapping a target element from database tables" on page 2288
To map a target element from a database table, set up the Mapping node to retrieve the relevant rows from the database and populate the message target elements with values from database.

"Mapping a target element from database stored procedures" on page 2290
Use a DB2 or Oracle database stored procedure as a source in a Mapping node.

"Mapping a target element from database user-defined functions" on page 2292
Use an Oracle database user-defined function as a source in a Mapping node.

Mapping from source: by name:

The Map by Name wizard is used to map complex types by examining the names of source elements and target elements to create mappings.

About this task

The Map by Name wizard can also be used to map database columns. The following steps describe how to map from source using the Map by Name wizard.

Using the Map by Name wizard

Procedure

1. Click the source and target complex elements, database, schema, table, database stored procedure, or database user-defined function that you want to map.
2. Click **Map** > **Map by Name**. The Map by Name wizard opens to allow you to perform mapping by name.
3. Choose the appropriate option from the Select how to map from source to target wizard:
   - **Map leaves of the selected nodes**. This option maps the leaves of the source element to the leaves of the target element, that match each other; this option is selected by default.
   - **Map immediate children of the selected nodes**. This option maps only the immediate children of the source element to the immediate children of the target element that match each other. This option is available only when the selected source and target elements have immediate children that are mappable.
4. After selecting the **Map leaves of the selected nodes** or **Map immediate children of the selected nodes** option, specify how names are matched.
   - **Case sensitive**. This option enables you to select whether you want to match the case sensitivity of the name; this option is not selected by default.
   - **Alphanumeric characters (Letters and digits only)**. This option excludes special characters (for example & and -) from the name; this option is selected by default.

   The preceding two options are independent of one another, and you can select their values separately.
5. Specify the Mapping Criteria between the sources and targets.

   If the source and target names that you are using satisfy more than one of the following options, the order in which names are matched is:

   a. Same
   b. Synonym
   c. Similar

   Any target that is matched during an earlier step is excluded from name matching in a later step.

   - **Create mappings between sources and targets with the same name**. This option matches items of the same name, and is selected by default.

     Whether the two names are considered to be the same, depends on your selections for **Case sensitive** and **Alphanumeric characters (Letters and digits only)**.

     For example, if you have used the default options for **Case sensitive** and **Alphanumeric characters (Letters and digits only)**, GIVEN_NAME and GivenName are considered to be a match.

     However, if you have selected **Case sensitive** as well as **Alphanumeric characters (Letters and digits only)**, the two names are considered to be identical only if they contain the same alphanumeric characters in the same order, and the characters are of the same case.

See “Mapping by Same Name” on page 2245 for further information.

- **Create mappings when source and target names are more similar than**. This option allows you to specify how similar two words have to be to create a mapping between them by varying the result from zero to 100 percent. The result is displayed and the default value is 60; see “Sample similarity values” on page 2245 for some examples of how similar words are matched to one another.
- **Create mappings between source and target names defined as synonyms in file**. This option allows you to create mappings for word pairs that are defined in a synonym file. A synonym file is a flat text file with file extension `.txt` or `.csv`. See “Creating and using a synonym file” on page 2251 for further information about how you create a synonym file from a spreadsheet written in Microsoft Excel.

  See “Format of the synonym file” on page 2247 for further information about the synonym file itself, and “Algorithm used to match synonyms” on page 2252 for further information about the methods that are used to match synonyms in a synonym file.

6. Click **Finish** to complete the process, or **Next** to obtain further options.

What to do next

The Map by Name wizard opens automatically when you use the drag-and-drop method to map from source where the source and target are complex types with different type definitions, or where the source type is not derived from the target type.

Related concepts:

“Message Mapping editor” on page 4981
You use the Message Mapping editor to create and edit message mappings.

“Creating message mappings” on page 2232
You can create message mappings either in the Broker Development view of the Toolkit, or from a Mapping node.

Related tasks:

“Mapping from source: by selection” on page 2240
You can map from source fields by using Map from Source, or by using the drag-and-drop method.

“Mapping a target element from source message elements” on page 2254

“Mapping a target element from database tables” on page 2288
To map a target element from a database table, set up the Mapping node to retrieve the relevant rows from the database and populate the message target elements with values from database.

“Mapping a target element from database stored procedures” on page 2290
Use a DB2 or Oracle database stored procedure as a source in a Mapping node.

“Mapping a target element from database user-defined functions” on page 2292
Use an Oracle database user-defined function as a source in a Mapping node.

“Creating and using a synonym file” on page 2251
You can create a synonym file manually, or by generating a synonym file from the information that is contained in a Microsoft Excel spreadsheet, using the following set of instructions.

“Mapping by Same Name” on page 2245
Options are available when you select **Create mappings between sources and targets with the same name** or a similar name.

Related reference:

"Format of the synonym file" on page 2247
`Map by Name` allows you to create mappings between specific sources and targets by putting the names of the sources and targets in a file called the synonym file.

"Algorithm used to match synonyms" on page 2252
The way in which synonyms are matched by the `Map by Name` function, to create mappings between specific sources and targets, follows a particular set of rules.

"Sample similarity values" on page 2245
The following table lists words that are similar to one another, together with their similarity value in percent.

Selecting matches:

Use the Map by Name wizard to select the mappings that you want to create.

About this task

When you have specified how you require the names to be matched on the initial panel of the Map by Name wizard, and have selected **Next**, you see a panel that displays all the matches found.

You can now select the options that you require:

Procedure

1. Select a row in the **Selectable Mapping Targets** column that you want to change. Selecting a folder tree node results in the entire tree branch being selected or not selected.
2. Click **Edit** to start the **Select Mapping Source** dialog.
3. To select a mapping target, select the appropriate tree node check box. Conversely, to remove a mapping target, clear the appropriate tree node check box.
4. Ensure that you have selected only the number of matches that you require. The third column displays the number of sources selected for each mapping target. The cell has a value greater than one when the source of a map contains several elements of certain names under various containers, and the source names match to the same target name.
5. Click **Finish** to complete the mapping process, or click **Back** to change the matches that you have set up. When you click **Finish**, you obtain a warning message if either, or both, of the following conditions apply:
   a. More than a few sources to map to the same target.
   b. Many targets for which you want to create mappings.

Related concepts:

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

"Creating message mappings" on page 2232
You can create message mappings either in the Broker Development view of the Toolkit, or from a Mapping node.

Related tasks:

"Mapping from source: by selection" on page 2240
You can map from source fields by using Map from Source, or by using the drag-and-drop method.

"Mapping a target element from source message elements" on page 2254

"Mapping a target element from database tables" on page 2288
To map a target element from a database table, set up the Mapping node to retrieve the relevant rows from the database and populate the message target elements with values from database.

"Mapping a target element from database stored procedures" on page 2290
Use a DB2 or Oracle database stored procedure as a source in a Mapping node.

"Mapping a target element from database user-defined functions" on page 2292
Use an Oracle database user-defined function as a source in a Mapping node.

"Using message mappings" on page 2228
Message mappings define the blueprint for creating a message.

Sample similarity values:

The following table lists words that are similar to one another, together with their similarity value in percent.

Word1	Word2	Similarity value %
catalog	catalogue	85
anasthesia	anaesthesia	84
recognize	recognise	75
color	colour	66
theater	theatre	66
tire	tyre	33
intro	introduction	53
abbr	abbreviation	42
name	fullname	60
firstname	fullname	40
id	identification	14
NCName	Non colonized name	40
USA	United States of America	0
faq	frequently asked questions	0

Related tasks:

"Mapping from source: by name" on page 2241
The Map by Name wizard is used to map complex types by examining the names of source elements and target elements to create mappings.

"Selecting matches" on page 2244
Use the Map by Name wizard to select the mappings that you want to create.

Mapping by Same Name:

Options are available when you select **Create mappings between sources and targets with the same name** or a similar name.

About this task

When you select **Create mappings between sources and targets with the same name**, the following rules apply:

1. Any target field that has a fixed value is excluded in name matching. Any target that is already mapped, or under a container that is already mapped, is excluded from name matching.
2. If a source and a target have the same name, it is a match, regardless of the kind of, and XSD type of, the source and target. An element, an attribute, and a database column can all form a match if their names are the same.
3. XML namespaces are excluded from name matching. Therefore, `abc:something` and `xyz:something` are considered the same, as are `{http://www.abc.com}:something` and `{http://www.xyz.com}:something`.
4. When multiple sources have the `same` name as one target, one mapping is created. However, when multiple targets have the `same` name as one source, multiple mappings are created, each for one source and one target.
5. When performing `Map from Source` for a selected source and a selected target, the WebSphere Message Broker Toolkit might insert some `for` and `if` statements based on the repeatability (maximum occurrences) of the selected source and target, and their containers.

   The same process occurs for `Map by Name`, based on the repeatability of the selected sources and targets, and their containers.

   However, there are not any `for` or `if` statements inserted on descendants of the selected source and target.
6. When you select the **Map leaves of the selected nodes** option, the following steps are taken to match names:

   a. Compare the relative path and item name of the selected source or target

   b. Compare the item name without relative path

   For example, if you invoke the action **Create mappings between sources and targets with the same name** and have a:

   - Source path for `partNum` of `$source/po:purchaseOrder/`**`items`**`/item/partNum`, where **items** is the selection that you made in the source.
   - Target path for `partNum` of `$target/`**`po:purchaseOrder`**`/items/item/partNum`, where **po:purchaseOrder** is the selection that you made in the target.

   During step a) the source and target path names involved in the same-name test are `/items/item/partNum` and `item/partNum`.

   During step b) the source and target item names that are involved in same-name test are `partNum` and `partNum`; that is, name matches are done using short names without their paths.

   Note that sources and targets matched in a previous step do not participate in a later step.

Mapping by similar name

1. Fixed value targets and mapped targetstargets and mapped targets are excluded in name matching; see Point 1 in the preceding section.
2. The *similarity* test is done using the name of an element, an attribute, or a database column regardless of its type; see Point 2 in the preceding section.
3. The *similarity* test applies in the same way to case sensitivity and alphanumeric characters as for **Mapping by same name**.
4. Namespace or namespace prefixes do not participate in the *similarity* test; see Point 3 in the preceding section.
5. The behavior for the situation when multiple sources are similar to one target, and when multiple targets are similar to one source, is the same as Point 4 in the preceding section.

6. The repeatability (maximum occurrences) of containers and descendants of the selected source and target are handled in the same way as in Point 5 in the preceding section.
7. When you select the option **Create mappings when source and target names are more similar than**, you must also select **Create mappings between sources and targets with the same name**.
8. When you select **Map leaves of the selected nodes**, the following steps are taken to match names.

   Sources and targets matched in a previous step are ignored in later steps:
   - The path names starting after the selected source and target are the same.
   - The item names excluding the path are the same.
   - The item names excluding the path are similar.
9. You can select the similarity threshold for two words to be considered similar.
10. You cannot use any other similarity algorithm.

Related concepts:

"Creating message mappings" on page 2232
You can create message mappings either in the Broker Development view of the Toolkit, or from a Mapping node.

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

Related tasks:

"Mapping from source: by name" on page 2241
The Map by Name wizard is used to map complex types by examining the names of source elements and target elements to create mappings.

"Selecting matches" on page 2244
Use the Map by Name wizard to select the mappings that you want to create.

"Creating and using a synonym file" on page 2251
You can create a synonym file manually, or by generating a synonym file from the information that is contained in a Microsoft Excel spreadsheet, using the following set of instructions.

"Using message mappings" on page 2228
Message mappings define the blueprint for creating a message.

Related reference:

"Algorithm used to match synonyms" on page 2252
The way in which synonyms are matched by the `Map by Name` function, to create mappings between specific sources and targets, follows a particular set of rules.

"Format of the synonym file"
`Map by Name` allows you to create mappings between specific sources and targets by putting the names of the sources and targets in a file called the synonym file.

Format of the synonym file:

`Map by Name` allows you to create mappings between specific sources and targets by putting the names of the sources and targets in a file called the synonym file.

Synonyms, in the context of the synonym file, are groups of words that represent mappings that you want to create.

File type:
A synonym file can reside anywhere in your file system, only if the encoding used in the synonym file is the same as that used by the Eclipse Toolkit system.

However, if the synonym file uses a specific encoding that is, or might be, different from the encoding of the Eclipse Toolkit, the file must reside in a project in the WebSphere Message Broker Toolkit.

If the synonym file is created outside the WebSphere Message Broker Toolkit, and uses a specific encoding, save the file under a WebSphere Message Broker Toolkit project and click **Refresh** to make the file visible in the navigator.

The synonym file uses Tab-separated or comma-separated files only. If you have written your mapping requirement in any external application, for example, Microsoft Word or Microsoft Excel, you must export the relevant data in a format that the synonym file supports.

Item names in the file:
A synonym file contains the names of items to be mapped, without the path to the item or the namespace of the item.

For example, if you want to map `partNum` to `partNumber` in the following XML, you must put `partNum` in the synonym file, not `item/partNum`, `items/item/partNum`, or `purchaseOrder/items/item/partNum`.

```
<po:purchaseOrder xmlns:po="http://www.ibm.com">
 <items>
    <item>
         <partnum>100-abc</partnum>
         <productName>Acme Integrator</productName>
         <quantity>22</quantity>
         <USPrice>100.99</USPrice>
         <po:comment>Acme Integrator</po:comment>
         <shipDate>2008-12-01</shipDate>
    </item>
 </items>
</po:purchaseOrder>
```

Synonyms in the file can:

- Be case sensitive or not case sensitive
- Contain the entire mapping item name
- Have non alphanumeric characters removed

Rows in the synonym file:
In the synonym file, each row represents one group of names to be mapped between each other and each row must contain at least two names. Names within a row are separated by commas in `.csv` files, and by Tab characters in `.txt` files.

A synonym file can contain an optional special row at the top. This top row contains key words **Source**, **Target**, or **Source_Target**, separated by the same delimiter used in the remainder of the file. The top row is used to indicate whether the synonyms are to be used to match names in mapping the source or the target:

- If the first word in the top row is **Target**, the first name only, in each subsequent row is searched in the mapping target for name matching.
- If the second word in the top row is **Source**, the second name only, in each subsequent row is searched in the mapping source for name matching.

- If the third word in the top row is **Source_Target**, the third name only, in each subsequent row is searched in both the mapping source and mapping target for name matching.

The top row must not contain fewer key words than the maximum number of names in any row in the file.

If the top row contains any word other than **Source**, **Target**, or **Source_Target**, the top row is ignored and it is assumed that the top row is missing. If you omit the optional top row, every name in the synonym file is considered to be **Source_Target**; that is, any name found either in the mapping source or in the mapping target is matched.

If a synonym file contains two rows:

car	*automobile*
automobile	*vehicle*

car and *vehicle* are not considered to be synonyms.

In order to make all three words synonyms, your synonym file can have either:

- One row with all three words -

car	*automobile*	*vehicle*

or

- Three rows -

car	*automobile*
automobile	*vehicle*
car	*vehicle*

Special characters:
You can write synonym files manually, or export them from another application; for example, Microsoft Excel. Item names in synonym files reflect the application domain and do not have to match exactly the names in the XML schema or the relational database column.

For example, a synonym file might contain the row:

```
summer      l'été
```

As `l'été` does not conform to the XML NCName format, you could name the element `l_été`. As long as all the alphanumeric characters in the synonym file match those in the schema, you can use the file with the option **Letters and digits only, ignore non-alphanumeric characters**.

Many mapping requirements are written in Microsoft Excel, and cells in a Microsoft Excel file might contain specific characters like double quotation marks, space, new line, comma, and so on. When such a Microsoft Excel file is saved as a Tab-separated or comma-separated file, they contain additional double quotation marks.

Two groups of synonyms in a synonym file are delimited either by a Line Feed (LF) character, or Line Feed followed by a Carriage Return (LFCR). A Carriage Return (CR) character by itself does not end a group of synonyms.

Leading and trailing space characters adjacent to the delimiter (comma or Tab character) are ignored. Blank rows, or rows that contain only space characters, are permitted and ignored in a synonym file.

Different editors might inject different space characters into a synonym file; spaces are not used to delimit synonyms, and spaces are ignored unless they are inside double quotation marks.

If a synonym contains a comma, a double quotation mark, a carriage return, or a leading or trailing space that is significant, the synonym must be enclosed in double quotation marks. A double quotation mark within a synonym is escaped with another double quotation mark. For example:

```
"comma,separated"
"double""quote"
  "with<CR>
   newline"
  "  spaces   "
```

When the synonym file is read by the WebSphere Message Broker Toolkit, the double quotation marks at the beginning and end of the synonym are removed and the WebSphere Message Broker Toolkit stores the following in the synonym table:

```
comma,separated
double"quote
  with<CR>newline
    spaces
```

The WebSphere Message Broker Toolkit reads a synonym file containing these special characters correctly, and you should select the **Letters and digits only, ignore non-alphanumeric characters** option when using the synonym file.

Related concepts:

"Creating message mappings" on page 2232
You can create message mappings either in the Broker Development view of the Toolkit, or from a Mapping node.

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

Related tasks:

"Mapping from source: by name" on page 2241
The Map by Name wizard is used to map complex types by examining the names of source elements and target elements to create mappings.

"Selecting matches" on page 2244
Use the Map by Name wizard to select the mappings that you want to create.

"Mapping by Same Name" on page 2245
Options are available when you select **Create mappings between sources and targets with the same name** or a similar name.

"Using message mappings" on page 2228
Message mappings define the blueprint for creating a message.

"Creating and using a synonym file" on page 2251
You can create a synonym file manually, or by generating a synonym file from the information that is contained in a Microsoft Excel spreadsheet, using the following set of instructions.

Related reference:

"Algorithm used to match synonyms" on page 2252
The way in which synonyms are matched by the `Map by Name` function, to create mappings between specific sources and targets, follows a particular set of rules.

Creating and using a synonym file:

You can create a synonym file manually, or by generating a synonym file from the information that is contained in a Microsoft Excel spreadsheet, using the following set of instructions.

About this task

The following set of instructions describes how to create a synonym file where the original mapping requirement is written in Microsoft Excel. If your original requirement is written in a table in Microsoft Word you must copy and paste the table into Microsoft Excel before you begin.

Amend the process to allow for extra facilities that your enterprise uses.

Procedure

1. Select the section of the Microsoft Excel spreadsheet that you require. For example, if you have a Product that you want to map to a Part number, select that section of the spreadsheet.
2. Remove all columns from the spreadsheet, except the ones containing the source field name and the target field name.

   You might have to edit some of the cells. For example, if your mapping instruction includes the phrase `based on`, remove this phrase.
3. If the source or target fields contain paths, remove the paths to leave only the short names of the item.

   However, it is helpful to sort the column before removing the paths. Sorted path names can indicate which is the best source or target to select when invoking the action.

   If all the interested sources (or targets) start with the same path prefix, you might consider selecting the lowest source (or target) node in the tree which has that common path prefix.
4. Remove all rows that do not have a source field name and a target field name. For example, if you have an obsolete product that no longer has a part number and you have put `n/a` in the source, remove that row.
5. Select the **Save As** function in Microsoft Excel to save the spreadsheet into a format acceptable by our WebSphere Message Broker Toolkit.

   You can use either a Tab delimited `.txt` file or a comma delimited `.csv` file.

   A comma delimited file can be opened using Microsoft Excel and it looks like the original Microsoft Excel file; the file can also be viewed using a text editor.
6. Create the mappings using the synonym file.

   Select the options in the **Map by Name** wizard that match your requirements; for example, select the default options of **Map leaves of the selected nodes** and **Alphanumeric characters (Letters and digits only)**.

   When you have chosen these options, select **Create mappings between source and target names defined as synonyms in file**.

   If you need to map same-name sources to targets, and the synonym file does not contain rows with those names (for example a row with **car,car**), select the

Create mappings between sources and targets with the same name option, in addition to **Create mappings between source and target names defined as synonyms in file**.

You can select both **Create mappings between sources and targets with the same name** and **Create mappings when source and target names are more similar than**, in addition to **Create mappings between source and target names defined as synonyms in file**, if your synonym file does not contain a row **color,colour** and you want to map between them.

7. Click **Finish**.
8. Edit mapping expressions based on what is required.

   For example, if you have the following value:

   ```
   PRODUCT  $SOURCE/Batch/Detail/Replace/PartNumber
   ```

   edit the mapping expression to:

   ```
   xs:boolean($source/Batch/Detail/Replace/PartNumber = 1)
   ```

9. Create all the mappings that the Map by Synonym process has not generated. Use the drag-and-drop method, **Map from Source**, or the **Enter Expression** process.

Related concepts:

"Creating message mappings" on page 2232
You can create message mappings either in the Broker Development view of the Toolkit, or from a Mapping node.

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

Related tasks:

"Mapping from source: by name" on page 2241
The Map by Name wizard is used to map complex types by examining the names of source elements and target elements to create mappings.

"Selecting matches" on page 2244
Use the Map by Name wizard to select the mappings that you want to create.

"Mapping by Same Name" on page 2245
Options are available when you select **Create mappings between sources and targets with the same name** or a similar name.

"Using message mappings" on page 2228
Message mappings define the blueprint for creating a message.

Related reference:

"Format of the synonym file" on page 2247
`Map by Name` allows you to create mappings between specific sources and targets by putting the names of the sources and targets in a file called the synonym file.

"Algorithm used to match synonyms"
The way in which synonyms are matched by the `Map by Name` function, to create mappings between specific sources and targets, follows a particular set of rules.

Algorithm used to match synonyms:

The way in which synonyms are matched by the `Map by Name` function, to create mappings between specific sources and targets, follows a particular set of rules.

1. Fixed value targets and mapped targets are excluded in name matching; see point 1 in “Mapping by Same Name” on page 2245.
2. The synonym matching is done by using the name of an element, an attribute, or a database column regardless of its type; see point 2 in “Mapping by Same Name” on page 2245.
3. The synonym matching of alphanumeric characters is not case-sensitive, and is identical to that used in “Mapping from source: by name” on page 2241.
4. Namespace or namespace prefixes do not participate in synonym matching; see point 3 in “Mapping by Same Name” on page 2245.
5. The behavior for the situation when multiple sources are synonyms of one target, and when multiple targets are synonyms of one source, is the same as point 4 in “Mapping by Same Name” on page 2245.
6. The repeatability (maximum occurrences) of containers and descendants of the selected source and target are handled in the same way as in point 5 in “Mapping by Same Name” on page 2245.
7. If a source and a target have the `same` name, they are not considered a match under the option for synonyms. If you require a mapping for same-name sources and targets, you must also select the **same name** option.
8. In addition to mapping synonyms, you might want to create mappings for some, but not all, same-name sources and targets. In this case, you have two options:
   - Clear **Create mappings between sources and targets with the same name**, and include the same-name sources and targets in the synonym file
   - Select **Create mappings between sources and targets with the same name**, and clear the unwanted mappings on the second page of the wizard.
9. When you select **Map leaves of the selected nodes** together with both same name and synonym mapping options, the following steps are taken to match names.

   Sources and targets matched in a previous step do not participate in a later step:
   - The path names starting after the selected the source and target are the same
   - The item names excluding the path are the same
   - The item names excluding the path are synonyms
10. When you select **Map leaves of the selected nodes**, and you require synonyms to be mapped without mapping same names, only the item names are checked for synonyms; paths are ignored.

Related concepts:

“Creating message mappings” on page 2232
You can create message mappings either in the Broker Development view of the Toolkit, or from a Mapping node.

“Message mapping overview” on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

“Message Mapping editor” on page 4981
You use the Message Mapping editor to create and edit message mappings.

Related tasks:

“Mapping from source: by name” on page 2241
The Map by Name wizard is used to map complex types by examining the names of source elements and target elements to create mappings.

"Selecting matches" on page 2244
Use the Map by Name wizard to select the mappings that you want to create.

"Mapping by Same Name" on page 2245
Options are available when you select **Create mappings between sources and targets with the same name** or a similar name.

"Creating and using a synonym file" on page 2251
You can create a synonym file manually, or by generating a synonym file from the information that is contained in a Microsoft Excel spreadsheet, using the following set of instructions.

"Using message mappings" on page 2228
Message mappings define the blueprint for creating a message.

Related reference:

"Format of the synonym file" on page 2247
`Map by Name` allows you to create mappings between specific sources and targets by putting the names of the sources and targets in a file called the synonym file.

Mapping a target element from source message elements:
About this task

You can map the following objects:

- Simple source elements to simple target elements
- Source structures to target structures (where the source and target are of the same type)
- Source structures to target structures (where the source and target are of a different type)
- Simple source elements to target structures (where the target structure is a complex type element with mixed content)
- Source structures to simple target elements (where the source structure is a complex type element with mixed content)
- Multiple simple source elements to a simple target element

The following sections describe how to perform mapping for these particular scenarios using the Message Mapping editor.

Mapping simple source elements to simple target elements

In the following example, the source element called Name does not contain the same children as the target element called Name:

Source	Target
Name Title First_name Middle_name Last_name	Name Title First_names Last_name

To map one of the child elements, drag the element from the Source pane onto the corresponding element in the Target pane; for example, drag the Last_name source element onto the Last_name target element.

The mapping is represented by a line between the source element and the target element and an entry for the mapping in Xpath format appears in the Spreadsheet pane. A triangular icon indicates which elements in the Source and Target panes have been mapped.

Mapping source structures to target structures (where the source and target are of the same type)

In the following example, the source element called Name has the same structure as the target element called Name:

Source	Target
Name Title First_name Middle_name Last_name	Name Title First_name Middle_name Last_name

To map the entire source structure to the target structure, drag the parent element (Name) from the Source pane onto the corresponding element (Name) in the Target pane. All the child elements are mapped.

Mapping source structures to target structures (where the source and target are of a different type)

In the following example, the source element called Name has a different structure to the target element called DifferentName:

Source	Target
Name Title First_name Middle_name Last_name	DifferentName Title FirstName LastName

To map the entire source structure to the target structure, drag the parent element (Name) from the Source pane onto the corresponding element (DifferentName) in the Target pane. The Map By Name wizards opens. Select **Map leaves** and **Map items of same and similar names** to map all child elements in the target. The source element Middle_name will not be mapped, as there is no target element with the same or a similar name.

Mapping simple source elements to target structures

In the following example, source element *Name1* contains three simple elements to be mapped: *Title*, *FirstName*, and *LastName*. The target element *Name2* contains two attributes, *Title* and *GivenName*, and uses mixed content to carry *Surname*.

Source	Target
Name1 Title FirstName LastName	Name2 (mixed) @Title @GivenName

The following mappings can be created using **Map from Source** or drag and drop:

```
Title <- -> @Title
FirstName <- -> @GivenName
LastName <- -> Name2
```

If the following source XML document is used:

```
<Name1>
 <Title>Mr.</Title>
 <FirstName>John</FirstName>
 <LastName>Doe</LastName>
</Name1>
```

the following target XML document will be produced from the mappings:

```
<Name2 GivenName="John" Title="Mr.">Doe</Name2>
```

Mapping source structures to simple target elements

In the following example, the source element *Name1* contains two attributes, *Title* and *GivenName*, and uses mixed content to carry *Surname*. The target element *Name2* contains three simple elements to be mapped to.

Source	Target
Name1 (mixed) @Title @GivenName	Name2 Title FirstName LastName

The following mappings can be created using **Map from Source** or drag and drop:

```
@Title <- -> Title
@GivenName <- -> FirstName
Name1 <- -> LastName
```

If the following source XML document is used:

```
<Name1 GivenName="John" Titile="Mr.">Doe</Name1>
```

the following target XML document will be produced from the mappings:

```
<Name2>
 <Title>Mr.</Title>
 <FirstName>John</FirstName>
 <LastName>Doe</LastName>
</Name2>
```

Mapping multiple source elements to a simple target element

In the following example, you want to concatenate the *First_name* and *Middle_name* source elements to form a single target element called *First_names*:

Source	Target
Name Title First_name Middle_name Last_name	Name Title First_names Last_name

To map multiple source elements to a simple target element, Ctrl+click the appropriate source elements (First_name and Middle_name) and the target element (First_names), then click **Map** > **Map from Source**. A concatenate function appears in the Spreadsheet pane; you can edit this function to define how the concatenated target element looks, for example, by adding a white space between the two source elements.

Results

To customize the target element (for example, to make the target value equal to the source value plus one), see "Setting the value of a target element using an expression or function" on page 2261. You cannot map a simple element if one of its ancestors also has a mapping. For example, you cannot map Properties from source to target, then map Properties/MessageFormat.

Related concepts:

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

"Creating message mappings" on page 2232
You can create message mappings either in the Broker Development view of the Toolkit, or from a Mapping node.

Related tasks:

"Mapping from source: by selection" on page 2240
You can map from source fields by using Map from Source, or by using the drag-and-drop method.

"Mapping from source: by name" on page 2241
The Map by Name wizard is used to map complex types by examining the names of source elements and target elements to create mappings.

"Setting the value of a target element using an expression or function" on page 2261

Related reference:

"Message Mapping editor Source pane" on page 4983
Details of the elements present in the Source pane of the Message Mapping Editor.

"Message Mapping editor Target pane" on page 4987
Details of the elements present in the Target pane of the Message Mapping Editor.

"Message Mapping editor Spreadsheet pane" on page 4991
Details of the actions available to you in the Spreadsheet pane of the Message Mapping Editor.

"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.

Setting the value of a target element to a constant:

Use the Message Mapping editor to set the value of a target element to a constant.

About this task

Procedure

1. In the Target pane, right-click the target element or attribute and click **Enter Expression**. If the target element or attribute has a default value, this value is added to the Edit pane.
2. Enter the required constant in the Edit pane and click **Enter**. When entering the constant, observe the following rules:
   - Enclose string element values in single quotation marks.
   - Enter numeric element values without quotation marks.
   - For boolean element values enter `0` for false or `1` for true, without quotation marks. Alternatively, you can enter the `fn:false()` function for false, or the `fn:true()` function for true.

The Spreadsheet pane is updated with the value that you have defined.

Results

You cannot set a value for a simple element if one of its ancestors also has a mapping. For example, you cannot map Properties from source to target, then set a value for `Properties/MessageFormat`.

Example

You can also set a target element to a WebSphere MQ constant or an ESQL constant.

Related concepts:

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

"Creating message mappings" on page 2232
You can create message mappings either in the Broker Development view of the Toolkit, or from a Mapping node.

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

"Mapping node functions" on page 4997
You can configure your message mappings to use a variety of predefined and user-defined functions.

Related tasks:

"Setting the value of a target element using an expression or function" on page 2261

"Using message mappings" on page 2228
Message mappings define the blueprint for creating a message.

"Setting the value of a target element to a WebSphere MQ constant"
There are two ways to set the value of a target element to a WebSphere MQ constant, depending on whether the target element has an entry in the Map Script column of the Message Mapping editor Spreadsheet pane.

"Setting the value of a target element to an ESQL constant" on page 2260
There are two ways to set the value of a target element to an ESQL constant, depending on whether the target element has an entry in the Map Script column of the Message Mapping editor Spreadsheet pane.

Related reference:

"Message Mapping editor Target pane" on page 4987
Details of the elements present in the Target pane of the Message Mapping Editor.

"Message Mapping editor Edit pane" on page 4990
Details of how you use the Edit pane of the Message Mapping Editor.

"Message Mapping editor Spreadsheet pane" on page 4991
Details of the actions available to you in the Spreadsheet pane of the Message Mapping Editor.

"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.

Setting the value of a target element to a WebSphere MQ constant:

There are two ways to set the value of a target element to a WebSphere MQ constant, depending on whether the target element has an entry in the Map Script column of the Message Mapping editor Spreadsheet pane.

About this task

Procedure

- If the target element has an entry in the Map Script column:
  1. In the Spreadsheet pane, select the target element.
  2. Enter `$mq:` followed by the WebSphere MQ constant in the Edit pane.
  3. Press **Enter**.

  The Spreadsheet pane is updated with the expression for a WebSphere MQ constant.
- If the target element does not have an entry in the Map Script column:
  1. In the Target pane, right-click the target element and click **Enter Expression**.
  2. Enter `$mq:` followed by the WebSphere MQ constant in the Edit pane.
  3. Press **Enter**.

  The Spreadsheet pane is updated with the expression for a WebSphere MQ constant.

Example

The following examples demonstrate how to enter a WebSphere MQ constant in the Edit pane:

```
$mq:MQ_MSG_HEADER_LENGTH
$mq:MQMD_CURRENT_VERSION
```

When the map is saved a warning message is displayed if the expression entered for the WebSphere MQ constant is incorrect, for example, if the constant is not recognized. This is an example of the warning message: `The target "$target/purchaseOrder/comment" is not referencing a valid variable`.

Content Assist (**Edit** > **Content Assist**) provides a list of the WebSphere MQ constants available.

1. Select `$mq: ( MQ constants )`
2. Select **Edit** > **Content Assist** again to display a list of the available constants.

WebSphere MQ constants that can be used as values for target elements, grouped by the parameter or field to which they relate, can be found in the *WebSphere MQ Constants* book.

Related concepts:

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

"Mapping node functions" on page 4997
You can configure your message mappings to use a variety of predefined and user-defined functions.

"Mapping node syntax" on page 4995

Related tasks:

"Setting the value of a target element to a constant" on page 2257
Use the Message Mapping editor to set the value of a target element to a constant.

"Setting the value of a target element using an expression or function" on page 2261

Related reference:

"Message Mapping editor Target pane" on page 4987
Details of the elements present in the Target pane of the Message Mapping Editor.
"Message Mapping editor Edit pane" on page 4990
Details of how you use the Edit pane of the Message Mapping Editor.
"Message Mapping editor Spreadsheet pane" on page 4991
Details of the actions available to you in the Spreadsheet pane of the Message Mapping Editor.
"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.

Setting the value of a target element to an ESQL constant:

There are two ways to set the value of a target element to an ESQL constant, depending on whether the target element has an entry in the Map Script column of the Message Mapping editor Spreadsheet pane.

About this task

Procedure

- If the target element has an entry in the Map Script column:
  1. In the Spreadsheet pane, select the target element.
  2. Enter `$esql:` followed by the ESQL constant in the Edit pane.
  3. Press **Enter**.

  The Spreadsheet pane is updated with the expression for a WebSphere MQ constant.
- If the target element does not have an entry in the Map Script column:
  1. In the Target pane, right-click the target element and click **Enter Expression**.
  2. Enter `$esql:` followed by the ESQL constant in the Edit pane.
  3. Press **Enter**.

  The Spreadsheet pane is updated with the expression for a WebSphere MQ constant.

Example

The following examples demonstrate how to enter an ESQL constant in the Edit pane:

```
$esql:ValidateLocalError
$esql:ParseComplete
```

When the map is saved a warning message is displayed if the expression entered for the ESQL constant is incorrect, for example, if the constant is not recognized. This is an example of the warning message: `The target "$target/purchaseOrder/comment" is not referencing a valid variable`.

Content Assist (**Edit** > **Content Assist**) provides a list of the WebSphere MQ constants available.

1. Select `$esql: ( ESQL Constants )`
2. Select **Edit** > **Content Assist** again to display a list of the available constants.

Related concepts:

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

"Mapping node functions" on page 4997
You can configure your message mappings to use a variety of predefined and user-defined functions.

"Mapping node syntax" on page 4995

Related tasks:

"Setting the value of a target element to a constant" on page 2257
Use the Message Mapping editor to set the value of a target element to a constant.

"Setting the value of a target element using an expression or function"

Related reference:

"Message Mapping editor Target pane" on page 4987
Details of the elements present in the Target pane of the Message Mapping Editor.

"Message Mapping editor Edit pane" on page 4990
Details of how you use the Edit pane of the Message Mapping Editor.

"Message Mapping editor Spreadsheet pane" on page 4991
Details of the actions available to you in the Spreadsheet pane of the Message Mapping Editor.

"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.

"ESQL constants" on page 5302
Use these constants to make or parse a bit stream.

Setting the value of a target element using an expression or function:

About this task

There are two ways to set the value of a target element to an expression, depending on whether the target element has an entry in the Map Script column of the Message Mapping editor Spreadsheet pane:

Procedure

- If the target element has an entry in the Map Script column:
  1. In the Spreadsheet pane, select the target element.
  2. Enter the required expression in the Edit pane.
  3. Press **Enter**.

  The Spreadsheet pane is updated with the value or expression.
- If the target element does not have an entry in the Map Script column:
  1. In the Target pane, right-click the target element and click **Enter Expression**. If the target element has a default value, this value is added to the Edit pane.
  2. Enter the required expression in the Edit pane.
  3. Press **Enter**.

  The Spreadsheet pane is updated with the value or expression.

Example

The following examples demonstrate techniques for entering mapping expressions in the Edit pane.

- If the target element is derived from a source element, drag the source element or elements onto the Edit pane; for example:

  ```
  $source/Properties/MessageSet
  ```

- Use arithmetic expressions, such as:
  ```
  $source/Properties/Priority + 1
  ```
- Use mapping, Xpath or ESQL function names. Content Assist (**Edit** > **Content Assist**) provides a list of available functions. For example:
  ```
  esql:upper($source/Properties/ReplyIdentifier)
  ```
- You can perform casting in the Edit pane; for example:
  ```
  xs:string($source/Properties/CodedCharSetId)
  ```

You cannot enter an expression for a simple element if one of its ancestors also has a mapping. For example, you cannot map Properties from source to target, then set a value of Properties/MessageFormat.

Related concepts:

"Mapping node functions" on page 4997
You can configure your message mappings to use a variety of predefined and user-defined functions.

"Mapping node syntax" on page 4995

Related tasks:

"Setting the value of a target element to a constant" on page 2257
Use the Message Mapping editor to set the value of a target element to a constant.

"Setting the value of a target element to a WebSphere MQ constant" on page 2258
There are two ways to set the value of a target element to a WebSphere MQ constant, depending on whether the target element has an entry in the Map Script column of the Message Mapping editor Spreadsheet pane.

Related reference:

"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.

Creating a BLOB output message using a message map:

Use the Message Mapping editor to create a bit stream from a message source, and create it as a BLOB output message.

Before you begin

Before you start:

Create a mapping that includes a BLOB message as a target; see "Creating a message mapping file from a Mapping node" on page 2236.

About this task

Take the following steps:

Procedure

1. Right-click the BLOB message that you want to map in the Target pane, and select **Enter Expression** from the menu.
2. In the Edit pane, type `esql:asbitstream()`.
3. Drag the source field to the Edit pane, placing it between the parentheses, for example:
   ```
   esql:asbitstream($source/po:purchaseOrder)
   ```

Alternatively, you can use content assist to select the `esql:asbitstream` function. In the Edit pane press Ctrl+Space to display a list of available functions and associated parameters. The `asbitstream` function is an ESQL Field function. The function can take other parameters; see "ESQL mapping functions" on page 4998.

When you move the cursor out of the Edit pane, or press Enter, the mapping is displayed between the fields in the Source and Target panes.

Related tasks:

"Storing a BLOB message in a database table using a message map" on page 2297
Use the Message Mapping editor to create a bit stream from a BLOB message, and store it in a database table.

"Mapping from a BLOB message to an output message"
Use the Message Mapping editor to parse a BLOB message.

"Mapping from a BLOB field in a database table to an output message" on page 2298
Use the Message Mapping editor to parse a bit stream from a field in a database table into a folder in a target message.

Related reference:

"ESQL mapping functions" on page 4998
Some predefined ESQL functions are available for use with message maps.

"Message Mapping editor Edit pane" on page 4990
Details of how you use the Edit pane of the Message Mapping Editor.

Mapping from a BLOB message to an output message:

Use the Message Mapping editor to parse a BLOB message.

Before you begin

Before you start:

Create a mapping; see "Creating a message mapping file from a Mapping node" on page 2236.

About this task

Take the following steps:

Procedure

1. Right-click the element in the target pane, and select **Enter Expression** from the menu.
2. In the Edit pane, type `msgmap:element-from-bitstream()`.
3. Drag the BLOB to the Edit pane, placing it between the parentheses, for example:

   ```
   msgmap:element-from-bitstream($source/BLOB)
   ```

   Alternatively, you can use content assist to select the `msgmap:element-from-bitstream` function. In the Edit pane press Ctrl+Space to display a list of available functions and associated parameters. The function can take other parameters; see "Predefined mapping functions" on page 5007. When you move the cursor out of the Edit pane, or press Enter, the mapping is displayed between the fields in the Source and Target panes.

Related tasks:

"Creating a message mapping file from a Mapping node" on page 2236
You can use a Mapping node to create a message map with messages and databases as both sources and targets.

"Storing a BLOB message in a database table using a message map" on page 2297
Use the Message Mapping editor to create a bit stream from a BLOB message, and store it in a database table.

"Creating a BLOB output message using a message map" on page 2262
Use the Message Mapping editor to create a bit stream from a message source, and create it as a BLOB output message.

"Mapping from a BLOB field in a database table to an output message" on page 2298
Use the Message Mapping editor to parse a bit stream from a field in a database table into a folder in a target message.

Related reference:

"Predefined mapping functions" on page 5007
Some predefined mapping functions are provided for use with message maps.

Deleting a source or target element:
About this task

The following steps describe how to delete source and target elements using the "Message Mapping editor" on page 4981:

Procedure

- To delete a source path, modify the expression so that it no longer uses the source value to compute the target.

  If this is the last use of the source path, the line linking the source and target is removed. If the expression no longer has any value, the target becomes unmapped.
- To delete a target from the Edit pane, click the target and click **Delete**.

  The target structure is preserved if possible.
  - If you delete a "for" row, clicking **Delete** removes the single row.
  - If you have an `if`, an `elseif`, or an `else` statement, clicking **Delete**:
    1. On an `elseif` statement deletes the `elseif` statement and the contents of the statement.
    2. On an `else` statement deletes the `else` statement and the contents of the statement.
    3. On an `if` statement with a subsequent `elseif` statement, deletes the `if` statement and the contents of the statement, and causes the subsequent `elseif` statement to become an `if` statement.
    4. On an `if` statement with only a subsequent `else` statement, deletes everything except the contents of the `else` statement.
    5. On an `if` statement with no subsequent statements, deletes everything except the contents of the `if` statement.
- To delete a database source, click the SELECT statement then remove all references to the source manually. Alternatively, delete the SELECT source in the Source pane then remove all references to the source manually.
- To delete a database target, delete the INSERT, UPDATE or DELETE statement. Alternatively, update or delete the statement in the Target pane.

Related concepts:

“Message Mapping editor” on page 4981
You use the Message Mapping editor to create and edit message mappings.

“Message mapping overview” on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

Related tasks:

“Using message mappings” on page 2228
Message mappings define the blueprint for creating a message.

Related reference:

“Message mappings” on page 4981
Edit and configure message maps using the Message Mapping editor.

Configuring conditional mappings:

How to set the value of a target element conditionally in a Mapping node.

Procedure

1. In the Spreadsheet pane of the Message Mapping editor, select the target element and click **Map** > **If**.

   One row is added to the Spreadsheet pane, above the target element:

   - In ths row, Map Script is set to "if".

     Its value is an expression that is evaluated to see whether it is true. If true, the target element is set to the value specified in its "Value" column. Initially, its "Value" column is set to ‘fn:true()’, which means that the condition is always met, and the target element is always set to the "Value" column.

2. Change the expression in the if row’s "Value" column by selecting the cell, or the if row, in the Spreadsheet pane, and setting the value in the Edit pane.

   Amend the expression in the Edit pane to specify the correct condition statement by performing the following steps:

   a. Select any database columns that are pertinent to the condition statement, and drag them from the Source pane into the Edit pane.
   b. Select any source message elements with values that are pertinent to the condition statement, and drag them from the Source pane into the Edit pane.
   c. Open Content Assist by clicking **Edit** > **Content Assist** and select the functions to be applied to the condition.

3. Add further condition statements by selecting the if row in the Spreadsheet pane, and clicking **Map** > **Else If**.

   Two rows are added to Spreadsheet pane, below the target element:

   - In the first row, Map Script is set to `elseif`. Process this as described in Step 2.
   - In the second row, Map Script is set to the target element. Its `Value` cell is initially blank. Set this value as described in “Setting the value of a target element to a constant” on page 2257, and “Setting the value of a target element using an expression or function” on page 2261.

4. To set the value of a target element when the `if` statement is not true, select the `if` statement for the target element in the Spreadsheet pane, and click **Map** > **Else**.

   Two rows are added to Spreadsheet pane, below the target element:

   - In the first row, Map Script is set to `else`. You cannot enter anything in the Value column of this row.

- In the second row, Map Script is set to the target element; its value is initially blank. Set this value as described in “Setting the value of a target element to a constant” on page 2257, and “Setting the value of a target element using an expression or function” on page 2261.

Related tasks:

“Setting the value of a target element to a constant” on page 2257
Use the Message Mapping editor to set the value of a target element to a constant.

“Setting the value of a target element using an expression or function” on page 2261

Related reference:

“Message mappings” on page 4981
Edit and configure message maps using the Message Mapping editor.

Configuring mappings for repeating elements:
About this task

To configure the Mapping node to process repeating elements, use the ‘For’ option in the Message Mapping editor Spreadsheet pane. The following combinations of repeating elements are possible:

Procedure

- repeating source and non-repeating target
- non-repeating source and repeating target
- repeating source and repeating target

Results

By default, if the source is a database, it is processed as a repeating source.

Related tasks:

“Configuring a repeating source and a non-repeating target”
To map a repeating source element to a non-repeating target element, drag elements between the Message Mapping editor Source and Target panes.

“Configuring a non-repeating source and a repeating target” on page 2268
To map a non-repeating source element to a repeating target element, drag elements between the Message Mapping editor Source and Target panes.

“Configuring a repeating source and a repeating target” on page 2269

Related reference:

“Message mappings” on page 4981
Edit and configure message maps using the Message Mapping editor.

Configuring a repeating source and a non-repeating target:

To map a repeating source element to a non-repeating target element, drag elements between the Message Mapping editor Source and Target panes.

About this task

The following items appear in the Spreadsheet pane:

- A "for" row with Value set to the repeating source element.
- An "if" row with Value set to `msgmap:occurrence($source/...) = 1`.
- A row with Map Script set to the target field and Value set to the source field.

The first occurrence of the source field is mapped to the target field. The "for"' row specifies that a loop is to be iterated for the specified repeating element. The if row restricts the logic to a single occurrence of the repeating element. See “Configuring conditional mappings” on page 2265 for more information on conditional logic in a mapping node.

Procedure

1. To map an occurrence other than the first, change the expression in the if row to `msgmap:occurrence($source/...) = n`, where *n* is the occurrence that you want to map.

   If the repeating source field is within one or more repeating structures, a hierarchy of `for` and `if` rows is placed in the Spreadsheet pane, one for each level of repetition.
2. If the source field contains a numeric data type, mapping all occurrences of a repeating source field to a non-repeating target results in the sum of all the source elements. Perform this mapping by selecting the source element and target element and clicking **Map** > **Accumulate**.

   This action sets the following value in the Spreadsheet pane for the target element:

   ```
   fn:sum($source/...)
   ```

   The result of the accumulate action is a numeric value. If your target has a different data type, you must cast the result to the appropriate type for the selected target. For example, if your target is xs:string type, you must alter the results of the accumulate action from `fn:sum($source/x/y/z)` to `xs:string(fn:sum($source/x/y/z))`, in order to cast the result to the correct data type for your target.

   You cannot map different occurrences of a repeating source element to different non-repeating target elements.

Related concepts:

“Message Mapping editor” on page 4981
You use the Message Mapping editor to create and edit message mappings.

“Creating message mappings” on page 2232
You can create message mappings either in the Broker Development view of the Toolkit, or from a Mapping node.

Related tasks:

“Configuring conditional mappings” on page 2265
How to set the value of a target element conditionally in a Mapping node.

Related reference:

“Message Mapping editor Source pane” on page 4983
Details of the elements present in the Source pane of the Message Mapping Editor.

“Message Mapping editor Target pane” on page 4987
Details of the elements present in the Target pane of the Message Mapping Editor.

“Message Mapping editor Spreadsheet pane” on page 4991
Details of the actions available to you in the Spreadsheet pane of the Message Mapping Editor.

“Message mappings” on page 4981
Edit and configure message maps using the Message Mapping editor.

Configuring a non-repeating source and a repeating target:

To map a non-repeating source element to a repeating target element, drag elements between the Message Mapping editor Source and Target panes.

About this task

The first occurrence of the target element is set to the value of the source element.

To map to an occurrence other than the first, complete the following steps:

Procedure

1. If the target element is not shown in the Spreadsheet pane, right-click its lowest ancestor row, then click **Insert Children**. Repeat this action until the target element is shown.
2. Right-click the target element and click **Insert Sibling After** or **Insert Sibling Before** to select the location to insert the repeating target elements. The **Insert Sibling After** or **Insert Sibling Before** options are not enabled if there is nothing valid to be inserted at the selected location. Selecting either of these opens the Insert Sibling Statement wizard.
3. Select the element to insert from the list of valid items.
4. Enter the number of instances to be added and click **OK**. The number of instances to be added must be less than or equal to the maximum occurrence specified for the selected element.

Results

The specified number of instances of the repeating target element are added to the Spreadsheet pane. The inserted statements do not have a mapping expression and any children are not displayed. Right-click each element, then click **Insert Children** to display any child elements.

What to do next

By repeating the **Insert Sibling After** and **Insert Sibling Before** action, it is possible to insert more repeating elements in the target than the maximum occurrence specifies. Verify that the number of repeating elements is valid, and delete any unwanted entries.

Related concepts:

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

"Creating message mappings" on page 2232
You can create message mappings either in the Broker Development view of the Toolkit, or from a Mapping node.

Related reference:

"Message Mapping editor Source pane" on page 4983
Details of the elements present in the Source pane of the Message Mapping Editor.

"Message Mapping editor Target pane" on page 4987
Details of the elements present in the Target pane of the Message Mapping Editor.

"Message Mapping editor Edit pane" on page 4990
Details of how you use the Edit pane of the Message Mapping Editor.

"Message Mapping editor Spreadsheet pane" on page 4991
Details of the actions available to you in the Spreadsheet pane of the Message Mapping Editor.

"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.

Configuring a repeating source and a repeating target:
About this task

To map a repeating source element to a repeating target element drag elements between the Message Mapping editor Source and Target panes. The following items appear in the Spreadsheet pane:

Procedure

- A 'for' row with Value set to the repeating source element.
- A row with Map Script set to the target field and Value set to the source field.

Results

All occurrences of the source element are mapped to the respective occurrences of the target element. You can map repeating source structures to repeating target structures if the source and target are of the same complex type.

Related concepts:

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

"Creating message mappings" on page 2232
You can create message mappings either in the Broker Development view of the Toolkit, or from a Mapping node.

Related reference:

"Message Mapping editor Source pane" on page 4983
Details of the elements present in the Source pane of the Message Mapping Editor.

"Message Mapping editor Target pane" on page 4987
Details of the elements present in the Target pane of the Message Mapping Editor.

"Message Mapping editor Spreadsheet pane" on page 4991
Details of the actions available to you in the Spreadsheet pane of the Message Mapping Editor.

"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.

Populating a message map:

Use the Insert Children wizard to add elements from the Target pane to the Spreadsheet pane. The Insert Children wizard creates child structures for the selected parent structure.

About this task

When you add a message target to a message map, $target in the Spreadsheet pane is populated by default with Properties and the message body root. The Properties fields MessageSet, MessageType and MessageFormat, are added together with their default values, unless the selected message is in the BLOB domain. Other message elements and their children can be added to the Spreadsheet pane without creating

mappings by using the Insert Children wizard. The following steps show how to populate the Spreadsheet pane with other message elements using the Insert Children wizard:

Using the Insert Children wizard

Procedure

1. Right-click a parent element in the Spreadsheet pane and click **Insert Children**. The Insert Children wizard is displayed.
2. Select the items you want to create mappings for. Items required in the target message are selected by default. The selected items are added to the Spreadsheet pane.
3. Repeat **Insert Children** to add further child elements to the Spreadsheet pane.

Results

If any target elements are missing warning messages are displayed in the Message Mapping Editor. These warning messages indicate the name and expected position of the missing elements. You can used the Insert Children wizard to add the missing elements.

What to do next

You can also use the Insert Children wizard to add target elements to the Spreadsheet pane when there are existing mappings. Any existing mappings are not altered by the wizard.

If the target map is a submap the Spreadsheet pane is populated by default with the selected element or attribute root. You can use the Insert Children wizard in the submap to add any child elements to the Spreadsheet pane in the same way.

Related concepts:

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

"Creating message mappings" on page 2232
You can create message mappings either in the Broker Development view of the Toolkit, or from a Mapping node.

Related tasks:

"Mapping from source: by selection" on page 2240
You can map from source fields by using Map from Source, or by using the drag-and-drop method.

"Mapping from source: by name" on page 2241
The Map by Name wizard is used to map complex types by examining the names of source elements and target elements to create mappings.

"Mapping a target element from source message elements" on page 2254

"Mapping a target element from database tables" on page 2288
To map a target element from a database table, set up the Mapping node to retrieve the relevant rows from the database and populate the message target elements with values from database.

"Mapping a target element from database stored procedures" on page 2290
Use a DB2 or Oracle database stored procedure as a source in a Mapping node.

"Mapping a target element from database user-defined functions" on page 2292
Use an Oracle database user-defined function as a source in a Mapping node.

Configuring the LocalEnvironment:
About this task

You can set values in the LocalEnvironment in the same way as setting values in other elements of a message. Add the LocalEnvironment to your message map using the **Add or Remove Headers and Folders** dialog as described in "Mapping headers and folders." If you set any values in the target LocalEnvironment, set the mapping mode property for the Mapping node to a value that contains LocalEnvironment. To do this, select the mapping node in your message flow and click **Properties** > **Basic** > **Mapping Mode**.

You cannot map Local Environment objects that are not listed.

Related concepts:

"Local environment tree structure" on page 1056
The local environment tree is a part of the logical message tree in which you can store information while the message flow processes the message.

Related tasks:

"Configuring message mappings" on page 2237
Use the Message Mapping editor to configure a message mapping.

Related reference:

"Mapping node" on page 4571
Use the Mapping node to construct one or more new messages and populate them with various types of information.

"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.

Configuring message headers:

You can set values for headers in the same way as setting values in other elements of a message.

About this task

Add the appropriate headers to your message map using the **Add or Remove Headers and Folders** dialog box as described in "Mapping headers and folders." If you set any values in the target LocalEnvironment, set the Mapping mode property for the Mapping node to a value that contains LocalEnvironment. To do this, select the mapping node in your message flow and click **Properties** > **Basic** > **Mapping Mode**.

You cannot map headers that are not listed.

Related tasks:

"Mapping headers and folders"
Include message headers and folders for source and target messages in a message map.

Mapping headers and folders:

Include message headers and folders for source and target messages in a message map.

Before you begin

Before mapping headers and folders, ensure that you do the following tasks:

1. Create a message flow project
2. Create a message flow
3. Define message flow content
4. Create a message map file from the navigator or create a message map from a node.

About this task

The following types of message headers and folders can be included for source and target messages in a message map (note that a submap does not include message headers):

- LocalEnvironment
- Properties
- MQ Headers
- HTTP Headers
- JMS Transport Header
- Email Headers

If you choose not to map message headers or the LocalEnvironment explicitly in your message map, the output message is produced with the same message headers as the input message. When you Populate the message map, the Properties folder for the source and target are displayed in the message map, with MessageSet and MessageType initially set based on the target message.

MessageFormat is set to the default wire format of the message set if the parser domain is MRM. The other properties are blank initially, and the message headers are copied from the input message.

Alternatively, if you choose to map any message headers or the LocalEnvironment in your message map, no message headers are copied from the input message. You must add mappings for these headers to ensure that the target message contains appropriate headers to make a valid output message.

If your target message contains an MQRFH2 header, you must select from either the MQRFH2 or MQRFH2C parser in the Add or Remove Headers and Folders dialog. For more information about the MQRFH2 and MQRFH2C parsers, see "The MQRFH2 and MQRFH2C parsers" on page 4253.

To add message headers or other folders to a message map:

Procedure

1. Right-click your message map in the Broker Development view and select **Open** or right-click your mapping node and select **Open Map** to open the Message Mapping editor.
2. Right-click `$source` in the Source pane and select **Add or Remove Headers and Folders** to add message headers or other folders to the source message. The Add or Remove Headers and Folders dialog box opens.
3. Ensure that **Selected headers and other folders** is selected. If **No folders (map body element only)** is selected your map is a submap, and cannot have

headers associated with it. You can change the submap to a message map by selecting **Selected headers and other folders**.

4. Select the headers that you want to map from the list. If you want to map MQ Headers or HTTP Headers, you must select individual headers by expanding the list. If you are using MQ Headers you must include the MQMD, and so this is automatically selected for you.
5. Click **OK** to add the selected message headers or folders to the message map.
6. Right-click `$target` in the Target pane and select **Add or Remove Headers and Folders** to add message headers or other folders to the output message.
7. Repeat steps 3 to 5 to add the headers and folders that you require to the target message.
8. Configure the message header and folder mappings in the same way as other mappings.

What to do next

You can use **Add or Remove Headers and Folders** to remove message headers or the LocalEnvironment folder. Right-click on either the `$source` or the `$target` to open the Select Message Headers dialog box. Clear the headers or other folders to remove them from the message map. Removing a message header or other folder from the message map removes any associated mappings that you have created. You can remove the Properties folder from the message map, but all built-in parsers require some values in the Properties folder for the output message.

You can map multiple instances of a header by right-clicking on the header in the Message Mapping editor Spreadsheet pane and selecting **Insert Sibling Before** or **Insert Sibling After**. Select the header from the Insert Sibling Statement dialog.

Related concepts:

"Message tree structure" on page 1045
The message tree is a part of the logical message tree in which the broker stores its internal representation of the message body.

"Local environment tree structure" on page 1056
The local environment tree is a part of the logical message tree in which you can store information while the message flow processes the message.

Related tasks:

"Creating a message map file in the Broker Development view" on page 2233

"Configuring message mappings" on page 2237
Use the Message Mapping editor to configure a message mapping.

"Configuring the LocalEnvironment" on page 2271

Related reference:

"Headers and Mapping node" on page 5018
This topic lists the headers that can be manipulated by the Mapping node.

Adding messages or message components to the source or target:
About this task

You can add additional messages or message components as sources or targets in your message map. To add a message or message component to a source or target:

Procedure

1. From the Message Mapping editor, click **Map** > **Add Sources and Targets**
   The Add Map Sources and Targets wizard opens.

Alternatively, right-click in the Source pane and click **Add Sources** or right-click in the Target pane and click **Add Targets**.

2. Select messages or message components from the message sets that are in your WebSphere Message Broker Toolkit workspace.

   If you cannot find the messages or message components that you expect, clear the **Apply working set filtering to artifact selection(s) on the page** check box.

   If one does not already exist, a project reference is created from your message flow project to the message set project that contains the selected messages or message components.

Results

You can also add sources and targets by dragging the resources from the Broker Development view in the Broker Application Development perspective onto the source or target pane of your message map. Select resources under Messages or Elements and Attributes or Types from your Message Definitions and drag them onto the source or target pane. If you add a message to the message map, Properties are also added. If you add an element, attribute or type to the to the message map a global element for a submap is created. Your message map must use messages, global elements or global types, but not a combination of more than one type.

What to do next

A Mapping node can have only one source message, but can have several target messages. Therefore, you cannot add a source message if one already exists.

Related concepts:

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

"WebSphere Message Broker Toolkit" on page 31
The WebSphere Message Broker Toolkit is an integrated development environment and graphical user interface based on the Eclipse platform.

Related tasks:

"Adding a database as a source or target"
Add a database as a source, and database tables as targets, to message maps that support database mappings.

Related reference:

"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.

"Mapping node" on page 4571
Use the Mapping node to construct one or more new messages and populate them with various types of information.

Adding a database as a source or target:

Add a database as a source, and database tables as targets, to message maps that support database mappings.

Before you begin

You must create a database definition for your database before you can add it, or the associated tables, to a message map.

About this task

You can add database sources and targets in a number of different ways:

Procedure

- In an existing message map, click **Select Data Source** to add a database as a source.
  1. In the Spreadsheet pane, select the location to add a database table source to your mapping. For example, select `$target`.
  2. Click **Map** > **Select Data Source**. Alternatively, right-click in the Spreadsheet pane and click **Select Data Source**. The Select Database As Mapping Source wizard opens.
  3. Select your database from the list. If you cannot find the Data Sources or Data Targets that you expect, clear **Apply working set filtering to artifact selection(s) on the page** to view additional options.
  4. When you select a database source during map file creation, the database source is placed in the map script as a sibling (as opposed to a container) of the one or more message targets in the map. When you create a mapping from an item in the database source, the mapper moves the database source in the map script structure to the lowest container of all mappings that use the database source. You can move the database source in the map script by using drag-and-drop or copy-paste actions.
- You can specify the databases and database tables that you want to use in the New Message Map wizard when you create a message map.
  1. Create a message map file by using **File** > **New** > **Message Map**, or by right-clicking your mapping node and selecting **Open Map**.
  2. From **Select map sources**, select the Database Sources for your message map.
  3. From **Select map targets**, select the database tables to use as targets in your message map. If you are not creating a message map from a DataDelete, DataInsert, or DataUpdate node, expand the relevant database operation and select from the list of tables. You can select from the following database operations:
     - **Table Inserts**
     - **Table Updates**
     - **Table Deletes**

     If you cannot find the Data Sources or Data Targets that you expect, clear **Apply working set filtering to artifact selection(s) on the page** to view additional options.
  4. When you select a database source during map file creation, the database source is placed in the map script as a sibling (as opposed to a container) of the one or more message targets in the map. When you create a mapping from an item in the database source, the mapper moves the database source in the map script structure to the lowest container of all mappings that use the database source. You can move the database source in the map script by using drag-and-drop or copy-paste actions.
- In an existing message map, click **Add Sources and Targets** to add a database as a source and database tables as a target.
  1. From the Message Mapping editor, click **Map** > **Add Sources and Targets**. Alternatively, right-click in the Source pane and click **Add Sources**, or right-click in the Target pane and click **Add Targets**.
  2. From **Select map sources**, select the Database Sources for your message map.

3. From Data Targets, in **Select map targets**, select the database tables to use as targets in your message map. If you are not creating a message map from a DataDelete, DataInsert, or DataUpdate node, expand the relevant database operation, and select from the list of tables. You can select from the following database operations:
   - **Table Inserts**
   - **Table Updates**
   - **Table Deletes**

   If you cannot find the Data Sources or Data Targets that you expect, clear **Apply working set filtering to artifact selection(s) on the page** to view additional options.

   You can map to an element within a View, the name of which is annotated with `(read-only view)`.

   You cannot select View in the Add Maps dialog for **Insert**, **Update**, or **Delete**, because View is read only.
4. When you add a database source using the **Add Sources** action, the database source is placed in the map script as a sibling (as opposed to a container) of the one or more message targets in the map. When you create a mapping from an item in the database source, the mapper moves the database source in the map script structure to the lowest container of all mappings that use the database source. You can move the database source in the map script by using drag-and-drop or copy-paste actions.

Results

When you have added the database as a source:

- The Source pane contains a $db:select entry.
- The Spreadsheet pane contains a $db:select entry.

When you have added the database table as a target:

- The Target pane and Spreadsheet pane contain one of the following entries:
  - a $db:insert entry
  - a $db:update entry
  - a $db:delete entry

  You can change the database operation on a selected table by using the **Change Database Operation** dialog.

What to do next

You cannot add a database as a source or a target to an Extract node.

Related concepts:

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

Related tasks:

"Creating a message map file in the Broker Development view" on page 2233

"Mapping a target element from database tables" on page 2288
To map a target element from a database table, set up the Mapping node to retrieve the relevant rows from the database and populate the message target elements with values from database.

"Mapping a target element from database stored procedures" on page 2290
Use a DB2 or Oracle database stored procedure as a source in a Mapping node.

"Mapping a target element from database user-defined functions" on page 2292
Use an Oracle database user-defined function as a source in a Mapping node.

"Change database operation of a message map" on page 2286
If you have created a message map that performs a database operation such as data insert, data update or data delete on a database table you might want to change the database operation that the map performs.

Related reference:

"Message Mapping editor Source pane" on page 4983
Details of the elements present in the Source pane of the Message Mapping Editor.

"Message Mapping editor Target pane" on page 4987
Details of the elements present in the Target pane of the Message Mapping Editor.

"Message Mapping editor Spreadsheet pane" on page 4991
Details of the actions available to you in the Spreadsheet pane of the Message Mapping Editor.

Modifying databases using message mappings:

Create message mappings to read, update, and write to databases.

About this task

Select one or more of the following topics to work with databases by using message mappings:

Procedure

- "Adding database definitions to the WebSphere Message Broker Toolkit" on page 2278
- "Importing large databases to the WebSphere Message Broker Toolkit" on page 2280
- "Creating a message map file from a DataInsert node" on page 2282
- "Creating a message map file from a DataUpdate node" on page 2283
- "Creating a message map file from a DataDelete node" on page 2284
- "Change database operation of a message map" on page 2286
- "Mapping from a message and database" on page 2287
- "Mapping a target element from database tables" on page 2288
- "Mapping a target element from database stored procedures" on page 2290
- "Mapping a target element from database user-defined functions" on page 2292
- "Deleting data from a database with a mapping node" on page 2294
- "Creating a database to database mapping" on page 2295
- "Adding a database as a source or target" on page 2274

Related concepts:

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

Related tasks:

"Creating a message map file in the Broker Development view" on page 2233

"Creating a message mapping file from a Mapping node" on page 2236
You can use a Mapping node to create a message map with messages and

databases as both sources and targets.

"Configuring message mappings" on page 2237
Use the Message Mapping editor to configure a message mapping.

Adding database definitions to the WebSphere Message Broker Toolkit:

Use the New Database Definition File wizard to add database definitions to the WebSphere Message Broker Toolkit.

About this task

You must have a database definition defined in the WebSphere Message Broker Toolkit to create database mappings. You can also use database definitions in other nodes, such as the Compute node, to validate references to database sources and tables. Database definitions are stored in a data design project. You must associate the data design project with all message flow projects that you want to use the database definitions with.

Complete the following steps to add a database definition to the WebSphere Message Broker Toolkit:

Procedure

1. Click **File** > **New** > **Database Definition**. The New Database Definition File wizard is displayed.
2. Select an existing data design project, or click **New** to create a new data design project.
3. Select the database type and version that you want to connect to from the **Database and Version** list. Ensure that you select a database from the list that is supported by WebSphere Message Broker; you can use this wizard in a shell-share environment with other Rational products that support other databases or versions.

   For a list of databases supported by the broker, see "Supported databases" on page 3591.
4. Click **Next**.
5. Either select to create a new database connection or select a connection to use from the list of existing connections. If you select to use an existing connection, the existing database definition is overwritten.
6. Click **Next**.
7. If you selected to create a new connection:
   a. Optional: You can enter a custom value for the Connection Name if you clear **Use default naming convention**.
   b. Enter values for the Connection to the database, for example, Database name, Host name and Port number.
   c. Enter values for the User ID and Password to connect to the database. Click **Test Connection** to verify the settings you have selected for your database. The default Port number for a DB2 database is 50000. If the connection fails, enter other values such as 50001, 50002 and so on, for the Port number, and test the connection again.
   d. Click **Next**. An error is generated if any of the connection details are wrong. If you specify a Database that already has a database definition in the data design project, click **Yes** in the Confirm file overwrite window to overwrite the existing database definition.
8. Alternatively, if you selected to use an existing connection:

a. Click **Yes** in the Confirm file overwrite window to overwrite the existing database definition.
b. Enter values for the User ID and Password to connect to the database, and click **Next**.

9. Select one or more database schemas from the list and click **Next**.
10. Select the elements that you require on the Database Elements page. You can select any option, in addition to **Tables**, on the Database Elements page.
    a. Select **Views** to see all the database views in the Data Project Explorer
    b. Select **Routines** to add stored procedures and user defined functions to the database definition file.

    If you select other additional options, the database definition files that you create contain more information than the Compute, Database, or Mapping nodes require.
11. Click **Finish**.
12. Add the data design project as a reference to the message flow project:
    a. Right-click the message flow project, and click **Properties**.
    b. Click **Project References**, and select the data design project from the list to add as a referenced project.
    c. Click **OK**.

Results

A new database definition file is added to your data design project. The database definition file name has the following format:`<database>.dbm`. Database definition files are associated with the Data Project Explorer view and the Data Source Explorer view. Tools are available in these views for working with your databases.

What to do next

Database definition files in the WebSphere Message Broker Toolkit are not automatically updated. If you make a change to your database, you must re-create the database definition files.

Related concepts:

"Client application programming interfaces" on page 1038
You can configure the nodes in your message flows to customize the behavior of those nodes by using one or more of the supported programming interfaces.

"Project references" on page 44
When a projects refers to other projects in the workspace, this is called a *project reference*. When one project references another, the files in the referenced project are available for use by the referring project. For example to configure certain nodes, to create libraries of reusable message flows, and to enable Content Assist in the ESQL editor.

Related tasks:

"Modifying databases using message mappings" on page 2277
Create message mappings to read, update, and write to databases.

"Creating a message map file from a DataInsert node" on page 2282

"Creating a message map file from a DataUpdate node" on page 2283

"Creating a message map file from a DataDelete node" on page 2284

"Adding a database as a source or target" on page 2274
Add a database as a source, and database tables as targets, to message maps that support database mappings.

Related reference:

"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.

"Supported databases" on page 3591
You can optionally configure databases to contain data that is accessed by your message flows. Databases from IBM and other suppliers are supported at specific versions on supported operating systems.

"Broker Application Development perspective" on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

Importing large databases to the WebSphere Message Broker Toolkit:

Use the Data Source Explorer view of the WebSphere Message Broker Toolkit to select which options you require when you import large databases.

About this task

You must have a database definition defined in the WebSphere Message Broker Toolkit to create database mappings. The following steps tell you how to import specific element definitions into your database definition.

Complete the following steps to import specific database definitions to the WebSphere Message Broker Toolkit:

Procedure

1. Open the Data perspective of the WebSphere Message Broker Toolkit.
2. In the Data Source Explorer view select **Connections**, and right-click to start the New Connection wizard.
3. To create a connection:
   a. Optional: You can enter a custom value for the Connection Name if you clear **Use default naming convention**.
   b. Enter values for the Connection to the database, for example, Database name, Host name, and Port number.
   c. Enter values for the user ID and password to connect to the database. Click **Test Connection** to verify that the settings you have selected for your database. The default Port number for a DB2 database is 50000. If the connection fails, enter other values such as 50001, 50002, and so on, for the Port number, and test the connection again.
   d. Click **Next**. An error is generated if any of the connection details are wrong.
4. On the Filter panel, enable the filter and select the schemas that you require.
5. Click **Finish** to complete the creation of a database connection.
6. In the Data Source Explorer view, expand the folders of the new Database Connection to locate the database elements for which you need definitions.
7. Select the folder that contains the elements you require, right-click, and select **Filter**.
8. On the Filter panel, enable filtering and select the elements that you require.
9. Click **Finish**. The Data Source Explorer view now shows only the elements that you selected.
10. Switch to the Broker Application Development perspective.

11. Click **File** > **New** > **Database Definition**. The New Database Definition File wizard is displayed.
12. Select an existing data design project, or click **New** to create a data design project.
13. Select the database type and version that you want to connect to from the **Database and Version** list. Ensure that you select a database from the list that is supported by WebSphere Message Broker; you can use this wizard in a shell-share environment with other Rational products that support other databases or versions.

    For a list of databases supported by the broker, see "Supported databases" on page 3591.
14. Click **Next**.
15. On the Select Connection panel, select **Use an existing connection**. This connection is the one that you created in steps 1 on page 2280 to 9 on page 2280 earlier.
16. Click **Next**.
17. Enter values for the user ID and password to connect to the database, and click **Next**.
18. Select one or more database schemas from the list and click **Next**.
19. Select the elements that you require on the Database Elements page. You can select any option, in addition to **Tables**, on the Database Elements page.
    a. Select **Views** to see all the database views in the Data Project Explorer
    b. Select **Routines** to add stored procedures and user-defined functions to the database definition file.
20. Click **Finish**.
21. Add the data design project as a reference to the message flow project:
    a. Right-click the message flow project, and click **Properties**.
    b. Click **Project References**, and select the data design project you created, from the list, to add as a referenced project.
    c. Click **OK**.
22. Carry out the following procedure, to ensure that only the elements you selected appear in a map:
    a. Drag a Mapping node node onto a flow editor.
    b. Double-click the Mapping node node to bring up the map creation dialog.
    c. Follow the prompts to create the map, selecting the data source that references your new database definition as either a source or target.

What to do next

Database definition files in the WebSphere Message Broker Toolkit are not automatically updated. If you change your database, you must re-create the database definition files.

Related concepts:

"Client application programming interfaces" on page 1038
You can configure the nodes in your message flows to customize the behavior of those nodes by using one or more of the supported programming interfaces.

"Project references" on page 44
When a projects refers to other projects in the workspace, this is called a *project reference*. When one project references another, the files in the referenced project are available for use by the referring project. For example to configure certain nodes, to create libraries of reusable message flows, and to enable Content Assist in the ESQL editor.

Related tasks:

"Modifying databases using message mappings" on page 2277
Create message mappings to read, update, and write to databases.

"Creating a message map file from a DataInsert node"

"Creating a message map file from a DataUpdate node" on page 2283

"Creating a message map file from a DataDelete node" on page 2284

"Adding a database as a source or target" on page 2274
Add a database as a source, and database tables as targets, to message maps that support database mappings.

Related reference:

"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.

"Supported databases" on page 3591
You can optionally configure databases to contain data that is accessed by your message flows. Databases from IBM and other suppliers are supported at specific versions on supported operating systems.

"Broker Application Development perspective" on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

Creating a message map file from a DataInsert node:

You can use a DataInsert node to create mappings to insert new data into a database from a message, another database or both.

Before you begin

Before creating a message map file, ensure you do the following:

1. Create a message flow project
2. Create a message flow
3. Define message flow content that includes a DataInsert node
4. Create a database definition

About this task

To create a message map (.msgmap) file from a DataInsert node:

Procedure

1. From the Broker Application Development perspective, open your message flow, right-click your DataInsert node, and click **Open Map**. The New Message Map for Data Insert Node wizard opens.
2. Select the combination of Messages, Data Sources or both that you want to use as sources for your map from Select map sources.

   If you cannot find the Messages or Data Sources that you expect, clear the **Apply working set filtering to artifact selection(s) on the page** check box.

3. From the Select map targets pane, select the tables under Table Inserts into which you want to insert new data. The tables that you select are added to the new message map as targets.
4. Select **OK** to create the new message map. The "Message Mapping editor" on page 4981 opens with the selected sources and targets.

What to do next

After you have created a message map file, you can now configure the message mappings.

Related concepts:

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

Related tasks:

"Editing a default-generated map manually" on page 2312
A map that is generated by the Message Mapping editor might not do everything that you want; you can change maps manually to enhance its operation.

"Creating and calling submaps and subroutines" on page 2298
Use submaps, ESQL subroutines, or both, to map source elements to target elements.

"Configuring message mappings" on page 2237
Use the Message Mapping editor to configure a message mapping.

Related reference:

"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.

"Mapping node" on page 4571
Use the Mapping node to construct one or more new messages and populate them with various types of information.

"DataInsert node" on page 4386
Use the DataInsert node to interact with a database in the specified ODBC data source.

Creating a message map file from a DataUpdate node:

You can use a DataUpdate node to create mappings to update existing data in a database from a message, another database or both.

Before you begin

Before creating a message map file, ensure you do the following:

1. Create a message flow project
2. Create a message flow
3. Define message flow content that includes a DataUpdate node
4. Create a database definition

About this task

To create a message map (.msgmap) file from a DataUpdate node:

Procedure

1. From the Broker Application Development perspective, open your message flow, right-click your DataUpdate node, and click **Open Map**. The New Message Map for Data Update Node wizard opens.
2. Select the combination of Messages, Data Sources or both that you want to use as sources for your map from Select map sources.

   If you cannot find the Messages or Data Sources that you expect, select the **Show all resources in workspace** check box.
3. From the Select map targets pane, select the tables under Table Updates in which you want to update data. The tables that you select are added to the new message map as targets.
4. Select **OK** to create the new message map. The "Message Mapping editor" on page 4981 opens with the selected sources and targets.

What to do next

After you have created a message map file, you can now configure the message mappings.

Related concepts:

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

Related tasks:

"Editing a default-generated map manually" on page 2312
A map that is generated by the Message Mapping editor might not do everything that you want; you can change maps manually to enhance its operation.

"Creating and calling submaps and subroutines" on page 2298
Use submaps, ESQL subroutines, or both, to map source elements to target elements.

"Configuring message mappings" on page 2237
Use the Message Mapping editor to configure a message mapping.

Related reference:

"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.

"Mapping node" on page 4571
Use the Mapping node to construct one or more new messages and populate them with various types of information.

"DataUpdate node" on page 4390
Use the DataUpdate node to interact with a database in the specified ODBC data source.

Creating a message map file from a DataDelete node:

You can use a DataDelete node to create mappings to delete data from a database based on information from an input message, another database or both.

Before you begin

Before creating a message map file, ensure you do the following:

1. Create a message flow project
2. Create a message flow
3. Define message flow content that includes a DataDelete node

4. Create a database definition

About this task

To create a message map (.msgmap) file from a DataDelete node:

Procedure

1. From the Broker Application Development perspective, open your message flow, right-click your DataDelete node, and click **Open Map**. The New Message Map for Data Delete Node wizard opens.
2. Select the combination of Messages, Data Sources or both that you want to use as sources for your map from Select map sources.

   If you cannot find the Messages or Data Sources that you expect, clear the **Apply working set filtering to artifact selection(s) on the page** check box.
3. From the Select map targets pane, select the tables under Table Deletes from which you want to delete data. The tables that you select are added to the new message map as targets.
4. Select **OK** to create the new message map. The "Message Mapping editor" on page 4981 opens with the selected sources and targets.

What to do next

After you have created a message map file, you can now configure the message mappings.

Related concepts:

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

Related tasks:

"Editing a default-generated map manually" on page 2312
A map that is generated by the Message Mapping editor might not do everything that you want; you can change maps manually to enhance its operation.

"Deleting data from a database with a mapping node" on page 2294
You can use a DataDelete or Mapping node to delete data from a database, based on information from an input message, another database or both.

"Creating and calling submaps and subroutines" on page 2298
Use submaps, ESQL subroutines, or both, to map source elements to target elements.

"Configuring message mappings" on page 2237
Use the Message Mapping editor to configure a message mapping.

Related reference:

"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.

"Mapping node" on page 4571
Use the Mapping node to construct one or more new messages and populate them with various types of information.

"DataDelete node" on page 4382
Use the DataDelete node to interact with a database in the specified ODBC data source.

Change database operation of a message map:

If you have created a message map that performs a database operation such as data insert, data update or data delete on a database table you might want to change the database operation that the map performs.

About this task

You might also have created a database mapping by dragging a table from the Broker Development view onto a message map and want to change the default insert operation to another database operation.

To change the database operation of a database table in your message map:

Procedure

1. From the Broker Application Development perspective, open your message map.
2. Right-click on the target database table in the target pane and click **Change Database Operation**. The Select Database Operation dialog is displayed.
3. Select the database operation you want to perform on the selected table:
   - Insert
   - Update
   - Delete
4. Click **OK** to change the database operation on the selected table.

What to do next

If you change the database operation of your message map to or from data delete you must re-create any mappings to your target database columns.

Related concepts:

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

Related tasks:

"Editing a default-generated map manually" on page 2312
A map that is generated by the Message Mapping editor might not do everything that you want; you can change maps manually to enhance its operation.

"Creating and calling submaps and subroutines" on page 2298
Use submaps, ESQL subroutines, or both, to map source elements to target elements.

"Configuring message mappings" on page 2237
Use the Message Mapping editor to configure a message mapping.

Related reference:

"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.

"Mapping node" on page 4571
Use the Mapping node to construct one or more new messages and populate them with various types of information.

"DataDelete node" on page 4382
Use the DataDelete node to interact with a database in the specified ODBC data source.

Mapping from a message and database:

You can create a message map that uses both a message and a database as a source.

Before you begin

Before creating a message map file, ensure you complete the following steps:

1. Create a message flow project
2. Create a message flow
3. Define message flow content
4. Create database definitions

About this task

The following instructions describe how to specify a message and a database as the data source.

Procedure

1. Right-click a node that supports mapping, such as the Mapping node, and click **Open Map**.
2. Follow the on-screen instructions to complete the New Message Map wizard:
   a. Select the combination of Messages and Data Sources that you want to use as sources for your message map from Select map sources.
   b. Select the combination of Messages, Data Targets or both that you want to use as targets for your map from Select map targets.
3. Perform mapping as usual from the source message.
4. Follow the guidance in "Mapping a target element from database tables" on page 2288 to create the mappings from a source database to the target message or database table.
5. Follow the guidance in "Mapping a target element from database stored procedures" on page 2290 to create the mappings from a database stored procedure to the target message or database table.
6. Follow the guidance in "Mapping a target element from database user-defined functions" on page 2292 to create the mappings from a database user-defined function to the target message or database table.

Related tasks:

"Creating a message map file in the Broker Development view" on page 2233

"Configuring message mappings" on page 2237
Use the Message Mapping editor to configure a message mapping.

"Populating a message map" on page 2269
Use the Insert Children wizard to add elements from the Target pane to the Spreadsheet pane. The Insert Children wizard creates child structures for the selected parent structure.

"Mapping a target element from source message elements" on page 2254

"Mapping a target element from database tables" on page 2288
To map a target element from a database table, set up the Mapping node to retrieve the relevant rows from the database and populate the message target elements with values from database.

"Mapping a target element from database stored procedures" on page 2290
Use a DB2 or Oracle database stored procedure as a source in a Mapping node.

"Mapping a target element from database user-defined functions" on page 2292
Use an Oracle database user-defined function as a source in a Mapping node.
"Creating a database to database mapping" on page 2295
You can create a message map that uses a database as both the source and target. The contents of the source database can be used to interact with the same or a different database table. The message map can also include a message as a source, but a message is not required. You can, for example, use a timer node to schedule regular updates to a database.

Mapping a target element from database tables:

To map a target element from a database table, set up the Mapping node to retrieve the relevant rows from the database and populate the message target elements with values from database.

About this task

You can add a database as a source for a mapping in several ways, as described in "Adding a database as a source or target" on page 2274. After you have added a database to the mapping, the Spreadsheet pane contains a $db:select entry in the Map Script column. By default, its value is fn:true(), which means that all rows are retrieved from the database table. In database SQL, you can restrict the number of rows by adding a WHERE clause to a database call. In the Mapping node, the equivalent method of restricting the number of selected rows is to use a $db:select expression.

Complete the following steps to restrict the number of rows that are selected in a Mapping node:

Procedure

1. In the Spreadsheet pane, click the $db:select row. fn:true() is entered in the Edit pane.
2. Edit the expression in the Edit pane to specify the correct condition for the database call. To help you achieve this condition, you can:
   a. Select all database columns that are relevant to the rows that are retrieved, and drag them from the Source pane to the Edit pane. These database column names are used in an SQL WHERE clause.
   b. Select all source message elements with values that are relevant to the rows that are retrieved, and drag them from the Source pane into the Edit pane. These values can be matched against the selected database columns.
   c. Click **Edit** > **Content Assist** to open Content Assist.
   d. From Content Assist, select the functions to apply to message elements in the database call.

Results

The following example of a $db:select entry shows a database column that is matched against a constant or a field from an input message:

```
$db:select_1.BROKER50.JDOE.RESOLVEASSESSOR.ASSESSORTYPE = 'WBI' or $db:select_1.BROKER50.JDOE.
RESOLVEASSESSOR.ASSESSORTYPE = $source/tns:msg_tagIA81CONF/AssessorType
```

A $db:select entry retrieves all qualifying rows, therefore more than one row might be retrieved. By default, the selection is treated as repeating, which is indicated by the 'for' row below $db:select in the Spreadsheet pane.

After you have configured the $db:select, populate the target message from the database by dragging the database column from the Source Pane to the message element in the Target pane. The mapping is indicated by a line between the database column in the Source pane and the element in the Target pane. An entry for this map in XPath format also appears in the Spreadsheet pane. Triangular icons are displayed in the Source and Target panes next to objects that have been mapped.

You can map to an element within a View, the name of which is annotated with `(read-only view)`.

What to do next

Using database selects

By default, a $db:select entry is accompanied by a 'for' row that iterates over the select result set. Ensure that your 'for' row is in the correct position for your mapping. The behavior of the map is determined by the position of the 'for' row in the Spreadsheet pane. For example, if the results of the $db:select statement matched five rows in the database, and the 'for' row is the parent of the $target entry in the Spreadsheet pane, five complete messages are generated by the mapping node. If the 'for' row is positioned within the message body, one message is generated with five repeating elements in the message body.

A mapping can contain multiple 'for' rows, associated with a $db:select entry, that perform a single database select and iterate over the results multiple times. For example, multiple 'for' rows can be used in conditional mappings, where an individual 'for' row is used with a 'condition' or an 'else'.

A 'for' row is not always required and can be deleted in the following circumstances:

- If the database select returns only one row.
- If you use an aggregate XPath function on the select results.

  For example: fn:sum or fn:count.

All $db:select expressions must be within the scope of the $db:select entry in the Spreadsheet pane, meaning that each one must be a descendant of the select statement. If a $db:select expression is out of scope, the Message Mapping editor moves the $db:select entry to a position where the $db:select expression is in scope. Ensure that the position of the $db:select entry is correct for your message mapping.

Database table join

Database table join is supported for tables within the same database. For example, consider the following two tables where PRODUCT_ID and PART_NUMBER match.

```
Table    Column        Row 1    Row 2    Row 3      Row 4
ORDER    PRODUCT_ID    456      456      345        123
         QUANTITY      100      200      300        400

PRODUCT  PART_NUMBER   123      456      789        012
         PART_NAME     pen      pencil   paperclip  glue
         PRICE         0.25     0.15     0.02       0.99
```

A $db:select expression with the following syntax joins the tables:

```
$db:select.MY_DB.SCHEMA1.ORDER.PRODUCT_ID=$db:select.MY_DB.SCHEMA2.PRODUCRT.PART_NUMBER
```

The $db:select expression in the example generates the following result set.

```
                Row 1    Row 2    Row 3
PRODUCT_ID      456      456      123
QUANTITY        100      200      400
PART_NUMBER     456      456      123
PART_NAME       pencil   pencil   pen
PRICE           0.15     0.15     0.25
```

You can then use the 'for' row to iterate through the results set in the same way as results from a single table.

Related concepts:

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

"Creating message mappings" on page 2232
You can create message mappings either in the Broker Development view of the Toolkit, or from a Mapping node.

Related tasks:

"Mapping a target element from database stored procedures"
Use a DB2 or Oracle database stored procedure as a source in a Mapping node.

"Mapping a target element from database user-defined functions" on page 2292
Use an Oracle database user-defined function as a source in a Mapping node.

Related reference:

"Mapping node" on page 4571
Use the Mapping node to construct one or more new messages and populate them with various types of information.

"Message Mapping editor Source pane" on page 4983
Details of the elements present in the Source pane of the Message Mapping Editor.

"Message Mapping editor Target pane" on page 4987
Details of the elements present in the Target pane of the Message Mapping Editor.

"Message Mapping editor Edit pane" on page 4990
Details of how you use the Edit pane of the Message Mapping Editor.

"Message Mapping editor Spreadsheet pane" on page 4991
Details of the actions available to you in the Spreadsheet pane of the Message Mapping Editor.

"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.

Mapping a target element from database stored procedures:

Use a DB2 or Oracle database stored procedure as a source in a Mapping node.

Before you begin

Before you start:

You must include the database stored procedure in your database definition. See "Adding database definitions to the WebSphere Message Broker Toolkit" on page 2278. Only DB2 and Oracle databases are supported for this task.

About this task

To map a target element from a database stored procedure, set up the Mapping node to:

- run the stored procedure to retrieve the relevant rows from the database
- populate the message target elements with values from the database

Complete these steps to map a target element from a stored procedure:

Procedure

1. Add a stored procedure as a source for the mapping, as described in "Adding a database as a source or target" on page 2274. After you have added a stored procedure to the mapping, the Source pane and the Spreadsheet pane contain $db:proc entries. These $db:proc entries contain details of the parameters used by the stored procedure. For each parameter it shows its name, its mode (either IN, INOUT, or OUT), and its data type.

   The $db:proc entry in the Spreadsheet pane also contains a 'for' row. By default, the position of the 'for' row in the $db:proc entry indicates that when there is more than one record in the result set, the output message body contains repeating elements. If you want a separate message to be output for each record in a result set, move the $target element in the Spreadsheet pane below the 'for' row.

   If you are using a DB2 stored procedure which specifies a maximum number of result sets, the Source pane also includes a list of result sets. If you are using an Oracle stored procedure which specifies ref cursor parameters of mode INOUT or OUT, the Mapping node treats them as named result sets rather than parameters. The Mapping node does not support ref cursor parameters of mode IN.
2. If your stored procedure has parameters of mode IN or INOUT provide values for each of them:
   a. Type an expression in the **Value** field next to the parameter in the Spreadsheet pane.
   b. Select the parameter in the Spreadsheet pane, drag an element from the source message to the Edit pane, and press Enter.
3. If your stored procedure sets a return value, right-click on the $db:proc entry in the Source pane and select **Toggle Add/Remove Stored Procedure Return Value**. An element called ReturnValue of type INTEGER is added to the Source pane.

   DB2 on distributed systems returns SQLCODE if the stored procedure does not set a return value.

   DB2 on z/OS and Oracle stored procedures do not set a return value.
4. For each result set that your stored procedure returns, define the columns it contains:
   a. Right-click on the result set entry in the Source pane and select **Add or Remove Result Set Columns**. The Add or Remove Result Set Columns window opens.
   b. (Optional) Select an item from the **Restore previous settings from** list. The **Result set columns** list is replaced.
   c. (Optional) Select columns in the **Available database table columns** list, and add them to the **Result set columns** list.

d. (Optional) Type a column name in the **New column** field, and add it to the **Result set columns** list. Use this method to add calculated columns that are named in the stored procedure.

e. (Optional) Change the name in the **Save current settings to** field that is used to save the columns shown in the **Result set columns** when you click **Finish**. The settings are saved in the `\.metadata` directory. You can retrieve these settings in another message map, as described in step 4a. If you do not change the default name the settings are automatically retrieved when the stored procedure is used in another message map.

Related concepts:

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

"Creating message mappings" on page 2232
You can create message mappings either in the Broker Development view of the Toolkit, or from a Mapping node.

Related tasks:

"Mapping a target element from database tables" on page 2288
To map a target element from a database table, set up the Mapping node to retrieve the relevant rows from the database and populate the message target elements with values from database.

"Mapping a target element from database user-defined functions"
Use an Oracle database user-defined function as a source in a Mapping node.

Related reference:

"Mapping node" on page 4571
Use the Mapping node to construct one or more new messages and populate them with various types of information.

"Message Mapping editor Source pane" on page 4983
Details of the elements present in the Source pane of the Message Mapping Editor.

"Message Mapping editor Target pane" on page 4987
Details of the elements present in the Target pane of the Message Mapping Editor.

"Message Mapping editor Edit pane" on page 4990
Details of how you use the Edit pane of the Message Mapping Editor.

"Message Mapping editor Spreadsheet pane" on page 4991
Details of the actions available to you in the Spreadsheet pane of the Message Mapping Editor.

"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.

Mapping a target element from database user-defined functions:

Use an Oracle database user-defined function as a source in a Mapping node.

Before you begin

Before you start:

You must include the database user-defined function in your database definition. See "Adding database definitions to the WebSphere Message Broker Toolkit" on page 2278. Only Oracle databases are supported for this task.

About this task

To map a target element from a database user-defined function, set up the Mapping node to:

- run the user-defined function to retrieve the relevant rows from the database
- populate the message target elements with values from the database

Complete these steps to map a target element from a user-defined function:

Procedure

1. Add a user-defined function as a source for the mapping, as described in "Adding a database as a source or target" on page 2274. After you have added a user-defined function to the mapping, the Source pane and the Spreadsheet pane contain $db:func entries. These $db:func entries contain details of the parameters used by the user-defined function. For each parameter it shows its name, its mode (either IN, INOUT, or OUT), and its data type.

   The $db:func entry in the Spreadsheet pane also contains a 'for' row. By default, the position of the 'for' row in the $db:func entry indicates that when there is more than one record in the result set, the output message body contains repeating elements. If you want a separate message to be written for each record in a result set, move the $target element in the Spreadsheet pane below the 'for' row.

   If you are using an Oracle user-defined function which specifies ref cursor parameters of mode INOUT or OUT, the Mapping node treats them as named result sets rather than parameters. The Mapping node does not support ref cursor parameters of mode IN.

   If you are using an Oracle user-defined function which returns a value of ref cursor data type, the Mapping node treats it as a named result set.
2. If your user-defined function has parameters of mode IN or INOUT provide values for each of them:
   a. Type an expression in the **Value** field next to the parameter in the Spreadsheet pane.
   b. Select the parameter in the Spreadsheet pane, drag an element from the source message to the Edit pane, and press Enter.
3. For each result set that your user-defined function returns, define the columns it contains:
   a. Right-click on the result set entry in the Source pane and select **Add or Remove Result Set Columns**. The Add or Remove Result Set Columns window opens.
   b. (Optional) Select an item from the **Restore previous settings from** list. The **Result set columns** list is replaced.
   c. (Optional) Select columns in the **Available database table columns** list, and add them to the **Result set columns** list.
   d. (Optional) Type a column name in the **New column** field, and add it to the **Result set columns** list. Use this method to add calculated columns that are named in the user-defined function.
   e. (Optional) Change the name in the **Save current settings to** field that is used to save the columns shown in the **Result set columns** when you click **Finish**. The settings are saved in the`\.metadata` directory. You can retrieve these settings in another message map, as described in step 3a. If you do not change the default name the settings are automatically retrieved when the user-defined function is used in another message map.

Related concepts:

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

"Creating message mappings" on page 2232
You can create message mappings either in the Broker Development view of the Toolkit, or from a Mapping node.

Related tasks:

"Mapping a target element from database tables" on page 2288
To map a target element from a database table, set up the Mapping node to retrieve the relevant rows from the database and populate the message target elements with values from database.

"Mapping a target element from database stored procedures" on page 2290
Use a DB2 or Oracle database stored procedure as a source in a Mapping node.

Related reference:

"Mapping node" on page 4571
Use the Mapping node to construct one or more new messages and populate them with various types of information.

"Message Mapping editor Source pane" on page 4983
Details of the elements present in the Source pane of the Message Mapping Editor.

"Message Mapping editor Target pane" on page 4987
Details of the elements present in the Target pane of the Message Mapping Editor.

"Message Mapping editor Edit pane" on page 4990
Details of how you use the Edit pane of the Message Mapping Editor.

"Message Mapping editor Spreadsheet pane" on page 4991
Details of the actions available to you in the Spreadsheet pane of the Message Mapping Editor.

"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.

Deleting data from a database with a mapping node:

You can use a DataDelete or Mapping node to delete data from a database, based on information from an input message, another database or both.

Before you begin

You must do the following before you can delete data from a database using a mapping node:

1. Create a message flow project
2. Create a message flow
3. Define message flow content that includes a DataDelete or a Mapping node
4. Create a message map file from a DataDelete node or Create a message map file from a Mapping node

About this task

You cannot create mappings to delete data from a database by dragging from the source to the target. Instead, you select rows to delete based on the content of the source. You can use an expression to match the content of the source to the target field, for example, use the following instructions to delete all rows in the database that match the content of a field from the input message:

Procedure

1. Right-click your DataDelete or Mapping node, and click **Open Map**. The "Message Mapping editor" on page 4981 opens with your selected sources and targets.
2. Select `$db:delete` in the Spreadsheet pane.
3. Drag the appropriate source element from the message in the Source pane to the Edit pane. For example, `$source/shipTo/accNum`.
4. Drag the appropriate target database field from the Target pane to the Edit pane. For example, `$db:delete.SAMPLE.MYSCHEMA.CUSTOMER.CONTACT_ID`.
5. Change the expression in the Edit pane to set the target field to be equal to the source element. For example, `$source/shipTo/accNum = $db:delete.SAMPLE.MYSCHEMA.CUSTOMER.CONTACT_ID`.

What to do next

You can use conditional mappings such as If statements to create more complex mappings that define which data to delete from a database. You can also use conditional statements in a Mapping node to perform different database operations depending on the content of the input message. For example, you can add a Table Inserts target, a Table Updates target and a Table Deletes target to a message map, then use conditional statements to define which of the operations to perform.

Related concepts:

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

Related tasks:

"Configuring message mappings" on page 2237
Use the Message Mapping editor to configure a message mapping.

"Configuring conditional mappings" on page 2265
How to set the value of a target element conditionally in a Mapping node.

Related reference:

"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.

"Mapping node" on page 4571
Use the Mapping node to construct one or more new messages and populate them with various types of information.

"DataDelete node" on page 4382
Use the DataDelete node to interact with a database in the specified ODBC data source.

Creating a database to database mapping:

You can create a message map that uses a database as both the source and target. The contents of the source database can be used to interact with the same or a different database table. The message map can also include a message as a source, but a message is not required. You can, for example, use a timer node to schedule regular updates to a database.

Before you begin

Before creating a message map file with a database to database mapping, ensure you do the following:

1. Create a message flow project

2. Create a message flow
3. Define message flow content
4. Create database definitions

About this task

To create a database to database mapping:

Procedure

1. Right-click a node that supports database mapping in your flow, such as the Mapping node, and click **Open Map**. The New Message Map wizard opens for your node.
2. Select the Data Sources and any Messages that you want to use as sources for your map from **Select map sources**.

   If you cannot find the Messages or Data Sources that you expect, clear the **Apply working set filtering to artifact selection(s) on the page** check box.
3. From Select map targets expand the database operation that you want to perform. You can select from the following database operations:
   - Table Inserts
   - Table Updates
   - Table Deletes
4. Select the database tables that you want to map.

   You can create a message map that performs a combination of database inserts, updates or deletes by selecting database tables from different database operations. For example, if you want to create a conditional mapping that updates data in a database if it already exists, but inserts the data if it does not already exist in the database, you can select the same database table under Table Inserts and Table Updates.
5. Select **OK** to create the new message map. The "Message Mapping editor" on page 4981 opens with the selected sources and targets.

What to do next

After you have created a message map file, you can now configure the message mappings.

Related concepts:

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

Related tasks:

"Creating a message map file in the Broker Development view" on page 2233

"Change database operation of a message map" on page 2286
If you have created a message map that performs a database operation such as data insert, data update or data delete on a database table you might want to change the database operation that the map performs.

"Configuring message mappings" on page 2237
Use the Message Mapping editor to configure a message mapping.

Related reference:

"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.

"Mapping node" on page 4571
Use the Mapping node to construct one or more new messages and populate them

with various types of information.

Storing a BLOB message in a database table using a message map:

Use the Message Mapping editor to create a bit stream from a BLOB message, and store it in a database table.

Before you begin

Before you start:

Create a mapping; see "Creating a message mapping file from a Mapping node" on page 2236.

About this task

Take the following steps:

Procedure

1. In the Target pane, right-click the column that will store the bitstream, and select **Enter Expression** from the menu.
2. In the Edit pane, type `esql:asbitstream()`. You can use content assist; `asbitstream` is a *Field function*.
3. Drag the source field, for example `$source/po:purchaseOrder`, to the Edit pane, placing it between the parentheses. The entry in the Edit pane looks like this:

   ```
   esql:asbitstream($source/po:purchaseOrder, 'purchaseOrder',
   'PurchaseOrder', 'XML1', 0, 0, $esql:FolderBitStream)
   ```

   Alternatively, you can use content assist to select the `esql:asbitstream` function. In the Edit pane press Ctrl+Space to display a list of available functions and associated parameters. The `asbitstream` function is an ESQL Field function. The function can take other parameters; see "ESQL mapping functions" on page 4998.

   When you move the cursor out of the Edit pane, or press Enter, the mapping is displayed between the fields in the Source and Target panes.

Related tasks:

"Creating a message mapping file from a Mapping node" on page 2236
You can use a Mapping node to create a message map with messages and databases as both sources and targets.

"Creating a BLOB output message using a message map" on page 2262
Use the Message Mapping editor to create a bit stream from a message source, and create it as a BLOB output message.

"Mapping from a BLOB message to an output message" on page 2263
Use the Message Mapping editor to parse a BLOB message.

"Mapping from a BLOB field in a database table to an output message" on page 2298
Use the Message Mapping editor to parse a bit stream from a field in a database table into a folder in a target message.

Related reference:

"ESQL mapping functions" on page 4998
Some predefined ESQL functions are available for use with message maps.

Mapping from a BLOB field in a database table to an output message:

Use the Message Mapping editor to parse a bit stream from a field in a database table into a folder in a target message.

Before you begin

Before you start:

Create a mapping; see "Creating a message mapping file from a Mapping node" on page 2236.

About this task

Take the following steps:

Procedure

1. Right-click the element in the target pane, and select **Enter Expression** from the menu.
2. In the Edit pane, type `msgmap:element-from-bitstream()`.
3. Drag the field from the database table to the Edit pane, placing it between the parentheses, for example:

   ```
   msgmap:element-from-bitstream($db:select.RESERVDB.USER.XMLFLIGHTTB.FLIGHTDATE)
   ```

   Alternatively, you can use content assist to select the `msgmap:element-from-bitstream` function. In the Edit pane press Ctrl+Space to display a list of available functions and associated parameters. The function can take other parameters; see "Predefined mapping functions" on page 5007. For example:

   ```
   msgmap:element-from-bitstream($db:select.RESERVDB.USER.XMLFLIGHTTB.FLIGHTDATE,
   'ReserveMessageSet', 'FlightMessage', 'XML1', 0, 0, $esql:FolderBitStream)
   ```

   When you move the cursor out of the Edit pane, or press Enter, the mapping is displayed between the fields in the Source and Target panes.

Related tasks:

"Creating a message mapping file from a Mapping node" on page 2236
You can use a Mapping node to create a message map with messages and databases as both sources and targets.

"Storing a BLOB message in a database table using a message map" on page 2297
Use the Message Mapping editor to create a bit stream from a BLOB message, and store it in a database table.

"Creating a BLOB output message using a message map" on page 2262
Use the Message Mapping editor to create a bit stream from a message source, and create it as a BLOB output message.

"Mapping from a BLOB message to an output message" on page 2263
Use the Message Mapping editor to parse a BLOB message.

Related reference:

"Predefined mapping functions" on page 5007
Some predefined mapping functions are provided for use with message maps.

Creating and calling submaps and subroutines:

Use submaps, ESQL subroutines, or both, to map source elements to target elements.

About this task

The following topics describe how to work with submaps and ESQL subroutines:

Procedure

- "Creating a new submap"
- "Creating a new submap for a wildcard source" on page 2300
- "Creating a submap to modify a database" on page 2301
- "Converting a message map to a submap" on page 2302
- "Converting an inline mapping to a submap" on page 2304
- "Calling a submap" on page 2305
- "Calling a map from ESQL" on page 2306
- "Calling an ESQL routine" on page 2308
- "Creating and calling your own user-defined ESQL routine" on page 2308

Related concepts:

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

Creating a new submap:
About this task

This topic describes how to create a new submap. There are three ways to create a new submap:

Procedure

- **Using File > New > Message Map**
  1. From the Broker Application Development perspective, click **File** > **New** > **Message Map**. The New Message Map wizard opens.
  2. Specify the project name and the name for the new submap.
  3. Specify that the new map is a submap by selecting the option: **Submap called by another map**.
  4. Select the combination of Message Components or Data Sources that you want to use as sources for your map from **Select map sources** and select the combination of Message Components or Data Targets that you want to use as targets for your map from **Select map targets**.

     If you cannot find the Message Components, Data Sources or Data Targets that you expect, clear the **Apply working set filtering to artifact selection(s) on the page** check box.
  5. Click **Finish**.

  The new submap opens in the Message Mapping editor.
- **Using Create new submap**
  1. From the Broker Application Development perspective, open the message map for the required node.
  2. In the Source pane, expand the tree and select the source.
  3. In the Target pane, expand the tree and select the target.
  4. Right-click either the source or target, then click **Create New Submap**.

The new submap opens in the Message Mapping editor. If the original map file was called `simple_mapping.msgmap`, the new submap is called `simple_mapping_submap0.msgmap`.

- **Using Convert to submap**
  1. From the Broker Application Development perspective, open the message map.
  2. Select one of the following types of submap to create a new submap from an inline mapping:
     - Element statement that maps a global element or an element of global type
     - Attribute statement that maps either a global attribute or an attribute of a global type
     - Database insert statement
     - Database update statement
     - Database delete statement
  3. Right-click the mapping statement that you want to convert to a submap or database submap in the Script pane, and click **Convert to submap**. A new submap is created and a statement is added to the original message map to call the new submap.

  The new submap opens in the Message Mapping editor. If the original map file was called `simple_mapping.msgmap`, the new submap is called `simple_mapping_submap0.msgmap`.

Related concepts:

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

Related tasks:

"Creating a message map file in the Broker Development view" on page 2233

"Creating a new submap for a wildcard source"
You can map a wildcard value in the source to a wildcard value in the target.

"Converting a message map to a submap" on page 2302

"Calling a submap" on page 2305

"Calling a map from ESQL" on page 2306

"Creating and calling submaps and subroutines" on page 2298
Use submaps, ESQL subroutines, or both, to map source elements to target elements.

Creating a new submap for a wildcard source:

You can map a wildcard value in the source to a wildcard value in the target.

About this task

You might expect a wildcard in a Mapping node for example, when you are using a SOAP message (where the Body element contains a wildcard). This type of wildcard represents the payload of the message, where the payload is a message that is defined elsewhere in the message set. The submap can involve from 0 to n source wildcards and 0 or 1 target wildcards.

The "Message Mapping editor" on page 4981 shows three kinds of wildcard, all of which allow you to create a submap:

Mapper construct	Message model construct	Choose concrete item for submap
Wildcard element	Wildcard element	Global element
Wildcard attribute	Wildcard attribute	Global attribute
Message with Wildcard Message child	Group with `Composition` of Message and `Content Validation` of Open or Open Defined	Message

Procedure

1. Switch to the Broker Application Development perspective.
2. Open the message map for the required node.
3. In the Source pane, expand the tree and select the source wildcard.
4. In the Target pane, expand the tree and select the target wildcard.
5. Right-click either the source or the target wildcard, and click **Create new submap**. The Wildcard Specification wizard opens.
6. From the Wildcard Specification wizard, select the concrete item that will replace the source wildcard, according to the values shown in the table at the beginning of this topic.
7. Click **Next**.
8. From the Wildcard Specification wizard, select the concrete item that will replace the target wildcard, according to the values shown in the table at the beginning of this topic.
9. Click **Finish**.
10. Click **OK**. The submap opens in the Message Mapping editor.
11. From the submap, map the source message elements to the target message elements as required.
12. Click **OK**.

Related concepts:

"Mapping node" on page 4994
The Mapping node has one or more mappings that are stored in message map files (with a `.msgmap` file extension). These files are configured using the Message Mapping editor.

Related tasks:

"Creating a new submap" on page 2299

"Calling a submap" on page 2305

"Calling a map from ESQL" on page 2306

"Creating and calling submaps and subroutines" on page 2298
Use submaps, ESQL subroutines, or both, to map source elements to target elements.

Creating a submap to modify a database:

About this task

Use the Create New Database Submap wizard to create a submap to modify a database.

You must have an existing message map from which to call the submap. The following steps describe how to create a submap to modify a database:

Procedure

1. In the Broker Application Development perspective, open the calling message map.
2. In the Source pane, right-click the message component containing the fields to be used to modify the database and click **Create New Database Submap**. The source can be a wildcard, an element, or an attribute. The Create New Database Submap wizard opens.
3. If the selected source is a wildcard, select a message or message component for the source wildcard from **Select a defined item to replace the source wildcard** pane. If you cannot find the message components that you expect, clear the **Apply working set filtering to artifact selection(s) on the page** check box.
4. From **Select database submap targets** expand the database operation that you want to perform. You can select from the following database operations:
   - **Table Inserts**
   - **Table Updates**
   - **Table Deletes**
5. Select the database tables that you want to map. If you cannot find the Data Targets that you expect, select the **Show all resources in workspace** check box.
6. Click **OK**. A new submap is created with the selected message or message component in the Source pane, and the database table in the Target pane. In the calling message map `$db:call` is added to the Target pane.

What to do next

After you have created the submap file, configure the message mappings for the database table.

Related concepts:

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

"Mapping node" on page 4994
The Mapping node has one or more mappings that are stored in message map files (with a `.msgmap` file extension). These files are configured using the Message Mapping editor.

"WebSphere Message Broker Toolkit" on page 31
The WebSphere Message Broker Toolkit is an integrated development environment and graphical user interface based on the Eclipse platform.

Related tasks:

"Creating a new submap" on page 2299

"Calling a map from ESQL" on page 2306

"Creating and calling submaps and subroutines" on page 2298
Use submaps, ESQL subroutines, or both, to map source elements to target elements.

Related reference:

"Message Mapping editor Target pane" on page 4987
Details of the elements present in the Target pane of the Message Mapping Editor.

"Broker Application Development perspective" on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

Converting a message map to a submap:

About this task

You can convert between a message map and a submap in order to change the usage of the map. You might convert a message map to a submap because you want to reuse the same mappings for multiple nodes. Use the following instructions to convert a message map to a submap for each message in the message map.

Procedure

1. From the Broker Application Development perspective right-click your message map and click **Open**.
2. Right-click `$source` in the Source pane and select **Add or Remove Headers and Folders**. The Add or Remove Headers and Folders dialog opens.
3. Select **No folders (map body element only)**. Any previously selected headers or folders are cleared.
4. Click **OK** to remove the headers and folders.
5. Repeat steps 2 to 4 to select to map body element only from your target message under `$target` in the Target pane.
6. Delete target map statements for existing mappings to properties, message headers or other folders such as LocalEnvironment. These mappings are flagged with warning messages after the headers are removed.
7. Remove the reference to the new submap from any mapping nodes. If a reference to the submap exists in the Mapping Routine property of a mapping node an error message is displayed on the message flow.
8. Save the submap, and check for any broken references as indicated by errors or warnings in the Problems view.

Results

The submap is now ready to be used. See calling a submap for more information.

Example

To convert a submap to a message map, click **Add or Remove Headers and Folders** for the source and target messages, and select to map **Selected headers**. You must ensure that no other maps call the changed map, check for errors in the Problems view to indicate this problem. See mapping headers and folders for more information about mapping headers, Properties and the LocalEnvironment.

Related concepts:

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

Related tasks:

"Editing a default-generated map manually" on page 2312
A map that is generated by the Message Mapping editor might not do everything that you want; you can change maps manually to enhance its operation.

"Creating and calling submaps and subroutines" on page 2298
Use submaps, ESQL subroutines, or both, to map source elements to target elements.

"Configuring message mappings" on page 2237
Use the Message Mapping editor to configure a message mapping.

Related reference:

"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.
"Mapping node" on page 4571
Use the Mapping node to construct one or more new messages and populate them with various types of information.
"DataDelete node" on page 4382
Use the DataDelete node to interact with a database in the specified ODBC data source.

Converting an inline mapping to a submap:

To change the usage of a map, you can convert between inline mappings in a message map and a submap.

About this task

You might convert parts of an existing message map to a submap because you want to reuse the same mappings for multiple nodes. You can convert inline mappings to submaps from messages or databases. You must select one of the following types of statement to create a submap or a database submap:

- Element statement that maps a global element or an element of global type
- Attribute statement that maps either a global attribute or an attribute of a global type
- Database insert statement
- Database update statement
- Database delete statement

The target that is added to the new submap is the global element, attribute, type, or database insert, update, or delete that you select. The source that is added to the new submap is the appropriate global element or type from the source from the selected mappings that are included in the submap. If mappings included in the selected statement do not reference any source, only a target is added to the submap. If the source contains a database select that is not referenced by any other part of the original message map, a database select is added as a source to the submap, and removed from the original message map. However, if the source contains a database select that is referenced by any other part of the original message map, the original message map retains the select, and the submap performs a separate select. If you do not want to perform two database select operations, do not use a database submap under these conditions.

Use the following instructions to convert inline mappings in a message map to a submap or database submap:

Procedure

1. From the Broker Application Development perspective right-click your message map and click **Open**.
2. Right-click the mapping statement that you want to convert to a submap or database submap in the Script pane, and click **Convert to submap**. A new submap is created and a statement is added to the original message map to call the new submap.

Results

The submap is now ready to be used.

Related concepts:

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

Related tasks:

"Editing a default-generated map manually" on page 2312
A map that is generated by the Message Mapping editor might not do everything that you want; you can change maps manually to enhance its operation.

"Creating and calling submaps and subroutines" on page 2298
Use submaps, ESQL subroutines, or both, to map source elements to target elements.

"Configuring message mappings" on page 2237
Use the Message Mapping editor to configure a message mapping.

Related reference:

"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.

"Mapping node" on page 4571
Use the Mapping node to construct one or more new messages and populate them with various types of information.

Calling a submap:
About this task

Use the Call Existing Submap wizard to call a submap. The submap must already be in the workspace.

If a submap does not exist, use the **Create New Submap** menu option to create a submap that you can call. This action creates the new submap in the same folder as the calling map. It also allocates a default map operation name to the new submap. If the source or target in the calling map is a wildcard, a wizard allows you to choose a replacement element.

You can also map from a wildcard to a wildcard.

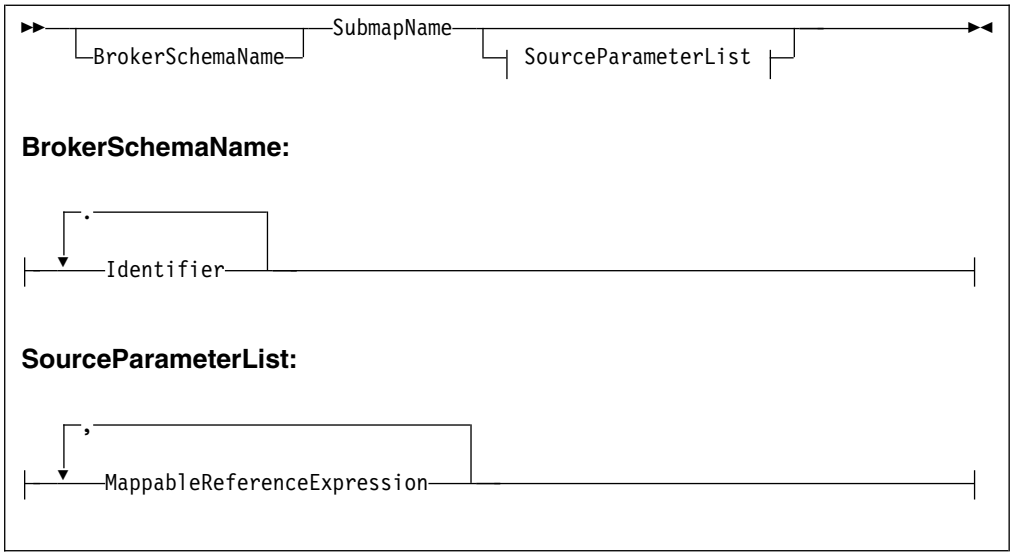
The following steps describe how to call a submap:

Procedure

1. In the Broker Application Development perspective, open the calling map.
2. In the Source and Target panes, select one or more sources and one target. Any of the sources or the target can be a wildcard, an element, or an attribute.
3. Click **Map** > **Call Existing Submap**. The Call Existing Submap wizard opens.
4. Complete the wizard, following the on-screen instructions.

Results

The call to the submap takes the following format:



Only source parameters appear in the call and only message parameters appear in the list.

Related concepts:

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

"Mapping node" on page 4994
The Mapping node has one or more mappings that are stored in message map files (with a `.msgmap` file extension). These files are configured using the Message Mapping editor.

"WebSphere Message Broker Toolkit" on page 31
The WebSphere Message Broker Toolkit is an integrated development environment and graphical user interface based on the Eclipse platform.

Related tasks:

"Creating a new submap" on page 2299

"Calling a map from ESQL"

"Creating and calling submaps and subroutines" on page 2298
Use submaps, ESQL subroutines, or both, to map source elements to target elements.

Related reference:

"Message Mapping editor Target pane" on page 4987
Details of the elements present in the Target pane of the Message Mapping Editor.

"Broker Application Development perspective" on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

Calling a map from ESQL:

You can use the Message Mapping editor to perform mappings to a certain level of complexity. To create even more complex mappings, use ESQL. ESQL is particularly suitable for interacting with databases.

Before you begin

If a map does not already exist, create one.

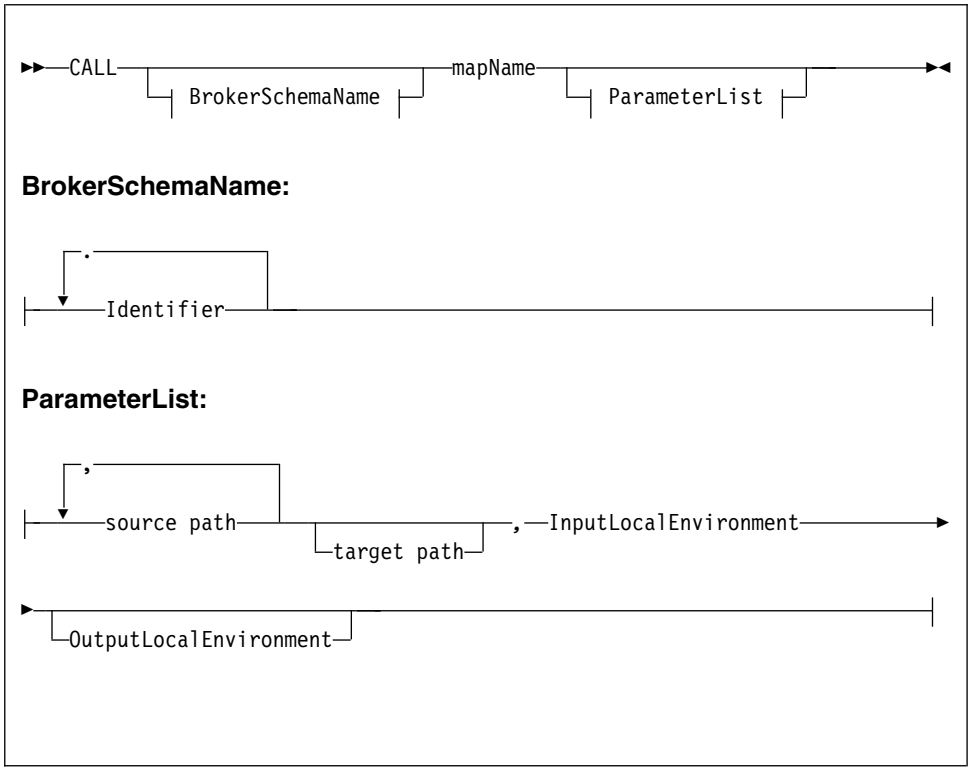
About this task

The following steps describe how to call a map (which can be a submap) from ESQL. Calling a map from ESQL uses different parameters to when you call a submap from another map due to this extra level of complexity (when calling a map from ESQL, the two local environment parameters are added at the end of the CALL statement).

Procedure

1. Right-click a node that supports ESQL and click **Open ESQL**. The ESQL file opens for that node.
2. Add a CALL statement to call a map. Alternatively, press Ctrl+Space to access ESQL content assist, which provides a drop-down list that includes the map name and expected parameters.

   The following syntax diagram demonstrates the CALL statement:



Results

Notes:

1. Only source parameters appear in the call and only message parameters appear in the list.
2. If the map builds a message target, include the `target path` and `OutputLocalEnvironment` parameters. If the map does not build a message target (for example, if the map interacts with a database), these two parameters do not appear.

Related tasks:

"Calling a submap" on page 2305

"Creating a new submap" on page 2299

"Creating a new submap for a wildcard source" on page 2300
You can map a wildcard value in the source to a wildcard value in the target.

"Creating and calling submaps and subroutines" on page 2298
Use submaps, ESQL subroutines, or both, to map source elements to target elements.

Calling an ESQL routine:

About this task

To call an existing ESQL routine from a mapping, select the routine from the Call Existing ESQL Routine wizard. The ESQL routine must already exist in the workspace.

Procedure

1. Switch to the Broker Application Development perspective.
2. Open the required mapping.
3. In the Source pane, select the required source.
4. In the Target pane, select the required target.
5. Right-click either the Source or Target pane and click **Call ESQL Routine**. The Call ESQL routine wizard opens.
6. Select the routine where the parameters and return types match the source and target selection.
7. Click **OK**.

Related concepts:

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

Related tasks:

"Creating and calling submaps and subroutines" on page 2298
Use submaps, ESQL subroutines, or both, to map source elements to target elements.

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"Message Mapping editor Source pane" on page 4983
Details of the elements present in the Source pane of the Message Mapping Editor.

"Message Mapping editor Target pane" on page 4987
Details of the elements present in the Target pane of the Message Mapping Editor.

"Broker Application Development perspective" on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

Creating and calling your own user-defined ESQL routine:

For complex mappings, you can create user-defined ESQL functions that can be called from the Message Mapping editor.

About this task

This topic describes how to create a user-defined ESQL function, and how to use it in an existing message map.

Procedure

1. Create a new ESQL file, or open an existing ESQL file.
2. Enter your ESQL function in the ESQL file. Ensure that you do not enter the ESQL in any existing modules.
3. Save the ESQL file.
4. Right-click your Mapping node node, and click **Open Map** to open your message map in the Message Mapping editor.
5. Select the target that you want to generate using your ESQL function from the appropriate target message or target database table.
6. In the Edit pane, enter the expression to call the ESQL function and any parameters to pass to the function. For example:

   ```
   esql:concatValues($source/Pager/Text, ' Powered by IBM.')
   ```

   Where `concatValues` is the name of the user-defined ESQL function and the following parameters:

   - `$source/Pager/Text` is a field in the source message
   - `' Powered by IBM.'` is text

   The following code is the ESQL used for the user-defined ESQL function in the preceding example:

   ```
   CREATE FUNCTION concatValues(IN val INTEGER, IN str CHAR) RETURNS CHAR
    BEGIN
     return str || ' plus int val ' || CAST(val AS CHAR);
    END;
   ```

   You can also use **Edit** > **Content Assist** to select user-defined ESQL functions. The user-defined ESQL functions are located at the end of the list of ESQL functions.
7. Save the message map file by clicking **File** > **Save**.

Related concepts:

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

"Client application programming interfaces" on page 1038
You can configure the nodes in your message flows to customize the behavior of those nodes by using one or more of the supported programming interfaces.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"Message Mapping editor Spreadsheet pane" on page 4991
Details of the actions available to you in the Spreadsheet pane of the Message Mapping Editor.

"Message Mapping editor Edit pane" on page 4990
Details of how you use the Edit pane of the Message Mapping Editor.

"Mapping node" on page 4571
Use the Mapping node to construct one or more new messages and populate them with various types of information.

"CREATE FUNCTION statement" on page 5091
The CREATE FUNCTION statement defines a callable function or procedure.

Calling a Java method:

To call an existing Java method from a mapping node, select the method from the Call Existing Java Method wizard, or enter an XPath expression in the Edit pane.

Before you begin

See "Message mapping tips and restrictions" on page 2314 for information about the types of methods that are available through the wizard, and through content assist.

Using the wizard:
About this task

To use the Call Existing Java Method wizard, take the following steps:

Procedure

1. Switch to the Broker Application Development perspective.
2. Open the required message map.
3. If the method requires input parameters, select one or more fields in the Source pane. Your choice determines which methods are subsequently displayed in the wizard. If you select no source fields, the wizard shows only methods that take no parameter. If you select two fields, the wizard displays only methods that take two parameters, and so on.
4. In the Target pane, select the required target field to be mapped to the return value of the Java method. The target field must be a simple, non-wildcard type.
5. Right-click either the Source or Target pane and click **Call Java Method**. The Call Existing Java Method wizard opens.
6. Select the method, then click **OK**.

Entering an XPath expression:
About this task

You can enter the expression directly, without using content assist. Enter a function call expression, with the following syntax:

```
java:package_name.class_name.method_name (parameters)
```

You can omit the package name if there is no package, or if you are using a default package.

To use content assist, take the following steps:

Procedure

1. Switch to the Broker Application Development perspective.
2. Open the required message map.
3. In the Edit pane, click **Edit** > **Content Assist**.
4. Select **java: (Java Methods)**, then click **Edit** > **Content Assist**. All qualifying Java methods are displayed.
5. Select the method.

6. If the method requires input parameters, drag the appropriate source fields to the method's parameter area. The number of source fields included must match the number of input parameters that the method takes.

Example

This is an example of a method that takes one input parameter:

```
java:mypackage1.MyClass1.myMethod1($source/po:purchaseOrder/po:comment)
```

Separate parameters by a comma:

```
java:mypackage1.MyClass1.myMethod1($source/po:purchaseOrder/name,
  $source/po:purchaseOrder/phone)
```

Related concepts:

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

Related tasks:

"Creating and calling submaps and subroutines" on page 2298
Use submaps, ESQL subroutines, or both, to map source elements to target elements.

Related reference:

"Message Mapping editor Source pane" on page 4983
Details of the elements present in the Source pane of the Message Mapping Editor.

"Message Mapping editor Target pane" on page 4987
Details of the elements present in the Target pane of the Message Mapping Editor.

"Broker Application Development perspective" on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

Transforming a SOAP request message:
About this task

SOAP is an XML-based language defined by the W3C for sending data between applications. A SOAP message comprises an envelope containing:

- An optional header (containing one or more header blocks)
- A mandatory body.

For common envelope message formats, such as SOAP, where both the envelope and the messages that can appear within that envelope have to be modeled, use the Message Mapping editor to select from available messages at points in the model that are defined with **Composition="message"** and **Content validation="open"** or **"open defined"**.

Define the mappings by selecting from the allowed constituent messages. For example, in the case of SOAP, the outer level message is called Envelope and has one mandatory child element called Body, which is modeled with **Composition="message"**. If the permitted content of Body is modeled by separate messages Msg1 ... MsgN, define mappings for each separate message (Envelope.Body.Msg1 to Envelope.Body.MsgN).

For complex type elements with type composition message, the Message Mapping editor follows these rules:

Content validation	Messages offered
Closed	Messages available in any message sets in the workspace
Open defined	Any message defined within the current message set
Open	The Message Mapping editor does not support open or open defined content when the type composition is NOT message

Mapping an embedded message

When you are working with type composition message, with content open or open-defined (and no children defined), map the embedded message using a submap:

Procedure

1. In the main map, expand the levels (both source and target) of Envelope and Body until you find the wildcard message, and select this on both the source and target sides.
2. Right-click either the source or target and click **Create New Submap**.
3. From the dialog box, select a source (for example `reqmess`) and a target (for example `rspmess`).
4. With the submap open in the Message Mapping editor, make the appropriate mappings between the source (`reqmess`) and target (`rspmess`).

Related concepts:

"Message model objects: Wildcard attributes" on page 1187
For XML messages, a *wildcard attribute* enables unmodeled attributes to be present in a message.

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

Related tasks:

"Creating a new submap for a wildcard source" on page 2300
You can map a wildcard value in the source to a wildcard value in the target.

Related reference:

"Wildcard element properties" on page 6064
Different types of properties are available for a wildcard element.

"Wildcard attribute properties" on page 6060
Different types of properties are available for a wildcard attribute.

Editing a default-generated map manually:

A map that is generated by the Message Mapping editor might not do everything that you want; you can change maps manually to enhance its operation.

About this task

You can edit the structure directly by inserting, moving, copying, pasting, and deleting rows. The pop-up menu provides a list of available editing actions with their keyboard equivalents. Here are some specific operations that you might want to perform:

Procedure

- "Creating message headers"
- "Creating conditional mappings"

Creating message headers:
About this task

When you create a map from a Mapping node, or if you select the option **Message map called by a message flow node** from the New Message Map wizard, the map that is created allows additional elements including WebSphere MQ, HTTP, and JMS headers to be mapped.

If you use a Mapping node for a database to message mapping without specifying a source message, the Message Mapping editor cannot generate an output header for the map file that is created. To ensure that an output header is created, perform one of the following steps:

Procedure

- When you create the message map, add message headers to the target message and ensure that all mandatory fields in the header are set.
- Add an additional source message to the map. The source message must be the same message as the intended target message. You do not need to create any mappings from the source message because the headers from the source message are automatically copied to the output message tree.

Creating conditional mappings:
About this task

When a mapping involves one of the following items:

- Schema choice group
- Derived type element
- Substitution group member
- Wildcard character
- Repeating element

the default mapping that is generated by the Message Mapping editor might be placed under an `if` statement. If the `if` statement is not what you had expected, edit the statements; here are the changes that you can make:

Procedure

- Move statements in or out of an `if/elseif/else` block.
- Reorder `if` and `elseif` statements.
- Create new `elseif` statements.
- Create new `if` statements.

What to do next

See the "Configuring conditional mappings" on page 2265 topic for more information about conditional mappings.

:

Related concepts:

"Message mapping tips and restrictions"
Information to help you use message mapping.

"Client application programming interfaces" on page 1038
You can configure the nodes in your message flows to customize the behavior of those nodes by using one or more of the supported programming interfaces.

Related tasks:

"Creating a message map file in the Broker Development view" on page 2233

"Configuring message mappings" on page 2237
Use the Message Mapping editor to configure a message mapping.

"Configuring conditional mappings" on page 2265
How to set the value of a target element conditionally in a Mapping node.

"Accessing the MQRFH2 header" on page 2456
Code ESQL statements to access the fields of the MQRFH2 header.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Mapping node" on page 4571
Use the Mapping node to construct one or more new messages and populate them with various types of information.

Message mapping tips and restrictions:

Information to help you use message mapping.

These tips assume that you have created a Mapping node within the message flow, opened the Message Mapping editor, and selected both a source and a target message:

- "Mapping a message when the source is a list and the target is a list from source, but with a new entry at the top of the list"
- "Changing the target message domain" on page 2315
- "Overriding the database schema name" on page 2315
- "Mapping batch messages" on page 2316
- "Mapping restrictions" on page 2316
- "Requirements for calling a Java method" on page 2316

Mapping a message when the source is a list and the target is a list from source, but with a new entry at the top of the list

1. Expand the target to display the element for which you want to create a new first instance. This might be a structure or a simple element.
2. Right-click the element and click **If**. An `if` line appears and wraps around the element.
3. Right-click the `if` line and click **Else If**. There are now two entries in the spreadsheet for your element.
4. Set the first of these entries to values of your choice.
5. Right-click the second entry and click **For**. A `for` line appears in the spreadsheet.
6. Set the second entry to the value or values mapped from the source.
7. Set the `for` entry to the looping condition.
8. Click **For**, then drag the source field that represents the loop condition to the Expression editor.

Changing the target message domain

When you first create a mapping, you nominate a message set for the target message. The parser that is associated with the output message is determined by the Message Domain property of the message set. For example, when a message set is first created, the default message domain is MRM. Therefore, the Mapping node generates ESQL with the following format:

```
SET OutputRoot.MRM.Fielda...
```

If you change the runtime parser to XMLNSC, for example, the Mapping node generates ESQL with the following format:

```
SET OutputRoot.XMLNSC.MessageA.FieldA...
```

The parser of the source message is determined by the contents of the MQRFH2 header or by the properties of the input node. The Mapping node generates a target message with a parser that matches the message domain of the message set. The Mapping node supports the following message domains:

MRM
XMLNSC
XMLNS
MIME
SOAP
DataObject
JMSMap
JMSStream
XML
BLOB
IDOC

To change the message domain property of your message set:

1. Open the message set file `messageset.mset`.
2. Change the Message Domain property to a supported domain.
3. Save your message set, and save any message flows and message maps that reference your message set, if they have not already been saved. Saving these files generates updated ESQL for mapping the changed message set.

   If you have made no updates to your flows or message maps after changing the message domain of your message set, you must clean the related message flow projects so that updated ESQL code can be generated:

   a. Select a project and click **Project** > **Clean Project**.
   b. Select **Clean all projects** or **Clean selected projects**.
   c. Click **OK**.
4. Deploy the changed message set.
5. Deploy the message flow that contains the mappings, and test your ESQL in a Compute node and in other nodes to ensure that the message flow still functions as expected.

Overriding the database schema name

To change the database schema name that is generated in ESQL, use the Override Database Schema wizard in the Specify Runtime Schema dialog box. The default name is the schema name of the database definitions that are imported into the

WebSphere Message Broker Toolkit. Use the Specify Runtime Schema dialog box to change the value.

Mapping batch messages

You can configure a message mapping that sorts, orders, and splits the components of a multipart message into a series of batch messages. These components can be messages or objects, and they can have different formats; in this case, each component is converted and the message is reassembled before being forwarded.

1. Use a "RouteToLabel node" on page 4673 in the message flow to receive multipart messages as input.

   The RouteToLabel node is the next node in sequence after the "Mapping node" on page 4571, and causes the flow to jump automatically to the specified label. You can specify a single RouteToLabel value in a splitting map for all maps that output a message assembly. You can also use conditions to set the RouteToLabel value, depending on the values in the source message.
2. Use the "Message Mapping editor" on page 4981 to build maps that transform and propagate batch messages using a single node, without having to define an intermediate data structure.

Multipart messages can also contain repeating embedded messages, where each repeated instance of a message is propagated separately. Embedded messages must be from the same message set as the parent message.

Mapping restrictions

Unless stated explicitly, you can achieve the required functionality by calling an ESQL function or procedure. The following restrictions apply:

- Exceptions cannot be thrown directly in Mapping nodes.
- Self-defined elements cannot be manipulated in Mapping nodes (support for wildcard symbols is limited if the wildcard symbols represent embedded messages).
- The Environment tree cannot be manipulated in the Mapping node.
- User variables cannot be defined or set.
- CASE expressions cannot be emulated; you must use IF ... ELSE.
- Trees cannot be copied from input to output in order to modify elements within the copied tree. For example, the following ESQL cannot be modeled in a Mapping node:

  ```
  SET OutputRoot.MQMD = InputRoot.MQMD; SET OutputRoot.MQMD.ReplyToQ = 'NEW.QUEUE';
  ```

  You must set each field in the structure individually if you intend to modify one or more sibling fields.

Requirements for calling a Java method

All of the following conditions must be satisfied for the method to be shown in the Call Existing Java Method wizard, or in content assist:

- The method must be public static, in a Java project.
- The method must be in a Java project accessible from the project map.
- If the method is inside a JAR file, the JAR file must be in the Java build path. To add a JAR file to the build path:
  1. Right-click the project folder for the project on which you are working, and click **Properties**.

2. Click **Java Build Path** in the left pane.
3. Click the **Libraries** tab.
4. Complete one of the following steps:
   - To add an internal dependency, click **Add JARs**, select the JAR file that you want to add, then click **OK**.
   - To add an external dependency, click **Add External JARs**, select the JAR file that you want to add, then click **Open**. Copy the JAR file to the `shared-classes` directory required. For more details of the `shared-classes` directories available and the effects of each, see "Java shared classloader" on page 2637. If you do not copy the JAR file to a valid `shared-classes` directory, `ClassNotFoundException` exceptions are generated at run time.

- The default scope of the search is all methods in .java source files in the workspace, excluding application libraries and application JAR files in the Java build path, and all the JRE system libraries. To change the scope, change the preferences in the Toolkit:
  1. Click **Window** > **Preferences**.
  2. Expand the **Broker Development** node, then click **Message Map Editor**.
  3. Select and clear the check boxes as appropriate.
- The method must have a return value.
- Return values and Java parameters must be one of the following data types:

Data types	Equivalent XML schema type for mapping	Comments
java.lang.Long	byte, unsignedShort, long, unsignedByte, short, int, unsignedInt	
java.lang.Double	float, double	
java.math.BigDecimal	nonNegativeInteger, negativeInteger, integer, nonPositiveInteger, positiveInteger, unsignedLong, decimal	
java.lang.String	NCName, Name, IDREF, normalizedString, string, anyURI, NOTATION, token, NMTOKEN, language, ID, ENTITIES, QName, ENTITY	
byte[]	base64Binary, hexBinary	
com.ibm.broker.plugin.MbDate	date, gYear, gMonth, gDay, gYearMonth, gMonthDay	
com.ibm.broker.plugin.MbTime	time	
com.ibm.broker.plugin.MbTimestamp	dateTime	
java.lang.Boolean	boolean	
com.ibm.broker.plugin.MbElement	A complex type	Valid only for input parameters, not return values.

- The method must not have a `throws` clause.
- The number of source fields selected must match the number of input parameters that the method takes.
- If you are using a working set, only the methods in the current working set are displayed. Clear the check box **Apply working set filtering to artifact selection(s) on the page** to display all the methods in the workspace. If you are

not using a working set, all the methods in the entire workspace are displayed. Content assist shows all methods in the workspace, whether you are using a working set or not.

When you create the BAR file you must select the Java project or JAR file that contains the method that you are calling.

Related concepts:

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

"MRM parser and domain" on page 1111
You can use the MRM domain to parse and write a wide range of message formats.

"XML parsers and domains" on page 1084
You can use XML domains to parse and write messages that conform to the W3C XML standard.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"Mapping node" on page 4571
Use the Mapping node to construct one or more new messages and populate them with various types of information.

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"RouteToLabel node" on page 4673
Use the RouteToLabel node in combination with one or more Label nodes to dynamically determine the route that a message takes through the message flow, based on its content.

"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.

Message mapping scenarios

About this task

This section contains some message mapping scenarios that demonstrate how to make the most of the message mapping functions:

Procedure

- "Scenario A: Mapping an airline message" on page 2319
- "Scenario B: Simple message enrichment" on page 2330
- "Scenario C: Using a broker as auditor" on page 2338
- "Scenario D: Complex message enrichment" on page 2345
- "Scenario E: Resolving a choice with alternative message data" on page 2368
- "Scenario F: Updating the value of a message element" on page 2369

Related tasks:

"Using message mappings" on page 2228
Message mappings define the blueprint for creating a message.

Scenario A: Mapping an airline message:

This scenario demonstrates how to create, configure, and deploy a new message mapping.

About this task

The message flow that is used in this example reads an XML input message (an airline message), then uses a Mapping node to achieve the following transformations:

- Convert the input message from XML to COBOL
- Modify an element in the input message from the results that are obtained by a database lookup
- Concatenate two elements in the input message to form a single element in the output message

Follow these steps to complete this scenario:

Procedure

1. "Example names and values"
2. "Connect to the database and obtain the definition files" on page 2321
3. "Create the message flow" on page 2323
4. "Create the mapping file" on page 2324
5. "Configure the mapping file" on page 2325
6. "Deployment of the mapping" on page 2329

What to do next

Next: Go to "Example names and values."

Related tasks:

"Using message mappings" on page 2228
Message mappings define the blueprint for creating a message.

"Transforming and enriching messages" on page 2227
Transform and enrich messages by using one or more of the techniques described in this section.

Related reference:

"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.

"Mapping node" on page 4571
Use the Mapping node to construct one or more new messages and populate them with various types of information.

Example names and values:

View the resources that are created for the airline message scenario.

About this task

The following table describes the names and definitions of the resources that are created.

Resource	Name	Definition
Alias name	AIRLINEDBALIAS	The same as connection name and database name in this case
Broker archive (BAR) file name	AIRLINE	Contains the message flow and message set projects, and the mapping file, and is deployed to the default execution group for the run time
COBOL copybook	AirlineRequest.cbl	Controls the structure of the COBOL output message
Connection name	AIRLINECONN	The same as alias name and connection name in this case
Database	AIRLINEDB	Contains the table XREF and is the same as the connection name and the alias name in this case
Database table (table tree)	XREF	Contains lookup information (in this case the two-code airline city code abbreviations STATE=Illinois, ABBREV=IL)
Default project	AIRLINE_MFP	The default message flow project. You copy the database definitions to this project.
Default queue manager	WBRK6_DEFAULT_QUEUE_MANAGER	The default queue manager that controls the message queue
ESQL select operation	`$db:select.AIRLINEDB.AIRLINE_SCHEMTREE. XREF.ABBERV`	The ESQL select operation that performs a qualified database select operation
Input (XML) message	`c:\airline\data\AirlineRequest.xml`	The input message (in this case an XML message)
Input message source fields	FirstName,LastName	The source elements in the input message that are concatenated
Input queue name property	AIRLINE_Mapping_IN	The input queue
Mapping node rename	XML_TO_COBOL	The name of the node in the message flow that performs the mapping (the node was renamed from its default name)
Message mapping file name	AIRLINE.msgmap	The file that contains the mapping configuration used by the Mapping node
Message Set property	AIRLINE_MSP2	The message set project name
Message Type property	msg_AIRLINEREQUEST	The message type
Message Format	Binary1	The custom wire format (CWF) for COBOL output message
Message flow name	AIRLINE_Mapping	The name of the message flow
Message flow project	AIRLINE_MFP	The name of the message flow project
Message set projects	AIRLINE_MSP1,AIRLINE_MSP2	The names of the message set projects

Resource	Name	Definition
Msg Domain node property	MRM	The message domain node property
Msg Set Name node property	AIRLINE_MSP1	The message set name node property
Msg Type property	AirlineRequest	The message type property
Msg Format node property	XML1	The input message format
Output message target field	NAME	The result of concatenating FirstName and LastName in the input message. NAME is the element that is created in the output message.
Output queue name property	AIRLINE_Mapping_OUT	The output queue name
Resource folder	airline\resources	The folder where the mapping resources are stored
Schema tree	AIRLINE_SCHEMTREE	The name of the schema tree
Source	ABBREV	The source
Source tree	$source/AirlineRequest	The source tree
Source message	AirlineRequest	The source message
Target	STATE	The target
Target message	AIRLINEREQUEST	The target message
Target tree	$target/AIRLINEREQUEST	The target tree
XPath concatenation function	`fn:concat(fn:concat($source/ AirlineRequest/Purchase/ Customer/FirstName,' '), $source/AirlineRequest/ Purchase/Customer/LastName)`	The W3C XPath 1.0 Specification function that concatenates FirstName and LastName

What to do next

Now go to "Connect to the database and obtain the definition files."

Related tasks:

"Using message mappings" on page 2228
Message mappings define the blueprint for creating a message.

"Message mapping scenarios" on page 2318

Related reference:

"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.

Connect to the database and obtain the definition files:

Demonstrate how to define a database connection to enable a message flow to access a database table. You must define the database to the WebSphere Message Broker Toolkit.

Before you begin

Before you start:

Create a message flow project.

About this task

The Database Definition wizard uses the JDBC interface to communicate with the database server; therefore, you must ensure that the database client JAR file on your local or WebSphere Message Broker Toolkit host is at a compatible level to allow communication with the database server. If necessary, consult your database vendor for further advice.

Procedure

1. Switch to the Broker Application Development perspective.
2. Select the message flow project that you want to create the database definition files in, and click **File** > **New** > **Database Definition**. The New Database Definition File wizard opens.
3. Select an existing data design project, or click **New** to create a new data design project.
4. Select the database type and version that you want to connect to from the **Database and Version** list. Ensure that you select a database from the list that is supported by WebSphere Message Broker; you can use this wizard in a shell-share environment with other Rational products that support other databases or versions.

   For a list of databases supported by the broker, see "Supported databases" on page 3591.
5. Click **Next**.
6. Select **To create a new database connection**, and click **Next**.
7. Clear **Use default naming convention**, and enter a connection name; for example, AIRLINEDBALIAS.
8. Enter values for the Connection to the database; for example, Database name, Host name, and Port number.
9. Enter values for the User ID and Password to connect to the database. Click **Test Connection** to verify the settings that you have selected for your database. The default Port number for a DB2 database is 50000. If the connection fails, enter other values such as 50001, 50002 and so on, for the Port number, and test the connection again.
10. Click **Next**.
11. Select one or more database schemas from the list, and click **Next**.
12. Click **Finish**.
13. Add the data design project as a reference to the message flow project:
    a. Right-click the message flow project, and click **Properties**.
    b. Click **Project References**, and select the data design project from the list to add as a referenced project.
    c. Click **OK**.

Results

The database definition is added to a new Data design project. You have now defined the database to the mapping tools.

What to do next

Now go to "Create the message flow."

Related reference:

"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.

"Supported databases" on page 3591
You can optionally configure databases to contain data that is accessed by your message flows. Databases from IBM and other suppliers are supported at specific versions on supported operating systems.

Create the message flow:

Before you begin

Before you start:

1. Create a message flow project.
2. "Connect to the database and obtain the definition files" on page 2321.
3. Create a message flow by adding an MQInput node, and renaming the node (for example, to AIRLINE_Mapping_IN).
4. Set the queue name property (for example, to AIRLINE_Mapping_IN).
5. Add an MQOutput node to the message flow, and rename the node (for example, to AIRLINE_Mapping_OUT).

About this task

This topic demonstrates how to specify a message flow project, add a Mapping node, wire the nodes, and set the node properties.

Procedure

1. Switch to the Broker Application Development perspective.
2. Open the message flow (for example, AIRLINE_Mapping) within the message flow project (for example, AIRLINE_MFP). This message flow forms the starting point for the mapping task.
3. Open the palette of nodes and add a Mapping node to the message flow. You might need to scroll down to find the Mapping node.
4. Rename the Mapping node (for example, to XML_TO_COBOL) by right-clicking the node and clicking **Rename**.
5. Wire the node terminals (for example, AIRLINE_Mapping_IN> XML_TO_COBOL> AIRLINE_Mapping_OUT).
6. Modify the properties of the MQInput node (for example, AIRLINE_Mapping_IN) by right-clicking the node and clicking **Properties**.
7. Click **OK**.
8. Modify the properties of the Mapping node (for example, XML_TO_COBOL).
9. Set the data source as the database name (for example, AIRLINEDBALIAS)
10. Click **OK**.

Results

You have now created the required message flow, wired the nodes, and set the node properties.

What to do next

Now go to "Create the mapping file."

Related reference:

"Message Flow editor" on page 6810
The Message Flow editor is the default editor provided by the Broker Application Development perspective for defining a graphical representation of a message flow in the WebSphere Message Broker Toolkit, and for setting properties for individual message flow nodes.

"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.

Create the mapping file:
Before you begin

Before you start:

Follow the instructions in these topics:

1. "Connect to the database and obtain the definition files" on page 2321
2. "Create the message flow" on page 2323

About this task

This topic demonstrates how to create a new mapping file, specify how it will be used, and specify the source and target mappable elements.

Procedure

1. Switch to the Broker Application Development perspective.
2. From the message flow, right-click the mapping node (for example XML_TO_COBOL) and click **Open Map**. The New Message Map wizard opens.
3. Select the combination of Messages, Data Sources or both that you want to use as sources for your map from **Select map sources** (for example, AirlineRequest) from the first message set project (for example, AIRLINE_MSP1). If you cannot find the Messages, Data Sources or Data Targets that you expect, clear the **Apply working set filtering to artifact selection(s) on the page** check box.
4. Select the combination of Messages, Data Targets or both that you want to use as targets for your map from **Select map targets** (for example, AIRLINEREQUEST) from the second message set project (for example, AIRLINE_MSP2).
5. Click **Finish**.

Results

You have now created the mapping file, defined its usage, and specified the source and target mappable elements.

What to do next

Now go to "Configure the mapping file" on page 2325.

Related reference:

"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.

Configure the mapping file:
Before you begin

Before you start:

Follow the instructions in these topics:

1. "Connect to the database and obtain the definition files" on page 2321
2. "Create the message flow" on page 2323
3. "Create the mapping file" on page 2324

About this task

This set of topics demonstrates how to configure the mapping file by:

Procedure

1. Specifying a data source
2. Mapping the message properties
3. Writing an XPath function that concatenates the elements in the input message
4. Specifying an ESQL select command

What to do next

Now go to "Specify the data source."

Related reference:

"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.

Specify the data source:
Before you begin

Before you start:

Follow the instructions in these topics:

1. "Connect to the database and obtain the definition files" on page 2321
2. "Create the message flow" on page 2323
3. "Create the mapping file" on page 2324

About this task

This topic demonstrates how to specify the database to use as the source for the mapping.

Procedure

1. From the Message Mapping editor Spreadsheet pane, select an item. The item that you select determines the scope of the $db:select entry that is created by the action. For example, you can select $target, an element, an attribute, a For condition, or another $db:select entry. The Select database as mapping source dialog box opens.
2. Right-click and click **Select data source**.

3. From the Select Database as mapping source page, select a database (for example, AIRLINEDB) and click **Finish**. The Message Mapping editor adds the sources of the database table (for example, the XREF table) to the tree in the Message Mapping editor Source pane.

Results

You have now added the data source to the Message Mapping editor Source pane.

What to do next

Now go to "Map the message properties."

Related reference:

"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.

Map the message properties:

This topic demonstrates how to map the message set, message type and message format properties.

Before you begin

Before you map the message properties, ensure you do the following:

1. "Connect to the database and obtain the definition files" on page 2321
2. "Create the message flow" on page 2323
3. "Create the mapping file" on page 2324
4. "Specify the data source" on page 2325

About this task

To map the message properties:

Procedure

1. From the Message Mapping editor Spreadsheet pane, expand the entries by clicking **+** to reveal the message properties.
2. Right-click $target and click **Insert Children**.
3. Right-click Properties and click **Insert Children**. The MessageSet, MessageType and MessageFormat properties contain default values for the target message.
4. To change the format of the output message, change the MessageFormat property to the appropriate value. You must use quotation marks around the value of MessageFormat, because the values are string literals and without quotation marks, the values will be interpreted as XPath locations.
5. From the Message Mapping editor Source pane, expand the properties for the $source tree, and for each remaining property, map the source element to its corresponding target element by dragging from source to target. Alternatively, select Properties in both the source and the target panes and use **Map by Name** to map all of the properties.
6. Save the map by clicking **File** > **Save**.

Results

You have now mapped message set, message type and message format properties.

What to do next

Now go to "Add the XPath concatenate function."

Related reference:

"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.

Add the XPath concatenate function:
Before you begin

Before you start:

Follow the instructions in these topics:

1. "Connect to the database and obtain the definition files" on page 2321
2. "Create the message flow" on page 2323
3. "Create the mapping file" on page 2324
4. "Specify the data source" on page 2325
5. "Map the message properties" on page 2326

About this task

This topic demonstrates how to write an XPath function that concatenates the FirstName and LastName from the input message, and adds a white space separator in the target NAME element. When you add the XPath expression and save the map, link lines are automatically generated between the source and target to indicate that these elements are mapped.

Procedure

1. From the Message Mapping editor Source pane, select the first source to concatenate (for example, FirstName), Ctrl+click to select the second source to concatenate (for example, LastName), and drag both elements onto the target (for example, NAME) in the Target pane.
2. From the Message Mapping editor Spreadsheet pane, select the target (for example, NAME).
3. From the Edit pane, enter the XPath function (for example, `fn:concat($source/AirlineRequest/Purchase/Customer/FirstName, ' ', $source/AirlineRequest/Purchase/Customer/LastName)`
4. Save the map by clicking **File** > **Save**.

Results

You have now added an XPath function that concatenates the two source elements in the input message into a single target element in the output message.

What to do next

Now go to "Add the database Select operation."

Related reference:

"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.

Add the database Select operation:

Before you begin

Before you start:

Follow the instructions in these topics:

1. "Connect to the database and obtain the definition files" on page 2321
2. "Create the message flow" on page 2323
3. "Create the mapping file" on page 2324
4. "Specify the data source" on page 2325
5. "Map the message properties" on page 2326
6. "Add the XPath concatenate function" on page 2327

About this task

This topic provides instructions on how to add a database select operation that makes a qualified selection from a data source. In the spreadsheet pane the `$db:select` statement has the default value `fn:true()`, which returns all entries in the table. You must therefore replace this value with one that qualifies the selection, for example:

```
$db:select.LAB13STA.ARGOSTR.XREF.STATE=$source/AirlineRequest/Purchase/Customer/State
```

The XPath in this example selects only the records from the database where the value in the STATE column for each record matches the value of the State field from the input message. In the spreadsheet pane the `$db:select` statement is associated with a `For` entry which is used to iterate over the mappings for the target message. For each row in the database matching the `$db:select` statement a separate target message is created with the mappings beneath `$target`.

The following steps describe how to create message mappings to generate a target message based on records in a database matching the contents of an input message:

Procedure

1. In the spreadsheet pane replace the existing value `fn:true()` with the value to match in the database (for example a field in the input message as shown in the preceding example).
2. Create mappings from the database fields in the Source pane to include in the target message, by dragging them from the source pane onto the target elements. A `$db:select` statement is added to the value column in the spreadsheet pane (for example, `$db:select.AIRLINEDB.AIRLINE_SCHEMTREE.XREF.ABBREV`).
3. Create any mappings you require from the source message to the target message.
4. Save the mapping by clicking **File** > **Save**.
5. Save the message flow.
6. Check for any errors in the Problems view.

Results

You have now made a qualified selection from the database.

What to do next

Now go to "Deployment of the mapping."

Related reference:

"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.

Deployment of the mapping:

How to map to the run time, creating a broker archive (BAR) file for deployment in the default execution group.

Before you begin

Before you start:

Follow the instructions in these topics:

1. "Connect to the database and obtain the definition files" on page 2321
2. "Create the message flow" on page 2323
3. "Create the mapping file" on page 2324
4. "Configure the mapping file" on page 2325

About this task

Create a broker archive (BAR) file, which contains the message flow, message set projects and the mapping file, for deployment to the run time and the default execution group.

Procedure

1. From the Broker Application Development perspective, right-click the project under the Broker Archives heading.
2. Click **New** > **Message Broker Archive**.
3. Name the BAR file (for example, AIRLINE).
4. Click **Add to**. The Add to broker archive page is displayed.
5. Select the message flow project and message set projects that are used by this flow (for example, AIRLINE_MFP,AIRLINE_MSP1, AIRLINE_MSP2) and click **OK**. The projects are added to the BAR file. A status indicator and message panel show when the process has completed.
6. Check to ensure that the required projects have been included in the BAR file.
7. Save the BAR file by clicking **File** > **Save**.
8. For deployment of the BAR file, right-click the BAR file and click **Deploy File**. The Deploy a BAR file page is displayed.
9. Select the default execution group, and click **OK**. A message in the Broker Administration message dialog box indicates successful deployment, and the deployed message flow project and message set projects are displayed in the Navigator view. A message in the Deployment log also indicates successful deployment.

Results

You have completed this scenario.

Related reference:

"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.

Scenario B: Simple message enrichment:

This scenario demonstrates simple message enrichment. Use the WebSphere Message Broker Toolkit to create message flows and message sets, and to create and deploy broker archive (BAR) files.

The scenario also involves creating a broker, and inputting instance messages that can contain MQRFH2 headers.

This scenario uses repeating instances and requires the following mapping functions:

- MRM in, MRM out (non-namespace)
- Map a simple and complex element source - target
- Map the same element source - target
- Map a different element source - target
- Map an attribute source - target
- Map a one-sided element (edit mapping)
- Map a one-sided attribute (edit mapping)
- Perform arithmetic on numeric field mapping
- Map a repeating simple element - single instance
- Map all instances of a repeating simple element
- No MQRFH2 header

The names and values used for message flows, message sets, elements and attributes, and the expressions and code samples are for illustrative purposes only.

Follow these steps to complete this scenario:

1. "Develop a message flow and message model for simple and complex element mapping" on page 2331
2. "Develop a message flow and message model for a target-only element" on page 2333
3. "Develop a message flow and message model for dealing with repeating elements" on page 2334
4. "Develop a message flow and message model for a simple message without an MQRFH2 header" on page 2336

Related tasks:

"Using message mappings" on page 2228
Message mappings define the blueprint for creating a message.

"Transforming and enriching messages" on page 2227
Transform and enrich messages by using one or more of the techniques described in this section.

Related reference:

"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.

"Mapping node" on page 4571
Use the Mapping node to construct one or more new messages and populate them with various types of information.

Develop a message flow and message model for simple and complex element mapping:
About this task

This is the first stage of the scenario to perform simple message enrichment. This topic demonstrates how to develop a message flow and message model for simple and complex element mapping, where there is the same source and target, a different source and target, or an attribute source and target. This task also involves changing field values and creating an instance document.

Procedure

1. From the Broker Application Development perspective, create the following resources:
   a. a message set project (for more details, see "Creating a message set" on page 2842).
   b. a message set called MAPPING3_SIMPLE_messages. Ensure that the message set is namespace enabled with XML wire format.
   c. a message definition file (no target namespace) called SIMPLE.
2. Create a message called addev1 that has the following structure:

```
addev1
        ssat          (xsd:string) local attribute
        ssel          (xsd:string) local element
        dsel1        (xsd:string) local element
        atel           local complex element
             latt       (xsd:string) attribute
        cel1          local complex element
             intel      (xsd:int) local element
             strel      (xsd:string) local element
        dsel2         (xsd:string) global element
        cel2           (cel2ct) global complex type
            intel        (xsd:int) local element
            fltel         (xsd:float) local element
```

3. Create a message flow project called MAPPING3_SIMPLE_flows.
4. Create a message flow called addev1 that contains the following mapping: MQInput -> Mapping -> MQOutput.
5. Open the map in the Message Mapping editor and select message addev1 as both source and target
6. Expand all levels of both messages and wire the elements as shown:

```
ssat --- ssat
ssel --- ssel
dsel1 -- dsel2
latt ---- latt
cel1 --- cel1
dsel2 -- dsel1
(cel2)
   intel ---- fltel
   fltel ---- intel
```

7. In the Spreadsheet pane, set the following expression:

```
dsel1  |  esql:upper($source/addev1/dsel2)
@latt  |   esql:upper($source/addev1/atel/@latt)
(cel2)
    intel   |  $source/addev1/cel2/fltel + 10
    fltel    |  $source/addev1/cel2/intel div 10
```

8. Create an instance document with the appropriate RFH2 header and the following data:

```
<addev1 ssatt="hello">
<ssel>this</ssel>
<dsel1>first</dsel1>
```

```
<atel latt="attrib"/>
<cel1>
<intel>2</intel>
<strel>lcomp</strel>
</cel1>
<dsel2>second</dsel2>
<cel2>
<intel>252</intel>
<fltel>3.89E+1</fltel>
</cel2>
</addev1>
```

Results

You have created the following resources:

- message set MAPPING3_SIMPLE_messages, which you have populated with message addev1
- message flow addev1 in project MAPPING3_SIMPLE_flows, which contains the mapping addev1_Mapping.msgmap
- a file that contains an instance message

What to do next

Now deploy the message set and message flow.

Deploy the message set and message flow:
About this task

This is the second stage of the scenario to perform simple message enrichment. This topic demonstrates how to deploy the message set and message flow and run the data through the broker.

Procedure

1. Create a broker archive (BAR) file called addev1.
2. Add the message set MAPPING3_SIMPLE_messages and the message flow addev1 to the BAR file.
3. Deploy the BAR file to the broker.
4. Put the instance document on the input queue.

Results

The output message looks like this:

```
<addev1 ssat="hello">
<ssel>this</ssel>
<dsel1>SECOND</dsel1>
<atel latt="ATTRIB"/>
<cel1>
<intel>2</intel>
<strel>lcomp</strel>
</cel1>
<dsel2>first</dsel2>
<cel2>
<intel>48</intel>
<fltel>2.5E+1</fltel>
</cel2>
</addev1>
```

What to do next

Now go to "Develop a message flow and message model for a target-only element."

Develop a message flow and message model for a target-only element:
Before you begin

Before you start

Perform the steps in the following topic:

1. "Develop a message flow and message model for simple and complex element mapping" on page 2331

About this task

This is the third stage of the scenario to perform simple message enrichment. This topic demonstrates how to develop a message flow and message model for a target-only element. It also involves attributing a mapping and creating an instance document.

Procedure

1. Create a message called addev2, which has the following structure:

```
addev2
    matt             (xsd:string) local attribute
    ssel              (xsd:string)  local element
    csel              local complex element
        elatt          (xsd:string)  local attribute
```

2. Create a second message called trigger, which has the following structure:

```
trigger
   start               (xsd:string) local element
```

3. Create a message flow called addev2, which contains the following mapping: MQInput -> Mapping -> MQOutput.
4. Open the map and select trigger as the source and addev2 as the target.
5. In the Spreadsheet pane, expand the target message fully and set the target fields as shown:

```
matt    |   'first attribute'
ssel     |   'string element'
elatt     |   'second attribute'
```

6. Expand the Properties folder in the Spreadsheet pane and set the following value:

```
MessageType   |   'addev2'
```

7. Create an instance document with the appropriate RFH2 header and the following data:

```
<trigger>
<start>yes</start>
</trigger>
```

Results

You have created the following resources:

- two messages called addev2 and trigger
- a message flow called addev2, which contains the mapping addev2_Mapping.msgmap

- a file that contains an instance message

What to do next

Now deploy the message set and message flow.

Deploy the message set and message flow:
About this task

This is the fourth stage of the scenario to perform simple message enrichment. This topic demonstrates how to deploy the message set and message flow and run the data through the broker.

Procedure

1. Create a broker archive (BAR) file called addev2.
2. Add the message set MAPPING3_SIMPLE_messages and the message flow addev2 to the BAR file.
3. Deploy the BAR file to the broker.
4. Put the instance document on the input queue.

Results

The output message looks like this:

```
<addev2 matt="first attribute">
<ssel>string element</ssel>
<csel elatt="second attribute"></csel>
</addev2>
```

What to do next

Now go to "Develop a message flow and message model for dealing with repeating elements."

Develop a message flow and message model for dealing with repeating elements:
Before you begin

Before you start

Perform the steps in the following topics:

1. "Develop a message flow and message model for simple and complex element mapping" on page 2331
2. "Develop a message flow and message model for a target-only element" on page 2333

About this task

This is the fifth stage of the scenario to perform simple message enrichment. This topic demonstrates how to develop a message flow and message model for dealing with repeating elements, a single instance and all instances.

Procedure

1. Create a message called addev3, which has the following structure:

```
addev3
    frepstr        (xsd:string) local element, minOcc=3, maxOcc=3
    vrepstr       (xsd:string) local element, minOcc=1, maxOcc=4
    urepstr       (xsd:string)  local element, minOcc=1, maxOcc=-1
```

2. Create a message flow called addev3, which contains the following mapping: MQInput -> Mapping -> MQOutput.
3. Open the map and select addev3 as both source and target
4. In the upper pane, map each source to the corresponding target, as illustrated in this example:

```
frepstr --- frepstr
vrepstr --- vrepstr
urepstr --- urepstr
```

5. In the Spreadsheet pane, expand fully the target addev3.
6. Highlight and delete the For item above the vrepstr entry.
7. Create an instance message with the appropriate RFH2 header and the following data:

```
<addev3>
<frepstr>this</frepstr>
<frepstr>that</frepstr>
<frepstr>other</frepstr>
<vrepstr>only one</vrepstr>
<vrepstr>extra</vrepstr>
<urepstr>first</urepstr>
<urepstr>second</urepstr>
<urepstr>third</urepstr>
<urepstr>fourth</urepstr>
<urepstr>fifth</urepstr>
</addev3>
```

Results

You have created the following resources:

- a message called addev3
- a message flow called addev3, which contains the mapping addev3_Mapping.msgmap
- a file that contains an instance message

What to do next

Now deploy the message set and message flow.

Deploy the message set and message flow:
About this task

This is the sixth stage of the scenario to perform simple message enrichment. This topic demonstrates how to deploy the message set and message flow and run the data through the broker.

Procedure

1. Create a broker archive (BAR) file called addev3.
2. Add the message set MAPPING3_SIMPLE_messages and the message flow addev3 to the BAR file.
3. Deploy the BAR file to the broker.
4. Put the instance document on the input queue.

Results

The output message looks like this:

```
<addev3>
<frepstr>this</frepstr>
<frepstr>that</frepstr>
<frepstr>other</frepstr>
<vrepstr>only one</vrepstr>
<urepstr>first</urepstr>
<urepstr>second</urepstr>
<urepstr>third</urepstr>
<urepstr>fourth</urepstr>
<urepstr>fifth</urepstr>
</addev3>
```

What to do next

Now go to "Develop a message flow and message model for a simple message without an MQRFH2 header."

Develop a message flow and message model for a simple message without an MQRFH2 header:

Before you begin

Before you start

You must complete the following tasks:

1. "Develop a message flow and message model for simple and complex element mapping" on page 2331
2. "Develop a message flow and message model for a target-only element" on page 2333
3. "Develop a message flow and message model for dealing with repeating elements" on page 2334

About this task

This is the seventh stage of the scenario to perform simple message enrichment. This topic demonstrates how to develop a message flow and message model for a simple message without an MQRFH2 header.

Procedure

1. Create a message set called MAPPING3_SIMPLE_xml. A message set project is also automatically created; this message set project has the same name as the message set that you created.
2. On the message set parameters page, set the *Message Domain* property to XML.
3. Create a message definition file called SIMPLE.
4. Create a message called addev4 that has the following structure:

```
addev4
   str1                  (xsd:string)  local element
   cel                    local complex element
       int1               (xsd:int) local element
       bool1             (xsd:boolean) local element
```

5. Create a message flow called addev4 that contains the following connected nodes: MQInput -> Mapping -> MQOutput.
6. Select the Input Message Parsing properties tab of the MQInput node, and set the *Message Domain* property to XML.

7. Open the map and select addev4 as both source and target.
8. Map the inputs to the corresponding outputs, as shown in this example:

```
str1 --- str1
int1 --- int1
bool1 --- bool1
```

9. Create an instance message that has no MQRFH2 header, but contains the following data:

```
<addev4>
<str1>this</str1>
<cel>
<int1>452</int1>
<bool1>0</bool1>
</cel>
</addev4>
```

Results

You have created the following resources:

- A message set called MAPPING3_SIMPLE_xml that contains the message addev4
- A message flow called addev4 that contains the mapping addev4_Mapping.msgmap
- A file that contains an instance message

What to do next

Now deploy the message set and message flow.

Deploy the message set and message flow:
About this task

This is the final stage of the scenario to perform simple message enrichment. This section describes how to deploy the message set and message flow, and how to the run the data through the broker.

Procedure

1. Create a broker archive (BAR) file called addev4.
2. Add the message flow called addev4 to the BAR file.
3. Deploy the BAR file to the broker.
4. Put the instance document on the input queue.

Results

The output message has the following content:

```
<addev4>
<str1>this</str1>
<cel>
<int1>452</int1>
<bool1>0</bool1>
</cel>
</addev4>
```

You have completed the scenario.

Related concepts:

"Message sets overview" on page 1162
A message set is a container for grouping messages and associated message resources (elements, types, groups).

"Broker archive" on page 3216
The unit of deployment to the broker is the *broker archive* or *BAR* file.

Related tasks:

"Creating a message set" on page 2842
Use the New Message Set wizard to create a message set.

Chapter 11, "Packaging and deploying," on page 3209
Package resources that you create in the WebSphere Message Broker Toolkit, such as message flows, and deploy them to execution groups on brokers.

Related reference:

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

Scenario C: Using a broker as auditor:

This scenario demonstrates how to use a broker as an auditor. Use the WebSphere Message Broker Toolkit to create message flows and message sets, and to create and deploy broker archive (BAR) files.

It also involves creating a broker, and sending instance messages that can contain MQRFH2 headers.

The scenario uses database updates that have been defined by using mappings. The broker receives a confirmation for a provisional booking, the message flow inserts a row into a database table representing the confirmation, updates a counter in another table representing the key of the confirmation, and deletes the provisional booking from a third table.

This scenario uses the DataDelete, DataInsert and DataUpdate nodes in the message flow, and requires the following mapping functions:

- Mapping in DataInsert node
- Combine input data into single insert
- Mapping in DataUpdate node
- Mapping in DataDelete node
- BAR file to override data source

The names and values used for message flows, message sets, elements and attributes, and the expressions and code samples are for illustrative purposes only.

Follow these steps to complete this scenario:

1. "Develop a message flow" on page 2339
2. "Deploy the message set and message flow" on page 2341
3. "Override the data source of one of the nodes" on page 2343
4. "Create a BAR file, edit the configuration, and deploy" on page 2344

Related tasks:

"Using message mappings" on page 2228
Message mappings define the blueprint for creating a message.

"Transforming and enriching messages" on page 2227
Transform and enrich messages by using one or more of the techniques described in this section.

Related reference:

"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.

"DataDelete node" on page 4382
Use the DataDelete node to interact with a database in the specified ODBC data source.

"DataInsert node" on page 4386
Use the DataInsert node to interact with a database in the specified ODBC data source.

"DataUpdate node" on page 4390
Use the DataUpdate node to interact with a database in the specified ODBC data source.

Develop a message flow:

Create a message flow to use in mapping scenario C, which shows how to use your broker as an auditor.

About this task

This step is the first task in creating "Scenario C: Using a broker as auditor" on page 2338. Develop a message flow to map several fields of input data into a single insert record for a database. See also how to update another table, delete a third table, and develop corresponding message models and instance messages.

Procedure

1. Create a database called MAPDB, and a table called CONFIRMATION, which contains the following columns:

   ```
   RESID                INTEGER
   ```

2. Populate the CONFIRMATION table with the value shown:

   ```
   9052
   ```

3. Create another table called RESERVATION, which contains the following columns:

   ```
   RESID           INTEGER
   NAME            VARCHAR(20)
   PARTY           INTEGER
   PAYMENT         DECIMAL(8,2)
   ```

4. Populate the RESERVATION table with the values shown:

   ```
   8214,'ARCHIBALD',2,0.0
   2618,'HENRY', 4, 120.0
   9052,'THAW', 3, 85.0
   ```

5. Create another table called PROVISIONAL, which contains the following columns:

   ```
   RESID                   INTEGER
   ```

6. Populate the PROVISIONAL table with the values shown:

   ```
    8214 2618
   ```

7. Create a Windows ODBC Data Source Name for the database, then create a definition for the database in the workbench by clicking **File** > **New** > **Database Definition File**. For information about how to complete this step,

including how to choose a supported database and version, follow the instructions provided in "Adding database definitions to the WebSphere Message Broker Toolkit" on page 2278.

8. Create a message set project and a message set called MAPPING3_AUDIT_messages (ensuring that the message set is namespace enabled, with XML wire format) and create a message definition file called AUDIT.
9. Create a message called addev1, which has the structure:

```
addev1
   id                (xsd:int) local element
   status            (xsd:string) local element
   name              (xsd:string) local element
   size              (xsd:int) local element
   payment           (xsd:decimal) local element
```

10. Create a message flow project called MAPPING3_AUDIT_flows.
11. Create a message flow called addev1, which contains the following structure: MQInput ->DataInsert -> DataUpdate -> DataDelete -> MQOutput.
12. For the DataDelete node, set the Data Source property to MAPDB.
13. Open the mapping for the DataInsert node and select MAPPING3_AUDIT_messages addev1 as the source, and MAPDB.SCHEMA.CONFIRMATION as the target.
14. Wire the source to the target as shown:

```
addev1             MAPDB
       id -------------- RESID
```

15. For the DataUpdate node, set the Data Source property to MAPDB.
16. Open the mapping for the DataUpdate node and select MAPPING3_AUDIT_messages addev1 as the source, and MAPDB.SCHEMA.RESERVATION as the target.
17. Wire the source to the target as shown:

```
addev1             MAPDB
       id -------------- RESID
       name ---------- NAME
       size ------------ PARTY
       payment ------- PAYMENT
```

18. In the Message Mapping editor Spreadsheet pane, select $db:update and change fn:true() to $db:update.MAPDB.MQSI.RESERVATION.RESID = $source/addev1/id and $source/addev1/status = 'CONFIRM'.
19. For the DataDelete node, set the Data Source property to MAPDB.
20. Open the mapping for the DataDelete node and select MAPPING3_AUDIT_messages addev1 as the source, and MAPDB.SCHEMA.PROVISIONAL as the target.
21. In the Message Mapping editor Spreadsheet pane, select $db:delete and change fn:false() to $db:delete.MAPDB.MQSI.PROVISIONAL.RESID = $source/addev1/id.
22. Create the following instance message with appropriate MQRFH2 headers:

```
<addev1>
<id>8214</id>
<status>CONFIRM</status>
<name>ARCHIBALD</name>
<size>2</size>
<payment>1038.0</payment>
</addev1>
```

Results

You have created the following resources:

- A message set called MAPPING3_AUDIT_messages, which is populated with the message addev1
- A message flow called addev1 in project MAPPING3_AUDIT_flows, which contains the mapping files `addev1_DataInsert.msgmap`, `addev1_DataUpdate.msgmap`, and `addev1_DataDelete.msgmap`
- The database MAPDB with populated tables CONFIRMATION, RESERVATION, and PROVISIONAL
- A file that contains an instance message for test.

What to do next

Next: Continue with the next step, "Deploy the message set and message flow."

Related concepts:

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

Related tasks:

"Adding database definitions to the WebSphere Message Broker Toolkit" on page 2278
Use the New Database Definition File wizard to add database definitions to the WebSphere Message Broker Toolkit.

"Message mapping scenarios" on page 2318

Related reference:

"Scenario C: Using a broker as auditor" on page 2338
This scenario demonstrates how to use a broker as an auditor. Use the WebSphere Message Broker Toolkit to create message flows and message sets, and to create and deploy broker archive (BAR) files.

"Supported databases" on page 3591
You can optionally configure databases to contain data that is accessed by your message flows. Databases from IBM and other suppliers are supported at specific versions on supported operating systems.

"Mapping node" on page 4571
Use the Mapping node to construct one or more new messages and populate them with various types of information.

Deploy the message set and message flow:

Deploy the message flow that you have created to support mapping scenario C, which shows how to use your broker as an auditor.

Before you begin

Before you start

Perform the first step for this scenario, described in "Develop a message flow" on page 2339.

About this task

This step is the second task in creating "Scenario C: Using a broker as auditor" on page 2338. Deploy the message set and message flow, and run the instance messages through the broker.

Procedure

1. Create a BAR file called `addev1`.
2. Add the message set MAPPING3_AUDIT_messages, and the message flow addev1, to the BAR file.
3. Deploy the BAR file to the broker.
4. Put the instance document on the input queue.

Results

The output messages are the same as the input messages. The database table has the following content:

```
CONFIRMATION
RESID
-----------
       9052
       8214

RESERVATION
RESID       NAME                 PARTY       PAYMENT
----------- -------------------- ----------- ----------
       8214 ARCHIBALD                      2    1038.00
       2618 HENRY                          4     120.00
       9052 THAW                           3      85.00

PROVISIONAL
RESID
-----------
       2618
```

What to do next

Next: Continue with the next step, "Override the data source of one of the nodes" on page 2343.

Related concepts:

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

Related tasks:

"Message mapping scenarios" on page 2318

Related reference:

"Scenario C: Using a broker as auditor" on page 2338
This scenario demonstrates how to use a broker as an auditor. Use the WebSphere Message Broker Toolkit to create message flows and message sets, and to create and deploy broker archive (BAR) files.

"Mapping node" on page 4571
Use the Mapping node to construct one or more new messages and populate them with various types of information.

Override the data source of one of the nodes:

Override the data source property for a node in the message flow that you have created to support mapping scenario C, which shows how to use your broker as an auditor.

Before you begin

Before you start

Complete the first two steps for this scenario:

1. "Develop a message flow" on page 2339
2. "Deploy the message set and message flow" on page 2341

About this task

This step is the third task in creating "Scenario C: Using a broker as auditor" on page 2338. Override the data source of one of the nodes by changing the configuration of its broker archive (BAR) file.

Procedure

1. Create a database called ALTDB, and a table called CONFIRMATION, which contains the following columns:

   ```
   RESID                INTEGER
   ```

2. Create a Windows ODBC Data Source Name for the database, then register the database in the WebSphere Message Broker Toolkit by clicking **File** > **New** > **Database Definition File**. For information about how to complete this step, including how to choose a supported database and version, follow the instructions provided in "Adding database definitions to the WebSphere Message Broker Toolkit" on page 2278.

Results

You have created a database called ALTDB with a table called CONFIRMATION.

What to do next

Next: Continue with the next step, "Create a BAR file, edit the configuration, and deploy" on page 2344.

Related concepts:

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

Related tasks:

"Adding database definitions to the WebSphere Message Broker Toolkit" on page 2278
Use the New Database Definition File wizard to add database definitions to the WebSphere Message Broker Toolkit.

"Message mapping scenarios" on page 2318

Related reference:

"Scenario C: Using a broker as auditor" on page 2338
This scenario demonstrates how to use a broker as an auditor. Use the WebSphere Message Broker Toolkit to create message flows and message sets, and to create and deploy broker archive (BAR) files.

"Supported databases" on page 3591
You can optionally configure databases to contain data that is accessed by your message flows. Databases from IBM and other suppliers are supported at specific versions on supported operating systems.

"Mapping node" on page 4571
Use the Mapping node to construct one or more new messages and populate them with various types of information.

Create a BAR file, edit the configuration, and deploy:

Create a broker archive file and deploy the message flow for mapping scenario C, which shows how to use your broker as an auditor.

Before you begin

Before you start

Perform the first three steps for this scenario:

1. "Develop a message flow" on page 2339
2. "Deploy the message set and message flow" on page 2341
3. "Override the data source of one of the nodes" on page 2343

About this task

This step is the fourth and last task in creating "Scenario C: Using a broker as auditor" on page 2338. Learn how to create a broker archive (BAR) file, edit the configuration, and deploy.

Procedure

1. Add the message flow addev1 to the BAR file again.
2. Select the Manage and Configure tab of the BAR file editor.
3. Expand the message flow, and click the DataInsert icon.
4. In the Properties view, change the Data Source field from MAPDB to ALTDB, and save the BAR file.
5. Deploy the BAR file to the broker.
6. Put the instance document on the input queue.

Results

The output message is the same as the input. In the ALTDB database, the table has the following content:

```
CONFIRMATION
RESID
-----------
       8214
```

What to do next

Next: You have now completed scenario C. If you want more information about how to use maps, return to "Message mapping scenarios" on page 2318, and explore another scenario.

Related concepts:

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

Related tasks:

"Message mapping scenarios" on page 2318

Related reference:

"Scenario C: Using a broker as auditor" on page 2338
This scenario demonstrates how to use a broker as an auditor. Use the WebSphere Message Broker Toolkit to create message flows and message sets, and to create and deploy broker archive (BAR) files.

"Mapping node" on page 4571
Use the Mapping node to construct one or more new messages and populate them with various types of information.

Scenario D: Complex message enrichment:

This scenario demonstrates complex message enrichment and uses complex message manipulation. Use the WebSphere Message Broker Toolkit to create message flows and message sets, and to create and deploy broker archive (BAR) files.

The scenario also involves creating a broker, and putting instance messages that can contain MQRFH2 headers to input queues.

This scenario requires the following mapping functions:

- MRM in, MRM out (namespace)
- Other nodes required to complete message
- Conditional mapping
- CASE mapping (both syntax formats)
- If/condition
- Combining multiple source fields into a single target field (inter namespace)
- Nested repeating complex and simple elements
- Target data derived from database
- String, numeric, datetime functions
- User-defined ESQL procedures and functions
- User-defined Java routines

The names and values used for message flows, message sets, elements, and attributes, and the expressions and code samples, are for illustrative purposes only.

Follow these steps to complete this scenario:

1. "Develop a message flow that contains other nodes" on page 2346
2. "Develop a message flow to map target fields from multiple other fields" on page 2349
3. "Develop a message flow and message model for mapping a complex nested, repeating message" on page 2352
4. "Develop a message flow for populating a target from a database" on page 2358
5. "Develop a message flow using a user-defined ESQL function" on page 2361
6. "Develop a message flow using a user-defined Java procedure" on page 2364

Related tasks:

"Using message mappings" on page 2228
Message mappings define the blueprint for creating a message.

"Transforming and enriching messages" on page 2227
Transform and enrich messages by using one or more of the techniques described in this section.

Related reference:

"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.

"Mapping node" on page 4571
Use the Mapping node to construct one or more new messages and populate them with various types of information.

Develop a message flow that contains other nodes:

Create a message flow to use in mapping scenario D, which describes complex message enrichment.

About this task

This step is the first task in creating "Scenario D: Complex message enrichment" on page 2345. Learn how to complete the following procedures:

- Develop a message flow that contains other nodes (for example, a Filter node)
- Create mappings that use conditions
- Develop corresponding message models, which use all main data types, and instance messages

Procedure

1. Switch to the Broker Application Development perspective
2. Create the following resources:
   - A message set project and a message set called MAPPING3_COMPLEX_messages, ensuring that the message set is namespace enabled with XML wire format
   - A message definition file called COMPLEX, which has a target namespace www.complex.net, with prefix comp
3. Create messages addev1, addev1s and addev1n with the following structures:

```
addev1
   bool      (xsd:boolean) local element
   bin        (xsd:hexBinary) local element
   dat        (xsd:dateTime) local element
   dec       (xsd:decimal) local element
   dur        (xsd:duration) local element
   flt          (xsd:float) local element
   int         (xsd:int) local element
   str         (xsd:string) local element
addev1s
   bin        (xsd:hexBinary) local element
   dat        (xsd:dateTime) local element
   dur        (xsd:duration) local element
   str         (xsd:string) local element
addev1n
   dec       (xsd:decimal) local element
   flt          (xsd:float) local element
   int         (xsd:int) local element
```

4. Create a message flow project called MAPPING3_COMPLEX_flows.
5. Create a message flow called addev1 which contains the following structure:

```
MQInput -> Filter -> Mapping -> Compute
                  \                       \ --> RCD -> MQOutput
                   \-> Mapping1-----------/
```

6. In the Filter node, set the following ESQL:

```
IF Body.bool THEN
            RETURN TRUE;
        ELSE
            RETURN FALSE;
        END IF;
```

7. In the Mapping node that is connected to the Filter node true terminal (Mapping1), open the map and select addev1 as source and addev1s as target.
8. Wire the source to target as shown:

```
bin --- bin
dat --- dat
dur --- dur
str --- str
```

9. In the Message Mapping editor Spreadsheet pane, expand Properties and set the following values:

```
MessageType   |     'addev1s'
```

10. Right-click the target dat and click **If**.
11. Replace the condition fn:true() with `$source/comp:addev1/str = 'dat'`.
12. Set the value for dat to `$source/comp:addev1/dat + xs:duration("P3M")`.
13. Right-click the condition and click **Else**.
14. Right-click the target dur and click **If**.
15. Replace the condition fn:true() with `$source/comp:addev1/str = 'dur'`.
16. Set the value for dur to `$source/comp:addev1/dur + xs:duration("P1Y")`.
17. Right-click the condition and click **Else**.
18. Open the map for the node that is connected to the false terminal (Mapping) of the Filter node and select addev1 as source and addev1n as target.
19. Wire the source to target as shown:

```
dec --- dec
flt --- flt
int --- int
```

20. In the Message Mapping editor Spreadsheet pane, expand Properties and set the following values:

```
MessageType     |     'addev1n'
```

21. Set the ESQL in the Compute node to:

```
CALL CopyMessageHeaders();
        SET OutputRoot.MRM.dec = InputBody.dec * 10;
        SET OutputRoot.MRM.flt = InputBody.flt * 10;
        SET OutputRoot.MRM.int = InputBody.int * 10;
```

22. In the ResetContentDescriptor node, set the Message Domain to XMLNS and select **Reset Message Domain**.
23. Create three instance messages with the appropriate MQRFH2 headers:

```
<comp:addev1 xmlns:comp="http://www.complex.net">
<bool>1</bool>
<bin><![CDATA[010203]]></bin>
<dat>2005-05-06T00:00:00+00:00</dat>
<dec>19.34</dec>
<dur>P2Y4M</dur>
<flt>3.245E+2</flt>
<int>2104</int>
<str>dat</str>
</comp:addev1>
```

```
<comp:addev1 xmlns:comp="http://www.complex.net">
<bool>1</bool>
<bin><![CDATA[010203]]></bin>
<dat>2005-05-06T00:00:00+00:00</dat>
<dec>19.34</dec>
<dur>P2Y4M</dur>
<flt>3.245E+2</flt>
<int>2104</int>
<str>dur</str>
</comp:addev1>
```

```
<comp:addev1 xmlns:comp="http://www.complex.net">
<bool>0</bool>
<bin><![CDATA[010203]]></bin>
<dat>2005-05-06T00:00:00+00:00</dat>
<dec>19.34</dec>
<dur>P2Y4M</dur>
<flt>3.245E+2</flt>
<int>2104</int>
<str>dat</str>
</comp:addev1>
```

Results

You have created the following resources:

- A message set called MAPPING3_COMPLEX_messages, which is populated with the messages addev1, addev1s and addev1n
- A message flow called addev1 in the project MAPPING3_COMPLEX_flows, which contains the mapping files `addev1_Mapping.msgmap` and `addev1._Mapping1.msgmap`
- Files that contain instance messages for test

What to do next

Now deploy the message set and message flow.

Deploy the message set and message flow:
About this task

This step is the second task in creating "Scenario D: Complex message enrichment" on page 2345. Deploy the message set and message flow and run the instance messages through the broker.

Procedure

1. Create a BAR file called addev1.
2. Add the message set MAPPING3_COMPLEX_messages and the message flow addev1 to the BAR file.
3. Deploy the BAR file to the broker.
4. Put the instance documents on the input queue.

Results

The output messages have the following content:

```
<comp:addev1s xmlns:comp="http://www.complex.net">
<bin><![CDATA[010203]]></bin>
<dat>2005-08-06T00:00:00-01:00</dat>
<dur>P2Y4M</dur>
<str>dat</str>
</comp:addev1s>
```

What to do next

Next: Continue with the next step, "Develop a message flow to map target fields from multiple other fields."

Related concepts:

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

Related tasks:

"Message mapping scenarios" on page 2318

Related reference:

"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.

"Scenario D: Complex message enrichment" on page 2345
This scenario demonstrates complex message enrichment and uses complex message manipulation. Use the WebSphere Message Broker Toolkit to create message flows and message sets, and to create and deploy broker archive (BAR) files.

"Mapping node" on page 4571
Use the Mapping node to construct one or more new messages and populate them with various types of information.

Develop a message flow to map target fields from multiple other fields:

Develop a message flow to map target fields from multiple other fields, and develop corresponding message models and instance documents to use in mapping scenario D, which describes complex message enrichment.

Before you begin

Before you start

Perform the first two tasks for this scenario, described in step "Develop a message flow that contains other nodes" on page 2346.

About this task

This step is the third task in creating "Scenario D: Complex message enrichment" on page 2345.

Procedure

1. In the COMPLEX message definition, in namespace www.complex.net, create a message called addev2, which has the following structure:

```
addev2
   firstname            (xsd:string) local element
   lastname             (xsd:string) local element
   branch                (xsd:string) local element
   accountno           (xsd:string) local element
   balance               (xsd:decimal) local element
   transvalue            local complex element, base type xsd:decimal
      transdir            (xsd:string) local attribute
```

2. In the message set MAPPING3_COMPLEX_messages, create a new message definition file called COMP2, which has the target namespace www.comp2.net, with prefix c2.

3. In the COMP2 message definition, create a message called addev2out, which has the structure:

```
addev2out
    accountdetails      (xsd:string) local element
    transvalue            (xsd:decimal) local element
    balance               (xsd:decimal) local element
```

4. Create a message flow called addev2, which contains the following structure: MQInput -> Mapping -> MQOutput.
5. Open the map for the Mapping node, and select addev2 as the source and addev2out as the target.
6. Wire the source to target as shown:

```
accountno --- accountdetails
balance --- balance
transvalue --- transvalue
```

7. In the Message Mapping editor Spreadsheet pane, expand Properties and set the following values:

```
MessageType   |     'addev2out'
```

8. Set the accountdetails target to `fn:concat($source/comp:addev2/accountno, $source/comp:addev2/branch, $source/comp:addev2/lastname, $source/comp:addev2/firstname)`.
9. Right-click the target transvalue and click **If**.
10. Change the value of the `if` statement from fn:true() to `$source/comp:addev2/transvalue/@transdir = 'DEBIT'`.
11. Select transvalue and set its value to `$source/comp:addev2/transvalue * (-1)`.
12. Right-click the `if` statement and click **Else**.
13. Right-click the target balance and click **If**.
14. Change the value of the `if` statement from fn:true() to `$source/comp:addev2/transvalue/@transdir = 'DEBIT'`.
15. Select balance and set its value to `$source/comp:addev2/balance - $source/comp:addev2/transvalue`.
16. Right-click the `if` statement and click **Else If**.
17. Change the value of the `elseif` statement from fn:true() to `$source/comp:addev2/transvalue/@transdir = 'CREDIT'`.
18. Select balance following the second condition statement and set its Value to `$source/comp:addev2/balance + $source/comp:addev2/transvalue`.
19. Create two instance messages with the appropriate MQRFH2 headers:

```
<comp:addev2 xmlns:comp="http://www.complex.net">
<firstname>Brian</firstname>
<lastname>Benn</lastname>
<branch>52-84-02</branch>
<accountno>567432876543</accountno>
<balance>1543.56</balance>
<transvalue transdir="DEBIT">25.28</transvalue>
</comp:addev2>
```

```
<comp:addev2 xmlns:comp="http://www.complex.net">
<firstname>Brian</firstname>
<lastname>Benn</lastname>
<branch>52-84-02</branch>
<accountno>567432876543</accountno>
<balance>1543.56</balance>
<transvalue transdir="CREDIT">25.28</transvalue>
</comp:addev2>
```

Results

You have created the following resources:

- A message called addev2 in the message definition called COMPLEX
- A message called addev2out in the message definition called COMP2
- A message flow called addev2, which contains the mapping file `addev2_Mapping.msgmap`
- Files that contain instance messages for test

What to do next

Now deploy the message set and message flow

Deploy the message set and message flow:
About this task

This step is the fourth task in creating "Scenario D: Complex message enrichment" on page 2345. Deploy the message set and message flow and run the instance messages through the broker.

Procedure

1. Create a BAR file called addev2.
2. Add the message set MAPPING3_COMPLEX_messages and the message flow addev2 to the BAR file.
3. Deploy the BAR file to the broker.
4. Put the instance documents on the input queue.

Results

The output messages have the following content:

```
<c2:addev2out xmlns:c2="http://www.comp2.net"  xmlns:comp="http://www.complex.net">
<accountdetails>567432876543 52-84-02 Benn Brian</accountdetails>
<transvalue>-25.28</transvalue>
<balance>1518.28</balance>
</c2:addev2out>
```

What to do next

Next: Continue with the next step, "Develop a message flow and message model for mapping a complex nested, repeating message" on page 2352.

Related concepts:

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

Related tasks:

"Message mapping scenarios" on page 2318

Related reference:

"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.

"Scenario D: Complex message enrichment" on page 2345
This scenario demonstrates complex message enrichment and uses complex message manipulation. Use the WebSphere Message Broker Toolkit to create message flows and message sets, and to create and deploy broker archive (BAR) files.

"Mapping node" on page 4571
Use the Mapping node to construct one or more new messages and populate them with various types of information.

Develop a message flow and message model for mapping a complex nested, repeating message:

Develop a message flow and message model for mapping a complex nested, repeating message, and develop corresponding instance documents, to use in mapping scenario D, which describes complex message enrichment.

Before you begin

Before you start

Perform the previous tasks for this scenario, described in the following steps:

1. "Develop a message flow that contains other nodes" on page 2346
2. "Develop a message flow to map target fields from multiple other fields" on page 2349

About this task

This step is the fifth task in creating "Scenario D: Complex message enrichment" on page 2345.

Procedure

1. In the COMPLEX message definition, in namespace www.complex.net, create a message called addev3, which has the following structure:

```
addev3
 choice
    sstr                    (xsd:string) local element
    intrep                 (xsd:int) local element, minOcc=2, maxOcc=6
    dur                     (xsd:duration) local element
 choice
    comp1               local complex element
       dat1                (xsd:date) local element
       sval                (xsd:string) local element
    comp2               local complex element
       bool1              (xsd:boolean) local element
       dat2                (xsd:date) local element
    comprep            local complex element, minOcc=1, maxOcc=4
       int1                (xsd:int) local element
       dec1               (xsd:decimal) local element
 binel                     (xsd:hexBinary) local element
 lelem                     local complex element, base type xsd:string
    latt                    (xsd:int) local attribute
 lcomp                    local complex element
    head                    (xsd:string) local element
    incomp                local complex element
       count                (xsd:int) local element
       comp:gcompel   global complex element, minOcc=0, maxOcc=-1
          fstr                (xsd:string) local element
          multel             local complex element
             in1              (xsd:boolean) local element
             in2              (xsd:string) local element
             in3              (xsd:float) local element
    footer                  (xsd:string) local element
 repstr                     (xsd:string) local element, minOcc=1, maxOcc=-1
```

2. Create a message flow called addev3, which contains the following structure: MQInput -> Mapping -> MQOutput.

3. Open the map for the Mapping node, and select addev3 as the source and target.
4. Map each source element to its corresponding target element:

```
sstr --- sstr
intrep --- intrep
dur --- dur
dat1 --- dat1
sval --- sval
bool1 --- bool1
dat2 --- dat2
int1 --- int1
dec1 --- dec1
binel --- binel
lelem --- lelem
latt --- latt
head --- head
count --- count
fstr --- fstr
multel --- multel
footer --- footer
repstr --- repstr
```

5. In the Message Mapping editor Spreadsheet pane, for the `if` statement, change fn:true() to `fn:exists($source/comp:addev3/sstr)`.
6. For the `elseif` statement, change fn:true() to `fn:exists($source/comp:addev3/intrep)`.
7. For the second `elseif` statement, change fn:true() to `fn:exists($source/comp:addev3/dur)`.
8. For the first complex choice `if` statement, change fn:true() to `fn:exists($source/comp:addev3/comp1)`.
9. For the second complex choice `elseif` statement, change fn:true() to `fn:exists($source/comp:addev3/comp2)`.
10. For the third complex choice `elseif` statement, change fn:true() to `fn:exists($source/comp:addev3/comprep)`.
11. Create the following instance messages, with appropriate MQRFH2 headers:

```
<comp:addev3 xmlns:comp="http://www.complex.net">
<sstr>first</sstr>
<comp1>
<dat1>2005-06-24</dat1>
<sval>date value</sval>
</comp1>
<binel><![CDATA[3132333435]]></binel>
<lelem latt="24">twenty four</lelem>
<lcomp>
<head>nesting start</head>
<incomp>
<count>3</count>
<comp:gcompel>
<fstr>first</fstr>
<multel>
<in1>1</in1>
<in2>C</in2>
<in3>2.45E+1</in3>
</multel>
</comp:gcompel>
<comp:gcompel>
<fstr>second</fstr>
<multel>
<in1>1</in1>
<in2>D</in2>
<in3>7.625E+3</in3>
</multel>
```

```
</comp:gcompel>
<comp:gcompel>
<fstr>third</fstr>
<multel>
<in1>0</in1>
<in2>C</in2>
<in3>4.9E+0</in3>
</multel>
</comp:gcompel>
</incomp>
<footer>nesting end</footer>
</lcomp>
<repstr>abc</repstr>
<repstr>def</repstr>
<repstr>ghi</repstr>
<repstr>jkl</repstr>
<repstr>mno</repstr>
</comp:addev3>
<comp:addev3 xmlns:comp="http://www.complex.net">
<intrep>45</intrep>
<intrep>12</intrep>
<intrep>920</intrep>
<comp2>
<bool1>1</bool1>
<dat2>2005-06-24</dat2>
</comp2>
<binel><![CDATA[3132333435]]></binel>
<lelem latt="24">twenty four</lelem>
<lcomp>
<head>nesting start</head>
<incomp>
<count>5</count>
<comp:gcompel>
<fstr>first</fstr>
<multel>
<in1>1</in1>
<in2>C</in2>
<in3>2.45E+1</in3>
</multel>
</comp:gcompel>
<comp:gcompel>
<fstr>second</fstr>
<multel>
<in1>1</in1>
<in2>D</in2>
<in3>7.625E+3</in3>
</multel>
</comp:gcompel>
<comp:gcompel>
<fstr>third</fstr>
<multel>
<in1>0</in1>
<in2>C</in2>
<in3>4.9E+0</in3>
</multel>
</comp:gcompel>
<comp:gcompel>
<fstr>fourth</fstr>
<multel>
<in1>1</in1>
<in2>F</in2>
<in3>2.98E+1</in3>
</multel>
</comp:gcompel>
<comp:gcompel>
<fstr>fifth</fstr>
<multel>
```

```
<in1>0</in1>
<in2>D</in2>
<in3>8.57E-2</in3>
</multel>
</comp:gcompel>
</incomp>
<footer>nesting end</footer>
</lcomp>
<repstr>abc</repstr>
</comp:addev3>
```

```
<comp:addev3 xmlns:comp="http://www.complex.net">
<dur>P2Y2M</dur>
<comp3>
<int1>6</int1>
<dec1>2821.54</dec1>
</comp3>
<comp3>
<int1>41</int1>
<dec1>0.02</dec1>
</comp3>
<binel><![CDATA[3132333435]]></binel>
<lelem latt="24">twenty four</lelem>
<lcomp>
<head>nesting start</head>
<incomp>
<count>0</count>
</incomp>
<footer>nesting end</footer>
</lcomp>
<repstr>abc</repstr>
<repstr>def</repstr>
<repstr>ghi</repstr>
<repstr>jkl</repstr>
<repstr>mno</repstr>
<repstr>pqr</repstr>
<repstr>stu</repstr>
<repstr>vwx</repstr>
</comp:addev3>
```

Results

You have created the following resources:

- A message called addev3 in the message definition COMPLEX
- A message flow called addev3, which contains the mapping file `addev3_Mapping.msgmap`
- Files that contain instance messages for test

What to do next

Now deploy the message set and message flow.

Deploy the message set and message flow:
About this task

This step is the sixth task in creating "Scenario D: Complex message enrichment" on page 2345. Learn how to deploy the message set and message flow and run the instance messages through the broker.

Procedure

1. Create a BAR file called addev3.

2. Add the message set MAPPING3_COMPLEX_messages and the message flow addev3 to the BAR file.
3. Deploy the BAR file to the broker.
4. Put the instance documents on the input queue.

Results

The output messages look like this:

```
<comp:addev3 xmlns:comp="http://www.complex.net">
<sstr>first</sstr>
<comp1>
<dat1>2005-06-24</dat1>
<sval>date value</sval>
</comp1>
<binel><![CDATA[3132333435]]></binel>
<lelem latt="24">twenty four</lelem>
<lcomp>
<head>nesting start</head>
<incomp>
<count>3</count>
<comp:gcompel>
<fstr>first</fstr>
<multel>
<in1>1</in1>
<in2>C</in2>
<in3>2.45E+1</in3>
</multel>
</comp:gcompel>
<comp:gcompel>
<fstr>second</fstr>
<multel>
<in1>1</in1>
<in2>D</in2>
<in3>7.625E+3</in3>
</multel>
</comp:gcompel>
<comp:gcompel>
<fstr>third</fstr>
<multel>
<in1>0</in1>
<in2>C</in2>
<in3>4.9E+0</in3>
</multel>
</comp:gcompel>
</incomp>
<footer>nesting end</footer>
</lcomp>
<repstr>abc</repstr>
<repstr>def</repstr>
<repstr>ghi</repstr>
<repstr>jkl</repstr>
<repstr>mno</repstr>
</comp:addev3>
```

```
<comp:addev3 xmlns:comp="http://www.complex.net">
<intrep>45</intrep>
<intrep>12</intrep>
<intrep>920</intrep>
<comp2>
<bool1>1</bool1>
<dat2>2005-06-24</dat2>
</comp2>
<binel><![CDATA[3132333435]]></binel>
<lelem latt="24">twenty four</lelem>
<lcomp>
```

```
<head>nesting start</head>
<incomp>
<count>5</count>
<comp:gcompel>
<fstr>first</fstr>
<multel>
<in1>1</in1>
<in2>C</in2>
<in3>2.45E+1</in3>
</multel>
</comp:gcompel>
<comp:gcompel>
<fstr>second</fstr>
<multel>
<in1>1</in1>
<in2>D</in2>
<in3>7.625E+3</in3>
</multel>
</comp:gcompel>
<comp:gcompel>
<fstr>third</fstr>
<multel>
<in1>0</in1>
<in2>C</in2>
<in3>4.9E+0</in3>
</multel>
</comp:gcompel>
<comp:gcompel>
<fstr>fourth</fstr>
<multel>
<in1>1</in1>
<in2>F</in2>
<in3>2.98E+1</in3>
</multel>
</comp:gcompel>
<comp:gcompel>
<fstr>fifth</fstr>
<multel>
<in1>0</in1>
<in2>D</in2>
<in3>8.57E-2</in3>
</multel>
</comp:gcompel>
</incomp>
<footer>nesting end</footer>
</lcomp>
<repstr>abc</repstr>
</comp:addev3>
<comp:addev3 xmlns:comp="http://www.complex.net">
<dur>P2Y2M</dur>
<comp3>
<int1>6</int1>
<dec1>2821.54</dec1>
</comp3>
<comp3>
<int1>41</int1>
<dec1>0.02</dec1>
</comp3>
<binel><![CDATA[3132333435]]></binel>
<lelem latt="24">twenty four</lelem>
<lcomp>
<head>nesting start</head>
<incomp>
<count>0</count>
</incomp>
<footer>nesting end</footer>
</lcomp>
```

```
<repstr>abc</repstr>
<repstr>def</repstr>
<repstr>ghi</repstr>
<repstr>jkl</repstr>
<repstr>mno</repstr>
<repstr>pqr</repstr>
<repstr>stu</repstr>
<repstr>vwx</repstr>
</comp:addev3>
```

What to do next

Next: Continue with the next step, "Develop a message flow for populating a target from a database."

Related concepts:

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

Related tasks:

"Message mapping scenarios" on page 2318

Related reference:

"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.

"Scenario D: Complex message enrichment" on page 2345
This scenario demonstrates complex message enrichment and uses complex message manipulation. Use the WebSphere Message Broker Toolkit to create message flows and message sets, and to create and deploy broker archive (BAR) files.

"Mapping node" on page 4571
Use the Mapping node to construct one or more new messages and populate them with various types of information.

Develop a message flow for populating a target from a database:

Develop a message flow that updates a message from a database for mapping scenario D, complex message enrichment.

Before you begin

Before you start

Perform the previous tasks for this scenario, described in the following steps:

1. "Develop a message flow that contains other nodes" on page 2346
2. "Develop a message flow to map target fields from multiple other fields" on page 2349
3. "Develop a message flow and message model for mapping a complex nested, repeating message" on page 2352

About this task

This step is the seventh task in creating "Scenario D: Complex message enrichment" on page 2345. Develop a message flow for populating a target from a database, and create a corresponding message model and instance documents.

Procedure

1. Create a database called MAPDB and create a table called TRANSACTION, which has the following columns:

```
ACCOUNT        VARCHAR(12)
TDATE               DATE
VALUE              DECIMAL(8,2)
```

2. Populate the database with the values shown:

```
'12345678901', '2005-04-25', -14.25
'12345678901', '2005-04-25', 100.00
'12345678901','2005-05-15', 2891.30
'12345678901','2005-06-11', -215.28
```

3. Create a Windows ODBC Data Source Name for the database.
4. Add a definition for the database to the workbench by clicking **File** > **New** > **Database Definition File**. For information about how to complete this step, including how to choose a supported database and version, follow the instructions provided in "Adding database definitions to the WebSphere Message Broker Toolkit" on page 2278.
5. In the COMPLEX message definition, in namespace www.complex.net, create a message called addev4in, which has the following structure:

```
addev4in
   account             (xsd:string) local element
   tdate                 (xsd:date) local element
```

6. In the COMP2 message definition, in namespace www.comp2.net, create a message called addev4out, which has the following structure:

```
addev4out
   account             (xsd:string) local element
   tdate                 (xsd:date) local element
   value                 (xsd:decimal) local element, minOcc=0, maxOcc=-1
```

7. Create a message flow called addev4, which contains the following structure: MQInput -> Mapping -> MQOutput.
8. Open the map for the Mapping node, and select addev4in as the source and addev4out as the target.
9. Map the input to outputs as shown:

```
account --- account
tdate --- tdate
```

10. In the Message Mapping editor Spreadsheet pane, right-click the target value, and click **Select Data Source**.
11. Select MAPDB, and click **Finish**.
12. In the top pane, expand the MAPDB tree and wire the values as shown:

```
VALUE  --- value
```

13. In the Spreadsheet pane, select the target $db:select and change fn:true() to: `$db:select.MAPDB.SCHEMA.TRANSACTION.ACCOUNT=$source/comp:addev4in/account` and `$db:select.MAPDB.SCHEMA.TRANSACTION.TDATE=$source/comp:addev4in/tdate`
14. Expand the Properties tree and set the following values:

```
MessageType     |    'addev4out'
```

15. Set the data source property for the Mapping node to MAPDB.
16. Create the following instance messages with appropriate MQRFH2 headers:

```
<comp:addev4in xmlns:comp="http://www.complex.net">
<account>12345678901</account>
<tdate>2005-05-15</tdate>
</comp:addev4in>
```

```
<comp:addev4in xmlns:comp="http://www.complex.net">
<account>12345678901</account>
<tdate>2005-04-25</tdate>
</comp:addev4in>
```

Results

You have created the following resources:

- A message called addev4in in a message definition called COMPLEX
- A message called addev4out in a message definition called COMP
- A message flow called addev4, which contains the mapping file `addev4_Mapping.msgmap`
- Files that contain instance messages

What to do next

Now deploy the message set and message flow.

Deploy the message set and message flow:
About this task

This step is the eighth task in creating "Scenario D: Complex message enrichment" on page 2345. Deploy the message set and message flow and run the instance messages through the broker.

Procedure

1. Create a BAR file called addev4.
2. Add the message set MAPPING3_COMPLEX_messages and the message flow addev4 to the BAR file.
3. Deploy the BAR file to the broker.
4. Put the instance documents on the input queue.

Results

The output messages have the following content:

```
<c2:addev4out xmlns:c2="http://www.comp2.net" xmlns:comp="http://www.complex.net" >
<account>12345678901</account>
<tdate>2005-05-15</tdate>
<value>2891.3</value>
</c2:addev4out>
```

What to do next

Next: Continue with the next step, "Develop a message flow using a user-defined ESQL function" on page 2361.

Related concepts:

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

Related tasks:

"Adding database definitions to the WebSphere Message Broker Toolkit" on page 2278
Use the New Database Definition File wizard to add database definitions to the WebSphere Message Broker Toolkit.

"Message mapping scenarios" on page 2318

Related reference:

"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.

"Scenario D: Complex message enrichment" on page 2345
This scenario demonstrates complex message enrichment and uses complex message manipulation. Use the WebSphere Message Broker Toolkit to create message flows and message sets, and to create and deploy broker archive (BAR) files.

"Supported databases" on page 3591
You can optionally configure databases to contain data that is accessed by your message flows. Databases from IBM and other suppliers are supported at specific versions on supported operating systems.

"Mapping node" on page 4571
Use the Mapping node to construct one or more new messages and populate them with various types of information.

Develop a message flow using a user-defined ESQL function:

Develop a message flow that calls an ESQL function for mapping scenario D, complex message enrichment.

Before you begin

Before you start

Perform the previous tasks for this scenario, described in the following steps:

1. "Develop a message flow that contains other nodes" on page 2346
2. "Develop a message flow to map target fields from multiple other fields" on page 2349
3. "Develop a message flow and message model for mapping a complex nested, repeating message" on page 2352
4. "Develop a message flow for populating a target from a database" on page 2358

About this task

This step is the ninth task in creating "Scenario D: Complex message enrichment" on page 2345. Develop a message flow that uses a user-defined ESQL function, and create corresponding message models and instance documents.

Procedure

1. In the COMPLEX message definition, in namespace www.complex.net, create messages called addev5in and addev5out, which have the following structures:

```
addev5in
   value1                  (xsd:decimal) local element
   operator               (xsd:string) local element
   value2                  (xsd:decimal) local element
   rate                      (xsd:decimal) local element

addev5out
   grossvalue             (xsd:decimal) local element
   netvalue                 (xsd:decimal) local element
```

2. Create a message flow called addev5, which contains the following structure: MQInput -> Mapping -> MQOutput.

3. Open the map for the Mapping node, and select addev5in as the source and addev5out as the target.
4. In the MAPPING3_COMPLEX_flows project, create an ESQL file called addev5, and code the following functions:

```
CREATE FUNCTION calcGrossvalue(IN value1 DECIMAL, IN operator CHAR,
 IN value2 DECIMAL) RETURNS DECIMAL
          BEGIN
              DECLARE outval DECIMAL;
              CASE operator
              WHEN 'PLUS' THEN
                  SET outval = value1 + value2;
              WHEN 'MINUS' THEN
                  SET outval = value1 - value2;
              WHEN 'MULTIPLY' THEN
                  SET outval = value1 * value2;
              WHEN 'DIVIDE' THEN
                  SET outval = value1 / value2;
              ELSE
                  THROW USER EXCEPTION MESSAGE 2949 VALUES('Invalid Operator', operator);
                  SET outval = -999999;
              END CASE;
            RETURN outval;
          END;
```

```
CREATE FUNCTION calcNetvalue(IN value1 DECIMAL, IN operator CHAR, IN value2 DECIMAL,
 IN rate DECIMAL) RETURNS DECIMAL
    BEGIN
        DECLARE grossvalue DECIMAL;
        SET grossvalue=calcGrossvalue(value1, operator, value2);
        RETURN (grossvalue * rate );
    END;
```

5. In the Message Mapping editor Spreadsheet pane, expand the message and select grossvalue.
6. In the Expression pane, enter the expression:

```
esql:calcGrossvalue($source/comp:addev5in/value1,
$source/comp:addev5in/operator,
$source/comp:addev5in/value2)
```

7. Select the target netvalue, and in the Expression pane, enter the following expression:

```
esql:calcNetvalue($source/comp:addev5in/value1,
$source/comp:addev5in/operator,
$source/comp:addev5in/value2,
$source/comp:addev5in/rate)
```

8. Expand the Properties tree and set the following values:

```
MessageType      |       'addev5out'
```

9. Create the following instance messages, with appropriate MQRFH2 headers:

```
<comp:addev5in xmlns:comp="http://www.complex.net">
<value1>125.32</value1>
<operator>PLUS</operator>
<value2>25.86</value2>
<rate>0.60</rate>
</comp:addev5in>
```

```
<comp:addev5in xmlns:comp="http://www.complex.net">
<value1>118.00</value1>
<operator>MINUS</operator>
<value2>245.01</value2>
<rate>0.30</rate>
</comp:addev5in>
```

```
<comp:addev5in xmlns:comp="http://www.complex.net">
<value1>254.02</value1>
<operator>MULTIPLY</operator>
<value2>3.21</value2>
<rate>0.75</rate>
</comp:addev5in>

<comp:addev5in xmlns:comp="http://www.complex.net">
<value1>1456.33</value1>
<operator>DIVIDE</operator>
<value2>18.58</value2>
<rate>0.92</rate>
</comp:addev5in>

<comp:addev5in xmlns:comp="http://www.complex.net">
<value1>254.02</value1>
<operator>MOD</operator>
<value2>3.21</value2>
<rate>0.75</rate>
</comp:addev5in>
```

Results

You have created the following resources:

- Messages called addev5in and addev5out in a message definition called COMPLEX
- A message flow called addev5, which contains the mapping file `addev5_Mapping.msgmap` and ESQL file `addev5.esql`
- Files that contain instance messages

What to do next

Now deploy the message set and message flow.

Deploy the message set and message flow:
About this task

This step is the tenth task in creating "Scenario D: Complex message enrichment" on page 2345. Deploy the message set and message flow, and run the instance messages through the broker.

Procedure

1. Create a BAR file called addev5.
2. Add the message set MAPPING3_COMPLEX_messages and the message flow addev5 to the BAR file.
3. Deploy the BAR file to the broker.
4. Put the instance documents on the input queue.

Results

The output messages have the following content:

```
<comp:addev5out xmlns:comp="http://www.complex.net">
<grossvalue>151.18</grossvalue>
<netvalue>90.708</netvalue>
</comp:addev5out>

<comp:addev5out xmlns:comp="http://www.complex.net">
<grossvalue>-127.01</grossvalue>
<netvalue>-38.103</netvalue>
</comp:addev5out>
```

```
<comp:addev5out xmlns:comp="http://www.complex.net">
<grossvalue>815.4042</grossvalue>
<netvalue>611.55315</netvalue>
</comp:addev5out>
```

```
<comp:addev5out xmlns:comp="http://www.complex.net">
<grossvalue>78.381593110871905274488697524219 59</grossvalue>
<netvalue>72.111065662002152852529601722282 02</netvalue>
</comp:addev5out>
```

If you do not see any message output, look for an entry in the Deployment Log with content similar to the following entry:

```
BIP2949 ( BRK.default ) A user generated ESQL exception has been thrown. The additional
information provided with this exception is: ''Invalid Operator'' ''MOD''
'addev5.Mapping.ComIbmCompute' '%5' '%6' '%7' '%8' '%9' '%10' '%11'
```

This exception was thrown by a THROW EXCEPTION statement, and shows the normal behavior of the THROW statement; this exception is user-generated, so the user action is determined by the message flow and the type of exception that is thrown.

What to do next

Next: Continue with the next step, "Develop a message flow using a user-defined Java procedure."

Related concepts:

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

Related tasks:

"Message mapping scenarios" on page 2318

Related reference:

"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.

"Scenario D: Complex message enrichment" on page 2345
This scenario demonstrates complex message enrichment and uses complex message manipulation. Use the WebSphere Message Broker Toolkit to create message flows and message sets, and to create and deploy broker archive (BAR) files.

"Mapping node" on page 4571
Use the Mapping node to construct one or more new messages and populate them with various types of information.

Develop a message flow using a user-defined Java procedure:

Develop a message flow that calls a Java procedure for mapping scenario D, complex message enrichment.

Before you begin

Before you start

Perform the previous tasks for this scenario, described in the following steps:

1. "Develop a message flow that contains other nodes" on page 2346
2. "Develop a message flow to map target fields from multiple other fields" on page 2349

3. "Develop a message flow and message model for mapping a complex nested, repeating message" on page 2352
4. "Develop a message flow for populating a target from a database" on page 2358
5. "Develop a message flow using a user-defined ESQL function" on page 2361

About this task

This step is the eleventh task in creating "Scenario D: Complex message enrichment" on page 2345. Develop a message flow using a user-defined Java procedure, and create corresponding message models and instance documents.

Procedure

1. In the COMPLEX message definition, in namespace www.complex.net, create messages called addev6in and addev6out, which have the following structures:

```
addev6in
    hexdata               (xsd:hexBinary) local element
addev6out
    decval                 (xsd:decimal) local element
    fltval                    (xsd:float) local element
    intval                  (xsd:int) local element
```

2. Create a message flow called addev6, which contains the following structure: MQInput -> Mapping -> MQOutput.
3. Open the map for the Mapping node, and select addev6in as the source and addev6out as the target.
4. In the MAPPING3_COMPLEX_flows project, create an ESQL file called `addev6` and put these functions in it:

```
CREATE PROCEDURE decFromBinary( IN hexval BLOB )
 RETURNS DECIMAL
 LANGUAGE JAVA
 EXTERNAL NAME "addev6.decFromBinary";
CREATE PROCEDURE fltFromBinary( IN hexval BLOB )
 RETURNS DECIMAL
 LANGUAGE JAVA
 EXTERNAL NAME "addev6.fltFromBinary";
CREATE PROCEDURE intFromBinary( IN hexval BLOB )
 RETURNS DECIMAL
 LANGUAGE JAVA
 EXTERNAL NAME "addev6.intFromBinary";
```

5. Create a Java source file called `addev6.java`, which has the following contents:

```
import java.lang.*;
import java.math.*;

public class addev6 {
    //
    // Return decimal element from binary string
    //
    public static BigDecimal decFromBinary( byte[] hexval) {
    // Look for element named decval
    String search = "decval";
    String snval = findElement(hexval ,search );
    // Convert the value to decimal type
    BigDecimal numval = new BigDecimal(snval);
    return numval;
    }
    //
    // Return float element from binary string
```

```
    //
    public static Double fltFromBinary( byte[] hexval) {
    // Look for element named fltval
    String search = "fltval";
    String snval = findElement(hexval ,search );
    // Convert the value to float type
    Double numval = new Double(snval);
    return numval;
    }
    //
    // Return integer element from binary string
    //
    public static Long intFromBinary( byte[] hexval) {
    // Look for element named intval
    String search = "intval";
    String snval = findElement(hexval ,search );
    // Convert the value to integer type
    Long numval = new Long(snval);
    return numval;
    }
    //
    // Locate the named element and its value in the binary data
    //
    private static String findElement( byte[] hexval, String search ) {
    // Convert bytes to string
    String hexstr = new String(hexval);
    // Fixed length label/value pairs (length=14)
    int nvals = hexstr.length() / 14;
    String numval = "";
    String[] label = new String[nvals];
    String[] value = new String[nvals];
    // Loop over number of label/value pairs
    for ( int i=0; i < nvals; i ++ ) {
        // get start position
        int st = i * 14;
        // label is length 6
        int endl = st + 6;
        // value is length 8
        int endv = endl + 8;
        // extract label and value from string
        label[i] = hexstr.substring( st, endl);
        value[i] = hexstr.substring( (endl+1), endv);
        // Check whether the current pair has the label requested
        if ( label[i].compareTo( search) == 0 ) {
        // trim padding from the value
        numval = value[i].trim();
        }
    }
    return numval;
    }
}
```

6. Compile the Java code, and add the location of the class file to the system classpath. You might have to restart Windows if you edit the CLASSPATH.
7. In the Message Mapping editor Spreadsheet pane, expand the target message and set the target decval to the value `esql:decFromBinary($source/comp:addev6in/bval)`.
8. Set the target fltval to `esql:fltFromBinary($source/comp:addev6in/bval)`.
9. Set the target intval to `esql:intFromBinary($source/comp:addev6in/bval)`.
10. Expand the Properties target and set the values shown:

    ```
    MessageType     |     'addev6out
    ```

11. Create the following instance message, with appropriate MQRFH2 headers:

```
<comp:addev6in xmlns:comp="http://www.complex.net">
<bval>
<![CDATA[64656376616c20202031342e3238666c7476616c
2020312e34452b32696e7476616c2020202020313230]]>
</bval>
</comp:addev6in>
```

Results

You have created the following resources:

- Messages called addev6in and addev6out in a message definition called COMPLEX
- A message flow called addev6, which contains the mapping file `addev6_Mapping.msgmap` and ESQL file `addev6.esql`
- A Java source file called `addev6.java` and a compiled class file called `addev6.class` in a place where the system CLASSPATH can find it
- Files that contain instance messages

What to do next

Now deploy the message set and message flow.

Deploy the message set and message flow:
About this task

This step is the final task in creating "Scenario D: Complex message enrichment" on page 2345. Deploy the message set and message flow, and run the instance message through the broker.

Procedure

1. Create a BAR file called addev6.
2. Add the message set MAPPING3_COMPLEX_messages and the message flow addev6 to the BAR file.
3. Deploy the BAR file to the broker.
4. Put the instance documents on the input queue.

Results

The output message has the following content:

```
<comp:addev6out xmlns:comp="http://www.complex.net">
<decval>14.28</decval>
<fltval>1.4E+2</fltval>
<intval>120</intval>
</comp:addev6out>
```

What to do next

Next: You have now completed scenario D. If you want more information about how to use maps, return to "Message mapping scenarios" on page 2318, and explore another scenario.

Related concepts:

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

Related tasks:

"Message mapping scenarios" on page 2318

Related reference:

"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.

"Scenario D: Complex message enrichment" on page 2345
This scenario demonstrates complex message enrichment and uses complex message manipulation. Use the WebSphere Message Broker Toolkit to create message flows and message sets, and to create and deploy broker archive (BAR) files.

"Mapping node" on page 4571
Use the Mapping node to construct one or more new messages and populate them with various types of information.

Scenario E: Resolving a choice with alternative message data:

This scenario demonstrates how to resolve a choice with alternative message data.

Before you begin

Before you start:

1. Create the appropriate message model, either by using the tooling or by importing the message structure files (for example, C header or XML Schema Definition files).
2. Create a message flow that has the following structure:

   ```
   MQInput > Mapping > MQOutput
   ```

About this task

The following message model is used in this example:

```
chsmess  (message)
 head   (xsd:string)
 choice  (group)
  str1  (xsd:string)
  int1  (xsd:int)
  dur1  (xsd:duration)
 footer  (xsd:string)
```

Procedure

1. Right-click the Mapping node and click **Open Map**.
2. Accept the default project and name, and click **Next**.
3. Accept the default usage and click **Next**.
4. Clear **Based on records in a database** and click **Next**.
5. Select `chsmess` as the source message and the target message, and click **OK**.
6. In the Connection pane, open the source and target trees by clicking the addition (+) icons.
7. Open the `chsmess` tree in the Source and Target panes in the same way.
8. In both Source and Target panes, click the addition (+) icon next to the `choice` group.
9. Click `head` in the Message Mapping editor Source pane and drag it onto `head` in the Target pane. A line joins them.
10. Repeat Step 10 for each corresponding element (`str1`, `int1`, `dur1`, and `footer`.)
11. In the Map Script | Value table, open the tree by clicking the `$target +` box.

12. Open the chsmess tree. A set of if-elseif elements appears.
13. Open each if or elseif element. One if or elseif statement exists for each choice; each if or elseif statement has the condition 1=1.
14. Click the first function (for example, for str1) and change it in the Edit pane so that it has the content $source/chsmess/head='str1. If the input element head has a value str1, the assignment str1 <- $source/chsmess/str1 takes place.
15. Click the second function (for example, for int1) and change it in the Expression editor so that it has the content $source/chsmess/head='int1'. If the input element head has a value int1, the assignment int1 <- $source/chsmess/int1 takes place.
16. Click the third function (for example, for dur1) and change it in the Expression editor so that it has the content $source/chsmess/head='dur1'. If the input element head has a value dur1, the assignment dur1 <- $source/chsmess/dur1 takes place.
17. Save the mapping by clicking **File** > **Save**.

Results

You have completed this scenario. The message model contains a choice that has been resolved by using other data in the instance message.

Related tasks:

"Using message mappings" on page 2228
Message mappings define the blueprint for creating a message.

"Transforming and enriching messages" on page 2227
Transform and enrich messages by using one or more of the techniques described in this section.

Related reference:

"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.

"Mapping node" on page 4571
Use the Mapping node to construct one or more new messages and populate them with various types of information.

Scenario F: Updating the value of a message element:

This scenario demonstrates how to update the value of a message element.

Before you begin

Before you start:

1. Create the appropriate message model, either by using the WebSphere Message Broker Toolkit, or by importing the message structure files (for example, C header or XML Schema Definition files).
2. Create a message flow that has the following structure:
   ```
   MQInput > Mapping > MQOutput
   ```

About this task

The message model used in this example has the following structure:

```
simple  (message)
  int (xsd:int)
  str (xsd:str)
```

Procedure

1. Right-click the Mapping node and click **Open Map**.
2. Select `simple` as the source message and the target message and click **OK**.
3. In the connection pane, open the source and target trees by clicking the addition (+) icons.
4. Open the `simple` trees on both sides in the same way.
5. Select `int` in the Message Mapping editor Source pane, and drag it onto `int` in the Target pane. A line joins them.
6. Select `str` in the Message Mapping editor Source pane and drag it onto `str` in the Target pane. A line joins them.
7. In the Map Script | Value table, open the tree by clicking the `$target +` box
8. Open the `simple` tree in the same way; both `int` and `str` have values (for example, `int $source/simple/int` and `str $source/simple/str`).
9. Select the value for `int`. The value appears in the Expression Editing pane.
10. Edit the value so that it has the content `$source/simple/int + 1` and press Enter. The value in the table is updated (the input value is incremented).
11. Select the value for `str` and edit it so that it has the content `esql:upper($source/simple/str)`, and press Enter. The value in the table is updated (the input value to is changed to uppercase).
12. Save the mapping by clicking **File** > **Save**.

Results

You have completed this scenario. The input and output messages have the same structure and format, but the element values have been modified.

Related tasks:

"Using message mappings" on page 2228
Message mappings define the blueprint for creating a message.

"Transforming and enriching messages" on page 2227
Transform and enrich messages by using one or more of the techniques described in this section.

Related reference:

"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.

"Mapping node" on page 4571
Use the Mapping node to construct one or more new messages and populate them with various types of information.

Developing ESQL

Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

About this task

You must create, for each node, an ESQL module in which you code the ESQL statements and functions to tailor the behavior of the node. You can access message content, or database content, or both, to achieve the results that you require. ESQL modules are maintained in ESQL files, managed through the Broker Application Development perspective.

This section provides the following information:

- "ESQL overview"
- "Managing ESQL files" on page 2390
- "Writing ESQL" on page 2413

You can use the ESQL debugger, which is part of the flow debugger, to debug the code that you write. The debugger steps through ESQL code statement by statement, so that you can view and check the results of every line of code that is run.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Debugging ESQL" on page 3182
When message flow processing has paused at a breakpoint that you have set in source code within a node that contains ESQL code, you can examine and modify the ESQL variables in the Flow Debugger.

"Accessing broker properties from ESQL" on page 2625
You can access broker properties, at run time, from the ESQL modules in your message flow nodes.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

ESQL overview

Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

This section contains introductory information about ESQL.

- For descriptions of ESQL user tasks, see "Writing ESQL" on page 2413.
- For reference information about ESQL, see "ESQL reference" on page 5019.

Read the following information before you proceed:

- An overview of message flows in "Message flows overview" on page 1022.
- An overview of message trees in "The message tree" on page 1041, and the topics within this container, paying special attention to "Logical tree structure" on page 1042.

ESQL is based on Structured Query Language (SQL) which is in common usage with relational databases such as DB2. ESQL extends the constructs of the SQL language to provide support for you to work with message and database content to define the behavior of nodes in a message flow.

The ESQL code that you create to customize nodes within a message flow is defined in an ESQL file, typically named `<message_flow_name>.esql`, which is associated with the message flow project. You can use ESQL in the following built-in nodes:

- "Compute node" on page 4340
- "Database node" on page 4354
- "DatabaseInput node" on page 4360
- "Filter node" on page 4452

You can also use ESQL to create functions and procedures that you can use in the following built-in nodes:

- "DataDelete node" on page 4382
- "DataInsert node" on page 4386
- "DataUpdate node" on page 4390
- "Extract node" on page 4412
- "Mapping node" on page 4571
- "Warehouse node" on page 4963

To use ESQL correctly and efficiently in your message flows, you must also understand the following concepts:

- Data types
- Variables
- Field references
- Operators
- Statements
- Functions
- Procedures
- Modules

Use the ESQL debugger, which is part of the flow debugger, to debug the code that you write. The debugger steps through ESQL code statement by statement, so that you can view and check the results of every line of code that is run.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"The message tree" on page 1041
A message tree is a structure that is created, either by one or more parsers when an input message bit stream is received by a message flow, or by the action of a message flow node.

"Client application programming interfaces" on page 1038
You can configure the nodes in your message flows to customize the behavior of those nodes by using one or more of the supported programming interfaces.

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Using message mappings" on page 2228
Message mappings define the blueprint for creating a message.

"Writing ESQL" on page 2413
How you can use ESQL to customize nodes.

"Debugging ESQL" on page 3182
When message flow processing has paused at a breakpoint that you have set in source code within a node that contains ESQL code, you can examine and modify the ESQL variables in the Flow Debugger.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

ESQL data types:

A data type defines the characteristics of an item of data, and determines how that data is processed. ESQL supports six data types, listed later in this section. Data that is retrieved from databases, received in a self-defining message, or defined in a message model (using MRM data types), is mapped to one of these basic ESQL types when it is processed in ESQL expressions.

Within a broker, the fields of a message contain data that has a definite data type. It is also possible to use intermediate variables to help process a message. You must declare all such variables with a data type before use. A variable's data type is fixed; If you try to assign values of a different type you get either an implicit cast or an exception. Message fields do not have a fixed data type, and you can assign values of a different type. The field adopts the new value and type.

It is not always possible to predict the data type that results from evaluating an expression. This is because expressions are compiled without reference to any kind of message schema, and so some type errors are not caught until run time.

ESQL defines the following categories of data. Each category contains one or more data types.

- Boolean
- Datetime
- Null
- Numeric
- Reference
- String

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Correlation names" on page 1069
A correlation name is a field reference that identifies a well-defined starting point in the logical message tree and is used in field references to describe a standard part of the tree format.

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

"Representation of ESQL datetime data types" on page 5030
When your application sends a message to a broker, the way in which the message data is interpreted depends on the content of the message itself and the configuration of the message flow. If your application sends a message to be interpreted either by the generic XML parser, or the MRM parser, that is tailored by an XML physical format, the application can include date or time data that is represented by any of the XML Schema primitive datetime data types.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"Data types for elements in an MRM message" on page 6254
A parser is supplied for the body of a message in the MRM domain; it associates each field with a specific data type.

"Data types of values from external databases" on page 5288
How database data types are implicitly cast to ESQL data types.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"CAST function" on page 5245

"Supported casts" on page 5273

ESQL variables:

An ESQL variable is a data field that is used to help process a message.

You must declare a variable and state its type before you can use it. The data type of a variable is fixed; if you code ESQL that assigns a value of a different type, either an implicit cast to the data type of the target is implemented or an exception is raised (if the implicit cast is not supported).

To define a variable and give it a name, use the DECLARE statement.

The names of ESQL variables are case-sensitive; therefore, make sure that you use the correct case in all places. The simplest way to guarantee that you are using the correct case is always to define variables using uppercase names.

The WebSphere Message Broker Toolkit marks variables that have not been defined. Remove all these warnings before deploying a message flow.

You can assign an initial value to the variable on the DECLARE statement. If an initial value is not specified, scalar variables are initialized with the special value NULL, and ROW variables are initialized to an empty state. Then, you can change the value of a variable by using the SET statement.

ESQL Variables declared at Module level 'belong' to a single node. However, variables declared at the Schema level are also given to each node that references that schema. So although variables at schema level are only declared once, each ESQL node has its own copy, which is not shared with any other node (unless the variable is marked `SHARED`).

Three types of built-in node can contain ESQL code and therefore support the use of ESQL variables:
- "Compute node" on page 4340
- "Database node" on page 4354
- "DatabaseInput node" on page 4360
- "Filter node" on page 4452

Scope, lifetime, and sharing characteristics of variables

The scope, lifetime, and sharing characteristics of variables describe how widespread and for how long a particular ESQL variable is available:

Scope Is a measure of the range over which a variable is visible. In the broker environment, the scope of variables is typically limited to the individual node.

Lifetime
Is a measure of the time for which a variable retains its value. In the broker environment, the lifetime of variables varies but is typically restricted to the life of a thread within a node.

Sharing characteristics
Indicate whether each thread has its own copy of a variable or whether one variable is shared between many threads. In the broker environment, variables are typically not shared.

Types of variable

External
External variables (defined with the EXTERNAL keyword) are also known as *user-defined properties*, see "User-defined properties in ESQL" on page 2376. They exist for the entire lifetime of a message flow and are visible to all messages passing through the flow. You can define external variables only at the module and schema level. You can modify the initial values of external variables (optionally set by the DECLARE statement) at design time, by using the Message Flow editor, or at deployment time, by using the Broker Archive editor. You can query and set the values of user-defined properties at run time by using the Administration API for WebSphere Message Broker (also known as the CMP API). For more information, see "Setting message flow user-defined properties at run time in a CMP application" on page 985.

Normal
Normal variables have a lifetime of just one message passing through a node. They are visible to that message only. To define normal variables, omit both the EXTERNAL and SHARED keywords.

Shared
Shared variables can be used to implement an in-memory cache in the message flow, see "Optimizing message flow response times" on page 3264. Shared variables have a long lifetime and are visible to multiple messages passing through a flow, see "Long-lived variables" on page 2378. Shared variables exist for the lifetime of the execution group process, the lifetime of the flow or node, or the lifetime of the node SQL that declares the variable (whichever is the shortest). Shared variables are initialized when the first message passes through the flow or node after each broker startup.

See also the ATOMIC option of the “BEGIN ... END statement” on page 5070. The BEGIN ATOMIC construct is useful when a number of changes must be made to a shared variable and it is important to prevent other instances seeing the intermediate states of the data.

Related concepts:

“User-defined properties in ESQL”
Access user-defined properties (UDPs) as variables in your ESQL program by specifying the EXTERNAL keyword on a DECLARE statement. For example, the ESQL statement `DECLARE today EXTERNAL CHARACTER 'monday'` defines a user-defined property called today with an initial value monday.

“User-defined properties” on page 1147
A user-defined property (UDP) is a property that is defined when you construct a message flow by using the Message Flow editor. This property can be used by the ESQL or Java program inside message flow nodes, such as a Compute node.

“Long-lived variables” on page 2378
You can use appropriate long-lived ESQL data types to cache data in memory.

Related tasks:

“Setting message flow user-defined properties at run time in a CMP application” on page 985
Use the CMP API to query, discover, and set message flow user-defined properties dynamically at run time. You can use the CMP API to set properties with a data type of character.

“Developing ESQL” on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

“Built-in nodes” on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

“Compute node” on page 4340
Use the Compute node to construct one or more new output messages.

“Database node” on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

“DatabaseInput node” on page 4360
Use the DatabaseInput node to detect events recorded in a database, and to retrieve the data affected by those events.

“Filter node” on page 4452
Use the Filter node to route a message according to message content.

“DECLARE statement” on page 5117
Use the DECLARE statement to define a variable, the data type of the variable and, optionally, its initial value.

“BEGIN ... END statement” on page 5070
The BEGIN ... END statement gives the statements defined within the BEGIN and END keywords the status of a single statement.

User-defined properties in ESQL:

Access user-defined properties (UDPs) as variables in your ESQL program by specifying the EXTERNAL keyword on a DECLARE statement. For example, the ESQL statement `DECLARE today EXTERNAL CHARACTER 'monday'` defines a user-defined property called today with an initial value monday.

Before you can use a user-defined property, you must define the property in the Message Flow editor when you construct a message flow that uses it. When you define a UDP in the Message Flow editor, you must define a value and the property type. The value can be a default value, which varies according to the type of the UDP. The value that is assigned to the UDP in the Message Flow editor takes precedence over a value that you have assigned to the UDP in your ESQL program.

You can also define a UDP for a subflow. A UDP has global scope and is not specific to a particular subflow. If you reuse a subflow in a message flow, and those subflows have identical UDPs, you cannot set the UDPs to different values.

Before you deploy the message flow that uses the UDP, you can change the value of the UDP in the Broker Archive editor. If you try to deploy a message flow that contains a UDP that has had no value assigned to it, a deployment failure occurs. For more information, see "Configuring a message flow at deployment time with user-defined properties" on page 2626.

You can use UDPs to set configuration data, and use them like typical properties. No external calls to user-written plug-ins or parsing of environment trees are involved, and parsing costs of reading data out of trees are removed. The value of the UDP is finalized in the variable at deployment time.

You can declare UDPs only in modules or schemas. You can query, discover, and set UDPs at run time, to dynamically change the behavior of a message flow. For more information, see "User-defined properties" on page 1147.

You can access UDPs from the following built-in nodes that use ESQL:
- Compute
- Database
- DatabaseInput
- Filter

For a description of how to access a UDP from a JavaCompute node, see "Accessing message flow user-defined properties from a JavaCompute node" on page 2659.

Related concepts:

"User-defined properties" on page 1147
A user-defined property (UDP) is a property that is defined when you construct a message flow by using the Message Flow editor. This property can be used by the ESQL or Java program inside message flow nodes, such as a Compute node.

"ESQL variables" on page 2374
An ESQL variable is a data field that is used to help process a message.

Related tasks:

"Configuring a message flow at deployment time with user-defined properties" on page 2626
Use user-defined properties (UDPs) to configure message flows at deployment and run time, without modifying program code. You can give a UDP an initial value when you declare it in your program, or when you use the Message Flow editor to create or modify a message flow.

"Setting message flow user-defined properties at run time in a CMP application" on page 985
Use the CMP API to query, discover, and set message flow user-defined properties dynamically at run time. You can use the CMP API to set properties with a data

type of character.

"Accessing message flow user-defined properties from a JavaCompute node" on page 2659
Customize a JavaCompute node to access properties that you have associated with the message flow in which the node is included.

Related reference:

"ESQL variables" on page 5048
ESQL variables can be described as *external variables*, *normal variables*, or *shared variables*; their use is defined in the DECLARE statement.

"DECLARE statement" on page 5117
Use the DECLARE statement to define a variable, the data type of the variable and, optionally, its initial value.

"Message Flow editor" on page 6810
The Message Flow editor is the default editor provided by the Broker Application Development perspective for defining a graphical representation of a message flow in the WebSphere Message Broker Toolkit, and for setting properties for individual message flow nodes.

"Broker Archive editor" on page 6794
Use the Broker Archive editor to create and manage broker archive (BAR) files.

Long-lived variables:

You can use appropriate long-lived ESQL data types to cache data in memory.

Sometimes data has to be stored beyond the lifetime of a single message passing through a flow. One way to store this data is to store the data in a database. Using a database is good for long-term persistence and transactionality, but access (particularly write access) is slow.

Alternatively, you can use appropriate long-lived ESQL data types to provide an in-memory cache of the data for a certain period of time. Using long-lived ESQL data types makes access faster than from a database, although this speed is at the expense of shorter persistence and no transactionality.

You create long-lifetime variables by using the SHARED keyword on the DECLARE statement. For further information, see "DECLARE statement" on page 5117.

The following sample demonstrates how to define shared variables using the DECLARE statement. The sample demonstrates how to store routing information in a database table and use shared variables to store the database table in memory in the message flow to improve performance.

- Message Routing

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Long-lived data types have an extended lifetime beyond that of a single message passing through a node. Long-lived data types are shared between threads and exist for the life of a message flow (the time between configuration changes to a message flow), as described in the following tables.

Table 23. Short lifetime variables

	Scope	**Life**	**Shared**
Schema & Module	Node	Thread within node	Not at all
Routine Local	Node	Thread within routine	Not at all
Block Local	Node	Thread within block	Not at all

Table 24. Long lifetime variables

	Scope	**Life**	**Shared**
Node Shared	Node	Life of node	All threads of flow
Flow Shared	Flow	Life of flow	All threads of flow

Features of long-lived ESQL data types include:

- The ability to handle large amounts of long-lifetime data.
- The joining of data to messages fast.
- On multiple processor machines, multiple threads can access the same data simultaneously.
- Subsequent messages can access the data left by a previous message.
- Long lifetime read-write data can be shared between threads, because there is no long-term association between threads and messages.
- In contrast to data stored in database tables in the environment, this type of data is stored privately; that is, within the broker.
- ROW variables can be used to create a modifiable copy of the input message; see "ESQL ROW data type" on page 5038.
- It is possible to create shared constants.

A typical use of these data types might be in a flow in which data tables are 'read-only' as far as the flow is concerned. Although the table data is not actually static, the flow does not change it, and thousands of messages pass through the flow before there is any change to the table data.

Examples include:

- A table which contains a day's credit card transactions. The table is created each day and that day's messages are run against it. Then the flow is stopped, the table updated and the next day's messages run. These flows might perform better if they cache the table data rather than read it from a database for each message.
- The accumulation and integration of data from multiple messages.

Related concepts:

"ESQL variables" on page 2374
An ESQL variable is a data field that is used to help process a message.

"User-defined properties in ESQL" on page 2376
Access user-defined properties (UDPs) as variables in your ESQL program by specifying the EXTERNAL keyword on a DECLARE statement. For example, the ESQL statement `DECLARE today EXTERNAL CHARACTER 'monday'` defines a user-defined property called today with an initial value monday.

Related reference:

"DECLARE statement" on page 5117
Use the DECLARE statement to define a variable, the data type of the variable

and, optionally, its initial value.

"ESQL ROW data type" on page 5038
The ROW data type holds a *tree structure*. A row in a database is a particular type of tree structure, but the ROW data type is not restricted to holding data from database rows.

Broker properties:

For each broker, WebSphere Message Broker maintains a set of properties. You can access some of these properties from your ESQL programs. A subset of the properties is also accessible from Java programs. It can be useful, at run time, to have real-time access to details of a specific node, flow, or broker.

Broker properties are divided into four categories:

- Properties that relate to a specific node
- Properties that relate to nodes in general
- Properties that relate to a message flow
- Properties that relate to the execution group

For a description of the broker, flow, and node properties that are accessible from ESQL and Java, see "Broker properties that are accessible from ESQL and Java" on page 5302.

Broker properties have the following characteristics.

- They are grouped by broker, execution group, flow, and node.
- They are case-sensitive. Their names always start with an uppercase letter.
- They return NULL if they do not contain a value.

All nodes for which user programs can edit ESQL support access to broker properties. These nodes are:

- Compute nodes
- Database nodes
- DatabaseInput nodes
- Filter nodes
- All derivatives of these nodes

User-defined properties can be queried, discovered, and set at run time to dynamically change the behavior of a message flow. You can use the Administration API for WebSphere Message Broker(CMP API) to manipulate these properties, which can be used by a systems monitoring tool to perform automated actions in response to situations that it detects in the monitored systems. For more information, see "User-defined properties" on page 1147.

A *complex property* is a property to which you can assign multiple values. Complex properties are displayed in a table in the Properties view, where you can add, edit, and delete values, and change the order of the values in the table. You cannot promote complex properties; therefore, they are not shown in the Promote properties dialog box. Nor can you configure complex properties; therefore, they are not supported in the Broker Archive editor. For an example of a complex property, see the `Query elements` property of the DatabaseRoute node.

For more information about editing the properties of a node, see "Configuring a message flow node" on page 1503.

Related concepts:

"User-defined properties in ESQL" on page 2376
Access user-defined properties (UDPs) as variables in your ESQL program by specifying the EXTERNAL keyword on a DECLARE statement. For example, the ESQL statement `DECLARE today EXTERNAL CHARACTER 'monday'` defines a user-defined property called today with an initial value monday.

"User-defined properties" on page 1147
A user-defined property (UDP) is a property that is defined when you construct a message flow by using the Message Flow editor. This property can be used by the ESQL or Java program inside message flow nodes, such as a Compute node.

Related tasks:

"Accessing broker properties from ESQL" on page 2625
You can access broker properties, at run time, from the ESQL modules in your message flow nodes.

"Accessing broker properties from the JavaCompute node" on page 2658
For each broker, WebSphere Message Broker maintains a set of properties. You can access some of these properties from your Java programs. It can be useful, during the run time of your code, to have real-time access to details of a specific node, flow, or broker.

Related reference:

"Broker properties that are accessible from ESQL and Java" on page 5302
You can access broker, message flow, and node properties from ESQL and Java.

"DatabaseRoute node" on page 4373
Use the DatabaseRoute node to route messages using information from a database in conjunction with XPath expressions.

ESQL field references:

An ESQL field reference is a sequence of period-separated values that identify a specific field (which might be a structure) within a message tree or a database table.

The path from the root of the information to the specific field is traced using the parent-child relationships.

A field reference is used in an ESQL statement to identify the field that is to be referenced, updated, or created within the message or database table. For example, you might use the following identifier as a message field reference:

```
Body.Invoice.Payment
```

You can use an ESQL variable of type REFERENCE to set up a dynamic pointer to contain a field reference. This might be useful in creating a fixed reference to a commonly-referenced point within a message; for example the start of a particular structure that contains repeating fields.

A field reference can also specify element types, XML namespace identifications, indexes and a type constraint; see "ESQL field reference overview" on page 5049 for further details.

The first name in a field reference is sometimes known as a *Correlation name*.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

“Correlation names” on page 1069
A correlation name is a field reference that identifies a well-defined starting point in the logical message tree and is used in field references to describe a standard part of the tree format.

“Message mapping overview” on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

Related tasks:

Chapter 9, “Developing message flow applications,” on page 1019
Develop message flows to process your business messages and data.

“Developing ESQL” on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

“Built-in nodes” on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

“ESQL field reference overview” on page 5049
You can use ESQL field references to form paths to message body elements.

ESQL operators:

An ESQL operator is a character or symbol that you can use in expressions to specify relationships between fields or values.

ESQL supports the following groups of operators:

- Comparison operators, to compare one value to another value (for example, "less than"). Refer to “ESQL simple comparison operators” on page 5057 and “ESQL complex comparison operators” on page 5058 for details of the supported operators and their use.
- Logical operators, to perform logical operations on one or two terms (for example, AND). Refer to “ESQL logical operators” on page 5062 for details of the supported operators and their use.
- Numeric operators, to indicate operations on numeric data (for example, +). Refer to “ESQL numeric operators” on page 5064 for details of the supported operators and their use.

There are some restrictions on the application of some operators to data types; not all lead to a meaningful operation. These are documented where they apply to each operator.

Operators that return a Boolean value (TRUE or FALSE), for example the "greater than" operator, are also known as *predicates*.

Related concepts:

“Message flows overview” on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

“Message mapping overview” on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

Related tasks:

Chapter 9, “Developing message flow applications,” on page 1019
Develop message flows to process your business messages and data.

“Developing ESQL” on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

“Built-in nodes” on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

“ESQL reference” on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

“ESQL operators” on page 5056
A list of the various groups of operators that ESQL supports.

ESQL statements:

An ESQL statement is an instruction that represents a step in a sequence of actions or a set of declarations.

ESQL provides a large number of different statements that perform different types of operation. All ESQL statements start with a keyword that identifies the type of statement and end with a semicolon. An ESQL program consists of a number of statements that are processed in the order they are written.

As an example, consider the following ESQL program:

```
DECLARE x INTEGER;
SET x = 42;
```

This program consists of two statements. The first starts with the keyword DECLARE and ends at the first semicolon. The second statement starts with the keyword SET and ends at the second semicolon. These two statements are written on separate lines and it is conventional (but not required) that they be so. You will notice that the language keywords are written in capital letters. This is also the convention but is not required; mixed case and lowercase are acceptable.

The first statement declares a variable called x of type INTEGER, that is, it reserves a space in the computer's memory large enough to hold an integer value and allows this space to be subsequently referred to in the program by the name x. The second statement sets the value of the variable x to 42. A number appearing in an ESQL program without decimal point and not within quotation marks is known as an integer literal.

ESQL has a number of data types and each has its own way of writing literal values. These are described in “ESQL data types” on page 2373.

For a full description of all the ESQL statements, see “ESQL statements” on page 5067.

Related concepts:

“Message flows overview” on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

"ESQL data types" on page 2373
A data type defines the characteristics of an item of data, and determines how that data is processed. ESQL supports six data types, listed later in this section. Data that is retrieved from databases, received in a self-defining message, or defined in a message model (using MRM data types), is mapped to one of these basic ESQL types when it is processed in ESQL expressions.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"ESQL statements" on page 5067
You can use ESQL statements to manipulate message trees, update databases, or interact with nodes.

ESQL nested statements:

An ESQL nested statement is a statement that is contained within another statement.

Consider the following ESQL program fragment:

```
IF Size> 100.00 THEN
  SET X = 0;
  SET Y = 0;
  SET REVERSE = FALSE;
ELSE
  SET X = 639;
  SET Y = 479;
  SET REVERSE = TRUE;
END IF;
```

In this example, you can see a single IF statement containing the optional ELSE clause. Both the IF and ELSE portions contain three nested statements. Those within the IF clause are processed if the operator > (greater than) returns the value TRUE (that is, if Size has a value greater than 100.00); otherwise, those within the ELSE clause are processed.

Many statements can have expressions nested within them, but only a few can have statements nested within them. The key difference between an expression and a statement is that an expression calculates a value to be used, whereas a statement performs an action (usually changing the state of the program) but does not produce a value.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

ESQL functions:

A function is an ESQL construct that calculates a value from a number of given input values.

A function usually has input parameters and can, but does not usually have, output parameters. It returns a value calculated by the algorithm described by its statement. This statement is usually a compound statement, such as BEGIN... END, because this allows an unlimited number of nested statements to be used to implement the algorithm.

ESQL provides a number of predefined, or "built-in", functions which you can use freely within expressions. You can also use the CREATE FUNCTION statement to define your own functions.

When you define a function, you must give it a unique name. The name is handled in a non-case sensitive way (that is, use of the name with any combination of uppercase and lowercase letters matches the declaration). This is in contrast to the names that you declare for schemas, constants, variables, and labels, which are handled in a case sensitive way, and which you must specify exactly as you declared them.

Consider the following ESQL program fragment:

```
SET Diameter = SQRT(Area / 3.142) * 2;
```

In this example, the function SQRT (square root) is given the value inside the brackets (itself the result of an expression, a divide operation) and its result is used in a further expression, a multiply operation. Its return value is assigned to the variable Diameter. See "Calling ESQL functions" on page 5168 for information about all the built-in ESQL functions.

In addition, an ESQL expression can refer to a function in another broker schema (that is, a function defined by a CREATE FUNCTION statement in an ESQL file in the same or in a different dependent project). To resolve the name of the called function, you must do one of the following:

- Specify the fully-qualified name (<SchemaName>.<FunctionName>) of the called function.
- Include a PATH statement to make all functions from the named schema visible. Note that this technique only works if the schemas do not contain identically-named functions. The PATH statement must be coded in the same ESQL file, but not within any MODULE.

Note that you cannot define a function within an EVAL statement or an EVAL function.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"Calling ESQL functions" on page 5168
Most ESQL functions belong to a schema called SQL and this is particularly useful if you have functions with the same name.

"CREATE FUNCTION statement" on page 5091
The CREATE FUNCTION statement defines a callable function or procedure.

"CREATE MODULE statement" on page 5101
The CREATE MODULE statement creates a module, which is a named container associated with a node.

"CREATE PROCEDURE statement" on page 5103
The CREATE PROCEDURE statement defines a callable function or procedure.

ESQL procedures:

An ESQL procedure is a subroutine that has no return value. It can accept input parameters from, and return output parameters to, the caller.

Procedures are very similar to functions. The main difference between them is that, unlike functions, procedures have no return value. Thus they cannot form part of an expression and are invoked by using the CALL statement. Procedures commonly have output parameters

You can implement a procedure in ESQL (an internal procedure) or as a database stored procedure (an external procedure). The ESQL procedure must be a single

ESQL statement, although that statement can be a compound statement such as BEGIN END. You cannot define a procedure within an EVAL statement or an EVAL function.

When you define a procedure, give it a name. The name is handled in a non-case sensitive way (that is, use of the name with any combination of uppercase and lowercase letters matches the declaration). That is in contrast to the names that you declare for schemas, constants, variables, and labels, which are handled in a case sensitive way, and which you must specify exactly as you declared them.

An ESQL expression can include a reference to a procedure in another broker schema (defined in an ESQL file in the same or a different dependent project). If you want to use this technique, either fully qualify the procedure, or include a PATH statement that sets the qualifier. The PATH statement must be coded in the same ESQL file, but not within a MODULE.

An external database procedure is indicated by the keyword EXTERNAL and the external procedure name. This procedure must be defined in the database and in the broker, and the name specified with the EXTERNAL keyword and the name of the stored database procedure must be the same, although parameter names do not have to match. The ESQL procedure name can be different from the external name it defines.

Overloaded procedures are not supported to any database. (An overloaded procedure is one that has the same name as another procedure in the same database schema which has a different number of parameters, or parameters with different types.) If the broker detects that a procedure has been overloaded, it raises an exception.

Dynamic schema name resolution for stored procedures is supported; when you define the procedure you must specify a wildcard for the schema that is resolved before invocation of the procedure by ESQL. This is explained further in “Invoking stored procedures” on page 2503.

Related concepts:

“Message flows overview” on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

“Message mapping overview” on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

Related tasks:

Chapter 9, “Developing message flow applications,” on page 1019
Develop message flows to process your business messages and data.

“Developing ESQL” on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

“Invoking stored procedures” on page 2503
To invoke a procedure that is stored in a database, use the ESQL CALL statement. The stored procedure must be defined by a CREATE PROCEDURE statement that has a Language clause of DATABASE and an EXTERNAL NAME clause that identifies the name of the procedure in the database and, optionally, the database schema to which it belongs.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"CREATE FUNCTION statement" on page 5091
The CREATE FUNCTION statement defines a callable function or procedure.

"CREATE MODULE statement" on page 5101
The CREATE MODULE statement creates a module, which is a named container associated with a node.

"CREATE PROCEDURE statement" on page 5103
The CREATE PROCEDURE statement defines a callable function or procedure.

ESQL modules:

A module is a sequence of declarations that define variables and their initialization, and a sequence of subroutine (function and procedure) declarations that define a specific behavior for a message flow node.

A module must begin with the `CREATE` *`node_type`* `MODULE` statement and end with an `END MODULE` statement. The *node_type* must be one of COMPUTE, DATABASE, DATABASEEVENT, or FILTER. For DatabaseInput nodes, the required *node_type* is DATABASEEVENT. For Compute, Database, and Filter nodes the required *node_type* is the same as the node. For example, you must use the COMPUTE *node_type* in ESQL modules for Compute nodes. For Compute, Database and Filter nodes, the entry point of the ESQL code is the function named MAIN, which has MODULE scope. ESQL modules for DatabaseInput nodes do not contain the MAIN function, but contain three procedures named READEVENTS, BUILDMESSAGE, and ENDEVENT. For more information about these three procedures, see "Configuring a DatabaseInput node" on page 2120.

Each module is identified by a name which follows `CREATE` *`node_type`* `MODULE`. The name might be created for you with a default value, which you can modify, or you can create it yourself. The name is handled in a non-case-sensitive way (that is, use of the name with any combination of uppercase and lowercase letters matches the declaration). This is in contrast to the names that you declare for schemas, constants, variables, and labels, which are handled in a case-sensitive way, and which you must specify exactly as you declared them.

You must create the code for a module in an ESQL file which has a suffix of `.esql`. You must create this file in the same broker schema as the node that references it. There must be one module of the correct type for each corresponding node, and it is specific to that node and cannot be used by any other node.

When you create an ESQL file (or complete a task that creates one), you indicate the message flow project and broker schema with which the file is associated and specify a name for the file.

Within the ESQL file, the name of each module is determined by the value of the corresponding property of the message flow node. For example, the property *`ESQL Module`* for the Compute node specifies the name of the module in the ESQL file for that node. The default value for this property is the name of the node. You can

specify a different name, but you must ensure that the value of the property and the name of the module that provides the required function are the same.

For Compute, Database, and Filter nodes, the module must contain the function MAIN, which is the entry point for the module. This function is included automatically if the module is created for you. Within MAIN, you can code ESQL to configure the behavior of the node. If you include ESQL within the module that declares variables, constants, functions, and procedures, these are of local scope only and can be used within this single module.

For DatabaseInput nodes, the module must contain the three procedures READEVENTS, BUILDMESSAGE, and ENDEVENT. These procedures are included automatically, together with comments to describe the procedure, if the module is created for you. Within the three procedures, you can code ESQL to configure the behavior of the node, see "Configuring a DatabaseInput node" on page 2120. If you include ESQL within the module that declares variables, constants, functions, and procedures, these are of local scope only and can be used within this single module.

If you want to reuse ESQL constants, functions, or procedures, you must declare them at broker schema level. You can then refer to these constants, functions or procedures from any resource within that broker schema, in the same or another project. If you want to use this technique, either fully qualify the procedure, or include a PATH statement that sets the qualifier. The PATH statement must be coded in the same ESQL file, but not within any MODULE.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"CREATE FUNCTION statement" on page 5091
The CREATE FUNCTION statement defines a callable function or procedure.

"CREATE MODULE statement" on page 5101
The CREATE MODULE statement creates a module, which is a named container associated with a node.

"CREATE PROCEDURE statement" on page 5103
The CREATE PROCEDURE statement defines a callable function or procedure.

"DECLARE statement" on page 5117
Use the DECLARE statement to define a variable, the data type of the variable and, optionally, its initial value.

Managing ESQL files

In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

About this task

The ESQL code for a node is contained in a module that is associated with the node. Each module must be created in an ESQL file (file extension `.esql`) . You can change the name of the module in the ESQL file from its default value of *`MessageFlow_NodeName`*, where *MessageFlow* is the name of the message flow and *NodeName* is the name of the node. The name of the module in the ESQL file must match the name specified for the module in the `ESQL Module` property of the corresponding node.

The following topics describe how to manage ESQL files.

- "Creating an ESQL file" on page 2391
- "Opening an existing ESQL file" on page 2393
- "Creating ESQL for a node" on page 2394
-
- "Modifying ESQL for a node" on page 2398
- "Saving an ESQL file" on page 2399
- "Copying an ESQL file" on page 2401
- "Renaming an ESQL file" on page 2405
- "Moving an ESQL file" on page 2406
- "Changing ESQL preferences" on page 2408
- "Deleting ESQL for a node" on page 2411
- "Deleting an ESQL file" on page 2412

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

"Broker schemas" on page 1036
A broker schema is a symbol space that defines the scope of uniqueness of the names of resources defined within it. The resources include message flows and other optional resources such as ESQL files, Java files, and mapping files.

"ESQL modules" on page 2388
A module is a sequence of declarations that define variables and their initialization, and a sequence of subroutine (function and procedure) declarations that define a specific behavior for a message flow node.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"Broker Application Development perspective" on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

"ESQL editor" on page 6798
The ESQL editor is the default editor provided by the Broker Application Development perspective for editing ESQL (`.esql`) files.

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

Creating an ESQL file:

When you include a node in your message flow that requires ESQL to customize its function (the Compute, Database, DatabaseInput, and Filter nodes), you must code the ESQL statements that provide the customization in an ESQL module in an ESQL file. You can use the same ESQL file for more than one module.

Before you begin

Before you start:

For background information, see "ESQL overview" on page 2371.

To complete this task, you must have created a message flow project to contain your ESQL file.

About this task

ESQL files are stored in a file system or in a shared repository. If you are using a file system, you can use the local file system or a shared drive. If you store files in a repository, you can use any of the available repositories that are supported by Eclipse, for example CVS.

To create an ESQL file, complete the following steps.

Procedure

1. Click **File** > **New** > **Message Flow ESQL File**.

   You can also press Ctrl+N. A dialog box opens, in which you can select the wizard to create a new object. Click Message Brokers; a list of objects that you can create for WebSphere Message Broker is shown. Click **Message Flow ESQL File**, then click **Next**. The New Message Flow ESQL File wizard opens.
2. Enter the name of the message flow project in which to create the ESQL file.

   You must enter the name of an existing message flow project. The dialog box opens with the current project name entered in the project name field. You can accept this value or change it to specify a different project. You can also click **Browse** to view a list of valid projects (projects that are defined and shown in the Navigator view), and select the appropriate value from that list. The list is filtered to only show projects in the active working set.

   If you type in the name of a project that does not exist, the error message `The specified project does not exist` is shown in the dialog box and you cannot continue until you specify a valid project name.
3. If you want the ESQL file to be defined in a specific broker schema, enter the name of the broker schema in the appropriate entry field, or click **Browse** to

select the broker schema from the list of valid broker schema for this project. (If the default broker schema is the only one that is defined in this project, **Browse** is disabled.)

4. Enter a name for the new ESQL file. If you enter a name that is already in use for an ESQL file in this project, the error message `The resource <name>.esql already exists` is shown and you must specify a different name.

   When creating ESQL files, the overall file path length must not exceed 256 characters, due to a Windows file system limitation. If you try to add a message flow to a broker archive file with ESQL or mapping files with a path length that exceeds 256 characters, the compiled message flow is not generated and cannot be deployed. Therefore, make sure that the names of your ESQL files, mapping files, projects, and broker schema are as short as possible.
5. Click **Finish**.

Results

The ESQL file opens in the editor, where you can edit the file, then save it.

An ESQL file can also be created automatically for you. You can right-click a Compute, Database, DatabaseInput, or Filter node, then click **Open ESQL**. If the module identified by the appropriate property does not already exist in the broker schema, a module is created automatically. This module is created in the file `<message_flow_name>.esql` in the same broker schema in the same project as the `<message_flow_name>.msgflow` file. If that ESQL file does not already exist, that is also created for you.

The contents of a single ESQL file do not have any specific relationship with message flows and nodes. It is your decision which modules are created in which files (unless the specified module, identified by the appropriate property, is created by default in the file `<message_flow_name>.esql` as described above). Monitor the size and complexity of the ESQL in each file, and split the file if it becomes difficult to view or manage.

If you create reusable subroutines (at broker schema level) in an ESQL file, you might want to refer to these routines from ESQL modules in another project. To refer to the routines, specify that the project that runs the subroutines depends on the project in which the ESQL file containing them is defined. You can specify this behavior when you create the second project, or you can update project dependencies by selecting the project, clicking **Properties**, and updating the dependencies in the Project Reference page of the Properties dialog box.

Related concepts:

"Project references" on page 44
When a projects refers to other projects in the workspace, this is called a *project reference*. When one project references another, the files in the referenced project are available for use by the referring project. For example to configure certain nodes, to create libraries of reusable message flows, and to enable Content Assist in the ESQL editor.

"Development repository" on page 45
Use a development repository to benefit from features such as version control and access control of files, which make it easier for teams to work on shared resources.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

"Broker schemas" on page 1036
A broker schema is a symbol space that defines the scope of uniqueness of the names of resources defined within it. The resources include message flows and other optional resources such as ESQL files, Java files, and mapping files.

"ESQL modules" on page 2388
A module is a sequence of declarations that define variables and their initialization, and a sequence of subroutine (function and procedure) declarations that define a specific behavior for a message flow node.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

Opening an existing ESQL file:

You can add to and modify ESQL code that you have created in an ESQL file.

Before you begin

Before you start:

To complete this task, you must have created an ESQL file, as described in "Creating an ESQL file" on page 2391.

About this task

To open an existing ESQL file, complete the following steps.

Procedure

1. In the Broker Development view, double-click the ESQL file that you want to open. The file is opened in the editor view.
2. Edit the contents of file to make your changes. The file can contain modules that relate to specific nodes in a message flow, PATH statements, and declarations at broker schema level, such as reusable constants and procedures. You can select the content that you want to work with by selecting its name in the Outline view. The code for the selected resource is highlighted.
3. Save and close the ESQL file.

Results

You can also open an ESQL file when you have a message flow open in the editor view by right-clicking an appropriate node (a Compute, Database, DatabaseInput, or Filter node), then clicking **Open ESQL**. In this case, the ESQL file that contains

this module is opened, and the module for the selected node is highlighted in the editor view.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

"Broker schemas" on page 1036
A broker schema is a symbol space that defines the scope of uniqueness of the names of resources defined within it. The resources include message flows and other optional resources such as ESQL files, Java files, and mapping files.

"ESQL modules" on page 2388
A module is a sequence of declarations that define variables and their initialization, and a sequence of subroutine (function and procedure) declarations that define a specific behavior for a message flow node.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

Creating ESQL for a node:

Create ESQL code to customize the behavior of a Compute, Database, DatabaseInput, or Filter node in an ESQL file.

Before you begin

Before you start:

This topic assumes that you have created an ESQL file. For more information, see "Creating an ESQL file" on page 2391.

About this task

In the ESQL file, create a module that is associated with a node in your message flow. A module can be associated with only one node of a particular type (Compute, Database, DatabaseInput, or Filter). Within the module, you can create and use functions and procedures in addition to the supplied statements and functions. You can also create local constants and variables.

If you have created constants, functions, or procedures at the broker schema level, you can also refer to these in the module. You can define routines at a level at which many different modules can use them, which can save you development time and maintenance effort.

To create ESQL for a node, complete the following steps.

Procedure

1. In the Broker Development view, double-click the message flow that includes the node for which you want to create ESQL. The message flow opens in the editor view.
2. Right-click a Compute, Database, DatabaseInput, or Filter node, then click **Open ESQL**. The default ESQL file for this message flow, *message_flow_name*.esql, opens in the editor view. If the file does not already exist, it is created, containing a skeleton module for this node at the end. The exact content depends on the type of node.

   If you have already created the file, it is opened in the editor view and a new module is created and highlighted.

   The following module is created for a Compute node:

```
CREATE COMPUTE MODULE module_name
       CREATE FUNCTION Main() RETURNS BOOLEAN
       BEGIN
              -- CALL CopyMessageHeaders();
              -- CALL CopyEntireMessage();
              RETURN TRUE;
       END;

       CREATE PROCEDURE CopyMessageHeaders() BEGIN
              DECLARE I INTEGER 1;
              DECLARE J INTEGER CARDINALITY(InputRoot.*[]);
              WHILE I < J DO
                     SET OutputRoot.*[I] = InputRoot.*[I];
                     SET I = I + 1;
              END WHILE;
       END;

       CREATE PROCEDURE CopyEntireMessage() BEGIN
              SET OutputRoot = InputRoot;
       END;
END MODULE;
```

   The module name is determined by the value that you have set for the corresponding node property. The default name is *message_flow_name_node_type*. The main function contains calls to two procedures (described in the following list) that are declared in the Compute node module following the function Main. These calls are commented out. To include the function that they provide, uncomment the lines and place them at the appropriate point in the ESQL that you create for Main.

   CopyMessageHeaders
   : This procedure loops through the headers contained in the input message and copies each one to the output message.

   CopyEntireMessage
   : This procedure copies the entire contents of the input message, including the headers, to the output message.

   If you create an ESQL module for a Database node, the following module is created:

```
CREATE DATABASE MODULE module_name
       CREATE FUNCTION Main() RETURNS BOOLEAN
       BEGIN
              RETURN TRUE;
       END;
END MODULE;
```

For a DatabaseInput node, the module that is created contains three procedures, ReadEvents, BuildMessage, and EndEvent. Each of these procedures contains boilerplate text, which describes how the procedure works. For more information about configuring DatabaseInput nodes, see "Configuring a DatabaseInput node" on page 2120. For a DatabaseInput node, the following first line of the module is created:

```
CREATE DATABASEEVENT MODULE module_name
```

For a Filter node, the module is identical to the module created for the Database node, except for the first line, which reads:

```
CREATE FILTER MODULE module_name
```

3. Add ESQL to this file to customize the behavior of the node.

   For Compute, Database, or Filter nodes, start by adding ESQL statements in the Main function, (after the BEGIN statement, and before RETURN TRUE). For DatabaseInput nodes, add ESQL statements in the ReadEvents, BuildMessage, and EndEvent procedures. You can add DECLARE statements in the module that are not within the Main function. To add a new line into the file, press Enter.

   To help you to code valid ESQL, the editor shows a list of valid statements and functions at the point of the cursor. To start this assistance, click **Edit** > **Content Assist**. On some systems, you can use the key combination Ctrl+Space. Scroll through the list to find and highlight the statement or function that you want, and press Enter. The appropriate code is inserted into your module.

   Content assistance is provided in the following areas:

   - Applicable keywords, based on language syntax.
   - Blocks of code that go together, such as `BEGIN END;`.
   - Constants that you have defined, identifiers, labels, functions, and procedures that can be used, where the routines can be in any projects, even if the current project does not reference them.
   - Database schema and table names after the database correlation name, table column names in INSERT, UPDATE, DELETE, and SELECT statements, and, in most cases, the WHERE clauses of those statements.
   - Elements of message field reference: runtime domain (parser) names, format of type expression, namespace identifiers, namespace-qualified element and attribute names, and format of index expression.
   - Content in the Properties folder under the output message.
   - For the DECLARE NAMESPACE statement, target namespaces of message sets and schema names.

   Content assistance works only if the ESQL can be parsed correctly. Errors such as END missing after BEGIN, and other unterminated block statements, cause parser failures and no content assistance is provided. Try content assistance in other areas around the statement where it does not work to narrow down the point of error. Alternatively, save the ESQL file; saving the file causes validation, and all syntax errors are written to the Problems view. Refer to the errors reported to understand and correct the ESQL syntax. If you use content assistance to generate most statements (such as block statements), these statements are correctly entered and there is less opportunity for error.
4. When you have finished working with this module, save and close the ESQL file.

Results

You can also open the ESQL file directly and create the module in that file by using the editor:

1. Open the ESQL file in which you want to create the module.
2. In the editor view, position your cursor on a new line and use content assistance to select the appropriate module skeleton for this type of node, for example `CREATE COMPUTE MODULE END MODULE;`. You can also type in this text, but you must ensure that what you type is consistent with the required skeleton, shown earlier. Use content assistance to give you additional help by inserting only valid ESQL, and by inserting matching end statements (for example, `END MODULE;`) where they are required.
3. Complete the coding of the module as appropriate.

Whichever method you use to open the ESQL file, be aware that the editor provides functions to help you to code ESQL. This section refers to content assistance; other functions are available. For information about these functions, see "ESQL editor" on page 6798.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Broker schemas" on page 1036
A broker schema is a symbol space that defines the scope of uniqueness of the names of resources defined within it. The resources include message flows and other optional resources such as ESQL files, Java files, and mapping files.

"ESQL modules" on page 2388
A module is a sequence of declarations that define variables and their initialization, and a sequence of subroutine (function and procedure) declarations that define a specific behavior for a message flow node.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Modifying ESQL for a node" on page 2398
To change the customization of a node that requires ESQL (Compute, Database, DatabaseInput, or Filter node), modify the ESQL statements in the module that you created for that node.

"Changing ESQL editor settings" on page 2409
When you open an ESQL file in the editor view, you can tailor the editor appearance by changing editor settings.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"ESQL editor" on page 6798
The ESQL editor is the default editor provided by the Broker Application Development perspective for editing ESQL (`.esql`) files.

Modifying ESQL for a node:

To change the customization of a node that requires ESQL (Compute, Database, DatabaseInput, or Filter node), modify the ESQL statements in the module that you created for that node.

Before you begin

Before you start:

This topic assumes that you have created ESQL code for a file. For more information, see "Creating ESQL for a node" on page 2394.

About this task

To modify ESQL code, complete the following steps.

Procedure

1. In the Broker Development view, select the message flow that you want to work with and double-click it. The message flow is opened in the editor view.
2. Right-click the node that corresponds to the ESQL module that you want to modify and click **Open ESQL**. The ESQL file is opened in the editor view. The module for this node is highlighted.
3. Make the required changes in the module by entering new statements, changing existing statements by over-typing, or deleting statements by using the Delete or backspace keys. You can use Content Assist, available from the **Edit** menu or, on some systems, by pressing Ctrl+Space). To get Content Assist to work with message references, you must set up a project reference from the project containing the ESQL to the project containing the message set. For information about setting up a project reference, see "Project references" on page 44.
4. You can change the name of the module with which you are working by over-typing the current name with the new one. In this case, you must also change the node property `ESQL Module` to reflect the new name, to ensure that the correct ESQL code is deployed with the node.
5. When you have finished working with this module, save and close the ESQL file.

Results

You can also open the ESQL file directly by double-clicking it in the Broker Development view. You can select the module with which you want to work from the Outline view.

The editor provides functions that you can use to help you modify your ESQL code. These functions are described in "ESQL editor" on page 6798.

You can also modify the ESQL source by clicking **Source** > **Format**. This option formats all selected lines of code (unless only partially selected, when they are ignored), or, if no lines are selected, formats the entire file (correcting alignments and indentation).

Adding comments to ESQL:

About this task

You can add and remove comments in your ESQL code.

Procedure

- To change an existing line of code into a comment line, click **Source** > **Comment**.
- To change a comment line to a code line, click **Source** > **Uncomment**.
- To create a new comment line, press Enter to create a new line, then either type the comment identifier `--` or click **Source** > **Comment**. You can enter any text after the identifier; everything that you type is ignored by the ESQL editor.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL modules" on page 2388
A module is a sequence of declarations that define variables and their initialization, and a sequence of subroutine (function and procedure) declarations that define a specific behavior for a message flow node.

"Project references" on page 44
When a projects refers to other projects in the workspace, this is called a *project reference*. When one project references another, the files in the referenced project are available for use by the referring project. For example to configure certain nodes, to create libraries of reusable message flows, and to enable Content Assist in the ESQL editor.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Creating ESQL for a node" on page 2394
Create ESQL code to customize the behavior of a Compute, Database, DatabaseInput, or Filter node in an ESQL file.

"Saving an ESQL file"
When you edit an ESQL file, you can save it to preserve the additions and modifications that you have made, and to force the editor to validate the content of the file.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"ESQL editor" on page 6798
The ESQL editor is the default editor provided by the Broker Application Development perspective for editing ESQL (`.esql`) files.

Saving an ESQL file:

When you edit an ESQL file, you can save it to preserve the additions and modifications that you have made, and to force the editor to validate the content of the file.

Before you begin

Before you start:

This task assumes that you have created or opened an ESQL file. For more information, see “Creating an ESQL file” on page 2391 or “Opening an existing ESQL file” on page 2393.

About this task

To save an ESQL file, complete the following steps.

Procedure

1. Change the contents of the ESQL file.
2. Save the file by clicking **File** > **Save** or **File** > **Save All**.

   When you save the file, the validator is called by the editor to check that the ESQL obeys all grammar and syntax rules (specified by the syntax diagrams and explanations in “ESQL reference” on page 5019).

   You can request additional validation when you set ESQL preferences:

   a. Click **Window** > **Preferences**. The Preferences dialog box is displayed.
   b. Expand **Broker Development** and **ESQL**, then click **Validation**.
   c. Select the level of validation that you require for each category of error:
      1) Unresolved identifiers
      2) Message references do not match message definitions
      3) Database references do not match database schema
      4) Use of deprecated keywords

      The default level is *warning*; you can change this value to *error* or *ignore*.

      Validating message definitions can affect response times in the editor, particularly if you have complicated ESQL that makes many references to a complex message definition. You might choose to delay this validation. Call validation when you have finished developing the message flow and are about to deploy it, to avoid runtime errors. For each error found, the editor writes the code line number and the reason for the error; errors are created as entries in the Problems view.
3. If you double-click the error, the editor positions your cursor on the line in which it found that error. The line is also highlighted by the error icon  . The editor might also find potential error situations, which it highlights as warnings (with the warning icon ); the editor also writes these warnings to the Problems view. For example, you might have included a `BROKER SCHEMA` statement that references an invalid schema (namespace).

   Check your code, and make the corrections required by that statement or function.

Save As:
About this task

You can save a copy of this ESQL file by using **File** > **Save As**.

Procedure

1. Click **File** > **Save As**.

2. Specify the message flow project in which you want to save a copy of the ESQL file. The project name defaults to the current project. You can accept this name, or choose another name from the valid options that are displayed in the File Save dialog box.
3. Specify the name for the new copy of the ESQL file. To save this ESQL file in the same project, either rename it, or confirm that you want to overwrite the current copy (that is, copy the file to itself).

   To save this ESQL file in another project, the project must exist. You can save the file with the same or another name in another project.
4. Click **OK**. The message flow is saved and the message flow editor validates its contents. The editor provides a report of all errors that it finds in the Problems view.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

"Broker schemas" on page 1036
A broker schema is a symbol space that defines the scope of uniqueness of the names of resources defined within it. The resources include message flows and other optional resources such as ESQL files, Java files, and mapping files.

"ESQL modules" on page 2388
A module is a sequence of declarations that define variables and their initialization, and a sequence of subroutine (function and procedure) declarations that define a specific behavior for a message flow node.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

Copying an ESQL file:

You can copy an ESQL file as a starting point for a new ESQL file that has similar function.

Before you begin

Before you start:

This topic assumes that you have created an ESQL file. For more information, see "Creating an ESQL file" on page 2391.

About this task

To copy an ESQL file, complete the following steps.

Procedure

1. In the Broker Development view, right-click the ESQL file that you want to copy, and click **Copy**.

   Alternatively, you can select the ESQL file and click **Edit** > **Copy**.
2. Right-click the broker schema in the message flow project to which you want to copy the ESQL file and click **Paste**. You can copy the ESQL file to the same broker schema within the same message flow project, or to a different broker schema within the same message flow project, or to a broker schema in a different message flow project.

   When you copy an ESQL file, the associated files (message flow, and mapping if present) are not automatically copied to the same target message flow project. If you want these files copied as well, you must do this explicitly following this procedure.

   If you want to use this ESQL file with another message flow, ensure that the modules within the ESQL file match the nodes that you have in the message flow, and that the node properties are set correctly.

Results

You can also copy an ESQL file by using **File** > **Save As**; for more information, see "Saving an ESQL file" on page 2399.

Related concepts:

"Project references" on page 44
When a projects refers to other projects in the workspace, this is called a *project reference*. When one project references another, the files in the referenced project are available for use by the referring project. For example to configure certain nodes, to create libraries of reusable message flows, and to enable Content Assist in the ESQL editor.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL modules" on page 2388
A module is a sequence of declarations that define variables and their initialization, and a sequence of subroutine (function and procedure) declarations that define a specific behavior for a message flow node.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Creating ESQL for a node" on page 2394
Create ESQL code to customize the behavior of a Compute, Database, DatabaseInput, or Filter node in an ESQL file.

"Saving an ESQL file" on page 2399
When you edit an ESQL file, you can save it to preserve the additions and modifications that you have made, and to force the editor to validate the content of the file.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your

message flows.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

Analyzing planned changes to ESQL objects:

Use impact analysis to analyze the effect of renaming or moving ESQL objects, or moving or changing the broker schema of an ESQL file.

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message flow project" on page 1425
- "Creating an ESQL file" on page 2391
- "Enabling and disabling indexing" on page 1454

About this task

This information covers the following tasks:

- "Renaming ESQL objects"
- "Moving ESQL objects" on page 2404
- "Moving an ESQL file" on page 2404

If a node in the message flow contains either an ESQL path expression, or an XPath expression, it might not be possible to unambiguously determine which structure in the message set the expression refers to. Ambiguity occurs when multiple constructs have the same name, and the parent of the construct cannot be determined. For example, consider a message definition that has the following constructs:

Global element declaration with name G1

Global element declaration with name G2

Global element declaration with name G3

G2 contains an element reference to global element G1

G3 contains a local element declaration with name G1

An ESQL path is in the MRM domain as follows:

```
InputRoot.MRM.ns1:G1
```

The construct G1 cannot be unambiguously determined. It can refer to G2/G1 or G3/G1. In this case, impact analysis reports both G2/G1 and G3/G1 as potential impacts. The list of secondary impacts might contain references to elements that are not in fact affected by the primary change.

Renaming ESQL objects:
About this task

You can rename the following objects:

- ESQL modules

- Schema-scope ESQL routines
- Schema-scope ESQL constants

Procedure

1. In the Broker Development view, right-click the object that you want to rename, then click **Impact Analysis** > **Rename**.
2. In the Impact Analysis - Rename Artifact window, type the new name of the object, then click **Analyze Impact**.

   The Rename Artifact dialog box shows the results of impact analysis, listing primary and secondary impacts.

   You can view the results of impact analysis in the Impact Analysis dialog box, or the Impact Analysis view of the WebSphere Message Broker Toolkit. For more information about how to view information about selected resources, mark changes as complete, copy results to an external application, and view previous results, see "Impact Analysis view" on page 6801.

Moving ESQL objects:
About this task

You can move the following objects to a different file:

- ESQL modules
- Schema-scope ESQL constants
- Schema-scope ESQL routines

Procedure

1. Right-click the object that you want to move, then click **Impact Analysis** > **Move**. The Impact Analysis - Move Artifact window is displayed.
2. In the bottom panel, under 'Choose the new container for the selected items', select the new ESQL file for the object, then click **Analyze Impact**.

   The Impact Analysis - Move Artifact dialog box shows the results of impact analysis, listing primary and secondary impacts.

   You can view the results of impact analysis in the Impact Analysis dialog box, or the Impact Analysis view of the WebSphere Message Broker Toolkit. For more information about how to view information about selected resources, mark changes as complete, copy results to an external application, and view previous results, see "Impact Analysis view" on page 6801.

Moving an ESQL file:
Procedure

1. In the Broker Development view, right-click the file, then click **Impact Analysis** > **Move**.
2. In the bottom panel, select the new broker schema for the object, then click **Analyze Impact**.

   The Impact Analysis - Move Artifact dialog box shows the results of impact analysis, listing primary and secondary impacts.

   You can view the results of impact analysis in the Impact Analysis dialog box, or the Impact Analysis view of the WebSphere Message Broker Toolkit. For more information about how to view information about selected resources, mark changes as complete, copy results to an external application, and view previous results, see "Impact Analysis view" on page 6801.

Related concepts:

"Impact analysis: analyzing the effects of planned changes to your applications" on page 1150
When developing an application, you might want to change the names of some artifacts, or move them. Impact analysis reports the artifacts that are likely to be affected by a particular change.

Related tasks:

"Enabling and disabling indexing" on page 1454
Enable indexing to support impact analysis.

"Analyzing planned changes to message model objects" on page 2897
Use impact analysis to analyze the effect of renaming message model objects.

"Analyzing the impact of changes to message maps" on page 2234
Use impact analysis to analyze the effect of renaming or moving message maps.

"Analyzing planned changes to message flows" on page 1436
Use impact analysis to analyze the effect of renaming or moving message flows, including subflows.

"Analyzing planned changes to message set resources" on page 1165
Use impact analysis to analyze the effect of renaming message definition or deployable WSDL files.

Related reference:

"Impact analysis: reference" on page 4174
Some artifacts are excluded from secondary analysis.

"Impact Analysis view" on page 6801
The Impact Analysis view shows information about selected resources, which changes have been marked as complete, and previous results. You can also use this view to copy results to the clipboard.

Renaming an ESQL file:

You might want to rename an ESQL file if you have renamed the message flow with which it is associated.

Before you begin

Before you start:

This task assumes that you have created an ESQL file, as described in "Creating an ESQL file" on page 2391.

To find out which other objects are likely to be affected by the change, see "Analyzing planned changes to ESQL objects" on page 2403.

About this task

To rename an ESQL file, complete the following steps.

Procedure

1. In the Broker Development view, right-click the ESQL file that you want to rename and click **Rename**. Alternatively, you can select the ESQL file, then click **File** > **Rename**.

   The Rename Resource dialog box opens.
2. Enter the new name for the ESQL file, then click **OK**.

   The ESQL file is renamed.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

"Broker schemas" on page 1036
A broker schema is a symbol space that defines the scope of uniqueness of the names of resources defined within it. The resources include message flows and other optional resources such as ESQL files, Java files, and mapping files.

"ESQL modules" on page 2388
A module is a sequence of declarations that define variables and their initialization, and a sequence of subroutine (function and procedure) declarations that define a specific behavior for a message flow node.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

Moving an ESQL file:

If you move a message flow from one broker schema to another, or from one project to another, you might want to move all ESQL files that are associated with that message flow.

Before you begin

Before you start

To complete this task, you must have completed the following task:
- "Creating an ESQL file" on page 2391

About this task

To move an ESQL file:

Procedure

1. Move the ESQL file in one of the following ways:
   - Drag the ESQL file that you want to move from its current location to a broker schema within the same or another message flow project.

     If the target location that you have chosen is not valid (for example, if an ESQL file of this name exists in the broker schema), the invalid icon is displayed and the move is not completed. If you have created an empty broker schema for this purpose, it might not be visible in the Broker

Development view if category mode is selected. To see an empty schema in the Broker Development view, click Hide Categories  .

- Right-click the ESQL file and click **Move**, or click **File** > **Move**. The Move dialog is displayed.

  Select the project and the broker schema from the list of valid targets that is shown in the dialog.

  Click **OK** to complete the move, or **Cancel** to cancel the request.

  If you click **OK**, the ESQL file is moved to its new location.

2. Check the Problems view for errors (indicated by the error icon ) or warnings (indicated by the warning icon ) generated by the move.

   The errors in the Problems view include those errors caused by broken references. When the move is completed, all references to this ESQL file are checked. If you have moved the file within the same named broker schema within the same message flow project, all references are still valid. If you have moved the file to another broker schema in the same or another message flow project, the references are broken. If you have moved the file to the same named broker schema in another message flow project, the references might be broken if the project references are not set correctly to recognize external references in this file. These errors occur because resources are linked by a fully qualified name.

3. Double-click each error or warning to open the message flow that has the error in the editor view, and highlight the node in error. You can now correct the error.

Results

When you move an ESQL file, its associated files (for example, the message flow file) are not automatically moved to the same target broker schema. You must move these files yourself.

Related concepts:

"Project references" on page 44
When a projects refers to other projects in the workspace, this is called a *project reference*. When one project references another, the files in the referenced project are available for use by the referring project. For example to configure certain nodes, to create libraries of reusable message flows, and to enable Content Assist in the ESQL editor.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

"Broker schemas" on page 1036
A broker schema is a symbol space that defines the scope of uniqueness of the names of resources defined within it. The resources include message flows and other optional resources such as ESQL files, Java files, and mapping files.

"ESQL modules" on page 2388
A module is a sequence of declarations that define variables and their initialization, and a sequence of subroutine (function and procedure) declarations that define a specific behavior for a message flow node.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

Changing ESQL preferences:

You can modify the way in which ESQL is displayed in the editor and validated by the editor:

Procedure

- "Changing ESQL editor settings" on page 2409
- "Changing ESQL validation settings" on page 2410

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

"Broker schemas" on page 1036
A broker schema is a symbol space that defines the scope of uniqueness of the names of resources defined within it. The resources include message flows and other optional resources such as ESQL files, Java files, and mapping files.

"ESQL modules" on page 2388
A module is a sequence of declarations that define variables and their initialization, and a sequence of subroutine (function and procedure) declarations that define a specific behavior for a message flow node.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"Broker Application Development perspective" on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

"ESQL editor" on page 6798
The ESQL editor is the default editor provided by the Broker Application Development perspective for editing ESQL (`.esql`) files.

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

Changing ESQL editor settings:

When you open an ESQL file in the editor view, you can tailor the editor appearance by changing editor settings.

Procedure

1. **To change ESQL editor settings:**
   a. Click **Window** > **Preferences**. The Preferences dialog box opens.
   b. Expand the item for ESQL on the left and click **ESQL Editor**.
   c. Update the settings available for tab width and colors:
      - Click the **General** tab to change the displayed tab width within the ESQL editor.
      - Click the **Colors** tab to change the color of the editor view background, and of the entities displayed in the editor view. These include comments and keywords in your ESQL code.
   d. When you have completed your changes, click either **Apply** (to apply your changes and leave the Preferences dialog box open), or **OK** (to apply your changes and close the dialog box). Alternatively, to close the dialog box and discard your changes, click **Cancel**.
   e. To return the ESQL editor settings to the initial values, click **Restore Defaults**. All values are reset to the original settings.

      If you change the editor settings when you have an editor session active, the changes are implemented immediately. If you do not have an editor session open, you see the changes when you next edit an ESQL file.
2. **To change font settings for the ESQL editor:**
   a. Click **Window** > **Preferences**. The Preferences dialog box opens.
   b. Expand the item for Workbench on the left of the Preferences dialog box, and click **Colors and Fonts**.
   c. On the **Colors and Fonts** tab, expand Basic.
   d. Select a font or text color option and click **Change**. The Font dialog box opens.
   e. When you have completed your changes, click either **Apply** (to apply your changes and leave the Preferences dialog box open) or **OK** (to apply your changes and close the dialog box). Alternatively, to close the dialog box and discard your changes, click **Cancel**.
   f. To return the ESQL editor settings to the initial values, click **Restore Defaults**.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

"Broker schemas" on page 1036
A broker schema is a symbol space that defines the scope of uniqueness of the names of resources defined within it. The resources include message flows and other optional resources such as ESQL files, Java files, and mapping files.

"ESQL modules" on page 2388
A module is a sequence of declarations that define variables and their initialization, and a sequence of subroutine (function and procedure) declarations that define a

specific behavior for a message flow node.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"ESQL editor" on page 6798
The ESQL editor is the default editor provided by the Broker Application Development perspective for editing ESQL (`.esql`) files.

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

Changing ESQL validation settings:

You can specify the level of validation that the ESQL editor performs when you save a `.esql` file. If the validation you have requested results in warnings, you can deploy a BAR file containing this message flow. However, if errors are reported, you cannot deploy the BAR file.

About this task

To change ESQL validation settings:

Procedure

1. Switch to the Broker Application Development perspective.
2. Click **Window** > **Preferences**. The Preferences dialog is displayed.
3. Expand the item for ESQL on the left and click Validation.
4. Update the settings for what is validated, and for what warnings or errors are reported. See "ESQL editor" on page 6798 for details of the settings and their values.
5. When you have completed your changes, click **Apply** to close the Preferences dialog, apply your changes and leave the Preferences dialog open. Click **OK** to apply your changes and close the dialog. Click **Cancel** to close the dialog and discard your changes.
6. If you want to return your ESQL editor preferences to the initial values, click **Restore Defaults**. All values are reset to the original settings.

Results

If you make changes to the validation settings, the changes are implemented immediately for currently open edit sessions and for subsequent edit sessions.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

"Broker schemas" on page 1036
A broker schema is a symbol space that defines the scope of uniqueness of the names of resources defined within it. The resources include message flows and other optional resources such as ESQL files, Java files, and mapping files.

"ESQL modules" on page 2388
A module is a sequence of declarations that define variables and their initialization, and a sequence of subroutine (function and procedure) declarations that define a specific behavior for a message flow node.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"Broker Application Development perspective" on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

"ESQL editor" on page 6798
The ESQL editor is the default editor provided by the Broker Application Development perspective for editing ESQL (`.esql`) files.

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

Deleting ESQL for a node:

If you delete a node from a message flow, you can delete the ESQL module that you created to customize its function.

Before you begin

Before you start

This topic assumes that you have created ESQL code for a node. For more information, see "Creating ESQL for a node" on page 2394.

About this task

To delete ESQL code, complete the following steps.

Procedure

1. Open the message flow with which you want to work by double-clicking it in the Broker Development view. The message flow is opened in the editor view.
2. Right-click the node for which you want to delete the ESQL module, then click **Open ESQL**. The ESQL file is opened in the editor view, with the module for this node highlighted.
3. Press the Delete or backspace key to delete the whole module.
4. When you have finished working with this module, save and close the ESQL file. Saving the file also validates the ESQL; for more details, see "Saving an ESQL file" on page 2399.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

"Broker schemas" on page 1036
A broker schema is a symbol space that defines the scope of uniqueness of the names of resources defined within it. The resources include message flows and other optional resources such as ESQL files, Java files, and mapping files.

"ESQL modules" on page 2388
A module is a sequence of declarations that define variables and their initialization, and a sequence of subroutine (function and procedure) declarations that define a specific behavior for a message flow node.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

Deleting an ESQL file:

If you delete a message flow, or if you have deleted all the ESQL code in an ESQL file, you can delete the ESQL file.

Before you begin

Before you start:

This task assumes that you have created an ESQL file. For more information, see "Creating an ESQL file" on page 2391.

About this task

To delete an ESQL file, complete the following steps.

Procedure

1. In the Broker Development view, right-click the ESQL file that you want to delete, and click **Delete**.

   You can also select the file in the Broker Development view, and click **Edit** > **Delete**. A dialog box is displayed that asks you to confirm the deletion.
2. Click **Yes** to delete the file, or **No** to cancel the delete request.

   If you maintain resources in a shared repository, a copy is retained in that repository. You can follow the instructions provided by the repository supplier to retrieve the file if required.

If you are using the local file system or a shared file system to store your resources, no copy of the file is retained. Be careful to select the correct file when you complete this task.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

"Broker schemas" on page 1036
A broker schema is a symbol space that defines the scope of uniqueness of the names of resources defined within it. The resources include message flows and other optional resources such as ESQL files, Java files, and mapping files.

"ESQL modules" on page 2388
A module is a sequence of declarations that define variables and their initialization, and a sequence of subroutine (function and procedure) declarations that define a specific behavior for a message flow node.

"Development repository" on page 45
Use a development repository to benefit from features such as version control and access control of files, which make it easier for teams to work on shared resources.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

Writing ESQL

How you can use ESQL to customize nodes.

About this task

When you create a message flow, you include input nodes that receive the messages and, optionally, output nodes that send out new or updated messages. If required by the processing that must be performed on the message, you can include other nodes after the input node that complete the actions that your applications need.

Some of the built-in nodes enable you to customize the processing that they provide. The Compute, Database, DatabaseInput, and Filter nodes require you to provide a minimum level of ESQL, and you can provide much more than the minimum to control precisely the behavior of each node. This set of topics discusses ESQL and the ways in which you can use it to customize these nodes.

The DataDelete, DataInsert, DataUpdate, Extract, Mapping, and Warehouse nodes provide a mapping interface with which you can customize their function. The

ways in which you can use the mapping functions associated with these nodes are described in developing message mappings, see “Using message mappings” on page 2228.

ESQL provides a rich and flexible syntax for statements and functions that enable you to check and manipulate message and database content. You can:

- Read the contents of the input message
- Modify message content with data from databases
- Modify database content with data from messages
- Construct new output messages created from all, part, or none of the input message (in the Compute node only)

The following topics provide more information about these and other tasks that you can perform with ESQL. Unless otherwise stated, these guidelines apply to messages in all message domains except the BLOB domain, for which you can implement a limited set of actions.

- “Tailoring ESQL code for different node types” on page 2416
- “Manipulating message body content” on page 2418
- “Manipulating other parts of the message tree” on page 2452
- “Transforming from one data type to another” on page 2473
- “Adding keywords to ESQL files” on page 2486
- “Interaction with databases using ESQL” on page 2487
- “Coding ESQL to handle errors” on page 2506
- “Accessing broker properties from ESQL” on page 2625
- “Configuring a message flow at deployment time with user-defined properties” on page 2626

The following topics provide additional information specific to the parser that you have specified for the input message:

- “Manipulating messages in the MRM domain” on page 2581
- “Manipulating messages in the XML domain” on page 2581
- “Manipulating messages in the XMLNS domain” on page 2563
- “Manipulating messages in the XMLNSC domain” on page 2546
- “Manipulating messages in the JMS domains” on page 2609
- “Manipulating messages in the IDOC domain” on page 2610
- “Manipulating messages in the MIME domain” on page 2612
- “Manipulating messages in the BLOB domain” on page 2615
- “Manipulating messages in the JSON domain” on page 2617

ESQL examples:
About this task

Most of the examples included in the topics listed previously show parser-independent ESQL. If examples include a reference to MRM, they assume that you have modeled the message in the MRM and that you have set the names of the MRM objects to be identical to the names of the corresponding tags or attributes in the XML source message. Some examples are also shown for the XML domain. Unless stated otherwise, the principals illustrated are the same for all message domains. For domain-specific information, use the appropriate link in the previous list.

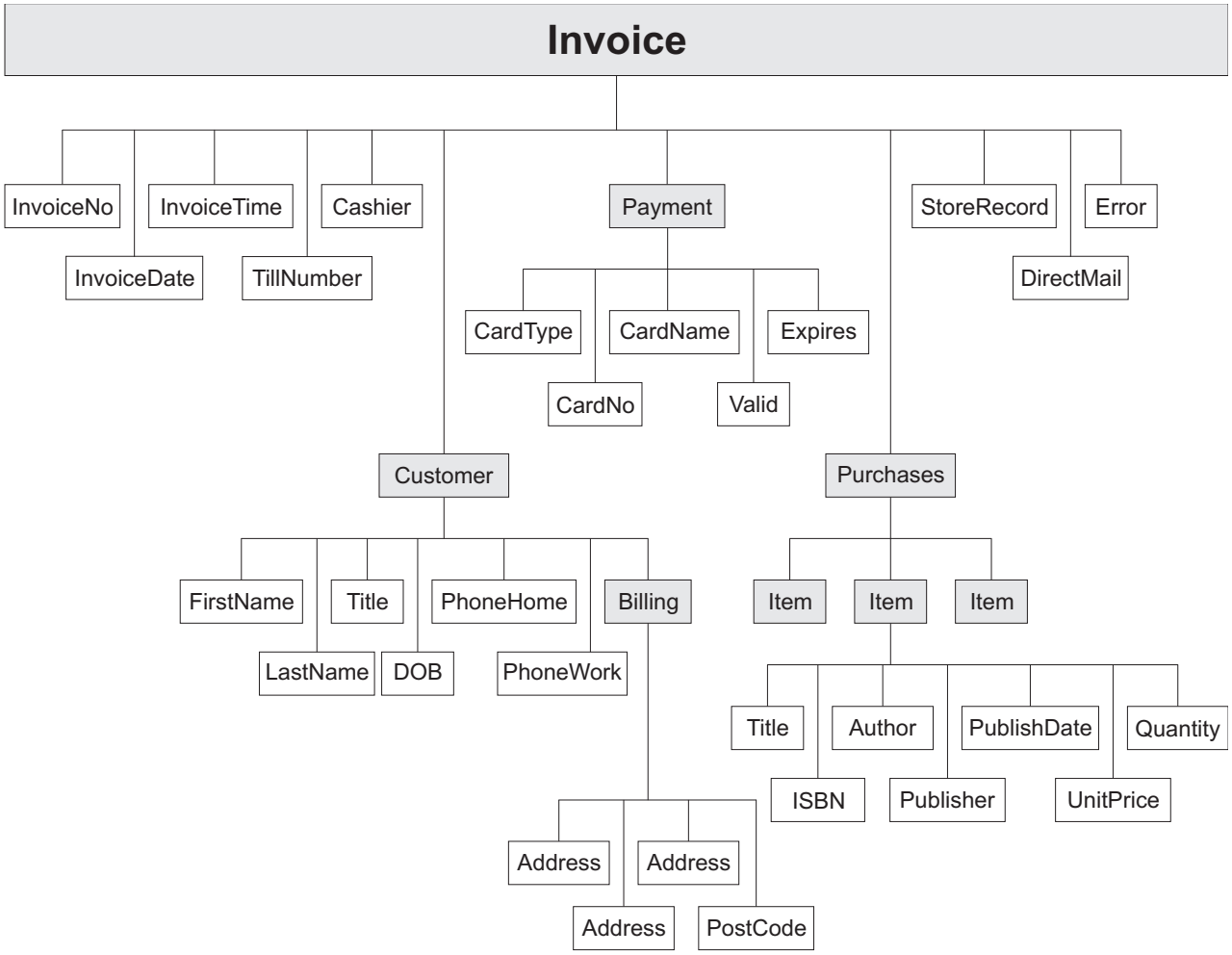
Most of the topics that include example ESQL use the ESQL sample message, Invoice, as the input message to the logic. This message is provided in XML source format (with tags and attributes), see “Example message” on page 5311. The example message is shown in the following diagram.

The topics specific to the MRM domain use the message that is created in the following sample:

- Video Rental

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

A few other input messages are used to show ESQL that provides function on messages with a structure or content that is not included in the Invoice or Video samples. Where this occurs, the input message is included in the topic that refers to it.



Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Using message mappings" on page 2228
Message mappings define the blueprint for creating a message.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"Example message" on page 5311

Tailoring ESQL code for different node types:

When you code ESQL to configure Compute, Database, DatabaseInput, and Filter node behavior, be aware of the limitations of each type of node.

About this task

Compute node

You can configure the Compute node to do any of the following operations:

- Update data in a database.
- Insert data into a database.
- Delete data from a database.
- Update the environment tree.
- Update the local environment tree.
- Create one or more output messages, with none, some, or all the content of the input message, and propagate these new messages to the next node in the message flow.

  To propagate the input LocalEnvironment to the output LocalEnvironment, remember to set the Compute node property `Compute mode` to an appropriate value. The Environment is always propagated in the output message.

Database node

You can configure the Database node to do any of the following operations:

- Update data in a database.
- Insert data into a database.
- Delete data from a database.
- Update the environment tree.
- Update the local environment tree.
- Propagate the input message to the next node in the message flow.

DatabaseInput node

You can configure the DatabaseInput node to do any of the following operations:

- Respond to updates that are made to the data in a database.
- Create one or more output messages, with none, some, or of all the content retrieved from a database, and propagate these new messages to the next node in the message flow.

Filter node

You can configure the Filter node to do any of the following operations:

- Update data in a database.
- Insert data into a database.
- Delete data from a database.
- Update the environment tree.
- Update the local environment tree.
- Propagate the input message to the next node in the message flow (the terminal through which the message is propagated depends on the result of the filter expression).

View the remaining tasks in this section to find the details of how you can perform these operations.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"DatabaseInput node" on page 4360
Use the DatabaseInput node to detect events recorded in a database, and to retrieve the data affected by those events.

"Filter node" on page 4452
Use the Filter node to route a message according to message content.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

Manipulating message body content:

The message body is always the last child of root, and is identified by its parser name, for example XML or MRM. You can refer to, modify, and create message body data.

About this task

The following topics describe how you can refer to, modify, and create message body data. The information provided here is domain independent.

- "Referencing field types" on page 2419
- "Accessing elements in the message body" on page 2420
- "Accessing known multiple occurrences of an element" on page 2425
- "Accessing unknown multiple occurrences of an element" on page 2427
- "Using anonymous field references" on page 2430
- "Creating dynamic field references" on page 2431
- "Creating new fields" on page 2434
- "Generating multiple output messages" on page 2437
- "Using numeric operators with datetime values" on page 2439
- "Calculating a time interval" on page 2441
- "Selecting a subfield from a larger field" on page 2442
- "Copying repeating fields" on page 2444
- "Manipulating repeating fields in a message tree" on page 2450

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"DatabaseInput node" on page 4360
Use the DatabaseInput node to detect events recorded in a database, and to retrieve the data affected by those events.

"Filter node" on page 4452
Use the Filter node to route a message according to message content.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

Referencing field types:

Some message parsers have complex models in which it is not enough to identify a field by its name and an array subscript. In these cases, you associate an optional field type with an element of data in the tree format.

About this task

Each element within the parsed tree can be one of three types:

Name element
A name element has a string, which is the name of the element, associated with it. An example of a name element is `XMLElement`, described in "XML element" on page 4267.

Value element
A value element has a value associated with it. An example of a value element is `XMLContent`, described in "XML content" on page 4267.

Name-value element
A name-value element is an optimization of the case where a name element contains only a value element and nothing else. The element contains both a name and a value. An example of a name-value element is `XMLAttribute`, described in "XML attribute" on page 4264.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"DatabaseInput node" on page 4360
Use the DatabaseInput node to detect events recorded in a database, and to retrieve the data affected by those events.

"Filter node" on page 4452
Use the Filter node to route a message according to message content.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

Accessing elements in the message body:

When you want to access the contents of a message, for reading or writing, use the structure and arrangement of the elements in the tree that is created by the parser from the input bit stream.

About this task

Follow the relevant parent and child relationships from the top of the tree downwards, until you reach the required element.

- If you are referring to the input message tree to interrogate its content in a Compute node, use correlation name `InputBody` followed by the path to the element to which you are referring. `InputBody` is equivalent to `InputRoot` followed by the parser name (for example, `InputRoot.MRM`), which you can use if you prefer.

- If you are referring to the output message tree to set or modify its content in the Compute node, use correlation name `OutputRoot` followed by the parser name (for example, `OutputRoot.MRM`).
- If you are referring to the input message to interrogate its contents in a Database or Filter node, use correlation name `Body` to refer to the start of the message. `Body` is equivalent to `Root` followed by the parser name (for example, `Root.XMLNS`), which you can use if you prefer. You cannot use the `Body` correlation name in a DatabaseInput node.

  You must use these different correlation names because there is only one message to which to refer in a Database or Filter node; you cannot create an output message in these nodes. Use a Compute node to create an output message.

When you construct field references, the names that you use must be valid ESQL identifiers that conform to ESQL rules. If you enclose an item in double quotation marks, ESQL interprets it as an identifier. If you enclose an item in single quotation marks, ESQL interprets it as a character literal. You must enclose all strings (character strings, byte strings, or binary (bit) strings) in quotation marks, as shown in the following examples. To include a single or double quotation mark within a string, include two consecutive single or double quotation marks.

Important: For a full description of field reference syntax, see "ESQL field reference overview" on page 5049.
For more information about ESQL data types, see "ESQL data types in message flows" on page 5020.

Assume that you have created a message flow that handles the message Invoice, shown in the figure in "Writing ESQL" on page 2413. If, for example, you want to interrogate the element `CardType` from within a Compute node, use the following statement:

```
IF InputBody.Invoice.Payment.CardType='Visa' THEN
   DO;
     -- more ESQL --
END IF;
```

If you want to make the same test in a Database or Filter node (where the reference is to the single input message), code:

```
IF Body.Invoice.Payment.CardType='Visa' THEN
   DO;
     -- more ESQL --
END IF;
```

If you want to copy an element from an input XML message to an output message in the Compute node without changing it, use the following ESQL:

```
SET OutputRoot.XMLNS.Invoice.Customer.FirstName =
               InputBody.Invoice.Customer.FirstName;
```

If you want to copy an element from an input XML message to an output message and update it, for example by folding to uppercase or by calculating a new value, code:

```
SET OutputRoot.XMLNS.Invoice.Customer.FirstName =
               UPPER(InputBody.Invoice.Customer.FirstName);
SET OutputRoot.XMLNS.Invoice.InvoiceNo = InputBody.Invoice.InvoiceNo + 1000;
```

If you want to set a STRING element to a constant value, code:

```
SET OutputRoot.XMLNS.Invoice.Customer.Title = 'Mr';
```

You can also use the equivalent statement:

```
SET OutputRoot.XMLNS.Invoice.Customer.Title VALUE = 'Mr';
```

If you want to update an INTEGER or DECIMAL, for example the element `TillNumber`, with the value `26`, use the following assignment (valid in the Compute node only):

```
SET OutputRoot.MRM.Invoice.TillNumber=26;
```

The integer data type stores numbers using the 64-bit twos complement form, allowing numbers in the range -9223372036854775808 to 9223372036854775807. You can specify hexadecimal notation for integers as well as normal integer literal format. The hexadecimal letters A to F can be written in uppercase or lowercase, as can the X after the initial zero, which is required. The following example produces the same result as the example shown earlier:

```
SET OutputRoot.MRM.Invoice.TillNumber= 0x1A;
```

The following examples show SET statements for element types that do not appear in the "Example message" on page 5311.

To set a FLOAT element to a non-integer value, code:

```
SET OutputRoot.MRM.FloatElement1 = 1.2345e2;
```

To set a BINARY element to a constant value, code:

```
SET OutputRoot.MRM.BinaryElement1 = X'F1F1';
```

For BINARY values, you must use an initial character X (uppercase or lowercase) and enclose the hexadecimal characters (also uppercase or lowercase) in single quotation marks, as shown.

To set a BOOLEAN element to a constant value (the value 1 equates to true, 0 equates to false), code:

```
SET OutputRoot.MRM.BooleanElement1 = true;
```

or

```
SET OutputRoot.MRM.BooleanElement1 = 1;
```

You can use the SELECT statement to filter records from an input message without reformatting the records, and without any knowledge of the complete format of each record. Consider the following example:

```
-- Declare local variable
DECLARE CurrentCustomer CHAR 'Smith';

-- Loop through the input message
SET OutputRoot.XMLNS.Invoice[] =
    (SELECT I FROM InputRoot.XMLNS.Invoice[] AS I
               WHERE I.Customer.LastName = CurrentCustomer
    );
```

This code writes all records from the input message to the output message if the WHERE condition (LastName = Smith) is met. All records that do not meet the

condition are not copied from input message to output message. I is used as an alias for the correlation name InputRoot.XMLNS.Invoice[].

The declared variable CurrentCustomer is initialized on the DECLARE statement: this option is the most efficient way of declaring a variable for which the initial value is known.

You can use this alias technique with other SELECT constructs. For example, if you want to select all the records of the input message, and create an additional record:

```
-- Loop through the input message
SET OutputRoot.XMLNS.Invoice[] =
    (SELECT I, 'Customer' || I.Customer.LastName AS ExtraField
              FROM InputRoot.XMLNS.Invoice[] AS I
    );
```

You could also include an AS clause to place records in a subfolder in the message tree:

```
-- Loop through the input message
SET OutputRoot.XMLNS.Invoice[] =
    (SELECT I AS Order
             FROM InputRoot.XMLNS.Invoice[] AS I
    );
```

If you are querying or setting elements that contain, or might contain, null values, be aware of the following considerations:

Querying null values

When you compare an element to the ESQL keyword NULL, this tests whether the element is present in the logical tree that has been created from the input message by the parser.

For example, you can check whether an invoice number is included in the current invoice message with the following statement:

```
IF InputRoot.XMLNS.Invoice.InvoiceNo IS NULL THEN
   DO;
     -- more ESQL --
END IF;
```

You can also use an ESQL reference, as shown in the following example:

```
DECLARE cursor REFERENCE TO InputRoot.MRM.InvoiceNo;

IF LASTMOVE(cursor) = FALSE THEN
   SET OutputRoot.MRM.Analysis = 'InvoiceNo does not exist in logical tree';
ELSEIF FIELDVALUE(cursor) IS NULL THEN
   SET OutputRoot.MRM.Analysis =
       'InvoiceNo does exist in logical tree but is defined as an MRM NULL value';
ELSE
   SET OutputRoot.MRM.Analysis = 'InvoiceNo does exist and has a value';
END IF;
```

For more information about declaring and using references, see “Creating dynamic field references” on page 2431. For a description of the LASTMOVE and FIELDVALUE functions, see “LASTMOVE function” on page 5237 and “FIELDTYPE function” on page 5231.

If the message is in the MRM domain, there are additional considerations for querying null elements that depend on the physical format. For further details, see "Querying null values in a message in the MRM domain" on page 2597.

Setting null values

You can use two statements to set null values:

1. If you set the element to NULL by using the following statement, the element is deleted from the message tree:

```
SET OutputRoot.XMLNS.Invoice.Customer.Title = NULL;
```

   If the message is in the MRM domain, there are additional considerations for null values that depend on the physical format. For further details, see "Setting null values in a message in the MRM domain" on page 2599.

   This technique is called implicit null processing.

2. If you set the value of this element to NULL as follows:

```
SET OutputRoot.XMLNS.Invoice.Customer.Title VALUE = NULL;
```

   the element is not deleted from the message tree. Instead, a special value of NULL is assigned to the element.

```
SET OutputRoot.XMLNS.Invoice.Customer.Title = NULL;
```

   If the message is in the MRM domain, the content of the output bit stream depends on the settings of the physical format null handling properties. For further details, see "Setting null values in a message in the MRM domain" on page 2599.

   This technique is called explicit null processing.

If you set an MRM complex element or an XML, XMLNS, or JMS parent element to NULL without using the VALUE keyword, that element and all its children are deleted from the logical tree.

Related concepts:

"Correlation names" on page 1069
A correlation name is a field reference that identifies a well-defined starting point in the logical message tree and is used in field references to describe a standard part of the tree format.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Accessing elements in a message in the MRM domain" on page 2584
You can use ESQL to manipulate the logical tree that represents a message in the message flow. This topic describes how to access data for elements in a message in the MRM domain.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.
"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.
"DatabaseInput node" on page 4360
Use the DatabaseInput node to detect events recorded in a database, and to retrieve the data affected by those events.
"Filter node" on page 4452
Use the Filter node to route a message according to message content.
"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.
"DECLARE statement" on page 5117
Use the DECLARE statement to define a variable, the data type of the variable and, optionally, its initial value.
"IF statement" on page 5134
The IF statement executes one set of statements based on the result of evaluating condition expressions.
"FIELDVALUE function" on page 5234
"LASTMOVE function" on page 5237
"SELECT function" on page 5260
The SELECT function combines, filters, and transforms complex message and database data.
"SET statement" on page 5159
The SET statement assigns a value to a variable.
"ESQL field reference overview" on page 5049
You can use ESQL field references to form paths to message body elements.
"ESQL reserved keywords" on page 5307
The keywords listed are reserved in uppercase, lowercase, or mixed case.
"Example message" on page 5311

Accessing known multiple occurrences of an element:

When you refer to or create the content of messages, it is very likely that the data contains repeating fields. If you know how many instances there are of a repeating field, and you want to access a specific instance of such a field, you can use an array index as part of a field reference.

About this task

For example, you might want to filter on the first line of an address, to expedite the delivery of an order. Three instances of the element Billling.Address are always present in the sample message. To test the first line, write an expression such as:

```
IF Body.Invoice.Customer.Billing.Address[1] = 'Patent Office' THEN
   DO;
     -- more ESQL --
END IF;
```

The array index [1] indicates that it is the first instance of the repeating field that you are interested in (array indices start at 1). An array index such as this can be

used at any point in a field reference, so you could, for example, filter on the following test:

```
IF Body.Invoice."Item"[1].Quantity> 2 THEN
   DO;
     -- more ESQL --
END IF;
```

You can refer to the last instance of a repeating field using the special [<] array index, and to instances relative to the last (for example, the second to last) as follows:

- `Field[<]` indicates the last element.
- `Field[<1]` indicates the last element.
- `Field[<2]` indicates the last but one element (the penultimate element).

You can also use the array index [>] to represent the first element, and elements relative to the first element in a similar way.

- `Field[>]` indicates the first element. This is equivalent to `Field[1]`.

The following examples refer to the "Example message" on page 5311 using these indexes:

```
IF Body.Invoice.Customer.Billing.Address[<] = 'Hampshire' THEN
      DO;
     -- more ESQL --
END IF;
IF Body.Invoice.Customer.Billing.Address[<2 ] = 'Southampton' THEN
   DO;
     -- more ESQL --
END IF;
```

You can also use these special indexes for elements that repeat an unknown number of times.

Deleting repeating fields:
About this task

If you pass a message with several repeats of an element through a message flow and you want to delete some of the repeats, be aware that the numbering of the repeats is reordered after each delete. For example, if you have a message with five repeats of a particular element, and in the message flow you have the following ESQL:

```
SET OutputRoot.MRM.e_PersonName[1] = NULL;
SET OutputRoot.MRM.e_PersonName[4] = NULL;
```

You might expect elements one and four to be deleted. However, because repeating elements are stored on a stack, when you delete one, the one above it takes its place. This means that, in the above example, elements one and five are deleted. To avoid this problem, delete in reverse order, that is, delete element four first, then delete element one.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Accessing unknown multiple occurrences of an element"
To access repeating fields in a message, you must use a construct that can iterate over all instances of a repeating field.

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"DatabaseInput node" on page 4360
Use the DatabaseInput node to detect events recorded in a database, and to retrieve the data affected by those events.

"Filter node" on page 4452
Use the Filter node to route a message according to message content.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"IF statement" on page 5134
The IF statement executes one set of statements based on the result of evaluating condition expressions.

"SET statement" on page 5159
The SET statement assigns a value to a variable.

"Example message" on page 5311

Accessing unknown multiple occurrences of an element:

To access repeating fields in a message, you must use a construct that can iterate over all instances of a repeating field.

About this task

You are likely to deal with messages that contain repeating fields with an unknown number of repeats (such as the `Item` field in "Example message" on page 5311).

To write a filter that takes into account all instances of the `Item` field, you need to use a construct that can iterate over all instances of a repeating field. The quantified predicate allows you to execute a predicate against all instances of a repeating field, and collate the results.

For example, you might want to verify that none of the items that are being ordered has a quantity greater than 50. To do this you could write:

```
FOR ALL Body.Invoice.Purchases."Item"[]
    AS I (I.Quantity <= 50)
```

With the quantified predicate, the first thing to note is the brackets [] on the end of the field reference after FOR ALL. These tell you that you are iterating over all instances of the `Item` field.

In some cases, this syntax appears unnecessary because you can get that information from the context, but it is done for consistency with other pieces of syntax.

The AS clause associates the name `I` with the current instance of the repeating field. This is similar to the concept of iterator classes used in some object oriented languages such as C++. The expression in parentheses is a predicate that is evaluated for each instance of the `Item` field.

A description of this example is:

Procedure

Iterate over all instances of the field Item inside `Body.Invoice`. For each iteration:

1. Bind the name `I` to the current instance of `Item`.
2. Evaluate the predicate `I.Quantity <= 50`. If the predicate:
   - Evaluates to TRUE for all instances of `Item`, return TRUE.
   - Is FALSE for any instance of `Item`, return FALSE.
   - For a mixture of TRUE and UNKNOWN, return UNKNOWN.

Results

The above is a description of how the predicate is evaluated if you use the ALL keyword. An alternative is to specify SOME, or ANY, which are equivalent. In this case the quantified predicate returns TRUE if the sub-predicate returns TRUE for any instance of the repeating field. Only if the sub-predicate returns FALSE for all instances of the repeating field does the quantified predicate return FALSE. If a mixture of FALSE and UNKNOWN values are returned from the sub-predicate, an overall value of UNKNOWN is returned.

In the following filter expression:

```
FOR ANY Body.Invoice.Purchases."Item"[]
    AS I (I.Title = 'The XML Companion')
```

the sub-predicate evaluates to TRUE. However this next expression returns FALSE:

```
FOR ANY Body.Invoice.Purchases."Item"[]
    AS I (I.Title = 'C Primer')
```

because the `C Primer` is not included on this invoice. If some of the items in the invoice do not include a book title field, the sub-predicate returns UNKNOWN, and the quantified predicate returns the value UNKNOWN.

To deal with the possibility of null values appearing, write this filter with an explicit check on the existence of the field, as follows:

```
FOR ANY Body.Invoice.Purchases."Item"[]
  AS I (I.Book IS NOT NULL AND I.Book.Title = 'C Primer')
```

The predicate IS NOT NULL ensures that, if an `Item` field does not contain a `Book`, a FALSE value is returned from the sub-predicate.

You can also manipulate arbitrary repeats of fields within a message by using a SELECT expression, as described in “Referencing columns in a database” on page 2489.

You can refer to the first and last instances of a repeating field using the [>] and [<] array indexes, and to instances relative to the first and last, even if you do not know how many instances there are. These indexes are described in “Accessing known multiple occurrences of an element” on page 2425.

Alternatively, you can use the CARDINALITY function to determine how many instances of a repeating field there are. For example:

```
DECLARE I INTEGER CARDINALITY(Body.Invoice.Purchases."Item"[])
```

Related concepts:

“Message flows overview” on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

“ESQL overview” on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

“Message modeling” on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

“Accessing known multiple occurrences of an element” on page 2425
When you refer to or create the content of messages, it is very likely that the data contains repeating fields. If you know how many instances there are of a repeating field, and you want to access a specific instance of such a field, you can use an array index as part of a field reference.

“Designing a message flow” on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you

must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"DatabaseInput node" on page 4360
Use the DatabaseInput node to detect events recorded in a database, and to retrieve the data affected by those events.

"Filter node" on page 4452
Use the Filter node to route a message according to message content.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"CARDINALITY function" on page 5238

"DECLARE statement" on page 5117
Use the DECLARE statement to define a variable, the data type of the variable and, optionally, its initial value.

"FOR function" on page 5235
The FOR field function evaluates an expression and assigns a resulting value of TRUE, FALSE, or UNKNOWN.

"Example message" on page 5311

Using anonymous field references:

You can refer to the array of all children of a particular element by using a path element of `*`.

About this task

For example:

```
InputRoot.*[]
```

is a path that identifies the array of all children of InputRoot. This is often used in conjunction with an array subscript to refer to a particular child of an entity by position, rather than by name. For example:

`InputRoot.*[<]`
: Refers to the last child of the root of the input message, that is, the body of the message.

`InputRoot.*[1]`
: Refers to the first child of the root of the input message, the message properties.

You might want to find out the name of an element that has been identified with a path of this kind. To do this, use the FIELDNAME function, which is described in "FIELDNAME function" on page 5229.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"DatabaseInput node" on page 4360
Use the DatabaseInput node to detect events recorded in a database, and to retrieve the data affected by those events.

"Filter node" on page 4452
Use the Filter node to route a message according to message content.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

Creating dynamic field references:

You can use a variable of type REFERENCE as a dynamic reference to navigate a message tree. This acts in a similar way to a message cursor or a variable pointer.

About this task

It is generally simpler and more efficient to use reference variables in preference to array indexes when you access repeating structures. Reference variables are

accepted everywhere that field references are accepted, and come with a set of statements and functions to allow detailed manipulation of message trees.

You must declare a dynamic reference before you can use it. A dynamic reference is declared and initialized in a single statement.

All examples in this topic use the "Example message" on page 5311 as their input message. The following example shows how to create and use a reference.

```
-- Declare the dynamic reference
DECLARE myref REFERENCE TO OutputRoot.XMLNS.Invoice.Purchases.Item[1];

--  Continue processing for each item in the array
WHILE LASTMOVE(myref)=TRUE
DO
-- Add 1 to each item in the array
   SET myref = myref + 1;
-- Move the dynamic reference to the next item in the array
   MOVE myref NEXTSIBLING;
END WHILE;
```

The example works if the message tree was created with typed fields, based on a message model for the "Example message" on page 5311. If this is not the case, you can modify the ESQL to work without a model, for example:

```
SET myref = CAST (myref AS INTEGER) + 1;
```

This example declares a dynamic reference, `myref`, which points to the first item in the array within Purchases. The value in the first item is incremented by one, and the pointer (dynamic reference) is moved to the next item. Once again the item value is incremented by one. This process continues until the pointer moves outside the scope of the message array (all the items in this array have been processed) and the LASTMOVE function returns FALSE.

The following examples show further examples.

```
DECLARE ref1 REFERENCE TO InputBody.Invoice.Purchases.Item[1];

DECLARE ref2 REFERENCE TO
 InputBody.Invoice.Purchases.NonExistentField;

DECLARE scalar1 CHARACTER;
DECLARE ref3 REFERENCE TO scalar1;
```

In the second example, `ref2` is set to point to InputBody because the specified field does not exist.

With the exception of the MOVE statement, which changes the position of the dynamic reference, you can use a dynamic reference anywhere that you can use a static reference. The value of the dynamic reference in any expression or statement is the value of the field or variable to which it currently points. For example, using the message in "Example message" on page 5311, the value of `Invoice.Customer.FirstName` is Andrew. If the dynamic reference `myref` is set to point at the Customer field as follows:

```
DECLARE myref REFERENCE TO Invoice.Customer;
```

you can then extend this dynamic reference to address children of that field:

```
SET myref.Billing.Address[1] = 'Oaklands';
```

This changes the address in the example to Oaklands Hursley Village Hampshire SO213JR.

The position of a dynamic reference remains fixed even if a tree is modified. To illustrate this point the steps that follow use the message in "Example message" on page 5311 as their input message and create a modified version of this message as an output message:

Procedure

1. Copy the input message to the output message.
2. To modify the output message, first declare a dynamic reference `ref1` that points at the first item, The XML Companion.

```
DECLARE ref1 REFERENCE TO
 OutputRoot.XMLNS.Invoice.Purchases.Item[1];
```

   The dynamic reference is now equivalent to the static reference `OutputRoot.XMLNS.Invoice.Purchases.Item[1].`
3. Use a create statement to insert a new first item for this purchase.

```
CREATE PREVIOUSSIBLING OF ref1 VALUES 'Item';
```

   The dynamic reference is now equivalent to the static reference `OutputRoot.XMLNS.Invoice.Purchases.Item[2].`

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"DatabaseInput node" on page 4360
Use the DatabaseInput node to detect events recorded in a database, and to retrieve the data affected by those events.

"Filter node" on page 4452
Use the Filter node to route a message according to message content.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"DECLARE statement" on page 5117
Use the DECLARE statement to define a variable, the data type of the variable and, optionally, its initial value.

"IF statement" on page 5134
The IF statement executes one set of statements based on the result of evaluating condition expressions.

"LASTMOVE function" on page 5237

"MOVE statement" on page 5145
The MOVE statement changes the field to which the *Target* reference variable points.

"SET statement" on page 5159
The SET statement assigns a value to a variable.

"WHILE statement" on page 5167
The WHILE statement evaluates a condition expression, and if it is TRUE executes a sequence of statements.

"Example message" on page 5311

Creating new fields:

You can use a Compute node to create a new output message by adding new fields to an existing input message.

About this task

This topic provides example ESQL code for a Compute node that creates a new output message based on the input message, to which are added a number of additional fields.

The input message received by the Compute node within the message flow is an XML message, and has the following content:

```
<TestCase description="This is my TestCase">
    <Identifier>ES03B305_T1</Identifier>
    <Sport>Football</Sport>
    <Date>01/02/2000</Date>
    <Type>LEAGUE</Type>
</TestCase>
```

The Compute node is configured and an ESQL module is created that includes the following ESQL. The following code copies the headers from the input message to the new output message, then creates the entire content of the output message body.

```
-- copy headers
DECLARE i INTEGER 1;
DECLARE numHeaders INTEGER CARDINALITY(InputRoot.*[]);

WHILE i < numHeaders DO
    SET OutputRoot.*[i] = InputRoot.*[i];
    SET i = i + 1;
END WHILE;

CREATE FIELD OutputRoot.XMLNS.TestCase.description TYPE NameValue VALUE 'This is my TestCase';
CREATE FIRSTCHILD OF OutputRoot.XMLNS.TestCase Domain('XMLNS') NAME 'Identifier'
                               VALUE InputRoot.XMLNS.TestCase.Identifier;
CREATE LASTCHILD OF OutputRoot.XMLNS.TestCase Domain('XMLNS') NAME 'Sport'
                               VALUE InputRoot.XMLNS.TestCase.Sport;
 CREATE LASTCHILD OF OutputRoot.XMLNS.TestCase Domain('XMLNS') NAME 'Date'
                               VALUE InputRoot.XMLNS.TestCase.Date;
 CREATE LASTCHILD OF OutputRoot.XMLNS.TestCase Domain('XMLNS') NAME 'Type'
                               VALUE InputRoot.XMLNS.TestCase.Type;
 CREATE FIELD OutputRoot.XMLNS.TestCase.Division[1].Number TYPE NameValue
                               VALUE 'Premiership';
 CREATE FIELD OutputRoot.XMLNS.TestCase.Division[1].Result[1].Number TYPE NameValue VALUE  '1';
 CREATE FIELD OutputRoot.XMLNS.TestCase.Division[1].Result[1].Home TYPE Name;
 CREATE LASTCHILD OF OutputRoot.XMLNS.TestCase.Division[1].Result[1].Home NAME 'Team'
                               VALUE 'Liverpool' ;
 CREATE LASTCHILD OF OutputRoot.XMLNS.TestCase.Division[1].Result[1].Home NAME 'Score'
                               VALUE '4';
 CREATE FIELD OutputRoot.XMLNS.TestCase.Division[1].Result[1].Away TYPE Name;
 CREATE LASTCHILD OF OutputRoot.XMLNS.TestCase.Division[1].Result[1].Away NAME 'Team'
                               VALUE 'Everton';
 CREATE LASTCHILD OF OutputRoot.XMLNS.TestCase.Division[1].Result[1].Away NAME 'Score'
                               VALUE '0';

 CREATE FIELD OutputRoot.XMLNS.TestCase.Division[1].Result[2].Number TYPE NameValue VALUE '2';
 CREATE FIELD OutputRoot.XMLNS.TestCase.Division[1].Result[2].Home TYPE Name;
 CREATE LASTCHILD OF OutputRoot.XMLNS.TestCase.Division[1].Result[2].Home NAME 'Team'
                               VALUE 'Manchester United';
 CREATE LASTCHILD OF OutputRoot.XMLNS.TestCase.Division[1].Result[2].Home NAME 'Score'
                               VALUE '2';
 CREATE FIELD OutputRoot.XMLNS.TestCase.Division[1].Result[2].Away TYPE Name;
 CREATE LASTCHILD OF OutputRoot.XMLNS.TestCase.Division[1].Result[2].Away NAME 'Team'
                               VALUE 'Arsenal';
 CREATE LASTCHILD OF OutputRoot.XMLNS.TestCase.Division[1].Result[2].Away NAME 'Score'
                               VALUE '3';

 CREATE FIELD OutputRoot.XMLNS.TestCase.Division[2].Number TYPE NameValue
                               VALUE '2';
 CREATE FIELD OutputRoot.XMLNS.TestCase.Division[2].Result[1].Number TYPE NameValue
                               VALUE  '1';
 CREATE FIELD OutputRoot.XMLNS.TestCase.Division[2].Result[1].Home TYPE Name;
 CREATE LASTCHILD OF OutputRoot.XMLNS.TestCase.Division[2].Result[1].Home NAME 'Team'
                               VALUE 'Port Vale';
 CREATE LASTCHILD OF OutputRoot.XMLNS.TestCase.Division[2].Result[1].Home NAME 'Score'
                               VALUE '9' ;
 CREATE FIELD OutputRoot.XMLNS.TestCase.Division[2].Result[1].Away TYPE Name;
 CREATE LASTCHILD OF OutputRoot.XMLNS.TestCase.Division[2].Result[1].Away NAME 'Team'
                               VALUE 'Brentford';
 CREATE LASTCHILD OF OutputRoot.XMLNS.TestCase.Division[2].Result[1].Away NAME 'Score'
                               VALUE '5';
```

The output message that results from the ESQL shown above has the following structure and content:

```
<TestCase description="This is my TestCase">
  <Identifier>ES03B305_T1</Identifier>
  <Sport>Football</Sport>
  <Date>01/02/2000</Date>
  <Type>LEAGUE</Type>
```

```
  <Division Number="Premiership">
    <Result Number="1">
      <Home>
        <Team>Liverpool</Team>
        <Score>4</Score>
      </Home>
      <Away>
        <Team>Everton</Team>
        <Score>0</Score>
      </Away>
    </Result>
    <Result Number="2">
      <Home>
        <Team>Manchester United</Team>
        <Score>2</Score>
      </Home>
      <Away>
        <Team>Arsenal</Team>
        <Score>3</Score>
      </Away>
    </Result>
  </Division>
  <Division Number="2">
    <Result Number="1">
      <Home>
        <Team>Port Vale</Team>
        <Score>9</Score>
      </Home>
      <Away>
        <Team>Brentford</Team>
        <Score>5</Score>
      </Away>
    </Result>
  </Division>
</TestCase>
```

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"DatabaseInput node" on page 4360
Use the DatabaseInput node to detect events recorded in a database, and to retrieve the data affected by those events.

"Filter node" on page 4452
Use the Filter node to route a message according to message content.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"DECLARE statement" on page 5117
Use the DECLARE statement to define a variable, the data type of the variable and, optionally, its initial value.

"CREATE statement" on page 5082
The CREATE statement creates a new message field.

Generating multiple output messages:

You can use the PROPAGATE statement to generate multiple output messages in the Compute node. The output messages that you generate can have the same or different content. You can also direct output messages to any of the four alternate output terminals of the Compute node, or to a Label node.

About this task

For example, to create three copies of the input message received by the Compute node, and send one to the standard Out terminal of the Compute node, one to the first alternate Out1 terminal of the Compute node, and one to the Label node ThirdCopy, code the following ESQL:

```
SET OutputRoot = InputRoot;
PROPAGATE;
SET OutputRoot = InputRoot;
PROPAGATE TO TERMINAL 'out1';
SET OutputRoot = InputRoot;
PROPAGATE TO LABEL 'ThirdCopy';
```

In the above example, the content of OutputRoot is reset before each PROPAGATE, because by default the node clears the output message buffer and reclaims the memory when the PROPAGATE statement completes. An alternative method is to instruct the node not to clear the output message on the first two PROPAGATE statements, so that the message is available for routing to the next destination. The code to achieve this result is:

```
SET OutputRoot = InputRoot;
PROPAGATE DELETE NONE;
PROPAGATE TO TERMINAL 'out1' DELETE NONE;
PROPAGATE TO LABEL 'ThirdCopy';
```

If you do not initialize the output buffer, an empty message is generated, and the message flow detects an error and throws an exception.

Also ensure that you copy all required message headers to the output message buffer for each output message that you propagate.

If you want to modify the output message content before propagating each message, code the appropriate ESQL to make the changes that you want before you code the PROPAGATE statement.

If you set up the contents of the last output message that you want to generate, and propagate it as the final action of the Compute node, you do not have to include the final PROPAGATE statement. The default action of the Compute node is to propagate the contents of the output buffer when it terminates. This is implemented by the RETURN TRUE statement, included as the final statement in the module skeleton.

For example, to generate three copies of the input message, and not perform any further action, include this code immediately before the RETURN TRUE statement:

```
SET OutputRoot = InputRoot;
PROPAGATE DELETE NONE;
PROPAGATE DELETE NONE;
```

Alternatively, you can modify the default behavior of the node by changing RETURN TRUE to RETURN FALSE:

```
SET OutputRoot = InputRoot;
PROPAGATE DELETE NONE;
PROPAGATE DELETE NONE;
PROPAGATE;
RETURN FALSE;
```

Three output messages are generated by the three PROPAGATE statements. The final RETURN FALSE statement causes the node to terminate but not propagate a final output message. Note that the final PROPAGATE statement does not include the DELETE NONE clause, because the node must release the memory at this stage.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Accessing the Properties tree" on page 2460
The Properties tree has its own correlation name, Properties, and you must use this in all ESQL statements that refer to or set the content of this tree.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"PROPAGATE statement" on page 5150
The PROPAGATE statement propagates a message to the downstream nodes.

"RETURN statement" on page 5155
The RETURN statement ends processing. What happens next depends on the programming context in which the RETURN statement is issued.

"SET statement" on page 5159
The SET statement assigns a value to a variable.

Using numeric operators with datetime values:

The following examples show the ESQL that you can code to manipulate datetime values with numeric operators.

About this task

Adding an interval to a datetime value

The simplest operation that you can perform is to add an interval to, or subtract an interval from, a datetime value. For example, you could write the following expressions:

```
DATE '2000-03-29' + INTERVAL '1' MONTH
TIMESTAMP '1999-12-31 23:59:59' + INTERVAL '1' SECOND
```

The following example shows how to calculate a retirement date by adding the retirement age to the birth date.

```
DECLARE retAge CHARACTER '65';
DECLARE birthDate DATE DATE '1953-06-01';

SET OutputRoot.XML.Test.retirementDate = birthDate + CAST(retAge AS INTERVAL YEAR);
```

The repetition of the word DATE in the above example is intentional. The first occurrence of DATE specifies the data type of the declared variable, birthDate. The second occurrence initializes the same variable with the constant character string that is enclosed in single quotation marks as a DATE.

Adding or subtracting two intervals

You can combine two interval values by using addition or subtraction. The two interval values must be of compatible types. It is not valid to add a year-month interval to a day-second interval as shown in the following example:

```
INTERVAL '1-06' YEAR TO MONTH + INTERVAL '20' DAY
```

The interval qualifier of the resultant interval is sufficient to encompass all the fields that are present in the two operand intervals. For example:

```
INTERVAL '2 01' DAY TO HOUR + INTERVAL '123:59' MINUTE TO SECOND
```

results in an interval with qualifier DAY TO SECOND, because both day and second fields are present in at least one of the operand values.

Subtracting two datetime values

You can subtract two datetime values to return an interval. You must include an interval qualifier in the expression to indicate what precision the result should be returned in. For example:

```
(CURRENT_DATE - DATE '1776-07-04') DAY
```

returns the number of days since the 4th July 1776, whereas:

```
(CURRENT_TIME - TIME '00:00:00') MINUTE TO SECOND
```

returns the age of the day in minutes and seconds.

Scaling intervals

You can multiply or divide an interval value by an integer factor:

```
INTERVAL '2:30' MINUTE TO SECOND / 4
```

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL operators" on page 2382
An ESQL operator is a character or symbol that you can use in expressions to specify relationships between fields or values.

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Calculating a time interval" on page 2441
You can use ESQL to calculate the time interval between two events, and to set a timer to be triggered after a specified interval.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

“ESQL INTERVAL data type” on page 5027
The INTERVAL data type holds an interval of time.
“ESQL simple comparison operators” on page 5057
The simple comparison operators >, <, >=, <=, =, and <>.
“ESQL logical operators” on page 5062
The logical operators AND, OR and NOT.
“ESQL numeric operators” on page 5064
The numeric operators +, -, *, /, and ||.
“Rules for ESQL operator precedence” on page 5066
How ESQL calculates expressions involving more than one operator.
“CAST function” on page 5245

Calculating a time interval:

You can use ESQL to calculate the time interval between two events, and to set a timer to be triggered after a specified interval.

About this task

This ESQL example calculates the time interval between an input WebSphere MQ message being put on the input queue, and the time that it is processed in the current Compute node.

(When you make a call to a CURRENT_ datetime function, the value that is returned is identical to the value returned to another call in the same node. This ensures that you can use the function consistently within a single node.)

```
CALL CopyMessageHeaders();
Declare PutTime INTERVAL;

SET PutTime = (CURRENT_GMTTIME - InputRoot.MQMD.PutTime) MINUTE TO SECOND;

SET OutputRoot.XMLNS.Test.PutTime = PutTime;
```

The output message has the format (although actual values vary):

```
<Test>
 <PutTime>INTERVAL &apos;1:21.862&apos; MINUTE TO SECOND</PutTime>
</Test>
```

Example

The following code snippet sets a timer, to be triggered after a specified interval from the start of processing, in order to check that processing has completed. If processing has not completed within the elapsed time, the firing of the timer might, for example, trigger some recovery processing.

The StartTime field of the timeout request message is set to the current time plus the allowed delay period, which is defined by a user-defined property on the flow. (The user-defined property has been set to a string of the form "HH:MM:SS" by the administrator.)

```
DECLARE StartDelyIntervalStr EXTERNAL CHARACTER '01:15:05';

CREATE PROCEDURE ValidateTimeoutRequest() BEGIN

  -- Set the timeout period
  DECLARE timeoutStartTimeRef REFERENCE TO
          OutputRoot.XMLNSC.Envelope.Header.TimeoutRequest.StartTime;
  IF LASTMOVE(timeoutStartTimeRef)
```

```
  THEN
  -- Already set
 ELSE
  -- Set it from the UDP StartDelyIntervalStr
  DECLARE startAtTime TIME CURRENT_TIME
         + CAST(StartDelyIntervalStr AS INTERVAL HOUR TO SECOND);

  -- Convert "TIME 'hh.mm.ss.fff'" to hh.mm.ss format
      -- needed in StartTime field
  DECLARE startAtTimeStr CHAR;
  SET startAtTimeStr = startAtTime;
  SET startAtTimeStr = SUBSTRING(startAtTimeStr FROM 7 FOR  8);
  SET OutputRoot.XMLNSC.Envelope.Header.TimeoutRequest.StartTime
                           = startAtTimeStr;
 END IF;
END;
```

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Using numeric operators with datetime values" on page 2439
The following examples show the ESQL that you can code to manipulate datetime values with numeric operators.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"ESQL INTERVAL data type" on page 5027
The `INTERVAL` data type holds an interval of time.

"ESQL numeric operators" on page 5064
The numeric operators `+`, `-`, `*`, `/`, and `||`.

"Rules for ESQL operator precedence" on page 5066
How ESQL calculates expressions involving more than one operator.

"CAST function" on page 5245

Selecting a subfield from a larger field:

You might have a message flow that processes a message containing delimited subfields. You can code ESQL to extract a subfield from the surrounding content if you know the delimiters of the subfield.

About this task

If you create a function that performs this task, or a similar one, you can invoke it both from ESQL modules (for Compute, Database, DatabaseInput, and Filter nodes) and from mapping files (used by DataDelete, DataInsert, DataUpdate, Extract, Mapping, and Warehouse nodes).

The following function example extracts a particular subfield of a message that is delimited by a specific character.

```
CREATE FUNCTION SelectSubField
       (SourceString CHAR, Delimiter CHAR, TargetStringPosition INT)
d       RETURNS CHAR
-- This function returns a substring at parameter position TargetStringPosition within the
-- passed parameter SourceString.  An example of use might be:
-- SelectSubField(MySourceField,' ',2) which will select the second subfield from the
-- field MySourceField delimited by a blank.  If MySourceField has the value
-- "First Second Third" the function will return the value "Second"
  BEGIN
    DECLARE DelimiterPosition INT;
    DECLARE CurrentFieldPosition INT 1;
    DECLARE StartNewString INT 1;
    DECLARE WorkingSource CHAR SourceString;
    SET DelimiterPosition = POSITION(Delimiter IN SourceString);
    WHILE CurrentFieldPosition < TargetStringPosition
      DO
       IF DelimiterPosition = 0  THEN
      -- DelimiterPosition will be 0 if the delimiter is not found
         -- exit the loop
        SET CurrentFieldPosition = TargetStringPosition;
       ELSE
        SET StartNewString = DelimiterPosition + 1;
        SET WorkingSource = SUBSTRING(WorkingSource FROM StartNewString);
        SET DelimiterPosition = POSITION(Delimiter IN WorkingSource);
        SET CurrentFieldPosition = CurrentFieldPosition + 1;
       END IF;
    END WHILE;
    IF DelimiterPosition> 0 THEN
       -- Remove anything following the delimiter from the string
       SET WorkingSource = SUBSTRING(WorkingSource FROM 1 FOR DelimiterPosition);
       SET WorkingSource = TRIM(TRAILING Delimiter FROM WorkingSource);
    END  IF;
    RETURN WorkingSource;
END;
```

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"DatabaseInput node" on page 4360
Use the DatabaseInput node to detect events recorded in a database, and to retrieve the data affected by those events.

"Filter node" on page 4452
Use the Filter node to route a message according to message content.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"DECLARE statement" on page 5117
Use the DECLARE statement to define a variable, the data type of the variable and, optionally, its initial value.

"POSITION function" on page 5212
POSITION is a string manipulation function that manipulates all data types (BIT, BLOB, and CHARACTER), and returns the position of one string within another.

"SET statement" on page 5159
The SET statement assigns a value to a variable.

"SUBSTRING function" on page 5218
SUBSTRING is a string manipulation function that manipulates all string data types (BIT, BLOB, and CHARACTER), and extracts characters from a string to create another string.

"TRIM function" on page 5221
TRIM is a string manipulation function that manipulates all string data types (BIT, BLOB, and CHARACTER), and removes trailing and leading singletons from a string.

"WHILE statement" on page 5167
The WHILE statement evaluates a condition expression, and if it is TRUE executes a sequence of statements.

Copying repeating fields:

You can configure a node with ESQL to copy repeating fields in several ways.

About this task

Consider an input XML message that contains a repeating structure:

```
...
 <Field_top>
   <field1></field1>
   <field1></field1>
   <field1></field1>
   <field1></field1>
   <field1></field1>
 </Field_top>
.....
```

You cannot copy this whole structure field with the following statement:

```
SET OutputRoot.XMLNS.Output_top.Outfield1 = InputRoot.XMLNS.Field_top.field1;
```

That statement copies only the first repeat, and therefore produces the same result as this statement:

```
SET OutputRoot.XMLNS.Output_top.Outfield1[1] = InputRoot.XMLNS.Field_top.field1[1];
```

You can copy the fields within a loop, controlling the iterations with the CARDINALITY of the input field:

```
SET I = 1;
SET J = CARDINALITY(InputRoot.XMLNS.Field_top.field1[]);
WHILE  I <= J DO
 SET OutputRoot.XMLNS.Output_top.Outfield1[I] = InputRoot.XMLNS.Field_top.field1[I];
 SET I = I + 1;
END WHILE;
```

This might be appropriate if you want to modify each field in the output message as you copy it from the input field (for example, add a number to it, or fold its contents to uppercase), or after it has been copied. If the output message already contained more Field1 fields than existed in the input message, the surplus fields would not be modified by the loop and would remain in the output message.

The following single statement copies the iterations of the input fields to the output fields, and deletes any surplus fields in the output message.

```
SET OutputRoot.XMLNS.Output_top.Outfield1.[] = InputRoot.XMLNS.Field_top.field1[];
```

The following example shows how you can rename the elements when you copy them into the output tree. This statement does not copy across the source element name, therefore each `field1` element becomes a `Target` element.

```
SET OutputRoot.XMLNS.Output_top.Outfield1.Target[] =
    (SELECT I FROM InputRoot.XMLNS.Field_top.field1[] AS I );
```

The next example shows a different way to do the same operation; it produces the same end result.

```
SET OutputRoot.XMLNS.Output_top.Outfield2.Target[]
              = InputRoot.XMLNS.Field_top.field1[];
```

The following example copies across the source element name. Each `field1` element is retained as a `field1` element under the `Target` element.

```
SET OutputRoot.XMLNS.Output_top.Outfield3.Target.[]
              = InputRoot.XMLNS.Field_top.field1[];
```

This example is an alternative way to achieve the same result, with `field1` elements created under the `Target` element.

```
SET OutputRoot.XMLNS.Output_top.Outfield4.Target.*[]
              = InputRoot.XMLNS.Field_top.field1[];
```

These examples show that there are several ways in which you can code ESQL to copy repeating fields from source to target. Select the most appropriate method to achieve the results that you require.

The principals shown here apply equally to all areas of the message tree to which you can write data, not just the output message tree.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"DatabaseInput node" on page 4360
Use the DatabaseInput node to detect events recorded in a database, and to retrieve the data affected by those events.

"Filter node" on page 4452
Use the Filter node to route a message according to message content.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"SET statement" on page 5159
The SET statement assigns a value to a variable.

"WHILE statement" on page 5167
The WHILE statement evaluates a condition expression, and if it is TRUE executes a sequence of statements.

"CARDINALITY function" on page 5238

A note about copying fields:

When you copy an input message element to an output element, the *value* and *type* of the output element is set to that of the input element. Therefore, if, for example, you have an input XML document with an attribute, and you want to set a Field element (rather than an attribute) in your output message to the value of the input attribute, you must include a TYPE clause cast to change the element-type from attribute to Field.

About this task

For example, given the following input:

```
<Field01 Attrib01='Attrib01_Value'>Field01_Value</Field01>
```

To create an output, such as in the following example:

```
<MyField_A MyAttrib_A='Attrib01_Value' MyAttrib_B='Field01_Value' >
  <MyField_B>Field01_Value</MyField_BC>
  <MyField_C>Attrib01_Value</MyField_C>
 </MyField_A'>
```

You would use the following ESQL:

```
-- Create output attribute from input attribute
SET OutputRoot.XMLNSC.MyField_A.MyAttrib_A = InputRoot.XMLNSC.Field01.Attrib01;
-- Create output field from input field
```

```
SET OutputRoot.XMLNSC.MyField_A.MyField_B = InputRoot.XMLNSC.Field01;

-- Create output attribute from input field value, noting we have to
-- "cast" back to an attribute  element
SET OutputRoot.XMLNSC.MyField_A.(XMLNSC.Attribute)MyAttrib_B =
               InputRoot.XMLNSC.Field01;

-- Create output field from input attribute value, noting we have to
-- "cast" back to a field element
SET OutputRoot.XMLNSC.MyField_A.(XMLNSC.Field)MyField_C =
               InputRoot.XMLNSC.Field01.Attrib01;
```

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"DatabaseInput node" on page 4360
Use the DatabaseInput node to detect events recorded in a database, and to retrieve the data affected by those events.

"Filter node" on page 4452
Use the Filter node to route a message according to message content.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"SET statement" on page 5159
The SET statement assigns a value to a variable.

"WHILE statement" on page 5167
The WHILE statement evaluates a condition expression, and if it is TRUE executes a sequence of statements.

"CARDINALITY function" on page 5238

Working with elements of type list:

The XML Schema specification allows an element, or attribute, to contain a list of values that are based on a simple type, with the individual values separated by white space.

About this task

In the message tree, an `xsd:list` element is represented as a name node, with an anonymous child node for each list item. Repeating lists can be handled without any loss of information.

Consider the following XML input message:

```
<message1>
  <listE1>one two three</listE1>
</message1>
```

This XML element produces the following message tree:

```
MRM
listEl  (Name)
 "one"   (Value)
 "two"   (Value)
 "three" (Value)
```

Individual list items can be accessed as `ElementName.*[n]`.

For example, use the following ESQL to access the third item of listE1:

```
SET X = FIELDVALUE (InputBody.message1.listE1.*[3]);
```

An attribute can also be of type `xsd:list`.

Consider the following XML input message:

```
<message1>
<Element listAttr="one two three"/>
</message1>
```

This XML element produces the following message tree:

```
MRM
Element (Name)
 listAttr (Name)
  "one" (Value)
  "two" (Value)
  "three" (Value)
```

As before, individual list items can be accessed as `AttrName.*[n]`.

For example, use the following ESQL to access the third item of listAttr:

```
SET X = FIELDVALUE (InputBody.message1.Element.listAttr.*[3]);
```

A list element can occur more than once.

Consider the following XML message:

```
<message1>
<listE1>one two three/listE1>
<listE1>four five six/listE1>
</message1>
```

The message tree for this XML message is:

```
MRM
listE1 (Name)
 "one" (Value)
 "two" (Value)
 "three" (Value)
listE1 (Name)
 "four" (Value)
 "five" (Value)
 "six" (Value)
```

Code the following ESQL to access the first item in the second occurrence of the list:

```
SET X = FIELDVALUE (InputBody.message1.listE1[2].*[1]);
```

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Accessing elements in the message body" on page 2420
When you want to access the contents of a message, for reading or writing, use the structure and arrangement of the elements in the tree that is created by the parser from the input bit stream.

"Mapping between a list and a repeating element"
This task shows how to map between a list and a repeating element.

Related reference:

"SET statement" on page 5159
The SET statement assigns a value to a variable.

Mapping between a list and a repeating element:

This task shows how to map between a list and a repeating element.

About this task

Consider the form of the following XML input message:

```
<MRM>
  <inner>abcde fghij 12345</inner>
</MRM>
```

where the element inner is of type `xsd:list`, and therefore has three associated string values, rather than a single value.

To copy the three values into an output message, where each value is associated with an instance of repeating elements as shown here:

```
<MRM>
  <str1>abcde</str1>
  <str1>fghij</str1>
  <str1>12345</str1>
</MRM>
```

you might expect that the following ESQL syntax works:

```
DECLARE D INTEGER;
SET D = CARDINALITY(InputBody.str1.*[]);
DECLARE M INTEGER 1;
```

```
WHILE M <= D DO
   SET OutputRoot.MRM.str1[M] = InputBody.inner.*[M];
   SET M = M + 1;
END WHILE;
```

However, the statement:

```
SET OutputRoot.MRM.str1[M] = InputBody.inner.*[M];
```

requests a tree copy from input to output. Because the output element does not yet exist, the statement creates it, and its value and type are set from the input.

Therefore, to create the output message with the required format, given an input element which is of type `xsd:list`, use the "FIELDVALUE function" on page 5234 to explicitly retrieve only the value of the input element:

```
SET OutputRoot.MRM.str1[M] = FIELDVALUE(InputBody.inner.*[M]);
```

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Accessing elements in the message body" on page 2420
When you want to access the contents of a message, for reading or writing, use the structure and arrangement of the elements in the tree that is created by the parser from the input bit stream.

Related reference:

"SET statement" on page 5159
The SET statement assigns a value to a variable.

"FIELDVALUE function" on page 5234

Manipulating repeating fields in a message tree:

This topic describes the use of the SELECT function, and other column functions, to manipulate repeating fields in a message tree.

About this task

Suppose that you want to perform a special action on invoices that have a total order value greater than a certain amount. To calculate the total order value of an `Invoice` field, you must multiply the `Price` fields by the `Quantity` fields in all the `Items` in the message, and total the result. You can do this using a SELECT expression as follows:

```
(
 SELECT SUM( CAST(I.Price AS DECIMAL) * CAST(I.Quantity AS INTEGER) )
  FROM Body.Invoice.Purchases."Item"[] AS I
)
```

The example assumes that you need to use CAST expressions to cast the string values of the fields `Price` and `Quantity` into the correct data types. The cast of the `Price` field into a decimal produces a decimal value with the *natural* scale and precision, that is, whatever scale and precision is necessary to represent the number. These CASTs would not be necessary if the data were already in an appropriate data type.

The SELECT expression works in a similar way to the quantified predicate, and in much the same way that a SELECT works in standard database SQL. The FROM

clause specifies what is being iterated, in this case, all Item fields in Invoice, and establishes that the current instance of Item can be referred to using I. This form of SELECT involves a column function, in this case the SUM function, so the SELECT is evaluated by adding together the results of evaluating the expression inside the SUM function for each Item field in the Invoice. As with standard SQL, NULL values are ignored by column functions, with the exception of the COUNT column function explained later in this section, and a NULL value is returned by the column function only if there are no non-NULL values to combine.

The other column functions that are provided are MAX, MIN, and COUNT. The COUNT function has two forms that work in different ways with regard to NULLs. In the first form you use it much like the SUM function above, for example:

```
SELECT COUNT(I.Quantity)
  FROM Body.Invoice.Purchases."Item"[] AS I
```

This expression returns the number of Item fields for which the Quantity field is non-NULL. That is, the COUNT function counts non-NULL values, in the same way that the SUM function adds non-NULL values. The alternative way of using the COUNT function is as follows:

```
SELECT COUNT(*)
  FROM Body.Invoice.Purchases."Item"[] AS I
```

Using COUNT(*) counts the total number of Item fields, regardless of whether any of the fields is NULL. The above example is in fact equivalent to using the CARDINALITY function, as in the following:

```
CARDINALITY(Body.Invoice.Purchases."Item"[])
```

In all the examples of SELECT given here, just as in standard SQL, you could use a WHERE clause to provide filtering on the fields.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"DatabaseInput node" on page 4360
Use the DatabaseInput node to detect events recorded in a database, and to retrieve the data affected by those events.

"Filter node" on page 4452
Use the Filter node to route a message according to message content.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"CARDINALITY function" on page 5238

"SELECT function" on page 5260
The SELECT function combines, filters, and transforms complex message and database data.

"Example message" on page 5311

Manipulating other parts of the message tree:

You can access message tree headers, the properties tree, the local environment tree, the environment tree and the exception list tree.

About this task

The following topics describe how you can access parts of the message tree other than the message body data. These parts of the logical tree are independent of the domain in which the message exists, and all these topics apply to messages in the BLOB domain in addition to all other supported domains.

- "Accessing headers" on page 2453
- "Accessing the Properties tree" on page 2460
- "Accessing the local environment tree" on page 2463
- "Accessing the environment tree" on page 2469
- "Accessing the ExceptionList tree using ESQL" on page 2471

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined

structure that is known and agreed by the sender and the receiver.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"DatabaseInput node" on page 4360
Use the DatabaseInput node to detect events recorded in a database, and to retrieve the data affected by those events.

"Filter node" on page 4452
Use the Filter node to route a message according to message content.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

Accessing headers:

If the input message received by an input node includes message headers that are recognized by the input node, the node invokes the correct parser for each header. You can access these headers using ESQL.

About this task

Parsers are supplied for most WebSphere MQ headers. The following topics provide some guidance for accessing the information in the MQMD, MQRFH2, and MQPCF headers, which you can follow as general guidance for accessing other headers also present in your messages.

Every header has its own correlation name, for example, MQMD, and you must use this name in all ESQL statements that refer to or set the content of this tree:

- "Accessing the MQMD header" on page 2455
- "Accessing the MQRFH2 header" on page 2456
- "Accessing the MQCFH header" on page 2458

For further details of the contents of these and other WebSphere MQ headers for which WebSphere Message Broker provides a parser, see "Element definitions for message parsers" on page 4237.

Accessing transport headers:

About this task

You can manipulate WebSphere MQ, HTTP, and JMS transport headers and their properties without writing Compute nodes:

- Use the MQHeader node to add, modify, or delete MQ Message Descriptor (MQMD) and MQ Dead Letter Header (MQDLH) headers.
- Use the HTTPHeader node to add, modify, or delete HTTP headers such as HTTPRequest and HTTPReply.
- Use the JMSHeader node to modify contents of the JMS Header_Values and Application properties so that you can control the node's output without programming.

:

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

Related reference:

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"Element definitions for message parsers" on page 4237

"MQHeader node" on page 4590
Use the MQHeader node to add, modify, or delete MQ Message Descriptor (MQMD) and MQ Dead Letter Header (MQDLH) headers.

"HTTPHeader node" on page 4470
Use the HTTPHeader node to add, modify, or delete HTTP headers such as HTTPInput, HTTPResponse, HTTPRequest and HTTPReply.

"JMSHeader node" on page 4529
Use the JMSHeader node to modify contents of the JMS Header_Values and Application properties so that you can control the node's output without

programming.

Accessing the MQMD header:

Code ESQL statements to access the fields of the MQMD header.

About this task

WebSphere MQ and WebSphere MQ Everyplace messages include an MQMD header.

You can refer to the fields within the MQMD, and you can update them in a Compute node.

For example, you might want to copy the message identifier MSGID in the MQMD to another field in your output message. To do that, code:

```
SET OutputRoot.MRM.Identifier = InputRoot.MQMD.MsgId;
```

If you send a message to an EBCDIC system from a distributed system, you might need to convert the message to a compatible CodedCharSetId and Encoding. To do this conversion, code the following ESQL in the Compute node:

```
SET OutputRoot.MQMD.CodedCharSetId = 500;
SET OutputRoot.MQMD.Encoding = 785;
```

The MQMD properties of CodedCharSetId and Encoding define the code page and encoding of the section of the message that follows (typically this is either the MQRFH2 header or the message body itself).

Differences exist in the way the Properties folder and the MQMD folder are treated with respect to which folder takes precedence for the same fields. For more information, see “Properties versus MQMD folder behavior for various transports” on page 1050.

Related concepts:

“Message flows overview” on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

“ESQL overview” on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

“How the message tree is populated” on page 1047
The message tree is initially populated by the input node of the message flow.

Related tasks:

“Designing a message flow” on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

“Defining message flow content” on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

“Managing ESQL files” on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database,

DatabaseInput, and Filter nodes.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"SET statement" on page 5159
The SET statement assigns a value to a variable.

"The MQMD parser" on page 4249
The elements of the MQMD parser are listed in this topic.

Accessing the MQRFH2 header:

Code ESQL statements to access the fields of the MQRFH2 header.

About this task

When you construct an MQRFH2 header in a Compute node, it includes two types of fields:

- Fields in the MQRFH2 header structure; for example, Format and NameValueCCSID.
- Fields in the MQRFH2 NameValue buffer; for example, mcd and psc.

To differentiate between these two field types, insert a value in front of the referenced field in the MQRFH2 field to identify its type; a value for the NameValue buffer is not required because this is the default. The value that you specify for the header structure is `(MQRFH2.Field)`.

For example:

- To create or change an MQRFH2 Format field, specify the following ESQL:

```
SET OutputRoot.MQRFH2.(MQRFH2.Field)Format = 'MQSTR   ';
```

- To create or change the psc folder with a topic:

```
SET OutputRoot.MQRFH2.psc.Topic = 'department';
```

- To add an MQRFH2 header to an outgoing message that is to be used to make a subscription request:

```
DECLARE I INTEGER 1;
DECLARE J INTEGER CARDINALITY(InputRoot.*[]);

WHILE I < J DO
 SET OutputRoot.*[I] = InputRoot.*[I];
 SET I=I+1;
END WHILE;

SET OutputRoot.MQRFH2.(MQRFH2.Field)Version = 2;
SET OutputRoot.MQRFH2.(MQRFH2.Field)Format = 'MQSTR';
SET OutputRoot.MQRFH2.(MQRFH2.Field)NameValueCCSID = 1208;
SET OutputRoot.MQRFH2.psc.Command = 'RegSub';
SET OutputRoot.MQRFH2.psc.Topic = "InputRoot"."MRM"."topel";
SET OutputRoot.MQRFH2.psc.QMgrName = 'DebugQM';
SET OutputRoot.MQRFH2.psc.QName = 'PUBOUT';
SET OutputRoot.MQRFH2.psc.RegOpt = 'PersAsPub';
```

Variable J is initialized to the value of the cardinality of the existing headers in the message. Using a variable is more efficient than calculating the cardinality on each iteration of the loop, which happens if you code the following WHILE statement:

```
WHILE I < CARDINALITY(InputRoot.*[]) DO
```

The MQRFH2 header can be parsed using either the MQRFH2 parser or the MQRFH2C compact parser. To consume less memory, use the MQRFH2C compact parser by selecting the `Use MQRFH2C compact parser for MQRFH2 Header` check box on the input node of the message flow. This results in paths that contain MQRFH2C instead of MQRFH2; for example: `SET OutputRoot.MQRFH2C.psc.Topic = 'department';`

Target MQRFH2 fields are created only if the headers are copied, and the MQRFH2C parser option is not selected on the MQInput node. In all other circumstances, an MQRFH2C field is created on output.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"CARDINALITY function" on page 5238

"DECLARE statement" on page 5117
Use the DECLARE statement to define a variable, the data type of the variable and, optionally, its initial value.

"SET statement" on page 5159
The SET statement assigns a value to a variable.

"WHILE statement" on page 5167
The WHILE statement evaluates a condition expression, and if it is TRUE executes a sequence of statements.

"MQRFH2 header" on page 6397
The MQRFH2 header is used to pass messages to and from a message broker that belongs to WebSphere Message Broker.

"The MQRFH2 and MQRFH2C parsers" on page 4253
The MQRFH2 header can be parsed using either the MQRFH2 parser or the MQRFH2C compact parser.

Accessing the MQCFH header:

Code ESQL statements to access the fields of the MQCFH header (root name MQPCF).

About this task

Messages that are of PCF format (MQPCF, MQADMIN, and MQEVENT) include the MQCFH header. You can process the contents of the MQCFH header, accessing parameters, parameter lists, strings, and groups.

The *ParameterCount* field is hidden from view to keep the value synchronized with the true number of parameters. As a result, you cannot directly view, access, or edit the *ParameterCount* field and this applies when you are using:

- ESQL
- The Mapping Node
- Trace, or debugging code.

You can implement your own *ParameterCount* field with a specific value, but this value will be overwritten by the actual number of parameters on the flow exit.

- To access or to construct parameters in the header, use the following syntax:

  ```
  SET OutputRoot.MQPCF.Parameter[nn] =
   Integer parameter ID
  ```

  If you access a 64-bit parameter, use the following syntax to differentiate from 32-bit parameters:

  ```
  SET OutputRoot.MQPCF.Parameter64[nn] =
   Integer parameter ID
  ```

  For example:

  ```
  SET OutputRoot.MQPCF.Parameter[1] =
   MQCACF_AUTH_PROFILE_NAME;
  ```

- For parameter lists, use the following syntax:

```
SET OutputRoot.MQPCF.ParameterList64[nn] =
 Integer parameter ID
SET OutputRoot.MQPCF.ParameterList64[nn].*[xx] =
 Integer parameter values
```

For example:

```
SET OutputRoot.MQPCF.ParameterList[1] =
 MQIACF_AUTH_ADD_AUTHS;
SET OutputRoot.MQPCF.ParameterList[1].*[1] =
 MQAUTH_SET;
SET OutputRoot.MQPCF.ParameterList[1].*[2] =
 MQAUTH_SET_ALL_CONTEXT;
```

- A byte string is a byte array data type, and is supported with this syntax:

```
SET OutputRoot.MQPCF.Parameter[nn] =
 Integer parameter ID
SET OutputRoot.MQPCF.Parameter[nn].* =
 Integer, String or ByteArray Parameter value
```

- A group is implemented as a folder containing more PCF, and requires the following syntax:

```
SET OutputRoot.MQPCF.Group[xx] =
 Group Parameter ID
```

For example:

```
SET OutputRoot.MQPCF.Group[1] =
 MQGACF_Q_ACCOUNTING_DATA;
SET OutputRoot.MQPCF.Group[1].Parameter[1] =
 MQCA_CREATION_DATE;
SET OutputRoot.MQPCF.Group[1].Parameter[1].* =
 '2007-02-05';
```

You can also use nested groups; for example;

```
SET OutputRoot.MQPCF.Group[1].Group[1] =
 MQGACF_Q_ACCOUNTING_DATA;
SET OutputRoot.MQPCF.Group[1].Group[1].Parameter[1] =
 MQCA_CREATION_DATE;
SET OutputRoot.MQPCF.Group[1].Group[1].Parameter[1].* =
 '2007-02-05';
```

- A filter is almost identical to a parameter, but it also includes an operator. Therefore the syntax is a tree with named values:

```
SET OutputRoot.MQPCF.Filter[xx] =
 Integer parameter ID
SET OutputRoot.MQPCF.Filter[xx].Operator =
 Integer Filter name
SET OutputRoot.MQPCF.Filter[xx].Value =
 Byte, Integer or String Filter Value
```

- The following is sample code that can be used as an example to create an MQPCF message in a Compute node:

```
CREATE NEXTSIBLING OF OutputRoot.Properties DOMAIN 'MQMD';
CREATE NEXTSIBLING OF OutputRoot.MQMD DOMAIN 'MQADMIN'
NAME 'MQPCF';
CREATE FIELD OutputRoot.MQPCF;
SET OutputRoot.MQMD.MsgType = MQMT_REQUEST;
SET OutputRoot.MQMD.ReplyToQ = 'REPLYQ';
DECLARE refRequest REFERENCE TO OutputRoot.MQPCF;
SET refRequest.Type = 16; --MQCFT_COMMAND_XR z/OS
SET refRequest.StrucLength = MQCFH_STRUC_LENGTH;
SET refRequest.Version = 3; -- required for z/OS
SET refRequest.Command = MQCMD_INQUIRE_Q;
SET refRequest.MsgSeqNumber = 1;
SET refRequest.Control = MQCFC_LAST;
/* First parameter: Queue Name. */
```

```
SET refRequest.Parameter[1] = MQCA_Q_NAME;
SET refRequest.Parameter[1].* = 'QUEUENAME.*';
SET refRequest.ParameterList[1] = MQIACF_Q_ATTRS;
SET refRequest.ParameterList[1].*  = MQIACF_ALL;
```

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"The MQCFH parser" on page 4245
The elements of the MQCFH parser are listed in this topic.

Accessing the Properties tree:

The Properties tree has its own correlation name, Properties, and you must use this in all ESQL statements that refer to or set the content of this tree.

About this task

The fields in the Properties tree contain values that define the characteristics of the message. For example, the Properties tree contains message template information for model-driven parsers, fields for the encoding and CCSID in which message data is encoded, and fields that hold the security identity of the message. For a full list of fields in this tree, see "Data types for elements in the Properties subtree" on page 4239.

You can interrogate and update these fields using the appropriate ESQL statements. If you create a new output message in the Compute node, you must set values for the message properties.

Setting output message properties:
About this task

If you use the Compute node to generate a new output message, you must set its properties in the Properties tree. The output message properties do not have to be the same as the input message properties.

For example, to set the output message properties for an output MRM message, set the following properties:

Property	Value
MessageSet	Message set identifier
MessageType	Message name[1]
MessageFormat	Physical format name[2]

Notes:

1. For details of the syntax of Message type, see "Specifying namespaces in the Message Type property" on page 1208.
2. The name that you specify for the physical format must match the name that you have defined for it. The default physical format names are Binary1, XML1, and Text1.

This ESQL procedure sets message properties to values passed in by the calling statement. You might find that you have to perform this task frequently, and you can use a procedure such as this in many different nodes and message flows. If you prefer, you can code ESQL that sets specific values.

```
CREATE PROCEDURE setMessageProperties(IN OutputRoot REFERENCE, IN setName char,
                 IN typeName char, IN formatName char) BEGIN
 /*****************************************************************************
 * A procedure that sets the message properties
 *****************************************************************************/
 set OutputRoot.Properties.MessageSet    = setName;
 set OutputRoot.Properties.MessageType   = typeName;
 set OutputRoot.Properties.MessageFormat = formatName;
END;
```

To set the output message domain, you can code ESQL statements that refer to the required domain in the second qualifier of the SET statement, the parser field. For example, the ESQL statement sets the domain to MRM:

```
SET OutputRoot.MRM.Field1 = 'field1 data';
```

This ESQL statement sets the domain to XMLNS:

```
SET OutputRoot.XMLNS.Field1 = 'field1 data';
```

Do not specify more than one domain in the ESQL for any single message. However, if you use PROPAGATE statements to generate several output messages, you can set a different domain for each message.

For information about the full list of elements in the Properties tree, see "Data types for elements in the Properties subtree" on page 4239.

Differences exist in the way the Properties folder and the MQMD folder are treated with respect to which folder takes precedence for the same fields. For more information, see "Properties versus MQMD folder behavior for various transports" on page 1050.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"Specifying namespaces in the Message Type property" on page 1208
When using the MRM domain, the *Message Type* property is used to specify the name of the message.

"Multipart messages" on page 1191
A *multipart message* contains one or more other messages within its structure. The contained message is sometimes referred to as an *embedded message*.

"How the message tree is populated" on page 1047
The message tree is initially populated by the input node of the message flow.

"Parsers" on page 1072
A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from the internal message tree representation.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

"Accessing the MQMD header" on page 2455
Code ESQL statements to access the fields of the MQMD header.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"Filter node" on page 4452
Use the Filter node to route a message according to message content.

"Data types for elements in the Properties subtree" on page 4239
A parser is supplied for the Properties subtree; it associates each field with a specific data type.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"SET statement" on page 5159
The SET statement assigns a value to a variable.

Accessing the local environment tree:

The local environment tree has its own correlation name, LocalEnvironment, and you must use this name in all ESQL statements that refer to or set the content of this tree.

About this task

The local environment tree is used by the broker, and you can refer to and modify this information. You can also extend the tree to contain information that you create yourself. You can create subtrees of this tree that you can use as a scratchpad or working area.

The message flow sets up information in two subtrees, Destination and WrittenDestination, below the LocalEnvironment root. You can refer to the content of both of these subtrees, and you can write to the Destination tree to influence the way in which the message flow processes your message. However, if you write to the Destination tree, follow the defined structure to ensure that the tree remains valid.

The WrittenDestination subtree contains the addresses to which the message has been written. Its name is fixed and it is created by the message flow when a message is propagated through the Out terminal of a request, output, or reply node. The subtree includes transport-specific information (for example, if the output message has been put to a WebSphere MQ queue, it includes the queue manager and queue names). You can use one of the following methods to obtain information about the details of a message after it has been sent by the nodes:

- Connect a Compute node to the Out terminal.
- Configure a user exit to process an output message callback event, as described in "Exploiting user exits" on page 2985.

The topic for each node that supports WrittenDestination information contains details about the data that it contains.

If you want the local environment tree to be included in the output message that is propagated by the Compute node, you must set the Compute node property `Compute mode` to a value that includes the local environment (for example, `All`). If you do not, the local environment tree is not copied to the output message.

The information that you insert into DestinationData or Defaults depends on the characteristic of the corresponding node property:

- If a node property is represented by a check box (for example, `New Message ID`), set the Defaults or DestinationData element to `Yes` (equivalent to selecting the check box) or `No` (equivalent to clearing the check box).
- If a node property is represented by a drop-down list (for example, `Transaction Mode`), set the Defaults or DestinationData element to the appropriate character string (for example `Automatic`).
- If a node property is represented by a text entry field (for example, `Queue Manager Name`), set the Defaults or DestinationData element to the character string that you would enter in this field.

If necessary, configure the sending node to indicate where the destination information is. For example, for the output node MQOutput, set `Destination Mode`:

- If you set `Destination Mode` to `Queue Name`, the output message is sent to the queue identified in the output node properties `Queue Name` and `Queue Manager Name`. Destination is not referenced by the node.
- If you set `Destination Mode` to `Destination List`, the node extracts the destination information from the Destination subtree. If you use this value you can send a single message to multiple destinations, if you configure Destination and a single output node correctly. The node checks the node properties only if a value is not available in Destination (as described above).
- If you set `Destination Mode` to `Reply To Queue`, the message is sent to the reply-to queue identified in the MQMD in this message (field ReplyToQ). Destination is not referenced by the node.

To find more information about ESQL procedures that perform typical updates to the local environment, see "Populating Destination in the local environment tree" on page 2467. Review the ESQL statements in these procedures to see how to modify the local environment. You can use these procedures unchanged, or modify them for your own requirements.

To find more information about how to extend the contents of this tree for your own purposes, see "Using scratchpad areas in the local environment" on page 2465.

For another example of how you can use the local environment to modify the behavior of a message flow, refer to the XML_PassengerQuery message flow in the following sample program:

- Airline Reservations

The Compute node in this message flow writes a list of destinations in the RouterList subtree of Destination that are used as labels by a later RouteToLabel node that propagates the message to the corresponding Label node. You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Local environment tree structure" on page 1056
The local environment tree is a part of the logical message tree in which you can store information while the message flow processes the message.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Creating destination lists" on page 1477
Create a list of destinations to indicate where a message is sent.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow

nodes and connecting them to form flows.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"HTTPRequest node" on page 4488
Use the HTTPRequest node to interact with a web service.

"MQOutput node" on page 4612
Use the MQOutput node to send messages to clients that connect to the broker using the WebSphere MQ Enterprise Transport and that use the MQI and AMI application programming interfaces.

"RouteToLabel node" on page 4673
Use the RouteToLabel node in combination with one or more Label nodes to dynamically determine the route that a message takes through the message flow, based on its content.

"Data types for elements in the MQ DestinationData subtree" on page 4240
The DestinationData subtree is part of the Destination subtree in the local environment. Local environment trees are created by input nodes when they receive a message and, optionally, by Compute nodes. When created, the trees are empty but you can create data in them by using ESQL statements coded in any of the SQL nodes.

Using scratchpad areas in the local environment:

The local environment tree includes a subtree called variables. This subtree is always created, but is never populated by the message flow. Use this area for your own purposes; for example, to pass information from one node to another. You can create other subtrees of the local environment tree.

About this task

The advantage of creating your own data in a scratchpad in the local environment is that this data can be propagated as part of the logical tree to subsequent nodes in the message flow. If you create a new output message in a Compute node, you can also include all or part of the local environment tree from the input message in the new output message.

To ensure that the information in the local environment is propagated further down the flow, the `Compute mode` property of the Compute node must be set to include the local environment as part of the output tree (for example, specify `LocalEnvironment and Message`). For further details about the `Compute mode` property, see "Setting the mode" on page 4344.

However, any data updates or additions that you make in one node are not retained if the message moves backwards through the message flow (for example, if an exception is thrown). If you create your own data, and want that data to be preserved throughout the message flow, you must use the environment tree.

You can set values in the variables subtree in a Compute node and those values can be used later by another node (Compute, Database, or Filter) for some purpose that you determine when you configure the message flow.

The local environment is not in scope in a Compute node, therefore you must use `InputLocalEnvironment` and `OutputLocalEnvironment` instead. For example, you might use the scratchpad in the local environment to propagate the destination of an output message to subsequent nodes in a message flow. Your first Compute node determines that the output messages from this message flow must go to WebSphere MQ queues. Include the following ESQL to insert this information into the local environment by setting the value of *OutputLocation* in the OutputLocalEnvironment:

```
SET OutputLocalEnvironment.Variables.OutputLocation = 'MQ';
```

Your second Compute node can access this information from its input message. In the ESQL in this node, use the correlation name InputLocalEnvironment to identify the local environment tree in the input message that contains this data. The following ESQL sets *queueManagerName* and *queueName* based on the content of *OutputLocation* in the local environment, by using InputLocalEnvironment:

```
IF InputLocalEnvironment.Variables.OutputLocation = 'MQ' THEN
   SET OutputLocalEnvironment.Destination.MQ.DestinationData.queueManagerName = 'myQManagerName';
   SET OutputLocalEnvironment.Destination.MQ.DestinationData.queueName = 'myQueueName';
END IF;
```

In the example, *queueManagerName* and *queueName* are set for the Destination subtree in the output message. You must set the `Compute mode` of the second Compute node to include the local environment tree in the output message. Configure the MQOutput node to use the destination list that you have created in the local environment tree by setting the `Destination Mode` property to `Destination List`.

For information about the full list of elements in the DestinationData subtree, see "Data types for elements in the MQ DestinationData subtree" on page 4240.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Local environment tree structure" on page 1056
The local environment tree is a part of the logical message tree in which you can store information while the message flow processes the message.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"Filter node" on page 4452
Use the Filter node to route a message according to message content.

"Data types for elements in the MQ DestinationData subtree" on page 4240
The DestinationData subtree is part of the Destination subtree in the local environment. Local environment trees are created by input nodes when they receive a message and, optionally, by Compute nodes. When created, the trees are empty but you can create data in them by using ESQL statements coded in any of the SQL nodes.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"SET statement" on page 5159
The SET statement assigns a value to a variable.

Populating Destination in the local environment tree:

Use the Destination subtree to set up the target destinations that are used by output nodes, the HTTPRequest node, the SOAPRequest node, the SOAPAsyncRequest node, and the RouteToLabel node. The following examples show how you can create and use an ESQL procedure to perform the task of setting up values for each of these uses.

About this task

Copy and use these procedures as shown, or you can modify or extend them to perform similar tasks.

If you are creating this ESQL code for a Compute node, you must configure the node by setting the `Compute Mode` property so that it has access to the local environment tree in the output message. You must select one of the three values `LocalEnvironment`, `LocalEnvironment And Message`, or `All`.

Adding a queue name for the MQOutput node with the `Destination Mode` property set to `Destination List`

```
CREATE PROCEDURE addToMQDestinationList(IN LocalEnvironment REFERENCE, IN newQueue char) BEGIN
  /******************************************************************************
  * A procedure that adds a queue name to the MQ destination list in the local environment.
  * This list is used by an MQOutput node that has its mode set to Destination list.
  *
  * IN LocalEnvironment: the LocalEnvironment to be modified.
  * IN queue: the queue to be added to the list
  *
  ******************************************************************************/
DECLARE I INTEGER CARDINALITY(LocalEnvironment.Destination.MQ.DestinationData[]);
  IF I = 0 THEN
   SET OutputLocalEnvironment.Destination.MQ.DestinationData[1].queueName = newQueue;
  ELSE
   SET OutputLocalEnvironment.Destination.MQ.DestinationData[I+1].queueName = newQueue;
  END IF;
 END;
```

For full details of these elements, see "Data types for elements in the MQ DestinationData subtree" on page 4240.

Changing the default URL for a SOAPRequest node or a SOAPAsyncRequest node request

```
CREATE PROCEDURE overrideDefaultSOAPRequestURL(IN LocalEnvironment REFERENCE, IN newUrl char) BEGIN
  /******************************************************************************
  * A procedure that changes the URL to which the SOAPRequest node or
  * SOAPAsyncRequest node sends the request.
```

```
  *
  * IN LocalEnvironment: the LocalEnvironment to be modified.
  * IN queue:  the URL to which to send the request.
  *
  ********************************************************************************/
SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.HTTP.WebServiceURL = newUrl;
END;
```

Changing the default URL for an HTTPRequest node request

```
CREATE PROCEDURE overrideDefaultHTTPRequestURL(IN LocalEnvironment REFERENCE, IN newUrl char) BEGIN
  /********************************************************************************
  * A procedure that changes the URL to which the HTTPRequest node sends the request.
  *
  * IN LocalEnvironment: the LocalEnvironment to be modified.
  * IN queue: the URL to which to send the request.
  *
  ********************************************************************************/
  SET OutputLocalEnvironment.Destination.HTTP.RequestURL = newUrl;
END;
```

Adding a label for the RouteToLabel node

```
CREATE PROCEDURE addToRouteToLabelList(IN LocalEnvironment REFERENCE, IN newLabel char) BEGIN
  /********************************************************************************
  * A procedure that adds a label name to the RouteToLabel list in the local environment.
  * This list is used by a RoteToLabel node.
  *
  * IN LocalEnvironment: the LocalEnvironment to be modified.
  * IN label:  the label to be added to the list
  *
  ********************************************************************************/
 IF LocalEnvironment.Destination.RouterList.DestinationData is null THEN
     SET OutputLocalEnvironment.Destination.RouterList.DestinationData."label" = newLabel;
  ELSE
     CREATE LASTCHILD OF LocalEnvironment.Destination.RouterList.DestinationData
     NAME 'label' VALUE newLabel;
  END IF;
END;
```

Setting up JMS destination lists

You can configure a JMSOutput node to send to multiple JMS Queues, or to publish to multiple JMS Topics by using a destination list that is created in the local environment tree by a transformation node. The following example shows how to set up JMS destination lists in the local environment tree:

```
CREATE PROCEDURE CreateJMSDestinationList() BEGIN
  SET OutputLocalEnvironment.Destination.JMSDestinationList.DestinationData[1]  = 'jndi://TestDestQueue1';
  SET OutputLocalEnvironment.Destination.JMSDestinationList.DestinationData[2]  = 'jndi://TestDestQueue2';
  SET OutputLocalEnvironment.Destination.JMSDestinationList.DestinationData[3]  = 'jndi://TestDestQueue3';
END;
```

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Local environment tree structure" on page 1056
The local environment tree is a part of the logical message tree in which you can store information while the message flow processes the message.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Data types for elements in the MQ DestinationData subtree" on page 4240
The DestinationData subtree is part of the Destination subtree in the local environment. Local environment trees are created by input nodes when they receive a message and, optionally, by Compute nodes. When created, the trees are empty but you can create data in them by using ESQL statements coded in any of the SQL nodes.

"CREATE PROCEDURE statement" on page 5103
The CREATE PROCEDURE statement defines a callable function or procedure.

"SET statement" on page 5159
The SET statement assigns a value to a variable.

Accessing the environment tree:

The environment tree has its own correlation name, Environment, and you must use this name in all ESQL statements that refer to, or set, the content of this tree.

About this task

The environment tree is always created when the logical tree is created for an input message. However, the message flow neither populates it, nor uses its contents. You can use this tree for your own purposes, for example, to pass information from one node to another. You can use the whole tree as a scratchpad or working area.

The advantage of creating your own data in environment is that this data is propagated as part of the logical tree to subsequent nodes in the message flow. If you create a new output message in a Compute node, the environment tree is also copied from the input message to the new output message. (In contrast to the local environment tree, which is only included in the output message if you explicitly request that it is).

Only one environment tree is present for the duration of the message flow. Any data updates, or additions, that you make in one node are retained, and all of the nodes in the message flow have access to the latest copy of this tree. Even if the message flows back through the message flow (for example, if an exception is thrown, or if the message is processed through the second terminal of the FlowOrder node), the latest state is retained. (In contrast to the local environment tree, which reverts to its previous state if the message flows back through the message flow.)

You can use this tree for any purpose you choose. For example, you can use the following ESQL statements to create fields in the tree:

```
SET Environment.Variables =
      ROW('granary' AS bread, 'reisling' AS wine, 'stilton' AS cheese);
SET Environment.Variables.Colors[] =
      LIST{'yellow', 'green', 'blue', 'red', 'black'};
SET Environment.Variables.Country[] = LIST{ROW('UK' AS name, 'pound' AS currency),
     ROW('USA' AS name, 'dollar' AS currency)};
```

This information is now available to all nodes to which a message is propagated, regardless of their relative position in the message flow.

For another example of how you can use environment to store information used by other nodes in the message flow, look at the Reservation message flow in the following sample:

- Airline Reservations

The Compute node in this message flow writes information to the subtree Environment.Variables that it has extracted from a database according to the value of a field in the input message.

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Environment tree structure" on page 1055
The environment tree is a part of the logical message tree in which you can store information while the message passes through the message flow.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"Filter node" on page 4452
Use the Filter node to route a message according to message content.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"ROW constructor function" on page 5267

"LIST constructor function" on page 5269

"ROW and LIST combined" on page 5270

"ROW and LIST comparisons" on page 5271
You can compare ROWs and LISTs against other ROWs and LISTs.
"SET statement" on page 5159
The SET statement assigns a value to a variable.

Accessing the ExceptionList tree using ESQL:

The ExceptionList tree has its own correlation name, ExceptionList, and you must use this in all ESQL statements that refer to or set the content of this tree.

About this task

This tree is created with the logical tree when an input message is parsed. It is initially empty, and is only populated if an exception occurs during message flow processing. It is possible that more than one exception can occur; if more than one exception occurs, the ExceptionList tree contains a subtree for each exception.

You can access the ExceptionList tree in Compute, Database, and Filter nodes, and you can update it in a Compute node. You must use the appropriate correlation name; ExceptionList for a Database or Filter node, and InputExceptionList for a Compute node.

You might want to access this tree in a node in an error handling procedure. For example, you might want to route the message to a different path based on the type of exception, for example one that you have explicitly generated using an ESQL THROW statement, or one that the broker has generated.

The following ESQL shows how you can access the ExceptionList and process each child that it contains:

```
-- Declare a reference for the ExceptionList
-- (in a Compute node use InputExceptionList)
DECLARE start REFERENCE TO ExceptionList.*[1];

-- Loop through the exception list children
WHILE start.Number IS NOT NULL DO
   -- more ESQL

   -- Move start to the last child of the field to which it currently points
   MOVE start LASTCHILD;
END WHILE;
```

The following example shows an extract of ESQL that has been coded for a Compute node to loop through the exception list to the last (nested) exception description and extract the error number. This error relates to the original cause of the problem and normally provides the most precise information. Subsequent action taken by the message flow can be decided by the error number retrieved in this way.

```
CREATE PROCEDURE getLastExceptionDetail(IN InputTree reference,OUT messageNumber integer,
OUT messageText char)
    /*********************************************************************
  * A procedure that will get the details of the last exception from a message
  * IN InputTree:  The incoming exception list
  * IN messageNumber:  The last message numberr.
  * IN messageText: The last message text.
  *********************************************************************/
   BEGIN
        -- Create a reference to the first child of the exception list
        declare ptrException reference to InputTree.*[1];
        -- keep looping while the moves to the child of exception list work
  WHILE lastmove(ptrException) DO
   -- store the current values for the error number and text
   IF ptrException.Number is not null THEN
          SET messageNumber = ptrException.Number;
          SET messageText = ptrException.Text;
     END IF;
     -- now move to the last child which should be the next exceptionlist
   move ptrException lastchild;
  END WHILE;
 END;
```

For more information about the use of ExceptionList, look at the subflow in the following sample which includes ESQL that interrogates the ExceptionList structure and takes specific action according to its content:

- Error Handler

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

For information on accessing the ExceptionList tree using Java, see “Accessing the ExceptionList tree using Java” on page 2656

Related concepts:

“Message flows overview” on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

“Exception list tree structure” on page 1066
The exception list tree is a part of the logical message tree in which the message flow writes information about exceptions that occur when a message is processed.

“ESQL overview” on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

“Message modeling” on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

“Handling errors in message flows” on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Accessing the ExceptionList tree using Java" on page 2656
The ExceptionList tree is created with the logical tree when an input message is parsed.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Exception list structure" on page 4224
An exception list contains information about exceptions, such as error numbers, the name of the node that generated the exception, and the reason for the exception.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"DECLARE statement" on page 5117
Use the DECLARE statement to define a variable, the data type of the variable and, optionally, its initial value.

"EVAL statement" on page 5131
The EVAL statement takes a character value, interprets it as an SQL statement, and processes that statement.

"WHILE statement" on page 5167
The WHILE statement evaluates a condition expression, and if it is TRUE executes a sequence of statements.

Transforming from one data type to another:

Code ESQL functions and statements to transform messages and data types in many ways.

About this task

The following topics provide guidance:

- "Casting data from message fields" on page 2474
- "Converting code page and message encoding" on page 2476
- "Converting EBCDIC NL to ASCII CR LF" on page 2480
- "Changing message format" on page 2484

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"Filter node" on page 4452
Use the Filter node to route a message according to message content.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

Casting data from message fields:

You can use the CAST function to transform the data type of one value to match the data type of the other. For example, you can use the CAST function when you process generic XML messages. All fields in an XML message have character values, so if you want to perform arithmetic calculations or datetime comparisons, for example, you must convert the string value of the field into a value of the appropriate type using CAST.

About this task

When you compare an element with another element, variable or constant, ensure that the value with which you are comparing the element is consistent (for example, character with character). If the values are not consistent, the broker generates a runtime error if it cannot provide an implicit casting to resolve the inconsistency. For details of what implicit casts are supported, see "Implicit casts" on page 5282.

In the "Example message" on page 5311, the field InvoiceDate contains the date of the invoice. If you want to refer to or manipulate this field, you must CAST it to

the correct format first. For example, to refer to this field in a test:

```
IF CAST(Body.Invoice.InvoiceDate AS DATE) = CURRENT_DATE THEN
```

This converts the string value of the InvoiceDate field into a date value, and compares it to the current date.

Another example is casting from integer to character:

```
DECLARE I INTEGER 1;
DECLARE C CHARACTER;

-- The following statement generates an error
SET C = I;

-- The following statement is valid
SET C = CAST(I AS CHARACTER);
```

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"Filter node" on page 4452
Use the Filter node to route a message according to message content.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"CAST function" on page 5245

"DECLARE statement" on page 5117
Use the DECLARE statement to define a variable, the data type of the variable and, optionally, its initial value.

"IF statement" on page 5134
The IF statement executes one set of statements based on the result of evaluating condition expressions.

"Implicit CASTs for comparisons" on page 5282
The standard SQL comparison operators>, <,>=, <=, =, <> are supported for comparing two values in ESQL.

"Implicit CASTs for arithmetic operations" on page 5285

"Implicit CASTs for assignment" on page 5287

"SET statement" on page 5159
The SET statement assigns a value to a variable.

"Example message" on page 5311

Converting code page and message encoding:

You can use ESQL within a Compute node to convert data for code page and message encoding.

About this task

If your message flow is processing WebSphere MQ messages, you can use WebSphere MQ facilities (including get and put options and WebSphere MQ data conversion exits) to provide these conversions. If you are not processing WebSphere MQ messages, or you choose not to use WebSphere MQ facilities, you can use WebSphere Message Broker facilities by coding the appropriate ESQL in a Compute node in your message flow.

The contents of the MQMD, the MQRFH2, and the message body of a message in the MRM domain that has been modeled with a CWF physical format can be subject to code page and encoding conversion. The contents of a message body of a message in the XML, XMLNS, and JMS domains, and those messages in the MRM domain that have been modeled with an XML or TDS physical format, are treated as strings. Only code page conversion applies; no encoding conversion is required.

For messages in the MRM domain modeled with a CWF physical format, you can set the MQMD CCSID and Encoding fields of the output message, plus the CCSID and Encoding of any additional headers, to the required target value.

For messages in the MRM domain modeled with an XML or TDS physical format, you can set the MQMD CCSID field of the output message, plus the CCSID of any additional headers. XML and TDS data is handled as strings and is therefore subject to CCSID conversion only.

An example WebSphere MQ message has an MQMD header, an MQRFH2 header, and a message body. To convert this message to a mainframe CodedCharSetId and Encoding, code the following ESQL in the Compute node:

```
SET OutputRoot.MQMD.CodedCharSetId = 500;
SET OutputRoot.MQMD.Encoding = 785;
SET OutputRoot.MQRFH2.CodedCharSetId = 500;
SET OutputRoot.MQRFH2.Encoding = 785;
```

The following example illustrates what you must do to modify a CWF message so that it can be passed from WebSphere Message Broker to IMS on z/OS.

Procedure

1. You have defined the input message in XML and are using an MQRFH2 header. Remove the header before passing the message to IMS.
2. The message passed to IMS must have MQIIH header, and must be in the z/OS code page. This message is modeled in the MRM and has the name IMS1. Define the PIC X fields in this message as logical type string for conversions between EBCDIC and ASCII to take place. If the fields are binary logical type, no data conversion occurs; binary data is ignored when a CWF message is parsed by the MRM parser.
3. The message received from IMS is also defined in the MRM and has the name IMS2. Define the PIC X fields in this message as logical type string for conversions between EBCDIC and ASCII to take place. If the fields are binary logical type, no data conversion occurs; binary data is ignored when a CWF message is parsed by the MRM parser.
4. Convert the reply message to the Windows code page. The MQIIH header is retained on this message.
5. You have created a message flow that contains the following nodes: :
   a. The outbound flow, **MQInput1 --> Compute1 --> MQOutput1**.
   b. The inbound flow, **MQInput2 --> Compute2 --> MQOutput2**.
6. Code ESQL in Compute1 (outbound) node as follows, specifying the relevant MessageSet ID. This code shows the use of the default CWF physical layer name. You must use the name that matches your model definitions. If you specify an incorrect value, the broker fails with message BIP5431.

```
-- Loop to copy message headers
DECLARE I INTEGER 1;
DECLARE J INTEGER CARDINALITY(InputRoot.*[]);

WHILE I < J - 1 DO
  SET OutputRoot.*[I] = InputRoot.*[I];
  SET I=I+1;
END WHILE;

SET OutputRoot.MQMD.CodedCharSetId = 500;
SET OutputRoot.MQMD.Encoding = 785;
SET OutputRoot.MQMD.Format = 'MQIMS   ';
SET OutputRoot.MQIIH.Version = 1;
SET OutputRoot.MQIIH.StrucLength = 84;
SET OutputRoot.MQIIH.Encoding = 785;
SET OutputRoot.MQIIH.CodedCharSetId = 500;
SET OutputRoot.MQIIH.Format = 'MQIMSVS ';
SET OutputRoot.MQIIH.Flags = 0;
SET OutputRoot.MQIIH.LTermOverride = '        ';
SET OutputRoot.MQIIH.MFSMapName = '        ';
SET OutputRoot.MQIIH.ReplyToFormat = 'MQIMSVS ';
SET OutputRoot.MQIIH.Authenticator = '        ';
SET OutputRoot.MQIIH.TranInstanceId = X'00000000000000000000000000000000';
SET OutputRoot.MQIIH.TranState = ' ';
SET OutputRoot.MQIIH.CommitMode = '0';
SET OutputRoot.MQIIH.SecurityScope = 'C';
SET OutputRoot.MQIIH.Reserved = ' ';
SET OutputRoot.MRM.e_elen08 = 30;
SET OutputRoot.MRM.e_elen09 = 0;
SET OutputRoot.MRM.e_string08 = InputBody.e_string01;
SET OutputRoot.MRM.e_binary02 = X'31323334353637383940';
SET OutputRoot.Properties.MessageSet = 'DHCJOEG072001';
SET OutputRoot.Properties.MessageType = 'IMS1';
SET OutputRoot.Properties.MessageFormat = 'Binary1';
```

The use of a variable, J, that is initialized to the value of the cardinality of the existing headers in the message, is more efficient than calculating the cardinality on each iteration of the loop, which happens if you code the following WHILE statement:

```
WHILE I < CARDINALITY(InputRoot.*[]) DO
```

7. Create ESQL in Compute2 (inbound) node as follows, specifying the relevant MessageSet ID. This code shows the use of the default CWF physical layer name. You must use the name that matches your model definition. If you specify an incorrect value, the broker fails with message BIP5431.

```
-- Loop to copy message headers
DECLARE I INTEGER 1;
DECLARE J INTEGER CARDINALITY(InputRoot.*[]);

WHILE I < J DO
  SET OutputRoot.*[I] = InputRoot.*[I];
  SET I=I+1;
END WHILE;

SET OutputRoot.MQMD.CodedCharSetId = 437;
SET OutputRoot.MQMD.Encoding = 546;
SET OutputRoot.MQMD.Format = 'MQIMS   ';
SET OutputRoot.MQIIH.CodedCharSetId = 437;
SET OutputRoot.MQIIH.Encoding = 546;
SET OutputRoot.MQIIH.Format = '        ';
SET OutputRoot.MRM = InputBody;
SET OutputRoot.Properties.MessageSet = 'DHCJOEG072001';
SET OutputRoot.Properties.MessageType = 'IMS2';
SET OutputRoot.Properties.MessageFormat = 'Binary1';
```

Results

You do not have to set any specific values for the MQInput1 node properties, because the message and message set are identified in the MQRFH2 header, and no conversion is required.

You must set values for message domain, set, type, and format in the MQInput node for the inbound message flow (MQInput2). You do not need to set conversion parameters.

One specific situation in which you might need to convert data in one code page to another is when messages contain newline characters and are passed between EBCDIC and ASCII systems. The required conversion for this situation is described in “Converting EBCDIC NL to ASCII CR LF” on page 2480.

Related concepts:

“Message flows overview” on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

“ESQL overview” on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

“Message modeling” on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

“Designing a message flow” on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

“Defining message flow content” on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

“Managing ESQL files” on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database,

DatabaseInput, and Filter nodes.

Related reference:

"Multicultural support" on page 3628
Multicultural support is available for a selection of languages on both distributed systems and z/OS.

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"Filter node" on page 4452
Use the Filter node to route a message according to message content.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"CARDINALITY function" on page 5238

"DECLARE statement" on page 5117
Use the DECLARE statement to define a variable, the data type of the variable and, optionally, its initial value.

"SET statement" on page 5159
The SET statement assigns a value to a variable.

"WHILE statement" on page 5167
The WHILE statement evaluates a condition expression, and if it is TRUE executes a sequence of statements.

"Supported code pages" on page 4176
Application messages must conform to supported code pages.

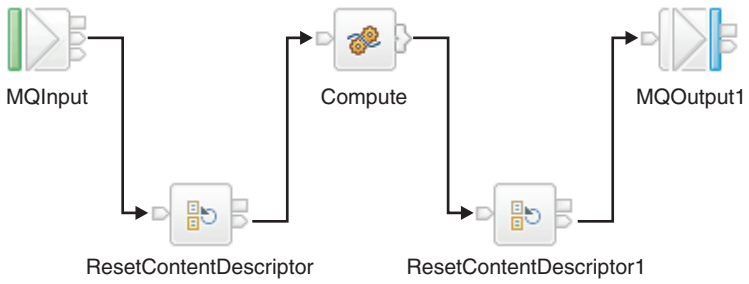
Converting EBCDIC NL to ASCII CR LF:

You might want to change newline (NL) characters in a text message to carriage return (CR) and line feed (LF) character pairs. This example shows one way in which you can convert these characters.

About this task

This conversion might be useful if messages from an EBCDIC platform (for example, using CCSID 1047) are sent to an ASCII platform (for example, using CCSID 437). Problems can arise because the EBCDIC NL character hex '15' is converted to the undefined ASCII character hex '7F'. The ASCII code page has no corresponding code point for the NL character.

In this example, a message flow is created that interprets the input message as a message in the BLOB domain. This message is passed into a ResetContentDescriptor node to reset the data to a message in the MRM domain. The message is called msg_nl (a set of repeating string elements delimited by EBCDIC NL characters). A Compute node is then used to create an output based on another message in the MRM domain called msg_crlf (a set of repeating string elements delimited by CR LF pairs). The message domain is then changed back to BLOB in another ResetContentDescriptor node. This message flow is shown in the following diagram.



The following instructions show how to create the messages and configure the message flow.

Procedure

1. Create the message models for the messages in the MRM domain:
   a. Create a message set project called myProj.
   b. Create a message set called myMessageSet with a TDS physical format (the default name is Text1).
   c. Create an element string1 of type xsd:string.
   d. Create a complex type called t_msg_nl and specify the following complex type properties:
      - `Composition` = Ordered Set
      - `Content Validation` = Closed
      - `Data Element Separation` = All Elements Delimited
      - `Delimiter` = <U+0085> (hex '0085' is the UTF-16 representation of an NL character)
      - `Repeat` = Yes
      - `Min Occurs` = 1
      - `Max Occurs` = 50 (the text of the message is assumed to consist of no more than 50 lines)
   e. Add Element string1, and set the following property:
      - `Repeating Element Delimiter` = <U+0085>
   f. Create a Message msg_nl, and set its associated complex type to t_msg_nl
   g. Create a complex type called t_msg_crlf, and specify the following complex type properties:
      - `Composition` = Ordered Set
      - `Content Validation` = Closed
      - `Data Element Separation` = All Elements Delimited
      - `Delimiter` = <CR><LF> (<CR> and <LF> are the mnemonics for the CR and LF characters)
      - `Repeat` = Yes
      - `Min Occurs` = 1
      - `Max Occurs` = 50
   h. Add Element string1, and set the following property:
      - `Repeating Element Delimiter` = <CR><LF>
   i. Create a Message msg_crlf, and set complex type to t_msg_crlf.
2. Configure the message flow shown in the previous figure:
   a. Start with the MQInput node:
      - Set `Message Domain` = BLOB
      - Set `Queue Name` = <Your input message queue name>

b. Add the ResetContentDescriptor node, connected to the Out terminal of the MQInput node:
   - Set Message Domain = MRM
   - Select Reset Message Domain
   - Set Message Set = <Your Message Set ID> (this field has a maximum of 13 characters)
   - Select Reset Message Set
   - Set Message Type = msg_nl
   - Select Reset Message Type
   - Set Message Format = Text1
   - Select Reset Message Format

c. Add the Compute node, connected to the Out terminal of the ResetContentDescriptor node:
   - Enter a name for the ESQL Module for this node, or accept the default (<message flow name>_Compute).
   - Right-click the Compute node, and select **Open ESQL**. Add the following ESQL code in the module:

```
-- Declare local working variables
DECLARE I INTEGER 1;
DECLARE J INTEGER CARDINALITY(InputRoot.*[]);

-- Loop to copy all message headers from input to output message
WHILE I < J DO
  SET OutputRoot.*[I] = InputRoot.*[I];
  SET I=I+1;
END WHILE;

-- Set new output message type which uses CRLF delimiter
SET OutputRoot.Properties.MessageType = 't_msg_crlf';

-- Loop to copy each instance of string1 child within message body
SET I = 1;
SET J = CARDINALITY("InputBody"."string1"[]);
WHILE I <= J DO
  SET "OutputRoot"."MRM"."string1"[I] = "InputBody"."string1"[I];
  SET I=I+1;
END WHILE;
```

The use of a variable, J, initialized to the value of the cardinality of the existing headers in the message, is more efficient than calculating the cardinality on each iteration of the loop, which happens if you code the following WHILE statement:

```
WHILE I < CARDINALITY(InputRoot.*[]) DO
```

d. Add the ResetContentDescriptor1 node, connected to the Out terminal of the Compute node:
   - Set Message Domain = BLOB
   - Select Reset Message Domain.

e. Finally, add the MQOutput node, connected to the Out terminal of the ResetContentDescriptor1 node. Configure its properties to direct the output message to the required queue or queues.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

Related reference:

"Multicultural support" on page 3628
Multicultural support is available for a selection of languages on both distributed systems and z/OS.

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

"MQOutput node" on page 4612
Use the MQOutput node to send messages to clients that connect to the broker using the WebSphere MQ Enterprise Transport and that use the MQI and AMI application programming interfaces.

"ResetContentDescriptor node" on page 4663
Use the ResetContentDescriptor node to request that the message is reparsed by a different parser.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"DECLARE statement" on page 5117
Use the DECLARE statement to define a variable, the data type of the variable and, optionally, its initial value.

"SET statement" on page 5159
The SET statement assigns a value to a variable.

"WHILE statement" on page 5167
The WHILE statement evaluates a condition expression, and if it is TRUE executes a sequence of statements.

"TDS Mnemonics" on page 5391
The Tagged/Delimited String Format (TDS) uses mnemonics for a number of properties for a message set, complex type, or both.

Changing message format:

Use the Compute node to copy part of an input message to an output message. The results of such a copy depend on the type of input and output parsers involved.

Like parsers:
About this task

Where both the source and target messages have the same folder structure at root level, a *like-parser-copy* is performed. For example:

```
SET OutputRoot.MQMD = InputRoot.MQMD;
```

This statement copies all the children in the MQMD folder of the input message to the MQMD folder of the output message.

Another example of a tree structure that supports a like-parser-copy is:

```
SET OutputRoot.XMLNS.Data.Account = InputRoot.XMLNS.Customer.Bank.Data;
```

To transform an input message in the MRM domain to an output message also in the MRM domain, you can use either the Compute or the Mapping node. The Mapping node can interpret the action that is required because it knows the format of both messages. Content Assist in the ESQL module for the Compute node can also use the message definitions for those messages. If the messages are not in the same namespace, you must use the Compute node.

To use Content Assist with message references, you must set up a project reference from the project containing the ESQL to the project containing the message set. For information about setting up a project reference, see “Project references” on page 44.

If both input and output messages are not in the MRM domain, you must use the Compute node and specify the structure of the messages yourself.

Unlike parsers:
About this task

Where the source and target messages have different folder structures at root level, you cannot make an exact copy of the message source. Instead, the *unlike-parser-copy* views the source message as a set of nested folders terminated by a leaf name-value pair. For example, copying the following message from XML to MRM:

```
<Name3><Name31>Value31</Name31>Value32</Name3>
```

produces a name element Name3, and a name-value element called Name31 with the value Value31. The second XML pcdata (Value32) cannot be represented and is discarded.

The unlike-parser-copy scans the source tree, and copies folders, also known as name elements, and leaf name-value pairs. Everything else, including elements flagged as *special* by the source parser, is not copied.

An example of a tree structure that results in an unlike-parser-copy is:

```
SET OutputRoot.MRM.Account = InputRoot.XMLNS.Data.Account;
```

If the algorithm used to make an unlike-parser-copy does not suit your tree structure, youmust further qualify the source field to restrict the amount of the tree that is copied.

Be careful when you copy information from input messages to output messages in different domains. You might code ESQL that creates a message structure or content that is not consistent with the rules of the parser that processes the output message. This action can result in an output message not being created, or being created with unexpected content. If you believe that the output message generated by a particular message flow does not contain the correct content, or have the expected form, check the ESQL that creates the output message, and look for potential mismatches of structure, field types, field names, and field values.

When copying trees between unlike parsers, you must set the message format of the target parser. For example, if a message set has been defined with XMLNS and CWF formats, the following commands are required to copy an input XMLNS stream to the MRM parser and set the latter to be generated in CWF format:

```
-- Copy message to the output, moving from XMLNS to MRM domains
SET OutputRoot.MRM = InputRoot.XMLNS.rootElement;

-- Set the CWF format for output by the MRM domain
SET OutputRoot.Properties.MessageType = '<MessageTypeName>';
SET OutputRoot.Properties.MessageSet = '<MessageSetName>';
SET OutputRoot.Properties.MessageFormat = 'CWF';
```

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Parsers" on page 1072
A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from the internal message tree representation.

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"Project references" on page 44
When a projects refers to other projects in the workspace, this is called a *project reference*. When one project references another, the files in the referenced project are available for use by the referring project. For example to configure certain nodes, to create libraries of reusable message flows, and to enable Content Assist in the ESQL editor.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"SET statement" on page 5159
The SET statement assigns a value to a variable.

Adding keywords to ESQL files:

You can add keywords to ESQL files to contain information that you want to associate with a message flow.

Use one or more of the following methods:

Comment fields

Add the keyword as a comment in the ESQL file:

```
-- $MQSI compiled by = John MQSI$
```

Static strings

Include the keyword as part of a static string in the ESQL file:

```
SET target = '$MQSI_target = production only MQSI$'
```

Variable string

Include the keyword value as a variable string in the ESQL file:

```
$MQSI_VERSION=$id$MQSI$
```

In this example, when the message flow source is extracted from the file repository, the repository's plug-in has been configured to substitute the identifier *id* with the actual version number. The identifier value that is required depends on the capability and configuration of the repository, and is not part of WebSphere Message Broker.

Restrictions within keywords

Do not use the following characters within keywords, because they cause unpredictable behavior:

```
^ $ . | \ < > ? + * = & [ ] ( )
```

You can use these characters in the values that are associated with keywords; for example:

- `$MQSI RCSVER=$id$ MQSI$` is acceptable
- `$MQSI $name=Fred MQSI$` is not acceptable

Related concepts:

"Message flow version and keywords" on page 1445
When you are developing a message flow, you can define the version of the message flow as well as other key information that you want to be associated with it.

"Version and keyword information for deployable objects" on page 1443
Use the Broker Archive editor to view the version and keyword information of deployable objects.

Related reference:

"Guidance for defining keywords" on page 4017
You can add extra information to an object in the form of one or more keywords.

Interaction with databases using ESQL:

Use ESQL statements and functions to read from, write to, and modify databases from your message flows.

About this task

ESQL has a number of statements and functions for accessing databases:

- The "CALL statement" on page 5077 invokes a stored procedure.
- The "DELETE FROM statement" on page 5127 removes rows from a database table.
- The "INSERT statement" on page 5135 adds a row to a database table.
- The "PASSTHRU function" on page 5297 can be used to make complex selections.
- The "PASSTHRU statement" on page 5147 can be used to invoke administrative operations (for example, creating a table).
- The "SELECT function" on page 5260 retrieves data from a table.
- The "UPDATE statement" on page 5163 changes one or more values stored in zero or more rows.

You can access user databases from Compute, Database, DatabaseInput, and Filter nodes. The same ESQL functions and procedures are supported in all these nodes.

You can use the data in the databases to update or create messages, or use the data in the message to update or create data in the databases.

Select **Throw exception on database error** and **Treat warnings as errors**, and set **`Transaction`** to `Automatic` on each node that access a database, to provide maximum flexibility.

For information about configuring the broker and the database to support access from message flows, see "Accessing databases from ESQL" on page 2115.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Configuring transactionality for message flows" on page 1290
A message flow runs in a single transaction, which is started when data is received by an input node, and can be committed or rolled back when all processing has completed.

"Accessing databases from message flows" on page 2112
Create and configure message flows to access user databases.

"Accessing multiple database tables" on page 2496
You can refer to multiple tables that you have created in the same database. Use the FROM clause on the SELECT statement to join the data from the two tables.

"Referencing columns in a database" on page 2489

"Selecting data from database columns" on page 2491
You can configure a Compute, Filter, or Database node to select data from database columns and include it in an output message.

"Changing database content" on page 2498
You can use Compute, Database, and Filter nodes to change the contents of a database by updating, inserting, or deleting data.

"Checking returns to SELECT" on page 2500
If a SELECT function returns no data, or no further data, this result is handled as a normal situation and no error code is set in SQLCODE, regardless of the setting of the `Throw Exception On Database Error` and `Treat Warnings As Errors` properties on the current node.

"Committing database updates" on page 2501
When you create a message flow that interacts with databases, you can choose whether the updates that you make are committed when the current node has completed processing, or when the current invocation of the message flow has terminated.

"Invoking stored procedures" on page 2503
To invoke a procedure that is stored in a database, use the ESQL CALL statement. The stored procedure must be defined by a CREATE PROCEDURE statement that has a Language clause of DATABASE and an EXTERNAL NAME clause that identifies the name of the procedure in the database and, optionally, the database schema to which it belongs.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

Related reference:

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"Filter node" on page 4452
Use the Filter node to route a message according to message content.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"Data types of values from external databases" on page 5288
How database data types are implicitly cast to ESQL data types.

Referencing columns in a database:
About this task

While the standard SQL SELECT syntax is supported for queries to an external database, there are a number of points to be borne in mind. You must prefix the name of the table with the keyword Database to indicate that the SELECT is to be targeted at the external database, rather than at a repeating structure in the message.

The basic form of database SELECT is:

```
SELECT ...
  FROM Database.TABLE1
  WHERE ...
```

If necessary, you can specify a schema name:

```
SELECT ...
  FROM Database.SCHEMA.TABLE1
  WHERE ...
```

where SCHEMA is the name of the schema in which the table TABLE1 is defined. Include the schema if the user ID under which you are running does not match the schema. For example, if your userID is USER1, the expression Database.TABLE1 is equivalent to Database.USER1.TABLE1. However, if the schema associated with the table in the database is db2admin, you must specify Database.db2admin.TABLE1. If you do not include the schema, and this does not match your current user ID, the broker generates a runtime error when a message is processed by the message flow.

If, as in the two previous examples, a data source is not specified, TABLE1 must be a table in the default database specified by the node's `data source` property. To access data in a database other than the default specified on the node's `data source` property, you must specify the data source explicitly. For example:

```
SELECT ...
  FROM Database.DataSource.SCHEMA.TABLE1
  WHERE ...
```

Qualify references to column names with either the table name or the correlation name defined for the table by the FROM clause. So, where you could normally execute a query such as:

```
SELECT column1, column2 FROM table1
```

you must write one of the following two forms:

```
SELECT T.column1, T.column2 FROM Database.table1 AS T

SELECT table1.column1, table1.column2 FROM Database.table1
```

This is necessary in order to distinguish references to database columns from any references to fields in a message that might also appear in the SELECT:

```
SELECT T.column1, T.column2 FROM Database.table1
  AS T WHERE T.column3 = Body.Field2
```

You can use the AS clause to rename the columns returned. For example:

```
SELECT T.column1 AS price, T.column2 AS item
  FROM Database.table1 AS T WHERE...
```

The standard select all SQL option is supported in the SELECT clause. If you use this option, you must qualify the column names with either the table name or the correlation name defined for the table. For example:

```
SELECT T.* FROM Database.Table1 AS T
```

When you use ESQL procedure and function names within a database query, the positioning of these within the call affects how these names are processed. If it is determined that the procedure or function affects the results returned by the query, it is not processed as ESQL and is passed as part of the database call.

This applies when attempting to use a function or procedure name with the column identifiers within the SELECT statement.

For example, if you use a CAST statement on a column identifier specified in the Select clause, this is used during the database query to determine the data type of the data being returned for that column. An ESQL CAST is not performed to that ESQL data type, and the data returned is affected by the database interaction's interpretation of that data type.

If you use a function or procedure on a column identifier specified in the WHERE clause, this is passed directly to the database manager for processing.

The examples in the subsequent topics illustrate how the results sets of external database queries are represented in WebSphere Message Broker. The results of database queries are assigned to fields in a message using a Compute node.

A column function is a function that takes the values of a single column in all the selected rows of a table or message and returns a single scalar result.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"Filter node" on page 4452
Use the Filter node to route a message according to message content.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"CARDINALITY function" on page 5238

"SELECT function" on page 5260
The SELECT function combines, filters, and transforms complex message and database data.

"Example message" on page 5311

Selecting data from database columns:

You can configure a Compute, Filter, or Database node to select data from database columns and include it in an output message.

About this task

The following example assumes that you have a database table called USERTABLE with two char(6) data type columns (or equivalent), called Column1 and Column2. The table contains two rows:

	Column1	**Column2**
Row 1	value1	value2
Row 2	value3	value4

Configure the Compute, Filter, or Database node to identify the database in which you have defined the table. For example, if you are using the default database (specified on the `data source` property of the node), right-click the node, select

Open ESQL, and code the following ESQL statements in the module for this node:

```
SET OutputRoot = InputRoot;
DELETE FIELD OutputRoot.*[<];
SET OutputRoot.XML.Test.Result[] =
  (SELECT T.Column1, T.Column2 FROM Database.USERTABLE AS T);
```

This ESQL produces the following output message:

```
<Test>
   <Result>
      <Column1>value1</Column1>
      <Column2>value2</Column2>
   </Result>
   <Result>
      <Column1>value3</Column1>
      <Column2>value4</Column2>
   </Result>
</Test>
```

Figure 4. Output message

To trigger the SELECT, send a trigger message with an XML body that is of the following form:

```
<Test>
   <Result>
      <Column1></Column1>
      <Column2></Column2>
   </Result>
   <Result>
      <Column1></Column1>
      <Column2></Column2>
   </Result>
</Test>
```

The exact structure of the XML is not important, but the enclosing tag must be `<Test>` to match the reference in the ESQL. If the enclosing tag is not `<Test>`, the ESQL statements result in top-level enclosing tags being formed, which is not valid XML.

If you want to create an output message that includes all the columns of all the rows that meet a particular condition, use the SELECT statement with a WHERE clause:

```
-- Declare and initialize a variable to hold the
--      test vaue (in this case the surname Smith)
DECLARE CurrentCustomer STRING 'Smith';

-- Loop through table records to extract matching information
SET OutputRoot.XML.Invoice[] =
    (SELECT R FROM Database.USERTABLE AS R
              WHERE R.Customer.LastName = CurrentCustomer
    );
```

The message fields are created in the same order as the columns occur in the table.

If you are familiar with SQL in a database environment, you might expect to code SELECT *. This syntax is not accepted by the broker, because you must start all references to columns with a correlation name to avoid ambiguities with declared

variables. Also, if you code SELECT I.*, this syntax is accepted by the broker, but the * is interpreted as the first child element, not all elements, as you might expect from other database SQL.

The assignment of the result set of a database into a parser-owned message tree requires the result set to exactly match the message definition. Because the generic XML parser is self-defining, the example creates a new subtree off the Invoice folder, and the parser can parse the new elements in the subtree. If the structure of the result set exactly matches the message definition, the result set can be assigned directly into the `OutputRoot` message body tree.

If the structure of the result set does not exactly match the MRM message definition, you must first assign the result set into a ROW data type, or an Environment tree that does not have a parser associated with it.

The required data can then be assigned to `OutputRoot` to build a message tree that conforms to the message definition.

Selecting data from a table in a case-sensitive database system:
About this task

If the database system is case sensitive, you must use an alternative approach. This approach is also necessary if you want to change the name of the generated field to something different:

```
SET OutputRoot = InputRoot;
SET OutputRoot.XML.Test.Result[] =
  (SELECT T.Column1 AS Column1, T.Column2 AS Column2
  FROM Database.USERTABLE AS T);
```

This example produces the output message as shown in Figure 4 on page 2492. Ensure that references to the database columns (in this example, T.Column1 and T.Column2) are specified in the correct case to match the database definitions exactly. If you do not match the database definitions exactly (for example if you specify T.COLUMN1), the broker generates a runtime error. `Column1` and `Column2` are used in the SELECT statement to match the columns that you have defined in the database, although you can use any values here; the values do not have to match.

Selecting bitstream data from a database:
About this task

These samples show how to retrieve XML bitstream data from a database, and include it in an output message. See “INSERT statement” on page 5135 for examples that show how you can insert bitstream data into a database.

Example

In the following example, bitstream data is held in a database column with a BLOB data type. The database table used in the example (TABLE1) is the one created in the “INSERT statement” on page 5135 examples, and the table contains the following columns:

- `MSGDATA`
- `MSGCCSID`
- `MSGENCODING`

If the bit stream from the database does not need to be interrogated or manipulated by the message flow, the output message can be constructed in the BLOB domain without any alteration.

In the following example, the message data, along with the MQMD header, is held in a database column with a BLOB data type. To re-create the message tree, including the MQMD header, from the bit stream, you can use a CREATE statement with a PARSE clause and DOMAIN('MQMD'). The output message can then be modified by the message flow:

```
SET Environment.Variables.DBResult = THE( SELECT T.* FROM Database.TABLE1 AS T);
DECLARE resultRef REFERENCE TO Environment.Variables.DBResult;

IF LASTMOVE(resultRef) THEN

  DECLARE outMsg BLOB resultRef.MSGDATA ;
  DECLARE outCCSID INT resultRef.MSGCCSID;
  DECLARE outEncoding INT resultRef.MSGENCODING;
  DECLARE outMsgPriority INT resultRef.MSGPRIORITY;
  DECLARE outMsgSeqNum INT resultRef.MSGSEQNUMBER;

  SET OutputRoot.Properties.CodedCharSetId = outCCSID;
  SET OutputRoot.Properties.Encoding = outEncoding ;

  CREATE LASTCHILD OF OutputRoot DOMAIN('MQMD') PARSE(outMsg, outEncoding, outCCSID);

    SET OutputRoot.MQMD.Version = MQMD_VERSION_2;


  SET OutputRoot.MQMD.Priority = outMsgPriority;
  SET OutputRoot.MQMD.MsgSeqNumber = outMsgSeqNum;

  DECLARE HDRL INT ;
  SET HDRL = LENGTH(BITSTREAM(OutputRoot.MQMD));
  CREATE FIELD OutputRoot."BLOB"."BLOB";
  DECLARE MSGB BLOB;
  SET MSGB = SUBSTRING(outMsg FROM HDRL +1);
  SET OutputRoot."BLOB"."BLOB" = MSGB;

END IF;
```

If you want to interrogate or manipulate a bit stream extracted from a database, you must re-create the original message tree. To re-create the XML message tree from the bit stream, you can use a CREATE statement with a PARSE clause. The output message can then be modified by the message flow.

For example, you might create a database table by using the following statement:

```
INSERT INTO Database.TABLE1(MSGDATA, MSGENCODING, MSGCCSID)
   VALUES (msgBitStream, inEncoding, inCCSID);
```

The following code snippet shows how to re-create the message tree in the XMLNS domain by using the data read from the table:

```
CALL CopyMessageHeaders();
  SET Environment.Variables.DBResult = THE( SELECT T.* FROM Database.TABLE1 AS T);
  DECLARE resultRef REFERENCE TO Environment.Variables.DBResult;
  IF LASTMOVE(resultRef) THEN
    DECLARE outCCSID INT resultRef.MSGCCSID;
    DECLARE outEncoding INT resultRef.MSGENCODING;
    DECLARE outMsg BLOB resultRef.MSGDATA;
    SET OutputRoot.Properties.CodedCharSetId = outCCSID;
    SET OutputRoot.Properties.Encoding = outEncoding;
    CREATE LASTCHILD OF OutputRoot DOMAIN('XMLNS') PARSE(outMsg, outEncoding, outCCSID);
```

```
    -- Now modify the message tree fields
    SET OutputRoot.XMLNS.A.B = 4;
    SET OutputRoot.XMLNS.A.E = 5;
  END IF;
```

In the following example, the data is held in a database column with a character data type, such as CHAR or VARCHAR. A cast is used to convert the data extracted from the database into BLOB format. If the bitstream data from the database does not need to be interrogated or manipulated by the message flow, the output message can be constructed in the BLOB domain, without any alteration.

```
CALL CopyMessageHeaders();
  SET Environment.Variables.DBResult = THE( SELECT T.* FROM Database.TABLE1 AS T);
  DECLARE resultRef REFERENCE TO Environment.Variables.DBResult;
  IF LASTMOVE(resultRef) THEN
    DECLARE outCCSID INT resultRef.MSGCCSID;
    DECLARE outMsg BLOB CAST(resultRef.MSGDATA AS BLOB CCSID outCCSID);
    SET OutputRoot.Properties.CodedCharSetId = outCCSID;
    SET OutputRoot.Properties.Encoding = resultRef.MSGENCODING;
    SET OutputRoot.BLOB.BLOB = outMsg;
  END IF;
```

In the following example, the data is held in a database column with a character data type, such as CHAR or VARCHAR. A cast is used to convert the data extracted from the database into BLOB format. To manipulate or interrogate this data within the message flow, you must re-create the original message tree. In this example, a `CREATE` statement with a `PARSE` clause is used to re-create the XML message tree in the `XMLNS` domain.

```
CALL CopyMessageHeaders();
  SET Environment.Variables.DBResult = THE( SELECT T.* FROM Database.TABLE1 AS T);
  DECLARE resultRef REFERENCE TO Environment.Variables.DBResult;
  IF LASTMOVE(resultRef) THEN
    DECLARE outCCSID INT resultRef.MSGCCSID;
    DECLARE outEncoding INT resultRef.MSGENCODING;
    DECLARE outMsg BLOB CAST(resultRef.MSGDATA AS BLOB CCSID outCCSID);
    SET OutputRoot.Properties.CodedCharSetId = outCCSID;
    SET OutputRoot.Properties.Encoding = outEncoding;
    CREATE LASTCHILD OF OutputRoot DOMAIN('XMLNS') PARSE(outMsg, outEncoding, outCCSID);
    -- Now modify the message tree fields
    SET OutputRoot.XMLNS.A.B = 4;
    SET OutputRoot.XMLNS.A.E = 5;
  END IF;
```

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"Filter node" on page 4452
Use the Filter node to route a message according to message content.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"DECLARE statement" on page 5117
Use the DECLARE statement to define a variable, the data type of the variable and, optionally, its initial value.

"DELETE statement" on page 5129
The DELETE statement detaches and destroys a portion of a message tree, allowing its memory to be reused. This statement is particularly useful when handling very large messages.

"SELECT function" on page 5260
The SELECT function combines, filters, and transforms complex message and database data.

"SET statement" on page 5159
The SET statement assigns a value to a variable.

"Data types of values from external databases" on page 5288
How database data types are implicitly cast to ESQL data types.

Accessing multiple database tables:

You can refer to multiple tables that you have created in the same database. Use the FROM clause on the SELECT statement to join the data from the two tables.

About this task

The following example assumes that you have two database tables called USERTABLE1 and USERTABLE2. Both tables have two char(6) data type columns (or equivalent).

USERTABLE1 contains two rows:

	Column1	**Column2**
Row 1	value1	value2
Row 2	value3	value4

USERTABLE2 contains two rows:

	Column3	Column4
Row 1	value5	value6
Row 2	value7	value8

All tables referenced by a single SELECT function must be in the same database. The database can be either the default (specified on the Data Source property of the node) or another database (specified on the FROM clause of the SELECT function).

Configure the Compute, Database, or Filter node that you are using to identify the database in which you have defined the tables. For example, if you are using the default database, right-click the node, select **Open ESQL**, and code the following ESQL statements in the module for this node:

```
SET OutputRoot.XML.Test.Result[] =
         (SELECT A.Column1 AS FirstColumn,
                 A.Column2 AS SecondColumn,
                 B.Column3 AS ThirdColumn,
                 B.Column4 AS FourthColumn
            FROM Database.USERTABLE1 AS A,
                 Database.USERTABLE2 AS B
           WHERE A.Column1 = 'value1' AND
                 B.Column4 = 'value8'
         );
```

This code results in the following output message content:

```
<Test>
  <Result>
    <FirstColumn>value1</FirstColumn>
    <SecondColumn>value2</SecondColumn>
    <ThirdColumn>value7</ThirdColumn>
    <FourthColumn>value8</FourthColumn>
  </Result>
</Test>
```

This example shows how to access data from two database tables. You can code more complex FROM clauses to access multiple database tables (although all the tables must be in the same database). You can also refer to one or more message trees, and can use SELECT to join tables with tables, messages with messages, or tables with messages. “Joining data from messages and database tables” on page 2531 provides an example of how to merge message data with data in a database table.

If you specify an ESQL function or procedure on the column identifier in the WHERE clause, it is processed as part of the database query and not as ESQL.

Consider the following example:

```
SET OutputRoot.XML.Test.Result =
   THE(SELECT ITEM T.Column1 FROM Database.USERTABLE1 AS T
   WHERE UPPER(T.Column2) = 'VALUE2');
```

This code attempts to return the rows where the value of `Column2` converted to uppercase is `VALUE2`. However, only the database manager can determine the value of `T.Column2` for any given row, therefore it cannot be processed by ESQL before the database query is issued, because the WHERE clause determines the rows that are returned to the message flow.

Therefore, the UPPER is passed to the database manager to be included as part of its processing. However, if the database manager cannot process the token within the SELECT statement, an error is returned.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

"Joining data from messages and database tables" on page 2531
You can use SELECT functions that interact with both message data and databases.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"Filter node" on page 4452
Use the Filter node to route a message according to message content.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"SELECT function" on page 5260
The SELECT function combines, filters, and transforms complex message and database data.

"SET statement" on page 5159
The SET statement assigns a value to a variable.

Changing database content:

You can use Compute, Database, and Filter nodes to change the contents of a database by updating, inserting, or deleting data.

About this task

The following ESQL code includes statements that show all three operations. This code is appropriate for a Database and Filter node; if you create this code for a Compute node, use the correlation name InputRoot in place of Root.

```
IF Root.XMLNS.TestCase.Action = 'INSERT' THEN
   INSERT INTO Database.STOCK (STOCK_ID, STOCK_DESC, STOCK_QTY_HELD,
   BROKER_BUY_PRICE, BROKER_SELL_PRICE, STOCK_HIGH_PRICE, STOCK_HIGH_DATE,
   STOCK_HIGH_TIME) VALUES
   (CAST(Root.XMLNS.TestCase.stock_id AS INTEGER),
    Root.XMLNS.TestCase.stock_desc,
    CAST(Root.XMLNS.TestCase.stock_qty_held AS DECIMAL),
    CAST(Root.XMLNS.TestCase.broker_buy_price AS DECIMAL),
    CAST(Root.XMLNS.TestCase.broker_sell_price AS DECIMAL),
    Root.XMLNS.TestCase.stock_high_price,
    CURRENT_DATE,
    CURRENT_TIME);

ELSEIF Root.XMLNS.TestCase.Action = 'DELETE' THEN

        DELETE FROM Database.STOCK WHERE STOCK.STOCK_ID =
                    CAST(Root.XMLNS.TestCase.stock_id AS INTEGER);

   ELSEIF Root.XMLNS.TestCase.Action = 'UPDATE' THEN

           UPDATE Database.STOCK as A SET STOCK_DESC = Root.XMLNS.TestCase.stock_desc
                  WHERE  A.STOCK_ID = CAST(Root.XMLNS.TestCase.stock_id AS INTEGER);
END IF;
```

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"Filter node" on page 4452
Use the Filter node to route a message according to message content.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

Checking returns to SELECT:

If a SELECT function returns no data, or no further data, this result is handled as a normal situation and no error code is set in SQLCODE, regardless of the setting of the `Throw Exception On Database Error` and `Treat Warnings As Errors` properties on the current node.

About this task

To recognize that a SELECT function has returned no data, include ESQL that checks what has been returned. You can use various methods:

1. EXISTS

   This ESQL returns a Boolean value that indicates if a SELECT function returned one or more values (TRUE), or none (FALSE).

   ```
   IF EXISTS(SELECT T.MYCOL FROM Database.MYTABLE) THEN
   ...
   ```

2. CARDINALITY

   If you expect an array in response to a SELECT, you can use CARDINALITY to calculate how many entries have been received.

   ```
   SET OutputRoot.XMLNS.Testcase.Results[] = (
       SELECT T.MYCOL FROM Database.MYTABLE)
   ......
   IF CARDINALITY (OutputRoot.XMLNS.Testcase.Results[])> 0 THEN
   ........
   ```

3. IS NULL

   If you have used either THE or ITEM keywords in your SELECT function, a scalar value is returned. If no rows have been returned, the value set is NULL. However, it is possible that the value NULL is contained within the column, and you might want to distinguish between these two cases.

   Distinguish between cases by including COALESCE in the SELECT function, for example:

   ```
   SET OutputRoot.XMLNS.Testcase.Results VALUE = THE (
      SELECT ITEM COALESCE(T.MYCOL, 'WAS NULL')
      FROM Database.MYTABLE);
   ```

   If this example returns the character string WAS NULL, it indicates that the column contained NULL, and not that no rows were returned.

In previous releases, an SQLCODE of 100 was set in most cases if no data, or no further data, was returned. An exception was raised by the broker if you chose to handle database errors in the message flow.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Exception list tree structure" on page 1066
The exception list tree is a part of the logical message tree in which the message flow writes information about exceptions that occur when a message is processed.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"Filter node" on page 4452
Use the Filter node to route a message according to message content.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"SET statement" on page 5159
The SET statement assigns a value to a variable.

"ESQL simple comparison operators" on page 5057
The simple comparison operators >, <, >=, <=, =, and <>.

"CARDINALITY function" on page 5238

Committing database updates:

When you create a message flow that interacts with databases, you can choose whether the updates that you make are committed when the current node has completed processing, or when the current invocation of the message flow has terminated.

About this task

The information in this topic does not apply to the DatabaseInput node. For more information about the DatabaseInput node, see “Event-based database integration” on page 2118.

For each node, select the appropriate option for the `Transaction` property to specify when its database updates are to be committed:

- Choose `Automatic` (the default) if you want updates made in this node to be committed or rolled back as part of the whole message flow. The actions that you define in the ESQL module are performed on the message and it continues through the message flow. If the message flow completes successfully, the updates are committed. If the message flow fails, the message and the database updates are rolled back.
- Choose `Commit` if you want to commit the action of the node on the database, irrespective of the success or failure of the message flow as a whole. The database update is committed when the node processing is successfully completed, that is, after all ESQL has been processed, even if the message flow itself detects an error in a subsequent node that causes the message to be rolled back.

The value that you choose is implemented for the database tables that you have updated. You cannot select a different value for each table.

If you have set `Transaction` to `Commit`, the behavior of the message flow and the commitment of database updates can be affected by the use of the PROPAGATE statement in the node's ESQL.

If you choose to include a PROPAGATE statement that generates one or more output messages from the node, the processing of the PROPAGATE statement is not considered complete until the entire path that the output message takes has completed. This path might include several other nodes, including one or more output nodes. Only then does the node that issues the PROPAGATE statement receive control back and its ESQL terminate. At that point, a database commit is performed, if appropriate.

If one of the nodes on the propagated path detects an error and throws an exception, the processing of the node in which you have coded the PROPAGATE statement never completes. If the error processing results in a rollback, the message flow and the database update in this node are rolled back. This behavior is consistent with the stated operation of the `Commit` option, but might not be the behavior that you expect.

Related concepts:

“Message flows overview” on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

“Designing a message flow” on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

“Defining message flow content” on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"Filter node" on page 4452
Use the Filter node to route a message according to message content.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"PROPAGATE statement" on page 5150
The PROPAGATE statement propagates a message to the downstream nodes.

Invoking stored procedures:

To invoke a procedure that is stored in a database, use the ESQL CALL statement. The stored procedure must be defined by a CREATE PROCEDURE statement that has a Language clause of DATABASE and an EXTERNAL NAME clause that identifies the name of the procedure in the database and, optionally, the database schema to which it belongs.

About this task

When you invoke a stored procedure with the CALL statement, the broker ensures that the ESQL definition and the database definition match:

- The external name of the procedure must match a procedure in the database.
- The number of parameters must be the same.
- The type of each parameter must be the same.
- The direction of each parameter (IN, OUT, INOUT) must be the same.

The following restrictions apply to the use of stored procedures:

- Overloaded procedures are not supported. (An overloaded procedure is one that has the same name as another procedure in the same database schema with a different number of parameters, or parameters with different types.) If the broker detects that a procedure has been overloaded, it raises an exception.
- In an Oracle stored procedure declaration, you are not permitted to constrain CHAR and VARCHAR2 parameters with a length, and NUMBER parameters with a precision or scale, or both. Use %TYPE when you declare CHAR, VARCHAR and NUMBER parameters to provide constraints on a formal parameter.

You can also invoke a database stored procedure or a database user-defined function from a Mapping node. See "Mapping a target element from database stored procedures" on page 2290 or "Mapping a target element from database user-defined functions" on page 2292.

Creating a stored procedure in ESQL:
About this task

When you define an ESQL procedure that corresponds to a database stored procedure, you can specify either a qualified name (where the qualifier is a database schema) or an unqualified name.

To create a stored procedure:

Procedure

1. Code a statement similar to this example to create an unqualified procedure:
   ```
   CREATE PROCEDURE myProc1(IN p1 CHAR) LANGUAGE DATABASE EXTERNAL NAME "myProc";
   ```
   The EXTERNAL NAME that you specify must match the definition you have created in the database, but you can specify any name you choose for the corresponding ESQL procedure.
2. Code a statement similar to this example to create a qualified procedure:
   ```
   CREATE PROCEDURE myProc2(IN p1 CHAR) LANGUAGE DATABASE EXTERNAL NAME "Schema1.myProc";
   ```
3. Code a statement similar to this example to create a qualified procedure in an Oracle package:
   ```
   CREATE PROCEDURE myProc3(IN p1 CHAR) LANGUAGE DATABASE EXTERNAL
                   NAME "mySchema.myPackage.myProc";
   ```

Results

For examples of stored procedure definitions in the database, see the "CREATE PROCEDURE statement" on page 5103.

Calling a stored procedure:
Procedure

1. Code a statement similar to this example to invoke an unqualified procedure:
   ```
   CALL myProc1('HelloWorld');
   ```
   Because it is not defined explicitly as belonging to any schema, the `myProc1` procedure must exist in the default schema (the name of which is the user name used to connect to the data source) or the command fails.
2. The following example calls the `myProc` procedure in schema `Schema1`.
   ```
   CALL myProc2('HelloWorld');
   ```
3. Code a statement similar to this example to invoke an unqualified procedure with a dynamic schema:
   ```
   DECLARE Schema2 char 'mySchema2';
   CALL myProc1('HelloWorld') IN Database.{'Schema2'};
   ```
   This statement calls the `myProc1` procedure in database `Schema2`, overriding the default "`username`" schema.

Calling a stored procedure that returns two result sets:
About this task

To call a stored procedure that takes one input parameter and returns one output parameter and two result sets:

Procedure

1. Define the procedure with a CREATE PROCEDURE statement that specifies one input parameter, one output parameter, and two result sets:

```
CREATE PROCEDURE myProc1 (IN P1 INT, OUT P2 INT)
  LANGUAGE DATABASE
  DYNAMIC RESULT SETS 2
  EXTERNAL NAME "myschema.myproc1";
```

2. To invoke the myProc1 procedure using a field reference, code:

```
/* using a field reference */
CALL myProc1(InVar1, OutVar2, Environment.ResultSet1[],
            OutputRoot.XMLNS.Test.ResultSet2[]);
```

3. To invoke the myProc1 procedure using a reference variable, code:

```
/* using a reference variable*/
DECLARE cursor REFERENCE TO OutputRoot.XMLNS.Test;

CALL myProc1(InVar1, cursor.OutVar2, cursor.ResultSet1[],
                     cursor.ResultSet2[]);
```

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL procedures" on page 2386
An ESQL procedure is a subroutine that has no return value. It can accept input parameters from, and return output parameters to, the caller.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"Filter node" on page 4452
Use the Filter node to route a message according to message content.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"CALL statement" on page 5077
The CALL statement calls (invokes) a routine.

"CREATE PROCEDURE statement" on page 5103
The CREATE PROCEDURE statement defines a callable function or procedure.

Coding ESQL to handle errors:

When you process messages in a message flow, errors can have a number of different causes and the message flow designer must decide how to handle those errors.

Introduction

When you process messages in message flows, errors can have the following causes:

- External causes; for example, the incoming message is syntactically invalid, a database used by the flow has been shut down, or the power supply to the machine on which the broker is running fails.
- Internal causes; for example, an attempt to insert a row into a database table fails because of a constraint check, or a character string that is read from a database cannot be converted to a number because it contains alphabetic characters.

  Internal errors can be caused by programs storing invalid data in the database, or by a flaw in the logic of a flow.

The message flow designer must decide how to handle errors.

Using default error-handling

The simplest strategy for handling ESQL errors is to do nothing, and use the broker's default behavior. The default behavior is to cut short the processing of the failing message, and to proceed to the next message. Input and output nodes provide options to control exactly what happens when processing is cut short.

If the input and output nodes are set to transactional mode, the broker restores the state before the message is processed:

1. The input message that has apparently been taken from the input queue is put back.
2. Any output messages that the flow has apparently written to output queues are discarded.

If the input and output nodes are not set to transactional mode:

1. The input message that was taken from the input queue is not put back.
2. Any output messages that the flow has written to output queues remain on the output queues.

Each of these strategies has its advantages. The transactional model preserves the consistency of data, while the non-transactional model maximizes the continuity of message processing. In the transactional model, the failing input message is put back onto the input queue, and the broker attempts to process it again. The most likely outcome of this scenario is that the message continues to fail until the retry limit is reached, at which point the message is placed on a dead letter queue. The reason for the failure to process the message is logged to the system event log (Windows) or syslog (UNIX). Therefore, the failing message holds up the processing of subsequent valid messages, and is left unprocessed by the broker.

Most databases operate transactionally so that all changes that are made to database tables are committed if the processing of the message succeeds, or rolled back if it fails, therefore maintaining the integrity of data. An exception to this situation is if the broker itself, or a database, fails (for example, the power to the

computers on which they are running is interrupted). In these cases, changes might be committed in some databases, but not in others, or the database changes might be committed but the input and output messages are not committed. If these possibilities concern you, make the flow coordinated and configure the databases that are involved.

Using customized error handling: The following list contains some general tips for creating customized error handlers.

- If you require something better than default error handling, the first step is to use a handler; see "DECLARE HANDLER statement" on page 5124. Create one handler per node to intercept all possible exceptions (or all those that you can predict).
- Having intercepted an error, the error handler can use whatever logic is appropriate to handle it. Alternatively, it can use a THROW statement or node to create an exception, which could be handled higher in the flow logic, or even reach the input node, causing the transaction to be rolled back; see "Throwing an exception" on page 2511.
- If a node generates an exception that is not caught by the handler, the flow is diverted to the Failure terminal, if one is connected, or handled by default error-handling if no Failure terminal is connected.

  Use Failure terminals to catch unhandled errors. Attach a simple logic flow to the Failure terminal. This logic flow could consist of a database or Compute node that writes a log record to a database (possibly including the message's bit stream), or writes a record to the event log. The flow could also contain an output node that writes the message to a special queue.

  The full exception tree is passed to any node that is connected to a Failure terminal; see "Exception list tree structure" on page 1066.
- Your error handlers are responsible for logging each error in an appropriate place, such as the system event log.

For a detailed description of the options that you can use to process errors in a message flow, see "Handling errors in message flows" on page 2823. For examples of what you can do, see "Throwing an exception" on page 2511 and "Capturing database state" on page 2512.

Writing code to detect errors

The following sections assume that the broker detects the error. It is possible, however, for the logic of the flow to detect an error. For example, when coding the flow logic, you could use the following elements:

- IF statements that are inserted specifically to detect situations that should not occur
- The ELSE clause of a case expression or statement to trap routes through the code that should not be possible

As an example of a flow logic-detected error, consider a field that has a range of possible integer values that indicate the type of message. It would not be good practice to leave to chance what would happen if a message were to arrive in which the field's value did not correspond to any known type of message. One way this situation could occur is if the system is upgraded to support extra types of message, but one part of the system is not upgraded.

Using your own logic to handle input messages that are not valid

Input messages that are syntactically invalid (and input messages that appear to be not valid because of erroneous message format information) are difficult to deal with, because the broker cannot determine the contents of the message. Typically, the best way to deal with these messages is to configure the input node to fully parse and validate the message. However, this configuration applies only to predefined messages; that is, MRM or IDoc.

If the input node is configured in this way, the following results are guaranteed if the input message cannot be parsed successfully:

- The input message never emerges from the node's normal output terminal (it goes to the Failure terminal).
- The input message never enters the main part of the message flow.
- The input message never causes any database updates.
- No messages are written to any output queues.

To deal with a failing message, connect a simple logic flow to the Failure terminal. The only disadvantage to this strategy is that if the normal flow does not require access to all of the message's fields, the forcing of complete parsing of the message affects performance.

Using your own logic to handle database errors

Database errors fall into three categories:

- The database is not working (for example, it is off line).
- The database is working but refuses your request (for example, a lock contention occurs).
- The database is working but it cannot do what you request (for example, read from a non-existent table).

If you require something better than default error handling, the first step is to use a handler (see "DECLARE HANDLER statement" on page 5124) to intercept the exception. The handler can determine the nature of the failure from the SQL state that is returned by the database.

A database is not working

If a database is not working at all, and is essential to the processing of messages, there is typically not much that you can do. The handler, having determined the cause, might complete one or more of the following actions:

- Use the RESIGNAL statement to re-throw the original error, therefore allowing the default error handler to take over
- Use a different database
- Write the message to a special output queue

  Be careful if you use an approach similar to this technique; the handler absorbs the exception, therefore all changes to other databases, or writes to queues, are committed.

A database refuses your request

The situation when a lock contention occurs is similar to the "Database not working" case because the database will have backed out *all* the database changes that you have made for the current message, not just the failing request. Therefore, unless you are sure that this was the only update, default error-handling is typically the best strategy, except possibly logging the error or passing the message to a special queue.

Impossible requests

An impossible request occurs when the database is working, but cannot complete the requested action. This type of error covers a wide variety of problems.

If, as discussed in the previous example, the database does not have a table of the name that the flow expects, default error-handling is typically the best strategy, except possibly logging the error or passing the message to a special queue.

Many other errors might be handled successfully, however. For example, an attempt to insert a row might fail because there is already such a row and the new row would be a duplicate. Or an attempt to update a row might fail because there is no such row (that is, the update action updated zero rows). In these cases, the handler can incorporate whatever logic you think appropriate. It might insert the missing row, or use the existing one (possibly making sure the values in it are suitable).

If you want an update of zero rows to be reported as an error, you must set the `Treat warnings as errors` property on the node to `true`, which is not the default setting.

Using your own logic to handle errors in output nodes

Errors that occur in MQOutput nodes report the nature of the error in the SQL state and give additional information in the *SQL native error* variable. Therefore, if something better than default error handling is required, the first step is to use a handler (see "DECLARE HANDLER statement" on page 5124) to intercept the exception. Such a handler typically surrounds only a single PROPAGATE statement.

Using your own logic to handle other errors

Besides those errors covered above, a variety of other errors can occur. For example, an arithmetic calculation might overflow, a cast might fail because of the unsuitability of the data, or an access to a message field might fail because of a type constraint. The broker offers two programming strategies for dealing with these types of error.

- The error causes an exception that is either handled or left to roll back the transaction.
- The failure is recorded as a special value that is tested for later.

In the absence of a type constraint, an attempt to access a non-existent message field results in the value null. Null values propagate through expressions, making the result null. Therefore, if an expression, however complex, does not return a null value, you know that all the values that it needed to calculate its result were not null.

Cast expressions can have a default clause. If there is a default clause, casts fail quietly; instead of throwing an exception, they simply return the default value. The default value could be an innocuous number (for example, zero for an integer), or a value that is clearly invalid in the context (for example, -1 for a customer number). Null might be particularly suitable because it is a value that is different from all others, and it will propagate through expressions without any possibility of the error condition being masked.

Handling errors in other nodes

Exceptions that occur in other nodes (that is, downstream of a PROPAGATE statement) might be caught by handlers in the normal way. Handling such errors intelligently, however, poses a problem: another node was involved in the original error, therefore another node, and not necessarily the originator of the exception, is likely to be involved in handling the error.

To help in these situations, the Database and Compute nodes have four terminals called Out1, Out2, Out3, and Out4. In addition, the syntax of the PROPAGATE statement includes target expression, message source, and control clauses to give more control over these terminals.

Related concepts:

"Exception list tree structure" on page 1066
The exception list tree is a part of the logical message tree in which the message flow writes information about exceptions that occur when a message is processed.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Capturing database state" on page 2512
If an error occurs when the broker accesses an external database, you can either let the broker throw an exception during node processing or use ESQL statements to process the exception within the node itself.

"Throwing an exception" on page 2511
If you detect an error or other situation in your message flow in which you want message processing to be ended, you can throw an exception in a message flow in two ways.

"Accessing the ExceptionList tree using ESQL" on page 2471
The ExceptionList tree has its own correlation name, ExceptionList, and you must use this in all ESQL statements that refer to or set the content of this tree.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

Related reference:

"DECLARE HANDLER statement" on page 5124
The DECLARE HANDLER statement creates an error handler for handling exceptions.

"PROPAGATE statement" on page 5150
The PROPAGATE statement propagates a message to the downstream nodes.

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

Throwing an exception:

If you detect an error or other situation in your message flow in which you want message processing to be ended, you can throw an exception in a message flow in two ways.

About this task

- Use the ESQL THROW EXCEPTION statement.

  Include the THROW statement anywhere in the ESQL module for a Compute, Database, or Filter node. Use the options on the statement to code your own data to be inserted into the exception.
- Include a THROW node in your message flow.

  Set the node properties to identify the source and content of the exception.

By using either statement options or node properties, you can specify a message identifier and values that are inserted into the message text to give additional information and identification to users who interpret the exception. You can specify any message in any catalog that is available to the broker. See "Using error logging from a user-defined extension" on page 3137 for more information.

The situations in which you might want to throw an exception are determined by the behavior of the message flow; decide when you design the message flow where this action might be appropriate. For example, you might want to examine the content of the input message to ensure that it meets criteria that cannot be detected by the input node (which might check that a particular message format is received).

The following example uses the "Example message" on page 5311 to show how you can use the ESQL THROW statement. To check that the invoice number is within a particular range, throw an exception for any invoice message received that does not fall in the valid range.

```
--Check for invoice number lower than permitted range
IF Body.Invoice.InvoiceNo < 100000 THEN
   THROW USER EXCEPTION CATALOG 'MyCatalog' MESSAGE 1234 VALUES
   ('Invoice number too low', Body.Invoice.InvoiceNo);

-- Check for invoice number higher than permitted range
ELSEIF Body.Invoice.InvoiceNo > 500000 THEN
      THROW USER EXCEPTION CATALOG 'MyCatalog' MESSAGE 1235 VALUES
   ('Invoice number too high', Body.Invoice.InvoiceNo);

ELSE DO
  -- invoice number is within permitted range
  -- complete normal processing
ENDIF;
```

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Exception list tree structure" on page 1066
The exception list tree is a part of the logical message tree in which the message flow writes information about exceptions that occur when a message is processed.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"Filter node" on page 4452
Use the Filter node to route a message according to message content.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"THROW statement" on page 5161
Use the THROW statement to generate a user exception.

Capturing database state:

If an error occurs when the broker accesses an external database, you can either let the broker throw an exception during node processing or use ESQL statements to process the exception within the node itself.

About this task

Letting the broker throw an exception during node processing is the default action; ESQL processing in the current node is abandoned. The exception is then propagated backwards through the message flow until an enclosing catch node, or the input node for this message flow, is reached. If the exception reaches the input node, an active transaction is rolled back.

Using ESQL statements to process the exception within the node itself requires an understanding of database return codes and a logical course of action to take when an error occurs. To enable this inline database error processing, you must clear the Filter, Database, or Compute node's `Throw Exception On Database Error` property. If you clear this property, the node sets the database state indicators SQLCODE, SQLSTATE, SQLNATIVEERROR, and SQLERRORTEXT, with appropriate information from the database manager instead of throwing an exception.

The indicators contain information only when an error (not a warning) occurs, unless you have selected the `Treat Warnings As Errors` property. If a database operation is successful, or returns success with information, the indicators contain their default success values.

You can use the values contained in these indicators in ESQL statements to make decisions about the action to take. You can access these indicators with the SQLCODE, SQLSTATE, SQLNATIVEERROR, and SQLERRORTEXT functions.

If you are attempting inline error processing, check the state indicators after each database statement is executed to ensure that you catch and assess all errors. When processing the indicators, if you meet an error that you cannot handle inline, you can raise a new exception either to deal with it upstream in a catch node, or to let it through to the input node so that the transaction is rolled back, for which you can use the ESQL THROW statement.

You might want to check for the special case in which a SELECT returns no data. This situation is not considered an error and SQLCODE is not set, therefore you must test explicitly for it; see “Checking returns to SELECT” on page 2500.

Using ESQL to access database state indicators

The following ESQL example shows how to use the four database state functions, and how to include the error information that is returned in an exception:

```
DECLARE SQLState1 CHARACTER;
DECLARE SQLErrorText1 CHARACTER;
DECLARE SQLCode1 INTEGER;
DECLARE SQLNativeError1 INTEGER;

-- Make a database insert to a table that does not exist --
INSERT INTO Database.DB2ADMIN.NONEXISTENTTABLE (KEY,QMGR,QNAME)
                              VALUES (45,'REG356','my TESTING 2');

--Retrieve the database return codes --
SET SQLState1 = SQLSTATE;
SET SQLCode1 = SQLCODE;
SET SQLErrorText1 = SQLERRORTEXT;
SET SQLNativeError1 = SQLNATIVEERROR;

--Use the THROW statement to back out the database and issue a user exception--
THROW USER EXCEPTION MESSAGE 2950 VALUES
( 'The SQL State' , SQLState1 , SQLCode1 , SQLNativeError1 ,
SQLErrorText1 );
```

You do not have to throw an exception when you detect a database error; you might prefer to save the error information returned in the local environment tree, and include a Filter node in your message flow that routes the message to error or success subflows according to the values saved.

The following sample program provides another example of ESQL that uses these database functions:

- Airline Reservations

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Exception list tree structure" on page 1066
The exception list tree is a part of the logical message tree in which the message flow writes information about exceptions that occur when a message is processed.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"Coding ESQL to handle errors" on page 2506
When you process messages in a message flow, errors can have a number of different causes and the message flow designer must decide how to handle those errors.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

"Checking returns to SELECT" on page 2500
If a SELECT function returns no data, or no further data, this result is handled as a normal situation and no error code is set in SQLCODE, regardless of the setting of the `Throw Exception On Database Error` and `Treat Warnings As Errors` properties

on the current node.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"Filter node" on page 4452
Use the Filter node to route a message according to message content.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"ESQL database state functions" on page 5168

"DECLARE statement" on page 5117
Use the DECLARE statement to define a variable, the data type of the variable and, optionally, its initial value.

"DECLARE HANDLER statement" on page 5124
The DECLARE HANDLER statement creates an error handler for handling exceptions.

"INSERT statement" on page 5135
The INSERT statement inserts a row into a database table.

"SET statement" on page 5159
The SET statement assigns a value to a variable.

"THROW statement" on page 5161
Use the THROW statement to generate a user exception.

Using the SELECT function:

The SELECT function is a convenient and powerful tool for accessing fields and transforming data in a message tree.

About this task

The following topics show by example how to use the SELECT function to transform a variety of messages. The examples are based on an XML input message that has been parsed in the XMLNS domain. However, the techniques shown in these topics can be applied to any message tree.

- "Transforming a simple message" on page 2516
- "Transforming a complex message" on page 2520
- "Returning a scalar value in a message" on page 2523
- "Joining data in a message" on page 2526
- "Translating data in a message" on page 2528
- "Joining data from messages and database tables" on page 2531

Related concepts:

"XMLNS parser" on page 1104
The XMLNS parser is a flexible, general-purpose XML parser.

Related reference:

"SELECT function" on page 5260
The SELECT function combines, filters, and transforms complex message and database data.

Transforming a simple message:

When you code the ESQL for a Compute node, use the SELECT function to transform simple messages.

About this task

This topic provides examples of simple message transformation. Review the examples and modify them for your own use. They are all based on the "Example message" on page 5311 as input.

Example

Consider the following ESQL:

```
SET OutputRoot.XMLNS.Data.Output[] =
   (SELECT R.Quantity, R.Author FROM InputRoot.XMLNS.Invoice.Purchases.Item[] AS R
   );
```

When this ESQL code processes the Invoice message, it produces the following output message:

```
<Data>
  <Output>
     <Quantity>2</Quantity>
     <Author>Neil Bradley</Author>
  </Output>
  <Output>
     <Quantity>1</Quantity>
     <Author>Don Chamberlin</Author>
  </Output>
  <Output>
     <Quantity>1</Quantity>
     <Author>Philip Heller, Simon Roberts</Author>
  </Output>
</Data>
```

Three Output fields are present, one for each Item field, because SELECT creates an item in its result list for each item described by its FROM list. Within each Output field, a Field is created for each field named in the SELECT clause. These fields are in the order in which they are specified within the SELECT clause, not in the order in which they appear in the incoming message.

The R that is introduced by the final AS keyword is known as a correlation name. It is a local variable that represents in turn each of the fields addressed by the FROM clause. The name chosen has no significance. In summary, this simple transform does two things:

1. It discards unwanted fields.
2. It guarantees the order of the fields.

You can perform the same transform with a procedural algorithm:

```
DECLARE i INTEGER 1;
DECLARE count INTEGER CARDINALITY(InputRoot.XMLNS.Invoice.Purchases.Item[]);

WHILE (i <= count)
   SET OutputRoot.XMLNS.Data.Output[i].Quantity = InputRoot.XMLNS.Invoice.Purchases.Item[i].Quantity;
   SET OutputRoot.XMLNS.Data.Output[i].Author   = InputRoot.XMLNS.Invoice.Purchases.Item[i].Author;
   SET i = i+1;
END WHILE;
```

These examples show that the SELECT version of the transform is much more concise. It also executes faster.

The following example shows a more advanced transformation:

```
SET OutputRoot.XMLNS.Data.Output[] =
    (SELECT R.Quantity AS Book.Quantity,
            R.Author   AS Book.Author
            FROM InputRoot.XMLNS.Invoice.Purchases.Item[] AS R
    );
```

In this transform, an AS clause is associated with each item in the SELECT clause. This clause gives each field in the result an explicit name rather than a field name that is inherited from the input. These names can be paths (that is, a dot-separated list of names), as shown in the example. The structure of the output message structure can be different from the input message. Using the same Invoice message, the result is:

```
<Data>
  <Output>
    <Book>
      <Quantity>2</Quantity>
      <Author>Neil Bradley</Author>
    </Book>
  </Output>
  <Output>
    <Book>
      <Quantity>1</Quantity>
      <Author>Don Chamberlin</Author>
    </Book>
  </Output>
  <Output>
    <Book>
      <Quantity>1</Quantity>
      <Author>Philip Heller, Simon Roberts</Author>
    </Book>
  </Output>
</Data>
```

The expressions in the SELECT clause can be of any complexity and there are no special restrictions. They can include operators, functions, and literals, and they can refer to variables or fields that are not related to the correlation name. The following example shows more complex expressions:

```
SET OutputRoot.XMLNS.Data.Output[] =
    (SELECT 'Start'                          AS Header,
            'Number of books:' || R.Quantity AS Book.Quantity,
            R.Author || ':Name and Surname'  AS Book.Author,
            'End'                            AS Trailer
            FROM InputRoot.XMLNS.Invoice.Purchases.Item[] AS R
    );
```

Using the same Invoice message, the result in this case is:

```
<Data>
 <Output>
  <Header>Start</Header>
  <Book>
   <Quantity>Number of books:2</Quantity>
   <Author>Neil Bradley:Name and Surname</Author>
  </Book>
  <Trailer>End</Trailer>
 </Output>
 <Output>
  <Header>Start</Header>
  <Book>
   <Quantity>Number of books:1</Quantity>
   <Author>Don Chamberlin:Name and Surname</Author>
  </Book>
  <Trailer>End</Trailer>
 </Output>
 <Output>
  <Header>Start</Header>
  <Book>
   <Quantity>Number of books:1</Quantity>
   <Author>Philip Heller, Simon Roberts:Name and Surname</Author>
  </Book>
  <Trailer>End</Trailer>
 </Output>
</Data>
```

As shown above, the AS clauses of the SELECT clause contain a path that describes the full name of the field that is to be created in the result. These paths can also specify (as is normal for paths) the type of field that is to be created. The following example transform specifies the field types. In this case, XML tagged data is transformed to XML attributes:

```
SET OutputRoot.XMLNS.Data.Output[] =
    (SELECT R.Quantity.* AS Book.(XML.Attribute)Quantity,
            R.Author.*   AS Book.(XML.Attribute)Author
            FROM InputRoot.XMLNS.Invoice.Purchases.Item[] AS R
    );
```

Using the same Invoice message, the result is:

```
<Data>
 <Output>
  <Book Quantity="2" Author="Neil Bradley"/>
 </Output>
 <Output>
  <Book Quantity="1" Author="Don Chamberlin"/>
 </Output>
 <Output>
  <Book Quantity="1" Author="Philip Heller, Simon Roberts"/>
 </Output>
</Data>
```

Finally, you can use a WHERE clause to eliminate some of the results. In the following example a WHERE clause is used to remove results in which a specific criterion is met. An entire result is either included or excluded:

```
SET OutputRoot.XMLNS.Data.Output[] =
    (SELECT R.Quantity AS Book.Quantity,
            R.Author   AS Book.Author
            FROM InputRoot.XMLNS.Invoice.Purchases.Item[] AS R
            WHERE R.Quantity = 2
    );
```

Using the same input message, the result is:

```
<Data>
 <Output>
  <Book>
   <Quantity>2</Quantity>
   <Author>Neil Bradley</Author>
  </Book>
 </Output>
</Data>
```

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"Filter node" on page 4452
Use the Filter node to route a message according to message content.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"CARDINALITY function" on page 5238

"DECLARE statement" on page 5117
Use the DECLARE statement to define a variable, the data type of the variable and, optionally, its initial value.

"SELECT function" on page 5260
The SELECT function combines, filters, and transforms complex message and database data.

"SET statement" on page 5159
The SET statement assigns a value to a variable.
"WHILE statement" on page 5167
The WHILE statement evaluates a condition expression, and if it is TRUE executes a sequence of statements.
"Example message" on page 5311

Transforming a complex message:

When you code the ESQL for a Compute node, use the SELECT function for complex message transformation.

About this task

This topic provides examples of complex message transformation. Review the examples and modify them for your own use. They are all based on the "Example message" on page 5311 as input.

Example

In this example, Invoice contains a variable number of Items. The transform is shown in the following example:

```
SET OutputRoot.XMLNS.Data.Statement[] =
    (SELECT I.Customer.Title                                    AS Customer.Title,
            I.Customer.FirstName || ' ' || I.Customer.LastName  AS Customer.Name,
            COALESCE(I.Customer.PhoneHome,'')                   AS Customer.Phone,
            (SELECT II.Title                          AS Desc,
                    CAST(II.UnitPrice AS FLOAT) * 1.6 AS Cost,
                    II.Quantity                       AS Qty
             FROM I.Purchases.Item[] AS II
             WHERE  II.UnitPrice> 0.0             )     AS Purchases.Article[],
            (SELECT SUM( CAST(II.UnitPrice AS FLOAT) *
                    CAST(II.Quantity  AS FLOAT) *
                    1.6                          )
             FROM I.Purchases.Item[] AS II      )       AS Amount,
            'Dollars'                                   AS Amount.(XML.Attribute)Currency

            FROM InputRoot.XMLNS.Invoice[] AS I
            WHERE I.Customer.LastName <> 'Brown'
    );
```

The output message that is generated is:

```
<Data>
 <Statement>
  <Customer>
   <Title>Mr</Title>
   <Name>Andrew Smith</Name>
   <Phone>01962818000</Phone>
  </Customer>
  <Purchases>
   <Article>
    <Desc Category="Computer" Form="Paperback" Edition="2">The XML Companion</Desc>
    <Cost>4.472E+1</Cost>
    <Qty>2</Qty>
   </Article>
   <Article>
    <Desc Category="Computer" Form="Paperback" Edition="2">
          A Complete Guide to DB2 Universal Database</Desc>
    <Cost>6.872E+1</Cost>
    <Qty>1</Qty>
   </Article>
   <Article>
    <Desc Category="Computer" Form="Hardcover" Edition="0">JAVA 2 Developers Handbook</Desc>
    <Cost>9.5984E+1</Cost>
    <Qty>1</Qty>
   </Article>
  </Purchases>
  <Amount Currency="Dollars">2.54144E+2</Amount>
 </Statement>
</Data>
```

This transform has nested SELECT clauses. The outer statement operates on the list of Invoices. The inner statement operates on the list of Items. The AS clause that is associated with the inner SELECT clause expects an array:

```
(SELECT II.Title                          AS Desc,
        CAST(II.UnitPrice AS FLOAT) * 1.6 AS Cost,
        II.Quantity                       AS Qty
 FROM I.Purchases.Item[] AS II
 WHERE  II.UnitPrice> 0.0             )
-- Note the use of [] in the next expression
   AS Purchases.Article[],
```

This statement tells the outer SELECT clause to expect a variable number of Items in each result. Each SELECT clause has its own correlation name: I for the outer SELECT clause and II for the inner one. Each SELECT clause typically uses its own correlation name, but the FROM clause in the inner SELECT clause refers to the correlation name of the outer SELECT clause:

```
(SELECT II.Title                          AS Desc,
        CAST(II.UnitPrice AS FLOAT) * 1.6 AS Cost,
        II.Quantity                       AS Qty
-- Note the use of I.Purchases.Item in the next expression
 FROM I.Purchases.Item[] AS II
 WHERE  II.UnitPrice> 0.0             ) AS Purchases.Article[],
```

This statement tells the inner SELECT clause to work with the current Invoice's Items. Both SELECT clauses contain WHERE clauses. The outer one uses one criterion to discard certain Customers, and the inner one uses a different criterion to discard certain Items. The example also shows the use of COALESCE to prevent missing input fields from causing the corresponding output field to be missing. Finally, it also uses the column function SUM to add together the value of all Items in each Invoice. Column functions are discussed in "Referencing columns in a database" on page 2489.

When the fields Desc are created, the whole of the input Title field is copied: the XML attributes and the field value. If you do not want these attributes in the output message, use the FIELDVALUE function to discard them; for example, code the following ESQL:

```
SET OutputRoot.XMLNS.Data.Statement[] =
    (SELECT I.Customer.Title                                  AS Customer.Title,
            I.Customer.FirstName || ' ' || I.Customer.LastName AS Customer.Name,
            COALESCE(I.Customer.PhoneHome,'')                 AS Customer.Phone,
            (SELECT FIELDVALUE(II.Title)                      AS Desc,
                    CAST(II.UnitPrice AS FLOAT) * 1.6 AS Cost,
                    II.Quantity                         AS Qty
             FROM I.Purchases.Item[] AS II
             WHERE  II.UnitPrice> 0.0             )      AS Purchases.Article[],
            (SELECT SUM( CAST(II.UnitPrice AS FLOAT) *
                    CAST(II.Quantity  AS FLOAT) *
                    1.6                           )
             FROM I.Purchases.Item[] AS II        ) AS Amount,
            'Dollars'                               AS Amount.(XML.Attribute)Currency

    FROM InputRoot.XMLNS.Invoice[] AS I
    WHERE I.Customer.LastName <> 'Brown'
    );
```

That code generates the following output message:

```
<Data>
 <Statement>
  <Customer>
   <Title>Mr</Title>
   <Name>Andrew Smith</Name>
   <Phone>01962818000</Phone>
  </Customer>
  <Purchases>
   <Article>
    <Desc>The XML Companion</Desc>
    <Cost>4.472E+1</Cost>
    <Qty>2</Qty>
   </Article>
   <Article>
    <Desc>A Complete Guide to DB2 Universal Database</Desc>
    <Cost>6.872E+1</Cost>
    <Qty>1</Qty>
   </Article>
   <Article>
    <Desc>JAVA 2 Developers Handbook</Desc>
    <Cost>9.5984E+1</Cost>
    <Qty>1</Qty>
   </Article>
  </Purchases>
  <Amount Currency="Dollars">2.54144E+2</Amount>
 </Statement>
</Data>
```

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

"Working with large XML messages" on page 2543
The tree representation of an XML message is typically bigger than the input bit stream. Manipulating a large message tree can require much storage but you can code ESQL statements that help to reduce the storage load on the broker.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"Filter node" on page 4452
Use the Filter node to route a message according to message content.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"CAST function" on page 5245

"COALESCE function" on page 5294

"SELECT function" on page 5260
The SELECT function combines, filters, and transforms complex message and database data.

"SET statement" on page 5159
The SET statement assigns a value to a variable.

"Example message" on page 5311

Returning a scalar value in a message:

Use a SELECT statement to return a scalar value by including both the **THE** and **ITEM** keywords.

About this task

For example:

```
1 + THE(SELECT ITEM T.a FROM Body.Test.A[] AS T WHERE T.b = '123')
```

Use of the ITEM keyword:
About this task

The following example shows the use of the ITEM keyword to select one item and create a single value.

```
SET OutputRoot.MQMD = InputRoot.MQMD;

SET OutputRoot.XMLNS.Test.Result[] =
   (SELECT ITEM T.UnitPrice FROM InputBody.Invoice.Purchases.Item[] AS T);
```

When the Invoice message is received as input, the ESQL shown generates the following output message:

```
<Test>
  <Result>27.95</Result>
  <Result>42.95</Result>
  <Result>59.99</Result>
</Test>
```

When the ITEM keyword is specified, the output message includes a list of scalar values. Compare this message to the one that is produced if the ITEM keyword is omitted, in which a list of fields (name-value pairs) is generated:

```
<Test>
  <Result>
    <UnitPrice>27.95</UnitPrice>
  </Result>
  <Result>
    <UnitPrice>42.95</UnitPrice>
  </Result>
  <Result>
    <UnitPrice>59.99</UnitPrice>
  </Result>
</Test>
```

Effects of the THE keyword:
About this task

The THE keyword converts a list containing one item to the item itself.

The two previous examples both specified a list as the source of the SELECT in the FROM clause (the field reference has [] at the end to indicate an array), so typically the SELECT function generates a list of results. Because of this behavior, you must specify a list as the target of the assignment (thus the "Result[]" as the target of the assignment). However, you often know that the WHERE clause that you specify as part of the SELECT returns TRUE for only one item in the list. In this case use the THE keyword.

The following example shows the effect of using the THE keyword:

```
SET OutputRoot.MQMD = InputRoot.MQMD;

SET OutputRoot.XMLNS.Test.Result =
    THE (SELECT T.Publisher, T.Author FROM InputBody.Invoice.Purchases.Item[]
         AS T WHERE T.UnitPrice = 42.95);
```

The THE keyword means that the target of the assignment becomes `OutputRoot.XMLNS.Test.Result` (the "[]" is not permitted). Its use generates the

following output message:

```
<Test>
  <Result>
    <Publisher>Morgan Kaufmann Publishers</Publisher>
    <Author>Don Chamberlin</Author>
  </Result>
</Test>
```

Selecting from a list of scalars:
About this task

Consider the following sample input message:

```
<Test>
 <A>1</A>
 <A>2</A>
 <A>3</A>
 <A>4</A>
 <A>5</A>
</Test>
```

If you code the following ESQL statements to process this message:

```
SET OutputRoot.XMLNS.Test.A[] =
  (SELECT ITEM A from InputBody.Test.A[]
   WHERE CAST(A AS INTEGER) BETWEEN 2 AND 4);
```

the following output message is generated:

```
      <A>2</A>
      <A>3</A>
      <A>4</A>
```

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database,

DatabaseInput, and Filter nodes.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"Filter node" on page 4452
Use the Filter node to route a message according to message content.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"SELECT function" on page 5260
The SELECT function combines, filters, and transforms complex message and database data.

"SET statement" on page 5159
The SET statement assigns a value to a variable.

"Example message" on page 5311

Joining data in a message:

The FROM clause of a SELECT function is not restricted to having one item. Specifying multiple items in the FROM clause produces the typical Cartesian product joining effect, in which the result includes an item for all combinations of items in the two lists.

About this task

Using the FROM clause in this way produces the same joining effect as standard SQL.

The Invoice message includes a set of customer details, payment details, and details of the purchases that the customer makes. Code the following ESQL to process the input "Example message" on page 5311:

```
SET OutputRoot.XMLNS.Items.Item[] =
   (SELECT D.LastName, D.Billing,
           P.UnitPrice, P.Quantity
    FROM InputBody.Invoice.Customer[] AS D,
         InputBody.Invoice.Purchases.Item[] AS P);
```

The following output message is generated:

```
<Items>
 <Item>
  <LastName>Smith</LastName>
  <Billing>
   <Address>14 High Street</Address>
   <Address>Hursley Village</Address>
   <Address>Hampshire</Address>
   <PostCode>SO213JR</PostCode>
  </Billing>
  <UnitPrice>27.95</UnitPrice>
  <Quantity>2</Quantity>
 </Item>
 <Item>
  <LastName>Smith</LastName>
  <Billing>
   <Address>14 High Street</Address>
   <Address>Hursley Village</Address>
   <Address>Hampshire</Address>
   <PostCode>SO213JR</PostCode>
  </Billing>
  <UnitPrice>42.95</UnitPrice>
  <Quantity>1</Quantity>
 </Item>
 <Item>
  <LastName>Smith</LastName>
  <Billing>
   <Address>14 High Street</Address>
   <Address>Hursley Village</Address>
   <Address>Hampshire</Address>
   <PostCode>SO213JR</PostCode>
  </Billing>
  <UnitPrice>59.99</UnitPrice>
  <Quantity>1</Quantity>
 </Item>
</Items>
```

Three results are produced, giving the number of descriptions in the first list (one) multiplied by the number of prices in the second (three). The results systematically work through all the combinations of the two lists. You can see this by looking at the *LastName* and *UnitPrice* fields selected from each result:

```
LastName Smith   UnitPrice 27.95
LastName Smith   UnitPrice 42.95
LastName Smith   UnitPrice 59.99
```

You can join data that occurs in a list and a non-list, or in two non-lists, and so on. For example:

```
OutputRoot.XMLNS.Test.Result1[] =
  (SELECT ... FROM InputBody.Test.A[], InputBody.Test.b);
OutputRoot.XMLNS.Test.Result1 =
  (SELECT ... FROM InputBody.Test.A, InputBody.Test.b);
```

The location of the [] in each case is significant. Any number of items can be specified in the FROM clause, not just one or two. If any of the items specify [] to indicate a list of items, the SELECT function returns a list of results (the list might contain only one item, but the SELECT function can return a list of items).

The target of the assignment must specify a list (so must end in []), or you must use the "THE function" on page 5242 if you know that the WHERE clause guarantees that only one combination is matched.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"Filter node" on page 4452
Use the Filter node to route a message according to message content.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"SELECT function" on page 5260
The SELECT function combines, filters, and transforms complex message and database data.

"SET statement" on page 5159
The SET statement assigns a value to a variable.

"THE function" on page 5242
The THE function returns the first element of a list.

"Example message" on page 5311

Translating data in a message:

You can translate data from one form to another.

About this task

A typical example of the requirement to translate data is if the items are known in one message by names, and in another message by numbers. For example:

```
Type Name          Type Code

Confectionary      2000
Newspapers         3000
Hardware           4000
```

Consider the following input message:

```
<Data>
  <Items>
    <Item>
      <Cat>1000</Cat>
      <Description>Milk Chocolate Bar</Description>
      <Type>Confectionary</Type>
    </Item>
    <Item>
      <Cat>1001</Cat>
      <Description>Daily Newspaper</Description>
      <Type>NewsPapers</Type>
    </Item>
    <Item>
      <Cat>1002</Cat>
      <Description>Kitchen Sink</Description>
      <Type>Hardware</Type>
    </Item>
  </Items>
  <TranslateTable>
    <Translate>
      <Name>Confectionary</Name>
      <Number>2000</Number>
    </Translate>
    <Translate>
      <Name>NewsPapers</Name>
      <Number>3000</Number>
    </Translate>
    <Translate>
      <Name>Hardware</Name>
      <Number>4000</Number>
    </Translate>
  </TranslateTable>
</Data>
```

This message has two sections; the first section is a list of items in which each item has a catalog number and a type; the second section is a table for translating between descriptive type names and numeric type codes. Include a Compute node with the following transform:

```
SET OutputRoot.XMLNS.Result.Items.Item[] =
   (SELECT M.Cat, M.Description, T.Number As Type
     FROM
       InputRoot.XMLNS.Data.Items.Item[]               As M,
       InputRoot.XMLNS.Data.TranslateTable.Translate[] As T
     WHERE M.Type = T.Name
   );
```

The following output message is generated:

```
<Result>
  <Items>
    <Item>
      <Cat>1000</Cat>
      <Description>Milk Chocolate Bar</Description>
      <Type>2000</Type>
    </Item>
    <Item>
      <Cat>1001</Cat>
      <Description>Daily Newspaper</Description>
      <Type>3000</Type>
    </Item>
    <Item>
      <Cat>1002</Cat>
      <Description>Kitchen Sink</Description>
      <Type>4000</Type>
    </Item>
  </Items>
</Result>
```

In the result, each type name has been converted to its corresponding code. In this example, both the data and the translate table were in the same message tree, although this is not a requirement. For example, the translate table could be coded in a database, or might have been set up in LocalEnvironment by a previous Compute node.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"Filter node" on page 4452
Use the Filter node to route a message according to message content.
"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.
"SELECT function" on page 5260
The SELECT function combines, filters, and transforms complex message and database data.
"SET statement" on page 5159
The SET statement assigns a value to a variable.

Joining data from messages and database tables:

You can use SELECT functions that interact with both message data and databases.

About this task

You can also nest a SELECT function that interacts with one type of data within a SELECT clause that interacts with the other type.

Consider the following input message, which contains invoice information for two customers:

```
<Data>
  <Invoice>
    <CustomerNumber>1234</CustomerNumber>
    <Item>
      <PartNumber>1</PartNumber>
      <Quantity>9876</Quantity>
    </Item>
    <Item>
      <PartNumber>2</PartNumber>
      <Quantity>8765</Quantity>
    </Item>
</Invoice>
  <Invoice>
    <CustomerNumber>2345</CustomerNumber>
    <Item>
      <PartNumber>2</PartNumber>
      <Quantity>7654</Quantity>
    </Item>
    <Item>
      <PartNumber>1</PartNumber>
    <Quantity>6543</Quantity>
    </Item>
</Invoice>
</Data>
```

Consider the following database tables, Prices and Addresses, and their contents:

```
PARTNO      PRICE
----------- ----------------------
          1            +2.50000E+001
          2            +6.50000E+00


PARTNO      STREET               CITY             COUNTRY
------      ------------------   --------------   -------
1234        22 Railway Cuttings  East Cheam       England
2345        The Warren           Watership Down   England
```

If you code the following ESQL transform:

```
-- Create a valid output message
SET OutputRoot.MQMD = InputRoot.MQMD;

-- Select suitable invoices
SET OutputRoot.XMLNS.Data.Statement[] =
   (SELECT I.CustomerNumber                          AS Customer.Number,
           A.Street                                  AS Customer.Street,
           A.City                                    AS Customer.Town,
           A.Country                                 AS Customer.Country,

        -- Select suitable items
          (SELECT II.PartNumber AS PartNumber,
                  II.Quantity   AS Quantity,
                  PI.Price      AS Price
           FROM Database.db2admin.Prices AS PI,
               I.Item[]                 AS II
           WHERE II.PartNumber = PI.PartNo     )    AS Purchases.Item[]

    FROM Database.db2admin.Addresses  AS A,
         InputRoot.XMLNS.Data.Invoice[] AS I

    WHERE I.CustomerNumber = A.PartNo
    );
```

the following output message is generated. The input message is augmented with the price and address information from the database table:

```
<Data>
  <Statement>
    <Customer>
      <Number>1234</Number>
      <Street>22 Railway Cuttings</Street>
      <Town>East Cheam</Town>
      <Country>England</Country>
    </Customer>
    <Purchases>
      <Item>
        <PartNumber>1</PartNumber>
        <Quantity>9876</Quantity>
        <Price>2.5E+1</Price>
      </Item>
      <Item>
        <PartNumber>2</PartNumber>
        <Quantity>8765</Quantity>
        <Price>6.5E+1</Price>
      </Item>
    </Purchases>
  </Statement>
  <Statement>
    <Customer>
      <Number>2345</Number>
      <Street>The Warren</Street>
      <Town>Watership Down</Town>
      <Country>England</Country>
    </Customer>
    <Purchases>
      <Item>
        <PartNumber>1</PartNumber>
        <Quantity>6543</Quantity>
        <Price>2.5E+1</Price></Item>
      <Item>
        <PartNumber>2</PartNumber>
        <Quantity>7654</Quantity>
        <Price>6.5E+1</Price>
      </Item>
    </Purchases>
  </Statement>
</Data>
```

You can nest the database SELECT clause within the message SELECT clause. In most cases, the code is not as efficient as the previous example, but you might find that it is better if the messages are small and the database tables are large.

```
-- Create a valid output message
SET OutputRoot.MQMD = InputRoot.MQMD;

-- Select suitable invoices
SET OutputRoot.XMLNS.Data.Statement[] =
    (SELECT I.CustomerNumber                  AS Customer.Number,

        -- Look up the address
        THE ( SELECT
                  A.Street,
                  A.City    AS Town,
                  A.Country
                FROM Database.db2admin.Addresses AS A
                WHERE A.PartNo = I.CustomerNumber
            )                               AS Customer,

        -- Select suitable items
        (SELECT
            II.PartNumber AS PartNumber,
            II.Quantity   AS Quantity,

            -- Look up the price
            THE (SELECT ITEM P.Price
              FROM Database.db2admin.Prices AS P
              WHERE P.PartNo = II.PartNumber
            )             AS Price

          FROM I.Item[] AS II           ) AS Purchases.Item[]

    FROM InputRoot.XMLNS.Data.Invoice[] AS I
    );
```

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"Filter node" on page 4452
Use the Filter node to route a message according to message content.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"SELECT function" on page 5260
The SELECT function combines, filters, and transforms complex message and database data.

"SET statement" on page 5159
The SET statement assigns a value to a variable.

Manipulating messages in the XML domains:

You can manipulate messages in the XML, XMLNS, and XMLNSC domains.

About this task

The following topics contain instructions about manipulating messages in the XMLNSC, XMLNS, and XML domains.

- "Working with XML messages"
- "Manipulating messages in the XMLNSC domain" on page 2546
- "Manipulating messages in the XMLNS domain" on page 2563
- "Manipulating messages in the XML domain" on page 2581

For information about dealing with MRM XML messages, see "Manipulating messages in the MRM domain" on page 2581.

Related concepts:

"XML parsers and domains" on page 1084
You can use XML domains to parse and write messages that conform to the W3C XML standard.

"Which XML parser should you use?" on page 1080
If your messages are general-purpose XML documents, you can use one of the dedicated XML domains (XMLNSC or XMLNS) to parse the message, or you can use the MRM domain to parse the message.

"XMLNSC parser" on page 1090
The XMLNSC parser is a flexible, general-purpose XML parser that offers high performance XML parsing and optional XML Schema validation.

"XMLNS parser" on page 1104
The XMLNS parser is a flexible, general-purpose XML parser.

"XML parser" on page 1110
The XML domain is very similar to the XMLNS domain, but the XML domain has no support for XML namespaces or opaque parsing.

Working with XML messages:

The following topics provide information about typical tasks for processing XML messages.

About this task

Some of this information is available publicly in Web pages and online tutorials. If you are new to XML, you will find it useful to also read about the XML standard.

- "Constructing an XML message tree"
- "Working with namespaces" on page 2537
- "Working with binary data" on page 2538
- "XMLNSC: Working with CData" on page 2539
- "XMLNSC: Working with XML messages and bit streams" on page 2541
- "Working with large XML messages" on page 2543

For details about XML Schema, see XML Schema Part 0: Primer on the World Wide Web Consortium (W3C) Web site.

Constructing an XML message tree:

When constructing an XML message tree, consider the order of fields in the tree.

About this task

Order of fields in the message tree

When you create an XML output message in a Compute node, the order of your lines of ESQL code is important, because the message elements are created in the order that you code them.

Consider the following XML message:

```
<Order>
  <ItemNo>1</ItemNo>
  <Quantity>2</Quantity>
</Order>
```

If you want to add a DocType Declaration to this, insert the DocType Declaration before you copy the input message to the output message.

For example:

```
SET OutputRoot.XMLNS.(XML.XmlDecl) = '';
SET OutputRoot.XMLNS.(XML.XmlDecl).(XML.Version) = '1.0';
SET OutputRoot.XMLNS.(XML.DocTypeDecl)Order ='';
SET OutputRoot.XMLNS.(XML.DocTypeDecl).(XML.SystemId)   = 'NewDtdName.dtd';
SET OutputRoot = InputRoot;
  -- more ESQL --
```

If you put the last statement to copy the input message before the XML-specific statements, the following XML is generated for the output message.

```
<Order>
  <ItemNo>1</ItemNo>
  <Quantity>2</Quantity>
</Order>
<?xml version="1.0"?>
```

This is not well-formed XML and causes an error when it is written from the message tree to a bit stream in the output node.

Setting the field type

If you copy a message tree from input to output without changing the domain, most of the syntax elements will be created by the parser (XMLNSC or XMLNS) and their field types will be correct. However, if you construct your message tree from a database query, or from another parser's message tree, you must ensure that you identify each syntax element correctly by using its field type. You can find full details of the field type constants used by XMLNSC and XMLNS in the following topics:

- "XMLNSC: Using field types" on page 1094
- "XML constructs" on page 4257

Working with namespaces:

The following example shows how to use ESQL to work with namespaces.

About this task

Namespace constants are declared at the start of the main module so that you can use prefixes in the ESQL statements instead of the full URIs of the namespaces. The namespace constants affect only the ESQL; they do not control the prefixes that are generated in the output message. The prefixes in the generated output message are controlled by namespace declarations in the message tree. You can include namespace declarations in the tree using the XML.NamespaceDecl field type. These elements are then used to generate namespace declarations in the output message.

When the output message is generated, if the parser encounters a namespace for which it has no corresponding namespace declaration, a prefix is automatically generated using prefixes of the form NSn where n is a positive integer.

```
CREATE COMPUTE MODULE xmlns_doc_flow_Compute
CREATE FUNCTION Main() RETURNS BOOLEAN
BEGIN
CALL CopyMessageHeaders();
-- Declaration of namespace constants --
These are only used by ESQL

DECLARE sp1 NAMESPACE 'http://www.ibm.com/space1';
DECLARE sp2 NAMESPACE 'http://www.ibm.com/space2';
DECLARE sp3 NAMESPACE 'http://www.ibm.com/space3';

-- Namespace declaration for prefix 'space1'
SET OutputRoot.XMLNS.message.(XML.NamespaceDecl)xmlns:space1 = 'http://www.ibm.com/space1';
SET OutputRoot.XMLNS.message.sp1:data1 = 'Hello!';

-- Default namespace declaration ( empty prefix )
SET OutputRoot.XMLNS.message.sp2:data2.(XML.NamespaceDecl)xmlns = 'http://www.ibm.com/space2';
SET OutputRoot.XMLNS.message.sp2:data2.sp2:subData1 = 'Hola!';
SET OutputRoot.XMLNS.message.sp2:data2.sp2:subData2 = 'Guten Tag!';
SET OutputRoot.XMLNS.message.sp3:data3 = 'Bonjour!';
RETURN TRUE;
END;

CREATE PROCEDURE CopyMessageHeaders()
BEGIN
DECLARE I INTEGER 1;
DECLARE J INTEGER CARDINALITY(InputRoot.*[]);
WHILE I < J DO SET OutputRoot.*[I] = InputRoot.*[I];
  SET I = I + 1;
END WHILE;
END;
END MODULE;
```

When this ESQL is processed, the following output message is generated:

```
<message xmlns:space1="http://www.ibm.com/space1">
 <space1:data1>Hello!</space1:data1>
 <data2 xmlns="http://www.ibm.com/space2">
  <subData1>Hola!</subData1>
  <subData2>Guten Tag!</subData2>
 </data2>
 <NS1:data3 xmlns="http://www.ibm.com/space3>Bonjour!</NS1:data3>
</message>
```

Working with binary data:

If you need to include binary data or non-valid characters in your XML documents, the safest method is to encode the data as a binary string.

About this task

Binary encodings for XML

There are two methods of encoding binary data in an XML document.

```
hexBinary: <nonXMLChars>000102030405060708080C0E0F</nonXMLChars>

base64Binary: <nonXMLChars>AAECAwQFBgcICwwODw==</nonXMLChars>
```

The base64Binary encoding makes better use of the available XML characters, and on average a base64-encoded binary field is 2/3 the size of its hexBinary equivalent. Base64Binary is widely used by the MIME format and by various XML-based standards.

You might prefer to use the simpler hexBinary encoding if you are sending data to applications that cannot decode base64 data, and if the increase in message size is not important.

Parsing binary data

The most straightforward way to parse any binary data is to use the XMLNSC parser with a message set:

1. Locate or construct an XML Schema that describes your input XML.
2. Import the XML Schema to create a message definition file.
3. In your message flow, set the node properties as follows:
   - On the Default page, set *Message Domain* to XMLNSC and *Message Set* to the name of your message set.
   - On the Validation page, set *Validation* to Content and Value.
   - In the XMLNSC properties, select the check box option *Build tree using XML Schema types*.

The XMLNSC parser automatically decodes your hexBinary or base64Binary data, being guided by the simple type of the element or attribute that contains the binary data. The message tree will contain the decoded BLOB value.

If you are using the XMLNS domain, you must parse the binary data as a string. It appears in the message tree as a CHARACTER value. If the data is encoded as hexBinary, you can use the ESQL CAST function to convert it to a BLOB value. If the data is encoded as base64Binary, the easiest approach is to use the function BASE64DECODE. For more information, see “BASE64DECODE function” on page 5290.

Generating binary data

You can generate binary data in your output XML in either hexBinary or base64Binary encoding.

For hexBinary, use the ESQL CAST statement to convert your BLOB data to a hexBinary string.

For base64Binary, you have two options:

- Call the function BASE64ENCODE. For more information, see "BASE64ENCODE function" on page 5291.
- Use the XMLNSC parser, and modify the type field on the syntax element, as shown in this example:

```
-- ESQL code to generate base64-encoded binary data
DECLARE myBLOB BLOB;
-- Populate myBLOB with your binary data
CREATE LASTCHILD OF OutputRoot.XMLNSC.message
       TYPE BITOR(XMLNSC.Attribute, XMLNSC.base64Binary)
       NAME myBase64Element
       VALUE myBLOB;
```

  Note that setting the field type to XMLNSC.base64Binary does not change the logical value in the message tree. In your message flow, it is still a BLOB, and if you ask for its string representation, it is reported as a hexBinary string. However, when the message tree is converted to a bit stream (in an output node, or by a call to ASBITSTREAM) the base64 conversion is performed automatically, and the output XML contains the correct base64 string.

XMLNSC: Working with CData:

A CData section can be used to embed an XML document within another XML document.

About this task

What is a CData section?

An XML element can contain text content:

```
<element>text content</element>
```

However, some characters cannot appear in that content. In particular, '<' and '&' both have special meaning to an XML parser. If they are included in the text content of an element, they change the meaning of the XML document.

For example, this is a badly formed XML document:

```
<element><text><content></element>
```

There are two ways to make the XML well-formed:

1. Use character entities:

   ```
   <element>&lt;text&gt;&lt;content&gt;</element>
   ```

2. Use a CData section:

   ```
   <element><![CDATA[<text><content>]]></element>
   ```

What can you use a CData section for?

In a CData section, you can include XML markup in the value of an element. However, non-valid XML characters cannot be included. Binary data also cannot be included in a CData section.

The most common use for CData is to embed one XML document within another. For example:

```
<outer>
    <embedXML>
        <![CDATA[<innerMsg></innerMsg>]]>
    </embedXML>
</outer>
```

You can even embed a badly-formed XML document in this way, because the XML parser does not attempt to parse the content of a CData section.

```
<outer>
    <embedXML>
        <![CDATA[<badXML></wrongClosingTag>]]>
    </embedXML>
</outer>
```

The following items are not valid within a CData section:

- Non-valid XML characters (see http://www.w3.org/TR/2006/REC-xml-20060816/#charsets)
- The text string ']]>' (because this terminates the CData section)

Because of these restrictions, do not use a CData section to include arbitrary text in your XML document, and do not try to use a CData section to hold binary data (unless it is encoded as hexBinary or base64Binary).

How do you add a CData section to an output XML message?

Consider the following input message :

```
 <TestCase>
    <Folder>
       <Field1>Value1</Field1>
       <Field2>Value2</Field2>
       <Field3>Value3</Field3>
    </Folder>
 </TestCase>
```

The following ESQL shows how to serialize a whole message:

```
 DECLARE wholeMsgBlob BLOB
   ASBITSTREAM(InputRoot.XMLNSC,
               InputRoot.Properties.Encoding,
               InputRoot.Properties.CodedCharSetId );
  DECLARE wholeMsgChar CHAR
   CAST(wholeMsgBlob AS CHAR CCSID InputRoot.Properties.CodedCharSetId);
 SET OutputRoot.XMLNSC.Output.(XMLNSC.CDataField)Field1 = wholeMsgChar;
```

This example serializes the InputRoot.XMLNSC.TestCase.Folder portion of the message tree.

If the output message tree were examined before an MQOutput node, this would show :

```
(0x01000010):XML        = (
  (0x01000000):Output = (
    (0x01000000):Field1 = (
      (0x02000001): = '<TestCase><Folder><Field1>Value1</Field1><Field2>Value2</Field2>
                      <Field3>Value3</Field3></Folder><Folder2><Field1>Value1</Field1>
```

```
                        <Field2>Value2</Field2><Field3>Value3</Field3><Folder2></TestCase>'
       )
       )
   )
```

As can be seen, each CData section contains a single scalar value that is the character representation of the portion of the XML message that is required.

This tree produces the following XML output message :

```
<Output>
 <Field1><![CDATA[<TestCase><Folder><Field1>Value 1</Field1>
                                    <Field2>Value 2</Field2>
                                    <Field3>Value 3</Field3></Folder>
                             <Folder2><Field1>Value 1</Field1>
                                      <Field2>Value 2</Field2>
                                      <Field3>Value 3</Field3></Folder2>
                  </TestCase>]]</Field1>
 </Output>
```

XMLNSC: Working with XML messages and bit streams:

Use the ASBITSTREAM function and the CREATE statement to manage XML message content.

About this task

The ASBITSTREAM function

If you code the ASBITSTREAM function with the parser mode option set to RootBitStream to parse a message tree to a bit stream, the result is an XML document that is built from the children of the target element in the normal way. This algorithm is identical to that used to generate the normal output bit stream. Because the target element is not included in the output bit stream, you must ensure that the children of the element follow the constraints for an XML document.

One constraint is that there must be only one body element in the message. You can use a well-formed bit stream obtained in this way to re-create the original tree using a CREATE statement with a PARSE clause.

If you code the ASBITSTREAM function with the parser mode option set to FolderBitStream to parse a message tree to a bit stream, the generated bit stream is an XML document built from the target element and its children. Any DocTypeDecl or XmlDecl elements are ignored, and the target element itself is included in the generated bit stream.

The advantage of this mode is that the target element becomes the body element of the document, and that body element can have multiple elements nested within it. Use this mode to obtain a bit stream description of arbitrary sub-trees owned by an XML parser. You can use bit streams obtained in this way to re-create the original tree using a CREATE statement with a PARSE clause, and a mode of FolderBitStream.

For further information about the ASBITSTREAM function, and some examples of its use, see “ASBITSTREAM function” on page 5224.

The CREATE statement with a PARSE clause

If you code a CREATE statement with a PARSE clause with the parser mode option set to RootBitStream to parse a bit stream to a message tree, the expected bit stream is a normal XML document. A field in the tree is created for each field in the document. This algorithm is identical to that used when parsing a bit stream from an input node. In particular, an element named 'XML', 'XMLNS', or 'XMLNSC' is created as the root element of the tree, and all the content in the message is created as children of that root.

If you code a CREATE statement with a PARSE clause with the parser mode option set to FolderBitStream to parse a bit stream to a message tree, the expected bit stream is a normal XML document. Any content outside the body element (such as an XML declaration or doctype) is discarded. The first element created during the parse corresponds to the body of the XML document, and from there the parse proceeds as normal.

For further information about the CREATE statement, and examples of its use, see "CREATE statement" on page 5082.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"Filter node" on page 4452
Use the Filter node to route a message according to message content.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"CREATE statement" on page 5082
The CREATE statement creates a new message field.
"ASBITSTREAM function" on page 5224
The ASBITSTREAM field function generates a bit stream for the subtree of a given field according to the rules of the parser that owns the field.

Working with large XML messages:

The tree representation of an XML message is typically bigger than the input bit stream. Manipulating a large message tree can require much storage but you can code ESQL statements that help to reduce the storage load on the broker.

About this task

When an input bit stream is parsed and a logical tree is created, the tree representation of an XML message is typically bigger, and in some cases much bigger, than the corresponding bit stream.

The reasons for this expansion include the following factors:
- The addition of the pointers that link the objects together
- Translation of character data into Unicode, which can double the size
- The inclusion of field names that might have been implicit in the bit stream
- The presence of control data that is associated with operation of the broker

Manipulating a large message tree can require much storage. If you design a message flow that handles large messages that are made up of repeating structures, you can code ESQL statements that help to reduce the storage load on the broker. These statements support both random and sequential access to the message, but assume that you do not need access to the whole message at one time.

These ESQL statements cause the broker to perform limited parsing of the message, and to keep in storage at one time, only that part of the message tree that reflects a single record. If your processing requires you to retain information from record to record (for example, to calculate a total price from a repeating structure of items in an order), you can either declare, initialize, and maintain ESQL variables, or you can save values in another part of the message tree; for example, in the local environment.

This technique reduces the memory that is used by the broker to the memory that is needed to hold the full input and output bit streams, plus the memory that is needed for the message trees of just one record. This technique also provides memory savings when even a few repeats are encountered in the message. The broker uses partial parsing and the ability to parse specified parts of the message tree, to and from the corresponding part of the bit stream.

To use these techniques in your Compute node, take any of the following steps.
- Copy the body of the input message as a bit stream to a special folder in the output message. This action creates a modifiable copy of the input message that is not parsed and therefore uses a minimum amount of memory.
- Avoid any inspection of the input message, which avoids the need to parse the message.
- Use a loop and a reference variable to step through the message one record at a time. For each record, use the following processes:
  – Use normal transforms to build a corresponding output subtree in a second special folder.

- Use the ASBITSTREAM function to generate a bit stream for the output subtree. The generated bit stream is stored in a BitStream element that is placed in the position in the output subtree that corresponds to its required position in the final bit stream.
- Use the DELETE statement to delete both the current input and output record message trees when you have completed their manipulation.
- When you have completed the processing of all records, detach the special folders so that they do not appear in the output bit stream.

You can vary these techniques to suit the processing that is required for your messages.

The following ESQL code provides an example of one implementation, and is a modification of the ESQL example in "Transforming a complex message" on page 2520. It uses a single SET statement with nested SELECT functions to transform a message that contains nested, repeating structures.

```
-- Copy the MQMD header
  SET OutputRoot.MQMD = InputRoot.MQMD;

-- Create a special folder in the output message to hold the input tree
-- Note : SourceMessageTree is the root element of an XML parser
  CREATE LASTCHILD OF OutputRoot.XMLNS.Data DOMAIN 'XMLNS' NAME 'SourceMessageTree';

-- Copy the input message to a special folder in the output message
-- Note : This is a root to root copy which will therefore not build trees
  SET OutputRoot.XMLNS.Data.SourceMessageTree = InputRoot.XMLNS;

-- Create a special folder in the output message to hold the output tree
  CREATE FIELD OutputRoot.XMLNS.Data.TargetMessageTree;

-- Prepare to loop through the purchased items
  DECLARE sourceCursor REFERENCE TO OutputRoot.XMLNS.Data.SourceMessageTree.Invoice;
  DECLARE targetCursor REFERENCE TO OutputRoot.XMLNS.Data.TargetMessageTree;
  DECLARE resultCursor REFERENCE TO OutputRoot.XMLNS.Data;
  DECLARE grandTotal   FLOAT     0.0e0;

-- Create a block so that it's easy to abandon processing
  ProcessInvoice: BEGIN
  -- If there are no Invoices in the input message, there is nothing to do
    IF NOT LASTMOVE(sourceCursor) THEN
      LEAVE ProcessInvoice;
    END IF;

  -- Loop through the invoices in the source tree
  InvoiceLoop : LOOP
    -- Inspect the current invoice and create a matching Statement
    SET targetCursor.Statement = THE (SELECT 'Monthly' AS (XML.Attribute)Type,
                                            'Full' AS (0x03000000)Style[1],
                                            I.Customer.FirstName AS Customer.Name,
                                            I.Customer.LastName AS Customer.Surname,
                                            (SELECT
                                              FIELDVALUE(II.Title) AS Title,
                                              CAST(II.UnitPrice AS FLOAT) * 1.6 AS Cost,
                                              II.Quantity AS Qty
                                            FROM I.Purchases.Item[] AS II
                                            WHERE II.UnitPrice> 0.0) AS Purchases.Article[],
                                            (SELECT
                                              SUM( CAST(II.UnitPrice AS FLOAT) *
                                                   CAST(II.Quantity  AS FLOAT) *
                                                   1.6                         )
                                            FROM I.Purchases.Item[] AS II) AS Amount,
                                            'Dollars' AS Amount.(XML.Attribute)Currency
                                          FROM sourceCursor AS I
                                          WHERE I.Customer.LastName <> 'White');
```

```
   -- Turn the current Statement into a bit stream
   DECLARE StatementBitStream BLOB
   ASBITSTREAM(targetCursor.Statement OPTIONS FolderBitStream);
   -- If the SELECT produced a result
   -- (that is, it was not filtered out by the WHERE clause),
   -- process the Statement
   IF StatementBitStream IS NOT NULL THEN
   -- create a field to hold the bit stream in the result tree
      CREATE LASTCHILD OF resultCursor
         Type  XML.BitStream
         NAME  'StatementBitStream'
         VALUE StatementBitStream;

   -- Add the current Statement's Amount to the grand total
   -- Note that the cast is necessary because of the behavior
   -- of the XML syntax element
      SET grandTotal = grandTotal
       + CAST(targetCursor.Statement.Amount AS FLOAT);
   END IF;

   -- Delete the real Statement tree leaving only the bit stream version
   DELETE FIELD targetCursor.Statement;

   -- Step onto the next Invoice,
   -- removing the previous invoice and any
   -- text elements that might have been
   -- interspersed with the Invoices

   REPEAT
     MOVE sourceCursor NEXTSIBLING;
     DELETE PREVIOUSSIBLING OF sourceCursor;
     UNTIL (FIELDNAME(sourceCursor) = 'Invoice')
      OR (LASTMOVE(sourceCursor) = FALSE)
   END REPEAT;

   -- If there are no more invoices to process, abandon the loop
   IF NOT LASTMOVE(sourceCursor) THEN
    LEAVE InvoiceLoop;
   END IF;

  END LOOP InvoiceLoop;
END ProcessInvoice;

-- Remove the temporary source and target folders
DELETE FIELD OutputRoot.XMLNS.Data.SourceMessageTree;
DELETE FIELD OutputRoot.XMLNS.Data.TargetMessageTree;

-- Finally add the grand total
SET resultCursor.GrandTotal = grandTotal;
```

This ESQL code produces the following output message:

```
<Data>
 <Statement Type="Monthly" Style="Full">
  <Customer>
   <Name>Andrew</Name>
   <Surname>Smith</Surname>
   <Title>Mr</Title>
  </Customer>
  <Purchases>
   <Article>
    <Title>The XML Companion</Title>
    <Cost>4.472E+1</Cost>
    <Qty>2</Qty>
   </Article>
   <Article>
```

```
    <Title>A Complete Guide to DB2 Universal Database</Title>
    <Cost>6.872E+1</Cost>
    <Qty>1</Qty>
   </Article>
   <Article>
    <Title>JAVA 2 Developers Handbook</Title>
    <Cost>9.5984E+1</Cost>
    <Qty>1</Qty>
   </Article>
  </Purchases>
  <Amount Currency="Dollars">2.54144E+2</Amount>
  </Statement>
  <GrandTotal>2.54144E+2</GrandTotal>
 </Data>
```

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

"Transforming a complex message" on page 2520
When you code the ESQL for a Compute node, use the SELECT function for complex message transformation.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"DELETE statement" on page 5129
The DELETE statement detaches and destroys a portion of a message tree, allowing its memory to be reused. This statement is particularly useful when handling very large messages.

"ASBITSTREAM function" on page 5224
The ASBITSTREAM field function generates a bit stream for the subtree of a given field according to the rules of the parser that owns the field.

Manipulating messages in the XMLNSC domain:

If you are writing ESQL to process messages in the XMLNSC domain, it is helpful to learn about the structure of the message tree that the XMLNSC parser builds.

About this task

The topics in this section provide information about how to write ESQL for processing messages that belong to the XMLNSC domain, and that are parsed by the XMLNSC parser. For background information, see "XMLNSC parser" on page 1090.

The following topics provide detailed information about the structure of the message tree that the XMLNSC parser builds, and the field types that it uses.

- "XMLNSC: The XML declaration" on page 2548
- "XMLNSC: The inline DTD" on page 2549
- "XMLNSC: The message body" on page 2549
- "XMLNSC: XML Schema support" on page 2559

If you are migrating from XML, XMLNS, or MRM XML, see "Migrating to XMLNSC" on page 2562.

For further information about processing XML messages, see "Working with XML messages" on page 2535.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"XML parsers and domains" on page 1084
You can use XML domains to parse and write messages that conform to the W3C XML standard.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Manipulating messages in the XML domain" on page 2581
The XML parser is like the XMLNS parser, but the XML parser has no support for namespaces or opaque parsing.

"Working with XML messages" on page 2535
The following topics provide information about typical tasks for processing XML messages.

Related reference:

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"DECLARE statement" on page 5117
Use the DECLARE statement to define a variable, the data type of the variable and, optionally, its initial value.

"SET statement" on page 5159
The SET statement assigns a value to a variable.

XMLNSC: The XML declaration:

The XML declaration is represented in the message tree by a syntax element with field type XMLNSC.XMLDeclaration.

If an XML declaration is created by the XMLNSC parser, its name is 'XmlDeclaration'. However, when a message tree is being produced, the name is not important: the XMLNSC parser recognizes this syntax element by its field type only. The following example shows a typical declaration:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE s1 PUBLIC "http://www.ibm.com/example.dtd" "example.dtd">
<s1>........</s1>
```

The XML Declaration has three optional attributes; Version, Standalone, and Encoding. The XMLNSC parser does not define special field types for these attributes. Instead, they are identified by their name, and by their position as a child of the XML Declaration element.

ESQL example code to create an XML declaration

To construct the XML declaration that is shown in the previous example, code the following ESQL statements:

```
CREATE FIRSTCHILD OF OutputRoot.XMLNSC TYPE XMLNSC.XmlDeclaration NAME 'XmlDeclaration';
SET OutputRoot.XMLNSC.(XMLNSC.XmlDeclaration)*.(XMLNSC.Attribute)Version = '1.0';
SET OutputRoot.XMLNSC.(XMLNSC.XmlDeclaration)*.(XMLNSC.Attribute)Encoding = 'UTF-8';
SET OutputRoot.XMLNSC.(XMLNSC.XmlDeclaration)*.(XMLNSC.Attribute)StandAlone = 'yes';
```

The first line is optional; if it is omitted, the XMLNSC.XMLDeclaration element is automatically created when it is referenced by the second line.

Java example code to create an XML declaration

To construct the XML declaration that is shown in the previous example, write the following Java code:

```
//Create the XML domain root node
MBElement xmlRoot =
 root.createElementAsLastChild(MbXMLNSC.PARSER_NAME);
//Create the XML declaration parent node
MbElement xmlDecl =
 xmlRoot.createElementAsFirstChild(MbXMLNSC.XML_DECLARATION);

xmlDecl.setName("XmlDeclaration");

MbElement version =
 xmlDecl.CreateElementAsFirstChild(MbXMLNSC.ATTRIBUTE, "Version", "1.0");
MbElement encoding =
 xmlDecl.CreateElementAsFirstChild(MbXMLNSC.ATTRIBUTE, "Encoding", "utf-8");
MbElement standalone =
 xmlDecl.CreateElementAsFirstChild(MbXMLNSC.ATTRIBUTE, "Standalone", "yes");
```

Note: In both the ESQL example and the Java example, 'Version', 'StandAlone', and 'Encoding' can all be written in lowercase.

XMLNSC: The inline DTD:

When you parse an XML document that has an inline DTD, the XMLNSC parser does not put the DTD information into the message tree. However, by using ESQL code, you can add XML entity definitions to the message tree, and these definitions are used when the message tree is produced by the XMLNSC parser.

ESQL example code for entity definition and entity reference

This example assumes that InputRoot.XMLNSC has been created from the following XML message:

```
<BookInfo dtn="BookInfo" edn="author" edv="A.N.Other"/>
```

The following output message is generated:

```
<!DOCTYPE BookInfo [<!ENTITY author "A.N.Other">]>
<BookInfo><entref>&author;</entref></BookInfo>
```

The ESQL to create the output message is:

```
DECLARE cursor REFERENCE TO InputRoot.XMLNSC.BookInfo;
DECLARE docTypeName CHARACTER cursor.dtn;
DECLARE authorRef CHARACTER 'author';
-- Create <!DOCTYPE BOOKInfo ...
SET OutputRoot.XMLNSC.(XMLNSC.DocumentType)* NAME = docTypeName;
-- Create  <!ENTITY author "A.N.Other" > ...
SET OutputRoot.XMLNSC.(XMLNSC.DocumentType){docTypeName}.(XMLNSC.EntityDefinition) {authorRef} =
cursor.edv;
-- Create the entity reference
SET OutputRoot.XMLNSC.(XMLNSC.Folder){docTypeName}.(XMLNSC.EntityReference)entref = authorRef;
```

Related concepts:

"XMLNSC parser" on page 1090
The XMLNSC parser is a flexible, general-purpose XML parser that offers high performance XML parsing and optional XML Schema validation.

"XMLNSC DTD support" on page 1103
The input XML message might contain an inline DTD.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

XMLNSC: The message body:

The XMLNSC parser builds a message tree from the body of an XML document.

The following topics describe how the XMLNSC parser builds a message tree from the body of an XML document:

- "XMLNSC: Using field types" on page 1094
- "XMLNSC: Attributes and elements" on page 2552
- "XMLNSC: Namespace declarations" on page 2554
- "XMLNSC: Element values and mixed content" on page 2556
- "XMLNSC: Comments and Processing Instructions" on page 2558

Related tasks:

"Manipulating messages in the XMLNSC domain" on page 2546
If you are writing ESQL to process messages in the XMLNSC domain, it is helpful to learn about the structure of the message tree that the XMLNSC parser builds.

XMLNSC: Using field types:

The XMLNSC parser sets the field type on every syntax element that it creates.

The field type indicates the type of XML construct that the element represents. The XMLNSC parser uses the field type when writing a message tree. The field type can be set by using ESQL or Java to control the output XML. The field types that are used by the XMLNSC parser must be referenced by using constants with names that are prefixed by 'XMLNSC.'

Tip: Field type constants that have the prefix 'XML.' are for use with the XMLNS and XML parsers only, and are not valid with the XMLNSC or MRM parsers.

Field types for creating syntax elements

Use the following field type constants to create syntax elements in the message tree. The XMLNSC parser uses these values when creating a message tree from an input message.

XML construct	XMLNSC Field Type constant	Value
Simple Element	XMLNSC.Field XMLNSC.CDataField	0x03000000 0x03000001
Attribute	XMLNSC.SingleAttribute XMLNSC.Attribute	0x03000101 0x03000100
Mixed content	XMLNSC.Value XMLNSC.CDataValue	0x02000000 0x02000001
Namespace Declaration	XMLNSC.SingleNamespaceDecl XMLNSC.NamespaceDecl	0x03000102 0x03000103
Complex element	XMLNSC.Folder	0x01000000
Inline DTD	XMLNSC.DocumentType	0x01000300
XML declaration	XMLNSC.XmlDeclaration	0x01000400
Entity reference	XMLNSC.EntityReference	0x02000100
Entity definition	XMLNSC.SingleEntityDefinition XMLNSC.EntityDefinition	0x03000301 0x03000300
Comment	XMLNSC.Comment	0x03000400
Processing Instruction	XMLNSC.ProcessingInstruction	0x03000401

Field types for path expressions (generic field types)

Use the following field type constants when querying the message tree by using a path expression; for example:

```
SET str = FIELDVALUE(InputRoot.e1.(XMLNSC.Attribute)attr1)
```

It is good practice to specify field types when querying a message tree built by the XMLNSC parser. This makes your ESQL code more specific and more readable, and it avoids incorrect results in some cases. However, care is required when choosing which field type constant to use. When you use the XMLNSC parser, use

a generic field type constant when querying the message tree. This allows your path expression to tolerate variations in the input XML.

The generic field type constants are listed in the following table:

XML construct	XMLNSC Field Type constant	Purpose
Tag	XMLNSC.Element	Matches any tag, whether it contains child tags (XMLNSC.Folder) or a value (XMLNSC.Field)
Element	XMLNSC.Field	Matches a tag which contains normal text, CData, or a mixture of both. Does not match tags which contain child tags.
Attribute	XMLNSC.Attribute	Matches single-quoted and double-quoted attributes
Mixed content	XMLNSC.Value	Matches normal text, CData, or a mixture of both
XML Declaration	XMLNSC.NamespaceDecl	Matches single- and double-quoted declarations

If you write

```
InputRoot.e1.(XMLNSC.DoubleAttribute)attrName
```

your path expression does not match a single-quoted attribute. If you use the generic field type constant *XMLNSC.Attribute*, your message flow works with either single-quoted or double-quoted attributes.

Note that you should always use the field type constants and not their numeric values.

Field types for controlling output format

The following field types are provided for XML Schema and base64 support. Do not use these field type constants in path expressions; use them in conjunction with XMLNSC.Attribute and XMLNSC.Field to indicate the required output format for DATE and BLOB values. See "XMLNSC: XML Schema support" on page 2559 for further information.

XMLNSC Field Type constant	Purpose	Value
XMLNSC.gYear	The value must be a DATE. If the field type includes this value, the DATE value is produced by using the XML Schema gYear format.	0x00000010
XMLNSC.gYearMonth	The value must be a DATE. If the field type includes this value, the DATE value is produced by using the XML Schema gYearMonth format.	0x00000040
XMLNSC.gMonth	The value must be a DATE. If the field type includes this value, the DATE value is produced by using the XML Schema gMonth format.	0x00000020
XMLNSC.gMonthDay	The value must be a DATE. If the field type includes this value, the DATE value is produced by using the XML Schema gMonthDay format.	0x00000050

XMLNSC Field Type constant	Purpose	Value
XMLNSC.gDay	The value must be a DATE. If the field type includes this value, the DATE value is produced by using the XML Schema gDay format.	0x00000030
XMLNSC.base64Binary	The value must be a BLOB. The value is produced with base64 encoding.	0x00000060
XMLNSC.List	The element must be XMLNSC.Attribute or XMLNSC.Field. If the field type includes this value, the values of all child elements in the message tree are produced as a space-separated list.	0x00000070

Field types for direct output

Use the following field types to produce pre-constructed segments of an XML document. Character escaping is not done; therefore, take extra care not to construct a badly-formed output document. Use these constants only after carefully exploring alternative solutions.

XMLNSC Field Type constant	Purpose	Value
XMLNSC.Bitstream	The value of this syntax element must be a BLOB. The value is written directly to the output bit stream. For more information about its usage, see "Working with large XML messages" on page 2543.	0x03000200
XMLNSC.AsisElementContent	The value of this syntax element must be CHARACTER. The value is written directly to the output bit stream. No character substitutions are performed. Use this element with care.	0x03000600

Related concepts:

"XMLNSC: The message body" on page 2549
The XMLNSC parser builds a message tree from the body of an XML document.

"XMLNSC: Element values and mixed content" on page 2556
The XMLNSC parser is a compact parser; therefore, an element with single content is parsed as a single syntax element. When an element has both child elements and some text, the text is called *mixed content*.

"XMLNSC: Comments and Processing Instructions" on page 2558
The XMLNSC parser discards comments and processing instructions because both comments and processing instructions are auxiliary information with no business meaning.

Related reference:

"XMLNSC: Attributes and elements"
The XMLNSC parser uses field types to represent attributes and elements.

"XMLNSC: Namespace declarations" on page 2554
The XMLNSC parser provides full support for namespaces.

XMLNSC: Attributes and elements:

The XMLNSC parser uses field types to represent attributes and elements.

Use the following field type constants when creating your own syntax elements in the message tree.

Table 25. Specific field type constants

XML Construct	XMLNSC Field Type constant	Value
Complex element	XMLNSC.Folder	0x01000000
Simple element	`XMLNSC.Field` `XMLNSC.CDataField`	`0x02000000` `0x02000001`
Attribute	`XMLNSC.SingleAttribute` `XMLNSC.Attribute`	`0x03000100` `0x03000101`

When accessing elements and attributes in the message tree, use generic field type constants which match all of the alternative values. Because there is only one type of Folder element, it is safe to use XMLNSC.Folder when querying the message tree.

Table 26. Generic field type constants

XML Construct	XMLNSC Field Type constant	Purpose
Element	XMLNSC.Field	Matches elements that contain normal text, CData, or a mixture of both
Attribute	XMLNSC.Attribute	Matches both single-quoted and double-quoted attributes

ESQL code examples

The following examples use this XML message:

```
<root id="12345">
    <id>ABCDE</id>
</root>
```

Note that the message contains an attribute and an element with the same name.

Example 1 : Query the value of an XML element

```
SET value = FIELDVALUE(InputRoot.XMLNSC.root.(XMLNSC.Field)id)
```

The result is that value is set to 'ABCDE'.

Example 2 : Query the value of an XML attribute

```
SET value = FIELDVALUE(InputRoot.XMLNSC.root.(XMLNSC.Attribute)id)
```

The result is that value is set to '12345'.

Example 3 : Create the example message by using ESQL

```
CREATE LASTCHILD OF OutputRoot.XMLNSC Type XMLNSC.Folder Name 'root';
-- Note : XMLNSC.Attribute could be used here as well.
SET OuputRoot.XMLNSC.root.(XMLNSC.Attribute)id = '12345';
SET OuputRoot.XMLNSC.root.(XMLNSC.Field)id = 'ABCDE';
```

The first line is optional because the element 'root' is created automatically by the following line if it does not already exist.

Related concepts:

"XMLNSC: The message body" on page 2549
The XMLNSC parser builds a message tree from the body of an XML document.

"XMLNSC: Element values and mixed content" on page 2556
The XMLNSC parser is a compact parser; therefore, an element with single content is parsed as a single syntax element. When an element has both child elements and some text, the text is called *mixed content*.

"XMLNSC: Comments and Processing Instructions" on page 2558
The XMLNSC parser discards comments and processing instructions because both comments and processing instructions are auxiliary information with no business meaning.

Related reference:

"XMLNSC: Using field types" on page 1094
The XMLNSC parser sets the field type on every syntax element that it creates.

"XMLNSC: Namespace declarations"
The XMLNSC parser provides full support for namespaces.

XMLNSC: Namespace declarations:

The XMLNSC parser provides full support for namespaces.

The XMLNSC parser sets the correct namespace on every syntax element that it creates while parsing a message, and stores namespace declarations in the message tree. The parser uses the namespace declarations to select the appropriate prefixes when outputting the message tree.

The XMLNSC parser uses the following field types to represent namespace declarations. Use the field type constants that are listed in this table when you create namespace declarations in the message tree.

Table 27. Specific field type constants

XML construct	XMLNSC field type constant	Value
Namespace declaration	`XMLNSC.SingleNamespaceDecl` `XMLNSC.DoubleNamespaceDecl`	`0x03000102` `0x03000103`

When accessing elements and attributes in the message tree, do not use the constants that are listed in the previous table; instead, use the generic field type constant that matches both of the values in the table above.

Table 28. Generic field type constants

XML construct	XMLNSC field type constant	Purpose
Namespace declaration	XMLNSC.NamespaceDecl	Matches namespace declarations in both single and double quotation marks

ESQL code examples

Example 1: Declaring a non-empty prefix

```
DECLARE space1 NAMESPACE 'namespace1';
SET OutputRoot.XMLNSC.space1:root.(XMLNSC.NamespaceDecl)xmlns:ns1 = space1;
SET OutputRoot.XMLNSC.space1:root.space1:example = 'ABCDE';
```

This creates the following XML message:

```
<ns1:root xmlns:ns1="namespace1">
    <ns1:example>ABCDE</ns1:example>
</ns1:root>
```

Note that the NAMESPACE constant `space1` is just a local variable in the ESQL; it does not affect the namespace prefix `ns1` that is defined by the NameSpaceDecl element and appears in the output message.

However, as shown here, `space1` can be used to initialize the NameSpaceDecl for `ns1`. This avoids the need to duplicate the namespace URI ('namespace1' in this example), which in practice is typically a much longer string.

Example 2: Declaring an empty prefix

```
DECLARE space1 NAMESPACE 'namespace1';
SET OutputRoot.XMLNSC.space1:root.(XMLNSC.NamespaceDecl)xmlns = space1;
SET OutputRoot.XMLNSC.space1:root.space1:example = 'ABCDE';
```

This creates the following XML message:

```
<root xmlns="namespace1">
    <example>ABCDE</example>
</root>
```

Note that the syntax elements `root` and `example` must have a non-empty namespace. The default namespace declaration means that any child element without a prefix is in the namespace `namespace1`.

Example 3: Example of incorrect usage

```
DECLARE space1 NAMESPACE 'namespace1';
SET OutputRoot.XMLNSC.root.(XMLNSC.NamespaceDecl)xmlns = space1;
SET OutputRoot.XMLNSC.root.example = 'ABCDE';
```

This example causes the XMLNSC parser to issue the message BIP5014 when it attempts to create the message tree. The elements `root` and `example` are both within the scope of the default namespace declaration; therefore, in ESQL, these elements must be qualified by a namespace prefix bound to that namespace.

Example 4: Adding a namespace declaration with a prefix

```
SET OutputRoot.(XMLNSC.DoubleNamespaceDecl)xmlns:ns2 = space1;
```

This example of a SET statement creates a namespace declaration with the name ns2 in the namespace xmlns.

```
CREATE LASTCHILD OF OutputRoot IDENTITY (XMLNSC.DoubleNamespaceDecl)xmlns:ns2 VALUE space1;
CREATE LASTCHILD OF OutputRoot TYPE XMLNSC.DoubleNamespaceDecl NAMESPACE 'xmlns' NAME 'ns2' VALUE
```

These examples of a CREATE statement also create a namespace declaration with the name ns2 in the namespace xmlns.

However, be aware that the following example of a CREATE statement creates a namespace declaration with the name xmlns:ns2 in the default namespace:

```
CREATE LASTCHILD OF OutputRoot TYPE XMLNSC.DoubleNamespaceDecl NAME 'xmlns:ns2' VALUE space1;
```

Related concepts:

"Namespaces in the message model" on page 1201
Use namespaces to qualify message model object names.

"XMLNSC: The message body" on page 2549
The XMLNSC parser builds a message tree from the body of an XML document.

"XMLNSC: Element values and mixed content"
The XMLNSC parser is a compact parser; therefore, an element with single content is parsed as a single syntax element. When an element has both child elements and some text, the text is called *mixed content*.

"XMLNSC: Comments and Processing Instructions" on page 2558
The XMLNSC parser discards comments and processing instructions because both comments and processing instructions are auxiliary information with no business meaning.

Related reference:

"XMLNSC: Using field types" on page 1094
The XMLNSC parser sets the field type on every syntax element that it creates.

"XMLNSC: Attributes and elements" on page 2552
The XMLNSC parser uses field types to represent attributes and elements.

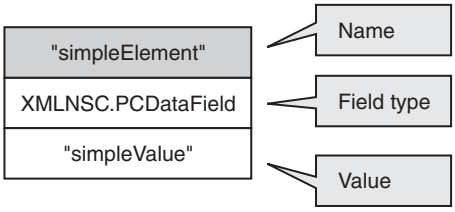
XMLNSC: Element values and mixed content:

The XMLNSC parser is a compact parser; therefore, an element with single content is parsed as a single syntax element. When an element has both child elements and some text, the text is called *mixed content*.

Element with simple content

The following XML fragment with a single content is parsed as a single syntax element:

```
<simpleElement>simpleValue</simpleElement>
```



The value of this element can be queried with this ESQL:

```
SET val = FIELDVALUE(InputRoot.XMLNSC.(XMLNSC.Field)simpleElement);
```

To generate an element with simple content in the output:

```
SET OutputRoot.XMLNSC.(PCDataField)simpleElement VALUE = 'simpleValue';
```

Note that XMLNSC.Field is used when querying the message tree, but XMLNSC.PCDataField is specified when constructing the output message. XMLNSC.PCDataField can be used to query the message tree; however, that would not work if the input message used a CData section, as shown in the following example:

```
<simpleElement><![CDATA[simpleValue]]></simpleElement>
```

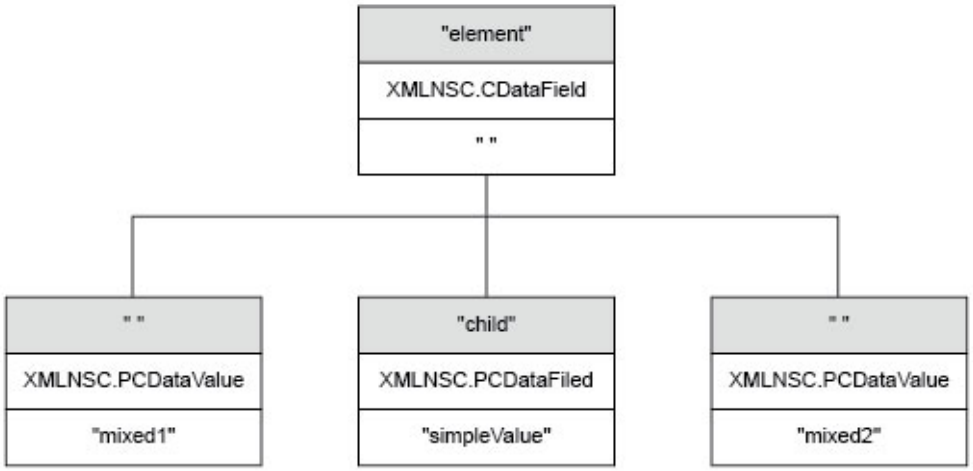
Element with mixed content

If an element has child elements, it is typically a 'folder', and does not have a value. When an element has both child elements and some text, the text is called 'mixed content'.

```
<element>mixed1<child>simpleValue</child>mixed2</element>
```

By default, mixed content is discarded because it is typically just formatting white space and has no business meaning. Mixed content can be preserved if you select the `Retain mixed content` check box on the Parser Options page of the node properties.

If mixed content is being preserved, the XMLNSC parser creates a Value child element for each distinct item of mixed content.



The mixed content can be queried with this ESQL:

```
SET mixed1 = FIELDVALUE(InputRoot.XMLNSC.(element).*[1];
```

The ESQL to construct the above XML fragment is:

```
CREATE ref REFERENCE TO OutputRoot.XMLNSC.element;
CREATE FIRSTCHILD OF ref TYPE XMLNSC.PCDataValue VALUE 'mixed1';
CREATE LASTCHILD OF ref NAME 'child' TYPE XMLNSC.PCDataField  VALUE 'simpleValue';
CREATE LASTSTCHILD OF ref TYPE XMLNSC.PCDataValue VALUE 'mixed2';
```

The following ESQL enables the *Retain mixed content* option:

```
DECLARE X BLOB;
-- assume that X contains an XML document
CREATE LASTCHILD OF OutputRoot
    PARSE(X OPTIONS XMLNSC.MixedContentRetainAll);
```

Element containing a CData section

A CData section is an XML notation that allows XML markup characters to be included in the content of an element.

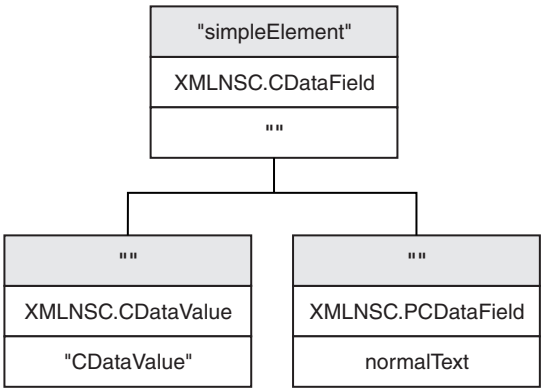
The following two XML fragments are identical in their meaning:

```
<simpleElement>simpleValue</simpleElement>
<simpleElement><![CDATA[simpleValue]]></simpleElement>
```

If the CData section is the only text content, the XMLNSC parser remembers that the input document contained a CData section by setting the field type to XMLNSC.CDataField instead of XMLNSC.PCDataField.

If the CData section is not the only text content, it is created as a child value element, with other child value elements representing the remaining text content. The following is an example of this:

```
<simpleElement><![CDATA[CDataValue]]>normalText</simpleElement>
```



See “XMLNSC: Working with CData” on page 2539 for more information about the correct use of CData in XML documents.

Related concepts:

“XMLNSC: The message body” on page 2549
The XMLNSC parser builds a message tree from the body of an XML document.

“XMLNSC: Comments and Processing Instructions”
The XMLNSC parser discards comments and processing instructions because both comments and processing instructions are auxiliary information with no business meaning.

Related reference:

“XMLNSC: Using field types” on page 1094
The XMLNSC parser sets the field type on every syntax element that it creates.

“XMLNSC: Attributes and elements” on page 2552
The XMLNSC parser uses field types to represent attributes and elements.

“XMLNSC: Namespace declarations” on page 2554
The XMLNSC parser provides full support for namespaces.

XMLNSC: Comments and Processing Instructions:

The XMLNSC parser discards comments and processing instructions because both comments and processing instructions are auxiliary information with no business meaning.

Comments

Comments can be preserved if you select the *Retain comments* check box on the Parser Options page of the node properties.

If *Retain comments* is selected, each comment in the input document is represented by a single syntax element with field type XMLNSC.Comment. The *Retain comments* option can also be accessed by using the following ESQL:

```
DECLARE X BLOB;
-- assume that X contains an XML document
CREATE LASTCHILD OF OutputRoot.XMLNSC
```

```
    PARSE(X DOMAIN XMLNSC
          NAME preserveComments
          OPTIONS XMLNSC.CommentsRetainAll);

-- do it again, this time discarding comments
CREATE LASTCHILD OF OutputRoot.XMLNSC
    PARSE(X DOMAIN XMLNSC
          NAME discardComments
          OPTIONS XMLNSC.CommentsRetainNone);
```

Processing Instructions

Processing instructions can be preserved if you select the *Retain processing instructions* check box on the Parser Options page of the node properties.

If *Retain processing instructions* is selected, each processing instruction the input document is represented by a single syntax element with field type XMLNSC.ProcessingInstruction. The *Retain processing instructions* option can also be accessed by using the following ESQL:

```
DECLARE X BLOB;
-- assume that X contains an XML document
CREATE LASTCHILD OF OutputRoot.XMLNSC
    PARSE(X DOMAIN XMLNSC
          NAME preserveProcessingInstructions
          OPTIONS XMLNSC.ProcessingInstructionsRetainAll);

-- do it again, this time discarding processing instructions
CREATE LASTCHILD OF OutputRoot.XMLNSC
    PARSE(X DOMAIN XMLNSC
          NAME discardProcessingInstructions
          OPTIONS XMLNSC.ProcessingInstructionsRetainNone);
```

Related concepts:

"XMLNSC: The message body" on page 2549
The XMLNSC parser builds a message tree from the body of an XML document.

"XMLNSC: Element values and mixed content" on page 2556
The XMLNSC parser is a compact parser; therefore, an element with single content is parsed as a single syntax element. When an element has both child elements and some text, the text is called *mixed content*.

Related reference:

"XMLNSC: Using field types" on page 1094
The XMLNSC parser sets the field type on every syntax element that it creates.

"XMLNSC: Attributes and elements" on page 2552
The XMLNSC parser uses field types to represent attributes and elements.

"XMLNSC: Namespace declarations" on page 2554
The XMLNSC parser provides full support for namespaces.

XMLNSC: XML Schema support:

Use the XMLNSC parser to parse and validate by using an XML Schema.

For information about how to configure the XMLNSC parser to use an XML Schema, see "XMLNSC parser" on page 1090.

The following topics describe how the XMLNSC parser uses the field type to hold information about XML Schema simple types. This behavior enables the parser to write dates and binary elements in the same form in which they were parsed, and

according to the XML Schema specification. It also allows the message flow developer to write dates, lists, and binary data in the correct XML Schema format.

- "XMLNSC: base64 support"
- "XMLNSC: XML Schema date formatting"
- "XMLNSC: XML List type support" on page 2561

Related concepts:

"XMLNSC parser" on page 1090
The XMLNSC parser is a flexible, general-purpose XML parser that offers high performance XML parsing and optional XML Schema validation.

Related tasks:

"Manipulating messages in the XMLNSC domain" on page 2546
If you are writing ESQL to process messages in the XMLNSC domain, it is helpful to learn about the structure of the message tree that the XMLNSC parser builds.

XMLNSC: base64 support:

The XMLNSC parser can produce binary data in base64-encoded format.

If `Validation` is set to `Content and Value`, and `Build tree using schema types` is enabled, the XMLSNC parser automatically decodes base64 data and creates a BLOB value in the message tree. When producing a message tree, the XMLNSC parser will 'base64-encode' a BLOB if the field type includes the constant *XMLNSC.base64Binary*.

ESQL code example to output base64 data

```
DECLARE Base64Data BLOB '0102030405060708090A0B0C0D0E0F';
-- Add in the base64Binary field type
DECLARE base64FieldType INTEGER XMLNSC.Field + XMLNSC.base64Binary;
CREATE LASTCHILD OF OutputRoot DOMAIN 'XMLNSC' NAME 'XMLNSC';
CREATE LASTCHILD OF OutputRoot.XMLNSC TYPE base64FieldType NAME 'myBinaryData' VALUE Base64Data;
```

Result : <myBinaryData>AQIDBAUGBwgJCgsMDQ4P</myBinaryData>

Note that this example does not depend on validation. The XMLNSC parser can produce base64 binary data even if `Validation` is set to `None`.

Related concepts:

"XMLNSC: XML Schema support" on page 2559
Use the XMLNSC parser to parse and validate by using an XML Schema.

"XMLNSC: XML Schema date formatting"
The XMLNSC parser can parse and write all of the XML Schema simple types.

"XMLNSC: XML List type support" on page 2561
The XMLNSC parser can automatically parse a space-separated list of values into individual syntax elements in the message tree if you select certain options.

XMLNSC: XML Schema date formatting:

The XMLNSC parser can parse and write all of the XML Schema simple types.

The rarely used types gYear, gYearMonth, gMonth, gMonthDay, and gDay do not map directly to a message broker data type. For these simple types, the XMLNSC parser adds one of the following constant values to the field type. This behavior allows the parser to produce the data for output in the same format as it was received.

Field types for controlling date format

The following field types are provided for XML Schema date format support. Do not use these field type constants in path expressions. Use them in conjunction with constants XMLNSC.Attribute and XMLNSC.Field to indicate the required output format for DATE values.

XMLNSC Field Type constant	Purpose	Value
XMLNSC.gYear	Value must be a DATE. If the field type includes this value, the DATE value is written by using the XML Schema gYear format.	0x00000010
XMLNSC.gYearMonth	Value must be a DATE. If the field type includes this value, the DATE value is written by using the XML Schema gYearMonth format.	0x00000040
XMLNSC.gMonth	Value must be a DATE. If the field type includes this value, the DATE value is written by using the XML Schema gMonth format.	0x00000020
XMLNSC.gMonthDay	Value must be a DATE. If the field type includes this value, the DATE value is written by using the XML Schema gMonthDay format.	0x00000050
XMLNSC.gDay	Value must be a DATE. If the field type includes this value, the DATE value is written by using the XML Schema gDay format.	0x00000030

ESQL code example

```
DECLARE gYear DATE '2007-01-01';
-- Add in the gYear field type
DECLARE gYearFieldType INTEGER XMLNSC.Field + XMLNSC.gYear;
CREATE LASTCHILD OF OutputRoot DOMAIN 'XMLNSC' NAME 'XMLNSC';
CREATE LASTCHILD OF OutputRoot.XMLNSC TYPE gYearFieldType NAME 'gYear' VALUE gYear;
```

Result : <gYear>2007</gYear>

XMLNSC: XML List type support:

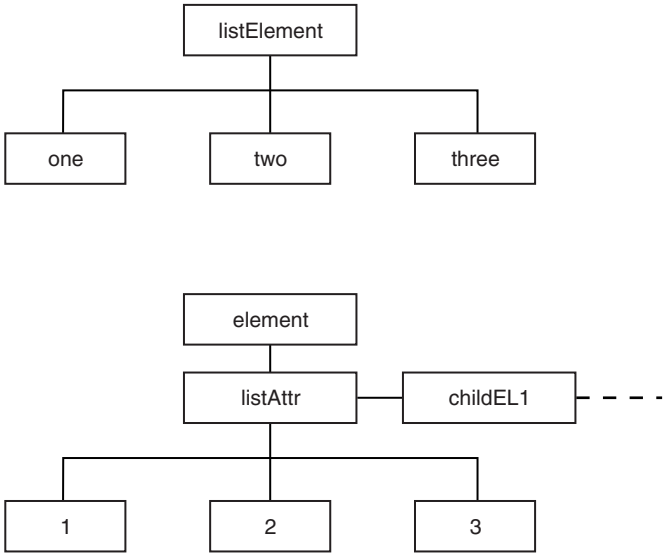
The XMLNSC parser can automatically parse a space-separated list of values into individual syntax elements in the message tree if you select certain options.

An element or an attribute can have multiple values separated by spaces, as shown in the following examples:

```
<listElement>one two three</listElement>
<element listAttribute="1 2 3"><childEL1/></element>
```

If your XML schema specifies a list type for an element or an attribute, and `Validation` is set to `Content and Value`, and `Build tree using schema types` is

enabled, the XMLNSC parser automatically parses the space-separated list of values into individual syntax elements in the message tree. The resulting message tree looks like this:
and for an attribute with a list value it looks like this:



ESQL code examples

Access the individual values in a list

```
SET val = InputRoot.XMLNSC.listElement.*[1];
```

Result : val = 'one'

```
SET val = InputRoot.XMLNSC.element.(XMLNSC.Attribute)listAttr.*[3];
```

Result : val='3'

Create a list element in the message tree

```
CREATE LASTCHILD OF OutputRoot.XMLNSC
   Name 'listElement'
   Type XMLNSC.List;
DECLARE listEl REFERENCE TO OutputRoot.XMLNSC.listElement;
DECLARE listValType INTEGER XMLNSC.PCDataValue;
CREATE LASTCHILD OF listEl TYPE listValType VALUE 'one';
CREATE LASTCHILD OF listEl TYPE listValType VALUE 'two';
CREATE LASTCHILD OF listEl TYPE listValType VALUE 'three';
```

Related concepts:

"XMLNSC: XML Schema support" on page 2559
Use the XMLNSC parser to parse and validate by using an XML Schema.

"XMLNSC: base64 support" on page 2560
The XMLNSC parser can produce binary data in base64-encoded format.

"XMLNSC: XML Schema date formatting" on page 2560
The XMLNSC parser can parse and write all of the XML Schema simple types.

Migrating to XMLNSC:

The XMLNSC parser offers the best combination of features and performance for most applications.

Reasons to migrate

If your message flow uses the XMLNS or XML domain, you might want to migrate a message flow to XMLNSC to take advantage of the XML schema validation. If your message flow uses the MRM domain, you might want to migrate to XMLNSC to obtain standards-compliant validation, and a large reduction in processor usage.

Migrating from the XMLNS or XML domain

The XMLNSC parser differs from the XMLNS parser in the following ways:

- The XMLNSC parser builds a compact message tree.
- It uses different field type constants.
- It discards inline DTDs

In most cases, the compact message tree has no effect on ESQL paths or XPath expressions. Typically, a simple message tree query produces the same results in XMLNSC as in the XMLNS or XML domain. Changing the correlation name from XMLNS to XMLNSC is often sufficient, but care must be taken with the following items:

- Empty elements and null values.

  The XMLNSC parser does not always handle empty elements and null values in the same way as XML and XMLNS.
- Complex XPath expressions that navigate to the value of an element, then to its parent in a single query.

  These expressions might produce different results in the XMLNSC domain.

The field type constants that are used by the XMLNSC parser are different from those constants used by XMLNS or XML. Every occurrence of XML.Attribute, XML.XmlDecl, for example, must be changed to use the equivalent XMLNSC field type constant.

The discarding of inline DTDs only affects message flows that process the DTD.

Migrating from MRM XML

The XMLNSC parser differs from the MRM XML parser in the following ways:

- The XMLNSC parser uses field types to identify the XML constructs in the message tree. The MRM parser distinguishes attributes from elements by matching the message tree against the message definition.
- When writing a message tree, the XMLNSC parser selects namespace prefixes by detecting and using xmlns attributes in the message tree. The MRM XML parser uses a table in the message set properties.
- The MRM parser does not include the root tag of the XML document in the message tree.

Migrating a message flow from MRM to XMLNSC typically requires extensive changes to your message flow. However, the migration usually delivers a large reduction in processor usage, and allows much more accurate control of the output XML.

Manipulating messages in the XMLNS domain:

When you write ESQL for processing messages in the XMLNS domain, it is helpful to understand the structure of the message tree that the XMLNS parser builds.

About this task

The topics in this section provide information about how to write ESQL for processing messages that belong to the XMLNS domain, and that are parsed by the XMLNS parser. Most of the information in these topics also applies to the XML domain, unless it refers to features that are not supported in the XML domain.

Refer to "XMLNS parser" on page 1104 for background information.

The following topics provided detailed information about the structure of the message tree that the XMLNS parser builds, and the field types that it uses.

- "XMLNS: The XML declaration" on page 2565
- "XMLNS: The DTD" on page 2568
- "XMLNS: The XML message body" on page 2569

You can find more information about processing XML messages in "Working with XML messages" on page 2535.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"XML parsers and domains" on page 1084
You can use XML domains to parse and write messages that conform to the W3C XML standard.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Manipulating messages in the XML domain" on page 2581
The XML parser is like the XMLNS parser, but the XML parser has no support for namespaces or opaque parsing.

"Working with XML messages" on page 2535
The following topics provide information about typical tasks for processing XML messages.

Related reference:

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"DECLARE statement" on page 5117
Use the DECLARE statement to define a variable, the data type of the variable and, optionally, its initial value.

"SET statement" on page 5159
The SET statement assigns a value to a variable.

XMLNS: The XML declaration:

The beginning of an XML message can contain an XML declaration.

The following is an example of a declaration:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE s1 PUBLIC "http://www.ibm.com/example.dtd" "example.dtd">
<s1>........</s1>
```

The XML Declaration is represented by the following types of syntax element:

- "XML.XMLDecl"
- "XML.version" on page 2566
- "XML.standalone" on page 2567

"XMLNS: XML declaration example" on page 2568 includes another example of an XML declaration and the tree structure that it forms.

Related concepts:

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"XML parsers and domains" on page 1084
You can use XML domains to parse and write messages that conform to the W3C XML standard.

Related tasks:

"Manipulating messages in the XMLNS domain" on page 2563
When you write ESQL for processing messages in the XMLNS domain, it is helpful to understand the structure of the message tree that the XMLNS parser builds.

"Adding an XML wire format" on page 2853
You can add an XML wire format physical format layer to a message set by using the Message Set editor.

Related reference:

"XMLNS: XML declaration example" on page 2568
The XMLNS parser creates a tree that represents an XML declaration.

XML.XMLDecl:

The XML Declaration is represented in the message tree by a syntax element with field type 'XML.XMLDecl'.

If the XML declaration is created by the XMLNS parser its name is 'XMLDecl'. However, when a message tree is being written, the name is not important; only the field type is used by the parser.

The following shows an example of a declaration:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE s1 PUBLIC "http://www.ibm.com/example.dtd" "example.dtd">
<s1>........</s1>
```

Related concepts:

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML

message for use by the MRM parser.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"XML parsers and domains" on page 1084
You can use XML domains to parse and write messages that conform to the W3C XML standard.

Related tasks:

"Manipulating messages in the XMLNS domain" on page 2563
When you write ESQL for processing messages in the XMLNS domain, it is helpful to understand the structure of the message tree that the XMLNS parser builds.

"Adding an XML wire format" on page 2853
You can add an XML wire format physical format layer to a message set by using the Message Set editor.

XML.version:

The XML version attribute in the XML declaration is represented in the message tree by a syntax element with field type 'XML.version'.

The value of the XML version attribute is the value of the version attribute. It is always a child of an XML.XmlDecl syntaxelement. In the following example, the version element contains the string value 1.0:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE s1 PUBLIC "http://www.ibm.com/example.dtd" "example.dtd">
<s1>........</s1>
```

Related concepts:

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"XML parsers and domains" on page 1084
You can use XML domains to parse and write messages that conform to the W3C XML standard.

Related tasks:

"Manipulating messages in the XMLNS domain" on page 2563
When you write ESQL for processing messages in the XMLNS domain, it is helpful to understand the structure of the message tree that the XMLNS parser builds.

"Adding an XML wire format" on page 2853
You can add an XML wire format physical format layer to a message set by using the Message Set editor.

XML.encoding:

The encoding attribute is represented by a syntax element with field type 'XML.encoding', and is always a child of an XML.XmlDecl syntax element.

In the following example, the encoding attribute has a value of UTF-8.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE s1 PUBLIC "http://www.ibm.com/example.dtd" "example.dtd">
<s1>........</s1>
```

You cannot specify WebSphere MQ encodings in this element.

In your ESQL, (XML,"Encoding") must include quotation marks, because Encoding is a reserved word.

Related concepts:

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"XML parsers and domains" on page 1084
You can use XML domains to parse and write messages that conform to the W3C XML standard.

Related tasks:

"Manipulating messages in the XMLNS domain" on page 2563
When you write ESQL for processing messages in the XMLNS domain, it is helpful to understand the structure of the message tree that the XMLNS parser builds.

"Adding an XML wire format" on page 2853
You can add an XML wire format physical format layer to a message set by using the Message Set editor.

XML.standalone:

The XML standalone element defines the existence of an externally-defined DTD. In the message tree it is represented by a syntax element with field type XML.standalone.

The value of the XML standalone element is the value of the standalone attribute in the XML declaration. It is always a child of an XML.XmlDecl syntax element. The only valid values for the standalone element are yes and no. The following is an example of this:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE s1 PUBLIC "http://www.ibm.com/example.dtd" "example.dtd">
<s1>........</s1>
```

A value of no indicates that this XML document is not standalone, and depends on an externally-defined DTD. A value of yes indicates that the XML document is self-contained. However, because the current release of WebSphere Message Broker does not resolve externally-defined DTDs, the setting of standalone is irrelevant, and is ignored.

Related concepts:

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"XML parsers and domains" on page 1084
You can use XML domains to parse and write messages that conform to the W3C XML standard.

Related tasks:

"Manipulating messages in the XMLNS domain" on page 2563
When you write ESQL for processing messages in the XMLNS domain, it is helpful to understand the structure of the message tree that the XMLNS parser builds.

"Adding an XML wire format" on page 2853
You can add an XML wire format physical format layer to a message set by using the Message Set editor.

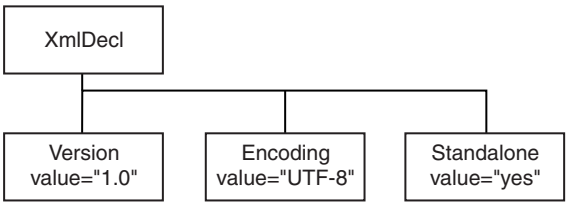
XMLNS: XML declaration example:

The XMLNS parser creates a tree that represents an XML declaration.

The following example shows an XML declaration in an XML document:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

The following figure shows the tree structure that the XMLNS parser creates for this declaration:



Related concepts:

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"XML parsers and domains" on page 1084
You can use XML domains to parse and write messages that conform to the W3C XML standard.

Related tasks:

"Manipulating messages in the XMLNS domain" on page 2563
When you write ESQL for processing messages in the XMLNS domain, it is helpful to understand the structure of the message tree that the XMLNS parser builds.

"Adding an XML wire format" on page 2853
You can add an XML wire format physical format layer to a message set by using the Message Set editor.

XMLNS: The DTD:

The document type declaration (DTD) of an XML message is represented by a syntax element with field type XML.DocTypeDecl, and its children. These comprise the DOCTYPE construct.

Only internal (inline) DTD subsets are represented in the syntax element tree. An inline DTD is a DTD that is declared within the XML document itself. It can be a complete DTD definition, or it can extend the definition in an external DTD.

External DTD subsets (identified by the SystemID or PublicId elements described later in this section) can be referenced in the message, but those referenced are not resolved by the broker.

The following field type constants can be used to reference the various parts of a DTD in the message tree:

- "XML DocTypeDecl" on page 4272
- "XML NotationDecl" on page 4275
- "XML entities" on page 4275
- "XML ElementDef" on page 4281
- "XML AttributeList" on page 4281
- "XML AttributeDef" on page 4282
- "XML DocTypePI" on page 4286
- "XML WhiteSpace and DocTypeWhiteSpace" on page 4287
- "XML DocTypeComment" on page 4286

"XML DTD example" on page 4288 shows an example of an XML DTD.

See "XML document type declaration" on page 4271 for more information about handling an inline DTD.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"XMLNS: The XML declaration" on page 2565
The beginning of an XML message can contain an XML declaration.

"XMLNS: The XML message body"
Every XML message must have a body. The body consists of a hierarchy of XML elements and other XML constructs that represent the message data.

XMLNS: The XML message body:

Every XML message must have a body. The body consists of a hierarchy of XML elements and other XML constructs that represent the message data.

The XMLNS parser assigns a field type to every syntax element that it creates. The value of the field type indicates the XML construct that it represents. In the following topics, each field type is discussed, with a series of example XML fragments.

The following common element types are discussed:

- "XML.element" on page 2573
- "XML.Attribute" on page 2573

"XML message body example" on page 2581 provides an example of an XML message body and the tree structure that is created from it using the syntax elements types that are listed above.

More complex XML messages might use some of the following syntax element types:

- "XML.CDataSection" on page 2575
- "XML.EntityReferenceStart and XML.EntityReferenceEnd" on page 2577
- "XML.comment" on page 2578
- "XML.ProcessingInstruction" on page 2578
- "XML.AsisElementContent" on page 2579
- "XML.BitStream" on page 2580

Related concepts:

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Manipulating messages in the XML domain" on page 2581
The XML parser is like the XMLNS parser, but the XML parser has no support for namespaces or opaque parsing.

"Adding an XML wire format" on page 2853
You can add an XML wire format physical format layer to a message set by using the Message Set editor.

Related reference:

"XMLNS: The XML declaration" on page 2565
The beginning of an XML message can contain an XML declaration.

"XMLNS: The DTD" on page 2568
The document type declaration (DTD) of an XML message is represented by a syntax element with field type XML.DocTypeDecl, and its children. These comprise the DOCTYPE construct.

Accessing attributes and elements:

The XMLNS parser sets the field type on every message tree element that it creates.

The field type indicates the type of XML construct that the element represents. The field types used by the XMLNS parser can be referenced using constants with names prefixed with 'XML.' Field type constants with prefix 'XML.' are for use with the XMLNS and XML parsers only; they do not work with the XMLNSC or MRM parsers.

	XMLNS Field Type constant
Tag	XML.Element
Attribute	`XML.Attribute` `XML.Attr`

By using the field type in this way, the XMLNS parser can distinguish between an element and an attribute that have the same name.

Example XML

```
<parent id="12345">
    <id>ABCDE</id>
</parent>
```

Example ESQL

```
SET value = FIELDVALUE(InputRoot.XMLNS.parent.(XML.Element)id)
Result : value is 'ABCDE'

SET value = FIELDVALUE(InputRoot.XMLNS.parent.(XML.Attr)id)
Result : value is '12345'
```

Example using SELECT to access multiple attributes

In the "Example message" on page 5311, the element Title within each Item element has three attributes: Category, Form, and Edition. For example, the first Title element contains:

```
<Title Category="Computer" Form="Paperback" Edition="2">The XML Companion</Title>
```

The element `InputRoot.XML.Invoice.Purchases.Item[1].Title` has four children in the logical tree: Category, Form, Edition, and the element value, which is "The XML Companion".

If you want to access the attributes for this element, you can code the following ESQL. This extract of code retrieves the attributes from the input message and creates them as elements in the output message. It does not process the value of the element itself in this example.

```
-- Set the cursor to the first XML.Attribute of the Title.
-- Note the * after (XML.Attribute) meaning any name, because the name might not be known
DECLARE cursor REFERENCE TO InputRoot.XMLNS.Invoice.Purchases.Item[1].Title.(XML.Attribute)*;
WHILE LASTMOVE(cursor) DO
-- Create a field with the same name as the XML.Attribute
-- and set its value to the value of the XML.Attribute
  SET OutputRoot.XML.Data.Attributes.{FIELDNAME(cursor)} = FIELDVALUE(cursor);
-- Move to the next sibling of the same TYPE to avoid the Title value
-- which is not an XML.Attribute
  MOVE cursor NEXTSIBLING REPEAT TYPE;
END WHILE;
```

When this ESQL is processed by the Compute node, the following output message is generated:

```
<Data>
  <Attributes>
    <Category>Computer</Category>
    <Form>Paperback</Form>
    <Edition>2</Edition>
  </Attributes>
</Data>
```

You can also use a SELECT statement:

```
SET OutputRoot.XMLNS.Data.Attributes[] =
  (SELECT FIELDVALUE(I.Title) AS title,
          FIELDVALUE(I.Title.(XML.Attribute)Category) AS category,
          FIELDVALUE(I.Title.(XML.Attribute)Form) AS form,
          FIELDVALUE(I.Title.(XML.Attribute)Edition) AS edition
     FROM InputRoot.XML.Invoice.Purchases.Item[] AS I);
```

This statement generates the following output message:

```
<Data>
  <Attributes>
    <title>The XML Companion</title>
    <category>Computer</category>
    <form>Paperback</form>
    <edition>2</edition>
  </Attributes>
  <Attributes>
    <title>A Complete Guide to DB2 Universal Database</title>
    <category>Computer</category>
    <form>Paperback</form>
    <edition>2</edition>
  </Attributes>
  <Attributes>
    <title>JAVA 2 Developers Handbook</title>
    <category>Computer</category>
    <form>Hardcover</form>
    <edition>0</edition>
  </Attributes>
</Data>
```

You can qualify the SELECT with a WHERE statement to narrow down the results to obtain the same output message as the one that is generated by the WHILE statement. This second example shows that you can create the same results with less, and less complex, ESQL.

```
SET OutputRoot.XMLNS.Data.Attributes[] =
  (SELECT FIELDVALUE(I.Title.(XML.Attribute)Category) AS category,
          FIELDVALUE(I.Title.(XML.Attribute)Form) AS form,
          FIELDVALUE(I.Title.(XML.Attribute)Edition) AS edition
     FROM InputRoot.XML.Invoice.Purchases.Item[] AS I)
     WHERE I.Title = 'The XML Companion');
```

This statement generates the following output message:

```
<Data>
  <Attributes>
    <Category>Computer</Category>
    <Form>Paperback</Form>
    <Edition>2</Edition>
  </Attributes>
  </Data>
```

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"XML parsers and domains" on page 1084
You can use XML domains to parse and write messages that conform to the W3C XML standard.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow

nodes and connecting them to form flows.

"Manipulating messages in the XMLNS domain" on page 2563
When you write ESQL for processing messages in the XMLNS domain, it is helpful to understand the structure of the message tree that the XMLNS parser builds.

"Working with XML messages" on page 2535
The following topics provide information about typical tasks for processing XML messages.

Related reference:

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"DECLARE statement" on page 5117
Use the DECLARE statement to define a variable, the data type of the variable and, optionally, its initial value.

"SET statement" on page 5159
The SET statement assigns a value to a variable.

XML.element:

This syntax element represents an XML element (a tag).

The name of the syntax element corresponds to the name of the XML element in the message. This element can have many children in the message tree, including attributes, elements, and content.

XML.tag is supported as an alternative to XML.element for compatibility with earlier versions of WebSphere Message Broker. Use XML.element in any new message flows that you create.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"XML.Attribute"
The XMLNS parser uses this field type for syntax elements that represent an XML attribute.

"XML message body example" on page 2581
The XMLNS parser creates a message tree that represents an XML document.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"XMLNS: The XML message body" on page 2569
Every XML message must have a body. The body consists of a hierarchy of XML elements and other XML constructs that represent the message data.

XML.Attribute:

The XMLNS parser uses this field type for syntax elements that represent an XML attribute.

Parsing

The name and value of the syntax element correspond to the name and value of the XML attribute that is represented. Attribute elements have no children and must always be children of an element.

Writing

When the XMLNS parser generates a bit stream from a message tree, occurrences of ampersand (&), less than (<), greater than (>), double quotation mark ("), and apostrophe ('), within the attribute value, are replaced by the predefined XML entities &, <, >, ", and '.

The XML.attr field type constant is also supported for compatibility with earlier versions of WebSphere Message Broker.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"XML.element" on page 2573
This syntax element represents an XML element (a tag).

"XML message body example" on page 2581
The XMLNS parser creates a message tree that represents an XML document.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"XMLNS: The XML message body" on page 2569
Every XML message must have a body. The body consists of a hierarchy of XML elements and other XML constructs that represent the message data.

XML.content:

The XMLNS parser uses this syntax element to represent character data (including an XML attribute).

The name and value of the syntax element correspond to the name and value of the XML attribute that is represented. Attribute elements have no children and must always be children of an element.

Writing

When the XMLNS parser generates a bit stream from a message tree, occurrences of ampersand (&), less than (<), greater than (>), double quotation mark ("), and apostrophe ('), within the attribute value, are replaced by the predefined XML entities &, <, >, ", and '.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"XML.element" on page 2573
This syntax element represents an XML element (a tag).

“XML.Attribute” on page 2573
The XMLNS parser uses this field type for syntax elements that represent an XML attribute.

“XML message body example” on page 2581
The XMLNS parser creates a message tree that represents an XML document.

Related tasks:

Chapter 9, “Developing message flow applications,” on page 1019
Develop message flows to process your business messages and data.

Related reference:

“XMLNS: The XML message body” on page 2569
Every XML message must have a body. The body consists of a hierarchy of XML elements and other XML constructs that represent the message data.

XML.CDataSection:

CData sections in the XML message are represented by a syntax element with field type XML.CdataSection.

The content of the CDataSection element is the value of the CDataSection element without the `<![CDATA[` that marks its beginning, and the `]]>` that marks its end.

For example, the following CData section:

```
<![CDATA[<greeting>Hello, world!</greeting>]]>
```

is represented by a CDataSection element with a string value of:

```
"<greeting>Hello, world!</greeting>"
```

Unlike Content, occurrences of <,>, &, ", and ' are not translated to their XML character entities (<, >, and &) when the CDataSection is produced.

When to use XML.CDataSection

A CData section is often used to embed one XML message within another. By using a CData section, you ensure that the XML reserved characters (<, >, and &) are not replaced with XML character entities.

XML.AsisElementContent also allows the production of unmodified character data, but XML.CDataSection is typically a better choice because it protects the outer message from errors in the embedded message.

Parsing the contents of a CDataSection

A common requirement is to parse the contents of a CData section to create a message tree, which you can achieve by using the ESQL statement CREATE with the PARSE clause; see “XMLNSC: Working with XML messages and bit streams” on page 2541.

Related concepts:

“Message flows overview” on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

“XML.element” on page 2573
This syntax element represents an XML element (a tag).

"XML message body example" on page 2581
The XMLNS parser creates a message tree that represents an XML document.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"XMLNS: The XML message body" on page 2569
Every XML message must have a body. The body consists of a hierarchy of XML elements and other XML constructs that represent the message data.

XML.NamespaceDecl:

A namespace declaration is represented by a syntax element with field type XML.NamespaceDecl.

Namespace declarations take one of two forms in the message tree:

- **Declaration using a namespace prefix**

  ```
  <ns1:e1 xmlns:ns1="namespace1"/>
  ```

  In the message tree, the syntax element for this namespace declaration is shown in the following table:

Namespace	`http://www.w3.org/2000/xmlns/`
Name	`ns1`
Value	`namespace1`

- **Declaration of a default namespace**

  A default namespace declaration is an xmlns attribute that defines an empty prefix

  ```
  <e1 xmlns="namespace1"/>
  ```

  In the message tree, the syntax element for this namespace declaration is shown in the following table:

Namespace	""
Name	`xmlns`
Value	`namespace1`

  Note that, in both cases, element 'e1' is in namespace 'namespace1'.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"XML.element" on page 2573
This syntax element represents an XML element (a tag).

"XML message body example" on page 2581
The XMLNS parser creates a message tree that represents an XML document.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

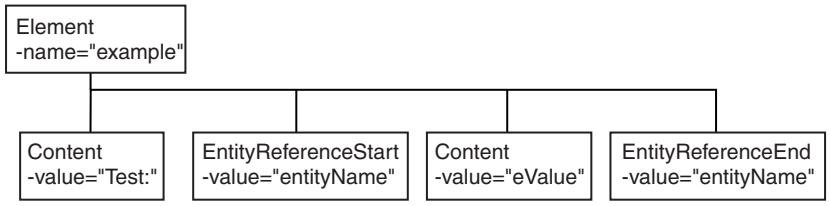
"XMLNS: The XML message body" on page 2569
Every XML message must have a body. The body consists of a hierarchy of XML elements and other XML constructs that represent the message data.

XML.EntityReferenceStart and XML.EntityReferenceEnd:

When an entity reference is encountered in the XML message, both the expanded form and the original entity name are stored in the syntax element tree. The name of the entity is stored as the value of the EntityReferenceStart and EntityReferenceEnd syntax elements, and any syntax elements between contain the entity expansion.

The following examples show the XML entity references in an XML document, and in tree structure form.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE example [ <!ENTITY entityName "eValue"> ]>
<example>Test: &entityName;</example>
```



The XML declaration and the document type declaration are not shown here. Refer to "XMLNS: The XML declaration" on page 2565 and "XMLNS: The DTD" on page 2568 for details of those sections of the syntax element tree.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"XML.element" on page 2573
This syntax element represents an XML element (a tag).

"XML.Attribute" on page 2573
The XMLNS parser uses this field type for syntax elements that represent an XML attribute.

"XML message body example" on page 2581
The XMLNS parser creates a message tree that represents an XML document.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"XMLNS: The XML message body" on page 2569
Every XML message must have a body. The body consists of a hierarchy of XML elements and other XML constructs that represent the message data.

"XMLNS: The XML declaration" on page 2565
The beginning of an XML message can contain an XML declaration.

"XMLNS: The DTD" on page 2568
The document type declaration (DTD) of an XML message is represented by a syntax element with field type XML.DocTypeDecl, and its children. These comprise the DOCTYPE construct.

XML.comment:

An XML.comment that is encountered outside the document type declaration is represented by a syntax element with field type XML.comment. The value of the element is the comment text from the XML message.

If the value of the element contains the character sequence `-->`, the sequence is replaced with the text `-->`. This ensures that the contents of the comment cannot prematurely terminate the comment. Occurrences of the following characters are not translated to their escape sequences:

```
< > & " '
```

The following is an example of the XML comment in an XML document:

```
<example><!-- This is a comment --></example>
```

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"XML message body example" on page 2581
The XMLNS parser creates a message tree that represents an XML document.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"XMLNS: The XML message body" on page 2569
Every XML message must have a body. The body consists of a hierarchy of XML elements and other XML constructs that represent the message data.

"XMLNS: The DTD" on page 2568
The document type declaration (DTD) of an XML message is represented by a syntax element with field type XML.DocTypeDecl, and its children. These comprise the DOCTYPE construct.

XML.ProcessingInstruction:

A processing instruction that is encountered outside the document type declaration is represented by a syntax element with field type XML.ProcessingInstruction.

This is a name-value element; the name of the syntax element is the processing instruction target name, and the value of the syntax element is the character data of the processing instruction. The value of the syntax element must not be empty. The name cannot be `XML` in either uppercase or lowercase.

If the value of the element contains the character sequence `?>`, the sequence is replaced with the text `?>`. This ensures that the content of the processing instruction cannot prematurely terminate the processing instruction. Occurrences of the following characters are not translated to their escape sequences:

```
< > & " '
```

The following shows an example of the XML processing instruction in an XML document:

```
<example><?target This is a PI.?></example>
```

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"XML message body example" on page 2581
The XMLNS parser creates a message tree that represents an XML document.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"XMLNS: The XML message body" on page 2569
Every XML message must have a body. The body consists of a hierarchy of XML elements and other XML constructs that represent the message data.

XML.AsisElementContent:

Use the special field type XML.AsisElementContent to precisely control generated XML.

XML.AsisElementContent is a special field type. Use the field type in a message flow to precisely control the XML that is generated in an output message, without the safeguards of the Element, Attribute, and Content syntax elements. The XMLNS parser never creates elements with this field type.

Try to avoid using AsisElementContent; there is typically a safer alternative approach. If you do use AsisElementContent, it is your responsibility to ensure that the output message is well-formed XML.

You might choose to use AsisElementContent if, for example, you want to suppress the usual behavior in which occurrences of ampersand (&), less than (<), greater than (>), double quotation mark ("), and apostrophe (') are replaced by the predefined XML entities &, <, >, ", and '.

The following example illustrates the use of AsisElementContent. The statement:

```
Set OutputRoot.XMLNS.(XML.Element)Message.(XML.Content) = '<rawMarkup>';
```

generates the following XML in an output message:

```
<Message>&lt;rawMarkup&gt;</Message>
```

However, the statement:

```
Set OutputRoot.XMLNS.(XML.Element)Message.(XML.AsisElementContent) = '<rawMarkup>';
```

generates the following XML in an output message:

```
<Message><rawMarkup></Message>
```

This shows that the value of an AsisElementContent syntax element is not modified before it is written to the output message.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"XML message body example" on page 2581
The XMLNS parser creates a message tree that represents an XML document.

Related tasks:

Chapter 9, “Developing message flow applications,” on page 1019
Develop message flows to process your business messages and data.

Related reference:

“XMLNS: The XML message body” on page 2569
Every XML message must have a body. The body consists of a hierarchy of XML elements and other XML constructs that represent the message data.

XML.BitStream:

BitStream is a specialized element designed to aid the processing of very large messages.

XML.Bitstream is a special field type. When writing an XML message, the value of the BitStream element is written directly into the message, and the name is not important. The BitStream element might be the only element in the message tree.

The value of the element must be of type BLOB; any other data type generates an error when writing the element. Ensure that the content of the element is appropriate for use in the output message; pay special attention to the CCSID and the encoding of the XML text in the BLOB.

Use of the BitStream element is similar to the use of the AsisElementContent element, except that the AsisElementContent type converts its value into a string, whereas the BitStream element uses its BLOB value directly. This is a specialized element designed to aid the processing of very large messages.

The following ESQL excerpts demonstrate some typical use of this element. First, declare the element:

```
DECLARE StatementBitStream BLOB
```

Initialize the contents of StatementBitStream from an appropriate source, such as an input message. If the source field is not of type BLOB, use the CAST statement to convert the contents to BLOB. Then create the new field in the output message; for example:

```
CREATE LASTCHILD OF resultCursor
 Type XML.BitStream
 NAME 'StatementBitStream'
 VALUE StatementBitstream;
```

Related concepts:

“Message flows overview” on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

“XML message body example” on page 2581
The XMLNS parser creates a message tree that represents an XML document.

Related tasks:

Chapter 9, “Developing message flow applications,” on page 1019
Develop message flows to process your business messages and data.

Related reference:

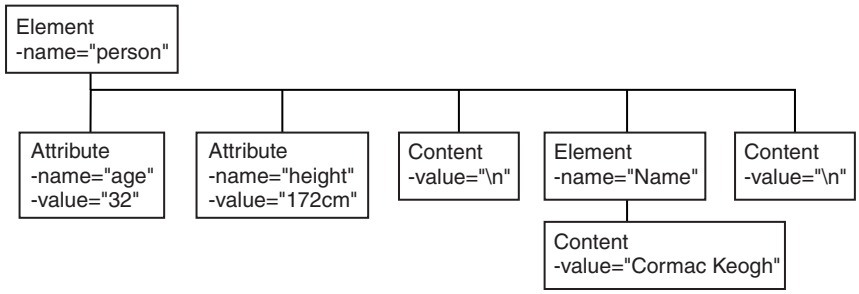
“XMLNS: The XML message body” on page 2569
Every XML message must have a body. The body consists of a hierarchy of XML elements and other XML constructs that represent the message data.

XML message body example:

The XMLNS parser creates a message tree that represents an XML document.

The following example shows the message tree that the XMLNS parser creates for the following snippet from a simple XML document:

```
<Person age="32" height="172cm">
<Name>Cormac Keogh</Name>
</Person>
```



Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"XMLNS: The XML message body" on page 2569
Every XML message must have a body. The body consists of a hierarchy of XML elements and other XML constructs that represent the message data.

Manipulating messages in the XML domain:

The XML parser is like the XMLNS parser, but the XML parser has no support for namespaces or opaque parsing.

About this task

For information about how to work with the XML parser, see "Manipulating messages in the XMLNS domain" on page 2563.

Manipulating messages in the MRM domain:

How to use messages that have been modeled in the MRM domain, and that are parsed by the MRM parser.

About this task

The following topics show you how to deal with messages that have been modeled in the MRM domain, and that are parsed by the MRM parser. The physical formats associated with the message models do not affect this information unless

specifically stated. Use this information in conjunction with the information about manipulating message body content; see “Manipulating message body content” on page 2418.

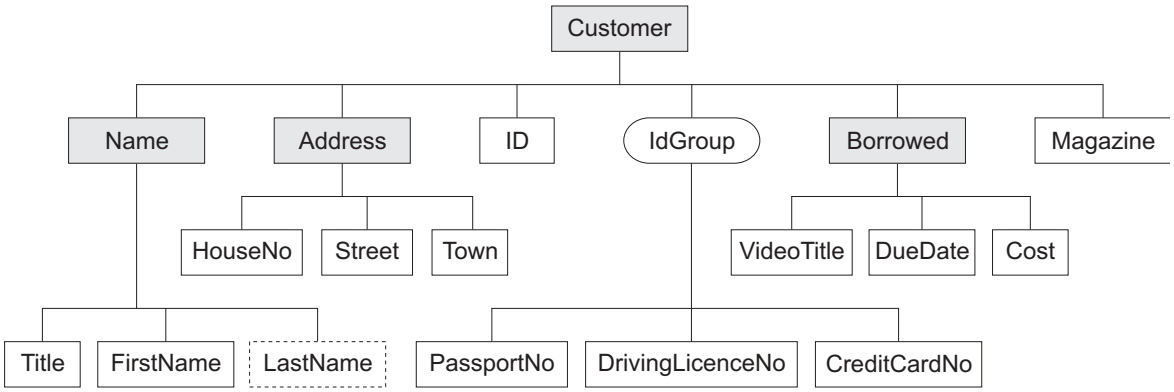
- “Accessing elements in a message in the MRM domain” on page 2584
- “Accessing multiple occurrences of an element in a message in the MRM domain” on page 2585
- “Accessing attributes in a message in the MRM domain” on page 2587
- “Accessing elements within groups in a message in the MRM domain” on page 2589
- “Accessing mixed content in a message in the MRM domain” on page 2592
- “Accessing embedded messages in the MRM domain” on page 2594
- “Accessing the content of a message in the MRM domain with namespace support enabled” on page 2596
- “Querying null values in a message in the MRM domain” on page 2597
- “Setting null values in a message in the MRM domain” on page 2599
- “Working with MRM messages and bit streams” on page 2601
- “Handling large MRM messages” on page 2605

The structure of the "Customer" message is shown in the following sample:

- Video Rental

The message is used in the samples in the topics listed previously to show ESQL that manipulates the objects that can be defined in a message model.

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.



The message includes a variety of structures that demonstrate how you can classify metadata to the MRM. Within an MRM message set, you can define the following objects: messages, types, groups, elements, and attributes. Folder icons that represent each of these types of objects are displayed for each message definition file in the Broker Application Development perspective.

Each message definition file can contribute to a namespace; in this sample, each namespace is completely defined by a single message definition file. You can combine several message definition files to form a complete message dictionary, which you can then deploy to a broker.

The video sample has three message definition files:

Customer.mxsd
Resides in the `no target` namespace

Address.mxsd
Resides in the namespace http://www.ibm.com/AddressDetails

Borrowed.mxsd
Resides in the namespace http://www.ibm.com/BorrowedDetails

Look at the video rental message structure sample for detailed information about the objects that are defined in this message model:

- Video Rental

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Manipulating message body content" on page 2418
The message body is always the last child of root, and is identified by its parser name, for example XML or MRM. You can refer to, modify, and create message body data.

Related reference:

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

Accessing elements in a message in the MRM domain:

You can use ESQL to manipulate the logical tree that represents a message in the message flow. This topic describes how to access data for elements in a message in the MRM domain.

About this task

You can populate an element with data with the SET statement:

```
SET OutputRoot.MRM.Name = UPPER(InputRoot.MRM.Name);
```

The field reference on the left hand side of the expression refers to the element called Name within the MRM message domain. This statement takes the input value for the Name field, converts it to uppercase, and assigns the result to the same element in the output message.

The Name element is defined in the noTarget namespace. No namespace prefix is specified in front of the Name part of the field reference in the example above. If you have defined an MRM element in a namespace other than the noTarget namespace, you must also specify a namespace prefix in the statement. For example:

```
DECLARE brw NAMESPACE 'http://www.ibm.com/Borrowed';

SET OutputRoot.MRM.brw:Borrowed.VideoTitle = 'MRM Greatest Hits';
```

For more information about using namespaces with messages in the MRM domain, see "Accessing the content of a message in the MRM domain with namespace support enabled" on page 2596.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that

are involved in working with message models.

"Accessing the content of a message in the MRM domain with namespace support enabled" on page 2596
Use namespaces where appropriate for messages that are parsed by the MRM parser.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"Filter node" on page 4452
Use the Filter node to route a message according to message content.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

Accessing multiple occurrences of an element in a message in the MRM domain:

You can use specific ESQL code to set the value of one occurrence of an element that has multiple occurrences in a message. You can also use arrow notation to indicate the direction of search when searching for multiple occurrences of an element.

About this task

You can access MRM domain elements following the general guidance given in "Accessing known multiple occurrences of an element" on page 2425 and "Accessing unknown multiple occurrences of an element" on page 2427. Further information specific to MRM domain messages is provided in this topic.

Consider the following statements:

```
DECLARE brw NAMESPACE 'http://www.ibm.com/Borrowed';

SET OutputRoot.MRM.brw:Borrowed[1].VideoTitle = 'MRM Greatest Hits Volume 1';
SET OutputRoot.MRM.brw:Borrowed[2].VideoTitle = 'MRM Greatest Hits Volume 2';
```

The above SET statements operate on two occurrences of the element Borrowed. Each statement sets the value of the child VideoTitle. The array index indicates which occurrence of the repeating element you are interested in.

When you define child elements of a complex type (which has its `Composition` property set to `Sequence`) in a message set, you can add the same element to the complex type more than once. These instances do not have to be contiguous, but you must use the same method (array notation) to refer to them in ESQL.

For example, if you create a complex type with a `Composition` of `Sequence` that contains the following elements:
- `StringElement1`
- `IntegerElement1`
- `StringElement1`

use the following ESQL to set the value of `StringElement1`:

```
SET OutputRoot.MRM.StringElement1[1] =
              'This is the first occurrence of StringElement1';
SET OutputRoot.MRM.StringElement1[2] =
              'This is the second occurrence of StringElement1';
```

You can also use the arrow notation (the greater than (>) and less than (<) symbols) to indicate the direction of search and the index to be specified:

```
SET OutputRoot.MRM.StringElement1[>] =
               'This is the first occurrence of StringElement1';
SET OutputRoot.MRM.StringElement1[<2] =
               'This is the last but one occurrence of
 StringElement1';
SET OutputRoot.MRM.StringElement1[<1] =
               'This is the last occurrence of StringElement1';
```

Refer to "Accessing known multiple occurrences of an element" on page 2425 and "Accessing unknown multiple occurrences of an element" on page 2427 for additional detail.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"Filter node" on page 4452
Use the Filter node to route a message according to message content.
"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.
"SET statement" on page 5159
The SET statement assigns a value to a variable.

Accessing attributes in a message in the MRM domain:

When an MRM message is parsed into a logical tree, attributes and the data that they contain are created as name-value pairs in the same way that MRM elements are. The ESQL that you code to interrogate and update the data held in attributes refers to the attributes in a similar manner.

About this task

Consider the Video Rental sample MRM message. The attribute LastName is defined as a child of the Name element in the Customer message. Here is an example input XML message:

```
<Customer xmlns:addr="http://www.ibm.com/AddressDetails"

xmlns:brw="http://www.ibm.com/BorrowedDetails">
   <Name LastName="Bloggs">
      <Title>Mr</Title>
      <FirstName>Fred</FirstName>
   </Name>
   <addr:Address>
      <HouseNo>13</HouseNo>
      <Street>Oak Street</Street>
      <Town>Southampton</Town>
   </addr:Address>
             <ID>P</ID>
   <PassportNo>J123456TT</PassportNo>
   <brw:Borrowed>
      <VideoTitle>Fast Cars</VideoTitle>
      <DueDate>2003-05-23T01:00:00</DueDate>
      <Cost>3.50</Cost>
   </brw:Borrowed>
   <brw:Borrowed>
      <VideoTitle>Cut To The Chase</VideoTitle>
      <DueDate>2003-05-23T01:00:00</DueDate>
      <Cost>3.00</Cost>
   </brw:Borrowed>
   <Magazine>0</Magazine>
</Customer>
```

When the input message is parsed, values are stored in the logical tree as shown in the following section of user trace:

```
(0x0100001B):MRM = (
  (0x01000013):Name = (
    (0x0300000B):LastName = 'Bloggs'
    (0x0300000B):Title = 'Mr'
    (0x0300000B):FirstName = 'Fred'
  )
  (0x01000013)http://www.ibm.com/AddressDetails:Address = (
    (0x0300000B):HouseNo = 13
    (0x0300000B):Street = 'Oak Street'
    (0x0300000B):Town = 'Southampton'
  )
  (0x0300000B):ID = 'P'
  (0x0300000B):PassportNo = 'J123456TT'
  (0x01000013)http://www.ibm.com/BorrowedDetails:Borrowed = (
    (0x0300000B):VideoTitle = 'Fast Cars'
    (0x0300000B):DueDate = TIMESTAMP '2003-05-23 00:00:00'
    (0x0300000B):Cost = 3.50
  )
  (0x01000013)http://www.ibm.com/BorrowedDetails:Borrowed = (
    (0x0300000B):VideoTitle = 'Cut To The Chase '
    (0x0300000B):DueDate = TIMESTAMP '2003-05-23 00:00:00'
    (0x0300000B):Cost = 3.00
  )
  (0x0300000B):Magazine = FALSE
```

The following ESQL changes the value of the LastName attribute in the output message:

```
SET OutputRoot.MRM.Name.LastName = 'Smith';
```

Be aware of the ordering of attributes when you code ESQL. When attributes are parsed, the logical tree inserts the corresponding name-value before the MRM element's child elements. In the previous example, the child elements Title and FirstName appear in the logical message tree after the attribute LastName. In the Broker Application Development perspective, the Outline view displays attributes after the elements. When you code ESQL to construct output messages, you must define name-value pairs for attributes before any child elements.

The following sample shows the structure of the Customer message:

- Video Rental

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"Filter node" on page 4452
Use the Filter node to route a message according to message content.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

Accessing elements within groups in a message in the MRM domain:

When an input message is parsed, structures that you have defined as groups in your message set are not represented in the logical tree, but its children are. If you want to refer to or update values for elements that are children of a groups, do not include the group in the ESQL statement. Groups do not have tags that appear in instance messages, and do not appear in user trace of the logical message tree.

About this task

Consider the following Video message:

```
<Customer xmlns:addr="http://www.ibm.com/AddressDetails"
xmlns:brw="http://www.ibm.com/BorrowedDetails">
   <Name LastName="Bloggs">
      <Title>Mr</Title>
      <FirstName>Fred</FirstName>
   </Name>
   <addr:Address>
      <HouseNo>13</HouseNo>
      <Street>Oak Street</Street>
      <Town>Southampton</Town>
   </addr:Address>
            <ID>P</ID>
   <PassportNo>J123456TT</PassportNo>
   <brw:Borrowed>
      <VideoTitle>Fast Cars</VideoTitle>
      <DueDate>2003-05-23T01:00:00</DueDate>
      <Cost>3.50</Cost>
   </brw:Borrowed>
   <brw:Borrowed>
      <VideoTitle>Cut To The Chase</VideoTitle>
      <DueDate>2003-05-23T01:00:00</DueDate>
      <Cost>3.00</Cost>
   </brw:Borrowed>
   <Magazine>0</Magazine>
</Customer>
```

When the input message is parsed, values are stored in the logical tree as shown in the following section of user trace:

```
(0x0100001B):MRM = (
  (0x01000013):Name = (
    (0x0300000B):LastName = 'Bloggs'
    (0x0300000B):Title = 'Mr'
    (0x0300000B):FirstName = 'Fred'
  )
  (0x01000013)http://www.ibm.com/AddressDetails:Address = (
    (0x0300000B):HouseNo = 13
    (0x0300000B):Street = 'Oak Street'
    (0x0300000B):Town = 'Southampton'
  )
  (0x0300000B):ID = 'P'
  (0x0300000B):PassportNo = 'J123456TT'
  (0x01000013)http://www.ibm.com/BorrowedDetails:Borrowed = (
    (0x0300000B):VideoTitle = 'Fast Cars'
    (0x0300000B):DueDate = TIMESTAMP '2003-05-23 00:00:00'
    (0x0300000B):Cost = 3.50
  )
  (0x01000013)http://www.ibm.com/BorrowedDetails:Borrowed = (
    (0x0300000B):VideoTitle = 'Cut To The Chase '
    (0x0300000B):DueDate = TIMESTAMP '2003-05-23 00:00:00'
    (0x0300000B):Cost = 3.00
  )
  (0x0300000B):Magazine = FALSE
```

Immediately following the element named ID, the MRM message definition uses a group which has a *Composition* of Choice. The group is defined with three children: PassportNo, DrivingLicenceNo, and CreditCardNo. The choice composition dictates that instance documents must use only one of these three possible alternatives. The example shown above uses the PassportNo element.

When you refer to this element in ESQL statements, you do not specify the group to which the element belongs. For example:

```
SET OutputRoot.MRM.PassportNo = 'J999999TT';
```

If you define messages within message sets that include XML and TDS physical formats, you can determine from the message data which option of a choice has been taken, because the tags in the message represent one of the choice's options. However, if your messages have CWF physical format, or are non-tagged TDS messages, it is not clear from the message data, and the application programs processing the message must determine which option of the choice has been selected. This is known as *unresolved choice handling*. For further information, see the description of the value of `Choice` in "Complex type logical properties" on page 5419.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"Filter node" on page 4452
Use the Filter node to route a message according to message content.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

Accessing mixed content in a message in the MRM domain:

When you define a complex type in a message model, you can optionally specify its content to be mixed. This setting, in support of mixed content in XML Schema, allows you to manipulate data that is included between elements in the message.

About this task

Consider the following example:

```
<MRM>
  <Mess1>
    abc
    <Elem1>def</Elem1>
    ghi
    <Elem2>jkl</Elem2>
    mno
    <Elem3>pqr</Elem3>
  </Mess1>
</MRM>
```

The strings abc, ghi, and mno do not represent the value of a particular element (unlike def, for example, which is the value of element Elem1). The presence of these strings means that you must model Mess1 with mixed content. You can model this XML message in the MRM using the following objects:

Message
The message *Name* property is set to `Mess1` to match the XML tag.

The *Type* property is set to `tMess1`.

Type The complex type *Name* property is set to `tMess1`.

The *Composition* property is set to `OrderedSet`.

The complex type has mixed content.

The complex type contains the following objects:

Element
The *Name* property is set to `Elem1` to match the XML tag.

The *Type* property is set to simple type `xsd:string`.

Element
The *Name* property is set to `Elem2` to match the XML tag.

The *Type* property is set to simple type `xsd:string`.

Element
The *Name* property is set to `Elem3` to match the XML tag.

The *Type* property is set to simple type `xsd:string`.

If you code the following ESQL:

```
SET OutputRoot.MRM.*[1] = InputBody.Elem3;
SET OutputRoot.MRM.Elem1 = InputBody.*[5];
SET OutputRoot.MRM.*[3] = InputBody.Elem2;
SET OutputRoot.MRM.Elem2 = InputBody.*[3];
SET OutputRoot.MRM.*[5] = InputBody.Elem1;
SET OutputRoot.MRM.Elem3 = InputBody*[1];
```

the mixed content is successfully mapped to the following output message:

```
<MRM>
  <Mess1>
    pqr
    <Elem1>mno</Elem1>
    jkl
    <Elem2>ghi</Elem2>
    def
    <Elem3>abc</Elem3>
  </Mess1>
</MRM>
```

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"Filter node" on page 4452
Use the Filter node to route a message according to message content.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"IF statement" on page 5134
The IF statement executes one set of statements based on the result of evaluating condition expressions.

"SET statement" on page 5159
The SET statement assigns a value to a variable.

Accessing embedded messages in the MRM domain:

If you have defined a multipart message, you have at least one message embedded within another. Within the overall complex type that represents the outer messages, you can model the inner message in two ways.

About this task

You can model the inner message in the following ways:

- An element (named E_outer1 in the following example) with its *Type* property set to a complex type that has been defined with its *Composition* property set to `Message`
- A complex type with its *Composition* property set to `Message` (named t_Embedded in the following example)

The ESQL that you need to write to manipulate the inner message varies depending on which of the above models you have used. For example, assume that you have defined:

- An outer message M_outer that has its *Type* property set to t_Outer.
- An inner message M_inner1 that has its *Type* set to t_Inner1
- An inner message M_inner2 that has its *Type* set to t_Inner2
- Type t_Outer that has its first child element named E_outer1 and its second child defined as a complex type named t_Embedded
- Type t_Embedded that has its *Composition* property set to `Message`
- Type t_Inner1 that has its first child element named E_inner11
- Type t_Inner2 that has its first child element named E_inner21
- Type t_outer1 that has its *Composition* property set to `Message`
- Element E_outer1 that has its *Type* property set to t_outer1

If you want to set the value of E_inner11, code the following ESQL:

```
SET OutputRoot.MRM.E_outer1.M_inner1.E_inner11 = 'FRED';
```

If you want to set the value of E_inner21, code the following ESQL:

```
SET OutputRoot.MRM.M_inner2.E_inner21 = 'FRED';
```

If you copy message headers from the input message to the output message, and your input message type contains a path, only the outermost name in the path is copied to the output message type.

When you configure a message flow to handle embedded messages, you can specify the path of a message type in either an MQRFH2 header (if one is present in the input message) or in the input node *Message Type* property in place of a name (for example, for the message modeled above, the path could be specified as M_Outer/M_Inner1/M_Inner2 instead of just M_Outer).

If you have specified that the input message has a physical format of either CWF or XML, any message type prefix is concatenated in front of the message type from the MQRFH2 or input node, giving a final path to use (for more information refer to "Multipart messages" on page 1191). The MRM uses the first item in the path as

the outermost message type, then progressively works inwards when it finds a complex type with its *Composition* property set to `Message`.

If you have specified that the input message has a physical format of TDS, a different process that uses message keys is implemented. This is described in "MRM TDS format: Multipart messages" on page 1241.

For more information about path concatenations, see "Message set properties" on page 5371.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"Multipart messages" on page 1191
A *multipart message* contains one or more other messages within its structure. The contained message is sometimes referred to as an *embedded message*.

"MRM TDS format: Multipart messages" on page 1241
The Tagged/Delimited String Format (TDS) supports both the Message Identity technique and the Message Path technique of identifying embedded messages within a multipart message.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"Filter node" on page 4452
Use the Filter node to route a message according to message content.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.
"SET statement" on page 5159
The SET statement assigns a value to a variable.
"Message set properties" on page 5371
Message sets have properties that you can set to define their characteristics and the way in which they are processed.

Accessing the content of a message in the MRM domain with namespace support enabled:

Use namespaces where appropriate for messages that are parsed by the MRM parser.

About this task

When you want to access elements of a message and namespaces are enabled, you must include the namespace when you code the ESQL reference to the element. If you do not do so, the broker searches the `no target` namespace. If the element is not found in the `no target` namespace, the broker searches all other known namespaces in the message dictionary (that is, within the deployed message set). For performance and integrity reasons, specify namespaces wherever they apply.

The most efficient way to refer to elements when namespaces are enabled is to define a namespace constant, and use this in the appropriate ESQL statements. This technique makes your ESQL code much easier to read and maintain.

Define a constant using the DECLARE NAMESPACE statement:

```
DECLARE ns01 NAMESPACE 'http://www.ns01.com'
.
.
SET OutputRoot.MRM.ns01:Element1 = InputBody.ns01:Element1;
```

`ns01` is interpreted correctly as a namespace because of the way that it is declared.

You can also use a CHARACTER variable to declare a namespace:

```
DECLARE ns02 CHARACTER 'http://www.ns02.com'
.
.
SET OutputRoot.MRM.{ns02}:Element2 = InputBody.{ns02}:Element2;
```

If you use this method, you must surround the declared variable with braces to ensure that it is interpreted as a namespace.

If you are concerned that a CHARACTER variable might get changed, you can use a CONSTANT CHARACTER declaration:

```
DECLARE ns03 CONSTANT CHARACTER 'http://www.ns03.com'
.
.
SET OutputRoot.MRM.{ns03}:Element3 = InputBody.{ns03}:Element3;
```

You can declare a namespace, constant, and variable within a schema, module, or function.

The following sample provides further examples of the use of namespaces:

- Video Rental

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"Namespaces in the message model" on page 1201
Use namespaces to qualify message model object names.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

Related reference:

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"DECLARE statement" on page 5117
Use the DECLARE statement to define a variable, the data type of the variable and, optionally, its initial value.

"SET statement" on page 5159
The SET statement assigns a value to a variable.

Querying null values in a message in the MRM domain:

You can use an ESQL statement to compare an element to NULL.

About this task

If you want to compare an element to NULL, code the statement:

```
IF InputRoot.MRM.Elem2.Child1 IS NULL THEN
   DO:
    -- more ESQL --
END IF;
```

If nulls are permitted for this element, this statement tests whether the element exists in the input message, or whether it exists and contains the MRM-supplied null value. The behavior of this test depends on the physical format:

- For an XML element, if the XML tag or attribute is not in the bit stream, this test returns TRUE.
- For an XML element, if the XML tag or attribute is in the bit stream and contains the MRM null value, this test returns TRUE.
- For an XML element, if the XML tag or attribute is in the bit stream and does not contain the MRM null value, this test returns FALSE.
- For a delimited TDS element, if the element has no value between the previous delimiter and its delimiter, this test returns TRUE.
- For a delimited TDS element, if the element has a value between the previous delimiter and its delimiter that is the same as the MRM-defined null value for this element, this test returns TRUE.
- For a delimited TDS element, if the element has a value between the previous delimiter and its delimiter that is not the MRM-defined null value, this test returns FALSE.
- For a CWF or fixed length TDS element, if the element's value is the same as the MRM-defined null value for this element, this test returns TRUE.
- For a CWF or fixed length TDS element, if the element's value is not the same as the MRM-defined null value, this test returns FALSE.

If you want to determine if the field is missing, rather than present but with null value, you can use the ESQL CARDINALITY function.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"MRM Custom wire format: NULL handling" on page 1216
CWF supports the handling of explicit NULL values within messages, if the logical nillable property of the element is set.

"MRM XML physical format: NULL handling" on page 1249
The purpose of null handling is to specify how messages deal with null values; that is, the absence of a meaningful value for an element.

"MRM TDS format: NULL handling" on page 1240
NULL handling dictates the way in which the MRM parser for TDS messages handles elements that have been set to Null.

Related tasks:

"Setting null values in a message in the MRM domain"
You can use implicit or explicit null processing to set the value of an element to NULL in an output message.

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"Filter node" on page 4452
Use the Filter node to route a message according to message content.

"CAST function" on page 5245

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"CARDINALITY function" on page 5238

"IF statement" on page 5134
The IF statement executes one set of statements based on the result of evaluating condition expressions.

Setting null values in a message in the MRM domain:

You can use implicit or explicit null processing to set the value of an element to NULL in an output message.

About this task

To set a value of an element in an output message, you normally code an ESQL statement similar to the following:

```
SET OutputRoot.MRM.Elem2.Child1 = 'xyz';
```

or its equivalent statement:

```
SET OutputRoot.MRM.Elem2.Child1 VALUE = 'xyz';
```

If you set the element to a non-null value, these two statements give identical results. However, if you want to set the value to null, these two statements do not give the same result:

Procedure

1. If you set the element to NULL using the following statement, the element is deleted from the message tree:

```
SET OutputRoot.MRM.Elem2.Child1 = NULL;
```

   The content of the output bit stream depends on the physical format:

   - For an XML element, neither the XML tag or attribute nor its value are included in the output bit stream.
   - For a Delimited TDS element, neither the tag (if appropriate) nor its value are included in the output bit stream. The absence of the element is typically conveyed by two adjacent delimiters.
   - For a CWF or Fixed Length TDS element, the content of the output bit stream depends on whether you have set the *Default Value* property for the element. If you have set this property, the default value is included in the bit stream. If you have not set the property, an exception is raised.

   This is called implicit null processing.
2. If you set the value of this element to NULL as follows:

```
SET OutputRoot.MRM.Elem2.Child1 VALUE = NULL;
```

   the element is not deleted from the message tree. Instead, a special value of NULL is assigned to the element. The content of the output bit stream depends on the settings of the physical format null-handling properties.

   This is called explicit null processing.

Results

Setting a complex element to NULL deletes that element and all its children.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"MRM Custom wire format: NULL handling" on page 1216
CWF supports the handling of explicit NULL values within messages, if the logical nillable property of the element is set.

"MRM XML physical format: NULL handling" on page 1249
The purpose of null handling is to specify how messages deal with null values; that is, the absence of a meaningful value for an element.

"MRM TDS format: NULL handling" on page 1240
NULL handling dictates the way in which the MRM parser for TDS messages

handles elements that have been set to Null.

Related tasks:

"Querying null values in a message in the MRM domain" on page 2597
You can use an ESQL statement to compare an element to NULL.

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"Filter node" on page 4452
Use the Filter node to route a message according to message content.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"SET statement" on page 5159
The SET statement assigns a value to a variable.

Working with MRM messages and bit streams:

When you use the ASBITSTREAM function or the CREATE FIELD statement with a PARSE clause you must consider various restrictions.

About this task

The ASBITSTREAM function

If you code the ASBITSTREAM function with the parser mode option set to *RootBitStream*, to parse a message tree to a bit stream, the result is an MRM document in the format specified by the message format that is built from the children of the target element in the normal way.

The target element must be a predefined message defined within the message set, or can be a self-defined message if you are using an XML physical format. This algorithm is identical to that used to generate the normal output bit stream. A well-formed bit stream obtained in this way can be used to re-create the original tree by using a CREATE statement with a PARSE clause.

If you code the ASBITSTREAM function with the parser mode option set to *FolderBitStream*, to parse a message tree to a bit stream, the generated bit stream is an MRM element built from the target element and its children. Unlike *RootBitStream* mode the target element does not have to represent a message; it can represent a predefined element within a message or self-defined element within a message.

So that the MRM parser can correctly parse the message, the path from the message to the target element within the message must be specified in the *Message Type*. The format of the path is the same as that used by message paths except that the message type prefix is not used.

For example, suppose the following message structure is used:

```
Message
    elem1
         elem11
         elem12
```

To serialize the subtree representing element `elem12` and its children, specify the message path `'message/elem1/elem12'` in the *Message Type*.

If an element in the path is qualified by a namespace, specify the namespace URI between {} characters in the message path. For example if element `elem1` is qualified by namespace `'http://www.ibm.com/temp'`, specify the message path as `'message/{http://www.ibm.com/temp}elem1/elem12'`

This mode can be used to obtain a bit stream description of arbitrary sub-trees owned by an MRM parser. When in this mode, with a physical format of XML, the XML bit stream generated is not enclosed by the 'Root Tag Name' specified for the Message in the Message Set. No XML declaration is created, even if not suppressed in the message set properties.

Bit streams obtained in this way can be used to re-create the original tree by using a CREATE statement with a PARSE clause (by using a mode of *FolderBitStream*).

The CREATE statement with a PARSE clause

If you code a CREATE statement with a PARSE clause, with the parser mode option set to *RootBitStream*, to parse a bit stream to a message tree, the expected bit stream is a normal MRM document. A field in the tree is created for each field in the document. This algorithm is identical to that used when parsing a bit stream from an input node

If you code a CREATE statement with a PARSE clause, with the parser mode option set to *FolderBitStream*, to parse a bit stream to a message tree, the expected bit stream is a document in the format specified by the Message Format, which is either specified directly or inherited. Unlike *RootBitStream* mode the root of the document does not have to represent an MRM message; it can represent a predefined element within a message or self-defined element within a message.

So that the MRM parser can correctly parse the message the path from the message to the target element within the message must be specified in the *Message Type*. The format of the message path is the same as that used for the ASBITSTREAM function described above.

Example of using the ASBITSTREAM function and CREATE statement with a PARSE clause in FolderBitStream mode

The following ESQL uses the message definition described above. The ESQL serializes part of the input tree by using the ASBITSTREAM function, then uses the CREATE statement with a PARSE clause to re-create the subtree in the output tree. The Input message and corresponding Output message are shown below the ESQL.

```
CREATE COMPUTE MODULE DocSampleFlow_Compute
 CREATE FUNCTION Main() RETURNS BOOLEAN
 BEGIN
  CALL CopyMessageHeaders();

  -- Set the options to be used by ASBITSTREAM and CREATE ... PARSE
  -- to be FolderBitStream and enable validation
  DECLARE parseOptions INTEGER BITOR(FolderBitStream, ValidateContent,
                       ValidateValue, ValidateLocalError);

  -- Serialise the elem12 element and its children from the input bitstream
  -- into a variable
       DECLARE subBitStream BLOB
         ASBITSTREAM(InputRoot.MRM.elem1.elem12
           OPTIONS parseOptions
           SET 'DocSample'
           TYPE 'message/elem1/elem12'
           FORMAT 'XML1');

        -- Set the value of the first element in the output tree
        SET OutputRoot.MRM.elem1.elem11 = 'val11';

        -- Parse the serialized sub-tree into the output tree
        IF subBitStream IS NOT NULL THEN
            CREATE LASTCHILD OF OutputRoot.MRM.elem1
                PARSE ( subBitStream
                        OPTIONS parseOptions
                        SET 'DocSample'
                        TYPE 'message/elem1/elem12'
                        FORMAT 'XML1');
        END IF;

  -- Convert the children of elem12 in the output tree to uppercase
  SET OutputRoot.MRM.elem1.elem12.elem121 =
   UCASE(OutputRoot.MRM.elem1.elem12.elem121);

  SET OutputRoot.MRM.elem1.elem12.elem122 =
   UCASE(OutputRoot.MRM.elem1.elem12.elem122);

        -- Set the value of the last element in the output tree
        SET OutputRoot.MRM.elem1.elem13 = 'val13';

  RETURN TRUE;
 END;

 CREATE PROCEDURE CopyMessageHeaders() BEGIN
  DECLARE I INTEGER 1;
  DECLARE J INTEGER CARDINALITY(InputRoot.*[]);
  WHILE I < J DO
   SET OutputRoot.*[I] = InputRoot.*[I];
   SET I = I + 1;
  END WHILE;
 END;

END MODULE;
```

Input message :

```
<message>
    <elem1>
        <elem11>value11</elem11>
        <elem12>
            <elem121>value121</elem121>
            <elem122>value122</elem122>
        </elem12>
        <elem13>value13</elem13>
    </elem1>
</message>
```

Output message :

```
<message>
    <elem1>
        <elem11>val11</elem11>
        <elem12>
            <elem121>VALUE121</elem121>
            <elem122>VALUE122</elem122>
        </elem12>
        <elem13>val13</elem13>
    </elem1
</message
```

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Accessing embedded messages in the MRM domain" on page 2594
If you have defined a multipart message, you have at least one message embedded within another. Within the overall complex type that represents the outer messages, you can model the inner message in two ways.

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"Filter node" on page 4452
Use the Filter node to route a message according to message content.

"ASBITSTREAM function" on page 5224
The ASBITSTREAM field function generates a bit stream for the subtree of a given field according to the rules of the parser that owns the field.

"CREATE statement" on page 5082
The CREATE statement creates a new message field.

"SET statement" on page 5159
The SET statement assigns a value to a variable.

Handling large MRM messages:

When an input bit stream is parsed, and a logical tree created, the tree representation of an MRM message is typically larger, and in some cases much larger, than the corresponding bit stream.

About this task

The reasons for this large size include:

- The addition of the pointers that link the objects together
- Translation of character data into Unicode that can double the original size
- The inclusion of field names that can be contained implicitly within the bit stream
- The presence of control data that is associated with the broker's operation

Manipulation of a large message tree can, therefore, demand a great deal of storage. If you design a message flow that handles large messages made up of repeating structures, you can code specific ESQL statements that help to reduce the storage load on the broker. These statements support both random and sequential access to the message, but assume that you do not need access to the whole message at one time.

These ESQL statements cause the broker to perform limited parsing of the message, and to keep only that part of the message tree that reflects a single record in storage at a time. If your processing requires you to retain information from record to record (for example, to calculate a total price from a repeating structure of items in an order), you can either declare, initialize, and maintain ESQL variables, or you can save values in another part of the message tree, for example LocalEnvironment.

This technique reduces the memory used by the broker to that needed to hold the full input and output bit streams, plus that required for one record's trees. It provides memory savings when even a small number of repeats is encountered in the message. The broker makes use of partial parsing, and the ability to parse specified parts of the message tree, to and from the corresponding part of the bit stream.

To use these techniques in your Compute node apply these general techniques:

- Copy the body of the input message as a bit stream to a special folder in the output message. This creates a modifiable copy of the input message that is not parsed and which therefore uses a minimum amount of memory.

- Avoid any inspection of the input message; this avoids the need to parse the message.
- Use a loop and a reference variable to step through the message one record at a time. For each record:
  - Use normal transforms to build a corresponding output subtree in a second special folder.
  - Use the ASBITSTREAM function to generate a bit stream for the output subtree that is stored in a *BitStream* element, placed in the position in the tree, that corresponds to its required position in the final bit stream.
  - Use the DELETE statement to delete both the current input and the output record message trees when you complete their manipulation.
  - When you complete the processing of all records, detach the special folders so that they do not appear in the output bit stream.

You can vary these techniques to suit the processing that is required for your messages. The following ESQL provides an example of one implementation.

The ESQL is dependant on a message set called `LargeMessageExanple` that has been created to define messages for both the Invoice input format and the Statement output format. A message called `AllInvoices` has been created that contains a global element called `Invoice` that can repeat one or more times, and a message called `Data` that contains a global element called `Statement` that can repeat one or more times.

The definitions of the elements and attributes have been given the correct data types, therefore, the CAST statements used by the ESQL in the XML example are no longer required. An XML physical format with name `XML1` has been created in the message set which allows an XML message corresponding to these messages to be parsed by the MRM.

When the `Statement` tree is serialized using the ASBITSTREAM function the *Message Set*, *Message Type*, and *Message Format* are specified as parameters. The *Message Type* parameter contains the path from the message to the element being serialized which, in this case, is `Data/Statement` because the `Statement` element is a direct child of the `Data` message.

The input message to the flow is the same Invoice example message used in other parts of the documentation except that it is contained between the tags:

```
<AllInvoices> ....  </AllInvoices>
```

Example

```
CREATE COMPUTE MODULE LargeMessageExampleFlow_Compute
 CREATE FUNCTION Main() RETURNS BOOLEAN
 BEGIN
  CALL CopyMessageHeaders();
    -- Create a special folder in the output message to hold the input tree
    -- Note : SourceMessageTree is the root element of an MRM parser
    CREATE LASTCHILD OF OutputRoot.MRM DOMAIN 'MRM' NAME 'SourceMessageTree';

    -- Copy the input message to a special folder in the output message
    -- Note : This is a root to root copy which will therefore not build trees
    SET OutputRoot.MRM.SourceMessageTree = InputRoot.MRM;

    -- Create a special folder in the output message to hold the output tree
    CREATE FIELD OutputRoot.MRM.TargetMessageTree;

    -- Prepare to loop through the purchased items
```

```
DECLARE sourceCursor REFERENCE TO OutputRoot.MRM.SourceMessageTree.Invoice;
DECLARE targetCursor REFERENCE TO OutputRoot.MRM.TargetMessageTree;
DECLARE resultCursor REFERENCE TO OutputRoot.MRM;
DECLARE grandTotal   FLOAT     0.0e0;

-- Create a block so that it's easy to abandon processing
ProcessInvoice: BEGIN
 -- If there are no Invoices in the input message, there is nothing to do
 IF NOT LASTMOVE(sourceCursor) THEN
    LEAVE ProcessInvoice;
 END IF;

 -- Loop through the invoices in the source tree
 InvoiceLoop : LOOP
    -- Inspect the current invoice and create a matching Statement
    SET targetCursor.Statement =
      THE (
         SELECT
           'Monthly'                        AS Type,
           'Full'                           AS Style,
           I.Customer.FirstName             AS Customer.Name,
           I.Customer.LastName              AS Customer.Surname,
           I.Customer.Title                 AS Customer.Title,
           (SELECT
              FIELDVALUE(II.Title)    AS Title,
              II.UnitPrice * 1.6    AS Cost,
              II.Quantity                AS Qty
           FROM I.Purchases.Item[] AS II
           WHERE II.UnitPrice> 0.0               ) AS Purchases.Article[],
           (SELECT
              SUM( II.UnitPrice *
                       II.Quantity  *
                     1.6                          )
            FROM I.Purchases.Item[] AS II                    ) AS Amount,
            'Dollars'                                  AS Amount.Currency
          FROM sourceCursor AS I
          WHERE I.Customer.LastName <> 'White'
       );

    -- Turn the current Statement into a bit stream
    -- The SET parameter is set to the name of the message set
     -- containing the MRM definition
    -- The TYPE parameter contains the path from the from the message
     -- to element being serialized
    -- The FORMAT parameter contains the name of the physical format
     -- name defined in the message
    DECLARE StatementBitStream BLOB
      ASBITSTREAM(targetCursor.Statement
       OPTIONS FolderBitStream
       SET 'LargeMessageExample'
       TYPE 'Data/Statement'
       FORMAT 'XML1');

    -- If the SELECT produced a result (that is, it was not filtered
     -- out by the WHERE clause), process the Statement
    IF StatementBitStream IS NOT NULL THEN
      -- create a field to hold the bit stream in the result tree
      -- The Type of the element is set to MRM.BitStream to indicate
       -- to the MRM Parser that this is a bitstream
      CREATE LASTCHILD OF resultCursor
         Type  MRM.BitStream
         NAME  'Statement'
         VALUE StatementBitStream;

      -- Add the current Statement's Amount to the grand total
      SET grandTotal = grandTotal + targetCursor.Statement.Amount;
    END IF;
```

```
        -- Delete the real Statement tree leaving only the bit stream version
        DELETE FIELD targetCursor.Statement;

        -- Step onto the next Invoice, removing the previous invoice and any
        -- text elements that might have been interspersed with the Invoices
        REPEAT
          MOVE sourceCursor NEXTSIBLING;
          DELETE PREVIOUSSIBLING OF sourceCursor;
        UNTIL (FIELDNAME(sourceCursor) = 'Invoice')
               OR (LASTMOVE(sourceCursor) = FALSE)
        END REPEAT;

        -- If there are no more invoices to process, abandon the loop
        IF NOT LASTMOVE(sourceCursor) THEN
          LEAVE InvoiceLoop;
        END IF;

     END LOOP InvoiceLoop;
    END ProcessInvoice;

    -- Remove the temporary source and target folders
    DELETE FIELD OutputRoot.MRM.SourceMessageTree;
    DELETE FIELD OutputRoot.MRM.TargetMessageTree;

    -- Finally add the grand total
    SET resultCursor.GrandTotal = grandTotal;

    -- Set the output MessageType property to be 'Data'
    SET OutputRoot.Properties.MessageType = 'Data';

  RETURN TRUE;
 END;

 CREATE PROCEDURE CopyMessageHeaders() BEGIN
  DECLARE I INTEGER 1;
  DECLARE J INTEGER CARDINALITY(InputRoot.*[]);
  WHILE I < J DO
   SET OutputRoot.*[I] = InputRoot.*[I];
   SET I = I + 1;
  END WHILE;
 END;

END MODULE;
```

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"Filter node" on page 4452
Use the Filter node to route a message according to message content.

"ASBITSTREAM function" on page 5224
The ASBITSTREAM field function generates a bit stream for the subtree of a given field according to the rules of the parser that owns the field.

"CREATE statement" on page 5082
The CREATE statement creates a new message field.

"SET statement" on page 5159
The SET statement assigns a value to a variable.

Manipulating messages in the JMS domains:

This topic provides information specific to dealing with messages that belong to the JMSMap and JMSStream domains. These messages are parsed by the generic XML parser.

About this task

Messages that belong to the JMS domains are processed by the XML parser, so you can follow the guidance provided for XML messages in "Manipulating messages in the XML domain" on page 2581, in conjunction with the information in "Manipulating message body content" on page 2418.

The JMSMap and JMSStream domains support MapMessage and StreamMessage messages. Other kinds of JMS message are supported by other domains. For further information about using JMS messages with WebSphere Message Broker, see "WebSphere Broker JMS Transport" on page 1681.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"JMS parsers and domains" on page 1116
The JMSMap and JMSStream domains can be used for modeling messages that are produced by the implementations of the Java Messaging Service standard.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message

flow.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Manipulating messages in the XML domain" on page 2581
The XML parser is like the XMLNS parser, but the XML parser has no support for namespaces or opaque parsing.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"Filter node" on page 4452
Use the Filter node to route a message according to message content.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

Manipulating messages in the IDOC domain:

Use ESQL from a Compute node to copy the incoming IDoc to the outgoing IDoc, and manipulate the message.

About this task

A valid IDoc message flows out of SAP and is sent to the MQSeries link for R/3.

When this IDoc has been committed successfully to the outbound WebSphere MQ queue, the input node of the message flow reads it from that queue and generates the syntax element tree.

The Compute node manipulates this syntax element tree and, when it has finished, passes the output message to subsequent nodes in the message flow. When the message reaches the output node, the IDOC parser is called to rebuild the bit stream from the tree.

The message flow must create an output message in a similar format to the input message.

See "Field names of the IDOC parser structures" on page 6333 for the field names in the DC (Control Structure) and DD (Data Structure) that are recognized by the IDOC parser

Procedure

Use the following ESQL example from a Compute node:

```
SET OutputRoot = InputRoot;
SET OutputRoot.IDOC.DC[1].tabnam = 'EDI_DC40  ';
SET OutputRoot.IDOC.DD[2].sdatatag.MRM.maktx = 'Buzzing all day';
```

Results

The first line of the code copies the incoming IDoc to the outgoing IDoc.

The second line sets the *tabname* of the first DC.

The third line uses the second DD segment, which in this example is of type `E2MAKTM001`, and sets the *maktx* field.

Accessing fields of the IDoc using ESQL:
About this task

Use the ESQL editor Content Assist to complete the SAP-defined fields of the IDoc.

After the `sdatatag` tag in an ESQL statement, the next tag is `MRM`, which you must enter manually, followed by the field name that is to be manipulated. Specify the name of the field within the message segment here, not the name of the message segment.

For example, the following code sets the segment name of the IDoc:

```
SET OutputRoot.IDOC.DD[I].segnam  = 'E2MAKTM001';
```

The following example sets the `msgfn` field within the `E2MAKTM001` segment:

```
SET OutputRoot.IDOC.DD[I].sdatatag.MRM.msgfn = '006';
```

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"IDOC parser and domain" on page 1126
The IDOC domain can be used to process messages that are sent to the broker by SAP R3 clients across the WebSphere MQ link for R3. Such messages are known as SAP ALE IDocs.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"SET statement" on page 5159
The SET statement assigns a value to a variable.

"Field names of the IDOC parser structures" on page 6333
The field names of the Control Structure (DC) and the Data Structure (DD) that are used by the IDOC parser.

Manipulating messages in the MIME domain:

A MIME message does not need to be received over a particular transport. For example, a message can be received over HTTP by using an HTTPInput node, or over WebSphere MQ by using an MQInput node. The MIME parser is used to process a message if the message domain is set to MIME in the input node properties, or if you are using WebSphere MQ, and the MQRFH2 header has a message domain of MIME.

This topic explains how to deal with messages that belong to the MIME domain, and are parsed by the MIME parser. Use this information in conjunction with the information in "Manipulating message body content" on page 2418.

You can manipulate the logical tree using ESQL before passing the message on to other nodes in the message flow. A message flow can also create a MIME domain tree using ESQL. When a MIME domain message reaches an output node, the MIME parser is called to rebuild the bit stream from the logical tree.

The following examples show how to manipulate MIME messages:
- "Creating a new MIME tree"
- "Modifying an existing MIME tree" on page 2613
- "Managing Content-Type" on page 2614

Creating a new MIME tree

A message flow often receives, modifies, and returns a MIME message. In this case, you can work with the valid MIME tree that is created when the input message is parsed. If a message flow receives input from another domain, such as XMLNS, and returns a MIME message, you must create a valid MIME tree. Use the following ESQL example in a Compute node to create the top-level structure for a single-part MIME tree:

```
CREATE FIELD OutputRoot.MIME TYPE Name;
DECLARE M REFERENCE TO OutputRoot.MIME;
CREATE LASTCHILD OF M TYPE Name NAME 'Data';
```

The message flow must also ensure that the MIME Content-Type is set correctly, as explained in "Managing Content-Type" on page 2614. The flow must then add the message data into the MIME tree. The following ESQL examples show how you can do this. In each case, a Data element is created with the domain BLOB.

- A bit stream from another part of the tree is used. This example shows how a bit stream could be created from an XML message that is received by the message flow. The flow then invokes the BLOB parser to store the data under the Data element.

```
DECLARE partData BLOB ASBITSTREAM(InputRoot.XMLNS);
CREATE LASTCHILD OF M.Data DOMAIN('BLOB') PARSE(partData);
```

- Instead of parsing the bit stream, create the new structure, then attach the data to it, as shown in this ESQL example:

```
DECLARE partData BLOB ASBITSTREAM(InputRoot.XMLNS);
CREATE LASTCHILD OF M.Data DOMAIN('BLOB') NAME 'BLOB';
CREATE LASTCHILD OF M.Data.BLOB NAME 'BLOB' VALUE partData;
```

Both of these approaches create the same tree structure. The first approach is better because explicit knowledge of the tree structure that the BLOB parser requires is not built into the flow.

More commonly, the Compute node must build a tree for a multipart MIME document. The following ESQL example shows how you can do this, including setting the top-level Content-Type property.

```
DECLARE part1Data BLOB ASBITSTREAM(InputRoot.XMLNS, InputProperties.Encoding, InputProperties.CodedCharSetId);

SET OutputRoot.Properties.ContentType = 'multipart/related; boundary=myBoundary';

CREATE FIELD OutputRoot.MIME TYPE Name;
DECLARE M REFERENCE TO OutputRoot.MIME;
CREATE LASTCHILD OF M TYPE Name NAME 'Parts';
CREATE LASTCHILD OF M.Parts TYPE Name NAME 'Part';
DECLARE P1 REFERENCE TO M.Parts.Part[1];
CREATE FIELD P1."Content-Type" TYPE NameValue VALUE 'text/plain';
CREATE FIELD P1."Content-Id"   TYPE NameValue VALUE 'part one';
CREATE LASTCHILD OF P1 TYPE Name NAME 'Data';
CREATE LASTCHILD OF P1.Data DOMAIN('BLOB') PARSE(part1Data);

CREATE LASTCHILD OF M.Parts TYPE Name NAME 'Part';
DECLARE P2 REFERENCE TO M.Parts.Part[2];
CREATE FIELD P2."Content-Type" TYPE NameValue VALUE 'text/plain';
CREATE FIELD P2."Content-Id"   TYPE NameValue VALUE 'part two';
CREATE LASTCHILD OF P2 TYPE Name NAME 'Data';
CREATE LASTCHILD OF P2.Data DOMAIN('BLOB') PARSE(part2Data);
```

Modifying an existing MIME tree

This ESQL example adds a new MIME part to an existing multipart MIME message. If the message is not multipart, it is not modified.

```
SET OutputRoot = InputRoot;

-- Check to see if the MIME message is multipart or not.
IF LOWER(InputProperties.ContentType) LIKE 'multipart/%'
THEN
  CREATE LASTCHILD OF OutputRoot.MIME.Parts NAME 'Part';

  DECLARE P REFERENCE TO OutputRoot.MIME.Parts.[<];
  CREATE FIELD P."Content-Type" TYPE NameValue VALUE 'text/xml';
  CREATE FIELD P."Content-ID"   TYPE NameValue VALUE 'new part';
  CREATE LASTCHILD OF P TYPE Name NAME 'Data';

  -- This is an artificial way of creating some BLOB data.
  DECLARE newBlob BLOB '4f6e652074776f2074687265650d0a';
  CREATE LASTCHILD OF P.Data DOMAIN('BLOB') PARSE(newBlob);
END IF;
```

If you receive a MIME message, for example; through an EmailInput node, and you know the format of your message, you might want to reparse the message. For example:

```
CREATE LASTCHILD OF OutputRoot.XMLNSC.emailData  DOMAIN('XMLNSC')
PARSE(InputRoot.MIME.Data.BLOB.BLOB,InputProperties.Encoding,
InputProperties.CodedCharSetId);
```

Managing Content-Type

When you create a new MIME message tree, or when you modify the value of the MIME boundary string, make sure that the MIME Content-Type header is set correctly by setting the ContentType value in the broker Properties subtree. The following example shows how to set the ContentType value for a MIME part with simple content:

```
SET OutputRoot.Properties.ContentType = 'text/plain';
```

Do not set the Content-Type value directly in the MIME tree or HTTP trees because the value is ignored or used inconsistently.

When you receive a MIME message, you can filter or route the message content based on the content-Type. The following example shows an XPath query that can be used in a Route node to filter the content based on whether the message contains an attachment or not:

```
starts-with($Root/MIME/Content-Type,"multipart")
```

Related concepts:

"MIME parser and domain" on page 1117
Use the MIME domain if your messages use the MIME standard for multipart messages.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Manipulating message body content" on page 2418
The message body is always the last child of root, and is identified by its parser name, for example XML or MRM. You can refer to, modify, and create message body data.

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

Related reference:

"MIME standard header fields" on page 6323
Check this quick reference to the common MIME headers.

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

Manipulating messages in the BLOB domain:

How to deal with messages that belong to the BLOB domain, and that are parsed by the BLOB parser.

About this task

You cannot manipulate the contents of a BLOB message, because it has no predefined structure. However, you can refer to its contents using its known position within the bit stream, and process the message with a minimum of knowledge about its contents.

The BLOB message body parser does not create a tree structure in the same way that other message body parsers do. It has a root element BLOB, that has a child element, also called BLOB, that contains the data.

You can refer to message content using substrings if you know the location of a particular piece of information within the BLOB data. For example, the following expression identifies the tenth byte of the message body:

```
InputBody.BLOB.BLOB[10]
```

The following expression references 10 bytes of the message data starting at offset 10:

```
SUBSTRING(InputBody.BLOB.BLOB from 10 for 10)
```

You can use the Mapping node to map to and from a predefined BLOB message, and to map to and from items of BLOB data.

Simple example to write a string in the output message:

About this task

The following simple example allows you to write some character data in your ESQL (for example, if you have read some character fields from a database) out as a BLOB:

```
CALL CopyMessageHeaders();
 -- CALL CopyEntireMessage();
 DECLARE mystring CHARACTER;
 SET mystring='hello';
 SET OutputRoot.BLOB.BLOB=CAST (mystring AS BLOB CCSID 1208);
```

Related concepts:

"BLOB parser and domain" on page 1124
The BLOB message domain includes all the messages with content that cannot be interpreted and subdivided into smaller sections of information.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

"Creating a BLOB output message using a message map" on page 2262
Use the Message Mapping editor to create a bit stream from a message source, and create it as a BLOB output message.

"Storing a BLOB message in a database table using a message map" on page 2297
Use the Message Mapping editor to create a bit stream from a BLOB message, and store it in a database table.

"Mapping from a BLOB message to an output message" on page 2263
Use the Message Mapping editor to parse a BLOB message.

"Mapping from a BLOB field in a database table to an output message" on page 2298
Use the Message Mapping editor to parse a bit stream from a field in a database table into a folder in a target message.

Related reference:

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"CARDINALITY function" on page 5238

"CASE function" on page 5243
CASE is a complex function that has two forms; the simple-when form and the searched-when form. In either form CASE returns a result, the value of which controls the path of subsequent processing.

"CAST function" on page 5245

"DECLARE statement" on page 5117
Use the DECLARE statement to define a variable, the data type of the variable and, optionally, its initial value.

"SET statement" on page 5159
The SET statement assigns a value to a variable.

"SUBSTRING function" on page 5218
SUBSTRING is a string manipulation function that manipulates all string data types (BIT, BLOB, and CHARACTER), and extracts characters from a string to create another string.

"WHILE statement" on page 5167
The WHILE statement evaluates a condition expression, and if it is TRUE executes a sequence of statements.

Manipulating messages in the JSON domain:

You can manipulate messages that belong to the JSON domain, which are parsed by the JSON parser.

Use this information in conjunction with the information in "Manipulating message body content" on page 2418.

The following sample is provided, which demonstrates the use of JSON in an HTTP REST service:

- RESTful Web Service Using JSON

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

The examples in the following topics show how to manipulate JSON messages:

- "Creating a JSON message" on page 2618
- "Modifying a JSON message" on page 2621

Related concepts:

"JSON parser and domain" on page 1128
JSON (JavaScript Object Notation) is a simple data-interchange format based on a subset of the JavaScript programming language.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Manipulating message body content" on page 2418
The message body is always the last child of root, and is identified by its parser name, for example XML or MRM. You can refer to, modify, and create message body data.

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

Creating a JSON message:

You can create JSON message data containing JSON objects, JSON arrays, or both, by creating elements in the logical message tree, under the `Data` element that is owned by the JSON parser root.

A JSON message can have either an anonymous object or an anonymous array as the root of the data. When creating a JSON array in the logical message tree, the JSON array name is placed in a tree element that has its type set to the JSON parser element type `JSON.Array`.

The items in the JSON array are placed in the logical tree as `NameValue` elements, as children of the `JSON.Array` element, with the required value. The names of these array item elements are not used by the JSON serializer, because JSON array items are anonymous. However, for consistency with the name used by the JSON parser, use the name `Item` when you define the array item elements.

Creating a JSON object message

The following example shows how to create a JSON message that is formatted with an object at the root level, with the form:

```
{ --- }
```

This example shows how to create a simple JSON object message with one name-value pair:

```
{"Message":"Hello World"}
```

The following ESQL code can be used to create the message:

```
SET OutputRoot.JSON.Data.Message = 'Hello World';
```

The following Java code can also be used:

```
MbElement outRoot = outMessage.getRootElement();
MbElement outJsonRoot = outRoot.createElementAsLastChild(MbJSON.PARSER_NAME);
MbElement outJsonData = outJsonRoot.createElementAsLastChild(MbElement.TYPE_NAME, MbJSON.DATA_ELE
MbElement outJsonTest = outJsonData.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "Message"
```

The following PHP code can also be used:

```
$output_message->{MB_JSON_PARSER_NAME}->{MB_JSON_DATA_ELEMENT_NAME}->Message = 'Hello World';
```

The broker message tree produced from this example is:

```
Message: ( ['json' : 0xc552990]
  (0x01000000:Object   ):Data    = (
    (0x03000000:NameValue):Message = 'Hello World' (CHARACTER)
 )
```

Creating a JSON array message

The following example shows how to create a message that is formatted with an array at the root level, with the form:

```
[ --- ]
```

This example shows how to create a JSON array message:

```
["valueA","valueB"]
```

The following ESQL code can also be used to create the array:

```
CREATE FIELD OutputRoot.JSON.Data IDENTITY (JSON.Array)Data;
CREATE LASTCHILD OF OutputRoot.JSON.Data TYPE NameValue NAME 'Item' VALUE 'valueA';
CREATE LASTCHILD OF OutputRoot.JSON.Data TYPE NameValue NAME 'Item' VALUE 'valueB';
```

The following Java code can also be used:

```
MbElement outJsonRoot =
          outRoot.createElementAsLastChild("JSON");
MbElement outJsonData =
   outJsonRoot.createElementAsLastChild(MbJson.Array, "Data", null);
   outJsonData.createElementAsLastChild(MbElement.TYPE_NAME_VALUE,
                                        "Item", "valueA");
   outJsonData.createElementAsLastChild(MbElement.TYPE_NAME_VALUE,
                                        "Item", "valueB");
```

The following PHP code can also be used:

```
$output_assembly->JSON->Data[] = array("valueA","valueB");
```

For more information about PHP arrays, see "Using PHP arrays with JSON" on page 2685.

The message is created in the broker logical message tree with the form:

```
Message: ( ['json' : 0xc552990]
  (0x01001000:Array     ):Data    = (
    (0x03000000:NameValue):Item = 'valueA' (CHARACTER)
    (0x03000000:NameValue):Item = 'valueB' (CHARACTER)
 )
```

Creating a JSON object array message

The following example shows how to create a message that is formatted with an array of objects at the root level, with the form:

```
[{--},{--},...]
```

This example shows how to create the JSON object array message:

```
[{"Nam1":"val1","Num1":1},{"Nam2":"val2","Num2":"}]
```

The following ESQL code can also be used to create the array:

```
CREATE FIELD OutputRoot.JSON.Data IDENTITY(JSON.Array)Data;
SET OutputRoot.JSON.Data.Item[1].Nam1 = 'val1';
SET OutputRoot.JSON.Data.Item[1].Nam1 = 1;
SET OutputRoot.JSON.Data.Item[2].Nam2 = 'val2';
SET OutputRoot.JSON.Data.Item[2].Nam2 = 2;
```

The following Java code can also be used:

```
MbElement jsonData = outMessage.getRootElement().createElementAsLastChild(MbJSON.PARSER_NAME);
MbElement jsonRootArray = jsonData.createElementAsLastChild(MbJSON.DATA_ELEMENT_NAME,null);

MbElement jsonFirstArrayItem = jsonRootArray.createElementAsLastChild(MbElement.TYPE_NAME, MbJSON.
```

```
jsonFirstArrayItem.createElementAsFirstChild(MbElement.TYPE_NAME_VALUE, "nam1", "val1");
jsonFirstArrayItem.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "num1", new Integer(1));

MbElement jsonSecondArrayItem = jsonRootArray.createElementAsLastChild(MbElement.TYPE_NAME, MbJSON.AR
jsonSecondArrayItem.createElementAsFirstChild(MbElement.TYPE_NAME_VALUE, "nam2", "val2");
jsonSecondArrayItem.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "num2", new Integer(2));
```

The message is created in the broker logical message tree with the form:

```
Message: ( ['json' : 0xc673900]
  (0x01001000:Array):Data = (
    (0x01000000:Object):Item = (
      (0x03000000:NameValue):nam1 = 'val1' (CHARACTER)
      (0x03000000:NameValue):num1 = 1 (INTEGER)
    )(0x01000000:Object):Item = (
      (0x03000000:NameValue):nam2 = 'val2' (CHARACTER)
      (0x03000000:NameValue):num2 = 2 (INTEGER)
    )
  )
)
```

Related concepts:

"Using PHP arrays with JSON" on page 2685
PHP arrays are associative maps, in which the key can be an integer or a string.

"Modifying a JSON message" on page 2621
You can modify JSON objects and JSON arrays.

"Manipulating messages in the JSON domain" on page 2617
You can manipulate messages that belong to the JSON domain, which are parsed by the JSON parser.

"JSON parser and domain" on page 1128
JSON (JavaScript Object Notation) is a simple data-interchange format based on a subset of the JavaScript programming language.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Manipulating message body content" on page 2418
The message body is always the last child of root, and is identified by its parser name, for example XML or MRM. You can refer to, modify, and create message body data.

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

Modifying a JSON message:

You can modify JSON objects and JSON arrays.

JSON data streams are parsed into the logical message tree and placed under the `Data` element that is owned by the `JSON` parser root. The JSON data objects and arrays can be accessed and modified from each supported language as follows:

- ESQL as `OutputRoot.JSON.Data.`*`path to required object or array`*
- Java as `/JSON/Data/`*`path to required object or array`*
- PHP as `output_assembly->JSON->Data->`*`path to required object or array`*

The following example shows a possible JSON message:

```
{
    "name" : "John Doe",
    "age" : -1,
    "known" : false,
    "address" : { "street" : null,
                  "city" : "unknown" },
    "belongings" : ["this", "that", "the other"]
}
```

The JSON parser parses the input JSON bit stream, to produce the following broker logical message tree:

```
(Message):JSON     = ( ['json' : 0xhhhhhh]
    (0x01000000:Object):Data      = (
      (0x03000000:NameValue): name    = 'John Doe' (CHARACTER)
      (0x03000000:NameValue): age     = -1 (INTEGER)
      (0x03000000:NameValue): known   = FALSE (BOOLEAN)
      (0x01000000:Object     ): address = (
        (0x03000000:NameValue): street = NULL (UNKNOWN)
        (0x03000000:NameValue): city = 'unknown' (CHARACTER)
      )
      (0x01001000:Array     ): belongings = (
        (0x03000000:NameValue): Item = 'this' (CHARACTER)
        (0x03000000:NameValue): Item = 'that' (CHARACTER)
        (0x03000000:NameValue): Item = 'the other' (CHARACTER)
      )
    )
  )
```

This message tree can be modified through ESQL as:

```
SET OutputRoot.JSON.Data.age =  InputRoot.JSON.Data.age + 22;   -- Set age to 21
SET OutputRoot.JSON.Data.belongings.Item[4] = 'an other';
SET OutputRoot.JSON.Data.belongings.Item[5] = 'and another';
```

The message tree can be modified through PHP as:

```
$output_assembly->JSON->Data->address->age = $input_assembly->JSON->Data->address->age + 22; // Se
$output_assembly->JSON->Data->belongings[2] = 'an other';
$output_assembly->JSON->Data->belongings[3] = 'and another';
```

For more information about PHP arrays, including multidimensional arrays, see "Using PHP arrays with JSON" on page 2685.

The message tree can be modified through Java as:

```
MbElement ageEl = message.getRootElement().getLastChild().getFirstElementByPath("/JSON/Data/age");
int age = ((Integer)ageEl.getValue()).intValue();
ageEl.setValue(age + 22);       // Set age to 21
inMessage.getRootElement().getLastChild().getFirstElementByPath("/JSON/Data/belongings/Item[3]").set
```

JSON with a multidimensional array

The following example shows JSON input containing a multidimensional array:

```
{
    "customer" : "Joe",
    "orders" : [ [ "thing1", 1, 10.1 ],
                 [ "thing2", 2, 20.2 ]  ]
}
```

The following broker message tree is produced:

```
(Message):JSON   = ( ['json' : 0xhhhhhh]
    (0x01000000:Object):Data   = (
      (0x03000000:NameValue):customer = 'Joe' (CHARACTER)
      (0x01001000:Array):orders = (
        (0x01001000:Array):Item   = (
          (0x03000000:NameValue):Item = 'thing1' (CHARACTER)
          (0x03000000:NameValue):Item = 1 (INTEGER)
          (0x03000000:NameValue):Item = 1.01E+1 (FLOAT)
        )
        (0x01001000:Array):Item   = (
          (0x03000000:NameValue):Item = 'thing2' (CHARACTER)
          (0x03000000:NameValue):Item = 2 (INTEGER)
          (0x03000000:NameValue):Item = 2.02E+1 (FLOAT)
        )
      )
    )
  )
```

This message tree is accessed through ESQL in the following way (you can use either the name *Item* or an asterisk (*) as a wildcard):

```
InputRoot.JSON.Data.orders.Item[1].Item[1]   -- 'thing1'
InputRoot.JSON.Data.orders.*[2].*[3]         -- 2.02E+1
```

The message tree is accessed through PHP as:

```
 $output_assembly->JSON->Data.orders[0][0]   // "thing1"
 $output_assembly->JSON->Data.orders[1][2]   // 2.02E+1
```

The message tree is accessed through Java in the following way (you can use either the name *Item*, which the JSON parser gives to array items, or an asterisk (*) as a wildcard):

```
inMessage.getRootElement().getFirstElementByPath("/JSON/Data/orders/Item[1]/Item[1]");  // 'thing1'
inMessage.getRootElement().getFirstElementByPath("/JSON/Data/orders/*[2]/*[3]");  // '2.02'
```

Related concepts:

"Manipulating messages in the JSON domain" on page 2617
You can manipulate messages that belong to the JSON domain, which are parsed by the JSON parser.

"Creating a JSON message" on page 2618
You can create JSON message data containing JSON objects, JSON arrays, or both, by creating elements in the logical message tree, under the `Data` element that is owned by the JSON parser root.

"Using PHP arrays with JSON" on page 2685
PHP arrays are associative maps, in which the key can be an integer or a string.

"JSON parser and domain" on page 1128
JSON (JavaScript Object Notation) is a simple data-interchange format based on a subset of the JavaScript programming language.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Manipulating message body content" on page 2418
The message body is always the last child of root, and is identified by its parser name, for example XML or MRM. You can refer to, modify, and create message body data.

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Managing ESQL files" on page 2390
In a message flow project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

Using the CALL statement to call a user-written routine:

The ESQL CALL statement calls routines that have been created and implemented in different ways.

About this task

A routine is a user-defined function or procedure that has been defined by one of the following statements:
- CREATE FUNCTION
- CREATE PROCEDURE

You can use the CALL statement to call a routine that has been implemented in any of the following ways:
- ESQL
- Java
- As a stored procedure in a database
- As a built-in (broker-provided) function

  You can use CALL to call built-in (broker-provided) functions and user-defined SQL functions, but typically you would use their names directly in expressions .

For details of the syntax and parameters of the CALL statement, see "CALL statement" on page 5077. For an example of the use of CALL, see the examples in

“CREATE PROCEDURE statement” on page 5103.

Calling an ESQL routine:
About this task

A routine is called as an ESQL method if the routine's definition specifies a LANGUAGE clause of ESQL or if the routine is a built-in function. An exact one-to-one matching of the data types and directions of each parameter, between the definition and the CALL, is required. An ESQL routine is allowed to return any ESQL data type, excluding List and Row.

Calling a Java routine:
About this task

A routine is called as a Java method if the routine's definition specifies a LANGUAGE clause of JAVA. An exact one-to-one matching of the data types and directions of each parameter, between the definition and the CALL, is required. If the Java method has a void return type, the INTO clause cannot be used because no value exists to return.

A Java routine can return any data type in the “ESQL-to-Java data-type mapping table” on page 5043, excluding List and Row.

Calling a database stored procedure:
About this task

A routine is called as a database stored procedure if the routine's definition has a LANGUAGE clause of DATABASE.

When a call is made to a database stored procedure, the broker searches for a definition (created by a CREATE PROCEDURE statement) that matches the procedure's local name. The broker then uses the following sequence to resolve the name by which the procedure is known in the database and the database schema to which it belongs:

1. If the CALL statement specifies an IN clause, the name of the data source, the database schema, or both, is taken from the IN clause.
2. If the name of the data source is not provided by an IN clause on the CALL statement, it is taken from the DATASOURCE attribute of the node.
3. If the database schema is not provided by an IN clause on the CALL statement, but is specified on the EXTERNAL NAME clause of the CREATE PROCEDURE statement, it is taken from the EXTERNAL NAME clause.
4. If no database schema is specified on the EXTERNAL NAME clause of the CREATE PROCEDURE statement, the database's user name is used as the schema name. If a matching procedure is found, the routine is called.

The chief use of the CALL statement's IN clause is that it allows the data source, the database schema, or both, to be chosen dynamically at run time. (The EXTERNAL SCHEMA clause also allows the database schema which contains the stored procedure to be chosen dynamically, but it is not as flexible as the IN clause and is retained only for compatibility with earlier versions. Its use in new applications is deprecated.)

If the called routine has any DYNAMIC RESULT SETS specified in its definition, the number of expressions in the CALL statement's *ParameterList* must match the number of parameters to the routine, plus the number of DYNAMIC RESULT

SETS. For example, if the routine has three parameters and two DYNAMIC RESULT SETS, the CALL statement must pass five parameters to the called routine. The parameters passed for the two DYNAMIC RESULT SETS must be list parameters; that is, they must be field references qualified with array brackets [ ]; for example, Environment.ResultSet1[].

A database stored procedure is allowed to return any ESQL data type, excluding Interval, List, and Row.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Invoking stored procedures" on page 2503
To invoke a procedure that is stored in a database, use the ESQL CALL statement. The stored procedure must be defined by a CREATE PROCEDURE statement that has a Language clause of DATABASE and an EXTERNAL NAME clause that identifies the name of the procedure in the database and, optionally, the database schema to which it belongs.

Related reference:

"CALL statement" on page 5077
The CALL statement calls (invokes) a routine.

"CREATE PROCEDURE statement" on page 5103
The CREATE PROCEDURE statement defines a callable function or procedure.

"ESQL-to-Java data-type mapping table" on page 5043
Table summarizing the mappings from ESQL to Java.

Accessing broker properties from ESQL:

You can access broker properties, at run time, from the ESQL modules in your message flow nodes.

About this task

You can use broker properties on the right side of regular SET statements. For example:

```
DECLARE mybroker CHARACTER;
SET mybroker = BrokerName;
```

where `BrokerName` is the broker property that contains the name of the broker on which the message flow is running. However, you cannot use broker properties on the left side of SET statements. This restriction exists because, at run time, broker properties are constants: they cannot be modified, therefore their values cannot be changed by SET statements. If a program tries to change the value of a broker property, the error message `Cannot assign to a symbolic constant` is issued.

Broker properties:

- Are grouped by broker, execution group, flow, and node.
- Are case sensitive. Their names always start with an uppercase letter.

- Return NULL if they do not contain a value.

If your ESQL code already contains a variable with the same name as one of the broker properties, your variable takes precedence; that is, your variable masks the broker property. To access the broker property, use the form `SQL.<broker_property_name>`. For example: `SQL.BrokerName`.

"Broker properties that are accessible from ESQL and Java" on page 5302 shows the broker, flow, and node properties that are accessible from ESQL and indicates which properties are also accessible from Java.

Related concepts:

"Broker properties" on page 1144
For each broker, WebSphere Message Broker maintains a set of properties. You can access some of these properties from your ESQL programs. A subset of the properties is also accessible from Java programs. It can be useful, at run time, to have real-time access to details of a specific node, flow, or broker.

Related tasks:

"Accessing broker properties from the JavaCompute node" on page 2658
For each broker, WebSphere Message Broker maintains a set of properties. You can access some of these properties from your Java programs. It can be useful, during the run time of your code, to have real-time access to details of a specific node, flow, or broker.

Related reference:

"Broker properties that are accessible from ESQL and Java" on page 5302
You can access broker, message flow, and node properties from ESQL and Java.

Configuring a message flow at deployment time with user-defined properties:

Use user-defined properties (UDPs) to configure message flows at deployment and run time, without modifying program code. You can give a UDP an initial value when you declare it in your program, or when you use the Message Flow editor to create or modify a message flow.

Before you begin

Before you start:

For an overview of user-defined properties, see "User-defined properties" on page 1147.

For an example of how to code a UDP statement, see "DECLARE statement" on page 5117.

About this task

In ESQL, you can define UDPs at the module or schema level. After a UDP has been defined in the Message Flow editor, you can modify the value before you deploy it.

To configure UDPs, complete the following steps.

Procedure

1. Open the broker archive (BAR) file. The contents of the BAR file are shown in the **Manage** page of the Broker Archive editor. On this page, you can expand a flow to show the individual nodes that it contains.
2. Click the message flow in which you are interested (not the `.cmf` compiled message flow file). The UDPs that are defined in that flow are displayed with their values in the **Properties** view.
3. If the value of the UDP is unsuitable for your current environment or task, change it to an appropriate value. The value of the UDP is set at the flow level, and is the same for all eligible nodes that are contained in the flow. If a subflow includes a UDP that has the same name as a UDP in the main flow, the value of the UDP in the subflow is not changed.
4. Save your BAR file.

Results

Next:

Deploy the message flow by following the instructions in "Deploying a broker archive file" on page 3235.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"User-defined properties" on page 1147
A user-defined property (UDP) is a property that is defined when you construct a message flow by using the Message Flow editor. This property can be used by the ESQL or Java program inside message flow nodes, such as a Compute node.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

"Setting message flow user-defined properties at run time in a CMP application" on page 985
Use the CMP API to query, discover, and set message flow user-defined properties dynamically at run time. You can use the CMP API to set properties with a data type of character.

"Deploying a broker archive file" on page 3235
After you have created and populated a broker archive (BAR) file, deploy the file to an execution group on a broker, so that the file contents can be used in the broker.

Related reference:

"DECLARE statement" on page 5117
Use the DECLARE statement to define a variable, the data type of the variable

and, optionally, its initial value.

"Broker Archive editor" on page 6794
Use the Broker Archive editor to create and manage broker archive (BAR) files.

"Message Flow editor" on page 6810
The Message Flow editor is the default editor provided by the Broker Application Development perspective for defining a graphical representation of a message flow in the WebSphere Message Broker Toolkit, and for setting properties for individual message flow nodes.

"Configurable message flow properties" on page 4020
When you add a message flow to a broker archive (BAR) file in preparation for deploying it to a broker, you can set additional properties that influence its run time operation. These properties are available for review and update when you select the **Manage and Configure** tab for the broker archive file.

Developing Java

When you use the JavaCompute node, you customize it to determine the exact processing that it provides.

About this task

To tailor the behavior of each node, create a Java class file that provides the processing that you want. You manage Java files through the Java perspective.

You can add any valid Java code to a JavaCompute node, making full use of the Java user-defined node API to process an incoming message. You can use the Java editing facilities of the Eclipse platform to develop your Java code. These facilities include:

- Code completion
- Integrated Javadoc documentation
- Automatic compilation

The Java user-defined node API includes some extra methods that simplify tasks that involve message routing and transformation. These tasks include accessing named elements in a message tree, setting their values, and creating elements, without the need to navigate the tree explicitly.

Use the Debug perspective to debug a message flow that contains a JavaCompute node. When control passes to a JavaCompute node during debugging, the perspective opens the Java debugger, and you can step through the Java class code for the node.

This section provides the following information on developing Java:

- "Managing Java Files" on page 2629
- "Writing Java" on page 2638

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

Related tasks:

Chapter 9, “Developing message flow applications,” on page 1019
Develop message flows to process your business messages and data.

“Debugging a message flow” on page 3157
Use the tasks described in this section of the documentation to manage and work with the flow debugger.

Related reference:

“JavaCompute node” on page 4514
Use the JavaCompute node to work with messages by using the Java language.

Related information:

Java user-defined extensions API

Managing Java Files

The Java code that you provide to modify or customize the behavior of a JavaCompute node is stored in a Java project. WebSphere Message Broker uses the Eclipse Java perspective for developing and administering Java files.

About this task

This section contains topics that describe how to manage these files:

- “Creating Java code for a JavaCompute node”
- “Opening an existing Java file” on page 2631
- “Saving a Java file” on page 2632
- “Adding Java code dependencies” on page 2633
- “Deploying JavaCompute node code” on page 2635

Related tasks:

“Accessing broker properties from the JavaCompute node” on page 2658
For each broker, WebSphere Message Broker maintains a set of properties. You can access some of these properties from your Java programs. It can be useful, during the run time of your code, to have real-time access to details of a specific node, flow, or broker.

“Writing Java” on page 2638
When you create a message flow, you include input nodes that receive messages and, optionally, output nodes that send out new or updated messages. If the message processing requires it, you can include other nodes after the input node that are customized in Java to complete the actions that your applications need.

Related reference:

“JavaCompute node” on page 4514
Use the JavaCompute node to work with messages by using the Java language.

Creating Java code for a JavaCompute node:

Use these instructions to associate Java code with your JavaCompute node.

Before you begin

Before you start

To complete this task, you must have already created a JavaCompute node in your message flow.

About this task

To associate Java code with a JavaCompute node, use one of the following methods:

Procedure

- Use the New Java Compute Node Class wizard to create template code. This is a preferred method.
  1. Right-click the node and click **Open Java**.
  2. Navigate the New Java Compute Node Class wizard until you reach the Java Compute Node Class Template page. On the Java Compute Node Class Template page, choose one of the following options:
     - For a filter node template code, select **Filtering message class**.
     - To change an incoming message, select **Modifying message class**.
     - To create a new message, select **Creating message class**.
  3. Click **Finish**.

  You have created template code for your JavaCompute node.
- Associate a JavaCompute node with an existing Java class that the wizard has previously generated; this is the safest way in which you can share the same Java code between multiple nodes. To associate a JavaCompute nodes with an existing Java class, perform the following steps:
  1. Right-click the JavaCompute node and click **Properties**.
  2. Enter the name of the Java class in the **Java Class** field.
  3. Click **OK**.

  You have associated your JavaCompute node with an existing Java class.
- Create a Java project from scratch. Before you add one or more classes to the project, you must perform the following steps:
  1. Open the `.project` file in the text editor, and ensure that the following builders and natures are set:

```
<buildSpec>
  <buildCommand>
   <name>org.eclipse.jdt.core.javabuilder</name>
   <arguments>
   </arguments>
  </buildCommand>
  <buildCommand>
   <name>com.ibm.etools.mft.java.builder.javabuilder</name>
   <arguments>
   </arguments>
  </buildCommand>
  <buildCommand>
   <name>com.ibm.etools.mft.jcn.jcnbuilder</name>
   <arguments>
   </arguments>
  </buildCommand>
  <buildCommand>
   <name>com.ibm.etools.mft.bar.barbuilder</name>
   <arguments>
   </arguments>
  </buildCommand>
 </buildSpec>
 <natures>
  <nature>org.eclipse.jdt.core.javanature</nature>
  <nature>com.ibm.etools.mft.bar.barnature</nature>
  <nature>com.ibm.etools.mft.jcn.jcnnature</nature>
 </natures>
```

2. Add the following plug-ins to the build path of the Java project:
   a. Open the properties of the Java project.
   b. Select **Java Build Path** and open the **Libraries** tab.
   c. Click **Add Variable**.
   d. Select the variable `JCN_HOME` and click **OK**.
   e. Double-click the variable you added to open the **Edit Variable Entry** dialog.
   f. Click **Extension** and select `javacompute.jar`.
   g. Repeat the previous four steps to add the variable `JCN_HOME/jplugin2.jar`.
3. Create the appropriate Java class and ensure that it extends from `com.ibm.broker.javacompute.MbJavaComputeNode`.

You have created your Java project.

What to do next

You can now perform the following tasks:

- "Opening an existing Java file"
- "Saving a Java file" on page 2632
- "Adding Java code dependencies" on page 2633

Related tasks:

"Developing Java" on page 2628
When you use the JavaCompute node, you customize it to determine the exact processing that it provides.

"Writing Java" on page 2638
When you create a message flow, you include input nodes that receive messages and, optionally, output nodes that send out new or updated messages. If the message processing requires it, you can include other nodes after the input node that are customized in Java to complete the actions that your applications need.

"Managing Java Files" on page 2629
The Java code that you provide to modify or customize the behavior of a JavaCompute node is stored in a Java project. WebSphere Message Broker uses the Eclipse Java perspective for developing and administering Java files.

Related reference:

"JavaCompute node" on page 4514
Use the JavaCompute node to work with messages by using the Java language.

Opening an existing Java file:

You can add to and modify Java code that you have created in a Java project.

Before you begin

Before you start

Before you start this task, complete the following tasks:

- Add a "JavaCompute node" on page 4514 to your message flow.
- "Creating Java code for a JavaCompute node" on page 2629

About this task

To open an existing Java file:

Procedure

1. Switch to the Java perspective.
2. In the Package Explorer view, double-click the Java file that you want to open. The file is opened in the editor view.
3. Work with the contents of the file to make your changes.

Results

You can also open a Java file when you have a message flow open in the editor view. Right-click the JavaCompute node, then select **Open Java**.

What to do next

Next:

You can now perform the following tasks:

- "Saving a Java file"
- "Adding Java code dependencies" on page 2633

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Managing Java Files" on page 2629
The Java code that you provide to modify or customize the behavior of a JavaCompute node is stored in a Java project. WebSphere Message Broker uses the Eclipse Java perspective for developing and administering Java files.

"Writing Java" on page 2638
When you create a message flow, you include input nodes that receive messages and, optionally, output nodes that send out new or updated messages. If the message processing requires it, you can include other nodes after the input node that are customized in Java to complete the actions that your applications need.

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"JavaCompute node" on page 4514
Use the JavaCompute node to work with messages by using the Java language.

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Saving a Java file:

When you edit your Java files, save them to preserve the additions and modifications that you have made.

Before you begin

Before you start

To complete this task, you must have completed the following tasks:
- Add a "JavaCompute node" on page 4514 to your message flow.
- "Creating Java code for a JavaCompute node" on page 2629

About this task

To save a Java file:

Procedure

1. Switch to the Java perspective.
2. Create a new Java file or open an existing Java file.
3. Make the changes to the contents of the Java file.
4. When you have finished working, click **File** > **Save** or **File** > **Save All** to save the file and retain all your changes.

What to do next

Next:

You can now perform the following task:
- "Adding Java code dependencies"

Related tasks:

"Managing Java Files" on page 2629
The Java code that you provide to modify or customize the behavior of a JavaCompute node is stored in a Java project. WebSphere Message Broker uses the Eclipse Java perspective for developing and administering Java files.

"Writing Java" on page 2638
When you create a message flow, you include input nodes that receive messages and, optionally, output nodes that send out new or updated messages. If the message processing requires it, you can include other nodes after the input node that are customized in Java to complete the actions that your applications need.

Related reference:

"JavaCompute node" on page 4514
Use the JavaCompute node to work with messages by using the Java language.

Adding Java code dependencies:

When you write Java code for a JavaCompute node, you can include references to other Java projects and JAR files.

Before you begin

Before you start

To complete this task, you must have completed the following tasks:
- Add a "JavaCompute node" on page 4514 to your message flow.
- Create Java code for a JavaCompute node.

About this task

The Java code in a JavaCompute node might contain references to other Java projects in the Eclipse workspace (internal dependencies), or to external JAR files, for example the JavaMail API (external dependencies), or a set of JAXB Java object classes (internal or external). If other JAR files are referenced, you must add the files to the project class path.

Procedure

1. Right-click the project folder for the project on which you are working, and click **Properties**.
2. Click **Java Build Path** in the left pane.
3. Click the **Libraries** tab.
4. Complete one of the following steps:
   - To add an internal dependency, click **Add JARs**, select the JAR file that you want to add, then click **OK**.
   - To add an external dependency, click **Add External JARs**, select the JAR file that you want to add, then click **Open**. Copy the JAR file to the `shared-classes` directory required. For more details of the `shared-classes` directories available and the effects of each, see "Java shared classloader" on page 2637. If you do not copy the JAR file to a valid `shared-classes` directory, `ClassNotFoundException` exceptions are generated at run time.

Results

You have now added a code dependency.

Related tasks:

"Managing Java Files" on page 2629
The Java code that you provide to modify or customize the behavior of a JavaCompute node is stored in a Java project. WebSphere Message Broker uses the Eclipse Java perspective for developing and administering Java files.

"Writing Java" on page 2638
When you create a message flow, you include input nodes that receive messages and, optionally, output nodes that send out new or updated messages. If the message processing requires it, you can include other nodes after the input node that are customized in Java to complete the actions that your applications need.

"Changing the location of the work path" on page 1011
The work path directory is the location where a component stores internal data, such as installation logs, component details, and trace output. The shared-classes directory is also located in the work path directory and is used for deployed Java code. If the work path directory does not have enough capacity, redirect the directory to another file system that has enough capacity.

"Java shared classloader" on page 2637
Loads all the JAR files located within the shared-classes directories. The precedence order of loading is dictated by the directories the JAR files are located in.

Related reference:

"JavaCompute node" on page 4514
Use the JavaCompute node to work with messages by using the Java language.

Deploying JavaCompute node code:

The WebSphere Message Broker Toolkit handles the deploying of JavaCompute node code automatically. When you create a BAR file and add the message flow, the WebSphere Message Broker Toolkit packages the compiled Java code and its dependencies into the BAR file.

Related reference:

"JavaCompute node" on page 4514
Use the JavaCompute node to work with messages by using the Java language.

JavaCompute node classloading:

Details the default Java classloader options and the precedence order of each type.

The JavaCompute node loads and runs a Java class defined as the `Java class` property on the node. Normally, this class is deployed, along with any other required classes, in a Java archive (JAR) file contained in the broker archive (BAR) file that is used to deploy the flow containing the JavaCompute node.

Any Java classes deployed in this way are loaded by an execution group-wide classloader. Whenever a new or changed JAR file is deployed, the execution group-wide classloader is deleted and recreated with all the currently deployed JAR files. At the same time all JavaCompute nodes refresh the Java classes being used within them, as well as recreating all the Java static variables. It is possible to modify this behavior by using the `JavaClassLoader configurable service` property on the node, which allows alternative classloaders to be used. For more details, see "JavaCompute node classloading using a configurable service" on page 2636.

The execution group-wide classloader first searches all the deployed JAR files for a required class. If a required class cannot be found, it defers to the shared classloader. The shared classloader looks in a set of directories on the broker machine and loads any JAR files found. It can be used to install any required JAR files that do not need to be repeatedly deployed, such as client libraries that the JavaCompute nodes need to use. For more details, see "Java shared classloader" on page 2637.

If the required class cannot be found in any of the deployed JAR files, or in the JAR files installed in the shared classes directories, a classloader containing all of the broker supplied classes is checked (for example: this classloader contains the `jplugin2.jar`), followed by the classpath, and then finally the Java virtual machine (JVM) system classloader.

Two key points must be considered when deciding which of the above mechanisms are used to load a class:

- Isolation between different applications (for example: adding classes to the classpath makes them available to every part of WebSphere Message Broker and can cause conflicts). `JavaClassLoader configurable service` can also be used to isolate JavaCompute nodes in the same execution group.
- Delegation from one classloader to another can only occur in one direction. If a class is resolved in the shared classloader, then it cannot directly create classes in the execution group-wide classloader.

Related concepts:

"JavaCompute node classloading using a configurable service" on page 2636
Details alternative configurable Java classloader options and the precedence order

of each type.

Related tasks:

"Creating Java code for a JavaCompute node" on page 2629
Use these instructions to associate Java code with your JavaCompute node.

"Deploying JavaCompute node code" on page 2635
The WebSphere Message Broker Toolkit handles the deploying of JavaCompute node code automatically. When you create a BAR file and add the message flow, the WebSphere Message Broker Toolkit packages the compiled Java code and its dependencies into the BAR file.

"Java shared classloader" on page 2637
Loads all the JAR files located within the shared-classes directories. The precedence order of loading is dictated by the directories the JAR files are located in.

Related reference:

"JavaCompute node" on page 4514
Use the JavaCompute node to work with messages by using the Java language.

Related information:

Java user-defined extensions API

JavaCompute node classloading using a configurable service:

Details alternative configurable Java classloader options and the precedence order of each type.

The JavaCompute node loads and runs a Java class defined as the `Java class` property on the node. Normally, this class is deployed, along with any other required classes, in a Java archive (JAR) file contained in the broker archive (BAR) file that is used to deploy the flow containing the JavaCompute node.

Any Java classes deployed in this way are loaded by an execution group-wide classloader. It is possible to override this behavior by using the `JavaClassLoader configurable service` property on the node.

The classloader defined by the `JavaClassLoader configurable service` has a list of JAR files, defined by the `includedDeployedJars` property, which it will own and use. Whenever these JAR files are deployed, the configurable service classloader receives the JAR files and uses them for resolving classes. The JAR files are no longer available to the execution group classloader, and any node using that classloader will not have access to any classes contained in those JAR files.

The configurable service classloader first searches all the deployed JAR files it has received for a required class. If a required class cannot be found, it defers to the shared classloader. The shared classloader looks in a set of directories on the broker machine and loads any JAR files found. It can be used to install any required JAR files that do not need to be repeatedly deployed, such as client libraries that the JavaCompute nodes need to use. For more details, see "Java shared classloader" on page 2637. This mechanism can be overridden by setting the `sharedJarPath` property on the configurable service in order to use a specified directory to find installed JAR files, rather than the shared classes directories.

If the required class cannot be found in any of the deployed JAR files, or in the JAR files installed in the shared classes directories, a classloader containing all of the broker supplied classes is checked (for example: this classloader contains the `jplugin2.jar`), followed by the classpath, and then finally the Java virtual machine (JVM) system classloader.

For information about how to create a `JavaClassLoader configurable service`, see "**mqsicreateconfigurableservice** command" on page 3849. Properties of the `JavaClassLoader configurable service` are defined in "JavaClassLoader configurable service" on page 3777.

Related tasks:

"Creating Java code for a JavaCompute node" on page 2629
Use these instructions to associate Java code with your JavaCompute node.

"Deploying JavaCompute node code" on page 2635
The WebSphere Message Broker Toolkit handles the deploying of JavaCompute node code automatically. When you create a BAR file and add the message flow, the WebSphere Message Broker Toolkit packages the compiled Java code and its dependencies into the BAR file.

"Java shared classloader"
Loads all the JAR files located within the shared-classes directories. The precedence order of loading is dictated by the directories the JAR files are located in.

Related reference:

"JavaCompute node" on page 4514
Use the JavaCompute node to work with messages by using the Java language.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

Related information:

"JavaClassLoader configurable service" on page 3777

Java user-defined extensions API

Java shared classloader:

Loads all the JAR files located within the shared-classes directories. The precedence order of loading is dictated by the directories the JAR files are located in.

Before you begin

Before you start

Determine the broker *workpath* to use by running the **mqsireportbroker** command as follows:

```
mqsireportbroker <my_broker_name>
```

See the "**mqsireportbroker** command" on page 3919 for an example output from the command. On z/OS the *workpath* is usually referred to as the *component* directory.

About this task

JAR files are loaded in the following precedence order:

Procedure

1. JAR files placed in the execution group `shared-classes` directory allow only a single defined execution group to access them. Files placed in here are loaded first. No other execution groups can use them.

   Add the JAR files to the following directory:

   - For Windows

workpath\config\<*my_broker_name*>\<*my_eg_label*>\shared-classes

- For Linux, UNIX, and z/OS

  workpath/config/<*my_broker_name*>/<*my_eg_label*>/shared-classes

Ensure that the broker name, and any execution groups created, contain only characters that are valid on your file system. You may also need to create the required directory structure.

All files placed into the execution group `shared-classes` directory that have a `.jar` extension, are loaded and made available in the broker Java environment for that execution group. JAR files in this directory take precedence over JAR files in the top level `shared-classes` directory.

Note: The execution group `shared-classes` directory is not automatically removed when the execution group is deleted. Manually delete the directory if no longer required.

2. JAR files placed in the top level `shared-classes` directory are made available to all brokers and all execution groups. Files placed in here are loaded after any files placed in the execution group `shared-classes` directory.

   Add the JAR files to the following directory:

   - For Windows

     workpath\shared-classes

   - For Linux, UNIX and z/OS

     workpath/shared-classes

Related reference:

"JavaCompute node" on page 4514
Use the JavaCompute node to work with messages by using the Java language.

"**mqsireportbroker** command" on page 3919
Use the **mqsireportbroker** command to display broker registry entries.

Writing Java

When you create a message flow, you include input nodes that receive messages and, optionally, output nodes that send out new or updated messages. If the message processing requires it, you can include other nodes after the input node that are customized in Java to complete the actions that your applications need.

About this task

You can customize the processing that some of the built-in nodes provide. In a JavaCompute node, you can provide Java code that controls precisely the behavior of the node. This set of topics discusses how you can use Java to customize the JavaCompute node.

You can use a JavaCompute node to check and manipulate message content. The node can read the contents of the input message, then construct new output messages that are created from all, part, or none of the input message.

Use the Debug perspective to debug a message flow that contains a JavaCompute node. When control passes to a JavaCompute node during debugging, the perspective opens the Java debugger, allowing you to step through the Java class code for the node.

This section provides more information about writing Java:

- Manipulating message body data

- Manipulating other parts of the message tree
- Accessing broker properties
- Accessing user-defined properties
- "Adding keywords to JAR files" on page 2660
- Interacting with databases
- "Calling an Enterprise Java Bean" on page 2666
- Handling exceptions
- Logging errors

Related concepts:

"Debug perspective" on page 6789

Related tasks:

"Managing Java Files" on page 2629
The Java code that you provide to modify or customize the behavior of a JavaCompute node is stored in a Java project. WebSphere Message Broker uses the Eclipse Java perspective for developing and administering Java files.

"Accessing transport headers" on page 2453

"Debugging a message flow" on page 3157
Use the tasks described in this section of the documentation to manage and work with the flow debugger.

Related reference:

"JavaCompute node" on page 4514
Use the JavaCompute node to work with messages by using the Java language.

Related information:

Java user-defined extensions API

Manipulating message body data by using a JavaCompute node:

You can manipulate message body data by using a JavaCompute node.

About this task

The message body is always the last child of root, and its parser name identifies it, for example XML or MRM.

The following topics describe how to refer to, modify, and create message body data. The information provided here is domain independent:

- "Accessing elements in a message tree from a JavaCompute node" on page 2640
- "Transforming a message by using a JavaCompute node" on page 2642
- "Creating a simple filter by using a JavaCompute node" on page 2647
- "Propagating a message to the JavaCompute node Out and Alternate terminals" on page 2648
- "Extracting information from a message by using XPath 1.0 and a JavaCompute node" on page 2648

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined

structure that is known and agreed by the sender and the receiver.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

Related reference:

"JavaCompute node" on page 4514
Use the JavaCompute node to work with messages by using the Java language.

Accessing elements in a message tree from a JavaCompute node:

Access the contents of a message, for reading or writing, using the structure and arrangement of the elements in the tree that the parser creates from the input bit stream.

About this task

Follow the relevant parent and child relationships from the top of the tree downwards until you reach the required element.

The message tree is passed to a JavaCompute node as an argument of the evaluate method. The argument is an MbMessageAssembly object. MbMessageAssembly contains four message objects:

- Message
- Local Environment
- Global Environment
- Exception List

These objects are read-only, except for Global Environment. If you try to write to the read-only objects, an MbReadOnlyException is issued.

This topic contains the following information about accessing elements in a message tree:

- "Traversing the element tree"
- "Accessing information about an element" on page 2642

Traversing the element tree:
About this task

The following table shows the Java methods that you can use to access element trees, and the equivalent ESQL field type constant for each point in the tree.

Java accessor from MbMessageAssembly	ESQL field type constant
getMessage().getRootElement()	InputRoot
getMessage().getRootElement().getLastChild()	InputBody
getLocalEnvironment().getRootElement()	InputLocalEnvironment
getGlobalEnvironment().getRootElement()	Environment
getExceptionList().getRootElement()	InputExceptionList

Use the following methods to traverse a message tree from an element of type MbElement:

getParent()
returns the parent of the current element

getPreviousSibling()
returns the previous sibling of the current element

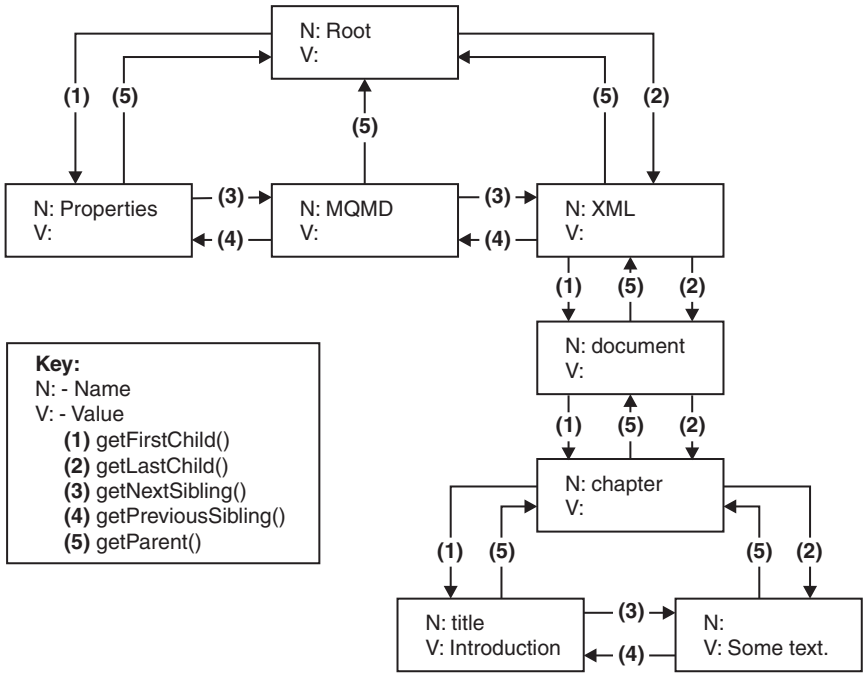
getNextSibling()
returns the next sibling of the current element

getFirstChild()
returns the first child of the current element

getLastChild()
returns the last child of the current element

The following example shows a simple XML message and the logical tree that would be created from the message. The message has been sent using WebSphere MQ. The logical tree diagram also shows the methods to call in order to navigate around the tree.

```
<document>
  <chapter title='Introduction'>
    Some text
  </chapter>
</document>
```



The tree used in this diagram is the one that is created by parsing the previous XML example.

- From the Root part of the tree, calling **getFirstChild()** navigates to Properties. Also from Root, calling **getLastChild()** returns XML.
- From Properties, calling **getParent()** returns Root, and calling **getNextSibling()** returns MQMD.

- From MQMD, calling **getPreviousSibling()** returns Properties, calling **getParent()** returns Root, and calling **getNextSibling()** returns XML.
- From XML, calling **getPreviousSibling()** returns MQMD, calling **getParent()** returns Root, calling **getFirstChild()** returns document, and calling **getLastChild()** also returns document.
- From document, calling **getParent()** returns XML, calling **getFirstChild()** returns chapter, and calling **getLastChild()** also returns chapter.
- From chapter, calling **getParent()** returns document, calling **getFirstChild()** returns title, and calling **getLastChild()** returns the child that contains the message data "Some text.".

The following Java code accesses the chapter element in the logical tree for an XML message that does not contain white spaces. The XML parser retains white space in the parsed tree, but the XMLNS and XMLNSC parsers do not.

```
MbElement root = assembly.getMessage().getRootElement();
MbElement chapter = root.getLastChild().getFirstChild().getFirstChild();
```

Accessing information about an element:
About this task

Use the following methods to return information about the referenced element:

getName()
returns the element name as a java.lang.String

getValue()
returns the element value

getType()
returns the generic type, which is one of the following types:
- NAME: an element of this type has a name, but no value.
- VALUE: an element of this type has a value, but no name.
- NAME/VALUE: an element of this type has both a value and a name.

getSpecificType()
returns the parser-specific type of the element

getNamespace()
returns the namespace URI of this element

Related tasks:

"Writing Java" on page 2638
When you create a message flow, you include input nodes that receive messages and, optionally, output nodes that send out new or updated messages. If the message processing requires it, you can include other nodes after the input node that are customized in Java to complete the actions that your applications need.

Related reference:

"JavaCompute node" on page 4514
Use the JavaCompute node to work with messages by using the Java language.

Related information:

Java user-defined extensions API

Transforming a message by using a JavaCompute node:

You can transform a message by using the JavaCompute node.

The following topics describe how to transform messages by using the JavaCompute node:

- "Creating a new message by using a JavaCompute node"
- "Copying a message by using a JavaCompute node" on page 2644
- "Setting, copying, and moving message elements by using a JavaCompute node" on page 2644
- "Creating new elements by using a JavaCompute node" on page 2646

Creating a new message by using a JavaCompute node:

Many message transformation scenarios require a new outgoing message to be built. The *Create Message Class* template in the JavaCompute node wizard generates template code for this.

About this task

In the template code, the default constructor of MbMessage is called to create a blank message, as shown in the following Java code:

```
MbMessage outMessage = new MbMessage();
```

The headers can be copied from the incoming message by using the supplied utility method, copyMessageHeaders(), as shown in this Java code:

```
copyMessageHeaders(inMessage, outMessage);
```

The new message body can now be created. First, add the top level parser element. For XML, this is:

```
MbElement outRoot = outMessage.getRootElement();
MbElement outBody = outRoot.createElementAsLastChild(MbXMLNSC.PARSER_NAME);
```

The remainder of the message can then be built up by using the **createElement** methods and the extended syntax of the broker XPath implementation.

When you want to create a BLOB message, that is handled as a single byte string by using the BLOB parser domain. The message data is added as a byte array to the single element named "BLOB" under the parser level element, described as follows:

```
String myMsg = "The Message Data";
MbElement outRoot = outMessage.getRootElement();
// Create the Broker Blob Parser element
MbElement outParser = outRoot.createElementAsLastChild(MbBLOB.PARSER_NAME);
// Create the BLOB element in the Blob parser domain with the required text
MbElement outBody = outParser.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "BLOB", myMsg.getBytes());
```

You can use the following code to create a BLOB message in a JavaCompute node:

```
MbElement outMsgRootEl = outMessage.getRootElement();

String parserName = MbBLOB.PARSER_NAME;
String messageType = "";
String messageSet = "";
String messageFormat = "";
int encoding = 0;
int ccsid = 0;
int options = 0;
outMsgRootEl.createElementAsLastChildFromBitstream(responseBodyXmlData,
  parserName, messageType, messageSet, messageFormat, encoding, ccsid,
  options);
```

To add a JSON message in a JavaCompute node, see “Creating a JSON message” on page 2618.

Related tasks:

“Accessing elements in a message tree from a JavaCompute node” on page 2640
Access the contents of a message, for reading or writing, using the structure and arrangement of the elements in the tree that the parser creates from the input bit stream.

Related reference:

“JavaCompute node” on page 4514
Use the JavaCompute node to work with messages by using the Java language.

Copying a message by using a JavaCompute node:

You can copy a message by using a JavaCompute node.

About this task

The incoming message and message assembly are read-only. To modify a message, a copy of the incoming message must be made. The *Modifying Message Class* template in the JavaCompute node wizard generates this copy. The following copy constructors are called:

```
MbMessage outMessage = new MbMessage(inAssembly.getMessage());
MbMessageAssembly outAssembly = new MbMessageAssembly(inAssembly, outMessage);
```

The new outAssembly object is propagated to the next node.

Related tasks:

“Accessing elements in a message tree from a JavaCompute node” on page 2640
Access the contents of a message, for reading or writing, using the structure and arrangement of the elements in the tree that the parser creates from the input bit stream.

Related reference:

“JavaCompute node” on page 4514
Use the JavaCompute node to work with messages by using the Java language.

Setting, copying, and moving message elements by using a JavaCompute node:

Transform elements in the message as it passes through a JavaCompute node in the message flow.

Before you begin

Before you start:

The incoming message and message assembly are read-only. Therefore, to modify a message, you must make a copy of it first. For more information, see “Copying a message by using a JavaCompute node.”

About this task

- “Setting information about an element” on page 2645
- “Moving and copying elements” on page 2645

The Java API reference information provides details about each of the methods used in the following sections:

Setting information about an element:
About this task

Use these methods to set information about the referenced element:

setName()
Sets the name of the element

setValue()
Sets the value of the element

setSpecificType()
Sets the parser-specific type of the element

setNamespace()
Sets the namespace URI of the element

Moving and copying elements:
About this task

Use a JavaCompute node to copy or detach an element from a message tree by using the following methods:

`detach()`
The element is detached from its parent and siblings, but any child elements are left attached

`copy()` A copy of the element and its attached children is created

Use one of four methods to attach an element or subtree that you have copied on to another tree:

`addAsFirstChild(`*element*`)`
Adds an unattached element as the first child of *element*

`addAsLastChild(`*element*`)`
Adds an unattached element as the last child of *element*

`addBefore(`*element*`)`
Adds an unattached element as the previous sibling of *element*

`addAfter(`*element*`)`
Adds an unattached element as the next sibling of *element*

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Writing Java" on page 2638
When you create a message flow, you include input nodes that receive messages and, optionally, output nodes that send out new or updated messages. If the message processing requires it, you can include other nodes after the input node that are customized in Java to complete the actions that your applications need.

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

Related reference:

"JavaCompute node" on page 4514
Use the JavaCompute node to work with messages by using the Java language.

Related information:

Java user-defined extensions API

Creating new elements by using a JavaCompute node:

You can use a JavaCompute node to create new elements.

About this task

Use the following methods in a JavaCompute node to create new elements in a message tree:

- createElementAsFirstChild()
- createElementAsLastChild()
- createElementBefore()
- createElementAfter()

The method returns a reference to the newly-created element. Each method has three overloaded forms:

createElement...(int type)
Creates a blank element of the specified type. Valid generic types are:

- MbElement.TYPE_NAME. This type of element has only a name, for example an XML element.
- MbElement.TYPE_VALUE. This type of element has only a value, for example XML text that is not contained in an XML element.
- MbElement.TYPE_NAME_VALUE. This type of element has both a name and a value, for example an XML attribute.

Specific type values can also be assigned. The meaning of this type information is dependent on the parser. Element name and value information must be assigned by using the setName() and setValue() methods.

createElement...(int type, String name, Object value)
Method for setting the name and value of the element at creation time.

createElement...(String parserName)
A special form of createElement...() that is only used to create top-level parser elements.

This example Java code adds a new chapter element to the XML example given in "Accessing elements in a message tree from a JavaCompute node" on page 2640:

```
MbElement root = outMessage.getRootElement();
MbElement document = root.getLastChild().getFirstChild();
MbElement chapter2 = document.createElementAsLastChild(MbElement.TYPE_NAME,"Chapter",null);

// add title attribute
MbElement title2 = chapter2.createElementAsFirstChild(MbElement.TYPE_NAME_VALUE,
 "title", "Message Flows");
```

This produces the following XML output:

```
<document>
 <chapter title="Introduction">
  Some text.
 </chapter>
 <chapter title="Message Flows"/>
</document>
```

Related tasks:

"Accessing elements in a message tree from a JavaCompute node" on page 2640
Access the contents of a message, for reading or writing, using the structure and arrangement of the elements in the tree that the parser creates from the input bit stream.

Related reference:

"JavaCompute node" on page 4514
Use the JavaCompute node to work with messages by using the Java language.

Creating a simple filter by using a JavaCompute node:

The JavaCompute node has two output terminals: Out and Alternate. To use the JavaCompute node as a filter node, propagate a message to either the Out or Alternate terminal based on the message content.

Before you begin

Before you start

To complete this task, you must have added a "JavaCompute node" on page 4514 to your message flow.

About this task

Use the JavaCompute node creation wizard to generate template code for a filter node:

Procedure

Select the *Filtering Message Class* template in the JavaCompute node creation wizard to create a filter node.
The following template code is produced. It passes the input message to the Out terminal without doing any processing on the message.

```
public class jcn2 extends MbJavaComputeNode {

  public void evaluate(MbMessageAssembly assembly) throws MbException {
    MbOutputTerminal out = getOutputTerminal("out");
    MbOutputTerminal alt = getOutputTerminal("alternate");

    MbMessage message = assembly.getMessage();

    // ----------------------------------------------------------
    // Add user code below

    // End of user code
    // ----------------------------------------------------------

    // The following should only be changed
    // if not propagating message to the 'out' terminal

    out.propagate(assembly);
  }
}
```

The template produces a partial implementation of a method called evaluate(). The broker calls evaluate() once for each message that passes through the node. The parameter that is passed to evaluate() is the message assembly. The message assembly encapsulates the message that is passed on from the previous node in the message flow.
Add custom code to the template, and propagate messages to both the Out and Alternate terminals to create a message filter.

Related tasks:

"Propagating a message to the JavaCompute node Out and Alternate terminals"
The JavaCompute node has two output terminals, Out and Alternate. Therefore, you can use the node both as a filter node and as a message transformation node. After you have processed the message, propagate the message to an output terminal by using a propagate() method.

Related reference:

"JavaCompute node" on page 4514
Use the JavaCompute node to work with messages by using the Java language.

Propagating a message to the JavaCompute node Out and Alternate terminals:

The JavaCompute node has two output terminals, Out and Alternate. Therefore, you can use the node both as a filter node and as a message transformation node. After you have processed the message, propagate the message to an output terminal by using a propagate() method.

About this task

To propagate the message assembly to the Out terminal use the following method:

```
out.propagate(assembly);
```

To propagate the message assembly to the Alternate terminal, use the following method:

```
alt.propagate(assembly);
```

To propagate the same MbMessage object multiple times, call the finalizeMessage() method on the MBMessage object, so that any changes made to the message are reflected in the bit stream that is generated downstream of the JavaCompute node; for example:

```
MbMessage outMessage = new MbMessage(inMessage);
MbMessageAssembly outAssembly = new MbMessageAssembly(inAssembly, outMessage);
...
newMsg.finalizeMessage(MbMessage.FINALIZE_NONE);
out.propagate(outAssembly);
...
newMsg.finalizeMessage(MbMessage.FINALIZE_NONE);
out.propagate(outAssembly);
```

Related reference:

"JavaCompute node" on page 4514
Use the JavaCompute node to work with messages by using the Java language.

Extracting information from a message by using XPath 1.0 and a JavaCompute node:

XPath is a query language designed for use with XML documents, but you can use it with any tree structure to query contents.

WebSphere Message Broker uses XPath to select elements from the logical message tree regardless of the format of the bit stream. The terminology used in this topic is based on the terminology used in the W3C definition of XPath 1.0. For more information about XPath, see "Using XPath" on page 1506; and for more information about the W3C definition of the XPath 1.0 standard, see W3C XPath 1.0 Specification. For examples of XPath use, see the MbXPath topic in the "Java user-defined node API" on page 5312 documentation.

This topic contains the following information:

Using the evaluateXPath method to extract message information

The evaluateXPath() method is included in the Java user-defined node API. It supports XPath 1.0, with the following exceptions:

- Namespace axis and namespace node type. The namespace axis returns the actual XML namespace declaration nodes for a particular element. You can therefore manipulate XML prefix or URI declarations within an XPath expression. This axis returns an empty node set for bit streams that are not XML.
- If you use the **id()** function, it throws an MbRecoverableException.

The evaluateXPath() method can be called on a MbMessage object (for absolute paths), or on a MbElement object (for relative paths). The XPath expression is passed to the method as a string parameter. A second form of this method is provided that takes an MbXPath object. This object encapsulates an XPath expression along with variable bindings and namespace mappings, if these are required.

The evaluateXPath() method returns an object of one of these four types, depending on the expression return type:

- java.lang.Boolean, representing the XPath Boolean type
- java.lang.Double, representing the XPath number type
- java.lang.String, representing the XPath string type
- java.util.List, representing the XPath node set. The List interface represents an ordered sequence of objects, in this case MbElements. It allows direct access to the elements, or the ability to get an Iterator or an MbElement array.

XPath variable binding

XPath 1.0 supports the ability to refer to variables that have been assigned before the expression that contains them is evaluated. The MbXPath class has three methods for assigning and removing these variable bindings from user Java code. The value must be one of the four XPath 1.0 supported types:

- Boolean
- node set
- number
- string

XPath namespace support

For XML messages, namespaces are referred to by using a mapping from an abbreviated namespace prefix to the full namespace URI, as shown in the following XML example:

```
<ns1:aaa xmlns:ns1='http://mydomain.com/namespace1'
         xmlns:ns2='http://mydomain.com/namespace2'>
  <ns2:aaa>
    <ns1:bbb/>
  </ns2:aaa>
</ns1:aaa>
```

The namespace prefix is convenient for representing the namespace, but is meaningful only within the document that defines that mapping. The namespace URI defines the global meaning. Also, the concept of a namespace prefix is not meaningful for documents that are generated in a message flow, because a namespace URI can be assigned to a syntax element without an XMLNS mapping having been defined.

For this reason, the XMLNSC and MRM parsers expose only the namespace URI to the broker and to user code (ESQL or user-defined code). By using ESQL, you can set up your own mappings to create abbreviations to these potentially long URIs. These mappings are not related in any way to the prefixes that are defined in the XML document (although they can be the same name).

By using the XPath processor you can map namespace abbreviations on to URIs that are expanded at evaluation time. The MbXPath class contains methods to assign and remove these namespace mappings. For example:

```
MbXPath xp = new MbXPath("/aaa/other:aaa/bbb");
xp.addNamespacePrefix("other", "http://mydomain.com/namespace2");
xp.setDefaultNamespace("http://mydomain.com/namespace2");
List nodeset = (List)message.evaluateXPath(xp);
```

Updating a message by using XPath extensions

The XPath implementation in WebSphere Message Broker provides the following extra functions for modifying the message tree:

set-local-name(*object*)
: Sets the local part of the expanded name of the context node to the value specified in the argument. *object* can be any valid expression, and is converted to a string as if a call to the string function is used.

set-namespace-uri(*object*)
: Sets the namespace URI part of the expanded name of the context node to the value specified in the argument. *object* can be any valid expression, and is converted to a string as if a call to the string function is used.

set-value(*object*)
: This function sets the string value of the context node to the value specified in the argument. *object* can be any valid expression, and is converted to a string as if a call to the string function is used.

To allow for syntax element trees to be built as well as modified, the following axis is available in addition to the 13 that are defined in the XPath 1.0 specification:

select-or-create::*name* or ?*name*
: ?*name* is equivalent to select-or-create::*name*. If *name* is @name, an attribute

is created or selected. This selects child nodes matching the specified name, or creates new nodes according to the following rules:

- ?*name* selects children called *name* if they exist. If a child called *name* does not exist, ?*name* creates it as the last child, then selects it.
- ?$*name* creates *name* as the last child, then selects it.
- ?^*name* creates *name* as the first child, then selects it.
- ?>*name* creates *name* as the next sibling, then selects it.
- ?<*name* creates *name* as the previous sibling, then selects it.

Related tasks:

"Using XPath" on page 1506
XPath provides an alternative method to ESQL for entering expressions in the property fields of specific built-in nodes.

Related reference:

"ESQL-to-XPath mapping table" on page 5046
A table that summarizes the mappings from ESQL to XPath.

"JavaCompute node" on page 4514
Use the JavaCompute node to work with messages by using the Java language.

Related information:

Java user-defined extensions API

Manipulating other parts of the message tree by using a JavaCompute node:

You can access parts of the message tree other than the message body data. These parts of the logical tree are independent of the domain in which the message exists, and all these topics apply to messages in all supported domains, including the BLOB domain. You can access all parts of the message tree by using a JavaCompute node.

About this task

Elements of the message tree can be accessed in the same way as the message body data, by using a JavaCompute node.

- "Accessing headers by using a JavaCompute node" on page 2652
- "Updating the local environment with the JavaCompute node" on page 2655
- "Updating the Global Environment with the JavaCompute node" on page 2656

Related concepts:

"Logical tree structure" on page 1042
The logical tree structure is the internal (broker) representation of a message. It is also known as the message assembly.

"Environment tree structure" on page 1055
The environment tree is a part of the logical message tree in which you can store information while the message passes through the message flow.

"Local environment tree structure" on page 1056
The local environment tree is a part of the logical message tree in which you can store information while the message flow processes the message.

"Exception list tree structure" on page 1066
The exception list tree is a part of the logical message tree in which the message flow writes information about exceptions that occur when a message is processed.

"Message tree structure" on page 1045
The message tree is a part of the logical message tree in which the broker stores its internal representation of the message body.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

Accessing headers by using a JavaCompute node:

You can access headers by using a JavaCompute node.

About this task

If an input node receives an input message that includes message headers that the input node recognizes, the node starts the correct parser for each header. Parsers are supplied for most WebSphere MQ headers. The following topics provide guidance for accessing the information in the MQMD and MQRFH2 headers that you can follow when accessing other headers that are present in your messages.

- "Copying message headers by using a JavaCompute node"
- "Accessing the MQMD header by using a JavaCompute node" on page 2653
- "Accessing the MQRFH2 header by using a JavaCompute node" on page 2654

For further details of the contents of these and other WebSphere MQ headers for which WebSphere Message Broker provides a parser, see "Element definitions for message parsers" on page 4237.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

Related reference:

"Element definitions for message parsers" on page 4237

Copying message headers by using a JavaCompute node:

You can copy message headers by using a JavaCompute node.

The *Modifying Message Class* template in the JavaCompute node wizard generates the following code to copy message headers using a JavaCompute node:

```
public void copyMessageHeaders(MbMessage inMessage, MbMessage outMessage) throws MbException
{
 MbElement outRoot = outMessage.getRootElement();
 MbElement header = inMessage.getRootElement().getFirstChild();

 while(header != null && header.getNextSibling() != null)
 {
  outRoot.addAsLastChild(header.copy());
  header = header.getNextSibling();
 }
}
```

Related reference:

"JavaCompute node" on page 4514
Use the JavaCompute node to work with messages by using the Java language.

Accessing the MQMD header by using a JavaCompute node:

WebSphere MQ and WebSphere MQ Everyplace messages include an MQMD header. You can use a JavaCompute node to refer to the fields in the MQMD, and to update them.

About this task

The following Java code shows how to add an MQMD header to your message:

```
public void addMqmd(MbMessage msg) throws MbException
{
 MbElement root = msg.getRootElement();

 // create a top level 'parser' element with parser class name
 MbElement mqmd = root.createElementAsFirstChild("MQHMD");

 // specify next parser in chain
 mqmd.createElementAsFirstChild(MbElement.TYPE_NAME_VALUE,
  "Format",
  "XMLNS");
}
```

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

Accessing the MQRFH2 header by using a JavaCompute node:

You can use a JavaCompute node to add an MQRFH2 header to an outgoing message.

About this task

When you construct MQRFH2 headers in a JavaCompute node, two types of field exist:

- Fields in the MQRFH2 header structure (for example, Format and NameValueCCSID)
- Fields in the MQRFH2 NameValue buffer (for example mcd and psc)

The following code adds an MQRFH2 header to an outgoing message that is to be used to make a subscription request:

```
public void addRfh2(MbMessage msg) throws MbException
{
 MbElement root = msg.getRootElement();
 MbElement body = root.getLastChild();

 // insert new header before the message body
 MbElement rfh2 = body.createElementBefore("MQHRF2");

 rfh2.createElementAsFirstChild(MbElement.TYPE_NAME_VALUE, "Version", new Integer(2));
 rfh2.createElementAsFirstChild(MbElement.TYPE_NAME_VALUE, "Format", "MQSTR");
 rfh2.createElementAsFirstChild(MbElement.TYPE_NAME_VALUE, "NameValueCCSID", new Integer(1208));

 MbElement psc = rfh2.createElementAsFirstChild(MbElement.TYPE_NAME, "psc", null);
 psc.createElementAsFirstChild(MbElement.TYPE_NAME, "Command", "RegSub");
 psc.createElementAsFirstChild(MbElement.TYPE_NAME, "Topic", "department");
 psc.createElementAsFirstChild(MbElement.TYPE_NAME, "QMgrName", "QM1");
 psc.createElementAsFirstChild(MbElement.TYPE_NAME, "QName", "PUBOUT");
 psc.createElementAsFirstChild(MbElement.TYPE_NAME, "RegOpt", "PersAsPub");

 MbXPath xp = new MbXPath("/MQMD/Format" + "[set-value(´MQHRF2´)]", root);
 root.evaluateXPath(xp);
}
```

In this example, the `MQHRF2` parameter is the parser class name, which is different from the parser element name (MQRFH2). For a list of the parsers, root element names, and class names for different headers, see "Available parsers" on page 6689.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow

nodes and connecting them to form flows.

Related reference:

"MQRFH2 header" on page 6397
The MQRFH2 header is used to pass messages to and from a message broker that belongs to WebSphere Message Broker.

Updating the local environment with the JavaCompute node:

The local environment tree is part of the logical message tree in which you can store information while the message flow processes the message.

About this task

The following information shows how to update the local environment:

Procedure

1. Make a new copy of the local environment to update it. Use the full version of the copy constructor to create a new MbMessageAssembly object, as shown in the following example:

```
MbMessage env = assembly.getLocalEnvironment();
MbMessage newEnv = new MbMessage(env);

newEnv.getRootElement().createElementAsFirstChild(
  MbElement.TYPE_NAME_VALUE,
  "Status",
  "Success");

MbMessageAssembly outAssembly = new MbMessageAssembly(
  assembly,
  newEnv,
  assembly.getExceptionList(),
  assembly.getMessage());

getOutputTerminal("out").propagate(outAssembly);
```

2. Edit the copy to update the local environment.

Related concepts:

"Local environment tree structure" on page 1056
The local environment tree is a part of the logical message tree in which you can store information while the message flow processes the message.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Writing Java" on page 2638
When you create a message flow, you include input nodes that receive messages and, optionally, output nodes that send out new or updated messages. If the message processing requires it, you can include other nodes after the input node that are customized in Java to complete the actions that your applications need.

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Creating destination lists" on page 1477
Create a list of destinations to indicate where a message is sent.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

Related information:

Java user-defined extensions API

Updating the Global Environment with the JavaCompute node:

The Global Environment tree is always created when the logical tree is created for an input message. However, the message flow neither populates it nor uses its contents. You can use this tree for your own purposes, for example to pass information from one node to another. You can use the whole tree as a scratchpad or working area.

About this task

The Global Environment can be altered across the message flow, therefore do not make a copy of it to alter. The following Java code shows how to alter the Global Environment:

```
MbMessage env = assembly.getGlobalEnvironment();
env.getRootElement().createElementAsFirstChild(MbElement.TYPE_NAME_VALUE, "Status", "Success");

getOutputTerminal("out").propagate(assembly);
```

Related concepts:

"Environment tree structure" on page 1055
The environment tree is a part of the logical message tree in which you can store information while the message passes through the message flow.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

Accessing the ExceptionList tree using Java:

The ExceptionList tree is created with the logical tree when an input message is parsed.

About this task

The tree is initially empty, and is only populated if an exception occurs during message flow processing. It is possible that more than one exception can occur; if more than one exception occurs, the ExceptionList tree contains a subtree for each exception.

You can access the ExceptionList tree in JavaCompute nodes from the `MbMessageAssembly` parameter of your `evaluate()` method.

You can access the ExceptionList tree in a node in an error handling procedure. For example, you might want to route the message to a different path based on the type of exception.

You can use the querying capabilities within XPath to carry out this task. The descendant axis (//) of XPath gives you the ability to search for an element by name regardless of its depth in the tree. For example:

```
//ParserException
```

returns all elements in the tree named `ParserException`.

If you are specifically interested in a particular message, you can use a predicate (see predicates for further information) to narrow down the result set. For example:

```
//ParserException[Number=5016]
```

returns only the exception that contains `Number=5016`.

If you only want to extract the text message associated with this exception, the following XPath expression returns this as a `java.lang.String`:

```
string(//ParserException[Number=5016]/Text)
```

The following Java code extracts this text from your code:

```
String text =
(String)inAssembly.getExceptionList().evaluateXPath("string(//ParserException[Number=5016]/Text)");
```

For information on accessing the ExceptionList tree using ESQL, see "Accessing the ExceptionList tree using ESQL" on page 2471

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Exception list tree structure" on page 1066
The exception list tree is a part of the logical message tree in which the message flow writes information about exceptions that occur when a message is processed.

Related tasks:

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Accessing the ExceptionList tree using ESQL" on page 2471
The ExceptionList tree has its own correlation name, ExceptionList, and you must use this in all ESQL statements that refer to or set the content of this tree.

Related reference:

"JavaCompute node" on page 4514
Use the JavaCompute node to work with messages by using the Java language.

"Exception list structure" on page 4224
An exception list contains information about exceptions, such as error numbers, the name of the node that generated the exception, and the reason for the exception.

Accessing broker properties from the JavaCompute node:

For each broker, WebSphere Message Broker maintains a set of properties. You can access some of these properties from your Java programs. It can be useful, during the run time of your code, to have real-time access to details of a specific node, flow, or broker.

About this task

Broker properties are divided into four categories:

- Properties that relate to a specific node
- Properties that relate to nodes in general
- Properties that relate to a message flow
- Properties that relate to the execution group

"Broker properties that are accessible from ESQL and Java" on page 5302 includes a table that shows the groups of properties that are accessible from Java. The table also indicates if the properties are accessible from ESQL.

Broker properties have the following characteristics.

- They are grouped by broker, execution group, flow, and node.
- They are case sensitive. Their names always start with an uppercase letter.
- They return NULL if they do not contain a value.

To access broker properties in a JavaCompute node, call methods on the following classes:

- MbBroker
- MbExecutionGroup
- MbMessageFlow
- MbNode

For example:

```
String brokerName = getBroker().getName();
```

Related concepts:

"Broker properties" on page 1144
For each broker, WebSphere Message Broker maintains a set of properties. You can access some of these properties from your ESQL programs. A subset of the properties is also accessible from Java programs. It can be useful, at run time, to have real-time access to details of a specific node, flow, or broker.

"User-defined properties in ESQL" on page 2376
Access user-defined properties (UDPs) as variables in your ESQL program by specifying the EXTERNAL keyword on a DECLARE statement. For example, the ESQL statement `DECLARE today EXTERNAL CHARACTER 'monday'` defines a user-defined property called today with an initial value monday.

Related tasks:

"Accessing broker properties from ESQL" on page 2625
You can access broker properties, at run time, from the ESQL modules in your message flow nodes.

"Managing Java Files" on page 2629
The Java code that you provide to modify or customize the behavior of a JavaCompute node is stored in a Java project. WebSphere Message Broker uses the Eclipse Java perspective for developing and administering Java files.

"Writing Java" on page 2638
When you create a message flow, you include input nodes that receive messages and, optionally, output nodes that send out new or updated messages. If the message processing requires it, you can include other nodes after the input node that are customized in Java to complete the actions that your applications need.

Related reference:

"Broker properties that are accessible from ESQL and Java" on page 5302
You can access broker, message flow, and node properties from ESQL and Java.

Accessing message flow user-defined properties from a JavaCompute node:

Customize a JavaCompute node to access properties that you have associated with the message flow in which the node is included.

About this task

To access these properties from a JavaCompute node, use the `getUserDefinedAttribute(`*name*`)` method, where *name* is the name of the property that you are accessing. The type of the object that is returned depends on the type of the property that you are accessing. The object has one of a set of types:

- MbDate
- MbTime
- MbTimestamp
- Boolean
- byte[]
- String
- Integer 32-bit values
- Long 64-bit values
- Double
- BigDecimal
- BitSet

You cannot access user-defined properties in the constructor. To access them at initialization time, implement the following method, and use it to access the user-defined properties.

```
public void onInitialize() throws MbException
{
   // access the user-defined properties here
}
```

You can use the Administration API for WebSphere Message Broker (also known as the CMP API) to change the value of user-defined properties. Use the getUserDefinedPropertyNames(), getUserDefinedProperty(), and setUserDefinedProperty() methods to query, discover, and set user-defined properties, as described in "Setting message flow user-defined properties at run time in a CMP application" on page 985.

Related concepts:

"User-defined properties" on page 1147
A user-defined property (UDP) is a property that is defined when you construct a message flow by using the Message Flow editor. This property can be used by the ESQL or Java program inside message flow nodes, such as a Compute node.

"The Administration API for WebSphere Message Broker" on page 54
The Administration API for WebSphere Message Broker is a programming interface that your applications can use to control brokers and their resources through a remote interface.

Related tasks:

"Setting message flow user-defined properties at run time in a CMP application" on page 985
Use the CMP API to query, discover, and set message flow user-defined properties dynamically at run time. You can use the CMP API to set properties with a data type of character.

Related reference:

"JavaCompute node" on page 4514
Use the JavaCompute node to work with messages by using the Java language.

Adding keywords to JAR files:

If a BAR file contains JAR files, you can associate keywords with the JAR files.

Before you begin

Before you start:

Read about keywords in "Message flow version and keywords" on page 1445.

About this task

To add keywords to a JAR file:

Procedure

1. Add a file called `META-INF/keywords.txt` to the root of the JAR file.
2. Add your keywords to the `META-INF/keywords.txt` file, because this file is parsed for keywords when it is deployed. Keywords have this format:
   `$MQSI keyword = value MQSI$`

Example

For example, a deployed BAR file contains *compute.jar*, and *compute.jar* contains the file `META-INF/keywords.txt` with the following contents:

```
# META-INF/keywords.txt
$MQSI modified date = 3 Nov MQSI$
$MQSI author = john MQSI$
```

This content means that the keywords "modified date" and "author" are associated with the deployed file *compute.jar* in Administration API (also known as the CMP API) applications, including the WebSphere Message Broker Toolkit.

What to do next

You have now added keywords to your JAR file.

Next:

When you have added keywords to your JAR file, you can display them in the BAR file editor; for more information, see "Version and keyword information for deployable objects" on page 1443.

Related concepts:

"Message flow version and keywords" on page 1445
When you are developing a message flow, you can define the version of the message flow as well as other key information that you want to be associated with it.

"Version and keyword information for deployable objects" on page 1443
Use the Broker Archive editor to view the version and keyword information of deployable objects.

Related reference:

"Guidance for defining keywords" on page 4017
You can add extra information to an object in the form of one or more keywords.

Interacting with databases by using the JavaCompute node:

Access databases from Java code included in the JavaCompute node.

About this task

Choose from the following options for database interaction:

- Broker JDBCProvider for type 4 connections
- MbSQLStatement
- JDBC API in an unmanaged environment
- SQLJ

If you use JDBCProvider for type 4 connections or MbSQLStatement, the databases that you access can participate in globally coordinated transactions. In all other cases, database access cannot be globally coordinated.

Broker JDBCProvider for type 4 connections:
About this task

You can establish JDBC type 4 connections to interact with databases from your JavaCompute nodes. The broker supports type 4 drivers, but does not supply them. You must obtain these drivers from your database vendor; for information about supported drivers, see "Supported databases" on page 3591.

Use the broker JDBCProvider for type 4 connections to benefit from the following advantages:

- Use broker configuration facilities to define the connection, and to provide optional security, in preference to coding these actions.
- Configure the broker and the databases to coordinate access and updates with other resources that you access from your message flows, except when the broker is running on z/OS.

- Use the broker Java API getJDBCType4Connection to initiate the connection, then perform SQL operations by using the standard JDBC APIs. The broker manages the connections, thread affinity, connection pooling, and lifecycle. If a connection is idle for approximately 1 minute, or if the message flow completes, the broker closes the connection.

If the broker is running on a distributed system, you can configure the databases and the connections to be coordinated with other resource activity. Global coordination on distributed systems is provided by WebSphere MQ, and can include interactions with local or remote databases, including remote databases that are defined on z/OS systems. If you establish a JDBC type 4 connection to a database from a broker that is running on z/OS, coordination is not provided. For information about setting up connections and coordination, see "Enabling JDBC connections to the databases" on page 683.

Before you can include this function in the code that you write for the node, you must configure the required environment. Decide whether your database requires security of access, and whether you want the database updates to participate in globally coordinated transactions. For the required and optional tasks, see "Enabling JDBC connections to the databases" on page 683.

When you have configured the JDBCProvider, you can establish a JDBC type 4 connection to the database by using the getJDBCType4Connection call on the MbNode interface. The following code provides an example of its use:

```
public class MyJavaCompute extends MbJavaComputeNode {
    public void evaluate(MbMessageAssembly inAssembly) throws MbException {
      MbOutputTerminal out = getOutputTerminal("out");
      MbMessage inMessage = inAssembly.getMessage();

      // create new message
      MbMessage outMessage = new MbMessage(inMessage);
      MbMessageAssembly outAssembly = new MbMessageAssembly(inAssembly,outMessage);

      try {
        // Obtain a java.sql.Connection using a JDBC Type4 datasource - in this example for a
        // JDBC broker configurable service called "MyDB2"

        Connection conn = getJDBCType4Connection("MyDB2",
                    JDBC_TransactionType.MB_TRANSACTION_AUTO);

        // Example of using the Connection to create a java.sql.Statement
        Statement stmt = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                    ResultSet.CONCUR_READ_ONLY);
        ResultSet srs0 = stmt.executeQuery("SELECT NAME, CITY FROM MySchema.MyTable");

        stmt.executeUpdate("UPDATE MySchema.MyTable SET CITY = \"Springfield\" WHERE Name = \"Bart\"");
        .
        // Perform other database updates
        .

      } catch (SQLException sqx ){
        sqx.printStackTrace();
      } finally {
        // Clear the outMessage
        outMessage.clearMessage();
      }
    }
  }
```

In this example:

- *MyDB2* is the name of the JDBCProvider configurable service. Use the name of the service that you have created to connect to your database.
- *MySchema* is the name of the database schema (not the name of the database).
- MB_TRANSACTION_AUTO defines the level of transaction coordination that is required by the node. Only this value is supported, and indicates that the coordination in the node is inherited from the coordination configured at message flow level.

To indicate a failure (and roll back a transaction), issue an exception from the JavaCompute node and the broker will handle the rollback.

The primary use of the getJDBCType4Connection call is in the `evaluate()` method of a JavaCompute node, where it is used to obtain a JDBC connection that is managed by the broker.

WebSphere Message Broker manages JDBC connections in the following ways:

- Non-pooled connections:
  - WebSphere Message Broker creates a JDBC connection on demand for each message flow instance that requires one.
  - Each JDBC connection is associated with the message flow instance for which it was created. This association is maintained until the connection is closed.
  - Each JDBC connection that is idle for 60 seconds is closed, and is no longer associated with a message flow instance.
  - After a JDBC connection that was associated with a message flow instance is closed, if the same message flow instance requires a JDBC connection, WebSphere Message Broker creates a new JDBC connection on demand.
- Pooled connections:
  - When a message flow instance requires a JDBC connection, WebSphere Message Broker assigns an unused connection from the pool.
  - If all pooled JDBC connections are being used, and the maximum pool size has not been reached, WebSphere Message Broker creates a new pooled JDBC connection. The maximum pool size is specified in the `maxConnectionPoolSize` property of the "JDBCProviders configurable service" on page 3778.
  - Each pooled JDBC connection remains associated with a message flow instance only for the processing of one input message.
  - When a message flow instance completes the processing of an input message, the association with a JDBC connection is removed, and the JDBC connection is returned to the pool.
  - Each pooled JDBC connection that is idle for 15 minutes is closed, and is removed from the pool.
  - Pooled JDBC connections are not applicable to the DatabaseRetrieve and DatabaseRoute nodes.

When using the getJDBCType4Connection call, your code must comply with the following restrictions:

- Do not include code that makes explicit transaction calls such as COMMIT or ROLLBACK. This restriction includes explicit transaction calls in a database stored procedure.
- Do not close a connection, or cache a connection in the JavaCompute node.

A secondary use of the getJDBCType4Connection call is in the `onitialize()` method of a JavaCompute node. The `onitialize()` method is called once, either

during deployment or on broker startup, before the message flow starts processing input. You can use the getJDBCType4Connection call in the `onitialize()` method to complete work with a database before the message flow starts; for example:

- To create an in-memory cache of read-only data that is retrieved from a database, to reduce the need to query the database in the message flow
- To prime a database with data before the message flow starts

When using the getJDBCType4Connection in the `onitialize()` method, ensure that any exceptions that might occur in this processing are handled. Any unhanded exception causes the deployment or startup of the message flow to fail. For more information, see "JavaCompute node" on page 4514.

MbSQLStatement:
About this task

The MbSQLStatement class provides full transactional database access by using ESQL and ODBC. The broker resource manager coordinates database access when using MbSQLStatement. Global coordination is provided by WebSphere MQ on distributed systems, and by RRS on z/OS. For information about how to set up the ODBC resources that are required, see "Enabling ODBC connections to the databases" on page 668.

Create instances of the MbSQLStatement class by using the createSQLStatement() method of MbNode, passing to the method the ODBC data source, a broker EQSL statement, and, optionally, the transaction mode.

- Calling select() on this object returns the results of the query.
- Calling execute() on this object runs a query where no results are returned, such as updating a table.

The following Java code shows how to access a database by using MbSQLStatement:

```
MbMessage newMsg = new MbMessage(assembly.getMessage());
MbMessageAssembly newAssembly = new MbMessageAssembly(assembly, newMsg);

String table = "dbTable";

MbSQLStatement state = createSQLStatement( "dbName",
 "SET OutputRoot.XMLNS.integer[] = PASSTHRU('SELECT * FROM " + table + "');" );

state.setThrowExceptionOnDatabaseError(false);
state.setTreatWarningsAsErrors(true);
state.select( assembly, newAssembly );

int sqlCode = state.getSQLCode();
if(sqlCode != 0)
{
 // Do error handling here
}

getOutputTerminal("out").propagate(assembly);
```

JDBC API in an unmanaged environment:
About this task

You can access standard Java APIs in the code that you write for your JavaCompute nodes, including JDBC calls. You can therefore use JDBC APIs to connect to a database, write to or read from the database, and disconnect from the database. On operating systems other than z/OS, the broker supports your JDBC

connection code calling both type 2 and type 4 JDBC drivers in this environment, but does not supply them. You must obtain these drivers from your database vendor. On z/OS, type 2 drivers are not supported.

If you choose this method to access databases, the broker does not support managing the transactions; your code must manage the local commit and rollback of database changes. Your code must also manage the connection lifecycle, connection thread affinity, and connection pooling. You must also monitor the access to databases when you use this technique to ensure that these connections do not cause interference with connections made by the broker. In particular, be aware that type 2 drivers bridge to an ODBC connection that might be in use in message flows that access databases from ESQL.

SQLJ:
About this task

SQLJ is a Java extension that you can use to embed static SQL statements within Java code. Create SQLJ files by using the WebSphere Message Broker Toolkit. The broker resource manager does not coordinate database access when using SQLJ.

Procedure

1. Enable SQLJ capability in the WebSphere Message Broker Toolkit:
   a. Select **Window** > **Preferences**.
   b. Expand **General**.
   c. Select **Capabilities**.
   d. Select **Data**.
   e. Click **OK**.
2. Create an SQLJ file within a Java project:
   a. Right-click the ***Java project*** in which you want to create the file.
   b. Select **New** > **Other**.
   c. Expand **Data**.
   d. Expand **SQLJ Applications**.
   e. Select **SQLJ File**.
   f. Click **Next**.
   g. Follow the directions given by the New SQLJ File wizard to generate the SQLJ file.

Results

You can now reference the class in this SQLJ file from a JavaCompute node class in this project or in another referenced project.

Related concepts:

"Project references" on page 44
When a projects refers to other projects in the workspace, this is called a *project reference*. When one project references another, the files in the referenced project are available for use by the referring project. For example to configure certain nodes, to create libraries of reusable message flows, and to enable Content Assist in the ESQL editor.

Related tasks:

"Enabling JDBC connections to the databases" on page 683
Configure connections to a user database through a JDBCProvider service.

"Enabling ODBC connections to the databases" on page 668
Set up the resources and environment that the broker requires for Open Database Connectivity (ODBC) connections to user databases on distributed systems.

"Writing Java" on page 2638
When you create a message flow, you include input nodes that receive messages and, optionally, output nodes that send out new or updated messages. If the message processing requires it, you can include other nodes after the input node that are customized in Java to complete the actions that your applications need.

Related reference:

"Supported databases" on page 3591
You can optionally configure databases to contain data that is accessed by your message flows. Databases from IBM and other suppliers are supported at specific versions on supported operating systems.

"JavaCompute node" on page 4514
Use the JavaCompute node to work with messages by using the Java language.

Related information:

Java user-defined extensions API

Calling an Enterprise Java Bean:

You can call an Enterprise Java Bean (EJB) from a JavaCompute node.

Before you begin

Before you start:

- Ensure that all required Java classes are in WebSphere Message Broker shared-classes directories, or are referenced in the CLASSPATH environment variable. You can use the wildcard character (*) at the end of a directory path specifier to load all JARs in that directory path.
- Ensure that the user JAR files that are needed for EJB access are referenced in CLASSPATH. For more information, see the documentation for the application server that is hosting the EJB.
- If you are using a version of WebSphere Message Broker before Version 6.0 Fix Pack 3, you must set the context loader by including the following statement in the node's Java code before the InitialContext is set:

  ```
  Thread currentThread().setContextClassLoader(this.getClass().getClassLoader());
  ```

Example

The following example shows how to call an EJB from a JavaCompute node:

```
public class CallAckNoAckEJB_JavaCompute extends MbJavaComputeNode {

   public void evaluate(MbMessageAssembly inAssembly) throws MbException {
                MbOutputTerminal out = getOutputTerminal("out");
                MbOutputTerminal alt = getOutputTerminal("alternate");

                MbMessage inMessage = inAssembly.getMessage();

                // create new message
                MbMessage outMessage = new MbMessage(inMessage);
                MbMessageAssembly outAssembly = new MbMessageAssembly(inAssembly,outMessage);

                try {
                        // ----------------------------------------------------------
                        // Add user code below
```

```
                        String response = null;
                        String responseMessage = null;

                        Properties properties = new Properties();
                        properties.put(Context.PROVIDER_URL, "iiop://localhost:2809");
                        properties.put(Context.INITIAL_CONTEXT_FACTORY, "com.ibm.websphere.naming.
WsnInitialContextFactory");

                        try {

                           Context initialContext = new InitialContext(properties);
                                Object obj = initialContext.lookup("ejb/com/acme/ejbs/AckNoAckHome");
                                AckNoAckHome ejbHome = (AckNoAckHome)javax.rmi.PortableRemoteObject.
narrow(obj,AckNoAckHome.class);

                                AckNoAck ackNoAck = ejbHome.create();
                                responseMessage = ackNoAck.getAck();
                                response = "Ack";
                        } catch(Exception e) {
                                responseMessage = e.getMessage();
                                response = "NoAck";
                        }

                        MbElement cursor = outMessage.getRootElement().getFirstElementByPath("/XML/AckNoAck");
                        cursor.createElementAsLastChild(MbElement.TYPE_NAME,"Response",null);
                        cursor.getLastChild().createElementAsLastChild(MbElement.TYPE_NAME,response,null);
                        cursor.getLastChild().getLastChild().createElementAsLastChild(MbElement.TYPE_VALUE,null,
responseMessage);

                        // End of user code
                        // ------------------------------------------------------------

                        // The following should only be changed
                        // if not propagating message to the 'out' terminal
                        out.propagate(outAssembly);

                } finally {
                        // clear the outMessage
                        outMessage.clearMessage();
                }
        }
}
```

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"User-defined node class loading" on page 3120
Details the Java classes packaging options and loading order precedence for user-defined nodes.

"JavaCompute node classloading" on page 2635
Details the default Java classloader options and the precedence order of each type.

Related tasks:

"Creating Java code for a JavaCompute node" on page 2629
Use these instructions to associate Java code with your JavaCompute node.

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Writing Java" on page 2638
When you create a message flow, you include input nodes that receive messages and, optionally, output nodes that send out new or updated messages. If the message processing requires it, you can include other nodes after the input node that are customized in Java to complete the actions that your applications need.

Related reference:

"JavaCompute node" on page 4514
Use the JavaCompute node to work with messages by using the Java language.

JavaCompute node exception handling:

The `evaluate()` method throws an `MbException`.

If your code throws other classes of checked exceptions, they must be caught and rethrown as `MbException`. Typically, you use the `MbUserException` class. `MbUserException` is a subclass of `MbException`, and has a public constructor. WebSphere Message Broker utility functions can throw other subclasses of MbException that have private constructors.

For information about what happens after the `evaluate()` method is processed, see "Default error handling" on page 1280.

About this task

Exceptions can occur during message flow processing (during the `evaluate()` method) and during the `onInitialize()` method. If you have implemented the `onInitialize()` method, and you detect an unrecoverable error in the node configuration, an exception of class `MBException` is issued and the flow fails to initialize. Either your flow fails to deploy, and error message BIP4157 is issued, or the flow does not start and does not appear in the list of running flows.

Do not run code in an `onInitialize()` method that depends on another external resource that might not be there because the broker will not retry calls to that method. If you need to initialize an external connection, for example, initialize it during the first call to the `evaluate()` method. If the initialization does not work, you can throw an exception that will be handled like any other flow exception.

Related reference:

"JavaCompute node" on page 4514
Use the JavaCompute node to work with messages by using the Java language.

Logging errors with the JavaCompute node:

The MbService class contains a number of static methods for writing to the Administration log or syslog. Define message catalogs by using Java resource bundles to store the message text.

About this task

Three levels of severity are supported:
- Information
- Warning
- Error

The following sample demonstrates the use of resource bundles and logging:
- JavaCompute Node

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Related tasks:

"Writing Java" on page 2638
When you create a message flow, you include input nodes that receive messages and, optionally, output nodes that send out new or updated messages. If the message processing requires it, you can include other nodes after the input node that are customized in Java to complete the actions that your applications need.

Related reference:

"JavaCompute node" on page 4514
Use the JavaCompute node to work with messages by using the Java language.

Related information:

Java user-defined extensions API

Using XSL Transform

Use the XSLTransform node to transform an XML message to another form of message, according to the rules provided by an XSL (Extensible Stylesheet Language) style sheet.

About this task

Use the XSLTransform node to transform an input XML message into another format using XSLT style sheets and to set the message domain, message set, message type, and message format for the generated message. It is imperative that the data can be parsed into a XML message. The style sheet, using the rules that are defined within it, can perform the following actions:

- Sort the data
- Select data elements to include or exclude based on some criteria
- Transform the data into another format

The Xalan-Java transformation engine (Apache Xalan-java XSLT processor) is used as the underlying transformation engine. For more information about XML Transformations, the W3C specification of the syntax, and semantics of the XSL Transformations language for transforming XML documents into other XML documents, see W3C XSL Transformations.

You can deploy style sheets and XML files to broker execution groups, to help with style sheet and XML file maintenance.

You can specify the location of the style sheet to apply to this transformation in three ways:

- Use the content of the XML data within the message itself to transform the message according to a style sheet that the message itself defines.
- Set a value in the LocalEnvironment folder. You must set this value in a node that precedes the XSLTransform node (for example, a Compute node). You can therefore use various inputs to determine which style sheet to use for this message, such as the content of the message data, or a value in a database.
- Use node properties to ensure that the transformation that is defined by this single style sheet is applied to every message that is processed by this node.

An XSLT (Extensible Stylesheet Language for Transformations) compiler is used for the transformation if the style sheet is not embedded within the message, and the node cache level (node property Stylesheet Cache Level) is greater than zero. If the XSLT is cached, the performance is improved because the XSLT is not parsed every time it is used.

If the prologue of the input message body contains an XML encoding declaration, the XSLTransform node ignores the encoding, and always uses the CodedCharSetId in the message property folder to decode the message.

The XSLT capability that is provided by the XSLTransform node requires XML processing APIs that are included in Xalan-Java and Xerces JAR files. The XSLTransform node provides Xalan-Java and Xerces JAR files that work correctly with the node. The Java JRE also includes Xalan-Java and Xerces JAR files, but you might experience unpredictable results when these Java XML processing methods are invoked by using an external Java method from a style sheet. Therefore, the calling of Java methods from a style sheet that directly or indirectly reference Java JRE XML processing methods is unsupported.

To find out more about the XSLTransform node and how to configure it, refer to the following topics:
- "XSLTransform node" on page 4968
- "Adding keywords to XSL style sheets" on page 4975
- "Using local environment variables to set properties" on page 4976
- "Deployed and non-deployed style sheets" on page 4978

Using PHP

You can use the PHP scripting language for message routing and transformation.

About this task

Support for the PHP scripting language is available on all operating systems on which WebSphere Message Broker is supported.

When you use the PHPCompute node, you can customize it to determine the exact processing that it provides. To tailor the behavior of each node, create a PHP file that provides the processing that you require. You can edit your PHP files by using a text editor such as the default Eclipse text editor.

The PHP API simplifies tasks that involve message routing and transformation tasks. These tasks include accessing named elements in a message tree, setting their values, and creating elements, without the need to navigate the message tree explicitly.

This section provides the following information about developing PHP:
- "PHP overview" on page 2671
- "Creating PHP code for a PHPCompute node" on page 2672
- "Using PHP arrays with XML" on page 2682
- "Using PHP arrays with JSON" on page 2685
- "Deploying code in a PHPCompute node" on page 2691
- "Calling Java from PHP" on page 2716
- "Creating and transforming messages using a PHPCompute node" on page 2701
- "Accessing elements in the message tree from a PHPCompute node" on page 2692
- "XML support" on page 2708
- "Routing a message using a PHPCompute node" on page 2709
- "Accessing other parts of the message tree using the PHPCompute node" on page 2710

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"PHPCompute node" on page 4639
Use the PHPCompute node to route and transform an incoming message, using the PHP scripting language.

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

"PHP API" on page 5313
The WebSphere Message Broker PHP API provides classes to represent the message, the message assembly, and the element tree.

PHP overview

WebSphere Message Broker provides support for the PHP scripting language.

WebSphere Message Broker provides a PHPCompute node, which is a programmable node that supports message transformation and routing by using the PHP scripting language. For example:

```
$output_assembly->XMLNSC->doc->item =
$input_assembly->MRM->structure->field;
```

This PHP code generates the following XML code:

```
<doc>
    <item>
           ... deep copy of field element from input tree
    </item>
</doc>
```

The PHPCompute node builds on this syntax to produce a powerful syntax for accessing WebSphere Message Broker trees.

Standard output and standard error messages from the PHP engine are written to the console log for the broker. For information about reading the console log, see "Standard system logs" on page 6866.

z/OS On z/OS, all PHP scripts must be encoded in UTF-8 format. All string literals in PHP scripts, and all character data that is passed to scripts from the message assembly, are in UTF-8 encoding. Typically, scripts that interact with the message assembly in PHP work in the same way on z/OS as they do on other operating systems. If a script uses byte streams in PHP (such as file access), the Multibyte String functions can be used to detect the code page and convert character data as required.

UTF-8 might not be the default code page of the broker, which means that any function that requires the broker to have a UTF-8 default code page might need to

be modified. For example, the asBitstream and addElementFromBitstream methods on the MbsElement object use the default code page of the broker as their default code page, rather than the code page of PHP.

The PHPCompute node provides support for Simple Network Management Protocol (SNMP). The Management Information Base (MIB) files are installed during the installation of WebSphere Message Broker, and their location is specified by the **MIBDIRS** environment variable.

For information about the PHP functions supported by WebSphere Message Broker, see "PHP extensions" on page 5324.

For more information about the PHP scripting language, see the PHP: Hypertext Preprocessor web site.

Related concepts:

"XML support" on page 2708
The PHP capability in WebSphere Message Broker provides support for XML.

Related tasks:

"Creating PHP code for a PHPCompute node"
Use these instructions to create your PHP code and associate it with your PHPCompute node.

"Using PHP" on page 2670
You can use the PHP scripting language for message routing and transformation.

Related reference:

"PHPCompute node" on page 4639
Use the PHPCompute node to route and transform an incoming message, using the PHP scripting language.

"PHP API" on page 5313
The WebSphere Message Broker PHP API provides classes to represent the message, the message assembly, and the element tree.

Creating PHP code for a PHPCompute node

Use these instructions to create your PHP code and associate it with your PHPCompute node.

About this task

To create your PHP code and associate it with a PHPCompute node, follow these steps:

Procedure

1. Ensure that your PHP code is in a PHP script file, with an extension of `.php`:
   - If the required PHP code exists, import the PHP script file into the workspace (see "Importing file systems into the WebSphere Message Broker Toolkit" on page 2931). Alternatively, ensure that the PHP script file is stored in the file system, so that the PHPCompute node can point to it directly.
   - If the required PHP code does not exist, create it by following the instructions in "Writing PHP code" on page 2673.
2. Associate the PHP code with a PHPCompute node in your message flow. Follow the instructions in "Associating PHP code with a PHPCompute node" on page 2675.

Related concepts:

"PHP overview" on page 2671
WebSphere Message Broker provides support for the PHP scripting language.

Related tasks:

"Using PHP" on page 2670
You can use the PHP scripting language for message routing and transformation.

"Importing file systems into the WebSphere Message Broker Toolkit" on page 2931
You can import file systems into the WebSphere Message Broker Toolkit by using the Import wizard, by dragging, or by copying.

"Writing PHP code"
Use these instructions to create your PHP code.

"Associating PHP code with a PHPCompute node" on page 2675
Use these instructions to associate your PHP code with a PHPCompute node.

Related reference:

"PHPCompute node" on page 4639
Use the PHPCompute node to route and transform an incoming message, using the PHP scripting language.

"PHP API" on page 5313
The WebSphere Message Broker PHP API provides classes to represent the message, the message assembly, and the element tree.

Writing PHP code:

Use these instructions to create your PHP code.

About this task

Procedure

1. In the Broker Application Development perspective, click **File** > **New** > **Other**. The Select a wizard pane is displayed.
2. Select **File** from the list of resources, then click **Next**.
3. Select the required parent folder from the list, then type the name of your new PHP script file in the **File Name** field. Ensure that the file you specify has an extension of `.php` (for example, `Hello.php`). The Eclipse text editor opens, with an empty pane in which you can type your PHP code.
4. Type your PHP code into your new PHP script file, by using the Eclipse text editor. The PHP script must be contained within the `<?php` and `?>` tags:

   ```
   <?php

   // Body of the script

   ?>
   ```

   You can create a PHP script with or without a class and evaluate method. The option you choose affects both the content of the script and the setting of the PHPCompute node's `Invoke evaluate method` property:

   - Create a script including a class and evaluate method:

     The `Invoke evaluate method` property of the PHPCompute node is selected by default, therefore a class and evaluate method are expected in the PHP script.

     The PHP code must contain a class with the same name as the PHP file (`Hello`, for example), and this class must contain a function called evaluate, with parameters for the input and output message assemblies:

```
<?php

class Hello {

  /**
   * An example of MessageBrokerSimpleTransform
   * @MessageBrokerSimpleTransform
   */
  function evaluate($output_assembly, $input_assembly) {
   // transformation code here
   // $output_assembly ->XMLNSC->... = $input_assembly->XMLNSC->...
  }

}

?>
```

For more information about the @MessageBrokerSimpleTransform annotation shown in this example, see "Using annotations" on page 2676.

- Create a script without a class and evaluate method:

  The global variable *$assembly* makes the incoming message assembly available to the script. The incoming message and message assembly are read only. As a result, for message transformation, you must make a new copy of the message and the assembly containing the new message:

```
<?php

$output_message = new MbsMessage();

// transformation code here
// $output_message->XMLNSC->... = $assembly->XMLNSC->...

$output_assembly = new MbsMessageAssembly($assembly, $output_message);
$output_assembly->propagate("out");

?>
```

  The `Invoke evaluate method` property of the PHPCompute node is selected by default, therefore a class and evaluate method are expected in the PHP script. If you use a PHP script without a class and evaluate method, remember to clear the **`Invoke evaluate method`** property of the PHPCompute node.

  You must explicitly propagate the message assembly to one of the output terminals before the end of the script.

What to do next

When you have created your PHP code, associate it with the PHPCompute node by following the instructions in "Associating PHP code with a PHPCompute node" on page 2675.

For information about the PHP scripting language, see the PHP: Hypertext Preprocessor Web site.

Related concepts:

"Using annotations" on page 2676
Annotations alter the behavior of the evaluate method when using the PHP class structure, and remove the need to repeat commonly used code (for transforming message trees) in each PHP script.

Related tasks:

"Using PHP" on page 2670
You can use the PHP scripting language for message routing and transformation.

"Associating PHP code with a PHPCompute node"
Use these instructions to associate your PHP code with a PHPCompute node.

Related reference:

"PHPCompute node" on page 4639
Use the PHPCompute node to route and transform an incoming message, using the PHP scripting language.

"PHP API" on page 5313
The WebSphere Message Broker PHP API provides classes to represent the message, the message assembly, and the element tree.

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

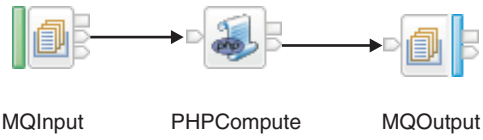
Associating PHP code with a PHPCompute node:

Use these instructions to associate your PHP code with a PHPCompute node.

About this task

Procedure

1. Create a message flow containing a PHPCompute node. For example, create a message flow containing an MQInput node, a PHPCompute node, and an MQOutput node. For information, see "Creating a message flow" on page 1431.
2. Connect the Out terminal of the MQInput node to the In terminal of the PHPCompute node.
3. Connect the Out terminal of the PHPCompute node to the In terminal of the MQOutput node.



4. On the MQInput node, set the `Queue name` property (on the **Basic** tab) to `InQueue`.
5. On the PHPCompute node, set the `PHP script` property (on the **Basic** tab) to `Hello.php`. You can either type the name of the PHP file into the field, or select **Browse** to navigate to the PHP files stored in your workspace. If you select **Browse**, the PHP files in the current message flow project are displayed, together with PHP files in any referenced projects. To add project references, right-click the message flow project, select **Properties**, then select the **Project references** category.
6. Set the following properties of the MQOutput node:
   a. On the **Basic** tab, set the `Queue name` property to `OutputQueue`
   b. On the **Validation** tab, set the `Validate` property to `Inherit`.
7. Save the message flow.

Results

You can display the PHP script file associated with the PHPCompute node by double-clicking the node in the message flow. The file `Hello.php` is opened in the Eclipse text editor.

Related tasks:

"Using PHP" on page 2670
You can use the PHP scripting language for message routing and transformation.

"Writing PHP code" on page 2673
Use these instructions to create your PHP code.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

Related reference:

"PHPCompute node" on page 4639
Use the PHPCompute node to route and transform an incoming message, using the PHP scripting language.

"PHP API" on page 5313
The WebSphere Message Broker PHP API provides classes to represent the message, the message assembly, and the element tree.

Using annotations:

Annotations alter the behavior of the evaluate method when using the PHP class structure, and remove the need to repeat commonly used code (for transforming message trees) in each PHP script.

When you use the PHP class structure with WebSphere Message Broker, the class must have the same name as the PHP file and it must implement a method called evaluate. The PHPCompute node instantiates the class and calls the evaluate method. For more information about developing PHP code, see "Creating PHP code for a PHPCompute node" on page 2672.

The following annotations are supported by the broker:

- "@MessageBrokerSimpleTransform" on page 2677
- "@MessageBrokerCopyTransform" on page 2678
- "@MessageBrokerRouter" on page 2679
- "@MessageBrokerLocalEnvironmentTransform" on page 2680

You can specify multiple annotations for an evaluate method. If the MessageBrokerCopyTransform and MessageBrokerSimpleTransform annotations are specified together, the MessageBrokerCopyTransform annotation takes precedence. The input assembly is available with both the MessageBrokerSimpleTransform and MessageBrokerCopyTransform annotations.

If no annotations are specified, the first argument to the evaluate method is a read-only assembly. Annotation names are case-sensitive, and annotations that are not recognized are ignored.

When you use an annotation, the output assembly is passed to the evaluate method as the first parameter, and the input assembly is passed as the second parameter. The second parameter is optional and is passed in if you have specified it in your evaluate method declaration.

Related concepts:

"Local environment tree structure" on page 1056
The local environment tree is a part of the logical message tree in which you can store information while the message flow processes the message.

Related tasks:

"Creating PHP code for a PHPCompute node" on page 2672
Use these instructions to create your PHP code and associate it with your PHPCompute node.

"Using PHP" on page 2670
You can use the PHP scripting language for message routing and transformation.

Related reference:

"PHPCompute node" on page 4639
Use the PHPCompute node to route and transform an incoming message, using the PHP scripting language.

"PHP API" on page 5313
The WebSphere Message Broker PHP API provides classes to represent the message, the message assembly, and the element tree.

@MessageBrokerSimpleTransform:

Use the @MessageBrokerSimpleTransform annotation to alter the behavior of the evaluate method in a PHP class.

The @MessageBrokerSimpleTransform annotation causes two parameters to be passed to the evaluate method. The first parameter is a reference to the output assembly, and the second parameter is a reference to the input assembly. The second parameter is optional.

If the MessageBrokerSimpleTransform and MessageBrokerCopyTransform annotations are specified together, the MessageBrokerCopyTransform annotation takes precedence.

The following example copies the subtree under element *aaa* into the output tree under element *bbb*:

```
<?php

class SimpleTransform {

/**

     * An example of MessageBrokerSimpleTransform
     * @MessageBrokerSimpleTransform

     */

    function evaluate($output_assembly, $input_assembly) {

        $output_assembly->XMLNSC->bbb = $input_assembly->XMLNSC->aaa;
    }

}

?>
```

Related concepts:

"Local environment tree structure" on page 1056
The local environment tree is a part of the logical message tree in which you can store information while the message flow processes the message.

"Using annotations" on page 2676
Annotations alter the behavior of the evaluate method when using the PHP class structure, and remove the need to repeat commonly used code (for transforming message trees) in each PHP script.

"@MessageBrokerCopyTransform"
Use the @MessageBrokerCopyTransform annotation to alter the behavior of the evaluate method in a PHP class.

"@MessageBrokerRouter" on page 2679
Use the @MessageBrokerRouter annotation to alter the behavior of the evaluate method in a PHP class.

"@MessageBrokerLocalEnvironmentTransform" on page 2680
Use the @MessageBrokerLocalEnvironmentTransform annotation to alter the behavior of the evaluate method in a PHP class.

Related tasks:

"Creating PHP code for a PHPCompute node" on page 2672
Use these instructions to create your PHP code and associate it with your PHPCompute node.

"Using PHP" on page 2670
You can use the PHP scripting language for message routing and transformation.

Related reference:

"PHPCompute node" on page 4639
Use the PHPCompute node to route and transform an incoming message, using the PHP scripting language.

@MessageBrokerCopyTransform:

Use the @MessageBrokerCopyTransform annotation to alter the behavior of the evaluate method in a PHP class.

The @MessageBrokerCopyTransform annotation causes one parameter to be passed to the evaluate method. This parameter is a reference to the output assembly with the contents of the input assembly already copied into it. The input assembly is available with the @MessageBrokerCopyTransform. If you declare a second parameter (which is optional) the input assembly is passed to it.

If the @MessageBrokerCopy Transform and @MessageBrokerSimpleTransform annotations are specified together, the @MessageBrokerCopyTransform annotation takes precedence.

The following example modifies the original XML message by adding an element called *Greeting* with the value *Hello World*:

```
<?php

class CopyTest {

    /**

     * An example of MessageBrokerCopyTransform

     *

     * @MessageBrokerCopyTransform

     */

    function evaluate($assembly) {
        $assembly->XMLNSC->doc->Greeting = "Hello World";
    }
```

```
}

?>
```

Related concepts:

"Local environment tree structure" on page 1056
The local environment tree is a part of the logical message tree in which you can store information while the message flow processes the message.

"Using annotations" on page 2676
Annotations alter the behavior of the evaluate method when using the PHP class structure, and remove the need to repeat commonly used code (for transforming message trees) in each PHP script.

"@MessageBrokerSimpleTransform" on page 2677
Use the @MessageBrokerSimpleTransform annotation to alter the behavior of the evaluate method in a PHP class.

"@MessageBrokerRouter"
Use the @MessageBrokerRouter annotation to alter the behavior of the evaluate method in a PHP class.

"@MessageBrokerLocalEnvironmentTransform" on page 2680
Use the @MessageBrokerLocalEnvironmentTransform annotation to alter the behavior of the evaluate method in a PHP class.

Related tasks:

"Creating PHP code for a PHPCompute node" on page 2672
Use these instructions to create your PHP code and associate it with your PHPCompute node.

"Using PHP" on page 2670
You can use the PHP scripting language for message routing and transformation.

Related reference:

"PHPCompute node" on page 4639
Use the PHPCompute node to route and transform an incoming message, using the PHP scripting language.

@MessageBrokerRouter:

Use the @MessageBrokerRouter annotation to alter the behavior of the evaluate method in a PHP class.

The @MessageBrokerRouter annotation causes the return value of the evaluate method to be used to specify the terminal through which the message is to be propagated. The terminal can be either the Out terminal (defined on the node) or a dynamic terminal that you have created. You can add output terminals dynamically to your node instance in the Message Flow editor. The string that is returned from the evaluate method must match either the name of the dynamic terminal that you have defined or the Out terminal. If no return value is specified, the output assembly is not propagated to the next node after the evaluate method returns.

The following example routes the message to the Out terminal if the value of the *threshold* element is greater than 10; otherwise the message is routed to the *other* terminal:

```
<?php
class RouteTest {
```

```
    /**
     * Basic routing of a message.
     *
     * @MessageBrokerRouter
     */
    function evaluate($assembly) {
 // Simple filter
 if ($assembly->XMLNSC->doc->threshold->getValue() > 10) {
       return 'out';
 } else {
  return 'other';
  }
 }
}
?>
```

For information about dynamic terminals, see "Using dynamic terminals" on page 1518.

Related concepts:

"Local environment tree structure" on page 1056
The local environment tree is a part of the logical message tree in which you can store information while the message flow processes the message.

"Using annotations" on page 2676
Annotations alter the behavior of the evaluate method when using the PHP class structure, and remove the need to repeat commonly used code (for transforming message trees) in each PHP script.

"@MessageBrokerCopyTransform" on page 2678
Use the @MessageBrokerCopyTransform annotation to alter the behavior of the evaluate method in a PHP class.

"@MessageBrokerSimpleTransform" on page 2677
Use the @MessageBrokerSimpleTransform annotation to alter the behavior of the evaluate method in a PHP class.

"@MessageBrokerLocalEnvironmentTransform"
Use the @MessageBrokerLocalEnvironmentTransform annotation to alter the behavior of the evaluate method in a PHP class.

Related tasks:

"Creating PHP code for a PHPCompute node" on page 2672
Use these instructions to create your PHP code and associate it with your PHPCompute node.

"Using PHP" on page 2670
You can use the PHP scripting language for message routing and transformation.

Related reference:

"PHPCompute node" on page 4639
Use the PHPCompute node to route and transform an incoming message, using the PHP scripting language.

@MessageBrokerLocalEnvironmentTransform:

Use the @MessageBrokerLocalEnvironmentTransform annotation to alter the behavior of the evaluate method in a PHP class.

The @MessageBrokerLocalEnvironmentTransform is similar to the @MessageBrokerSimpleTransform annotation but creates a copy of the local environment tree in the output assembly.

If the @MessageBrokerLocalEnvironmentTransform is used, nodes downstream of the PHPCompute node see changes to the local environment. If the @MessageBrokerLocalEnvironmentTransform is not used, the node can still modify the local environment, and all nodes in the flow (including upstream nodes) can see the changes.

The following example populates two new folders in the local environment tree, and copies the *Wildcard* subtree from the local environment into the output message:

```
<?php

class LocalEnvironmentTest {


    /**

     * Test local environment messages.

     *

     * @MessageBrokerSimpleTransform

     * @MessageBrokerLocalEnvironmentTransform

     */

    function evaluate($output_assembly, $input_assembly) {



        $output_assembly[MB_LOCAL_ENVIRONMENT]->Folder1 = 'some string';

        $output_assembly[MB_LOCAL_ENVIRONMENT]->Folder2->SubFolder =
          'another string';

        $output_assembly->XMLNSC->Message->InputLocalEnvironment =
          $input_assembly[MB_LOCAL_ENVIRONMENT]->Wildcard;

    }

}


?>
```

Related concepts:

"Local environment tree structure" on page 1056
The local environment tree is a part of the logical message tree in which you can store information while the message flow processes the message.

"Using annotations" on page 2676
Annotations alter the behavior of the evaluate method when using the PHP class structure, and remove the need to repeat commonly used code (for transforming message trees) in each PHP script.

"@MessageBrokerCopyTransform" on page 2678
Use the @MessageBrokerCopyTransform annotation to alter the behavior of the evaluate method in a PHP class.

"@MessageBrokerRouter" on page 2679
Use the @MessageBrokerRouter annotation to alter the behavior of the evaluate method in a PHP class.

"@MessageBrokerSimpleTransform" on page 2677
Use the @MessageBrokerSimpleTransform annotation to alter the behavior of the

evaluate method in a PHP class.

Related tasks:

"Creating PHP code for a PHPCompute node" on page 2672
Use these instructions to create your PHP code and associate it with your PHPCompute node.

"Using PHP" on page 2670
You can use the PHP scripting language for message routing and transformation.

Related reference:

"PHPCompute node" on page 4639
Use the PHPCompute node to route and transform an incoming message, using the PHP scripting language.

Using PHP arrays with XML

PHP arrays are associative arrays (maps) but you can treat them as lists by adding values without keys.

PHP supports the following syntax for building arrays:

```
$array[] = "aaa";
$array[] = "bbb";
```

PHP allows an object to function as an array, and you can use this capability to create repeating elements in a tree. For example:

```
$output_root->XMLNSC->a->b->c[] = $input_root->XMLNSC->a->b;
$output_root->XMLNSC->a->b->c[] = $input_root->XMLNSC->a->c;
```

The code shown above creates the following elements:

```
<a>
 <b>
  <c>
   ... // contents of $input_root->XMLNSC->a->b
  </c>
  <c>
   ... // contents of $input_root->XMLNSC->a->c
  </c>
 </b>
</a>
```

You can use the array operator in the path, as shown in the following example:

```
$output_root->XMLNSC->a->b[]->c = $input_root->XMLNSC->a->b;
$output_root->XMLNSC->a->b[]->c = $input_root->XMLNSC->a->c;
```

to create the following elements:

```
<a>
 <b>
  <c>
   ... // contents of $input_root->XMLNSC->a->b
  </c>
 </b>
 <b>
  <c>
   ... // contents of $input_root->XMLNSC->a->c
  </c>
 </b>
</a>
```

The following example uses no array operators:

```
$output_root->XMLNSC->a->b->c = $input_root->XMLNSC->a->b;
$output_root->XMLNSC->a->b->c = $input_root->XMLNSC->a->c;
```

The example above produces the following XML code:

```
<a>
 <b>
  <c>
   ... // contents of $input_root->XMLNSC->a->c (overwriting previous)
  </c>
 </b>
</a>
```

You can also iterate a set of repeating elements by using a *foreach* loop, as shown in the following example:

```
foreach ($input_root->XMLNSC->doc->item as $item) {
   $output_root->XMLNSC->msg->bit[] = $this->transformItem($item);
}
```

This example builds an output message with a repeating *bit* element, one for each *item* element in the input message. The content of each *bit* element is determined by the *transformItem* function, which refers to the *item* element as its parameter. The following sample shows an example of this syntax being used for message transformation:

- PHPCompute Node

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

You can assign standard PHP arrays into an element tree as a way of building subtrees by using a very compact syntax. The following example shows how to build a repeating element from an array of values:

```
$output_root->XMLNSC->doc->item[] = array('aaa', 'bbb', 'ccc');
```

This code builds a tree with three item elements:

```
<doc>
 <item>aaa</item>
 <item>bbb</item>
 <item>ccc</item>
</doc>
```

Although the PHP array looks like a list, it is an associative array with the keys *0*, *1*, and *2*. The following example shows how to assign key/value pairs into the element tree:

```
$output_root->XMLNSC->doc->item = array('book'  => 'PHP',
                                        'fruit' => 'apple',
                                        'dog'   => 'Spaniel'   );
```

Without the [] operator on the item element, the keys in the array are used to name the child elements:

```
<doc>
 <item>
  <book>PHP</book>
  <fruit>apple</fruit>
  <dog>Spaniel</dog>
 </item>
</doc>
```

You can also nest arrays to represent more complex structures. For example:

```
output_root->XMLNSC->doc->items =
             array('book'  => array('title'  => 'PHP',
                                    'author' => 'A N Other'),
                   'fruit' => 'apple',
                   'dog'   => array('breed'  => 'Spaniel',
                                    'ears'   => 'long') );
```

The preceding example produces the following XML code:

```
<doc>
 <items>
  <book>
   <title>PHP</title>
   <author>A N Other</author>
  </book>
  <fruit>apple</fruit>
  <dog>
   <breed>Spaniel</breed>
   <ears>long</ears>
  </dog>
 </items>
</doc>
```

Related concepts:

"PHP overview" on page 2671
WebSphere Message Broker provides support for the PHP scripting language.

"MbsElement arrays"
When an array of MbsElement objects is assigned to an element tree, the array acts as an associative array.

Related tasks:

"Using PHP" on page 2670
You can use the PHP scripting language for message routing and transformation.

"Writing PHP code" on page 2673
Use these instructions to create your PHP code.

Related reference:

"PHPCompute node" on page 4639
Use the PHPCompute node to route and transform an incoming message, using the PHP scripting language.

"PHP data types" on page 5323
PHP data types are supported by WebSphere Message Broker.

MbsElement arrays:

When an array of MbsElement objects is assigned to an element tree, the array acts as an associative array.

For example:

```
$output_assembly->XMLNSC->doc->folder = $input_assembly->xpath("//item");
```

This example generates the following result:

```
<doc>
 <folder>
  <item>
   ... deep copy of 1st item element
  </item>
  <item>
   ... deep copy of 2nd item element
  </item>
  <item>
```

```
   ... deep copy of 3rd item element
  </item>
 </folder>
</doc>
```

The name of each element on the right side (in this example, *item*) becomes the name of the child element in the target tree. However, if the [] operator is used on the left side, *item* is replaced with *folder*, as shown in the following example:

```
$output_assembly->XMLNSC->doc->folder[] = $input_assembly->xpath("//item");
```

The example shown above generates the following result:

```
<doc>
 <folder>
  ... deep copy of 1st item element
 </folder>
 <folder>
  ... deep copy of 2nd item element
 </folder>
 <folder>
  ... deep copy of 3rd item element
 </folder>
</doc>
```

Related concepts:

"Using PHP arrays with XML" on page 2682
PHP arrays are associative arrays (maps) but you can treat them as lists by adding values without keys.

"Accessing elements in the message tree from a PHPCompute node" on page 2692
Access the contents of a message, for reading or writing, by using the structure and arrangement of the elements in the tree that the parser creates from the input bit stream.

Related tasks:

"Using PHP" on page 2670
You can use the PHP scripting language for message routing and transformation.

Related reference:

"PHPCompute node" on page 4639
Use the PHPCompute node to route and transform an incoming message, using the PHP scripting language.

"PHP data types" on page 5323
PHP data types are supported by WebSphere Message Broker.

"PHP API" on page 5313
The WebSphere Message Broker PHP API provides classes to represent the message, the message assembly, and the element tree.

"MbsElement" on page 5313
The MbsElement class represents a single parsed element in a message (or other logical tree).

Using PHP arrays with JSON

PHP arrays are associative maps, in which the key can be an integer or a string.

Integer keys can be allocated automatically in ascending sequential order, to work in the same way as a traditional integer indexed array. PHP supports an empty index operator ([]), which can be used on the left side of an assignment to add an item to the array, using an integer index with the value of the highest current index plus 1.

PHP associative arrays are modeled as JSON object structures, because JSON arrays are anonymous.

Values in a JSON logical tree can be set using any of the PHP native types, including arrays. The type of array determines the resulting JSON logical tree shape:

- A JSON array is produced by assigning a contiguous integer-indexed PHP array into a JSON domain tree path. A PHP array variable is contiguous integer-indexed if it contains only integer keys that are sequenced from `0` to *n*`-1` (where *n* is the total number of items in the array). The tree elements have their type set automatically to `JSON.Array`, which produces a JSON array format in the bit stream when the tree is serialized.
- A JSON object is produced by assigning an associative PHP array into a JSON domain tree path. A PHP array is associative if it contains one or more string keys, or if it has integer keys that are not contiguously sequenced from `0` to *n*`-1`.

When a multidimensional PHP array variable is assigned to a JSON domain tree path, these rules are applied to each dimension of the array variable, so that an appropriate JSON array or object is formed in the message tree to model each dimension of the PHP array.

When you create a JSON message tree in PHP using individual value assignments, you can create JSON array items by appending the target path with the PHP array append operator (`[]`) in the assignment. You can use the `[]` operator on a path in the following ways:

- To create a JSON array and add the first item into it
- To append an item to the end of an existing JSON array

If you try to create an array item using a target path with the `[]` operator, and the path resolves to an existing JSON object element in the tree, an exception is thrown.

The PHPCompute node supports the PHP constant `MB_JSON_ARRAY`, which you can use in the MbElement `setType()` method to change an existing JSON object element into a JSON array. This method is typically used when elements have been created by copying from a tree owned by another parser domain.

To access or change an existing value of a JSON array item in a tree, you can use an array index (`[index]`) to select the required item. JSON array items cannot be created in the JSON tree by using the `[index]` form; attempts to do so result in an exception being thrown.

You can also access non-array elements in the JSON domain tree, as if the tree were an associative array, by using an index key name. For example, to select an element called `street`, which is a child of an element called `address`, use the following path construct:

```
->address['street']
```

The following examples demonstrate how you can use PHP arrays with JSON:

- "Creating a JSON array message from individual PHP variables" on page 2687
- "Creating a JSON array message from a PHP indexed array" on page 2687
- "Creating a JSON object message from a PHP associative array" on page 2688
- "Creating a JSON message from a PHP integer index multidimensional array" on page 2688

- "Creating a JSON message from a PHP mixed associative and integer indexed multidimensional array" on page 2688
- "Updating a JSON array in the logical tree" on page 2689
- "Copying an XML domain message subtree with repeating elements to a JSON domain tree and converting it to a JSON array using PHP" on page 2689
- "Working with a JSON object with a repeating name" on page 2690

Creating a JSON array message from individual PHP variables

You can create a JSON array and append items to it by using a target path with an array append index operator ([]):

```
$strVar = "this";
$boolVar = true;
$output_assembly->JSON->Data[] = $strVar;
$output_assembly->JSON->Data[] = $boolVar;
```

You must use the array append index operator ([]) to append to an array; you cannot use the [index] form.

The following broker message tree is produced:

```
(Message):JSON     = ( ['json' : 0xhhhhhh]
     (0x01001000:Array     ): Data = (
       (0x03000000:NameValue): Item = 'this' (CHARACTER)
       (0x03000000:NameValue): Item = TRUE (BOOLEAN)
     )
 )
```

This message tree is serialized to the following JSON bit stream:

```
["this",true]
```

You can extend this example to build a multidimensional JSON array:

```
$output_assembly->JSON->Data[][] = '00';
$data = $output_assembly->JSON->Data;
$data[0][] = '01';
$data[][]  = '10';
$data[1][] = '11';
```

The resulting message tree is serialized to the following JSON bit stream:

```
[["00","01"],["10","11"]]
```

Creating a JSON array message from a PHP indexed array

You can create a PHP array with contiguous integer keys, and assign it to the message tree element that you want to form the JSON array:

```
$arr = array("this", "that");
$output_assembly->JSON->Data = $arr;
```

The following broker message tree is produced:

```
(Message):JSON     = ( ['json' : 0xhhhhhh]
     (0x01001000:Array     ): Data = (
       (0x03000000:NameValue): 0 = 'this' (CHARACTER)
       (0x03000000:NameValue): 1 = 'that' (CHARACTER)
     )
 )
```

This message tree is serialized to the following JSON bit stream:

```
["this", "that" ]
```

Creating a JSON object message from a PHP associative array

This example shows how to create a PHP array with named keys, and assign it to the message tree element that you want to form the JSON object:

```
$arr = array('first' => "1st", 'second' => "2nd");
$output_assembly->JSON->Data = $arr;
```

The following broker message tree is produced:

```
(Message):JSON     = ( ['json' : 0xhhhhhh]
    (0x01000000:Object  ): Data = (
      (0x03000000:NameValue): first = '1st' (CHARACTER)
      (0x03000000:NameValue): second = '2nd' (CHARACTER)
    )
)
```

You can access the array elements using the key name in the array index:

```
$var1 = $output_assembly->JSON->Data['second']; // 2nd
$var2 = $output_assembly->JSON->Data['none'];   // null
```

This is serialized to the following JSON bit stream:

```
{"first":"1st","second":"2nd"}
```

Creating a JSON message from a PHP integer index multidimensional array

You can assign the array to the element that is to hold the data. The top level and the nested level of the PHP array are integer indexed arrays, so both are created as JSON arrays, resulting in a multidimensional JSON array:

```
$arr = array(array("a1","a2"),array("b1","b2"));
$output_assembly->JSON->Data = $arr;
```

The following broker message tree is produced:

```
(Message):JSON     = ( ['json' : 0xhhhhhh]
     (0x01001000:Array     ): Data = (
       (0x01001000:Array     ): 1 = (
         (0x03000000:NameValue): 1 = 'a1' (CHARACTER)
         (0x03000000:NameValue): 2 = 'a2' (CHARACTER)
       )
       (0x01001000:Array     ): 2 = (
         (0x03000000:NameValue): 1 = 'b1' (CHARACTER)
         (0x03000000:NameValue): 2 = 'b2' (CHARACTER)
       )
     )
)
```

This message tree is serialized to the following JSON bit stream:

```
[["a1","a2"],["b1","b2"]]
```

Creating a JSON message from a PHP mixed associative and integer indexed multidimensional array

You can assign the array to the element that is to hold the data. The top level of the PHP array is associative, with string key names, and so becomes a JSON object. The nested level of the PHP array is using integer indexed arrays, and is created as a JSON array.

```
$arr = array("top1" => array("a1","a2"), "top2" => array("b1","b2"));
$output_assembly->JSON->Data = $arr;
```

The following broker message tree is produced from the example shown above:

```
 (Message):JSON     = ( ['json' : 0xhhhhhh]
      (0x01000000:Object     ): Data = (
        (0x01001000:Array     ):top1 = (
          (0x03000000:NameValue): 1 = 'a1' (CHARACTER)
          (0x03000000:NameValue): 2 = 'a2' (CHARACTER)
        )
        (0x01001000:Array     ):top2 = (
          (0x03000000:NameValue): 1 = 'b1' (CHARACTER)
          (0x03000000:NameValue): 2 = 'b2' (CHARACTER)
        )
      )
  )
```

This message tree is serialized to the following JSON bit stream:

```
{"top1":["a1","a2"],"top2":["b1","b2"]}
```

Updating a JSON array in the logical tree

```
[["a1","a2"],["b1","b2"]]
```

The following broker message tree is produced from the JSON input shown above:

```
 (Message):JSON     = ( ['json' : 0xhhhhhh]
      (0x01001000:Array     ): Data = (
        (0x01001000:Array     ): Item = (
          (0x03000000:NameValue): Item = 'a1' (CHARACTER)
          (0x03000000:NameValue): Item = 'a2' (CHARACTER)
        )
        (0x01001000:Array     ): Item = (
          (0x03000000:NameValue): Item = 'b1' (CHARACTER)
          (0x03000000:NameValue): Item = 'b2' (CHARACTER)
        )
      )
  )
```

You can update this message with PHP, using `[index]` on the element that is holding the array to access existing elements, and the `[]` form to create additional elements:

```
output_assembly->JSON->Data[0][0] = "A1";
output_assembly->JSON->Data[1][1] = "B2";
output_assembly->JSON->Data[1][] = "New3";
```

This example produces the following JSON output:

```
[["A1","a2"],["b1","B2","new3"]]
```

Copying an XML domain message subtree with repeating elements to a JSON domain tree and converting it to a JSON array using PHP

You can use the PHP constant MB_JSON_ARRAY to explicitly test and set the Array type of a message tree element. For example, you can use MB_JSON_ARRAY if you are copying message tree data from an XML domain to a JSON domain, or from a JSON domain to an XML domain. The following XML input contains repeating elements:

```
<doc>
      <cats>
        <cat>thing1</cat>
        <cat>thing2</cat>
      </cats>
</doc>
```

The following broker message tree is produced by the XMLNSC parser:

```
(0x01000000:Folder):XMLNSC     = ( ['xmlnsc' : 0xhhhhhh]
      (0x01001000:Folder     ): doc = (
        (0x01001000:Folder     ): cats = (
          (0x03000000:NameValue): cat = 'thing1' (CHARACTER)
          (0x03000000:NameValue): cat = 'thing2' (CHARACTER)
        )
      )
 )
```

The message flow can convert this broker message tree into a JSON message, to be serialized in the following way:

```
{"cats":["thing1","thing2"]}
```

You can transform the message using PHP, as shown in the following example:

```
$output_assembly->JSON->Data = $input_assembly->XMLNSC->doc;
$output_assembly->JSON->Data->cats->setType(MB_JSON_ARRAY);
```

This transformation produces the following message tree, which serializes to JSON format:

```
(Message):JSON     = ( ['json' : 0xhhhhhh]
    (0x01000000:Obejct):Data     = (
        (0x01001000:Array     ): cats = (
          (0x03000000:NameValue): cat = 'thing1' (CHARACTER)
          (0x03000000:NameValue): cat = 'thing2' (CHARACTER)
        )
     )
 )
```

Working with a JSON object with a repeating name

The JSON specification states that JSON objects should not have duplicate names. However, it is possible for a message to contain this format; for example, in the following JSON message the name `cat` is repeated:

```
{"cat":"thing1","cat":"thing2" }
```

The JSON parser produces the following broker message tree from the JSON input shown above:

```
 (Message):JSON     = ( ['json' : 0xhhhhhh]
      (0x01001000:Object     ): Data = (
        (0x03000000:NameValue): cat = 'thing1' (CHARACTER)
        (0x03000000:NameValue): cat = 'thing2' (CHARACTER)
      )
  )
```

This is an invalid PHP data structure because it has a duplicate key, which cannot exist in an associative array. As a result, this form of JSON data cannot be accessed, created, or modified from a PHPCompute node. To process this type of JSON data, the message flow must use either ESQL or Java.

Related concepts:

"PHP overview" on page 2671
WebSphere Message Broker provides support for the PHP scripting language.

"MbsElement arrays" on page 2684
When an array of MbsElement objects is assigned to an element tree, the array acts as an associative array.

Related tasks:

"Using PHP" on page 2670
You can use the PHP scripting language for message routing and transformation.

"Writing PHP code" on page 2673
Use these instructions to create your PHP code.

Related reference:

"PHPCompute node" on page 4639
Use the PHPCompute node to route and transform an incoming message, using the PHP scripting language.

"PHP data types" on page 5323
PHP data types are supported by WebSphere Message Broker.

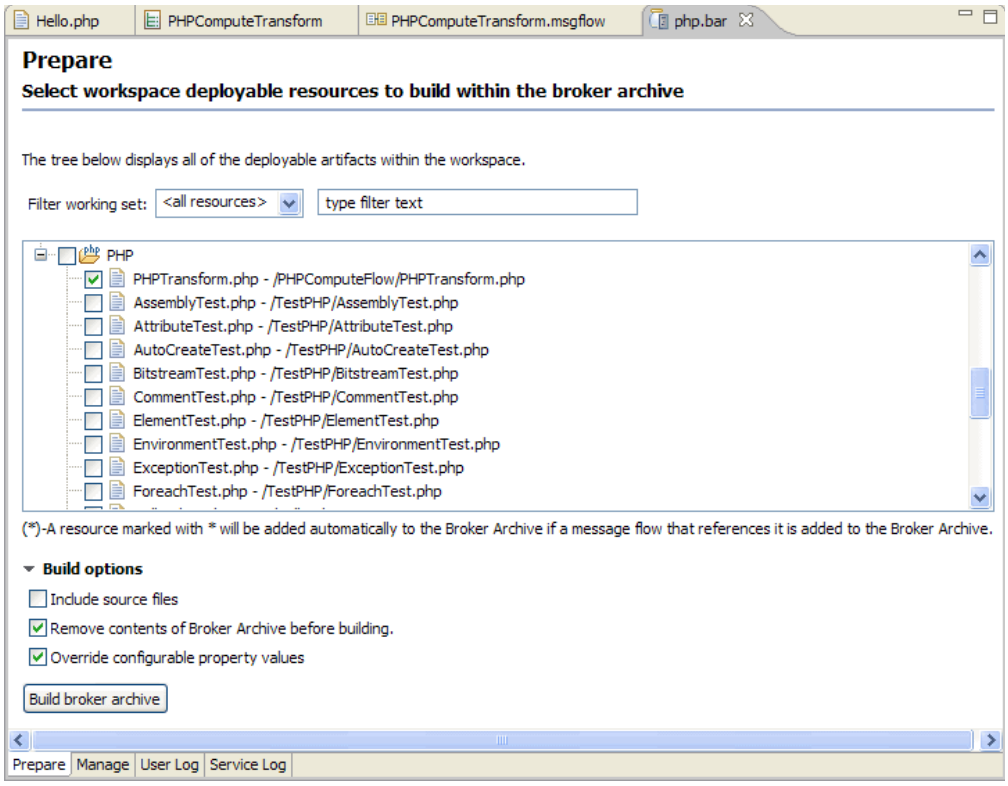
Deploying code in a PHPCompute node

The WebSphere Message Broker Toolkit deploys the PHPCompute node code automatically.

About this task

When you create a broker archive (BAR) file and add the message flow, the WebSphere Message Broker Toolkit packages the PHP code and its dependencies into the BAR file.

PHP files referenced by a PHPCompute node are added to the BAR file automatically when it is built, but you can also add additional PHP files to the BAR file by using the Broker Archive editor:



Related concepts:

"PHP overview" on page 2671
WebSphere Message Broker provides support for the PHP scripting language.

Related tasks:

"Using PHP" on page 2670
You can use the PHP scripting language for message routing and transformation.

Related reference:

"PHPCompute node" on page 4639
Use the PHPCompute node to route and transform an incoming message, using the PHP scripting language.

"Broker Archive editor" on page 6794
Use the Broker Archive editor to create and manage broker archive (BAR) files.

Accessing elements in the message tree from a PHPCompute node

Access the contents of a message, for reading or writing, by using the structure and arrangement of the elements in the tree that the parser creates from the input bit stream.

You can extract information from a message by using the PHPCompute node path syntax, or by using the MbsElement API methods. XPath 1.0 is supported as a means of accessing parts of the message.

Follow the relevant parent and child relationships from the top of the tree downwards until you reach the required element.

The following topics provide more information about accessing, extracting, and updating information in the message tree:

- "Traversing the element tree" on page 2693
- "Using SPL iterators with PHP" on page 2695
- "Accessing information about an element" on page 2701

Related concepts:

"PHP overview" on page 2671
WebSphere Message Broker provides support for the PHP scripting language.

"Using SPL iterators with PHP" on page 2695
The PHPCompute node provides support for the Standard PHP Library (SPL) syntax, which provides iterators that can be used in PHP code.

"Accessing information about an element" on page 2701
Use PHP methods to return information about an element.

Related tasks:

"Using PHP" on page 2670
You can use the PHP scripting language for message routing and transformation.

"Traversing the element tree" on page 2693
Use PHP methods to access element trees.

"Accessing other parts of the message tree using the PHPCompute node" on page 2710
You can use the PHPCompute node to access headers, broker properties, user-defined properties, the local environment, and the global environment.

"Using XPath" on page 1506
XPath provides an alternative method to ESQL for entering expressions in the property fields of specific built-in nodes.

Related reference:

"PHPCompute node" on page 4639
Use the PHPCompute node to route and transform an incoming message, using the PHP scripting language.

"PHP API" on page 5313
The WebSphere Message Broker PHP API provides classes to represent the message, the message assembly, and the element tree.

“PHP data types” on page 5323
PHP data types are supported by WebSphere Message Broker.

Traversing the element tree:

Use PHP methods to access element trees.

About this task

Use the following statements to traverse a message tree from an element:

getParent()
Returns the parent of the current element.

getPreviousSibling()
Returns the previous sibling of the current element.

getNextSibling()
Returns the next sibling of the current element.

getChild()
Returns the first child of the current element, whose name is given by the first parameter. The n^{th} occurrence of that child element can be returned by specifying the second optional parameter.

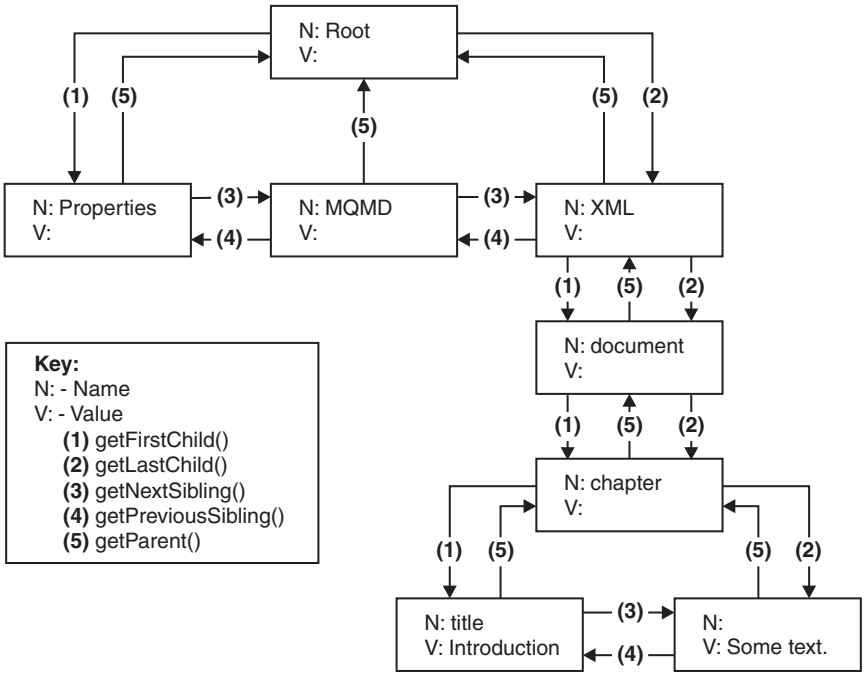
getChildren()
Returns all the child elements of the current element as an array of MbsElements. If the **namespace** parameter is specified, the array contains only the child elements with that namespace URI.

getFirstChild()
Returns the first child of the current element.

getLastChild()
Returns the last child of the current element.

The following example shows a simple XML message and the logical tree that is created from the message. The message has been sent by using WebSphere MQ. The logical tree diagram also shows the methods to call to navigate around the tree:

```
<document>
  <chapter title='Introduction'>
    Some text
  </chapter>
</document>
```



The tree used in this diagram is the one that is created by parsing the XML example given in this topic.

- From the Root part of the tree, calling **getFirstChild()** navigates to Properties. Also from Root, calling **getLastChild()** returns XML.
- From Properties, calling **getParent()** returns Root, and calling **getNextSibling()** returns MQMD.
- From MQMD, calling **getPreviousSibling()** returns Properties, calling **getParent()** returns Root, and calling **getNextSibling()** returns XML.
- From XML, calling **getPreviousSibling()** returns MQMD, calling **getParent()** returns Root, calling **getFirstChild()** returns document, and calling **getLastChild()** also returns document.
- From document, calling **getParent()** returns XML, calling **getFirstChild()** returns chapter, and calling **getLastChild()** also returns chapter.
- From chapter, calling **getParent()** returns document, calling **getFirstChild()** returns title, and calling **getLastChild()** returns the child that contains the message data "Some text.".

The following example shows how to use the MbsElement methods to navigate to the chapter element:

```
$chapter = $input_assembly->getLastChild()->getFirstChild()->getFirstChild();
```

The following example shows how to navigate to the chapter element by using the path syntax:

```
$chapter = $input_assembly->XML->document->chapter;
```

Related concepts:

"PHP overview" on page 2671
WebSphere Message Broker provides support for the PHP scripting language.

"Accessing information about an element" on page 2701
Use PHP methods to return information about an element.

"Accessing elements in the message tree from a PHPCompute node" on page 2692
Access the contents of a message, for reading or writing, by using the structure and arrangement of the elements in the tree that the parser creates from the input bit stream.

Related tasks:

"Using PHP" on page 2670
You can use the PHP scripting language for message routing and transformation.

"Accessing other parts of the message tree using the PHPCompute node" on page 2710
You can use the PHPCompute node to access headers, broker properties, user-defined properties, the local environment, and the global environment.

Related reference:

"PHPCompute node" on page 4639
Use the PHPCompute node to route and transform an incoming message, using the PHP scripting language.

"PHP API" on page 5313
The WebSphere Message Broker PHP API provides classes to represent the message, the message assembly, and the element tree.

"MbsElement" on page 5313
The MbsElement class represents a single parsed element in a message (or other logical tree).

Using SPL iterators with PHP:

The PHPCompute node provides support for the Standard PHP Library (SPL) syntax, which provides iterators that can be used in PHP code.

SPL iterators are classes that allow you to traverse a collection of data, processing each element of the collection in a specific way. A collection might be a simple serial list of elements, or a complex tree structure.

An SPL iterator, called MbsElementIterator, enables you to iterate over WebSphere Message Broker messages, and provides support for filtering and recursion.

For more information about the MbsElementIterator class, see "MbsElementIterator" on page 5316.

The following topics provide more information about iterating over elements in the message tree:

- "Iterating over elements" on page 2696
- "Recursive iterators" on page 2697
- "Iterating with a filter" on page 2698

Related concepts:

"PHP overview" on page 2671
WebSphere Message Broker provides support for the PHP scripting language.

"Accessing information about an element" on page 2701
Use PHP methods to return information about an element.

Related tasks:

"Using PHP" on page 2670
You can use the PHP scripting language for message routing and transformation.

"Traversing the element tree" on page 2693
Use PHP methods to access element trees.

"Accessing other parts of the message tree using the PHPCompute node" on page 2710
You can use the PHPCompute node to access headers, broker properties, user-defined properties, the local environment, and the global environment.

"Using XPath" on page 1506
XPath provides an alternative method to ESQL for entering expressions in the property fields of specific built-in nodes.

Related reference:

"MbsElementIterator" on page 5316
The MbsElementIterator implements the SPL RecursiveIterator.

"PHPCompute node" on page 4639
Use the PHPCompute node to route and transform an incoming message, using the PHP scripting language.

"PHP API" on page 5313
The WebSphere Message Broker PHP API provides classes to represent the message, the message assembly, and the element tree.

"PHP data types" on page 5323
PHP data types are supported by WebSphere Message Broker.

Iterating over elements:

Use the MbsElementIterator class to provide sequential iteration over elements in the message tree.

You can use an MbsElementIterator in a `foreach()` statement to iterate over the children of an MbsElement. The `foreach()` statement automatically calls methods on the iterator object to step through all the elements.

The following example shows how you can use MbsElementIterator to iterate over elements:

```
$it = new MbsElementIterator($input_assembly->XMLNSC->getFirstChild());

echo "\nIterating using an MbsElementIterator in a foreach loop\n";
foreach($it as $key=>$value)
{
  echo $key.'='.$value."\n";
}
```

For more information about the MbsElementIterator class, see "MbsElementIterator" on page 5316.

Related concepts:

"Recursive iterators" on page 2697
You can use a RecursiveIteratorIterator to iterate over a whole message tree, by using it to wrap around an MbsElementIterator.

"Iterating with a filter" on page 2698
Use a FilterIterator to filter elements in the message tree.

"Using SPL iterators with PHP" on page 2695
The PHPCompute node provides support for the Standard PHP Library (SPL) syntax, which provides iterators that can be used in PHP code.

"PHP overview" on page 2671
WebSphere Message Broker provides support for the PHP scripting language.

"Accessing information about an element" on page 2701
Use PHP methods to return information about an element.

Related tasks:

"Using PHP" on page 2670
You can use the PHP scripting language for message routing and transformation.

"Traversing the element tree" on page 2693
Use PHP methods to access element trees.

"Accessing other parts of the message tree using the PHPCompute node" on page 2710
You can use the PHPCompute node to access headers, broker properties, user-defined properties, the local environment, and the global environment.

"Using XPath" on page 1506
XPath provides an alternative method to ESQL for entering expressions in the property fields of specific built-in nodes.

Related reference:

"MbsElementIterator" on page 5316
The MbsElementIterator implements the SPL RecursiveIterator.

"PHPCompute node" on page 4639
Use the PHPCompute node to route and transform an incoming message, using the PHP scripting language.

"PHP API" on page 5313
The WebSphere Message Broker PHP API provides classes to represent the message, the message assembly, and the element tree.

"PHP data types" on page 5323
PHP data types are supported by WebSphere Message Broker.

Recursive iterators:

You can use a RecursiveIteratorIterator to iterate over a whole message tree, by using it to wrap around an MbsElementIterator.

The following example creates an MbsElementIterator and wraps it in a RecursiveIteratorIterator:

```
$it  = new MbsElementIterator($input_assembly->XMLNSC->getFirstChild());
$rii = new RecursiveIteratorIterator($it, RecursiveIteratorIterator::CHILD_FIRST);

foreach($rii as $key=>$value)
{
  echo $key.'='.$value."\n";
}
```

For more information about the MbsElementIterator class, see "MbsElementIterator" on page 5316.

Related concepts:

"Iterating over elements" on page 2696
Use the MbsElementIterator class to provide sequential iteration over elements in the message tree.

"Iterating with a filter" on page 2698
Use a FilterIterator to filter elements in the message tree.

"Using SPL iterators with PHP" on page 2695
The PHPCompute node provides support for the Standard PHP Library (SPL) syntax, which provides iterators that can be used in PHP code.

"PHP overview" on page 2671
WebSphere Message Broker provides support for the PHP scripting language.

"Accessing information about an element" on page 2701
Use PHP methods to return information about an element.

Related tasks:

"Using PHP" on page 2670
You can use the PHP scripting language for message routing and transformation.

"Traversing the element tree" on page 2693
Use PHP methods to access element trees.

"Accessing other parts of the message tree using the PHPCompute node" on page 2710
You can use the PHPCompute node to access headers, broker properties, user-defined properties, the local environment, and the global environment.

"Using XPath" on page 1506
XPath provides an alternative method to ESQL for entering expressions in the property fields of specific built-in nodes.

Related reference:

"MbsElementIterator" on page 5316
The MbsElementIterator implements the SPL RecursiveIterator.

"PHPCompute node" on page 4639
Use the PHPCompute node to route and transform an incoming message, using the PHP scripting language.

"PHP API" on page 5313
The WebSphere Message Broker PHP API provides classes to represent the message, the message assembly, and the element tree.

"PHP data types" on page 5323
PHP data types are supported by WebSphere Message Broker.

Iterating with a filter:

Use a FilterIterator to filter elements in the message tree.

The following example shows how to use a FilterIterator to iterate over a message tree, extracting only the *Toughened Steel* components:

```
class ToughenedSteelPartsIterator extends FilterIterator
{
  public function __construct(Iterator $it)
  {
    parent::__construct($it);
  }

  function accept()
  {
    /*** only check components ***/
    if (!strcmp($this->key(),"component"))
    {
      /*** Return true for Toughened Steel ***/
      if (!strcmp($this->current()->material,"Toughened Steel"))
      {
        return true;
      }
    }
    return false;
  }
}

class PhpComputeNode_Iterator_Filter
{
  /**
   * @MessageBrokerSimpleTransform
   */
  function evaluate($output_assembly, $input_assembly)
```

```
  {
    ob_start();
    echo "\nIterating over parts which have a material name of Toughened Steel\n";

    $it  = new MbsElementIterator($input_assembly->XMLNSC->getFirstChild());
    $rii = new ToughenedSteelPartsIterator(new RecursiveIteratorIterator($it, RecursiveIteratorIte

    foreach($rii as $key=>$value)
    {
      echo $key.' = '.$value->name." ".$value->price." ".$value->material."\n";
    }

    $output_assembly->XMLNSC->results->contents = ob_get_contents();
    ob_end_clean();
  }
}
```

The following message is used as input to the FilterIterator shown above:

```
<machine>
 <assembly name="Engine Block Top End">
  <assembly name="Cylinder Head">
   <assembly name="Tappet Assembly">
     <component name="Tappet" price="3.50" material="Toughened Steel"/>
     <component name="Coil String" price="1.50" material="Spring Steel"/>
     <component name="Circlip" price="1.50"  material="Spring Steel"/>
     <component name="Circlip" price="1.50"  material="Spring Steel"/>
     <component name="Seat" price="3.50" material="Toughened Steel"/>
   </assembly>
   <assembly name="Cam Shaft">
     <component name="Shaft" price="3.50" material="Mild Steel"/>
     <component name="Cam" price="1.50" material="Toughened Steel" />
     <component name="Cam" price="1.50" material="Toughened Steel" />
     <component name="Cam" price="1.50" material="Toughened Steel" />
     <component name="Cam" price="1.50" material="Toughened Steel" />
     <component name="Bearing" price="1.00" material="Toughened Steel" />
     <component name="Bearing" price="1.00" material="Toughened Steel" />
     <component name="Cam Chain Sprocket" price="4.20" material="Toughened Steel" />
     <component name="Cam Chain" price="4.20" material="Toughened Steel" />
     <component name="Tensioner" price="0.50" material="Plastic" />
   </assembly>
   <component name="Head" price="12.40" material="Alloy" />
   <component name="Plug" price="3.00"  material="Bakelite/Steel" />
  </assembly>
  <assembly name="Cylinder Block">
   <component name="Cylinder" price="20.00"  material="Alloy" />
   <component name="Sleeve" price="10.00"  material="Toughened Steel" />
  </assembly>
  <assembly name="Carburetor">
  <assembly name="Piston Assembly"/>
   <component name="Piston" price="20.00" material="Toughened Steel" />
   <component name="Ring" price="10.00" material="Spring Steel" />
   <component name="Ring" price="10.00"  material="Spring Steel" />
   <component name="Con Rod" price="10.00" material="Toughened Steel" />
  </assembly>
 </assembly>
</machine>
```

The resulting output shows only the parts whose material is *Toughened Steel*:

```
<results><contents>
Iterating Over parts which have a material name of Toughened Steel
component = Tappet 3.50 Toughened Steel
component = Seat 3.50 Toughened Steel
component = Cam 1.50 Toughened Steel
component = Cam 1.50 Toughened Steel
component = Cam 1.50 Toughened Steel
component = Cam 1.50 Toughened Steel
```

```
component = Bearing 1.00 Toughened Steel
component = Bearing 1.00 Toughened Steel
component = Cam Chain Sprocket 4.20 Toughened Steel
component = Cam Chain 4.20 Toughened Steel
component = Sleeve 10.00 Toughened Steel
component = Piston 20.00 Toughened Steel
component = Con Rod 10.00 Toughened Steel
</contents></results>
```

For more information about the MbsElementIterator class, see "MbsElementIterator" on page 5316.

Related concepts:

"Iterating over elements" on page 2696
Use the MbsElementIterator class to provide sequential iteration over elements in the message tree.

"Recursive iterators" on page 2697
You can use a RecursiveIteratorIterator to iterate over a whole message tree, by using it to wrap around an MbsElementIterator.

"Using SPL iterators with PHP" on page 2695
The PHPCompute node provides support for the Standard PHP Library (SPL) syntax, which provides iterators that can be used in PHP code.

"PHP overview" on page 2671
WebSphere Message Broker provides support for the PHP scripting language.

"Accessing information about an element" on page 2701
Use PHP methods to return information about an element.

Related tasks:

"Using PHP" on page 2670
You can use the PHP scripting language for message routing and transformation.

"Traversing the element tree" on page 2693
Use PHP methods to access element trees.

"Accessing other parts of the message tree using the PHPCompute node" on page 2710
You can use the PHPCompute node to access headers, broker properties, user-defined properties, the local environment, and the global environment.

"Using XPath" on page 1506
XPath provides an alternative method to ESQL for entering expressions in the property fields of specific built-in nodes.

Related reference:

"PHPCompute node" on page 4639
Use the PHPCompute node to route and transform an incoming message, using the PHP scripting language.

"MbsElementIterator" on page 5316
The MbsElementIterator implements the SPL RecursiveIterator.

"PHP API" on page 5313
The WebSphere Message Broker PHP API provides classes to represent the message, the message assembly, and the element tree.

"PHP data types" on page 5323
PHP data types are supported by WebSphere Message Broker.

Accessing information about an element:

Use PHP methods to return information about an element.

Use the following methods to return information about the referenced element:

getName()
Returns the name of the current element

getValue()
Returns the value of the current element

getType()
Returns the specific type of the current element. For a full list of type values, see:

- "PHP constants for type values" on page 5343

getNamespace()
Returns the namespace URI of the current element

Related concepts:

"PHP overview" on page 2671
WebSphere Message Broker provides support for the PHP scripting language.

"Accessing elements in the message tree from a PHPCompute node" on page 2692
Access the contents of a message, for reading or writing, by using the structure and arrangement of the elements in the tree that the parser creates from the input bit stream.

Related tasks:

"Using PHP" on page 2670
You can use the PHP scripting language for message routing and transformation.

"Traversing the element tree" on page 2693
Use PHP methods to access element trees.

"Accessing other parts of the message tree using the PHPCompute node" on page 2710
You can use the PHPCompute node to access headers, broker properties, user-defined properties, the local environment, and the global environment.

Related reference:

"PHPCompute node" on page 4639
Use the PHPCompute node to route and transform an incoming message, using the PHP scripting language.

"PHP constants for type values" on page 5343
You can use type values to create syntax elements in your message tree.

"PHP API" on page 5313
The WebSphere Message Broker PHP API provides classes to represent the message, the message assembly, and the element tree.

"MbsElement" on page 5313
The MbsElement class represents a single parsed element in a message (or other logical tree).

Creating and transforming messages using a PHPCompute node

You can use the PHPCompute node to create and copy messages, and to create and manipulate message elements.

About this task

These topics describe how to transform messages by using a PHPCompute node:

- "Creating a new message using a PHPCompute node"
- "Copying a message using a PHPCompute node" on page 2703
- "Setting and moving message elements using a PHPCompute node" on page 2704
- "Creating new elements using a PHPCompute node" on page 2706

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

Related reference:

"PHPCompute node" on page 4639
Use the PHPCompute node to route and transform an incoming message, using the PHP scripting language.

Creating a new message using a PHPCompute node:

Create a new output message by using the PHPCompute node.

About this task

When you use a PHPCompute node to create a new output message, the code required depends on whether the annotated evaluate method is defined in the PHP script.

If you use the @MessageBrokerSimpleTransform annotation, the new output message and message assembly objects are created automatically. For example:

```
<?php

class MyNode {

 /**
  * @MessageBrokerSimpleTransform
  */
 function evaluate($output_assembly, $input_assembly) {
  // $output_assembly refers to the new message
 }
}

?>
```

If you use a plain script without an annotated evaluate method, you must create the output message and message assembly objects explicitly:

```
<?php

// the output message must be created explicitly
$output_message = new MbsMessage();

// a new output message assembly must be created to hold this new message
$output_assembly = new MbsMessageAssembly($assembly, $output_message);

?>
```

Related concepts:

"PHP overview" on page 2671
WebSphere Message Broker provides support for the PHP scripting language.

"Accessing elements in the message tree from a PHPCompute node" on page 2692
Access the contents of a message, for reading or writing, by using the structure and arrangement of the elements in the tree that the parser creates from the input bit stream.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Using PHP" on page 2670
You can use the PHP scripting language for message routing and transformation.

"Creating and transforming messages using a PHPCompute node" on page 2701
You can use the PHPCompute node to create and copy messages, and to create and manipulate message elements.

"Creating a new message using a PHPCompute node" on page 2702
Create a new output message by using the PHPCompute node.

"Copying a message using a PHPCompute node"
Copy an existing message using the PHPCompute node.

"Creating new elements using a PHPCompute node" on page 2706
Use the PHPCompute node to create new elements in a message tree.

Related reference:

"PHPCompute node" on page 4639
Use the PHPCompute node to route and transform an incoming message, using the PHP scripting language.

Copying a message using a PHPCompute node:

Copy an existing message using the PHPCompute node.

About this task

When you use a PHPCompute node to copy an existing message, the code required depends on whether or not the annotated evaluate method is defined in the PHP script.

If you use the @MessageBrokerCopyTransform annotation, the new output message and message assembly objects are created automatically. For example:

```
<?php

class MyNode {

 /**
```

```
  * @MessageBrokerCopyTransform
  */
 function evaluate($output_assembly, $input_assembly) {
  // $output_assembly refers to the new message
 }
}

?>
```

If you use a plain script without an annotated evaluate method, you must create the output message and message assembly objects explicitly:

```
<?php

// create a copy of the input message
$output_message = new MbsMessage($assembly[MB_MESSAGE]);

// a new output message assembly must be created to hold this new message
$output_assembly = new MbsMessageAssembly($assembly, $output_message);

?>
```

Related concepts:

"PHP overview" on page 2671
WebSphere Message Broker provides support for the PHP scripting language.

"Accessing elements in the message tree from a PHPCompute node" on page 2692
Access the contents of a message, for reading or writing, by using the structure and arrangement of the elements in the tree that the parser creates from the input bit stream.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Using PHP" on page 2670
You can use the PHP scripting language for message routing and transformation.

"Creating and transforming messages using a PHPCompute node" on page 2701
You can use the PHPCompute node to create and copy messages, and to create and manipulate message elements.

"Creating a new message using a PHPCompute node" on page 2702
Create a new output message by using the PHPCompute node.

"Setting and moving message elements using a PHPCompute node"
You can transform elements in the message as it passes through a PHPCompute node in the message flow.

"Creating new elements using a PHPCompute node" on page 2706
Use the PHPCompute node to create new elements in a message tree.

Related reference:

"PHPCompute node" on page 4639
Use the PHPCompute node to route and transform an incoming message, using the PHP scripting language.

Setting and moving message elements using a PHPCompute node:

You can transform elements in the message as it passes through a PHPCompute node in the message flow.

About this task

The following sections show the methods that you can use to modify, move, and remove elements:

- "Setting information about an element"
- "Moving elements"
- "Removing elements"

The "PHP API" on page 5313 reference information provides details about each of the methods used in the following sections.

Setting information about an element:
About this task

Use these methods to set information about the referenced element:

setName()
Sets the name of the current element

setValue()
Sets the value of the current element

setType()
Sets the specific type of the current element. For a full list of type values, see:

- "PHP constants for type values" on page 5343

setNamespace()
Sets the namespace URI of the current element

You can also set the value of an element by using the assignment operator. For example, `$element = 'text';` is equivalent to `$element.setValue('text');`.

Moving elements:
About this task

Use a PHPCompute node to copy or detach an element from a message tree by using the following methods:

`detach()`
Detaches the current element from the tree

`detachAllChildren()`
Detaches all children of the current element from the tree

Use one of the following methods to attach an element or subtree that you have copied on to another tree:

`addElement(`*element*`)`
Adds an element as the last child (by default) of the current element

`addAttribute(`*attribute*`)`
Adds an attribute to the current element

Removing elements:
About this task

Use these methods to remove elements from the message tree:

detach()
Detaches the current element from the tree

detachAllChildren()
Detaches all children of the current element from the tree

You can also remove elements from the message tree by using the PHP unset() function. For example:

```
unset($output_assembly->XMLNSC->doc->folder->item);
$output_assembly->XMLNSC->doc->folder->item->detach();
```

Related concepts:

"PHP overview" on page 2671
WebSphere Message Broker provides support for the PHP scripting language.

"Accessing elements in the message tree from a PHPCompute node" on page 2692
Access the contents of a message, for reading or writing, by using the structure and arrangement of the elements in the tree that the parser creates from the input bit stream.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Using PHP" on page 2670
You can use the PHP scripting language for message routing and transformation.

"Creating and transforming messages using a PHPCompute node" on page 2701
You can use the PHPCompute node to create and copy messages, and to create and manipulate message elements.

"Creating a new message using a PHPCompute node" on page 2702
Create a new output message by using the PHPCompute node.

"Copying a message using a PHPCompute node" on page 2703
Copy an existing message using the PHPCompute node.

"Creating new elements using a PHPCompute node"
Use the PHPCompute node to create new elements in a message tree.

Related reference:

"PHPCompute node" on page 4639
Use the PHPCompute node to route and transform an incoming message, using the PHP scripting language.

"PHP constants for type values" on page 5343
You can use type values to create syntax elements in your message tree.

Creating new elements using a PHPCompute node:

Use the PHPCompute node to create new elements in a message tree.

About this task

You can create new elements in a message tree in several ways:

- By using the MbsElement addElement method. By default this creates an element as the last child of the current element. This method has the following parameters:
  - **name**
  - **value**
  - **namespace**

- **type** (optional)
- **position** (optional)

The **type** parameter is the parser-specific type of the new node, which defaults to the XML element type for XML parsers. For a full list of values for the **type** parameter, see:

- "PHP constants for type values" on page 5343

The **position** parameter can have one of the following values:
- *MB_FIRST_CHILD*
- *MB_LAST_CHILD*
- *MB_NEXT_SIBLING*
- *MB_PREVIOUS_SIBLING*

- By using the path syntax. Elements on the left side of an assignment expression are created (if they do not exist already) when they are referenced in a path. For example, the following code navigates the elements in the XMLNSC tree, creating them if necessary:

```
$output_assembly->XMLNSC->doc->folder->item = 'book';
```

- By using an MbsElement constructor. To create a PHP function that creates and returns a subtree (part of a message), you can instantiate an element, build extra elements (using either of the previous two methods described), and return the result. You can then assign the result into the output message. For example:

```
$output_assembly->XMLNSC->doc->part = create_subtree();

function create_subtree() {
 $element = new MbsElement();
 $element->folder->item = 'some value';
 return $element;
}
```

You can also use the MbsElement addElementFromBitstream method to create an element tree from a string containing an unparsed bit stream. Use this method to defer until run time the decision about which parser to use.

Related concepts:

"PHP overview" on page 2671
WebSphere Message Broker provides support for the PHP scripting language.

"Accessing elements in the message tree from a PHPCompute node" on page 2692
Access the contents of a message, for reading or writing, by using the structure and arrangement of the elements in the tree that the parser creates from the input bit stream.

Related tasks:

"Using PHP" on page 2670
You can use the PHP scripting language for message routing and transformation.

"Creating and transforming messages using a PHPCompute node" on page 2701
You can use the PHPCompute node to create and copy messages, and to create and manipulate message elements.

"Creating a new message using a PHPCompute node" on page 2702
Create a new output message by using the PHPCompute node.

"Copying a message using a PHPCompute node" on page 2703
Copy an existing message using the PHPCompute node.

"Setting and moving message elements using a PHPCompute node" on page 2704
You can transform elements in the message as it passes through a PHPCompute node in the message flow.

Related reference:

"PHPCompute node" on page 4639
Use the PHPCompute node to route and transform an incoming message, using the PHP scripting language.

"PHP constants for type values" on page 5343
You can use type values to create syntax elements in your message tree.

"PHP API" on page 5313
The WebSphere Message Broker PHP API provides classes to represent the message, the message assembly, and the element tree.

XML support

The PHP capability in WebSphere Message Broker provides support for XML.

XML namespaces

The path navigation syntax in the PHPCompute node is not namespace aware. As a result, the expression shown in the following example navigates through the catalogue and entry elements regardless of the namespace URI of the elements:

```
$ref->catalogue->entry
```

If you generate an output message that requires namespace elements, set the namespace URI after you create the path:

```
$table->entry = $ref->catalogue->entry;
$table->entry->setNamespace('http://www.ibm.com/namespaceURI');
```

Alternatively, you can create the entry element by using the addElement API method:

```
$value = $ref->catalogue->entry;
$table->addElement('entry', $value, 'http://www.ibm.com/namespaceURI');
```

XML attributes

XML attributes are stored in the element tree as MbsElements with a *type* value that identifies them as attributes. The path syntax supports addressing an attribute of an element, by using the array operator with the attribute name as the key; therefore, attributes function as map arrays on the element. For example, the following code returns the *name* attribute of the **folder** element:

```
$attr = $input->XMLNSC->doc->folder['name']
```

You can create attributes in a similar way; for example:

```
$output->XMLNSC->doc->folder['name'] = 'PHP';
```

This example generates the following XML code:

```
<doc>
 <folder name='PHP'/>
</doc>
```

Related concepts:

"PHP overview" on page 2671
WebSphere Message Broker provides support for the PHP scripting language.

Related tasks:

"Using PHP" on page 2670
You can use the PHP scripting language for message routing and transformation.

Related reference:

"PHPCompute node" on page 4639
Use the PHPCompute node to route and transform an incoming message, using

the PHP scripting language.

Routing a message using a PHPCompute node

Route a message by using the PHPCompute node as a filter node.

Before you begin

Before you start:

Add a "PHPCompute node" on page 4639 to your message flow.

About this task

By default, the output message assembly is propagated to the Out terminal after the evaluate method in the PHP script has been processed. However, the PHPCompute node also has dynamic output terminals, so that you can use it as a filter node by propagating a message to the appropriate terminal, based on the message content.

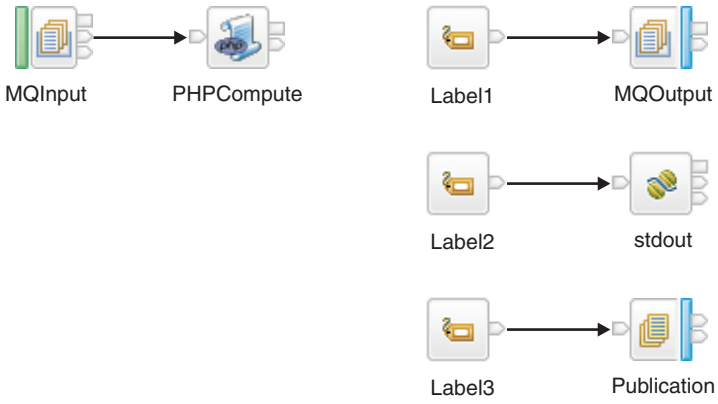
You can use the @MessageBrokerRouter annotation in your PHP code to route the message to a terminal specified by the string return value of the evaluate method. If no string is returned, the message is not propagated.

The following example shows the @MessageBrokerRouter annotation in a PHP script file:

```
/**
   * @MessageBrokerRouter
   */
  function evaluate($message) {
    if ($message->XMLNSC->doc->threshold->getValue() > 10) {
      return 'out';
    } else {
      return 'other';
    }
  }
```

For more information about using the @MessageBrokerRouter annotation for message routing, see "@MessageBrokerRouter" on page 2679.

Alternatively, you can propagate a message directly to a Label node. When you use this method, you do not need to use a RouteToLabel node, and you do not need to propagate messages to output terminals.



The following example shows the PHP code associated with the PHPCompute node in the message flow shown above. The PHP code specifies that the message is to be routed to the node called *Label2*:

```
...
$output->routeToLabel('Label2');
```

Related concepts:

"PHP overview" on page 2671
WebSphere Message Broker provides support for the PHP scripting language.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"@MessageBrokerRouter" on page 2679
Use the @MessageBrokerRouter annotation to alter the behavior of the evaluate method in a PHP class.

"Using annotations" on page 2676
Annotations alter the behavior of the evaluate method when using the PHP class structure, and remove the need to repeat commonly used code (for transforming message trees) in each PHP script.

Related tasks:

"Using PHP" on page 2670
You can use the PHP scripting language for message routing and transformation.

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"PHPCompute node" on page 4639
Use the PHPCompute node to route and transform an incoming message, using the PHP scripting language.

"Label node" on page 4569
Use the Label node to process a message that is propagated by a RouteToLabel node to dynamically determine the route that the message takes through the message flow.

Accessing other parts of the message tree using the PHPCompute node

You can use the PHPCompute node to access headers, broker properties, user-defined properties, the local environment, and the global environment.

About this task

The message tree is passed to a PHPCompute node as an argument of the evaluate method. The argument is an MbsMessageAssembly object. The message assembly contains four message objects:

- Parsed message
- LocalEnvironment
- GlobalEnvironment
- ExceptionList

The following constants are provided for accessing the different trees:

- MB_MESSAGE (default)
- MB_LOCAL_ENVIRONMENT
- MB_GLOBAL_ENVIRONMENT
- MB_EXCEPTION_LIST

The following topics describe how to access parts of the message tree:

- "Accessing headers with a PHPCompute node"
- "Updating the local environment with the PHPCompute node" on page 2713
- "Updating the global environment with the PHPCompute node" on page 2714
- "Accessing broker properties from the PHPCompute node" on page 2715
- "Accessing user-defined properties from a PHPCompute node" on page 2716

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"PHP overview" on page 2671
WebSphere Message Broker provides support for the PHP scripting language.

"Accessing elements in the message tree from a PHPCompute node" on page 2692
Access the contents of a message, for reading or writing, by using the structure and arrangement of the elements in the tree that the parser creates from the input bit stream.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Using PHP" on page 2670
You can use the PHP scripting language for message routing and transformation.

"Traversing the element tree" on page 2693
Use PHP methods to access element trees.

Related reference:

"PHPCompute node" on page 4639
Use the PHPCompute node to route and transform an incoming message, using the PHP scripting language.

Accessing headers with a PHPCompute node:

Access MQMD and MQRFH2 headers by using a PHPCompute node.

About this task

If an input node receives an input message that includes message headers that the input node recognizes, the node invokes the correct parser for each header. Parsers are supplied for most WebSphere MQ headers. This topic provides guidance for accessing the information in the MQMD and MQRFH2 headers that you can follow when accessing other headers that are present in your messages.

For more information about the contents of these and other WebSphere MQ headers for which WebSphere Message Broker provides a parser, see "Element definitions for message parsers" on page 4237.

Accessing the MQMD header:
About this task

WebSphere MQ messages include an MQMD header. You can use a PHPCompute node to refer to the fields within the MQMD, and to update them.

The following PHP code shows how to add an MQMD header to your message:

```
$output_assembly->MQMD->Version = 2;
```

Accessing the MQRFH2 header:
About this task

The following PHP code adds an MQRFH2 header to an outgoing message that is to be used to make a subscription request:

```
$output_assembly->MQRFH2->psc->Topic = 'department/sales';
```

Related concepts:

"PHP overview" on page 2671
WebSphere Message Broker provides support for the PHP scripting language.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Using PHP" on page 2670
You can use the PHP scripting language for message routing and transformation.

"Creating PHP code for a PHPCompute node" on page 2672
Use these instructions to create your PHP code and associate it with your PHPCompute node.

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

Related reference:

"The MQMD parser" on page 4249
The elements of the MQMD parser are listed in this topic.

"MQRFH2 header" on page 6397
The MQRFH2 header is used to pass messages to and from a message broker that belongs to WebSphere Message Broker.

"PHPCompute node" on page 4639
Use the PHPCompute node to route and transform an incoming message, using the PHP scripting language.

"Element definitions for message parsers" on page 4237

Updating the local environment with the PHPCompute node:

The local environment tree is part of the logical message tree in which you can store information while the message flow processes the message.

About this task

Follow these steps to update the local environment:

Procedure

1. Make a copy of the local environment to update.
2. Use the following PHP code to alter the copy of the local environment:

```
<?php

class LocalEnv {

  /**
   * @MessageBrokerLocalEnvironmentTransform
   */
  function evaluate($output_assembly, $input_assembly) {

    $output_assembly [MB_LOCAL_ENVIRONMENT] = $input_assembly->XMLNSC->Message;
    $output_assembly [MB_LOCAL_ENVIRONMENT]->Folder1 = 'some data';
    $output_assembly [MB_LOCAL_ENVIRONMENT]->Folder2->SubFolder = 'more data';
    }
}

?>
```

Related concepts:

"PHP overview" on page 2671
WebSphere Message Broker provides support for the PHP scripting language.

"Local environment tree structure" on page 1056
The local environment tree is a part of the logical message tree in which you can store information while the message flow processes the message.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Using PHP" on page 2670
You can use the PHP scripting language for message routing and transformation.

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Creating destination lists" on page 1477
Create a list of destinations to indicate where a message is sent.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

Related reference:

"PHPCompute node" on page 4639
Use the PHPCompute node to route and transform an incoming message, using the PHP scripting language.

Updating the global environment with the PHPCompute node:

Use PHP code associated with a PHPCompute to modify the global environment.

About this task

The Global Environment tree is always created when the logical tree is created for an input message. You can use this tree for your own purposes, for example to pass information from one node to another. You can use the whole tree as a scratchpad or working area.

The global environment can be altered across the message flow. The following PHP code shows how to alter the global environment:

```
<?php
class GlobalEnv {

  /**
   *    */
  function evaluate($output_assembly, $input_assembly) {

    $output_assembly [MB_GLOBAL_ENVIRONMENT] = $input_assembly->XMLNSC->Message;
    $output_assembly [MB_GLOBAL_ENVIRONMENT]->Folder1 = 'some data';
    $output_assembly [MB_GLOBAL_ENVIRONMENT]->Folder2->SubFolder = 'more data';
    }
}

?>
```

Related concepts:

"PHP overview" on page 2671
WebSphere Message Broker provides support for the PHP scripting language.

"Environment tree structure" on page 1055
The environment tree is a part of the logical message tree in which you can store information while the message passes through the message flow.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Using PHP" on page 2670
You can use the PHP scripting language for message routing and transformation.

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

Related reference:

"PHPCompute node" on page 4639
Use the PHPCompute node to route and transform an incoming message, using the PHP scripting language.

Accessing broker properties from the PHPCompute node:

For each broker, WebSphere Message Broker maintains a set of properties, and you can access some of these properties from the PHP code associated with a PHPCompute node. You can have real-time access to details of a specific node, flow, or broker.

About this task

Four categories of broker property exist, which relate to:

- A specific node
- Nodes in general
- A message flow
- An execution group

Broker properties:

- Are grouped by broker, execution group, flow, and node
- Are case sensitive. Their names always start with an uppercase letter
- Return NULL if they do not contain a value

The use of the MB_* array indexes in the $_ENV array is deprecated. Use the $_MB array instead:

Broker property	**PHP variable**
Work Path	*$_MB['WORK_PATH']*
Broker Name (label)	*$_MB['BROKER_NAME']*
Execution Group Name (label)	*$_MB['EXECUTION_GROUP_NAME']*
Node Name (label)	*$_MB['NODE_NAME']*
Message Flow Name (label)	*$_MB['MESSAGE_FLOW_NAME']*
Script Name	*$_MB['SCRIPT_NAME']*

Related concepts:

"PHP overview" on page 2671
WebSphere Message Broker provides support for the PHP scripting language.

"Broker properties" on page 1144
For each broker, WebSphere Message Broker maintains a set of properties. You can access some of these properties from your ESQL programs. A subset of the properties is also accessible from Java programs. It can be useful, at run time, to have real-time access to details of a specific node, flow, or broker.

Related tasks:

"Using PHP" on page 2670
You can use the PHP scripting language for message routing and transformation.

"Creating PHP code for a PHPCompute node" on page 2672
Use these instructions to create your PHP code and associate it with your PHPCompute node.

Related reference:

"PHPCompute node" on page 4639
Use the PHPCompute node to route and transform an incoming message, using the PHP scripting language.

Accessing user-defined properties from a PHPCompute node:

Customize a PHPCompute node to access properties that you have associated with the message flow in which the node is included.

About this task

To access these properties from a PHPCompute node, use the mb_get_user_defined_property(*name*) method, where *name* is the name of the property that you are accessing. This method returns a PHP datatype equivalent to the ESQL broker type in the property definition.

For more information about the ESQL to PHP mappings, see "PHP data types" on page 5323.

You can use the Administration API for WebSphere Message Broker (also known as the CMP API) to change the value of user-defined properties. Use the getUserDefinedPropertyNames(), getUserDefinedProperty(), and setUserDefinedProperty() methods to query, discover, and set user-defined properties, as described in "Setting message flow user-defined properties at run time in a CMP application" on page 985.

Related concepts:

"PHP overview" on page 2671
WebSphere Message Broker provides support for the PHP scripting language.

"The Administration API for WebSphere Message Broker" on page 54
The Administration API for WebSphere Message Broker is a programming interface that your applications can use to control brokers and their resources through a remote interface.

"User-defined properties" on page 1147
A user-defined property (UDP) is a property that is defined when you construct a message flow by using the Message Flow editor. This property can be used by the ESQL or Java program inside message flow nodes, such as a Compute node.

Related tasks:

"Using PHP" on page 2670
You can use the PHP scripting language for message routing and transformation.

"Setting message flow user-defined properties at run time in a CMP application" on page 985
Use the CMP API to query, discover, and set message flow user-defined properties dynamically at run time. You can use the CMP API to set properties with a data type of character.

Related reference:

"PHP data types" on page 5323
PHP data types are supported by WebSphere Message Broker.

"PHPCompute node" on page 4639
Use the PHPCompute node to route and transform an incoming message, using the PHP scripting language.

Calling Java from PHP

The IBM sMash Runtime for PHP provides access to Java classes and functionality from PHP. This Java Bridge can instantiate Java classes and call their methods.

You can manipulate values in an MRM or XMLNSC tree with xsd types that do not map directly to PHP types:

- xsd:decimal type java.math.BigDecimal
- xsd:dateTime com.ibm.broker.plugin.MbTimestamp

For example:

```
/**
* @MessageBrokerSimpleTransform
*/
function evaluate ($output, $input) {
 $number = $input->XMLNSC-doc->number->getValue();
 $signature = new JavaSignature (JAVA_STRING);
 $decimal = new Java("java.math.BigDecimal", $signature, "654.321");
 $sum = $number->add($decimal);
 $output->XMLNSC->doc->number = $sum;

 $timestamp = $input->XMLNSC->doc->date->getValue();
 $now = new Java("java.util.Date");
 $timestamp->setTime($now);
 $output->XMLNSC->doc->date = $timestamp;
}
```

Related tasks:

"Using PHP" on page 2670
You can use the PHP scripting language for message routing and transformation.

Related reference:

"PHPCompute node" on page 4639
Use the PHPCompute node to route and transform an incoming message, using the PHP scripting language.

"PHP data types" on page 5323
PHP data types are supported by WebSphere Message Broker.

Processing events

Using event driven message processing you can control the flow of messages through your message flows.

About this task

- "Using aggregation"
- "Using message collections" on page 2755
- Using message sequences
- "Configuring timeout flows" on page 2809

Related tasks:

"Connecting client applications" on page 1537
Connect your client applications to the broker by using one or more of the supported protocols from other resources and software servers in your network.

"Routing messages" on page 2209
Route messages through your message flow or the broker by using one or more of the techniques described in this section.

"Transforming and enriching messages" on page 2227
Transform and enrich messages by using one or more of the techniques described in this section.

Using aggregation

Use aggregation to generate multiple requests from a single input message, and coordinate the multiple responses to provide a single consolidated response to that input message.

About this task

Learn what aggregation is, and the advantages it can provide in your message flows. Configure your message flows to support message aggregation.

- "Message flow aggregation"
- "Configuring aggregation flows" on page 2721

Related concepts:

"Developing user-defined extensions" on page 2970
A user-defined extension is a component that you design and implement to extend the function of WebSphere Message Broker.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Routing messages" on page 2209
Route messages through your message flow or the broker by using one or more of the techniques described in this section.

"Transforming and enriching messages" on page 2227
Transform and enrich messages by using one or more of the techniques described in this section.

Message flow aggregation

Aggregation is the generation and fan-out of related requests that are derived from a single input message, and the fan-in of the corresponding replies to produce a single aggregated reply message.

The initial request that is received by the message flow, representing a collection of related request items, is split into the appropriate number of individual requests to satisfy the subtasks of the initial request. This process is known as *fan-out*, and it is provided by a message flow that includes aggregation nodes.

Replies from the subtasks are combined and merged into a single reply, which is returned to the original requester (or another target application) to indicate completion of the processing. This process is known as *fan-in*, and it is also provided by a message flow that includes aggregation nodes.

A message aggregation is initiated by a message flow that contains the AggregateControl node followed by an AggregateRequest node. The responses are aggregated back together using a flow that contains the AggregateReply node. The aggregation nodes work correctly only for transports that use a request/reply model; for example, WebSphere MQ Enterprise Transport.

WebSphere Message Broker provides three message flow nodes that support aggregation:

- The AggregateControl node
- The AggregateRequest node
- The AggregateReply node

When you include these nodes in your message flows, the multiple fan-out requests are issued in sequence from one thread in the execution group process.

If you use WebSphere MQ Enterprise Transport, the responses that are received by the fan-in flow must be valid reply messages that contain the reply identifier. You must set the reply identifier to the value of the message in the request message's message descriptor (MQMD), then store the reply identifier in the correlation

identifier field (CorrelId) of the MQMD. If the CorrelId is set to MQCI_NONE, the AggregateReply node issues an error because the reply message is not valid, and cannot be matched to a request message.

You can also use these aggregation nodes to issue requests to applications outside the broker environment. Messages can be sent asynchronously to external applications or services; the responses are retrieved from those applications, and the responses are combined to provide a single response to the original request message.

These nodes can help to improve response time because slow requests can be performed in parallel, and they do not need to follow each other sequentially. If the subtasks can be processed independently, and they do not need to be handled as part of a single unit of work, they can be processed by separate message flows.

You can design and configure a message flow that provides a similar function without using the aggregation nodes, by issuing the subtask requests to another application (for example, using the HTTPRequest node), and recording the results of each request in the local environment. After each subtask has completed, merge the results from the LocalEnvironment in a Compute node, and create the combined response message for propagating to the target application. However, all the subtasks are performed sequentially, and they do not provide the performance benefits of parallel operation that you can achieve if you use the aggregation nodes.

Examples of aggregation flows that use the aggregation nodes are provided in the following samples:

- Aggregation
- Airline Reservations

The Aggregation sample demonstrates a simple four-way aggregation, and the Airline Reservations sample simulates requests that are related to an airline reservation service, and illustrates the techniques that are associated with aggregation flows.

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

The aggregation nodes store state for aggregations on WebSphere MQ queues. By default, the following storage queues are used:

- SYSTEM.BROKER.AGGR.CONTROL
- SYSTEM.BROKER.AGGR.REPLY
- SYSTEM.BROKER.AGGR.REQUEST
- SYSTEM.BROKER.AGGR.UNKNOWN
- SYSTEM.BROKER.AGGR.TIMEOUT

However, you can create alternative storage queues and use an aggregation configurable service to specify which queues are to be used by the node. For more information, see “Configuring the storage of events for aggregation nodes” on page 753.

By default, the timeout on the AggregateControl node is set to 0. If you do not specify a timeout (or if you leave it set to 0), aggregation requests that WebSphere

MQ stores are never deleted unless all reply messages are returned. This situation might lead to a gradual build up of messages on the internal queues. Set the timeout to a value greater than zero to ensure that requests are removed and that queues do not fill up with redundant requests. It is good practice to set the timeout value to a large value, for example, 86400 seconds (24 hours), so that the queues clear old aggregation messages even if timeouts are not required or expected.

You can set the timeout either by setting the `Timeout` property on the AggregateControl node, or by using an Aggregation configurable service and specifying the `timeoutSeconds` property. For more information, see "Setting timeout values for aggregation" on page 2736.

The aggregation nodes use WebSphere MQ message expiry to manage timeout of messages. For message expiry to work, the aggregation nodes must browse the message queues. The aggregation nodes browse the queues automatically to ensure that expired messages are processed.

z/OS On z/OS, you can configure WebSphere MQ to run a scavenger process that browses the queues instead of the aggregation nodes. To enable the scavenger, set the broker's queue manager property EXPRYINT to a value of 5 seconds. If you do not set EXPRYINT, or set it to a value higher than 10 seconds, the aggregation nodes revert to browsing the aggregation queues automatically.

Related concepts:

"Message flow nodes" on page 1024
A message flow node is a processing step in a message flow. It can be a built-in node, a user-defined node, or a subflow node.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Configuring aggregation flows" on page 2721
Use aggregation message flows to generate and fan-out a number of related requests, fan-in the corresponding replies, and compile those replies into a single aggregated reply message, by using the AggregateControl, AggregateRequest, and AggregateReply nodes.

"Configuring the storage of events for aggregation nodes" on page 753
You can use an Aggregation configurable service to control the storage of events for AggregateControl and AggregateReply nodes.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"AggregateControl node" on page 4296
Use the AggregateControl node to mark the beginning of a fan-out of requests that are part of an aggregation.

"AggregateReply node" on page 4299
Use the AggregateReply node to mark the end of an aggregation fan-in. This node collects replies and combines them into a single compound message.

"AggregateRequest node" on page 4303
Use the AggregateRequest node to record the fact that request messages have been sent. This node also collects information that helps the AggregateReply node to construct the compound response message.

Airline Reservations

Configuring aggregation flows

Use aggregation message flows to generate and fan-out a number of related requests, fan-in the corresponding replies, and compile those replies into a single aggregated reply message, by using the AggregateControl, AggregateRequest, and AggregateReply nodes.

Before you begin

Before you start:

Read the following concept topic:

- "Message flow aggregation" on page 2718

About this task

By default, the messages are put onto a set of default storage queues (beginning SYSTEM.BROKER.AGGR) for processing, but you can use an Aggregation configurable service to specify alternative queues. You can also use the Aggregation configurable service to set a timeout for the aggregation.

For an overview of using aggregation in message flows, see "Message flow aggregation" on page 2718.

To configure aggregation flows see the following topics:

- "Creating the aggregation fan-out flow" on page 2722
- "Creating the aggregation fan-in flow" on page 2728
- "Associating fan-out and fan-in aggregation flows" on page 2733
- "Setting timeout values for aggregation" on page 2736
- "Processing timed out aggregation messages" on page 2739
- "Using multiple AggregateControl nodes" on page 2741
- "Correlating input request and output response aggregation messages" on page 2744
- "Using control messages in aggregation flows" on page 2745
- "Handling exceptions in aggregation flows" on page 2749
- "Configuring the storage of events for aggregation nodes" on page 753

The following sample demonstrates the use of aggregation message flows:

Airline Reservations

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message flow aggregation" on page 2718
Aggregation is the generation and fan-out of related requests that are derived from a single input message, and the fan-in of the corresponding replies to produce a single aggregated reply message.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

Related reference:

"AggregateControl node" on page 4296
Use the AggregateControl node to mark the beginning of a fan-out of requests that are part of an aggregation.

"AggregateReply node" on page 4299
Use the AggregateReply node to mark the end of an aggregation fan-in. This node collects replies and combines them into a single compound message.

"AggregateRequest node" on page 4303
Use the AggregateRequest node to record the fact that request messages have been sent. This node also collects information that helps the AggregateReply node to construct the compound response message.

Creating the aggregation fan-out flow:

The aggregation fan-out flow receives the initial input message and restructures it to present a number of requests to a number of target applications.

Before you begin

Before you start:

- For background information, see "Message flow aggregation" on page 2718.
- Create a message flow project, as described in "Creating a message flow project" on page 1425.

About this task

You can include the fan-out and fan-in flow in the same message flow. However, you might prefer to create two separate flows. For more information about the benefits of configuring separate message flows, see "Associating fan-out and fan-in aggregation flows" on page 2733.

To review an example of a fan-out flow that is supplied with WebSphere Message Broker, see the following sample:

- Airline Reservations

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

To create the fan-out flow:

Procedure

1. Create a message flow to provide the fan-out processing. For more information, see "Creating a message flow" on page 1431.
2. Add the following nodes to the message flow, then configure and connect them as described.

   Input node

   The input node receives an input message from which multiple request messages are generated. This node can be any one of the built-in nodes, or a user-defined input node.

   a. Select the input node to open the Properties view. The node properties are displayed.
   b. Specify the source of input messages for this node. For example, specify the name of a WebSphere MQ queue in the `Queue name` property. The MQInput node retrieves messages from this queue.
   c. Optional: set values for any other properties that you want to configure for this node. For example, to ensure that aggregate request messages are put under sync point, set the `Transaction mode` property to `Yes`. This option avoids the situation where the AggregateReply node receives response messages before it has received the control message that informs it of the aggregation instance. Putting the fan-out flow under transactional control ensures that the fan-out flow completes before any response messages get to the AggregateReply.
   d. Connect the input node's Out terminal to the In terminal of an AggregateControl node. This option represents the simplest configuration; if appropriate, you can include other nodes between the input node and the AggregateControl node. For example, store the request for audit purposes (in a DatabaseWarehouse node), or add a unique identifier to the message (in a Compute node).
   e. Optional: if your fan-out and fan-in flows are combined in one message flow, modify the `Order mode` property on the `Advanced` tab. Select the `By Queue Order` option and ensure that the `Logical Order` property is also selected. These options force the input node to be single threaded to maintain the logical order of the messages that arrive on the queue. Any additional instance threads that you make available are then shared among only the fan-in input nodes to improve the performance of aggregation. If your fan-in and fan-out flows are in separate message flows this step is not required because you can make additional threads available specifically to the fan-in flow.

   AggregateControl node

   The AggregateControl node updates the local environment that is associated with the input message with information required by the AggregateRequest node. The AggregateControl node creates the `LocalEnvironment.ComIbmAggregateControlNode` folder. This folder and

the fields in it are for internal use by WebSphere Message Broker and you should not rely on their existence or values when developing your message flows.

a. Select the AggregateControl node to open the Properties view. The node properties are displayed.
b. Set the `Aggregate name` property of the AggregateControl node to identify this particular aggregation. It is used later to associate this AggregateControl node with a specific AggregateReply node. The `Aggregate name` that you specify must be contextually unique in a broker.
c. Optional: set the `Timeout` property to specify how long the broker waits for replies to arrive before taking action (described in "Setting timeout values for aggregation" on page 2736). If a timeout is not set on the AggregateControl node then aggregate requests stored internally will not be removed unless all aggregate reply messages return. This situation might lead to a gradual build-up of messages on the internal queues. To avoid this situation, set the timeout to a value other than zero (zero means that a timeout never occurs) so that when the timeout is reached the requests are removed and the queues do not fill up with redundant requests. Even if timeouts are not required or expected, it is good practice to set the timeout value to a large value; for example, 86400 seconds (24 hours) so that the queues occasionally get cleared of old aggregations.
d. Connect the Out terminal of the AggregateControl node to the In terminal of one or more Compute nodes that provide the analysis and breakdown of the request in the input message that is propagated on this terminal.

Attention: The Control terminal of the AggregateControl node was deprecated at Version 6.0 and by default any connections from this terminal to the AggregateReply node (either direct or indirect) are ignored. This configuration maximizes the efficiency of aggregation flows and does not damage the reliability of aggregations. This configuration is the optimum configuration.

However, if you do want a control message to be sent from the AggregateControl node to the AggregateReply node, you must connect the Control terminal to the corresponding AggregateReply node on the fan-in flow (either directly or indirectly, as described in "Associating fan-out and fan-in aggregation flows" on page 2733). If you connect it indirectly to the AggregateReply node, for example through an MQOutput node, you must include a Compute node to add the appropriate headers to the message to ensure that it can be safely transmitted.

In addition, for the Control terminal and connections from it to be recognized, you must enable the environment variable MQSI_AGGR_COMPAT_MODE. However, choosing this option has implications regarding the performance and behavior of message aggregations. For a full description of these implications and the environment variable, see "Using control messages in aggregation flows" on page 2745.

Compute node

The Compute node extracts information from the input message and constructs a new output message.

If the target applications that handle the subtask requests can extract the information that they require from the single input message, you do not need to include a Compute node to split the message. You can pass the whole input message to all target applications.

If your target applications expect to receive an individual request, not the whole input message, you must include a Compute node to generate each individual subtask output message from the input message. Configure each Compute node in the following way, copying the appropriate subset of the input message to each output message:

a. Select the Compute node to open the Properties view. The node properties are displayed.

b. Select a value for the Basic property `Compute mode`. This property specifies which sections of the message tree are modified by the node. The AggregateControl node inserts elements into the local environment tree in the input message that the AggregateRequest node reads when the message reaches it. Ensure that the local environment is copied from the input message to the output message in the Compute node. This configuration happens automatically unless you specify a value that includes local environment (one of All, LocalEnvironment, LocalEnvironment and Message, or Exception and LocalEnvironment).

   If you specify one of these values, the broker assumes that you are customizing the Compute node with ESQL that writes to local environment, and that you are copying any elements in that tree that are required in the output message.

   To modify the local environment, add the following statement to copy the required aggregate information from input message to output message:

```
SET OutputLocalEnvironment.ComIbmAggregateControlNode =
           InputLocalEnvironment.ComIbmAggregateControlNode;
```

c. Optional: set values for any other properties that you want to configure for this node.

d. Connect the Out terminal of each Compute node to the In terminal of the output node that represents the destination of the output request message that you have created from the input message in this node.

Output node

Include an output node for each output message that you generate in your fan-out flow. Configure each node, as described later in this section, with the appropriate modifications for each destination.

The output node must be an output node that supports the request/reply model, such as an MQOutput node, or a mixture of these nodes (depending on the requirements of the target applications).

a. Select the output node to open the Properties view. The node properties are displayed.

b. Specify the destination for the output messages for this node. For example, specify the name of a WebSphere MQ queue in the `Queue name` property to which the MQOutput node sends messages. The

target application must process its request, and send the response to the reply destination indicated in its input message (for example the WebSphere MQ ReplyToQueue).

c. Click **Request** in the left view and set values for these properties to specify that replies are sent to the input queue of the fan-in flow.

d. Optional: set values for any other properties that you want to configure for this node.

e. Connect the Out terminal of the output node to the In terminal of an AggregateRequest node. When the message is propagated through the output node's Out terminal, the built-in output node updates the WrittenDestination folder in the associated local environment with additional information required by the AggregateRequest node.

The information written by the built-in nodes is queue name, queue manager name, message ID, and correlation ID (from the MQMD), and message reply identifier (set to the same value as message ID).

AggregateRequest node

Include an AggregateRequest node for each output message that you generate in your fan-out flow.

a. Select the AggregateRequest node to open the Properties view. The node properties are displayed.

b. Set the Basic property `Folder name` to a value that identifies the type of request that has been sent out. This value is used by the AggregateReply node to match up with the reply message when it is received in the fan-in flow. The folder name that you specify for each request that the fan-out flow generates must be unique.

The AggregateRequest node writes a record in WebSphere MQ for each message that it processes. This record enables the AggregateReply node to identify which request each response is associated with. If your output nodes are non-transactional, the response message might arrive at the fan-in flow before this database update is committed. For details on how you can use timeouts to avoid this situation, see "Setting timeout values for aggregation" on page 2736.

CAUTION:
Although the use of timeouts can help to avoid this situation described previously, configure your fan-out flow to be transactional. Therefore, response messages cannot get to the fan-in flow before the corresponding AggregateRequest nodes have committed their database updates.

3. To save the message flow and validate its configuration, press **Ctrl-S** or click **File** > **Save**.

What to do next

To collect the aggregation responses initiated by your fan-out flow, create a fan-in flow, as described in "Creating the aggregation fan-in flow" on page 2728.

Related concepts:

"Local environment tree structure" on page 1056
The local environment tree is a part of the logical message tree in which you can store information while the message flow processes the message.

"Message flow aggregation" on page 2718
Aggregation is the generation and fan-out of related requests that are derived from a single input message, and the fan-in of the corresponding replies to produce a single aggregated reply message.

"Developing user-defined extensions" on page 2970
A user-defined extension is a component that you design and implement to extend the function of WebSphere Message Broker.

Related tasks:

"Configuring aggregation flows" on page 2721
Use aggregation message flows to generate and fan-out a number of related requests, fan-in the corresponding replies, and compile those replies into a single aggregated reply message, by using the AggregateControl, AggregateRequest, and AggregateReply nodes.

"Creating the aggregation fan-in flow" on page 2728
The aggregation fan-in flow receives the responses to the request messages that are sent out by the fan-out flow, and constructs a combined response message containing all the responses received.

"Using control messages in aggregation flows" on page 2745
The default behavior is that connections between AggregateControl and AggregateReply nodes for sending control messages are ignored. This configuration optimizes performance and removes the possibility that response messages will be received by the AggregateReply node before the control message.

"Associating fan-out and fan-in aggregation flows" on page 2733
Associate the fan-out message flow processing with its corresponding fan-in message flow processing by setting the `Aggregate Name` property of the AggregateControl and AggregateReply nodes in your aggregation flow to the same value.

"Setting timeout values for aggregation" on page 2736
You can use two properties of the aggregation nodes to set timeout values for aggregated message processing.

"Using multiple AggregateControl nodes" on page 2741
You might find it useful to design a fan-out flow with multiple AggregateControl nodes, all with the same value set for the property `Aggregate Name`, but with different values for the `Timeout` property. You can reuse an aggregate name in only this situation.

"Handling exceptions in aggregation flows" on page 2749
When you use aggregation flows, exceptions might occur.

Related reference:

"AggregateControl node" on page 4296
Use the AggregateControl node to mark the beginning of a fan-out of requests that are part of an aggregation.

"AggregateReply node" on page 4299
Use the AggregateReply node to mark the end of an aggregation fan-in. This node collects replies and combines them into a single compound message.

"AggregateRequest node" on page 4303
Use the AggregateRequest node to record the fact that request messages have been sent. This node also collects information that helps the AggregateReply node to construct the compound response message.

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.
"MQOutput node" on page 4612
Use the MQOutput node to send messages to clients that connect to the broker using the WebSphere MQ Enterprise Transport and that use the MQI and AMI application programming interfaces.

Creating the aggregation fan-in flow:

The aggregation fan-in flow receives the responses to the request messages that are sent out by the fan-out flow, and constructs a combined response message containing all the responses received.

Before you begin

Before you start:

- Read an overview of aggregation in "Message flow aggregation" on page 2718.
- Create a message flow project, as described in "Creating a message flow project" on page 1425.

About this task

You can include the fan-out and fan-in flow within the same message flow. However, you might prefer to create two separate flows. For more information about the benefits of configuring separate message flows, see "Associating fan-out and fan-in aggregation flows" on page 2733. Do not deploy multiple copies of the same fan-in flow either to the same or to different execution groups.

If you do not configure the fan-out flow to be transactional, the timeout values that you have specified might result in the combined response message being generated before the fan-in flow has received all the replies. For more information, see "Creating the aggregation fan-out flow" on page 2722.

To review an example of a fan-in flow see the following sample:

- Airline Reservations

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

To create the fan-in flow:

Procedure

1. Create a message flow to provide the fan-in processing.
2. Add the following nodes in the editor view and configure and connect them as described:

   Input node
   The input node receives the responses to the multiple request messages that are generated from the fan-out flow.

   This node must be an input node that supports the request/reply model, such as an MQInput node, or a mixture of these nodes

(depending on the requirements of the applications that send these responses). The response that is received by each input node must be sent across the same protocol as the request to which it corresponds. For example, if you include an MQOutput node in the fan-out flow, the response to that request must be received by an MQInput node in this fan-in flow.

a. Select the input node to open the Properties view. The node properties are displayed.
b. Specify the source of input messages for this node; for example, specify the name of a WebSphere MQ queue in the Basic property Queue name from which the MQInput node retrieves messages.
c. Optional: specify values for any other properties that you want to configure for this node.
d. Connect the Out terminal of the input node to the In terminal of an AggregateReply node.

   Connect the terminals in this way to create the simplest configuration; if appropriate, you can include other nodes between the input node and the AggregateReply node; for example, you might want to store the replies for audit purposes (in a Warehouse node).

Include just one input node that receives all the aggregation response messages at the beginnings of the fan-in flow as described previously. If you include multiple input nodes, threads that are started by a specific reply input node might complete the aggregation and execution of the message flow while other threads are sending their response messages to the AggregateReply node and becoming eligible to timeout. Use a single input node to enable sequential processing of replies for each aggregation. Specify additional instances to provide greater processing throughput in this single node, see "Configurable message flow properties" on page 4020.

AggregateReply node

The AggregateReply node receives the inbound responses from the input node through its In terminal. The AggregateReply node stores each reply message for subsequent processing.

When all the replies for a particular group of aggregation requests have been collected, the AggregateReply node creates an aggregated reply message, and propagates it through the Out terminal.

a. Select the AggregateReply node to open the Properties view. The node properties are displayed.
b. Set the Aggregate name property of the AggregateReply node to identify this aggregation. Set this value to be the same value that you set for the Aggregate name property in the corresponding AggregateControl node in the fan-out flow.
c. Optional: to retain an unrecognized message before propagating it to the Unknown terminal, set a value for the Unknown message timeout property. If you are using separate fan-out and fan-in flows, set this value to a non-zero number if the control message might be delayed.
d. Optional: to explicitly handle unrecognized messages, connect the Unknown terminal to another node, or sequence of nodes. If you do not connect this terminal to another node in the message flow, messages propagated through this terminal are discarded.

e. Optional: if you have specified a timeout value for this aggregation in the AggregateControl node, and you want to explicitly handle timeouts that expire before all replies are received, connect the Timeout terminal to another node, or sequence of nodes. Partially complete aggregated replies are sent to the Timeout terminal if the timer expires. If you do not connect this terminal to another node in the message flow, messages propagated through this terminal are discarded.

f. Optional: specify values for any other properties that you want to configure for this node.

g. Connect the Out terminal of the AggregateReply node to the In terminal of a Compute node.

Attention: The Control terminal of the AggregateReply node was deprecated at Version 6.0, and by default any connections to this terminal (either direct or indirect) are ignored. This change maximizes the efficiency of aggregation flows and does not damage the reliability of aggregations. This configuration provides the optimum content.

However, if you want the AggregateReply node to receive, on its Control terminal, the control message that was sent by the corresponding AggregateControl node on the fan-out flow, you must make the necessary connections as described in "Creating the aggregation fan-out flow" on page 2722. Keep the path from the AggregateReply node to the output node as direct as possible to maximize the performance of aggregations. Do not modify the content of this control message.

In addition, for the Control terminal and connections to it to be recognized, you must enable the environment variable MQSI_AGGR_COMPAT_MODE. If you choose this option, the performance and behavior of message aggregations might be affected; for a full description of these implications and the environment variable, see "Using control messages in aggregation flows" on page 2745.

Aggregated messages which are sent from the AggregateReply node output terminals (Out and Timeout) are not validated. Validation of data must be done before messages are sent to the AggregateReply node, because it ignores validation options when reconstructing the stored messages.

Compute node

The Compute node receives the message that contains the combined responses. Typically, the format of this combined message is not valid for output, because the aggregated reply message has an unusual structure and cannot be parsed into the bit stream required by some nodes, for example the MQOutput node. The Out and Timeout terminals always propagate an aggregated reply message, which always requires further processing before it can be propagated to an MQOutput node. Therefore you must include a Compute node and configure this node to create a valid output message.

a. Select the Compute node to open the Properties view. The node properties are displayed.

b. Specify in the Basic property ESQL module the name of the ESQL module that customizes the function of this node.
c. Right-click the node and click **Open ESQL** to open the ESQL file that contains the module for this node. The module is highlighted in the ESQL editor view.
d. Code the ESQL to create a single output message from the aggregated replies in the input message.

   The aggregated reply message is propagated through the Out terminal. Information about how you can access its contents is provided in "Accessing the combined message contents."
e. Optional: specify values for any other properties that you want to configure for this node.
f. Connect the Out terminal of the Compute node to the In terminal of the output node that represents the destination of the single response message.

Output node

Include an output node for your fan-in flow. This node can be any of the built-in nodes, or a user-defined output node.

a. Select the output node to open the Properties view. The node properties are displayed.
b. Specify the destination for the output message for this node; for example, specify in the Basic property Queue name the name of a WebSphere MQ queue to which the MQOutput node sends messages.
c. Optional: specify values for any other properties that you want to configure for this node.

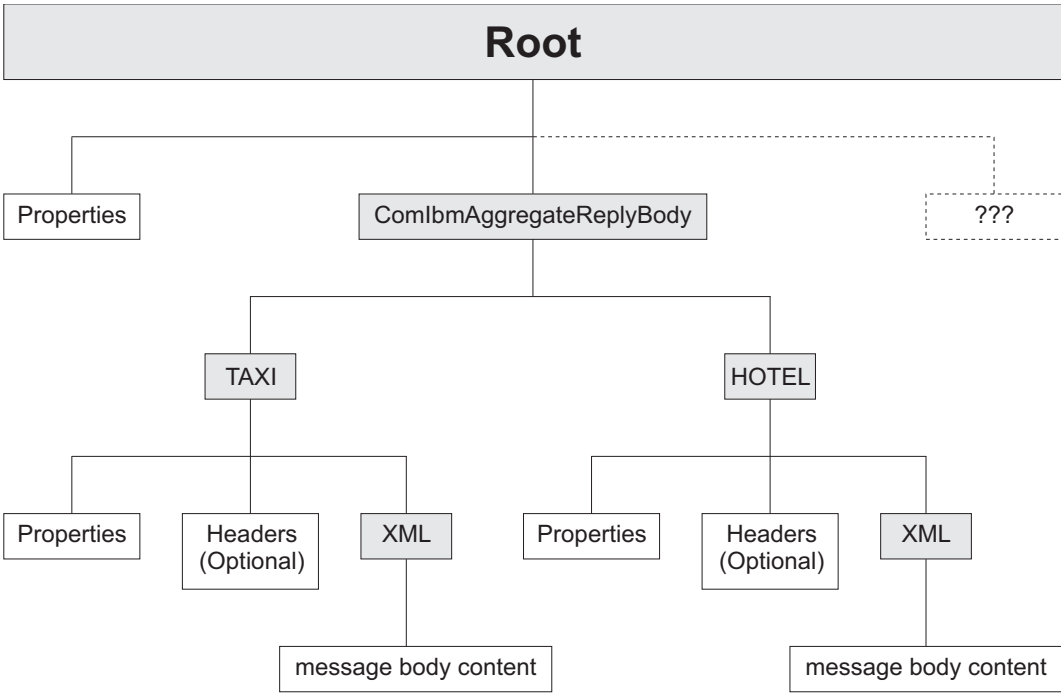
3. To save the message flow and validate its configuration, press Ctrl-S or click **File** > **Save**.

Accessing the combined message contents:
About this task

The AggregateReply node creates a folder in the combined message tree below Root, called ComIbmAggregateReplyBody. Below this folder, the node creates a number of subfolders using the names that you set in the AggregateRequest nodes. These subfolders are populated with the associated reply messages.

For example, the request messages might have folder names:
- TAXI
- HOTEL

The resulting aggregated reply message created by the AggregateReply node might have a structure like the following example:



Use ESQL within a Compute node to access the reply from the taxi company using the following correlation name:

```
InputRoot.ComIbmAggregateReplyBody.TAXI.xyz
```

The folder name does not have to be unique. If you have multiple requests with the folder name TAXI, you can access the separate replies using the array subscript notation, for example:

```
InputRoot.ComIbmAggregateReplyBody.TAXI[1].xyz
InputRoot.ComIbmAggregateReplyBody.TAXI[2].xyz
```

Related concepts:

"Message flow aggregation" on page 2718
Aggregation is the generation and fan-out of related requests that are derived from a single input message, and the fan-in of the corresponding replies to produce a single aggregated reply message.

"Developing user-defined extensions" on page 2970
A user-defined extension is a component that you design and implement to extend the function of WebSphere Message Broker.

Related tasks:

"Configuring aggregation flows" on page 2721
Use aggregation message flows to generate and fan-out a number of related requests, fan-in the corresponding replies, and compile those replies into a single aggregated reply message, by using the AggregateControl, AggregateRequest, and AggregateReply nodes.

"Creating the aggregation fan-out flow" on page 2722
The aggregation fan-out flow receives the initial input message and restructures it to present a number of requests to a number of target applications.

"Using control messages in aggregation flows" on page 2745
The default behavior is that connections between AggregateControl and AggregateReply nodes for sending control messages are ignored. This configuration optimizes performance and removes the possibility that response messages will be received by the AggregateReply node before the control message.

"Associating fan-out and fan-in aggregation flows"
Associate the fan-out message flow processing with its corresponding fan-in message flow processing by setting the `Aggregate Name` property of the AggregateControl and AggregateReply nodes in your aggregation flow to the same value.

"Setting timeout values for aggregation" on page 2736
You can use two properties of the aggregation nodes to set timeout values for aggregated message processing.

Related reference:

"AggregateControl node" on page 4296
Use the AggregateControl node to mark the beginning of a fan-out of requests that are part of an aggregation.

"AggregateReply node" on page 4299
Use the AggregateReply node to mark the end of an aggregation fan-in. This node collects replies and combines them into a single compound message.

"AggregateRequest node" on page 4303
Use the AggregateRequest node to record the fact that request messages have been sent. This node also collects information that helps the AggregateReply node to construct the compound response message.

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

"MQOutput node" on page 4612
Use the MQOutput node to send messages to clients that connect to the broker using the WebSphere MQ Enterprise Transport and that use the MQI and AMI application programming interfaces.

Associating fan-out and fan-in aggregation flows:

Associate the fan-out message flow processing with its corresponding fan-in message flow processing by setting the `Aggregate Name` property of the AggregateControl and AggregateReply nodes in your aggregation flow to the same value.

Before you begin

If you did not configure this property when you created your fan-in and fan-out flows, you must complete this task.

Before you start:

You must have completed the following tasks:
- "Creating the aggregation fan-out flow" on page 2722
- "Creating the aggregation fan-in flow" on page 2728

About this task

The `Aggregate Name` must be contextually unique within a broker. You can have only one AggregateControl node and one AggregateReply node with a particular `Aggregate Name`, although you can have more than one AggregateControl node with the same `Aggregate Name`, see "Using multiple AggregateControl nodes" on page 2741. Do not deploy a fan-in flow to multiple execution groups on the same broker; results are unpredictable.

You can either create the fan-out and fan-in flows in the same message flow, or in two different message flows. In either case, the two parts of the aggregation are associated when you set the `Aggregate Name` property.

How you configure your aggregation flow depends on a number of factors:

- The design of your message flow.
- The hardware on which the broker is running.
- The timeout values that you choose, see "Setting timeout values for aggregation" on page 2736.
- How you expect to maintain the message flows.

You can include the fan-out and fan-in flow within the same message flow. However, you might prefer to create two separate flows. The advantages of creating separate fan-out and fan-in flows are:

- You can modify the two flows independently.
- You can start and stop the two flows independently.
- You can deploy the two flows to separate execution groups to take advantage of multiprocessor systems, or to provide data segregation for security or integrity purposes.
- You can allocate different numbers of additional threads to the two flows, as appropriate, to maintain a suitable processing ratio.

The following sample shows the use of two flows for aggregation:

- Airline Reservations

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

To associate the fan-out flow with the fan-in flow:

Procedure

1. Switch to the Broker Application Development perspective.
2. Open the message flow that contains your fan-out flow.
3. Select the AggregateControl node to open the Properties view. The node properties are displayed.
4. Set the `Aggregate Name` property of the AggregateControl node to identify this aggregation. The `Aggregate Name` that you specify must be contextually unique within a broker.
5. `If you have separate fan-out and fan-in flows:`

a. Press Ctrl-S or click **File** > **Save** ***name*** on the taskbar menu (where *name* is the name of this message flow) to save the message flow and validate its configuration.
b. Open the message flow that contains your fan-in flow.

6. Select the AggregateControl node to open the Properties view. The node properties are displayed.
7. Set the `Aggregate Name` property of the AggregateReply node to the same value that you set for the `Aggregate Name` property in the corresponding AggregateControl node in the fan-out flow.
8. Press Ctrl-S or click **File** > **Save** ***name*** to save the message flow and validate its configuration.

What to do next

In WebSphere Message Broker, fan-out and fan-in flows were also associated by sending control messages from the AggregateControl node to the AggregateReply node. This facility is no longer available. For optimum performance, do not connect the AggregateControl and AggregateReply node. However, if you do want to use control messages in your aggregations, and you want to connect these two nodes, see "Using control messages in aggregation flows" on page 2745.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message flow aggregation" on page 2718
Aggregation is the generation and fan-out of related requests that are derived from a single input message, and the fan-in of the corresponding replies to produce a single aggregated reply message.

Related tasks:

"Configuring aggregation flows" on page 2721
Use aggregation message flows to generate and fan-out a number of related requests, fan-in the corresponding replies, and compile those replies into a single aggregated reply message, by using the AggregateControl, AggregateRequest, and AggregateReply nodes.

"Creating the aggregation fan-out flow" on page 2722
The aggregation fan-out flow receives the initial input message and restructures it to present a number of requests to a number of target applications.

"Setting timeout values for aggregation" on page 2736
You can use two properties of the aggregation nodes to set timeout values for aggregated message processing.

"Using multiple AggregateControl nodes" on page 2741
You might find it useful to design a fan-out flow with multiple AggregateControl nodes, all with the same value set for the property `Aggregate Name`, but with different values for the `Timeout` property. You can reuse an aggregate name in only this situation.

"Handling exceptions in aggregation flows" on page 2749
When you use aggregation flows, exceptions might occur.

"Using control messages in aggregation flows" on page 2745
The default behavior is that connections between AggregateControl and AggregateReply nodes for sending control messages are ignored. This configuration optimizes performance and removes the possibility that response messages will be received by the AggregateReply node before the control message.

Related reference:

"AggregateControl node" on page 4296
Use the AggregateControl node to mark the beginning of a fan-out of requests that are part of an aggregation.

"AggregateReply node" on page 4299
Use the AggregateReply node to mark the end of an aggregation fan-in. This node collects replies and combines them into a single compound message.

"AggregateRequest node" on page 4303
Use the AggregateRequest node to record the fact that request messages have been sent. This node also collects information that helps the AggregateReply node to construct the compound response message.

Setting timeout values for aggregation:

You can use two properties of the aggregation nodes to set timeout values for aggregated message processing.

Before you begin

Before you start:

To complete this task, you must have completed the following tasks:
- "Creating the aggregation fan-out flow" on page 2722
- "Creating the aggregation fan-in flow" on page 2728

About this task

Two situations might require the use of timeouts:
- The need to receive an aggregated reply message within a specified time. For this situation, you set the `Timeout` property of the AggregateControl node.
- The need to wait before propagating an unrecognized message to the Unknown terminal. For this situation, you set the `Unknown Message Timeout` property on the AggregateReply node.

The following sections describe each situation in more detail.

To receive an aggregated reply message within a specified time:
About this task

In some situations you might need to receive an aggregated reply message within a specified time. Some reply messages might be slow to return, or might never arrive. For these situations, complete the following steps.

1. Open the fan-out message flow.
2. Set the `Timeout` property of the AggregateControl node to specify how long (in seconds) the broker must wait for replies. By default, this property is set to 0 (zero), which means that there is no timeout and the broker waits indefinitely.

By default, timeout polling occurs every 5 seconds. Therefore, if you set the `Timeout` property to a value that is not a multiple of 5, an extra delay occurs. For example, if you set the `Timeout` property to 7 seconds, you see a delay of 3 seconds until timeout polling next occurs. You can change the default timeout polling interval by using the environment variable MQSI_AGGR_WAIT_TIMEOUT. Valid values are between 1000 and 5000 milliseconds. To change the default polling

interval, stop the broker, then restart the broker in an environment where you have set the MQSI_AGGR_WAIT_TIMEOUT environment variable.

You can also use an Aggregation configurable service to specify the timeout interval. You can create an Aggregation configurable service with the same name as the `Aggregate name` property of the AggregateControl node, and specify the timeout value in the `timeoutSeconds` property of the configurable service. Alternatively, you can create an Aggregation configurable service with the same name as the execution group. Values from Aggregation configurable services are taken in the following order:

1. If an Aggregation configurable service exists with the same name as the aggregation (defined in the `Aggregate name` property), that configurable service is used.
2. If no configurable service exists with the same name as aggregation, but a configurable service exists with the same name as the execution group, that configurable service is used.

If an Aggregation configurable service exists with the same name as an execution group, it can be used for every aggregation in the execution group. However, if a configurable service exists with the same name as an aggregation in the execution group, it is used for that specific aggregation. In each case, the value of the `timeoutSeconds` property of the configurable service overrides the `Timeout` property of the AggregateControl node.

Alternatively, you can use the `Timeout location` property of the AggregateControl node to specify the location of a timeout value in the incoming message. Any timeout value specified in this way overrides the values specified by the AggregateControl node and the Aggregation configurable service.

The timeout is determined by values in the message, the configurable service, or the node, in the following order:

1. If a timeout value is specified in the incoming message (in the location specified by the `Timeout location` property of the AggregateControl node) this value is used.
2. If no location is specified by the `Timeout location` property, or if the location in the message is empty or does not exist, the value specified by the `timeoutSeconds` property of the Aggregation configurable service is used (if one exists).
3. If no Aggregation configurable service has been defined, or if the `timeoutSeconds` property of the configurable service is not set, the value set in the `Timeout` property of the AggregateControl node is used.

For more information about the Aggregation configurable service, see "Configurable services properties" on page 3766.

If the timeout interval passes without all the replies arriving, the replies that have arrived are turned into an aggregated reply message by the corresponding AggregateReply node, and propagated to its Timeout terminal. You can process this partial response message in the same way as a complete aggregated reply message. If you prefer, you can provide special processing for incomplete aggregated replies.

To wait before propagating an unrecognized message to the Unknown terminal:

About this task

When a message arrives at the In terminal of an AggregateReply node, it is examined to see if it is an expected reply message. If it is not recognized, the message is propagated to the Unknown terminal. You might want the broker to wait for a specified period of time before propagating the message for the following reasons:

- The reply message might arrive before the work completed by the AggregateRequest node has been transactionally committed. This situation can be avoided by configuring the `Transaction mode` property of the input node, as described in "Creating the aggregation fan-out flow" on page 2722.
- The reply message might arrive before the control message. This situation can be avoided by leaving the Control terminal of the AggregateControl node unconnected. For further information about the implications of connecting the Control terminal, see "Using control messages in aggregation flows" on page 2745.

These situations are most likely to happen if you send the request messages out of syncpoint, and might result in valid replies being sent to the Unknown terminal. To reduce the likelihood of this event, complete the following steps.

1. Open the fan-in message flow.
2. Set the `Unknown Message Timeout` property of the AggregateReply node.

   When you set this property, a message that cannot be recognized immediately as a valid reply is held persistently in the broker for the number of seconds that you specify for this property.

If the unknown timeout interval expires, and the message is recognized, it is processed. The node also checks to see if this previously unknown message is the last reply needed to make an aggregation complete. If it is, the aggregated reply message is constructed and propagated.

If the unknown timeout interval expires and the message is still not recognized, the message is propagated to the Unknown terminal.

Related concepts:

"Message flow aggregation" on page 2718
Aggregation is the generation and fan-out of related requests that are derived from a single input message, and the fan-in of the corresponding replies to produce a single aggregated reply message.

Related tasks:

"Configuring aggregation flows" on page 2721
Use aggregation message flows to generate and fan-out a number of related requests, fan-in the corresponding replies, and compile those replies into a single aggregated reply message, by using the AggregateControl, AggregateRequest, and AggregateReply nodes.

"Creating the aggregation fan-out flow" on page 2722
The aggregation fan-out flow receives the initial input message and restructures it to present a number of requests to a number of target applications.

"Creating the aggregation fan-in flow" on page 2728
The aggregation fan-in flow receives the responses to the request messages that are sent out by the fan-out flow, and constructs a combined response message containing all the responses received.

"Associating fan-out and fan-in aggregation flows" on page 2733
Associate the fan-out message flow processing with its corresponding fan-in message flow processing by setting the `Aggregate Name` property of the AggregateControl and AggregateReply nodes in your aggregation flow to the same value.

"Avoiding thread starvation on fan-in flows" on page 2748
Follow this guidance to avoid thread starvation on fan-in flows if the Control terminal of the AggregateControl node in your fan-out flow is connected to output control messages to a queue.

"Processing timed out aggregation messages"
Assign additional processing threads to enable processing of timed out aggregation messages in the AggregateReply node.

"Using multiple AggregateControl nodes" on page 2741
You might find it useful to design a fan-out flow with multiple AggregateControl nodes, all with the same value set for the property `Aggregate Name`, but with different values for the `Timeout` property. You can reuse an aggregate name in only this situation.

"Handling exceptions in aggregation flows" on page 2749
When you use aggregation flows, exceptions might occur.

"Using control messages in aggregation flows" on page 2745
The default behavior is that connections between AggregateControl and AggregateReply nodes for sending control messages are ignored. This configuration optimizes performance and removes the possibility that response messages will be received by the AggregateReply node before the control message.

"Configuring the storage of events for aggregation nodes" on page 753
You can use an Aggregation configurable service to control the storage of events for AggregateControl and AggregateReply nodes.

Related reference:

"AggregateControl node" on page 4296
Use the AggregateControl node to mark the beginning of a fan-out of requests that are part of an aggregation.

"AggregateReply node" on page 4299
Use the AggregateReply node to mark the end of an aggregation fan-in. This node collects replies and combines them into a single compound message.

"AggregateRequest node" on page 4303
Use the AggregateRequest node to record the fact that request messages have been sent. This node also collects information that helps the AggregateReply node to construct the compound response message.

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

Processing timed out aggregation messages:

Assign additional processing threads to enable processing of timed out aggregation messages in the AggregateReply node.

Before you begin

Before you start:

To complete this task, the following conditions must be met:

- The AggregateReply node must be using an Aggregation configurable service.
- The associated Aggregation configurable service must have a queue prefix set that is unique across all Aggregation configurable services.
- The set of aggregation queues referred to by the associated Aggregation configurable service must only be used by a single execution group on a single broker.

About this task

By default the AggregateReply node uses a single thread to process timed out aggregation messages. In scenarios where a high volume of messages are expected to timeout, a backlog of messages can accumulate on the SYSTEM.BROKER.AGGR.*QueuePrefix*.TIMEOUT queue. Follow this task to assign additional processing threads for timeout processing in the AggregateReply node.

You can configure the number of threads that are used to process timeout messages by setting the `timeoutThreads` property of the associated Aggregation configurable service. For example, use the following command:

```
mqsichangeproperties MYBROKER -c Aggregation -o myAggregationService -n timeoutThreads -v 10
```

This command sets the timeout threads of an aggregation to 10 for all nodes that are configured to use the myAggregationService configurable service.

Related concepts:

"Message flow aggregation" on page 2718
Aggregation is the generation and fan-out of related requests that are derived from a single input message, and the fan-in of the corresponding replies to produce a single aggregated reply message.

Related tasks:

"Configuring aggregation flows" on page 2721
Use aggregation message flows to generate and fan-out a number of related requests, fan-in the corresponding replies, and compile those replies into a single aggregated reply message, by using the AggregateControl, AggregateRequest, and AggregateReply nodes.

"Creating the aggregation fan-out flow" on page 2722
The aggregation fan-out flow receives the initial input message and restructures it to present a number of requests to a number of target applications.

"Creating the aggregation fan-in flow" on page 2728
The aggregation fan-in flow receives the responses to the request messages that are sent out by the fan-out flow, and constructs a combined response message containing all the responses received.

"Associating fan-out and fan-in aggregation flows" on page 2733
Associate the fan-out message flow processing with its corresponding fan-in message flow processing by setting the `Aggregate Name` property of the AggregateControl and AggregateReply nodes in your aggregation flow to the same value.

"Avoiding thread starvation on fan-in flows" on page 2748
Follow this guidance to avoid thread starvation on fan-in flows if the Control terminal of the AggregateControl node in your fan-out flow is connected to output control messages to a queue.

"Using multiple AggregateControl nodes" on page 2741
You might find it useful to design a fan-out flow with multiple AggregateControl nodes, all with the same value set for the property `Aggregate Name`, but with different values for the `Timeout` property. You can reuse an aggregate name in only this situation.

"Handling exceptions in aggregation flows" on page 2749
When you use aggregation flows, exceptions might occur.

"Using control messages in aggregation flows" on page 2745
The default behavior is that connections between AggregateControl and AggregateReply nodes for sending control messages are ignored. This configuration optimizes performance and removes the possibility that response messages will be received by the AggregateReply node before the control message.

"Configuring the storage of events for aggregation nodes" on page 753
You can use an Aggregation configurable service to control the storage of events for AggregateControl and AggregateReply nodes.

Related reference:

"AggregateControl node" on page 4296
Use the AggregateControl node to mark the beginning of a fan-out of requests that are part of an aggregation.

"AggregateReply node" on page 4299
Use the AggregateReply node to mark the end of an aggregation fan-in. This node collects replies and combines them into a single compound message.

"AggregateRequest node" on page 4303
Use the AggregateRequest node to record the fact that request messages have been sent. This node also collects information that helps the AggregateReply node to construct the compound response message.

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

Using multiple AggregateControl nodes:

You might find it useful to design a fan-out flow with multiple AggregateControl nodes, all with the same value set for the property `Aggregate Name`, but with different values for the `Timeout` property. You can reuse an aggregate name in only this situation.

Before you begin

Before you start:

For background information, see "Message flow aggregation" on page 2718.

To complete this task, you must have created a message flow project by following the instructions in "Creating a message flow project" on page 1425.

About this task

You might want to use multiple AggregateControl nodes if, for example, you have created an aggregation flow that books a business trip, and you might have some requests that need a reply within two days. However, other, more urgent requests, need a reply within two hours.

To configure an aggregation flow that uses multiple AggregateControl nodes, complete the following steps.

Procedure

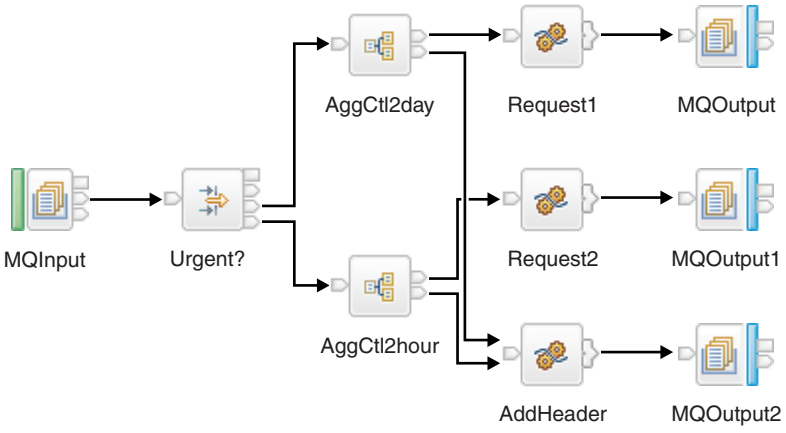
1. Create or open the fan-out message flow.

2. Configure the required number of AggregateControl nodes. Set the Basic property `Aggregate Name` of each node to the same value. For example, include two nodes and enter the name `JOURNEY` as the `Aggregate Name` for both.
3. Set the value for the `Timeout` property in each node to a different value. For example, set the `Timeout` in one node to two hours; set the `Timeout` in the second node to two days.
4. Configure a Filter node to receive incoming requests, check their content, and route them to the correct AggregateControl node.
5. Connect the nodes together to achieve the required result. For example, if you have configured the Filter node to test for requests with a priority field set to urgent, connect the true terminal to the AggregateControl node with the short timeout. Connect the false terminal to the AggregateControl node with the longer timeout. Connect the out terminals of the AggregateControl nodes to the following nodes in the fan-out flow.

   You must connect the two AggregateControl nodes in parallel, not in sequence. This means that you must connect both to the Filter node (one to the true terminal, one to the false), and both to the downstream nodes that handle the requests for the fan-out. Each input message must pass through only one of the AggregateControl nodes. If you connect the nodes such that a single message is processed by more than one AggregateControl node, duplicate records are created in the database by the AggregateRequest node and subsequent processing results are unpredictable.

Results

The following diagram shows an example fan-out message flow that uses this technique.



Related concepts:

"Message flow aggregation" on page 2718
Aggregation is the generation and fan-out of related requests that are derived from a single input message, and the fan-in of the corresponding replies to produce a single aggregated reply message.

Related tasks:

"Configuring aggregation flows" on page 2721
Use aggregation message flows to generate and fan-out a number of related requests, fan-in the corresponding replies, and compile those replies into a single aggregated reply message, by using the AggregateControl, AggregateRequest, and

AggregateReply nodes.

"Creating the aggregation fan-out flow" on page 2722
The aggregation fan-out flow receives the initial input message and restructures it to present a number of requests to a number of target applications.

"Creating the aggregation fan-in flow" on page 2728
The aggregation fan-in flow receives the responses to the request messages that are sent out by the fan-out flow, and constructs a combined response message containing all the responses received.

"Associating fan-out and fan-in aggregation flows" on page 2733
Associate the fan-out message flow processing with its corresponding fan-in message flow processing by setting the `Aggregate Name` property of the AggregateControl and AggregateReply nodes in your aggregation flow to the same value.

"Using control messages in aggregation flows" on page 2745
The default behavior is that connections between AggregateControl and AggregateReply nodes for sending control messages are ignored. This configuration optimizes performance and removes the possibility that response messages will be received by the AggregateReply node before the control message.

"Avoiding thread starvation on fan-in flows" on page 2748
Follow this guidance to avoid thread starvation on fan-in flows if the Control terminal of the AggregateControl node in your fan-out flow is connected to output control messages to a queue.

"Processing timed out aggregation messages" on page 2739
Assign additional processing threads to enable processing of timed out aggregation messages in the AggregateReply node.

"Setting timeout values for aggregation" on page 2736
You can use two properties of the aggregation nodes to set timeout values for aggregated message processing.

"Handling exceptions in aggregation flows" on page 2749
When you use aggregation flows, exceptions might occur.

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

Related reference:

"AggregateControl node" on page 4296
Use the AggregateControl node to mark the beginning of a fan-out of requests that are part of an aggregation.

"AggregateReply node" on page 4299
Use the AggregateReply node to mark the end of an aggregation fan-in. This node collects replies and combines them into a single compound message.

"AggregateRequest node" on page 4303
Use the AggregateRequest node to record the fact that request messages have been sent. This node also collects information that helps the AggregateReply node to construct the compound response message.

Correlating input request and output response aggregation messages:

If you want to correlate initial request messages with their combined response messages, you can do so using the ReplyIdentifier in the Properties folder of the response message.

Before you begin

Before you start:

To complete this task, you must have completed the following tasks:
- "Creating the aggregation fan-out flow" on page 2722
- "Creating the aggregation fan-in flow" on page 2728

About this task

In some cases you might want to correlate aggregation request messages with the combined response message produced by your fan-in flow, there are two ways of doing this:

- Store some correlation information in one of the requests sent out as part of the aggregation.
- Send the original request message directly back to the AggregateReply node as one of the aggregation requests. To do this, the CorrelId must be set to the MsgId, and the MQOutput node must have its MessageContext set to 'Pass all'.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message flow aggregation" on page 2718
Aggregation is the generation and fan-out of related requests that are derived from a single input message, and the fan-in of the corresponding replies to produce a single aggregated reply message.

Related tasks:

"Configuring aggregation flows" on page 2721
Use aggregation message flows to generate and fan-out a number of related requests, fan-in the corresponding replies, and compile those replies into a single aggregated reply message, by using the AggregateControl, AggregateRequest, and AggregateReply nodes.

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

Related reference:

"AggregateControl node" on page 4296
Use the AggregateControl node to mark the beginning of a fan-out of requests that are part of an aggregation.

"AggregateReply node" on page 4299
Use the AggregateReply node to mark the end of an aggregation fan-in. This node collects replies and combines them into a single compound message.

"AggregateRequest node" on page 4303
Use the AggregateRequest node to record the fact that request messages have been sent. This node also collects information that helps the AggregateReply node to construct the compound response message.

Using control messages in aggregation flows:

The default behavior is that connections between AggregateControl and AggregateReply nodes for sending control messages are ignored. This configuration optimizes performance and removes the possibility that response messages will be received by the AggregateReply node before the control message.

Before you begin

Before you start:

To complete this task, you must have completed the following tasks:
- "Creating the aggregation fan-out flow" on page 2722
- "Creating the aggregation fan-in flow" on page 2728

About this task

Control messages are not required to make aggregations work correctly. However, you can send control messages in your aggregation flows if you want. To send control messages in a message flow, see "Configuring message flows to send control messages" and "Configuring a broker environment to send control messages" on page 2746.

For a working example of aggregation (without the use of control messages), see the following sample:
- Airline Reservations

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Configuring message flows to send control messages:
About this task

To configure message flows to send control messages from an AggregateControl node to an AggregateReply node:

Procedure

1. If you have created the fan-out and fan-in flows in a single message flow:
   a. Open the aggregation message flow.
   b. Connect the Control terminal of the AggregateControl node to the Control terminal of the AggregateReply node to make the association.

   This connection is referred to as a direct connection between the two aggregation nodes.
2. If you have created separate fan-out and fan-in message flows:

a. Open the fan-out message flow.
b. Configure the AggregateControl node, see "Creating the aggregation fan-out flow" on page 2722.
c. At this stage, you can configure a Compute node that creates a valid output message that contains the control message. For example, to pass the control message to an MQOutput node, configure the Compute node to add an MQMD to the message and complete the required fields in that header. For example, you can code the following ESQL:

```
SET OutputRoot.MQMD.Version = MQMD_CURRENT_VERSION;
SET OutputRoot.MQMD.Format = MQFMT_STRING;
```

d. Configure an output node that represents the intermediate destination for the control message. For example, to send the control message to an intermediate WebSphere MQ queue, include an MQOutput node and identify the target queue in the Basic properties `Queue Manager Name` and `Queue Name`.
e. Connect the Control terminal of the AggregateControl node to the In terminal of the Compute node, and connect the Out terminal of the Compute node to the In terminal of the output node that represents the intermediate destination for the control message.
f. Open the fan-in message flow.
g. Configure one input node to receive the reply messages, see "Creating the aggregation fan-in flow" on page 2728. This input node also receives the control information from the AggregateControl node. For example, set the Basic property `Queue Name` of the MQInput node to receive the response and control message from an intermediate WebSphere MQ queue.
h. Add a Filter node to your fan-in flow after the input node and before the AggregateReply node, see "Avoiding thread starvation on fan-in flows" on page 2748.
i. Connect the Out terminal of the input node to the In terminal of the Filter node.
j. Connect the Out terminals of the Filter node to the Control terminal and in terminal of the AggregateReply node.

This connection is referred to as an indirect connection between the two aggregation nodes.

Configuring a broker environment to send control messages:
About this task

By default, in WebSphere Message Broker Version 7.0, all connections from the Control terminal of the AggregateRequest node to the AggregateReply node are ignored. For these connections to be active, create the MQSI_AGGR_COMPAT_MODE environment variable in the broker environment. By default, the environment variable does not exist. The existence of the environment variable means that connections from the AggregateControl node are active, regardless of the value to which the environment variable is set.

When the MQSI_AGGR_COMPAT_MODE environment variable has not been created, the default behavior for aggregation fan-out flows is used. If the Control terminal of the AggregateControl node is connected, either directly or indirectly, to the In terminal of the AggregateReply node, this connection is ignored and no control message is sent.

If the MQSI_AGGR_COMPAT_MODE environment variable is created, the default behavior for aggregation fan-out flows is not used, allowing you to send control messages from the AggregateControl node to the AggregateReply node. If the Control terminal of the AggregateControl node is connected, either directly or indirectly, to the In terminal of the AggregateReply node, see "Creating the aggregation fan-out flow" on page 2722, this connection is recognized and a control message is sent. Be aware that this configuration is not the optimal configuration and might affect performance.

To create the MQSI_AGGR_COMPAT_MODE variable to support connections between AggregateControl and AggregateReply nodes to be recognized:

- Windows On Windows:
  1. Open System Properties by clicking **Start** > **Control Panel** > **System**.
  2. Click the **Advanced** tab.
  3. Click **Environment Variables**.
  4. In the System variables pane, click **New**.
  5. Under **Variable name** type `MQSI_AGGR_COMPAT_MODE`.
  6. (Optional) You can type in the **Variable value** or leave it blank.
  7. For the environment variable to take effect, restart the computer.
- Linux UNIX z/OS On Linux, UNIX and z/OS:
  1. Edit the profile of the broker userid and include the following code:
     ```
     export MQSI_AGGR_COMPAT_MODE=
     ```
  2. Reload the profile.
  3. Restart the broker.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message flow aggregation" on page 2718
Aggregation is the generation and fan-out of related requests that are derived from a single input message, and the fan-in of the corresponding replies to produce a single aggregated reply message.

Related tasks:

"Configuring aggregation flows" on page 2721
Use aggregation message flows to generate and fan-out a number of related requests, fan-in the corresponding replies, and compile those replies into a single aggregated reply message, by using the AggregateControl, AggregateRequest, and AggregateReply nodes.

"Creating the aggregation fan-out flow" on page 2722
The aggregation fan-out flow receives the initial input message and restructures it to present a number of requests to a number of target applications.

"Associating fan-out and fan-in aggregation flows" on page 2733
Associate the fan-out message flow processing with its corresponding fan-in message flow processing by setting the `Aggregate Name` property of the AggregateControl and AggregateReply nodes in your aggregation flow to the same value.

"Setting timeout values for aggregation" on page 2736
You can use two properties of the aggregation nodes to set timeout values for aggregated message processing.

"Using multiple AggregateControl nodes" on page 2741
You might find it useful to design a fan-out flow with multiple AggregateControl nodes, all with the same value set for the property `Aggregate Name`, but with different values for the `Timeout` property. You can reuse an aggregate name in only this situation.

"Handling exceptions in aggregation flows" on page 2749
When you use aggregation flows, exceptions might occur.

"Avoiding thread starvation on fan-in flows"
Follow this guidance to avoid thread starvation on fan-in flows if the Control terminal of the AggregateControl node in your fan-out flow is connected to output control messages to a queue.

"Processing timed out aggregation messages" on page 2739
Assign additional processing threads to enable processing of timed out aggregation messages in the AggregateReply node.

Related reference:

"AggregateControl node" on page 4296
Use the AggregateControl node to mark the beginning of a fan-out of requests that are part of an aggregation.

"AggregateReply node" on page 4299
Use the AggregateReply node to mark the end of an aggregation fan-in. This node collects replies and combines them into a single compound message.

"AggregateRequest node" on page 4303
Use the AggregateRequest node to record the fact that request messages have been sent. This node also collects information that helps the AggregateReply node to construct the compound response message.

Avoiding thread starvation on fan-in flows:

Follow this guidance to avoid thread starvation on fan-in flows if the Control terminal of the AggregateControl node in your fan-out flow is connected to output control messages to a queue.

About this task

By not connecting the Control terminal, you can overcome the issues that are discussed here. For further information about connecting the Control terminal of the AggregateControl node, see "Using control messages in aggregation flows" on page 2745.

The Aggregate Reply node has two input terminals: In and Control. The use of the Control terminal is optional. If you use both of these terminals, the MQInput nodes that supply the two terminals must not use threads from the message flow additional instance pool. If the nodes do use these threads, they compete for resources, and the Control terminal's MQInput node typically takes all available threads because it is activated before the In terminal.

Configure each MQInput node to use additional instances that are defined on the node, not at the message flow level.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message flow aggregation" on page 2718
Aggregation is the generation and fan-out of related requests that are derived from a single input message, and the fan-in of the corresponding replies to produce a single aggregated reply message.

Related tasks:

"Creating the aggregation fan-in flow" on page 2728
The aggregation fan-in flow receives the responses to the request messages that are sent out by the fan-out flow, and constructs a combined response message containing all the responses received.

"Creating the aggregation fan-out flow" on page 2722
The aggregation fan-out flow receives the initial input message and restructures it to present a number of requests to a number of target applications.

"Associating fan-out and fan-in aggregation flows" on page 2733
Associate the fan-out message flow processing with its corresponding fan-in message flow processing by setting the `Aggregate Name` property of the AggregateControl and AggregateReply nodes in your aggregation flow to the same value.

"Setting timeout values for aggregation" on page 2736
You can use two properties of the aggregation nodes to set timeout values for aggregated message processing.

"Processing timed out aggregation messages" on page 2739
Assign additional processing threads to enable processing of timed out aggregation messages in the AggregateReply node.

"Using multiple AggregateControl nodes" on page 2741
You might find it useful to design a fan-out flow with multiple AggregateControl nodes, all with the same value set for the property `Aggregate Name`, but with different values for the `Timeout` property. You can reuse an aggregate name in only this situation.

"Handling exceptions in aggregation flows"
When you use aggregation flows, exceptions might occur.

"Configuring aggregation flows" on page 2721
Use aggregation message flows to generate and fan-out a number of related requests, fan-in the corresponding replies, and compile those replies into a single aggregated reply message, by using the AggregateControl, AggregateRequest, and AggregateReply nodes.

Related reference:

"AggregateControl node" on page 4296
Use the AggregateControl node to mark the beginning of a fan-out of requests that are part of an aggregation.

"AggregateReply node" on page 4299
Use the AggregateReply node to mark the end of an aggregation fan-in. This node collects replies and combines them into a single compound message.

"AggregateRequest node" on page 4303
Use the AggregateRequest node to record the fact that request messages have been sent. This node also collects information that helps the AggregateReply node to construct the compound response message.

Handling exceptions in aggregation flows:

When you use aggregation flows, exceptions might occur.

Before you begin

Before you start:

Complete the following tasks:
- "Creating the aggregation fan-out flow" on page 2722
- "Creating the aggregation fan-in flow" on page 2728

Dealing with exceptions:
About this task

If an error is detected downstream of an AggregateReply node, the broker issues an exception. Another node in the message flow might also issue an exception using the ESQL THROW statement. In either case, when an exception occurs, it is caught in one of two places:
- The input node on which the replies arrive
- The AggregateReply node

The following table lists events and describes what happens to an exception that occurs downstream of the AggregateReply node.

Event	Message propagated	Output terminal	Exception caught at
An expected reply arrives at the input node and is passed to the In terminal of the AggregateReply node. The reply is the last one that is needed to make an aggregation complete.	An aggregated reply message that contains all the replies	Out	Input node
An unexpected reply arrives at the input node and is passed to the AggregateReply node. The reply is not recognized as a valid reply, and the `Unknown Message Timeout` property is set to `0`.	Message received	Unknown	Input node
A timeout occurs because all the replies for an aggregation have not yet arrived.	An aggregated reply message that contains all the replies that have been received	Timeout	AggregateReply node
An unknown timeout occurs because a retained message is not identified as a valid reply.	Retained message	Unknown	AggregateReply node
An aggregation is discovered to be complete at some time other than when the last reply arrived.	An aggregated reply message that contains all the replies	Out	AggregateReply node

To handle errors that occur in aggregation flows, you must catch these exceptions at all instances of each of these nodes in the message flow.

Procedure

1. Switch to the Broker Application Development perspective.
2. Open the message flow with which you want to work.

3. To handle these exceptions yourself, connect the Catch terminal of each input and AggregateReply node to a sequence of nodes that handles the error that has occurred.

   For a unified approach to error handling, connect the Catch terminals of all these nodes to a single sequence of nodes, or create a subflow that handles errors in a single consistent manner, and connect that subflow to each Catch terminal.
4. If you want the broker to handle these exceptions using default error handling, do not connect the Catch terminals of these nodes.

Results

If you connect the Catch terminal of the AggregateReply node, and want to send the message that is propagated through this terminal to a destination from which it can be retrieved for later processing, include a Compute node in the catch flow to provide any transport-specific processing. For example, you must add an MQMD header if you want to put the message to a WebSphere MQ queue from an MQOutput node.

The following ESQL example shows you how to add an MQMD header and pass on the replies that are received by the AggregateReply node:

```
-- Add MQMD
SET OutputRoot.MQMD.Version = 2;
.
-- Include consolidated replies in the output message
SET OutputRoot.XMLNS.Data.Parsed = InputRoot.ComIbmAggregateReplyBody;
.
```

To propagate the information about the exception in the output message, set the `Compute mode` property of the Compute node to a value that includes `Exception`.

Related concepts:

"Message flow aggregation" on page 2718
Aggregation is the generation and fan-out of related requests that are derived from a single input message, and the fan-in of the corresponding replies to produce a single aggregated reply message.

Related tasks:

"Configuring aggregation flows" on page 2721
Use aggregation message flows to generate and fan-out a number of related requests, fan-in the corresponding replies, and compile those replies into a single aggregated reply message, by using the AggregateControl, AggregateRequest, and AggregateReply nodes.

"Creating the aggregation fan-out flow" on page 2722
The aggregation fan-out flow receives the initial input message and restructures it to present a number of requests to a number of target applications.

"Creating the aggregation fan-in flow" on page 2728
The aggregation fan-in flow receives the responses to the request messages that are sent out by the fan-out flow, and constructs a combined response message containing all the responses received.

"Associating fan-out and fan-in aggregation flows" on page 2733
Associate the fan-out message flow processing with its corresponding fan-in message flow processing by setting the `Aggregate Name` property of the AggregateControl and AggregateReply nodes in your aggregation flow to the same value.

"Setting timeout values for aggregation" on page 2736
You can use two properties of the aggregation nodes to set timeout values for aggregated message processing.

"Using multiple AggregateControl nodes" on page 2741
You might find it useful to design a fan-out flow with multiple AggregateControl nodes, all with the same value set for the property `Aggregate Name`, but with different values for the `Timeout` property. You can reuse an aggregate name in only this situation.

Related reference:

"AggregateControl node" on page 4296
Use the AggregateControl node to mark the beginning of a fan-out of requests that are part of an aggregation.

"AggregateReply node" on page 4299
Use the AggregateReply node to mark the end of an aggregation fan-in. This node collects replies and combines them into a single compound message.

"AggregateRequest node" on page 4303
Use the AggregateRequest node to record the fact that request messages have been sent. This node also collects information that helps the AggregateReply node to construct the compound response message.

Exceptions when dealing with unknown and timeout messages:

When timeout messages or unknown messages from unknown timeout processing are produced from an AggregateReply node, they originate from an internal queue and not from an MQInput node. This behavior affects how the error handling should be performed.

If a message that is sent down the timeout thread causes an exception, the message rolls back to the AggregateReply node and is sent to the Catch terminal. If this terminal is unattached or an exception occurs while processing the message, the timeout is rolled back onto the internal queue and is reprocessed. Potentially, this behavior can lead to an infinite loop, which can be stopped by deploying a version of the message flow that fixes the problem.

To avoid this infinite loop, take the following actions.

- Connect the Catch terminal to a set of nodes that handle errors.
- Ensure that the error-handling nodes cannot throw an exception by ensuring that they perform very simple operations; for example, converting the message to a BLOB, then writing it to a queue, or adding extra TryCatch nodes.

The failure terminal of the AggregateReply node is not used currently and messages are not passed to this terminal.

Configuring the storage of events for aggregation nodes:

You can use an Aggregation configurable service to control the storage of events for AggregateControl and AggregateReply nodes.

About this task

By default, the storage queues used by all aggregation nodes are:

- SYSTEM.BROKER.AGGR.CONTROL
- SYSTEM.BROKER.AGGR.REPLY
- SYSTEM.BROKER.AGGR.REQUEST

- SYSTEM.BROKER.AGGR.UNKNOWN
- SYSTEM.BROKER.AGGR.TIMEOUT

However, you can control the queues that are used by different aggregation nodes by creating alternative queues containing a *QueuePrefix*, and using an Aggregation configurable service to specify the names of those queues for storing events.

Follow these steps to specify the queues that are used to store event states, and to set the expiry time of an aggregation:

Procedure

1. Create the storage queues to be used by the aggregation nodes. The following queues are required:
   - SYSTEM.BROKER.AGGR.*QueuePrefix*.CONTROL
   - SYSTEM.BROKER.AGGR.*QueuePrefix*.REPLY
   - SYSTEM.BROKER.AGGR.*QueuePrefix*.REQUEST
   - SYSTEM.BROKER.AGGR.*QueuePrefix*.UNKNOWN
   - SYSTEM.BROKER.AGGR.*QueuePrefix*.TIMEOUT

   The *QueuePrefix* variable can contain any characters that are valid in a WebSphere MQ queue name, but must be no longer than eight characters and must not begin or end with a period (.). For example, `SET1` and `SET.1` are valid queue prefixes, but `.SET1` and `SET1.` are invalid.

   If you do not create the storage queues, WebSphere Message Broker creates the set of queues when the node is deployed; these queues are based on the default queues. If the queues cannot be created, the message flow is not deployed.
2. Use the **mqsicreateconfigurableservice** command to create an Aggregation configurable service. You can create a configurable service to be used with either a specific aggregation or with all aggregations in an execution group.
   a. If the configurable service is to be used with a specific aggregation, ensure that the name of the configurable service is the same as the name that you specify in the `Aggregate name` property on the AggregateControl and AggregateReply nodes. If the configurable service is to be used with all aggregations in an execution group, create the configurable service with the same name as the execution group.
   b. Set the **Queue prefix** property to the required value.
   c. Optional: Set the **Timeout** property to control the expiry time of an aggregation.

   For example, create a configurable service called `myAggregation`, which specifies queues prefixed with SYSTEM.BROKER.AGGR.SET1 and a timeout of 60 seconds:

   ```
   mqsicreateconfigurableservice MYBROKER -c Aggregation -o myAggregation
   -n queuePrefix,timeoutSeconds -v SET1,60
   ```

   You can use the **mqsideleteconfigurableservice** command to delete the Aggregation configurable service. However, the storage queues are not deleted automatically when the configurable service is deleted, so you must delete them separately.For more information, see "Configurable services properties" on page 3766
3. In the AggregateControl and AggregateReply nodes:
   a. Ensure that the name of the Aggregation configurable service is the same as the name specified in the `Aggregate name` property on the **Basic** tab; for example, `myAggregation`. If no Aggregation configurable service exists with

the same name as the `Aggregate name` property, and if a configurable service exists with the same name as the execution group, that configurable service is used instead.

b. Optional: Use the **mqsichangeproperties** and **mqsireportproperties** commands to change or view the properties of the configurable service. Alternatively, you can use the WebSphere Message Broker Explorer to view or modify a configurable service. For more information about working with configurable services, see "Using the WebSphere Message Broker Explorer to work with configurable services" on page 644.

What to do next

The properties for the configurable service are not used by the broker until you restart or redeploy the message flow, or restart the broker.

Related concepts:

"Using the WebSphere Message Broker Explorer to work with configurable services" on page 644
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view, add, modify and delete configurable services.

Related tasks:

"Viewing configurable services" on page 647
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view the properties of configurable services defined on your broker.

"Modifying an IBM defined configurable service" on page 648
You can create a new configurable service for an external service on which the broker relies, by modifying predefined configurable services provided by IBM. Use the WebSphere Message Broker Explorer to view and modify existing configurable services.

"Modifying a configurable service" on page 649
Use the WebSphere Message Broker Explorer to view and modify existing configurable services.

"Deleting a configurable service" on page 652
Use the WebSphere Message Broker Explorer to delete custom configurable services.

Related reference:

"AggregateControl node" on page 4296
Use the AggregateControl node to mark the beginning of a fan-out of requests that are part of an aggregation.

"AggregateRequest node" on page 4303
Use the AggregateRequest node to record the fact that request messages have been sent. This node also collects information that helps the AggregateReply node to construct the compound response message.

"AggregateReply node" on page 4299
Use the AggregateReply node to mark the end of an aggregation fan-in. This node collects replies and combines them into a single compound message.

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a

broker external resource.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

Using message collections

A message collection is a single message that contains multiple messages derived from one or more sources. You can use a Collector node to group together messages from one or more sources into a message collection, so that they can be processed together by downstream nodes.

About this task

The topics in this section describe how message collection works, and how you can configure your message flow to process message collections.

Message collections

A message collection is a single message that contains multiple messages derived from one or more sources.

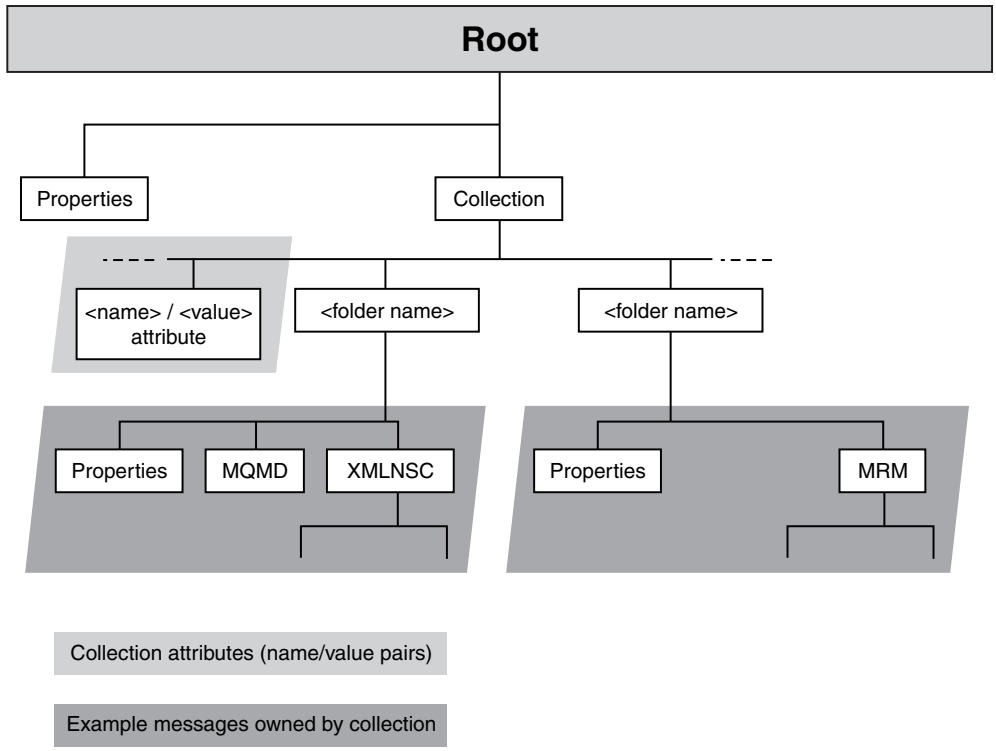
You can use a Collector node to group together messages from one or more sources into a message collection, so that they can be processed together by downstream nodes. You can also manually build a message collection using a Compute node.

Structure of a message collection

A message collection tree contains sub-trees that hold the content of the individual messages received by the Collector node. The message assembly that is propagated from the Collector node to other nodes in your message flow contains the following four components:

- Message (including transport headers)
- Local environment
- Global environment
- Exception list

The following figure shows an example of the message tree structure of a message collection.



The message collection in this example contains two messages, one received from WebSphere MQ, and one from a file input source.

A message collection has a Properties header and a single folder element named Collection. A message collection can also have zero or more attributes that are name/value pairs; the name of an attribute must be unique within a message collection. These are shown as `<name> / <value>` in the figure. A standard attribute for the message collection is an attribute called *`CollectionName`*. If you use a Collector node to generate a message collection, the value for the collection name is generated based on the values you configure in the node. The collection name attribute is not compulsory.

Within the Collection folder in the message collection tree structure are folders, shown as `<folder name>` in the diagram. These folders contain the message tree of each message added to the message collection. Each of these folders has a name, but this name does not have to be unique within the message collection. The value for the `<folder name>` is derived from the source of the input message.

Nested message collections are not permitted. You cannot therefore use a message collection as a source message for another message collection. For example, If you attempt to pass a message collection to a input terminal on a Collector node, an error is generated.

The LocalEnvironment, Environment and ExceptionList trees are not included in the structure, but are instead carried separately as a part of the message assembly. There is no concept of a LocalEnvironment associated with each message in a message collection.

Generating a message collection using a Collector node

You can use the Collector node to make multiple synchronous or asynchronous requests in parallel. The results of these requests can be joined together downstream if required. This is different from the behavior of the aggregation nodes where there is a fixed pattern of request/response and where reply messages are grouped by request id. In contrast, the collector node does not need an initial fan-out stage and can group together unrelated input messages by correlating their content. You can configure dynamic input terminals on a Collector node to receive messages from different sources. You can also configure properties on the Collector node, known as event handlers, to determine how messages are added to a message collection, and when a message collection is complete.

Processing a message collection

A message collection is supported by the following nodes only:

- Compute
- JavaCompute

Errors are generated by other nodes if they receive a message collection.

You can use ESQL or XPath expressions to access the content of messages in a message collection by referencing the folder names preceded by `InputRoot.Collection`. To access the contents of a message in a message collection using ESQL you can use code similar to the following ESQL:

```
InputRoot.Collection.folder1.XMLNSC
```

In XPath, the root element is the body of the message. The root element for a message collection is the Collection element. Therefore, to access the contents of a message in a message collection using XPath, you must use an expression similar to the following XPath:

```
/folder1/XMLNSC
```

Examples of XPath expressions that you can use to access the message collection are:

- `/*`: returns a list of all the messages in the message collection.
- `/@*`: returns a list of all the attributes of the message collection.
- `/@Name`: returns the value of the attribute `Name`.

You might not be able to determine the order of the messages within a message collection. If you generate a message collection using the Collector node, the messages are arranged in the same order as the messages arrived at the node.

Related concepts:

"Message flow nodes" on page 1024
A message flow node is a processing step in a message flow. It can be a built-in node, a user-defined node, or a subflow node.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Creating a message collection by using ESQL"
A message collection can be constructed by using ESQL. Using a message collection is useful if messages must be grouped together for parsing, or if the message collection must be constructed to represent a particular data structure, such as a CICS Transaction Server for z/OS channel data structure.

"Using message collections" on page 2755
A message collection is a single message that contains multiple messages derived from one or more sources. You can use a Collector node to group together messages from one or more sources into a message collection, so that they can be processed together by downstream nodes.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"Collector node" on page 4333
Use the Collector node to create message collections based on rules that you configure in the node.

Creating a message collection by using ESQL

A message collection can be constructed by using ESQL. Using a message collection is useful if messages must be grouped together for parsing, or if the message collection must be constructed to represent a particular data structure, such as a CICS Transaction Server for z/OS channel data structure.

Before you begin

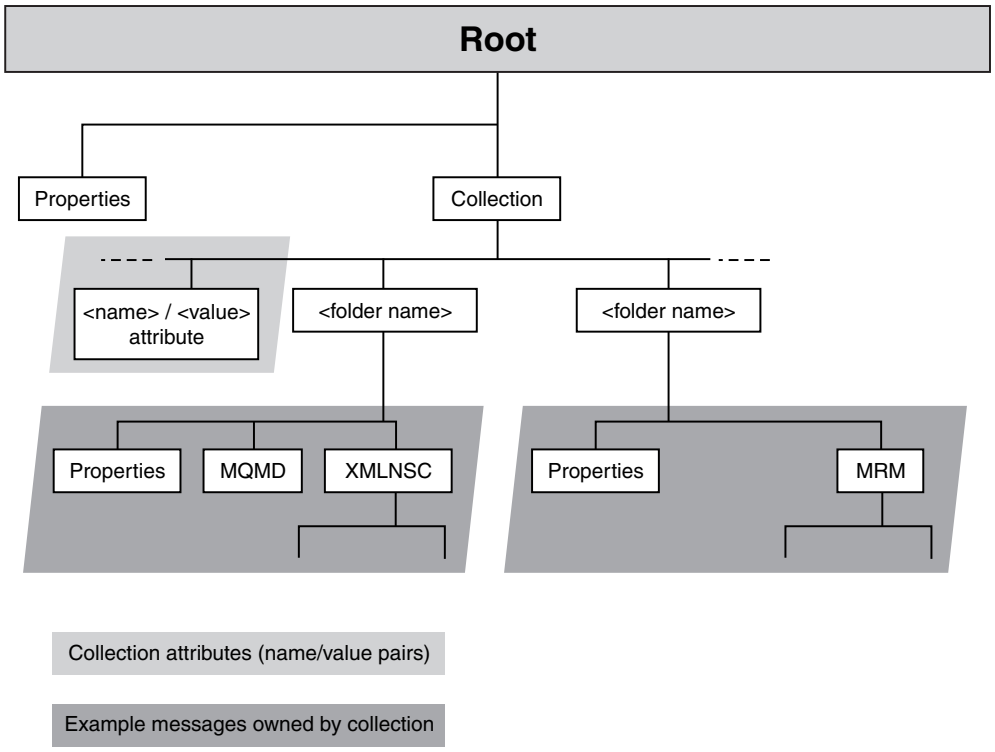
Before you start:

- Create a message flow project and a message set project, as described in "Creating a message flow project" on page 1425 and "Working with a message set project" on page 2838.
- For background information, read "Message collections" on page 2755.

About this task

A message collection is a message that consists of a `Properties` header and a single domain element named `Collection`. The `Collection` folder contains a number of child messages, each of which can contain a `Properties` folder, a number of headers (such as MQMD), and a body. A message collection can also have zero or more attributes that are name/value pairs. The name of an attribute must be unique within a message collection. A standard attribute for the message collection is an attribute called `CollectionName`.

The following figure shows an example of a message collection structure.



You can create a message collection by using ESQL to group messages together for parsing, or create a message collection that must be constructed to represent a particular data structure, such as a CICS channel data structure.

To configure a message collection by using ESQL, complete the following steps:

Procedure

1. Create a `Properties` folder for the collection by using the following ESQL statement:

   ```
   CREATE FIRSTCHILD of OutputRoot domain 'Properties' NAME 'Properties';
   ```

2. Create the `Collection` domain element by using the following statement:

   ```
   CREATE LASTCHILD OF OutputRoot DOMAIN 'Collection';
   ```

   As with message folders, the domain element is always the last child of the message.

3. Use the following statement to set an attribute in the collection called `CollectionName`:

   ```
   SET OutputRoot.Collection.CollectionName = 'myCollectionName';
   ```

4. The following ESQL shows an example procedure to create a message within the collection:

   ```
   SET OutputRoot.Collection.foldername.Properties.MessageSet     = set;
   SET OutputRoot.Collection.foldername.Properties.MessageType    = type;
   SET OutputRoot.Collection.foldername.Properties.MessageFormat  = format;
   SET OutputRoot.Collection.foldername.Properties.Encoding       = encoding;
   SET OutputRoot.Collection.foldername.Properties.CodedCharSetId = ccsid;

   SET OutputRoot.Collection.foldername.domain.content=some data;
   ```

Related concepts:

"Message collections" on page 2755
A message collection is a single message that contains multiple messages derived from one or more sources.

"Message tree structure" on page 1045
The message tree is a part of the logical message tree in which the broker stores its internal representation of the message body.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Using message collections" on page 2755
A message collection is a single message that contains multiple messages derived from one or more sources. You can use a Collector node to group together messages from one or more sources into a message collection, so that they can be processed together by downstream nodes.

"Writing ESQL" on page 2413
How you can use ESQL to customize nodes.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

Creating a message collection by using Java

A message collection can be constructed by using Java and the MbMessageCollection class. Using a message collection is useful if messages must be grouped together for parsing, or if the message collection must be constructed to represent a particular data structure, such as a CICS Transaction Server for z/OS channel data structure.

Before you begin

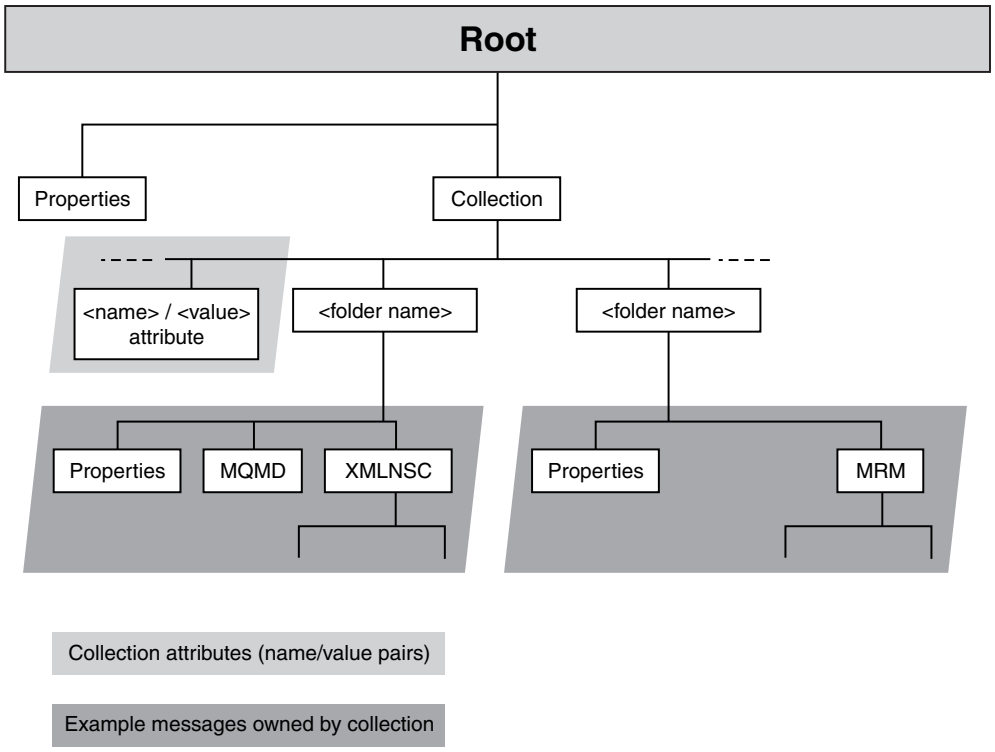
Before you start:

- Create a message flow project and a message set project. For more information, see "Creating a message flow project" on page 1425 and "Working with a message set project" on page 2838.
- For background information, read "Message collections" on page 2755.

About this task

A message collection is a message that consists of a `Properties` header and a single domain element named `Collection`. The `Collection` folder contains a number of child messages, each of which can contain a `Properties` folder, a number of headers (such as MQMD), and a body. A message collection can also have zero or more attributes that are name/value pairs. The name of an attribute must be unique within a message collection. A standard attribute for the message collection is an attribute called *`CollectionName`*.

The following figure shows an example of a message collection structure.



You can create a message collection by using Java, and the MbMessageCollection class, to group messages together for parsing, or create a message collection that must be constructed to represent a particular data structure, such as a CICS channel data structure.

To configure a message collection by using Java, complete the following steps:

Procedure

1. Create a new message by using the following example:

```
// create new message
MbMessageCollection outMessage = new MbMessageCollection();
MbMessageAssembly outAssembly = new MbMessageAssembly(inAssembly,
 outMessage);
```

2. Create a `Properties` folder for the collection by using the following example:

```
// create top level Properties folder and data
MbElement omroot = outMessage.getRootElement();
MbElement properties = omroot.createElementAsFirstChild("Properties");
MbElement property1 = properties.createElementAsLastChild(
 MbElement.TYPE_NAME_VALUE, "myProperty1", "propertyData1");
MbElement property2 = properties.createElementAsLastChild(
 MbElement.TYPE_NAME_VALUE, "myProperty2", "propertyData2");
```

3. Create the name value pairs by using the following example:

```
// create collection attributes (name/value pairs)
MbElement cn  = outMessage.createNameValue("CollectionName", "myCollectionName");
MbElement nv1 = outMessage.createNameValue("NAME1", "Value1");
MbElement nv2 = outMessage.createNameValue("NAME2", 12345);
```

   As with message folders, the domain element is always the last child of the message property.

4. The following example shows the procedure to create a message within the collection. Steps one, two, and three are repeated.

```
public void evaluate(MbMessageAssembly inAssembly) throws MbException {
 MbOutputTerminal out = getOutputTerminal("out");

 // create new message
 MbMessageCollection outMessage = new MbMessageCollection();
 MbMessageAssembly outAssembly = new MbMessageAssembly(inAssembly,
  outMessage);

 // create top level Properties folder and data
 MbElement omroot = outMessage.getRootElement();
 MbElement properties = omroot.createElementAsFirstChild("Properties");
 MbElement property1 = properties.createElementAsLastChild(
  MbElement.TYPE_NAME_VALUE, "myProperty1", "propertyData1");
 MbElement property2 = properties.createElementAsLastChild(
  MbElement.TYPE_NAME_VALUE, "myProperty2", "propertyData2");

 // create collection attributes (name/value pairs)
 MbElement cn  = outMessage.createNameValue("CollectionName", "myCollectionName");
 MbElement nv1 = outMessage.createNameValue("NAME1", "Value1");
 MbElement nv2 = outMessage.createNameValue("NAME2", 12345);

 // create folder 1
 MbElement folder1 = outMessage.createFolder("folder1");

 // create properties for folder 1
 MbElement folder1properties = folder1.createElementAsFirstChild("Properties");
 MbElement folder1property1 = folder1properties.createElementAsLastChild(
  MbElement.TYPE_NAME_VALUE, "myFolder1Property1", "folder1propertyData1");
 MbElement folder1property2 = folder1properties.createElementAsLastChild(
  MbElement.TYPE_NAME_VALUE, "myFolder1Property2", "folder1propertyData2");

 // create body of folder 1
 MbElement mrm = folder1.createElementAsLastChild("MRM");

 // create message domain element of folder 1
 MbElement msg = mrm.createElementAsLastChild(MbElement.TYPE_NAME,
  "msg", null);

 // create data within the message body for folder 1
 MbElement data = msg.createElementAsLastChild(
  MbElement.TYPE_NAME_VALUE, "data", "myData");

 // create folder 2
 MbElement folder2 = outMessage.createFolder("Folder2");

 // create properties for folder 2
 MbElement folder2properties = folder2.createElementAsFirstChild("Properties");
 MbElement folder2property1 = folder2properties.createElementAsLastChild(
  MbElement.TYPE_NAME_VALUE, "myFolder2Property1", "folder2propertyData1");
 MbElement folder2property2 = folder2properties.createElementAsLastChild(
  MbElement.TYPE_NAME_VALUE, "myFolder2Property2", "folder2propertyData2");

 // create body of folder 2
 MbElement xmlnsc = folder2.createElementAsLastChild("XMLNSC");

 // create message domain element of folder 2
 MbElement msg2 = xmlnsc.createElementAsLastChild(
  MbElement.TYPE_NAME, "msg2", null);

 // create data within the message body for folder 2
 MbElement data2 = msg2.createElementAsLastChild(
  MbElement.TYPE_NAME_VALUE, "myData2", "myXMLData");

 try {
```

```
     out.propagate(outAssembly);
    } finally {
     // clear the outMessage even if there's an exception
     outMessage.clearMessage();
    }
   }
```

Results

The following example output from a Trace node shows the message structure built by this Java code:

```
TraceOutput:  Root ( ['MQROOT' : 0xee3a90]
 (0x01000000:Name):Properties = ( ['MQPROPERTYPARSER' : 0xae4370]
  (0x03000000:NameValue):myProperty1 = 'propertyData1' (CHARACTER)
  (0x03000000:NameValue):myProperty2 = 'propertyData2' (CHARACTER)
 )
 (0x01000000:Name):Collection = ( ['COLLECTION' : 0x58d0b08]
  (0x03000000:NameValue):CollectionName = 'myCollectionName' (CHARACTER)
  (0x03000000:NameValue):NAME1          = 'Value1' (CHARACTER)
  (0x03000000:NameValue):NAME2          = 12345 (INTEGER)
  (0x01000000:Name     ):Folder1     = ( ['COLLECTIONFOLDER' : 0xee42e8]
   (0x01000000:Name ):Properties = ( ['MQPROPERTYPARSER' : 0xae39e8]
    (0x03000000:NameValue):myFolder1Property1 = 'folder1propertyData1' (CHARACTER)
    (0x03000000:NameValue):myFolder1Property2 = 'folder1propertyData2' (CHARACTER)
   )
   (0x01000021:Name+):MRM        = ( ['mrm' : 0xdce588]
    (0x01000000:Name):msg = (
     (0x03000000:NameValue):data = 'myData' (CHARACTER)
    )
   )
  )
  (0x01000000:Name     ):Folder2     = ( ['COLLECTIONFOLDER' : 0xee3d58]
   (0x01000000:Name  ):Properties = ( ['MQPROPERTYPARSER' : 0xae4cf8]
    (0x03000000:NameValue):myFolder2Property1 = 'folder2propertyData1' (CHARACTER)
    (0x03000000:NameValue):myFolder2Property2 = 'folder2propertyData2' (CHARACTER)
   )
   (0x01000000:Folder):XMLNSC     = ( ['xmlnsc' : 0xee2188]
    (0x01000000:Folder):msg2 = (
     (0x03000000:PCDataField):myData2 = 'myXMLData' (CHARACTER)
    )
   )
  )
 )
)
```

Related concepts:

"Message collections" on page 2755
A message collection is a single message that contains multiple messages derived from one or more sources.

"Message tree structure" on page 1045
The message tree is a part of the logical message tree in which the broker stores its internal representation of the message body.

Related tasks:

"Using message collections" on page 2755
A message collection is a single message that contains multiple messages derived from one or more sources. You can use a Collector node to group together messages from one or more sources into a message collection, so that they can be processed together by downstream nodes.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

Creating a flow that uses message collections

Use a Collector node in your message flow to group messages from one or more sources into a message collection. You can add dynamic input terminals to your Collector node for each message source that you want to configure for your message flow.

Before you begin

Before you start:

Complete the following tasks.

- For background information, read "Message collections" on page 2755.
- Create a message flow project, as described in "Creating a message flow project" on page 1425.

About this task

You can also use a Compute node to create a message collection by using ESQL, which is useful if messages must be grouped together for parsing, or if the message collection must be constructed to represent a particular data structure, such as a CICS Transaction Server for z/OS channel data structure. For more information about using ESQL to create a message collection, see "Creating a message collection by using ESQL" on page 2758.

Look at the following sample to see how to use the Collector node for message collection:

- Collector Node

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

To create a message flow to generate and process message collections:

Procedure

1. Create a new message flow.
2. Add the input nodes in the editor view. The input nodes receive the messages from which message collections are generated. You can use any of the built-in nodes, or user-defined input nodes. Configure and connect them as described.
   a. Add an input node for each source of input messages for your message flow, for example an MQInput node and a JMSInput node.
   b. Select each input node in turn to display its properties in the Properties view.
   c. Specify the source of input messages for each node. For example, specify the name of a WebSphere MQ queue in the Basic property `Queue Name` from which the MQInput node retrieves messages.
   d. Optional: Specify values for any other properties that you want to configure for each node.

3. Add the Collector node in the editor view. The Collector node receives messages from input nodes or other nodes in the message flow. You must add a dynamic input terminal to the Collector node for each input message source before you can connect the input nodes or any upstream nodes to the Collector node. Configure and connect them as described.
   a. Add a Collector node to your message flow.
   b. Right-click the Collector node and click **Add Input Terminal** to add a new dynamic input terminal to the Collector node. Add a new input terminal for each input source that you plan to add to your message flow; for more information about adding dynamic input see "Adding an input terminal to a Collector node for each input source" on page 2768.
   c. Connect the out terminal of each input node to a different dynamic input terminal of the Collector node. This represents the simplest configuration; if appropriate, you can include other nodes between the input node and the Collector node. For example, you might want to store the request for audit purposes (in a Warehouse node), or add a unique identifier to the message (in a Compute node).
4. Add processing nodes to your message flow. You can process message collections from a Collector node using the following nodes only:
   - Compute
   - JavaCompute

   You must connect either a Compute node or a JavaCompute node to the Collector node in your message flow. Use these nodes to process the message collection and propagate other messages. You can use ESQL or XPATH to access the contents of the individual messages in the message collection for processing. To process a message collection:
   a. Add a Compute node or a JavaCompute node to your message flow.
   b. Code your ESQL or Java to create single output messages from the message collection.
   c. Optional: Specify values for any other properties that you want to configure for this processing node.
   d. Connect the out terminal of the processing node to the in terminal of an output node or other processing node.
   e. Optional: Add other nodes to your message flow for further processing.
5. Include one or more output nodes for your message flow. These can be any of the built-in nodes, or a user-defined output node. An output node cannot process a message collection, therefore ensure that you connect the output node to a processing node that propagates single output messages. To configure an output node:
   a. Select each output node in turn to display its properties in the Properties view.
   b. Specify the destination properties for each node. For example, specify the name of a WebSphere MQ queue in the Basic property `Queue Name` to which the MQOutput node sends messages.
   c. Optional: Specify values for any other properties that you want to configure for each node.
6. Include processing for handling errors and expired message collections:
   a. Optional: Add processing nodes to your message flow to handle expired message collections. Connect these nodes to the Expire terminal of the Collector node.

b. Optional: Add processing or error handling nodes to handle any exceptions in your message flow. Connect these nodes to the Catch terminal of the Collector node

   If an error is detected downstream of the Collector node, the broker throws an exception. The message collection is propagated to the Collector node's Catch terminal. Connect the Catch terminal to a sequence of nodes that handles the errors, to avoid losing any data, and ensure that no further exceptions can be generated during error processing. The node connected to the Catch terminal must be either a Compute node or a JavaCompute node to handle the message collection.

7. Press Ctrl-S or click **File** > **Save** ***name*** on the taskbar menu (where *name* is the name of this message flow) to save the message flow and validate its configuration.

Results

If you want to control when complete message collections are propagated, you must also add additional nodes to your message flow. For more information, see "Using control messages with the Collector node" on page 2779.

What to do next

Next: Configure the Collector node.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message collections" on page 2755
A message collection is a single message that contains multiple messages derived from one or more sources.

Related tasks:

"Creating a message collection by using ESQL" on page 2758
A message collection can be constructed by using ESQL. Using a message collection is useful if messages must be grouped together for parsing, or if the message collection must be constructed to represent a particular data structure, such as a CICS Transaction Server for z/OS channel data structure.

"Using message collections" on page 2755
A message collection is a single message that contains multiple messages derived from one or more sources. You can use a Collector node to group together messages from one or more sources into a message collection, so that they can be processed together by downstream nodes.

"Configuring the Collector node" on page 2767
You can configure the Collector node to determine how messages are added to message collections. You can also use properties on the Collector node to control when message collections are propagated.

Related reference:

"Collector node" on page 4333
Use the Collector node to create message collections based on rules that you configure in the node.

Configuring the Collector node

You can configure the Collector node to determine how messages are added to message collections. You can also use properties on the Collector node to control when message collections are propagated.

Before you begin

Before you start:

This topic assumes that you have already created a message flow that contains a Collector node. For more information, see "Creating a flow that uses message collections" on page 2764.

About this task

Use the following topics to configure the Collector node:

- "Adding an input terminal to a Collector node for each input source" on page 2768
- "Setting event handler properties" on page 2769
- "Setting the collection expiry" on page 2772
- "Setting the collection name" on page 2774
- "Setting the event coordination property" on page 2775
- "Setting the persistence mode property" on page 2776
- "Setting the configurable service property" on page 2778

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Using message collections" on page 2755
A message collection is a single message that contains multiple messages derived from one or more sources. You can use a Collector node to group together messages from one or more sources into a message collection, so that they can be processed together by downstream nodes.

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

Related reference:

"Collector node" on page 4333
Use the Collector node to create message collections based on rules that you configure in the node.

Adding an input terminal to a Collector node for each input source:

Add new dynamic input terminals to the Collector node for all of the sources of messages for your message collections.

Before you begin

Before you start:

This task assumes that you have already created a message flow that contains a Collector node. For more information, see "Creating a flow that uses message collections" on page 2764.

About this task

To add a dynamic input terminal to the Collector node for each message source, complete the following steps.

Procedure

1. Right-click the Collector node and select **Add Input Terminal**.
2. In the dialog box that is displayed, enter a name of your choice for the terminal, and click **OK**. The name that you give to the input terminal is used as the folder name in the message collection.
3. Repeat steps 1 and 2 to add further input terminals.

What to do next

Next:

When you have created all the required input terminals on the Collector node, you can set the event handler properties. For more information see, "Setting event handler properties" on page 2769.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Using message collections" on page 2755
A message collection is a single message that contains multiple messages derived from one or more sources. You can use a Collector node to group together messages from one or more sources into a message collection, so that they can be processed together by downstream nodes.

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

Related reference:

"Collector node" on page 4333
Use the Collector node to create message collections based on rules that you configure in the node.

Setting event handler properties:

You can configure event handler properties for each dynamic input terminal on a Collector node. These event handler properties determine how the messages received by each terminal are added to message collections.

Before you begin

Before you start:

To complete this task, you must have completed the following tasks:
- "Creating a flow that uses message collections" on page 2764
- "Adding an input terminal to a Collector node for each input source" on page 2768

About this task

You can use one or more of the event handler properties to control the way that messages are added to message collections for each input terminal that you added to the Collector node. Incomplete message collections are stored on a WebSphere MQ queue. The message collections are stored in the order that they are generated by the Collector node (first in, first out). Each message collection has an event handler instance for each of the input terminals. The event handler determines whether an incoming message on that terminal is added to a message collection. The event handler instance maintains information about the state of the collection, the number of messages received, the timer, and the correlation string. When a new message is received on an input terminal, it is offered to the event handler for each message collection waiting on the queue in turn. When the message is accepted by one of the event handlers, it is added to the message collection. The accepted message is not offered to any other message collections. If all the event handlers reject the message, it is added to a new message collection, which is added to the end of the queue.

The first message accepted into a collection determines the correlation string for that message collection, if it is configured. Subsequent messages offered to that message collection are only accepted if their correlation string matches that of the collection. The first message accepted by each event handler starts the timeout timer, if it is configured. Each message accepted by each event handler increments the quantity count. An event handler becomes satisfied when the number of messages accepted equals the configured quantity, or when the timeout value is reached. When an event handler is satisfied, the event handler does not accept any more messages. A message collection is complete only when all of the message collection's event handlers are satisfied. The message collection is then ready for propagation.

You can configure the event handler properties by using the **Collection Definition** table, on the **Basic** tab of the Properties view.

To configure the event handler properties on the Collector node:

Procedure

1. Open the message flow with the Collector node.
2. Right-click the Collector node and select **Properties**.
3. Click the **Basic** tab.
4. Use the following instructions to configure the event handler properties that you want to set for each input terminal:
   - If you want to add a set number of messages to each message collection from one or more of the terminals, you must enter a value for `Quantity` in the Collection Definition table. This value is used to specify the number of messages that each configured input terminal accepts to complete a collection. For example, if you have set `Quantity` to wait for 2 messages on three of the input terminals, the message collection is not complete until 2 messages have been received on each of the three input terminals. The complete message collection contains 6 messages, 2 from each of the three terminals. As soon as more than 2 messages are received on one of the input terminals, the next message is added to a new message collection.
     a. In the Collection Definition table, click the row for the selected input terminal within the `Quantity` column.
     b. Enter a value for the number of input messages that you want to add to a message collection. If you select `Zero` or choose not to set this property, there is no limit to the number of messages accepted. In this case the value set on the `Timeout` property must be greater than zero. If you accept the default value of `1`; only one message from the selected terminal is added to a collection.

     You must enter a value for `Quantity` if `Timeout` is not set.
   - If you want to collect messages for a set amount of time before the message collection is propagated you must enter a value for `Timeout`. This value is used to specify the maximum time in seconds for which the selected input terminal accepts messages before completing a message collection. The timeout interval starts when the first message has arrived at the selected terminal. Any subsequent messages are added to the same message collection. When the timeout interval ends, no more messages are added to the message collection from this terminal. When the conditions on all the terminals are satisfied, the message collection is ready for propagation. When the next message reaches the selected input terminal, a new message collection is created and the timeout interval starts again. If a timeout is set on multiple input terminals, each terminal collects messages for the configured amount of time. During the timeout messages from all of the terminals are added to the same message collection.
     a. In the Collection Definition table, click the row for the selected input terminal within the `Timeout` column.
     b. Enter a value for the length of time in seconds that you want to wait to add messages to a message collection. For example, to wait for messages to add to a message collection for an hour, enter a value of `3600`. If you accept the default value `Zero`, timeout is not enabled and there is no limit on the time to wait for messages. In this case the value set on the `Quantity` property must be greater than zero.

     You must enter a value for `Timeout` if `Quantity` is not set.
   - If you want to add messages to different message collections based on the content of the message you must enter an XPath value for the `Correlation path`. This value is used to specify the path in the incoming message from which to extract the correlation string. The correlation string is the value that is extracted by the correlation path. If a correlation pattern is specified, the

correlation string is matched against the correlation pattern. Messages are only accepted into a message collection with the same correlation string. If you specify an asterisk (*) in the name of the message collection, it is replaced by the correlation string.

a. In the Collection Definition table, click the row for the selected input terminal within the `Correlation path` column.

b. Either select a predefined correlation path from the list, or enter your own correlation path using XPath. The correlation path must begin with a correlation name, which can be followed by zero or more path fields. For example, in the following message the correlation string is `xxx` in the name field:

```
<library>
 <name>xxx</name>
 <more>
  ...
 </more>
</library>
```

In this example, the correlation path using XPath is `$Body/library/name`.

The variables `$Root`, `$LocalEnvironment`, and `$Environment` are available to allow the path to start at the roots of the message, local environment, environment trees, and message body.

If the correlation path evaluates to an empty string the unmatched message is added to a default unnamed message collection.

If you define a value for `Correlation path`, you can optionally configure a `Correlation pattern`.

- If you want to match a substring of the message content from the `Correlation path`, you can define a pattern to match in the message by using `Correlation pattern`. The `Correlation pattern` contains a single wildcard character and optional text. The correlation string, used for the name of the message collection, is the part of the substring that matches the wildcard. For example, if the correlation path contains the filename `part1.dat` in a file header, and the correlation pattern is specified as `*.dat`, the correlation string is `part1`.

  If this property is set, only messages that have the same correlation string are added to the same message collection. The first message added to a message collection determines the correlation string that must be matched by all other messages in that message collection.

  a. In the Collection Definition table, click the row for the selected input terminal within the `Correlation pattern` column.

  b. Enter a value for the correlation pattern. The `Correlation pattern` must contain a single wildcard character: *. This wildcard character can optionally be surrounded by other text.

  If the correlation pattern fails to match the wildcard to a substring, the unmatched message is added to a default unnamed message collection.

5. Repeat step 4 on page 2770 for each of the input terminals that you added to your Collector node. You can configure different event handlers for different input sources.

What to do next

Note: Ensure that you set the event handler properties across different terminals carefully to match the expected delivery of messages to the terminals on the Collector node.

You can now configure the collection expiry, see “Setting the collection expiry.”

Related concepts:

“Message flows overview” on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

“Correlation names” on page 1069
A correlation name is a field reference that identifies a well-defined starting point in the logical message tree and is used in field references to describe a standard part of the tree format.

Related tasks:

“Using message collections” on page 2755
A message collection is a single message that contains multiple messages derived from one or more sources. You can use a Collector node to group together messages from one or more sources into a message collection, so that they can be processed together by downstream nodes.

“Designing a message flow” on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

“Creating a message flow” on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

“Defining message flow content” on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

“Using control messages with the Collector node” on page 2779
You can send control messages to the Collector node in order to control how complete message collections are propagated to other nodes in your message flow.

Related reference:

“Collector node” on page 4333
Use the Collector node to create message collections based on rules that you configure in the node.

Setting the collection expiry:

The collection expiry is a property on the Collector node to set a maximum timeout for adding messages to a message collection.

Before you begin

Before you start:

This topic assumes that you have already created a message flow that contains a Collector node. For more information, see “Creating a flow that uses message collections” on page 2764.

About this task

When messages are added to a message collection, the incomplete message collection is stored on a queue. If the message collection's event handlers are not satisfied, the incomplete message collection is stored on the queue indefinitely, and not propagated for further processing. If a Collector node has 2 input terminals,

and one of the terminals stops receiving messages, for example if the source application is not running, there is the potential for the queue of incomplete message collections to grow indefinitely. To ensure that these incomplete message collections are released after an appropriate amount of time, configure the *Collection Expiry* property. You can configure this timeout, as a value in seconds, in the *Collection Expiry* property on the Collector node. The collection expiry timeout starts when the first message is accepted into a message collection. The collection expiry overrides any individual event handler timers. When the collection expiry timeout has passed for a message collection, the incomplete message collection is propagated to the **Expire** terminal. Connect appropriate processing nodes to the **Expire** terminal, to handle any expired message collections in your message flow.

To configure a collection expiry:

Procedure

1. Open the message flow with the Collector node.
2. Right-click the Collector node and select **Properties**.
3. Click the **Basic** tab.
4. In `Collection Expiry`, enter a time in seconds for the collection expiry timeout.

What to do next

Next: Configure the collection name.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Using message collections" on page 2755
A message collection is a single message that contains multiple messages derived from one or more sources. You can use a Collector node to group together messages from one or more sources into a message collection, so that they can be processed together by downstream nodes.

"Using control messages with the Collector node" on page 2779
You can send control messages to the Collector node in order to control how complete message collections are propagated to other nodes in your message flow.

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

Related reference:

"Collector node" on page 4333
Use the Collector node to create message collections based on rules that you configure in the node.

Setting the collection name:

You can set a default name, or use a correlation string, for the name of your message collections, by using the `Collection name` property on the Collector node.

Before you begin

Before you start:

This task assumes that you have already created a message flow that contains a Collector node. For more information, see "Creating a flow that uses message collections" on page 2764.

About this task

Each message collection that is produced by the Collector node has a name. The collection name is the value that is associated with the CollectionName attribute in the message collection tree structure. Each message collection has only one name.

You can either use `Collection name` to set a default name to be used for each message collection, or you can use the event handler properties to create a correlation string to use for the message collection name. You can use the correlation string to generate a unique name for the message collection, based on the content of the input messages. To use the correlation string for the collection name, you must enter the wildcard symbol `*`. If you leave `Collection name` blank, or if it is set to `*` and the value of the correlation string is empty, the CollectionName attribute of the message collection is set to an empty value.

Any `*` characters in the collection name are replaced with the correlation string. The correlation string for each message collection is also copied into the local environment message that is associated with the propagated message collection. The location of the correlation string in the local environment is `Wildcard/WildcardMatch`.

To configure the message collection name:

Procedure

1. Open the message flow that contains the Collector node.
2. Right-click the Collector node and select **Properties**.
3. Click the **Basic** tab.
4. For the `Collection name` property, enter a name for the message collections that are generated by the Collector node. If you have set a value for `Correlation path` on your input terminals, you can use the `*` for the `Collection name` property to substitute the correlation string into the collection name value. Leave the collection name blank if you want to set your message collection name to an empty value.

What to do next

Next: "Setting the event coordination property" on page 2775.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Using message collections" on page 2755
A message collection is a single message that contains multiple messages derived from one or more sources. You can use a Collector node to group together messages from one or more sources into a message collection, so that they can be processed together by downstream nodes.

"Using control messages with the Collector node" on page 2779
You can send control messages to the Collector node in order to control how complete message collections are propagated to other nodes in your message flow.

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

Related reference:

"Collector node" on page 4333
Use the Collector node to create message collections based on rules that you configure in the node.

Setting the event coordination property:

Use the `Event coordination` property for controlling how message collections are propagated from the Collector node.

Before you begin

Before you start:

This topic assumes that you have already created a message flow that contains a Collector node. For more information, see "Creating a flow that uses message collections" on page 2764.

About this task

In addition to the dynamic input terminals that you can add to the Collector node, there is a static input terminal called **Control**. The purpose of this terminal is to allow an external resource to trigger the output from the collector node. Details are controlled through the Event coordination property settings.

Incomplete message collections that have exceeded the value for the *`Collection expiry`* timeout are immediately propagated to the Expire terminal, regardless of how you configure the *`Event coordination`* property.

To configure *`Event coordination`*:

Procedure

1. Open the message flow that contains the Collector node.
2. Right-click the Collector node and select **Properties**.
3. Click the **Advanced** tab.

4. Set the *Event coordination* property on the Collector node. Select from the following options:
   - If you select `Disabled`, messages to the **Control** terminal are ignored and message collections are propagated when they are complete.
   - If you select `All complete collections`, complete message collections are held on a queue. When a message is received on the **Control** terminal, all message collections on the queue are propagated to the **Out** terminal.
   - If you select `First complete collection`, complete message collections are held on a queue. When a message is received on the **Control** terminal, the first message collection on the queue is propagated to the **Out** terminal. If the queue is empty when a message arrives on the **Control** terminal, the next message collection that is completed is propagated to the **Out** terminal.

Results

You have completed configuration of the Collector node.

What to do next

Next: if you have configured your Collector node to use control messages, see "Using control messages with the Collector node" on page 2779.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Using message collections" on page 2755
A message collection is a single message that contains multiple messages derived from one or more sources. You can use a Collector node to group together messages from one or more sources into a message collection, so that they can be processed together by downstream nodes.

"Using control messages with the Collector node" on page 2779
You can send control messages to the Collector node in order to control how complete message collections are propagated to other nodes in your message flow.

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

Related reference:

"Collector node" on page 4333
Use the Collector node to create message collections based on rules that you configure in the node.

Setting the persistence mode property:

Use the `Persistence Mode` property to control whether incomplete message collections are stored persistently on the queues of a Collector node.

Before you begin

Before you start:

This topic assumes that you have already created a message flow that contains a Collector node. For more information, see "Creating a flow that uses message collections" on page 2764.

About this task

The storage of incoming messages and the Collector node state is handled internally by using WebSphere MQ queues. By default, incomplete message collections are stored non-persistently, which means that incomplete message collections persist if you restart your broker, but not if you restart your queue manager.

You can use the `Persistence Mode` property on the Collector node to store incomplete message collections on a queue persistently. If you set the `Persistence Mode` property to `Persistent`, incomplete message collections are not lost if you restart your queue manager. However, if you do set the property to `Persistent`, the overall performance of the Collector node might be affected.

To configure the `Persistence Mode`:

Procedure

1. Open the message flow that contains the Collector node.
2. Right-click the Collector node and select **Properties**.
3. Click the **Advanced** tab.
4. Set the `Persistence Mode` property on the Collector node to one of the following values.
   - If you select `Non Persistent`, messages and collection state are stored by the broker queue manager as non-persistent messages.
   - If you select `Persistent`, messages and collection state are stored by the broker queue manager as persistent messages.

Results

You have completed configuration of the Collector node.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Using message collections" on page 2755
A message collection is a single message that contains multiple messages derived from one or more sources. You can use a Collector node to group together messages from one or more sources into a message collection, so that they can be processed together by downstream nodes.

"Using control messages with the Collector node" on page 2779
You can send control messages to the Collector node in order to control how complete message collections are propagated to other nodes in your message flow.

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

Related reference:

"Collector node" on page 4333
Use the Collector node to create message collections based on rules that you configure in the node.

Setting the configurable service property:

Use the `Configurable service` property on the Collector node to specify a Collector configurable service that sets the values of properties at run time.

Before you begin

Before you start:

This task assumes that you have already created a message flow that contains a Collector node. For more information, see "Creating a flow that uses message collections" on page 2764.

About this task

You can use the `Configurable service` property on the Collector node to specify the name of a Collector configurable service to be used by the node. The Collector configurable service has the following properties:

`queuePrefix`
The identifier used to specify the names of queues in which events and collections are stored.

`collectionExpirySeconds`
The expiry time (in seconds) of a collection. This property overrides the value specified by the `Collection expiry` property on the Collector node.

To configure the `Configurable service` property, complete the following steps.

Procedure

1. Open the message flow that contains the Collector node.
2. Right-click the Collector node and select **Properties**.
3. Click the **Advanced** tab.
4. Specify the name of the Collector configurable service in the `Configurable service` field.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Using message collections" on page 2755
A message collection is a single message that contains multiple messages derived from one or more sources. You can use a Collector node to group together messages from one or more sources into a message collection, so that they can be processed together by downstream nodes.

"Using control messages with the Collector node"
You can send control messages to the Collector node in order to control how complete message collections are propagated to other nodes in your message flow.

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Configuring the storage of events for Collector nodes" on page 755
You can use a Collector configurable service to control the storage of events for Collector nodes.

Related reference:

"Collector node" on page 4333
Use the Collector node to create message collections based on rules that you configure in the node.

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

Using control messages with the Collector node

You can send control messages to the Collector node in order to control how complete message collections are propagated to other nodes in your message flow.

Before you begin

Before you start:

This task assumes that you have already created a message flow that contains a Collector node. For more information, see "Creating a flow that uses message collections" on page 2764.

About this task

You can control when complete message collections are propagated to other nodes for processing, using messages sent to the **Control** terminal. The exact behavior depends on the settings that you have chosen for the *Event coordination* property on the Collector node. If you want to use control messages to propagate completed messages collections you must set the *Event coordination* property to one of the following:

- `All complete collections`
- `First complete collection`

In these cases, the complete message collections are held on a queue until a control message is received. If you set *Event coordination* to `All complete collections`, all the message collections held on the queue are propagated to the **Out** terminal. If you set *Event coordination* to `First complete collection`, only the first message collection on the queue is propagated to the **Out** terminal. If there are no complete message collections on the queue, the next message collection to complete is immediately propagated to the **Out** terminal.

Incomplete message collections that have exceeded the value for *Collection expiry* are immediately propagated to the **Expire** terminal regardless of the setting of *Event coordination*.

If you want to propagate any complete message collections after a set amount of time for further processing, connect a TimeoutNotification node to the **Control** terminal of the Collector node. You can use the TimeoutNotification node to send a control message to propagate the message collections to ensure that messages are processing within a reasonable time, or to schedule processing tasks.

For more information about driving a message flow using the TimeoutNotification node, see "Automatically generating messages to drive a flow" on page 2817.

Alternatively, you can propagate complete message collections using a message from another application or message flow by connecting an input node to the **Control** terminal of the Collector node.

You can send any message to the **Control** terminal of the Collector node. The message received on the **Control** terminal is not examined by the broker and is discarded on receipt.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message collections" on page 2755
A message collection is a single message that contains multiple messages derived from one or more sources.

Related tasks:

"Using message collections" on page 2755
A message collection is a single message that contains multiple messages derived from one or more sources. You can use a Collector node to group together messages from one or more sources into a message collection, so that they can be processed together by downstream nodes.

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

Related reference:

"Collector node" on page 4333
Use the Collector node to create message collections based on rules that you configure in the node.

Configuring the storage of events for Collector nodes

You can use a Collector configurable service to control the storage of events for Collector nodes.

About this task

By default, the storage queues used by all Collector nodes are:

- SYSTEM.BROKER.EDA.EVENTS
- SYSTEM.BROKER.EDA.COLLECTIONS

These queues are also used by the Resequence node.

However, you can control the queues that are used by different Collector nodes by creating alternative queues that contain a *QueuePrefix* variable, and by using a Collector configurable service to specify the names of those queues for storing events.

Follow these steps to specify the queues that are used to store event states, and to set the expiry for the collection:

Procedure

1. Create the storage queues to be used by the Collector node. The following queues are required:
   - SYSTEM.BROKER.EDA.*QueuePrefix*.EVENTS
   - SYSTEM.BROKER.EDA.*QueuePrefix*.COLLECTIONS

   The *QueuePrefix* variable can contain any characters that are valid in a WebSphere MQ queue name, but must be no longer than eight characters and must not begin or end with a period (.). For example, `SET1` and `SET.1` are valid queue prefixes, but `.SET1` and `SET1.` are invalid.

   If you do not create the storage queues, WebSphere Message Broker creates the set of queues when the node is deployed; these queues are based on the default queues. If the queues cannot be created, the message flow is not deployed.
2. Use the **mqsicreateconfigurableservice** command to create a Collector configurable service. You can create a configurable service to be used with either a specific collection or with all collections in an execution group.
   a. If you are creating a configurable service to be used with a specific collection, ensure that the name of the configurable service is the same as the name that you specify in the `Configurable service` property on the Collector node. If you are creating a configurable service to be used with all collections in the execution group, ensure that the configurable service has the same name as the execution group.
   b. Set the **Queue prefix** property to the required value.
   c. Optional: Set the **Collection expiry** property.

   For example, create a Collector configurable service called `myCollectorService`, which uses queues prefixed with SYSTEM.BROKER.EDA.SET1, and with a collection expiry of 60 seconds:

```
mqsicreateconfigurableservice MYBROKER -c Collector -o myCollectorService
-n queuePrefix,collectionExpirySeconds -v SET1,60
```

You can use the **mqsideleteconfigurableservice** command to delete the Collector configurable service. However, the storage queues are not deleted automatically when the configurable service is deleted, so you must delete them separately.

For more information, see "Configurable services properties" on page 3766

3. In the Collector node:
   a. If the configurable service is to be used for a specific collection, specify the name of the configurable service in the `Configurable service` property on the **Advanced** tab; for example, `myCollectorService`. If you do not set the `Configurable service` property, and if a configurable service exists with the same name as the execution group, that configurable service is used instead.
   b. Optional: Use the **mqsichangeproperties** and **mqsireportproperties** commands to change or view the properties of the configurable service. Alternatively, you can use the WebSphere Message Broker Explorer to view or modify a configurable service. For more information about working with configurable services, see "Using the WebSphere Message Broker Explorer to work with configurable services" on page 644.

What to do next

The properties for the configurable service are not used by the broker until you restart or redeploy the message flow, or restart the broker.

Related concepts:

"Using the WebSphere Message Broker Explorer to work with configurable services" on page 644
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view, add, modify and delete configurable services.

Related tasks:

"Viewing configurable services" on page 647
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view the properties of configurable services defined on your broker.

"Modifying an IBM defined configurable service" on page 648
You can create a new configurable service for an external service on which the broker relies, by modifying predefined configurable services provided by IBM. Use the WebSphere Message Broker Explorer to view and modify existing configurable services.

"Modifying a configurable service" on page 649
Use the WebSphere Message Broker Explorer to view and modify existing configurable services.

"Deleting a configurable service" on page 652
Use the WebSphere Message Broker Explorer to delete custom configurable services.

Related reference:

"Collector node" on page 4333
Use the Collector node to create message collections based on rules that you configure in the node.

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

Using message sequences

You can maintain or change the sequence of messages in a message flow based on the sequence number and group ID contained in each message.

Before you begin

Before you start:

Read the following concept topics, which contain information that you need to understand before you can use message sequences:

- "Message sequencing" on page 2784
- "Sequence groups" on page 2786
- "Starting a message sequence" on page 2787
- "Ending a message sequence" on page 2790
- "Duplicate message processing" on page 2793

About this task

WebSphere Message Broker enables you to configure the sequence of messages in a message flow by using the following nodes:

- "Sequence node" on page 4736
- "Resequence node" on page 4651

The following topics describe the tasks involved in configuring message sequences:

- "Adding sequence numbers to messages" on page 2794
- "Reordering messages in a message flow" on page 2797
- "Maintaining the sequential order of messages" on page 2799
- "Handling missing messages" on page 2806
- "Configuring the storage of events for Resequence nodes" on page 758

These topics provide examples of how to use a Resequence node, and show how the node processes messages in a message flow:

- "Message sequencing scenario 1" on page 2800
- "Message sequencing scenario 2" on page 2803

Related concepts:

"Message sequencing" on page 2784
Use message sequencing to ensure that messages are delivered to the receiving application in a particular order.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"Resequence node" on page 4651
Use the Resequence node to control the sequence in which a group (or groups) of incoming messages are propagated in a message flow.

"Sequence node" on page 4736
Use the Sequence node to add a sequence number to one or more groups of input messages.

Message sequencing

Use message sequencing to ensure that messages are delivered to the receiving application in a particular order.

WebSphere Message Broker provides support for adding sequence numbers to messages, and for reordering messages in the message flow based on their sequence number. Messages can arrive in any order and you can use the Sequence and Resequence nodes to reorder the messages into the required sequence.

In some applications, the ability to process messages in a specific order is important for maintaining the integrity of the workflow. For example, a series of debits and credits against a bank account must be processed in the order in which they took place, and patient records that are received, processed, and forwarded must be sent on in the order in which they arrived.

Messages arriving in the message flow might or might not contain sequence numbers. If messages without sequence numbers are received from an input source, you can preserve the order in which the messages are received by using a Sequence node to generate a monotonically increasing sequence number for each message in the sequence group. When each message arrives at the Sequence node, the sequence number is incremented and stored with the message in a location specified on the node. The current sequence numbers for each active sequence group are stored on the following WebSphere MQ queues:

- SYSTEM.BROKER.SEQ.GROUP
- SYSTEM.BROKER.SEQ.NUMBER

For more information, see "Sequence node" on page 4736.

When the input messages contain sequence numbers, whether they were added by the Sequence node or already defined in an integer field in the message, you can use a Resequence node to change the order of the messages in the message flow.

When messages arrive at the Resequence node, they are held in a storage queue until all previous messages in the sequence have been propagated and committed. When each message becomes the next one in the sequence, it is taken off the queue and propagated down the Out terminal. This sequence of events ensures that messages are kept in the correct order even when message processing fails.

By default, the storage queues used by the Resequence node are:

- SYSTEM.BROKER.EDA.COLLECTIONS
- SYSTEM.BROKER.EDA.EVENTS

However, you can use a Resequence configurable service to specify alternative queues to be used by the Resequence node. For more information about using the Resequence configurable service, see “Configuring the storage of events for Resequence nodes” on page 758.

You can configure the Resequence node to time out if a message in the sequence fails to arrive in a specified period of time, and you can specify how subsequent messages are processed if a message is missing. For example, you can configure the node so that:

- The message sequence must always be maintained
- Gaps are allowed in the message sequence but all other messages must remain in sequence
- Occasional out-of-sequence messages are allowed

For information about how to configure the Resequence node for these scenarios, see “Handling missing messages” on page 2806.

You can divide messages into sequence groups, which can be processed independently, allowing multiple sequences to be processed at the same time. For more information about sequence groups and duplicate message processing, see “Sequence groups” on page 2786.

For more information about the way in which the beginning and end of sequences are controlled, see “Starting a message sequence” on page 2787.

Related tasks:

“Using message sequences” on page 2783
You can maintain or change the sequence of messages in a message flow based on the sequence number and group ID contained in each message.

“Adding sequence numbers to messages” on page 2794
You can add sequence numbers to messages entering a message flow by using the Sequence node.

“Reordering messages in a message flow” on page 2797
When messages entering a message flow contain sequence numbers and a group ID, you can use the Resequence node to re-establish the sequential order of the messages before propagating them through the message flow.

“Configuring the storage of events for Resequence nodes” on page 758
You can use a Resequence configurable service to control the storage of events for Resequence nodes.

“Handling missing messages” on page 2806
You can configure the Resequence node to control how missing messages in a sequence are processed.

Chapter 9, “Developing message flow applications,” on page 1019
Develop message flows to process your business messages and data.

Related reference:

“Resequence node” on page 4651
Use the Resequence node to control the sequence in which a group (or groups) of incoming messages are propagated in a message flow.

“Sequence node” on page 4736
Use the Sequence node to add a sequence number to one or more groups of input messages.

“Collector node” on page 4333
Use the Collector node to create message collections based on rules that you

configure in the node.

Sequence groups

Use sequence groups to control the way in which messages are grouped together for processing by Sequence and Resequence nodes.

By default, all messages arriving at the Sequence and Resequence nodes are ordered as part of a single sequence group. However, you can divide messages into multiple sequence groups, based on a sequence group identifier in the message, and order each sequence group independently. For example, you might have a message flow that receives, processes, and forwards patient records. It is important that the order of records is maintained for each individual patient, but ordering between patients is not necessary. In this case, the sequence group identifier would be the name or ID of the patient.

The group to which a message belongs is determined by the group identifier that is specified in the message. You can use the **Path to sequence group identifier** property on the Sequence and Resequence nodes to specify the location of the sequence group identifier in the message.

Messages that have the same group identifier are considered part of the same sequence group. If no sequence group is specified, a single default sequence group is used for all messages. However, if the **Path to sequence group identifier** property specifies a location in the message that does not exist, an error occurs.

Each sequence group can be associated with only one Sequence node. Multiple nodes can have a sequence group with the same name, but each of those sequence groups is associated with only one node and is separate from other groups with the same name on different nodes. For example, SequenceNode1 might have a sequence group called GroupA, and SequenceNode2 might also have a sequence group called GroupA, but they are separate groups.

Although you can reuse sequence groups when they have been closed, there is a risk that two occurrences of the same sequence group could overlap, with unpredictable results.

For example, if you have a sequence group including the numbers 1-10 and the group is used twice in close succession, it is possible for the second occurrence of sequence number *1* to arrive before sequence number *10* of the first occurrence. If this happens, a duplicate message exception occurs. For this reason, it is advisable to use a group name for only one set of sequence numbers, rather than reusing it in a Resequence node. If you do decide to reuse a sequence group, ensure that you reuse it only when you can be certain that the preceding use has been finished for a significant amount of time.

Even if all the numbers have been received by the first occurrence of the group, it can be difficult to know for certain exactly when the group is closed because it closes only when the final message has completely finished processing; this includes any processing that is required downstream of the node. Any message from the second occurrence fails as a duplicate unless all processing is complete. When a sequence group has started overlapping, it is very difficult to recover all the messages (in both uses of the group) in the correct order, although no messages are lost).

Related concepts:

"Message sequencing" on page 2784
Use message sequencing to ensure that messages are delivered to the receiving

application in a particular order.

"Starting a message sequence"
The start of a message sequence is determined by the `Start of sequence definition` property on the Resequence and Sequence nodes.

"Duplicate message processing" on page 2793
Each sequence number within a sequence group must be unique.

Related tasks:

"Using message sequences" on page 2783
You can maintain or change the sequence of messages in a message flow based on the sequence number and group ID contained in each message.

"Adding sequence numbers to messages" on page 2794
You can add sequence numbers to messages entering a message flow by using the Sequence node.

"Reordering messages in a message flow" on page 2797
When messages entering a message flow contain sequence numbers and a group ID, you can use the Resequence node to re-establish the sequential order of the messages before propagating them through the message flow.

"Configuring the storage of events for Resequence nodes" on page 758
You can use a Resequence configurable service to control the storage of events for Resequence nodes.

"Handling missing messages" on page 2806
You can configure the Resequence node to control how missing messages in a sequence are processed.

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"Resequence node" on page 4651
Use the Resequence node to control the sequence in which a group (or groups) of incoming messages are propagated in a message flow.

"Sequence node" on page 4736
Use the Sequence node to add a sequence number to one or more groups of input messages.

"Collector node" on page 4333
Use the Collector node to create message collections based on rules that you configure in the node.

Starting a message sequence

The start of a message sequence is determined by the `Start of sequence definition` property on the Resequence and Sequence nodes.

Using a Resequence node

When you use the Resequence node to reorder messages in a message flow, you use the `Start of sequence definition` property on the Resequence node to define how the reordered message sequence will start. You can specify the starting sequence number in one of the following ways:

As a literal number

Select `Literal` to specify a literal sequence number, which can be any positive or negative numeric value in the range -9223372036854775807 to 9223372036854775807. When a message with the specified sequence number arrives, it is identified as the first message in the sequence and the messages are propagated.

Using the smallest number received

Select `Automatic` and specify the length of time (in seconds) during which the node collects messages, before it identifies the message that contains the smallest sequence number. When the smallest number has been determined, that sequence number becomes the first in the message sequence.

For example, assume that you have a Resequence node with the following properties:

- `Path to sequence number` property with a value of `/doc/seq`
- `Path to sequence group identifier` property with a value of `/doc/grp`
- `Start of sequence definition` property set to `Automatic` with a value of `5`. This value means that, for any new group, the Resequence node collects messages for 5 seconds before determining the starting sequence number.
- `End of sequence definition` property set to `Automatic` with a value of `60`. This value means that, for any new group, the Resequence node waits for 60 seconds before determining the ending sequence number.

The following messages are received by the Resequence node:

```
<doc><grp>a<grp><seq>5</seq></doc>
<doc><grp>a<grp><seq>4</seq></doc>
<doc><grp>a<grp><seq>3</seq></doc>
<doc><grp>a<grp><seq>2</seq></doc>
<doc><grp>b<grp><seq>0</seq></doc>
<doc><grp>b<grp><seq>2</seq></doc>
```

At this point, the automatic period for the start of sequence expires (5 seconds), then the following messages are received:

```
<doc><grp>a<grp><seq>6</seq></doc>
<doc><grp>b<grp><seq>3</seq></doc>
```

For group *a*, the following messages are propagated to the Out terminal after 5 seconds:

```
<doc><grp>a<grp><seq>2</seq></doc>
<doc><grp>a<grp><seq>3</seq></doc>
<doc><grp>a<grp><seq>4</seq></doc>
<doc><grp>a<grp><seq>5</seq></doc>
<doc><grp>a<grp><seq>6</seq></doc>
```

For group *b*, the following message is propagated to the Out terminal after 5 seconds:

```
<doc><grp>b<grp><seq>0</seq></doc>
```

No more messages are received before a missing message timeout occurs, at which point the following messages are propagated to the Expire terminal:

```
<doc><grp>b<grp><seq>2</seq></doc>
<doc><grp>b<grp><seq>3</seq></doc>
```

Using predicate set on the Resequence node

Select `Predicate` and specify an XPath expression to calculate whether the message is the first in the sequence. The predicate evaluates to either True or False, and messages continue to be collected while the expression evaluates to False. When the expression of a message is evaluated to True, it indicates that the message is the first in the sequence.

For example, you might specify the following XPath expression:

```
/Employee/EmpStartSeq="10"
```

When the input message field *EmpStartSeq* contains the value *10*, the start of sequence predicate is evaluated to True, and the message is identified as the first in the sequence:

```
<Employee>
 <EmpStartSeq>10</EmpStartSeq>
</Employee>
```

Typically, the XPath expression evaluates to a Boolean; however, if other data types are returned, the predicate is determined in the following way:

Table 29.

Returned data type	True	False
Boolean	True	False
Numeric	Any non-zero value	0 or 0.0
String	Any string matching `true` (case-insensitive)	Any string not matching `true` (case insensitive)
NodeSet	Never	Always

When a message evaluates the expression to True (and is therefore identified as the start of the sequence), the node checks that the message has the smallest sequence number collected up to that point. If messages are found with lower sequence numbers, an exception is thrown.

When the first message that evaluates to true has been processed successfully, the XPath expressions of subsequent messages are not checked. If a message arrives with a lower sequence number than the message that was identified as the start of the sequence, an exception is thrown.

Using a Sequence node

When you use the Sequence node to add sequence numbers to messages in the message flow, you use the `Start of sequence definition` property to specify a literal number that is to be used as the starting sequence number. The value can be any positive or negative integer in the range -9223372036854775807 to 9223372036854775807.

The Sequence node allocates a monotonically increasing sequence number for each input message that arrives at the node, starting with the sequence number that you define in the `Start of sequence definition` property. However, this value can be overridden by the value of the *StartOfSequenceNumber* field in the *LocalEnvironment* of the incoming message. For example:
`InputLocalEnvironment.Sequence.StartOfSequenceNumber = 10`.

Related tasks:

"Using message sequences" on page 2783
You can maintain or change the sequence of messages in a message flow based on the sequence number and group ID contained in each message.

"Adding sequence numbers to messages" on page 2794
You can add sequence numbers to messages entering a message flow by using the Sequence node.

"Reordering messages in a message flow" on page 2797
When messages entering a message flow contain sequence numbers and a group ID, you can use the Resequence node to re-establish the sequential order of the messages before propagating them through the message flow.

"Configuring the storage of events for Resequence nodes" on page 758
You can use a Resequence configurable service to control the storage of events for Resequence nodes.

"Handling missing messages" on page 2806
You can configure the Resequence node to control how missing messages in a sequence are processed.

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"Resequence node" on page 4651
Use the Resequence node to control the sequence in which a group (or groups) of incoming messages are propagated in a message flow.

"Sequence node" on page 4736
Use the Sequence node to add a sequence number to one or more groups of input messages.

"Collector node" on page 4333
Use the Collector node to create message collections based on rules that you configure in the node.

Ending a message sequence

The end of a message sequence is determined by the `End of sequence definition` property on the Resequence and Sequence nodes.

Using a Resequence node

When you use the Resequence node to reorder messages in a message flow, you use the `End of sequence definition` property on the Resequence node to define how the reordered message sequence will end. You can specify the end sequence number in the Resequence node in one of the following ways:

As a literal number

Select `Literal` to specify a literal sequence number as the end of the sequence. This value can be any positive or negative numeric value in the range -9223372036854775807 to 9223372036854775807. When a message with the specified sequence number arrives, the sequence group is closed. If there are any messages missing from the sequence, the sequence group remains open for the period of time specified by the `Missing message timeout` property.

For example, assume that you have a Resequence node with the following properties:

- `Path to sequence number` property with a value of `/doc/seq`
- `Path to sequence group identifier` property with a value of `/doc/grp`
- `Start of sequence definition` property set to `Literal` with a value of `0`.
- `End of sequence definition` property set to `Literal` with a value of `6`. This value means that the group will be closed when a message with the sequence number 6 is received and when all earlier messages in the sequence have been received (or the missing message timeout expires).

The following messages are received by the Resequence node:

```
<doc><grp>a<grp><seq>6</seq></doc>
<doc><grp>a<grp><seq>5</seq></doc>
<doc><grp>a<grp><seq>4</seq></doc>
```

```
<doc><grp>a<grp><seq>3</seq></doc>
<doc><grp>a<grp><seq>2</seq></doc>
<doc><grp>a<grp><seq>1</seq></doc>
<doc><grp>a<grp><seq>0</seq></doc>
```

The following messages are propagated to the Out terminal:

```
<doc><grp>a<grp><seq>0</seq></doc>
<doc><grp>a<grp><seq>1</seq></doc>
<doc><grp>a<grp><seq>2</seq></doc>
<doc><grp>a<grp><seq>3</seq></doc>
<doc><grp>a<grp><seq>4</seq></doc>
<doc><grp>a<grp><seq>5</seq></doc>
<doc><grp>a<grp><seq>6</seq></doc>
```

When the sequence group has closed, any further messages arriving at the node for that sequence group are processed as part of a new instance of the group.

By an automatic timeout

Select `Automatic` and specify the length of time (in seconds) during which the node waits for new messages to arrive onto an empty queue. Each time the queue of messages waiting to be propagated is empty, the Resequence node starts a timer, which expires after the specified number of seconds. If no new messages arrive within the specified time, the sequence group is closed, and any new messages that arrive subsequently are treated as part of a new group. If new messages arrive at the node within the specified time limit, the timer is reset.

Using predicate set on the Resequence node

Select `Predicate` and specify an XPath expression to calculate whether the message is the last in the sequence. The predicate evaluates to either True or False, and messages continue to be collected while the expression evaluates to False. When the expression of a message is evaluated to True, it indicates that the message is the last in the sequence and the sequence group is closed. However, if earlier messages are missing from the sequence group, the sequence group remains open for the period of time specified by the `Missing message timeout` property.

When the sequence group has closed, any further messages arriving at the node for that sequence group are processed as part of a new instance of the group.

Typically, the XPath expression evaluates to a Boolean; however, if other data types are returned, the predicate is determined in the following way:

Table 30.

Returned data type	True	False
Boolean	True	False
Numeric	Any non-zero value	0 or 0.0
String	Any string matching `true` (case-insensitive)	Any string not matching `true` (case insensitive)
NodeSet	Never	Always

When a message evaluates the expression to True (and is therefore identified as the last message in the sequence), the node checks that the message has the highest sequence number collected up to that point. If messages are found with higher sequence numbers, an exception is thrown.

Using a Sequence node

When you use the Sequence node to add sequence numbers to messages, you use the `End of sequence definition` property to define how the message sequence will end. The Sequence node allocates a monotonically increasing sequence number for each input message that arrives at the node, ending with the sequence number that you define in the `End of sequence definition` property.

You can specify the end sequence number in the Sequence node in one of the following ways:

As a literal number
: Select `Literal` and specify a positive or negative numeric value as the end of sequence number (for example, 15). The value must be in the range -9223372036854775807 to 9223372036854775807. The monotonically increasing sequence ends when it reaches the end of sequence number specified by this property.

Using predicate set on the Sequence node
: Select `Predicate` to specify that the sequence ends when the XPath expression evaluates to True. For more information, see the table above.

By an automatic timeout
: Select `Automatic` and specify the length of time (in seconds) during which the node waits for new messages after a message for the sequence group has been propagated. If a new message arrives, the timer is canceled and reset when the message is propagated. If no new messages arrive within the specified time, the sequence group is closed, and any new messages that arrive subsequently are treated as part of a new group.

Related tasks:

"Using message sequences" on page 2783
You can maintain or change the sequence of messages in a message flow based on the sequence number and group ID contained in each message.

"Adding sequence numbers to messages" on page 2794
You can add sequence numbers to messages entering a message flow by using the Sequence node.

"Reordering messages in a message flow" on page 2797
When messages entering a message flow contain sequence numbers and a group ID, you can use the Resequence node to re-establish the sequential order of the messages before propagating them through the message flow.

"Configuring the storage of events for Resequence nodes" on page 758
You can use a Resequence configurable service to control the storage of events for Resequence nodes.

"Handling missing messages" on page 2806
You can configure the Resequence node to control how missing messages in a sequence are processed.

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"Resequence node" on page 4651
Use the Resequence node to control the sequence in which a group (or groups) of incoming messages are propagated in a message flow.

"Sequence node" on page 4736
Use the Sequence node to add a sequence number to one or more groups of input messages.

"Collector node" on page 4333
Use the Collector node to create message collections based on rules that you configure in the node.

Duplicate message processing

Each sequence number within a sequence group must be unique.

If a Resequence node receives a message containing a sequence number that has already been processed in the current sequence group, an exception is thrown (BIP4821) and the duplicate message is propagated to the Failure terminal. An exception occurs only if the duplicate message arrives after the first message with that sequence number has arrived and before the sequence group closes.

For example, assume that you have a Resequence node with the following properties:

- `Path to sequence number` property with a value of `/doc/seq`
- `Path to sequence group identifier` property with a value of `/doc/grp`
- `Start of sequence definition` property set to `Literal` with a value of `1`
- `End of sequence definition` property set to `Literal` with a value of `3`

The following messages arrive at the Resequence node:

```
<doc><grp>a<grp><seq>1</seq></doc>
<doc><grp>a<grp><seq>2</seq></doc>
<doc><grp>a<grp><seq>2</seq>/doc>
<doc><grp>a<grp><seq>3</seq></doc>
<doc><grp>a<grp><seq>2</seq></doc>
<doc><grp>a<grp><seq>1</seq></doc>
<doc><grp>a<grp><seq>3</seq></doc>
```

The following messages are propagated to the Out terminal:

```
<doc><grp>a<grp><seq>1</seq></doc>
<doc><grp>a<grp><seq>2</seq></doc>
<doc><grp>a<grp><seq>3</seq></doc>
<doc><grp>a<grp><seq>1</seq></doc>
<doc><grp>a<grp><seq>2</seq>/doc>
<doc><grp>a<grp><seq>3</seq></doc>
```

The second occurrence of the input message with sequence number 2 causes an exception to be thrown and is propagated to the Failure terminal:

```
<doc><grp>a<grp><seq>2</seq>/doc>
```

Related concepts:

"Sequence groups" on page 2786
Use sequence groups to control the way in which messages are grouped together for processing by Sequence and Resequence nodes.

"Message sequencing" on page 2784
Use message sequencing to ensure that messages are delivered to the receiving application in a particular order.

"Starting a message sequence" on page 2787
The start of a message sequence is determined by the `Start of sequence definition` property on the Resequence and Sequence nodes.

Related tasks:

"Using message sequences" on page 2783
You can maintain or change the sequence of messages in a message flow based on the sequence number and group ID contained in each message.

"Adding sequence numbers to messages"
You can add sequence numbers to messages entering a message flow by using the Sequence node.

"Reordering messages in a message flow" on page 2797
When messages entering a message flow contain sequence numbers and a group ID, you can use the Resequence node to re-establish the sequential order of the messages before propagating them through the message flow.

"Configuring the storage of events for Resequence nodes" on page 758
You can use a Resequence configurable service to control the storage of events for Resequence nodes.

"Handling missing messages" on page 2806
You can configure the Resequence node to control how missing messages in a sequence are processed.

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"Resequence node" on page 4651
Use the Resequence node to control the sequence in which a group (or groups) of incoming messages are propagated in a message flow.

"Sequence node" on page 4736
Use the Sequence node to add a sequence number to one or more groups of input messages.

"Collector node" on page 4333
Use the Collector node to create message collections based on rules that you configure in the node.

Adding sequence numbers to messages

You can add sequence numbers to messages entering a message flow by using the Sequence node.

Before you begin

Before you start:

Read the concept topic about "Message sequencing" on page 2784.

About this task

The Sequence node allocates a monotonically increasing sequence number for each input message that arrives at the node. As each message arrives at the Sequence node, the sequence number is incremented and stored with the message in the location specified by the `Path to store sequence number` property. The allocation of sequence numbers continues until the sequence ends, as specified by the `End of sequence definition` property.

You can divide input messages into independent sequence groups, based on an identifier defined in the message. Each group has a separate group identifier, and the sequence of messages within each group is managed independently.

The Sequence node allocates a sequence number to each message in the sequence group, and the next sequence number in the group is not allocated until the current message in the group has finished processing (either by being committed or rolled back). This ensures that sequencing is maintained for the group when there are multiple threads in the message flow.

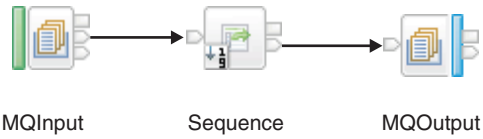
If you need to save the message with the newly assigned sequence number (for example, if you need to save the message to WebSphere MQ for processing by another flow), and if there is no convenient place in the message to save the sequence number, you can add an MQRFH2 header to the message before the Sequence node, and set the sequence number in a field in the `usr` folder.

Multiple sequence groups can be managed independently, in parallel, and sequence group state is preserved when the broker is restarted.

The following steps show how to create a message flow that adds a sequence number to each message in a sequence group.

Procedure

1. Create a message flow containing an MQInput node, a Sequence node, and an MQOutput node.
2. Connect the Out terminal of the MQInput node to the In terminal of the Sequence node.
3. Connect the Out terminal of the Sequence node to the In terminal of the MQOutput node.



4. On the MQInput node, specify the source of input messages for the node by setting the `Queue name` property (on the **Basic** tab) to the name of a WebSphere MQ queue, from which the MQInput node retrieves messages. For example: `SEQ.TASK1.IN1`.
5. Set the following properties of the Sequence node:
   a. On the **Basic** tab, set the following properties:
      - Set the `Path to store sequence number` property to the location in the message where the sequence number is to be set. For example, `$OutputBody/doc/seq`. The sequence number is also set in the local environment, with the *LocalEnvironment.Sequence.Number* variable.
      - Set the `Path to sequence group identifier` property to the location of the sequence group identifier in the message. For example, `$InputBody/doc/grp`. The sequence group identifier is also copied to the local environment, with the *LocalEnvironment.Sequence.Group* variable.
      - Set the `Start of sequence definition` property to `Literal` with the required starting value; for example, `0`.

        Although the starting sequence number must be specified by a literal number, the value can be overridden in the local environment by the *LocalEnvironment.Sequence.StartOfSequenceNumber* variable.

        The start of sequence message is indicated in the local environment by the *LocalEnvironment.Sequence.Start* variable, which takes a Boolean value.
      - Set the `End of sequence definition` property to one of the following values:
        – `Automatic` with the required timeout value; for example, `60`. This value specifies that the sequence group is closed automatically when the message queue in the node has been empty for 60 seconds.

- `Literal` with the required end value; for example, `100`. This value specifies that the sequence group is closed when the message with the sequence number `100` is processed.
- `Predicate` with the required XPATH expression; for example, `$InputBody/doc/endFlag`. This value specifies that the sequence group is closed when the `$InputBody/doc/endFlag` predicate evaluates to True (`$InputBody/doc/endFlag=True`).

The end of sequence message is indicated in the local environment by the *LocalEnvironment.Sequence.End* variable, which takes a Boolean value.

b. On the **Advanced** tab, set the `Persistence mode` property to `Non-persistent`. Select `Persistent` if you want the sequence to be preserved if the queue manager is restarted.

6. On the MQOutput node, set the `Queue name` property (on the **Basic** tab) to the name of a WebSphere MQ queue to which the MQOutput node sends messages. For example: `SEQ.TASK1.OUT1`.
7. Save your message flow.

Related concepts:

"Message sequencing" on page 2784
Use message sequencing to ensure that messages are delivered to the receiving application in a particular order.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Using message sequences" on page 2783
You can maintain or change the sequence of messages in a message flow based on the sequence number and group ID contained in each message.

"Reordering messages in a message flow" on page 2797
When messages entering a message flow contain sequence numbers and a group ID, you can use the Resequence node to re-establish the sequential order of the messages before propagating them through the message flow.

"Maintaining the sequential order of messages" on page 2799
You can maintain the order in which messages enter a message flow by using the Sequence and Resequence nodes.

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"Resequence node" on page 4651
Use the Resequence node to control the sequence in which a group (or groups) of incoming messages are propagated in a message flow.

"Sequence node" on page 4736
Use the Sequence node to add a sequence number to one or more groups of input messages.

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

"MQOutput node" on page 4612
Use the MQOutput node to send messages to clients that connect to the broker using the WebSphere MQ Enterprise Transport and that use the MQI and AMI application programming interfaces.

Reordering messages in a message flow

When messages entering a message flow contain sequence numbers and a group ID, you can use the Resequence node to re-establish the sequential order of the messages before propagating them through the message flow.

Before you begin

Before you start:

Read the concept topic about "Message sequencing" on page 2784.

Ensure that each message contains a monotonically increasing sequence number.

About this task

The sequence number might have been added to the message by the Sequence node or it might be another integer field in the message.

When the Resequence node receives an input message, it propagates the message only if it is the next one in the sequence. If the message is not next in the sequence, the Resequence node stores it until further messages arrive that allow the node to correct the sequence, at which point the node propagates the stored message. If a message fails to arrive, preventing the Resequence node from completing the sequence, the remaining messages are processed according to the way in which the Resequence node has been configured. For more information about this configuration, see "Handling missing messages" on page 2806.

A transaction break occurs at the Resequence node. When a message is delivered to the Resequence node, control is returned to the previous node in the message flow. All messages that are propagated from the Resequence node are propagated in a new transaction, even if the sequence is complete. For more information, see "Resequence node" on page 4651.

The message sequence is preserved when the broker is restarted. If the `Persistent` option is selected on the **Advanced** tab of the Resequence node, the sequence is also preserved when the queue manager is restarted.

The following steps show how to create a message flow that enables you to re-establish the sequential order of the messages in a sequence group:

Procedure

1. Switch to the Broker Application Development perspective.
2. Create a message flow containing an MQInput node, a Resequence node, and an MQOutput node.
3. Connect the Out terminal of the MQInput node to the In terminal of the Resequence node.
4. Connect the Out terminal of the Resequence node to the In terminal of the MQOutput node.

5. On the MQInput node, specify the source of input messages for the node by setting the `Queue name` property (on the **Basic** tab) to the name of a WebSphere MQ queue, from which the MQInput node retrieves messages. For example: `RESEQ.TASK1.IN1`.
6. Set the following properties of the Resequence node:
   a. On the **Basic** tab, set the following properties:
      - Set the `Path to sequence number` property to the location of the sequence number in the message. For example, `$InputBody/doc/seq`. The sequence number is also set in the local environment, with the *LocalEnvironment.Sequence.Number* variable.
      - Set the `Path to sequence group identifier` property to the location of the sequence group identifier in the message. For example, `$InputBody/doc/grp`. The sequence group identifier is also copied to the local environment, with the *LocalEnvironment.Sequence.Group* variable.
      - Set the `Start of sequence definition` property to the first sequence number in the group. For example, select `Literal` with a value of `0`.
      - Set the `End of sequence definition` property to `Automatic` with the required timeout value; for example, `60`. This value specifies that the sequence group is closed automatically when the message queue in the node has been empty for 60 seconds. The end of sequence message is indicated in the local environment by the *LocalEnvironment.Sequence.End* variable, which takes a Boolean value.
      - Set the `Missing message timeout` property to `10`. This value specifies that the Resequence node will wait for a missing message for 10 seconds before propagating subsequent messages in the sequence group to the Expire terminal. When the subsequent message is propagated, the sequence numbers of any missing (timed-out) messages are copied to the local environment, as *LocalEnvironment.Sequence.Missing* variables.
7. On the MQOutput node, set the `Queue name` property (on the **Basic** tab) to the name of a WebSphere MQ queue to which the MQOutput node sends messages. For example: `SEQ.TASK1.OUT1`.
8. Save your message flow.

Related concepts:

"Message sequencing" on page 2784
Use message sequencing to ensure that messages are delivered to the receiving application in a particular order.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Using message sequences" on page 2783
You can maintain or change the sequence of messages in a message flow based on the sequence number and group ID contained in each message.

Chapter 9, “Developing message flow applications,” on page 1019
Develop message flows to process your business messages and data.

Related reference:

“Resequence node” on page 4651
Use the Resequence node to control the sequence in which a group (or groups) of incoming messages are propagated in a message flow.

“Sequence node” on page 4736
Use the Sequence node to add a sequence number to one or more groups of input messages.

“MQInput node” on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

“MQOutput node” on page 4612
Use the MQOutput node to send messages to clients that connect to the broker using the WebSphere MQ Enterprise Transport and that use the MQI and AMI application programming interfaces.

“Compute node” on page 4340
Use the Compute node to construct one or more new output messages.

Maintaining the sequential order of messages

You can maintain the order in which messages enter a message flow by using the Sequence and Resequence nodes.

Before you begin

Before you start:

Read the concept topic about “Message sequencing” on page 2784.

About this task

When messages have been processed in a message flow, they might be propagated in a different order to the order in which they arrived at the input node, so the original message sequence can be altered. You can use the Sequence and Resequence nodes to rearrange the messages into sequential order based on a sequence number in the message, restoring the original sequence in which they entered the input node.

If the messages contain sequence numbers, you can use the Resequence node to re-establish the order in which they arrived at the input node. If the messages do not contain sequence numbers, you can use a Sequence node to apply sequence numbers to the messages, before reordering them into sequential order using the Resequence node.

Procedure

- If the messages contain sequence numbers, rearrange the messages into sequential order by creating a message flow containing a Resequence node. See “Reordering messages in a message flow” on page 2797.
- If the messages do not contain sequence numbers, establish the sequential order of the messages by completing the following steps:
  1. Create a message flow containing a Sequence node, which you can use to apply sequence numbers to the messages. See “Adding sequence numbers to messages” on page 2794.

2. Create a message flow containing a Resequence node, which you can use to re-establish the sequential order of the messages. See “Reordering messages in a message flow” on page 2797.

Related concepts:

“Message sequencing” on page 2784
Use message sequencing to ensure that messages are delivered to the receiving application in a particular order.

“Message flows overview” on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

“Using message sequences” on page 2783
You can maintain or change the sequence of messages in a message flow based on the sequence number and group ID contained in each message.

Chapter 9, “Developing message flow applications,” on page 1019
Develop message flows to process your business messages and data.

“Adding sequence numbers to messages” on page 2794
You can add sequence numbers to messages entering a message flow by using the Sequence node.

“Reordering messages in a message flow” on page 2797
When messages entering a message flow contain sequence numbers and a group ID, you can use the Resequence node to re-establish the sequential order of the messages before propagating them through the message flow.

Related reference:

“Resequence node” on page 4651
Use the Resequence node to control the sequence in which a group (or groups) of incoming messages are propagated in a message flow.

“Sequence node” on page 4736
Use the Sequence node to add a sequence number to one or more groups of input messages.

Message sequencing scenario 1

Change the sequence of messages received from WebSphere MQ, so that they are propagated in the correct order using a sequence number defined in an XML message.

Before you begin

Before you start:

Read the concept topic about “Message sequencing” on page 2784.

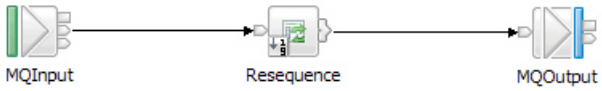
About this task

In this example task, the sequence is defined as starting at sequence number 0 (which is the default) and finishing at sequence number 8. The message flow is configured so that messages can arrive out of order but cannot be propagated out of order, even if one of the sequence numbers never arrives.

The following steps show how to write a message flow that can receive the XML document from WebSphere MQ, reorder the messages based on a sequence number in an XML message (in this example the path `$Root/XMLNSC/Doc/SeqNo` is used) and write it to a WebSphere MQ queue:

Procedure

1. Create a message flow called Resequence_Task1, containing an MQInput node, a Resequence node, and an MQOutput node. For more information about how to do this, see "Creating a message flow" on page 1431.
2. Connect the Out terminal of the MQInput node to the In terminal of the Resequence node.
3. Connect the Out terminal of the Resequence node to the In terminal of the MQOutput node.



4. Set the following properties of the MQInput node:
   a. On the **Basic** tab, set the `Queue name` property to `RESEQUENCE_TASK1_IN1`
   b. On the **Input Message Parsing** tab, set the `Message domain` property to `XMLNSC`.
5. On the Resequence node, set the following properties on the **Basic** tab:
   a. Set the `Path to sequence number` property to `$Root/XMLNSC/Doc/SeqNo`
   b. Set the `End of sequence definition` property to `Literal` with a value of 8.
6. On the MQOutput node, set the `Queue name` property (on the **Basic** tab) to `RESEQUENCE_TASK1_OUT1`.
7. Save the message flow.

Results

Message processing in the message flow:

This section describes the way in which the Resequence node processes the messages that enter the message flow.

1. The following messages arrive on the WebSphere MQ queue RESEQUENCE_TASK1_IN1:

```
<Doc><SeqNo>0</SeqNo></Doc>,<Doc><SeqNo>1</SeqNo></Doc>,<Doc><SeqNo>2</SeqNo></Doc>,
<Doc><SeqNo>3</SeqNo></Doc>,<Doc><SeqNo>4</SeqNo></Doc>,<Doc><SeqNo>5</SeqNo></Doc>,
<Doc><SeqNo>6</SeqNo></Doc>,<Doc><SeqNo>7</SeqNo></Doc>,<Doc><SeqNo>8</SeqNo></Doc>,
```

2. The Resequence node first receives the message with sequence number 0. The Resequence node creates a new sequence group to manage the reordering process; the new sequence group is a default group because no sequence group has been defined in the message. The message (with sequence number 0) is the first one in the sequence, so it is propagated to the Out terminal.
3. The Resequence node then receives the rest of the messages up to and including sequence number 8, and propagates them in the order in which they arrived. Each message is stored before it is propagated, and a different transaction is used for propagating the messages downstream from the Resequence node.
4. When the message containing sequence number 8 is processed, the sequence group is closed. Any new message in the same group that arrives later causes a new group to be created.
5. The next messages arrive on the RESEQUENCE_TASK1_IN1 queue:

```
<Doc><SeqNo>8</SeqNo></Doc>,<Doc><SeqNo>7</SeqNo></Doc>,<Doc><SeqNo>6</SeqNo></Doc>,
<Doc><SeqNo>5</SeqNo></Doc>,<Doc><SeqNo>4</SeqNo></Doc>,<Doc><SeqNo>3</SeqNo></Doc>,
<Doc><SeqNo>2</SeqNo></Doc>,<Doc><SeqNo>1</SeqNo></Doc>,<Doc><SeqNo>0</SeqNo></Doc>,
```

6. The Resequence node first receives the message containing sequence number 8. At this point, the Resequence node creates a new (default) sequence group. The message is the last one in the sequence so it is not propagated but is stored internally.
7. The Resequence node then receives the rest of the messages up to and including sequence number 0. All the messages are stored and none are propagated until the first number in the sequence is received (in this case, sequence number 0).
8. When the message with sequence number 0 is received, it is propagated down the message flow, followed by each of the other messages, in order, until the final message (sequence number 8) is propagated. At this point, the sequence group is closed again.

Related concepts:

"Message sequencing" on page 2784
Use message sequencing to ensure that messages are delivered to the receiving application in a particular order.

"Sequence groups" on page 2786
Use sequence groups to control the way in which messages are grouped together for processing by Sequence and Resequence nodes.

"Starting a message sequence" on page 2787
The start of a message sequence is determined by the `Start of sequence definition` property on the Resequence and Sequence nodes.

"Ending a message sequence" on page 2790
The end of a message sequence is determined by the `End of sequence definition` property on the Resequence and Sequence nodes.

Related tasks:

"Message sequencing scenario 2" on page 2803
Change the sequence of messages received from WebSphere MQ, so that they are propagated in the correct order using a sequence number defined in an XML message.

"Using message sequences" on page 2783
You can maintain or change the sequence of messages in a message flow based on the sequence number and group ID contained in each message.

"Reordering messages in a message flow" on page 2797
When messages entering a message flow contain sequence numbers and a group ID, you can use the Resequence node to re-establish the sequential order of the messages before propagating them through the message flow.

"Maintaining the sequential order of messages" on page 2799
You can maintain the order in which messages enter a message flow by using the Sequence and Resequence nodes.

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"Resequence node" on page 4651
Use the Resequence node to control the sequence in which a group (or groups) of incoming messages are propagated in a message flow.

"Sequence node" on page 4736
Use the Sequence node to add a sequence number to one or more groups of input messages.

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

"MQOutput node" on page 4612
Use the MQOutput node to send messages to clients that connect to the broker using the WebSphere MQ Enterprise Transport and that use the MQI and AMI application programming interfaces.

Message sequencing scenario 2

Change the sequence of messages received from WebSphere MQ, so that they are propagated in the correct order using a sequence number defined in an XML message.

Before you begin

Before you start:

Read the concept topic about "Message sequencing" on page 2784.

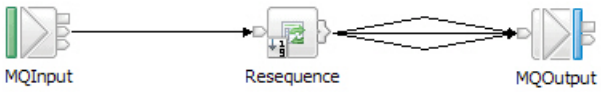
About this task

In this example task, the sequence is defined as starting at sequence number 0 (which is the default) and finishing when it has received no new messages in the group for 60 seconds. The message flow is configured so that messages can arrive out of order but the Resequence node attempts to propagate them in the correct order. If a required message in the sequence fails to arrive for 10 seconds, the Resequence node skips the missing message and propagates the next message in the sequence, even though it is now out of order.

The following steps show how to write a message flow that can receive the XML document from WebSphere MQ, reorder the messages based on a sequence number in an XML message (in this example the path `$Root/XMLNSC/Doc/SeqNo` is used) and write it to a WebSphere MQ queue:

Procedure

1. Create a message flow called Resequence_Task2, containing an MQInput node, a Resequence node, and an MQOutput node. For more information about how to do this, see "Creating a message flow" on page 1431.
2. Connect the Out terminal of the MQInput node to the In terminal of the Resequence node.
3. Connect the Out, Missing, and Expire terminals of the Resequence node to the In terminal of the MQOutput node.



4. Set the following properties of the MQInput node:
   a. On the **Basic** tab, set the `Queue name` property to `RESEQUENCE_TASK2_IN1`
   b. On the **Input Message Parsing** tab, set the `Message domain` property to `XMLNSC`.
5. On the Resequence node, set the following properties on the **Basic** tab:

a. Set the Path to sequence number property to $Root/XMLNSC/Doc/SeqNo
b. Set the Missing message timeout property to 10
c. Set the End of sequence definition property to Automatic with a value of 60.

6. On the MQOutput node, set the Queue name property (on the **Basic** tab) to RESEQUENCE_TASK2_OUT1.
7. Save the message flow.

Results

Message processing in the message flow:

This section describes the way in which the Resequence node processes the messages that enter the message flow:

1. The following messages arrive on the WebSphere MQ queue RESEQUENCE_TASK2_IN1:

```
<Doc><SeqNo>0</SeqNo></Doc>,<Doc><SeqNo>1</SeqNo></Doc>,<Doc><SeqNo>2</SeqNo></Doc>,
<Doc><SeqNo>3</SeqNo></Doc>,<Doc><SeqNo>4</SeqNo></Doc>,<Doc><SeqNo>5</SeqNo></Doc>,
<Doc><SeqNo>6</SeqNo></Doc>,<Doc><SeqNo>7</SeqNo></Doc>,<Doc><SeqNo>8</SeqNo></Doc>,
```

2. The Resequence node first receives the message with sequence number 0. The Resequence node creates a new sequence group to manage the reordering process; the new sequence group is a default group because no sequence group has been defined in the message. The message (with sequence number 0) is the first one in the sequence, so it is propagated to the Out terminal.
3. The Resequence node then receives the rest of the messages up to and including sequence number 8, and propagates them in the order in which they arrived. Each message is stored before it is propagated, and a different transaction is used for propagating the messages downstream from the Resequence node.
4. Sixty seconds elapse and the sequence group is closed.
5. The next messages arrive on the WebSphere MQ queue RESEQUENCE_TASK2_IN1:

```
<Doc><SeqNo>0</SeqNo></Doc>,<Doc><SeqNo>3</SeqNo></Doc>,<Doc><SeqNo>2</SeqNo></Doc>,
<Doc><SeqNo>4</SeqNo></Doc>,<Doc><SeqNo>6</SeqNo></Doc>,<Doc><SeqNo>7</SeqNo></Doc>,
```

6. The Resequence node first receives the message containing sequence number 0. At this point, the Resequence node creates a new (default) sequence group and propagates the message with sequence number 0 to the Out terminal.
7. The next message contains sequence number 3, which is out of sequence. The Missing message timeout property is set on the Resequence node with a value of 10 seconds, and the timer starts.
8. Sequence number 2 arrives, which is still not the next required message but it is lower than the lowest message currently stored, so the missing message timer is reset.
9. The Resequence node then receives the rest of the messages, all of which have a higher sequence number than 2, so they are stored but not propagated. The missing message timer is not reset this time because each number is later in the sequence than number 2.
10. Sequence number 1 fails to arrive during the specified period, and after 10 seconds the timeout period expires.
11. The Resequence node propagates all the messages that are stored, starting with the lowest sequence number (in this case, 2) followed by all other

messages up to the next missing number in the sequence (in this case, messages with sequence numbers 3 and 4). Messages 2, 3, and 4 are propagated to the Expire terminal.

12. The Resequence node is moved to an unordered state for this sequence group, and will not propagate any messages in this group to the Out terminal. The Resequence node remains in the unordered state for this sequence group until the group expires and is closed.
13. When another 10 seconds has passed, the missing message timer expires again and messages 6 and 7 are propagated to the Expire terminal. At this point there are no more missing messages so the missing message timer is not restarted.
14. The group expiry timer is started and the group is closed after 60 seconds have passed. The group expiry timer is never started when a missing message timer is running.

Related concepts:

"Message sequencing" on page 2784
Use message sequencing to ensure that messages are delivered to the receiving application in a particular order.

"Sequence groups" on page 2786
Use sequence groups to control the way in which messages are grouped together for processing by Sequence and Resequence nodes.

"Starting a message sequence" on page 2787
The start of a message sequence is determined by the `Start of sequence definition` property on the Resequence and Sequence nodes.

"Ending a message sequence" on page 2790
The end of a message sequence is determined by the `End of sequence definition` property on the Resequence and Sequence nodes.

Related tasks:

"Message sequencing scenario 1" on page 2800
Change the sequence of messages received from WebSphere MQ, so that they are propagated in the correct order using a sequence number defined in an XML message.

"Using message sequences" on page 2783
You can maintain or change the sequence of messages in a message flow based on the sequence number and group ID contained in each message.

"Reordering messages in a message flow" on page 2797
When messages entering a message flow contain sequence numbers and a group ID, you can use the Resequence node to re-establish the sequential order of the messages before propagating them through the message flow.

"Maintaining the sequential order of messages" on page 2799
You can maintain the order in which messages enter a message flow by using the Sequence and Resequence nodes.

Related reference:

"Resequence node" on page 4651
Use the Resequence node to control the sequence in which a group (or groups) of incoming messages are propagated in a message flow.

"Sequence node" on page 4736
Use the Sequence node to add a sequence number to one or more groups of input messages.

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI

application programming interfaces.

"MQOutput node" on page 4612
Use the MQOutput node to send messages to clients that connect to the broker using the WebSphere MQ Enterprise Transport and that use the MQI and AMI application programming interfaces.

Handling missing messages

You can configure the Resequence node to control how missing messages in a sequence are processed.

About this task

You can configure the Resequence node to time out if a message in the sequence fails to arrive in a specified period of time, and you can specify how subsequent messages are processed if a message is missing.

You can use the `Missing message timeout` property of the Resequence node to specify how long (in seconds) the node waits for the next message in the sequence before it moves on to the next message. Messages that arrive within the specified time limit are propagated in sequential order to the Out terminal. When the specified time limit has been exceeded, the messages are propagated in sequential order to the Expire terminal. Subsequent messages in the sequence group are also routed to the Expire terminal. If the missing message eventually arrives, it is propagated to the Missing terminal.

You can configure the Resequence node for any of the following scenarios:

- **The message sequence must always be maintained**

  If a message goes missing, you can route all subsequent messages to a holding queue after a specified timeout period. To configure the Resequence node in this way, wire the Out terminal to the main-line flow and wire the Expire and Missing terminals to separate branches for re-queueing.
- **Missing messages are allowed in the message sequence but all other messages must remain in sequence**

  If a message goes missing, you can pass over it and continue processing the rest of the sequence. If the missing message eventually arrives, you can either discard it or process it separately from the main-line processing. To configure the Resequence node in this way, wire the Out and Expire terminals to the main-line flow and leave the Missing terminal unwired (to discard the message) or wire it to a separate branch of the message flow.
- **Occasional out-of-sequence messages are allowed**

  You want the message flow to process the messages in sequential order, but you are prepared to tolerate occasional messages out of order. To configure the Resequence node in this way, wire the Out, Expire, and Missing terminals to the main-line message flow.

Related concepts:

"Message sequencing" on page 2784
Use message sequencing to ensure that messages are delivered to the receiving application in a particular order.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Using message sequences" on page 2783
You can maintain or change the sequence of messages in a message flow based on the sequence number and group ID contained in each message.

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Adding sequence numbers to messages" on page 2794
You can add sequence numbers to messages entering a message flow by using the Sequence node.

"Reordering messages in a message flow" on page 2797
When messages entering a message flow contain sequence numbers and a group ID, you can use the Resequence node to re-establish the sequential order of the messages before propagating them through the message flow.

"Configuring the storage of events for Resequence nodes" on page 758
You can use a Resequence configurable service to control the storage of events for Resequence nodes.

Related reference:

"Resequence node" on page 4651
Use the Resequence node to control the sequence in which a group (or groups) of incoming messages are propagated in a message flow.

"Sequence node" on page 4736
Use the Sequence node to add a sequence number to one or more groups of input messages.

Configuring the storage of events for Resequence nodes

You can use a Resequence configurable service to control the storage of events for Resequence nodes.

About this task

By default, the storage queues used by all Resequence nodes are:

- SYSTEM.BROKER.EDA.EVENTS
- SYSTEM.BROKER.EDA.COLLECTIONS

These queues are also used by the Collector node.

However, you can control the queues that are used by different Resequence nodes by creating alternative queues that contain a *QueuePrefix* variable, and by using a Resequence configurable service to specify the names of those queues for storing events.

Follow these steps to specify the queues that are used to store event states, and to set the timeout and the start and end of the sequence:

Procedure

1. Create the storage queues to be used by the Resequence node. The following queues are required:
   - SYSTEM.BROKER.EDA.*QueuePrefix*.EVENTS
   - SYSTEM.BROKER.EDA.*QueuePrefix*.COLLECTIONS

   The *QueuePrefix* variable can contain any characters that are valid in a WebSphere MQ queue name, but must be no longer than eight characters and must not begin or end with a period (.). For example, `SET1` and `SET.1` are valid queue prefixes, but `.SET1` and `SET1.` are invalid.

If you do not create the storage queues, WebSphere Message Broker creates the set of queues when the node is deployed; these queues are based on the default queues. If the queues cannot be created, the message flow is not deployed.

2. Use the **mqsicreateconfigurableservice** command to create a Resequence configurable service. You can create a configurable service to be used with either a specific sequence or with all sequences in an execution group.
   a. If you are creating a configurable service to be used with a specific sequence, ensure that the name of the configurable service is the same as the name that you specify in the `Configurable service` property on the Resequence node. If you are creating a configurable service to be used with all sequences in the execution group, ensure that the configurable service has the same name as the execution group.
   b. Set the **Queue prefix** property to the required value.
   c. Optional: Set the **Missing message timeout**, **Start of sequence**, and **End of sequence** properties.

   For example, create a Resequence configurable service called `myResequenceService`, which uses queues prefixed with SYSTEM.BROKER.EDA.SET1, with a missing message timeout of 60 seconds, and which waits five seconds before determining the start and end numbers in a sequence:

   ```
   mqsicreateconfigurableservice MYBROKER -c Resequence -o myResequenceService
   -n queuePrefix,missingMessageTimeoutSeconds,startSequenceSeconds,endSequenceSeconds -v SET1,60,5,
   ```

   You can use the **mqsideleteconfigurableservice** command to delete the Resequence configurable service. However, the storage queues are not deleted automatically when the configurable service is deleted, so you must delete them separately.For more information, see "Configurable services properties" on page 3766
3. In the Resequence node:
   a. If the configurable service is to be used for a specific sequence, specify the name of the configurable service on the **Advanced** tab; for example, `myResequenceService`. If you do not set the `Configurable service` property, and if a configurable service exists with the same name as the execution group, that configurable service is used instead.
   b. Optional: Use the **mqsichangeproperties** and **mqsireportproperties** commands to change or view the properties of the configurable service. Alternatively, you can use the WebSphere Message Broker Explorer to view or modify a configurable service. For more information about working with configurable services, see "Using the WebSphere Message Broker Explorer to work with configurable services" on page 644.

What to do next

The properties for the configurable service are not used by the broker until you restart or redeploy the message flow, or restart the broker.

Related concepts:

"Using the WebSphere Message Broker Explorer to work with configurable services" on page 644
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view, add, modify and delete configurable services.

Related tasks:

"Viewing configurable services" on page 647
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view the properties of configurable services defined on your broker.

"Modifying an IBM defined configurable service" on page 648
You can create a new configurable service for an external service on which the broker relies, by modifying predefined configurable services provided by IBM. Use the WebSphere Message Broker Explorer to view and modify existing configurable services.

"Modifying a configurable service" on page 649
Use the WebSphere Message Broker Explorer to view and modify existing configurable services.

"Deleting a configurable service" on page 652
Use the WebSphere Message Broker Explorer to delete custom configurable services.

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"Resequence node" on page 4651
Use the Resequence node to control the sequence in which a group (or groups) of incoming messages are propagated in a message flow.

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

Configuring timeout flows

Use the TimeoutControl and TimeoutNotification nodes in message flows to process timeout requests or to generate timeout notifications at specified intervals.

The following topics show how these nodes can be used in a message flow:

- "Sending timeout request messages" on page 2810
- "Sending a message after a timed interval" on page 2813
- "Sending a message multiple times after a specified start time" on page 2815
- "Automatically generating messages to drive a flow" on page 2817
- "Configuring the storage of events for timeout nodes" on page 760

Related concepts:

"Considering performance for timeout flows" on page 2822
When you design timeout flows, the decisions that you make can affect the performance of your brokers and applications.

Related reference:

"TimeoutControl node" on page 4932
Use the TimeoutControl node to process an input message that contains a timeout

request.

"TimeoutNotification node" on page 4936
Use the TimeoutNotification node to manage timeout-dependent message flows.

Sending timeout request messages

To set a controlled timeout, send a message with a set of elements with well known names to a TimeoutControl node. These elements control the properties of the timeout to be created or deleted.

Elements and format

The following example shows the elements and format of a timeout request message, showing the well known names and permissible values.

```
<TimeoutRequest>
  <Action>SET | CANCEL</Action>
  <Identifier>String (any alphanumeric string)</Identifier>
  <StartDate>String (TODAY | yyyy-mm-dd)</StartDate>
  <StartTime>String (NOW | hh:mm:ss)</StartTime>
  <Interval>Integer (seconds)</Interval>
  <Count>Integer (greater than 0 or -1)</Count>
  <IgnoreMissed>TRUE | FALSE</IgnoreMissed>
  <AllowOverwrite>TRUE | FALSE</AllowOverwrite>
</TimeoutRequest>
```

Message fields

The following table describes the fields in the message. The column headed M indicates whether the property is mandatory, and the column headed C indicates whether the property is configurable.

Property	M	C	Default	Description
Action	Yes	No	None	Set this element to either SET or CANCEL. An error is generated if you omit this element or set it to a different value. If you set it to CANCEL, the only other element that is required is the Identifier, which must match the Identifier of the TimeoutRequest that is to be canceled.
Identifier	Yes	No	None	Enter an alphanumeric string. An error is generated if you omit this element.
StartDate	No	No	TODAY	Set this element to TODAY or to a date specified in yyyy-mm-dd format. The default value is TODAY.
StartTime	No	No	NOW	Set this element to NOW or to a time in the future specified in hh:mm:ss format. The default value is NOW. StartTime is assumed to be the broker's local time. The start time can be calculated by adding an interval to the current time. If a delay occurs between the node that calculates the start time and the TimeoutControl node, the start time in the message will have passed by the time it reaches the TimeoutControl node. If the start time is more than approximately five minutes in the past, a warning is issued and the TimeoutControl node rejects the timeout request. If the start time is less than five minutes in the past, the node processes the request as if it were immediate. Therefore, ensure that the start time in the timeout request message is now or a time in the future.
Interval	No	Yes	0	Set this element to an integer that specifies the number of seconds between propagations of the message. The default value is 0.
Count	No	Yes	1	Set this element to an integer value that is either greater than 0 or is -1 (which specifies a timeout request that never expires). The default value is 1.

Property	M	C	Default	Description
IgnoreMissed	Yes	No	TRUE	Set this element to TRUE or FALSE to control whether timeouts that occur while either the broker or the timeout notification flow is stopped, are processed the next time that the broker or timeout notification flow is started. The default value is TRUE, which means that missed timeouts are ignored by the TimeoutNotification node when the broker or message flow is started. If this value is set to FALSE, the missed timeouts are all processed immediately by the TimeoutNotification node when the flow is started. You must set the `Request Persistence` property of the TimeoutControl node to Yes or Automatic (with the originating request message being persistent) for the stored timeouts to persist beyond the restart of the broker or the timeout notification flow.
AllowOverwrite	N	N	TRUE	Set this element to TRUE or FALSE, to specify whether subsequent timeout requests with a matching `Identifier` can overwrite this timeout request. The default value is TRUE.

How the TimeoutControl node uses these values

Set the `Request Location` on the TimeoutControl node to `InputRoot.XML.TimeoutRequest` to read these properties. If you want to obtain properties from a different part of your message, specify the appropriate correlation name for the parent element for the properties. The correlation name for the parent element can be in the local environment.

For details of how the TimeoutControl uses these values, see "TimeoutControl node" on page 4932

Related concepts:

"Sending timeout request messages" on page 2810
To set a controlled timeout, send a message with a set of elements with well known names to a TimeoutControl node. These elements control the properties of the timeout to be created or deleted.

Related tasks:

"Working with the predefined schema definition of an XML timeout request message" on page 2812
A predefined schema definition of an XML timeout request message is provided in the WebSphere Message Broker Toolkit.

"Configuring the storage of events for timeout nodes" on page 760
You can use a Timer configurable service to control the storage of events for TimeoutNotification and TimeoutControl nodes.

Related reference:

"Sending a message after a timed interval" on page 2813
Use TimeoutControl and TimeoutNotification nodes to send a message into a message flow after a timed interval.

"Sending a message multiple times after a specified start time" on page 2815
Use TimeoutControl and TimeoutNotification nodes to send a message into a message flow multiple times after a specified start time.

"Automatically generating messages to drive a flow" on page 2817
Using the TimeoutNotification node to automatically send a message into a message flow.

"Example XML timeout request message" on page 2812
The format used here is XML, but you can use any format that is supported by an

installed parser.

"TimeoutControl node" on page 4932
Use the TimeoutControl node to process an input message that contains a timeout request.

"TimeoutNotification node" on page 4936
Use the TimeoutNotification node to manage timeout-dependent message flows.

Working with the predefined schema definition of an XML timeout request message:

A predefined schema definition of an XML timeout request message is provided in the WebSphere Message Broker Toolkit.

About this task

Take the following steps to review the definition or to define it in a message set.

1. Create or select a message set project that contains the message set.
2. Create a new message definition file (**File** > **New** > **Message Definition File From...**).
3. Select **IBM supplied message** and click **Next**.
4. Expand the tree for **Message Brokers IBM supplied Message Definitions**.
5. Select **Message Broker Timeout Request** and click **Finish**.

Related concepts:

"Sending timeout request messages" on page 2810
To set a controlled timeout, send a message with a set of elements with well known names to a TimeoutControl node. These elements control the properties of the timeout to be created or deleted.

Related reference:

"Example XML timeout request message"
The format used here is XML, but you can use any format that is supported by an installed parser.

Example XML timeout request message:

The format used here is XML, but you can use any format that is supported by an installed parser.

```
<TimeoutRequest>
  <Action>SET</Action>
  <Identifier>TenTimes</Identifier>
  <StartDate>TODAY</StartDate>
  <StartTime>NOW</StartTime>
  <Interval>10</Interval>
  <Count>10</Count>
  <IgnoreMissed>TRUE</IgnoreMissed>
  <AllowOverwrite>TRUE</AllowOverwrite>
</TimeoutRequest>
```

For another example of a timeout request message, and for details of how to use the timeout nodes to add timeouts to message flows, see the timeout processing sample:

- Timeout Processing

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online

information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Related concepts:

"Sending timeout request messages" on page 2810
To set a controlled timeout, send a message with a set of elements with well known names to a TimeoutControl node. These elements control the properties of the timeout to be created or deleted.

Related tasks:

"Working with the predefined schema definition of an XML timeout request message" on page 2812
A predefined schema definition of an XML timeout request message is provided in the WebSphere Message Broker Toolkit.

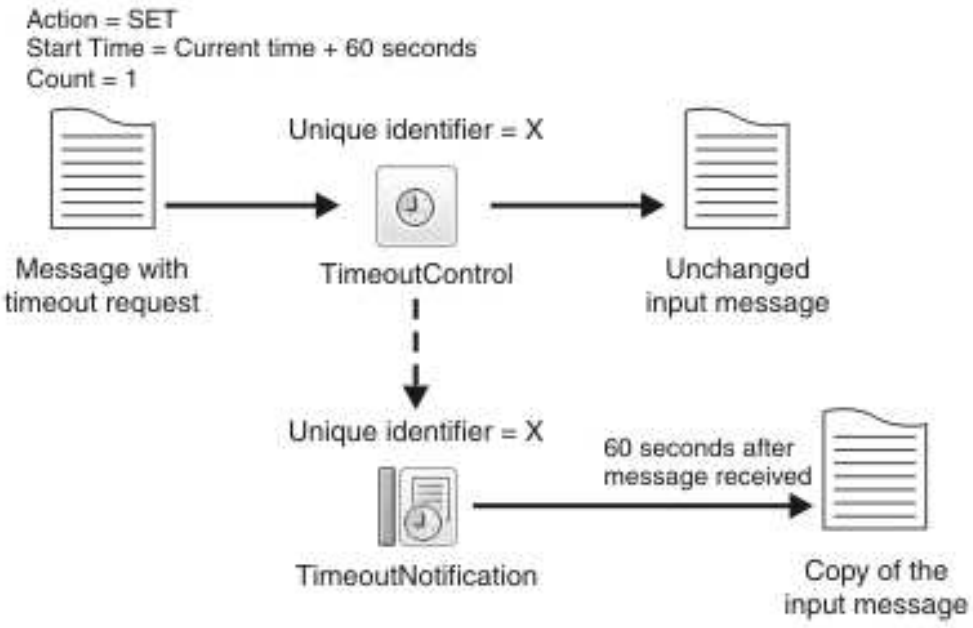
Sending a message after a timed interval

Use TimeoutControl and TimeoutNotification nodes to send a message into a message flow after a timed interval.

Aim

Use TimeoutControl and TimeoutNotification nodes to send a message into a message flow 60 seconds after the message is received.

Description of the flow



The diagram shows the path of a message that contains a timeout request through a TimeoutControl node. A TimeoutNotification node with an identifier matching the TimeoutControl node then processes the timeout request. The diagram also shows the message that the TimeoutNotification node produces after processing the timeout request.

The message comes into the TimeoutControl node with the following values set in the timeout request section of the message:

Action set to *SET*
Start Time set to *current time + 60*
Count set to *1*

The TimeoutControl node validates the timeout request; default values are assumed for properties that are not explicitly defined. The original message is then sent on to the next node in the message flow. If the request is valid, the TimeoutNotification node with the same `Unique identifier` as the TimeoutControl node propagates a copy of the message to the message flow 60 seconds after the message was received.

Timeout request messages are stored for processing on a queue used by the TimeoutNotification node. By default this queue is the SYSTEM.BROKER.TIMEOUT.QUEUE. However, you can use a Timer configurable service to specify an alternative timeout queue, which provides greater control over the storage of messages. For information about using an alternative timeout queue, see "Configuring the storage of events for timeout nodes" on page 760.

If a delay occurs between the node that calculates the start time and the TimeoutControl node, the start time in the message will have passed by the time it reaches the TimeoutControl node. If the start time is more than approximately five minutes in the past, a warning is issued and the TimeoutControl node rejects the timeout request. If the start time is less than five minutes in the past, the node processes the request as if it were immediate. Therefore, ensure that the start time in the timeout request message is a time in the future.

Look at the following sample for further details on constructing this type of message flow.

- Timeout Processing

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Related concepts:

"Sending timeout request messages" on page 2810
To set a controlled timeout, send a message with a set of elements with well known names to a TimeoutControl node. These elements control the properties of the timeout to be created or deleted.

"Handling timeout notification errors" on page 2833
The TimeoutNotification node takes various actions when it handles errors with transactional messages, depending on whether the Failure and Catch terminals are connected.

"Considering performance for timeout flows" on page 2822
When you design timeout flows, the decisions that you make can affect the performance of your brokers and applications.

Related tasks:

"Configuring the storage of events for timeout nodes" on page 760
You can use a Timer configurable service to control the storage of events for TimeoutNotification and TimeoutControl nodes.

Related reference:

"Sending a message multiple times after a specified start time" on page 2815
Use TimeoutControl and TimeoutNotification nodes to send a message into a message flow multiple times after a specified start time.

"Automatically generating messages to drive a flow" on page 2817
Using the TimeoutNotification node to automatically send a message into a message flow.

"TimeoutControl node" on page 4932
Use the TimeoutControl node to process an input message that contains a timeout request.

"TimeoutNotification node" on page 4936
Use the TimeoutNotification node to manage timeout-dependent message flows.

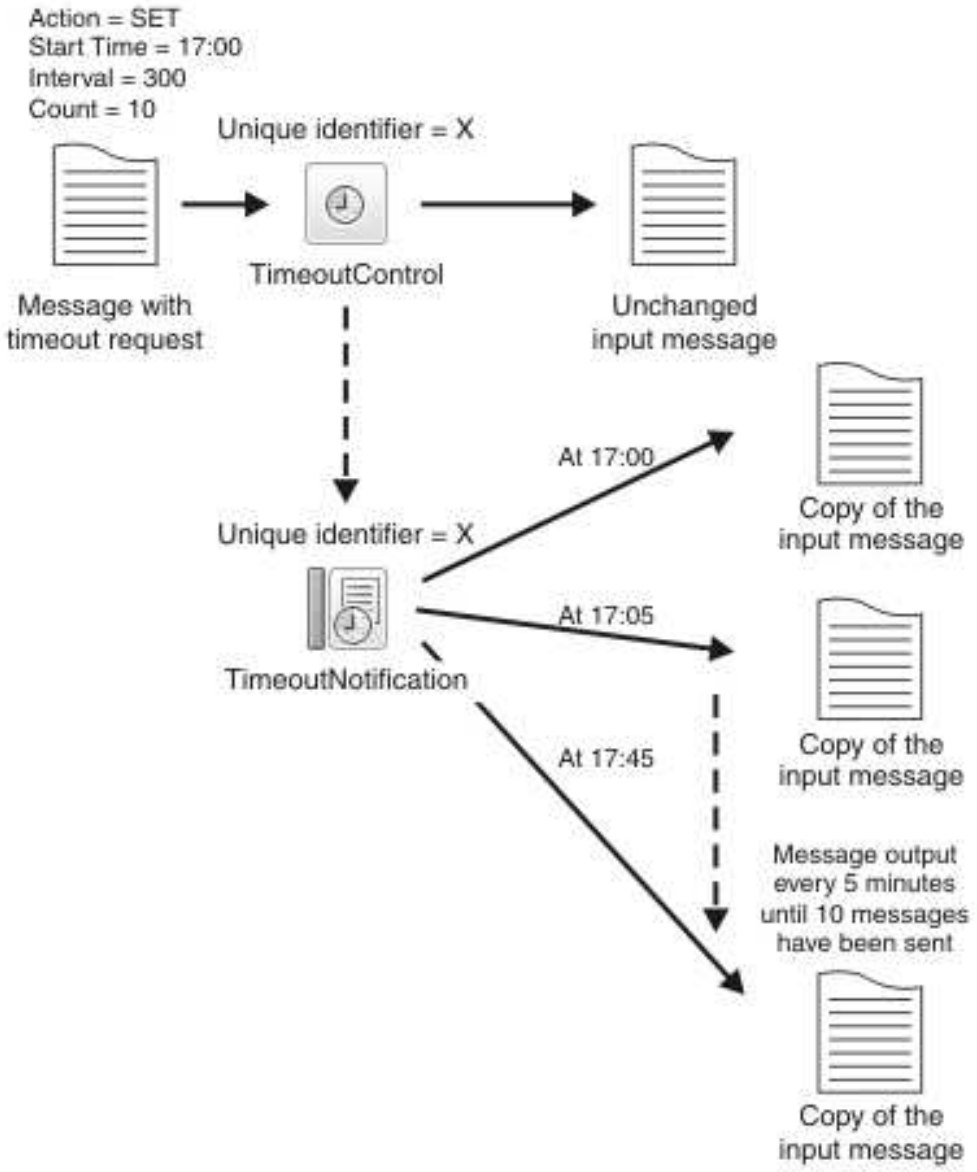
Sending a message multiple times after a specified start time

Use TimeoutControl and TimeoutNotification nodes to send a message into a message flow multiple times after a specified start time.

Aim

Use TimeoutControl and TimeoutNotification nodes to send a message into a message flow at 17:00 hours, then send the message again every 5 minutes until the message has been sent 10 times.

Description of the flow



The diagram shows the path of a message that contains a timeout request through a TimeoutControl node. A TimeoutNotification node with an identifier matching the TimeoutControl node then processes the timeout request. The diagram also shows the message that he TimeoutNotification node produces after processing the timeout request.

The message comes into the TimeoutControl node with the following values set in the timeout request section of the message:

Action set to *SET*
Start Time set to *17:00*
Interval set to *300*
Count set to *10*

The TimeoutControl node validates the timeout request; default values are assumed for properties that are not explicitly defined. The original message is then

sent on to the next node in the message flow. If the request is valid, the TimeoutNotification node with the same `Unique identifier` as the TimeoutControl node propagates a copy of the message to the message flow at 17:00. The message is sent again after an interval of 300 seconds, at 17:05. and every 300 seconds until the message has been sent 10 times, as the *Count* value in the timeout request specifies.

Look at the following sample for further details on constructing this type of message flow.

- Timeout Processing

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Related concepts:

"Sending timeout request messages" on page 2810
To set a controlled timeout, send a message with a set of elements with well known names to a TimeoutControl node. These elements control the properties of the timeout to be created or deleted.

"Handling timeout notification errors" on page 2833
The TimeoutNotification node takes various actions when it handles errors with transactional messages, depending on whether the Failure and Catch terminals are connected.

"Considering performance for timeout flows" on page 2822
When you design timeout flows, the decisions that you make can affect the performance of your brokers and applications.

Related tasks:

"Configuring the storage of events for timeout nodes" on page 760
You can use a Timer configurable service to control the storage of events for TimeoutNotification and TimeoutControl nodes.

Related reference:

"Sending a message after a timed interval" on page 2813
Use TimeoutControl and TimeoutNotification nodes to send a message into a message flow after a timed interval.

"Automatically generating messages to drive a flow"
Using the TimeoutNotification node to automatically send a message into a message flow.

"TimeoutControl node" on page 4932
Use the TimeoutControl node to process an input message that contains a timeout request.

"TimeoutNotification node" on page 4936
Use the TimeoutNotification node to manage timeout-dependent message flows.

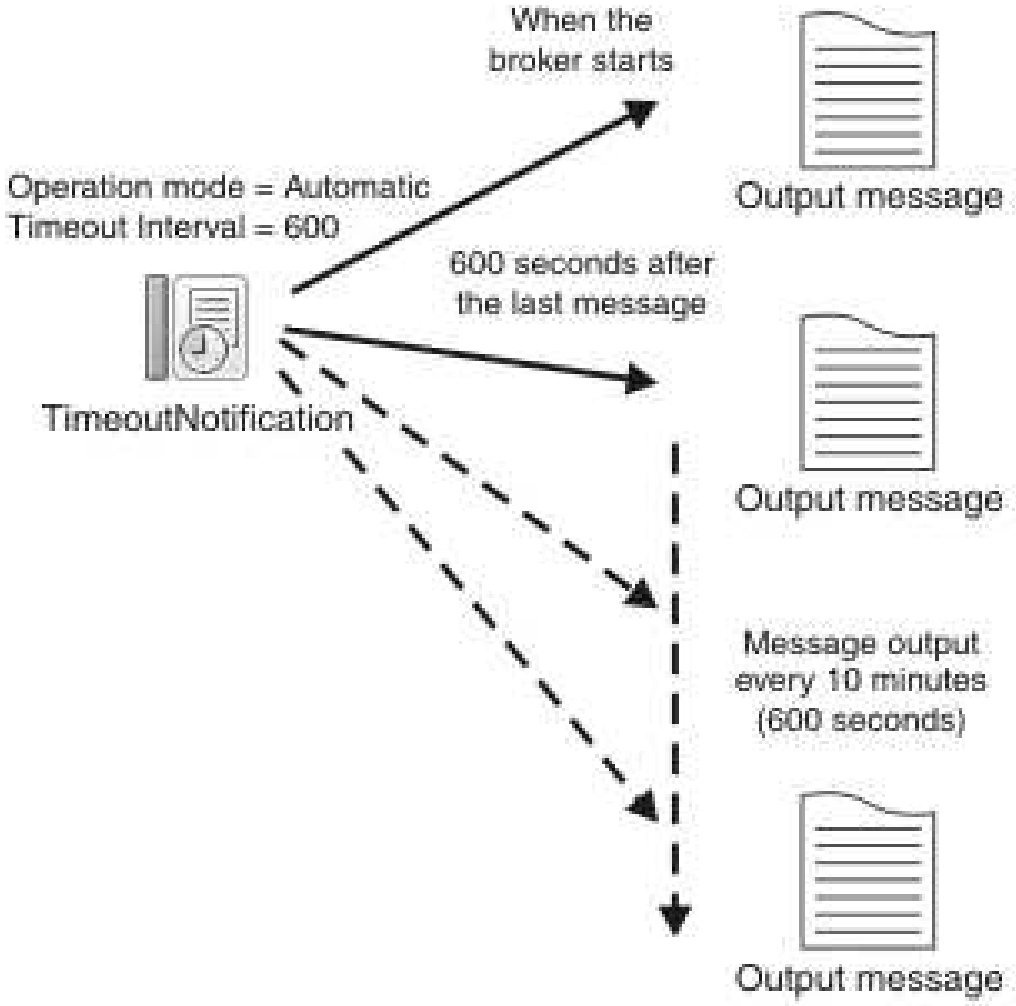
Automatically generating messages to drive a flow

Using the TimeoutNotification node to automatically send a message into a message flow.

Aim

Use the TimeoutNotification node to automatically send a message into a message flow every 10 minutes.

Description of the flow



The diagram shows a TimeoutNotification node automatically generating messages and propagating them every 10 minutes. To get the TimeoutNotification node to automatically generate messages, set the `Operation Mode` property of the node to automatic and specify a value for the `Timeout Interval` property. In this example the TimeoutNotification node has the following properties:

`Operation Mode` set to automatic
`Timeout Interval` set to 600

You can also use a Timer configurable service to control the timeout interval. A value set by the **`Timeout interval`** property of the Timer configurable service overrides the value specified on the TimeoutNotification node. For more information about the Timer configurable service, see "Configurable services properties" on page 3766.

When the broker has started, the TimeoutNotification node sends a message every 10 minutes (600 seconds). This message contains only a properties folder and a LocalEnvironment folder. A Compute node can then process this message to create a more meaningful message.

In the above example, the relevant property is IgnoreMissed, and for an automatic timeout this is implicitly set to TRUE. This means that if one of the period events is missed the event will not be resent, but instead the broker will trigger the event on the next scheduled timeout. If you want to be notified when events are missed, set a controlled timeout instead. For details of the properties you can set for a controlled timeout, see "Sending timeout request messages" on page 2810.

Look at the following sample for further details on constructing this type of message flow.

- Timeout Processing

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Related concepts:

"Sending timeout request messages" on page 2810
To set a controlled timeout, send a message with a set of elements with well known names to a TimeoutControl node. These elements control the properties of the timeout to be created or deleted.

"Handling timeout notification errors" on page 2833
The TimeoutNotification node takes various actions when it handles errors with transactional messages, depending on whether the Failure and Catch terminals are connected.

"Considering performance for timeout flows" on page 2822
When you design timeout flows, the decisions that you make can affect the performance of your brokers and applications.

Related tasks:

"Configuring the storage of events for timeout nodes" on page 760
You can use a Timer configurable service to control the storage of events for TimeoutNotification and TimeoutControl nodes.

Related reference:

"Sending a message after a timed interval" on page 2813
Use TimeoutControl and TimeoutNotification nodes to send a message into a message flow after a timed interval.

"Sending a message multiple times after a specified start time" on page 2815
Use TimeoutControl and TimeoutNotification nodes to send a message into a message flow multiple times after a specified start time.

"TimeoutControl node" on page 4932
Use the TimeoutControl node to process an input message that contains a timeout request.

"TimeoutNotification node" on page 4936
Use the TimeoutNotification node to manage timeout-dependent message flows.

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

Configuring the storage of events for timeout nodes

You can use a Timer configurable service to control the storage of events for TimeoutNotification and TimeoutControl nodes.

About this task

By default, the storage queue used by all timeout nodes is the SYSTEM.BROKER.TIMEOUT.QUEUE.

However, you can control the queues that are used by different timeout nodes by creating alternative queues that contain a *QueuePrefix* variable, and by using a Timer configurable service to specify the names of those queues for storing events.

Follow these steps to specify the queue that is used to store event states:

Procedure

1. Create the storage queue to be used by the timeout nodes. The following queue is required:
   - SYSTEM.BROKER.TIMEOUT.*QueuePrefix*.QUEUE

   The *QueuePrefix* variable can contain any characters that are valid in a WebSphere MQ queue name, but must be no longer than eight characters and must not begin or end with a period (.). For example, `SET1` and `SET.1` are valid queue prefixes, but `.SET1` and `SET1.` are invalid.

   If you do not create the storage queue, WebSphere Message Broker creates the queue when the node is deployed; this queue is based on the default queue. If the queue cannot be created, the message flow is not deployed.
2. Use the **mqsicreateconfigurableservice** command to create a Timer configurable service. You can create a configurable service to be used with either specific timeout requests or with all timeout requests in an execution group.
   a. If the configurable service is to be used with specific timeout requests, create the configurable service with the same name as the `Unique identifier` property on the TimeoutNotification and TimeoutControl nodes. If the configurable service is to be used with all timeout requests in an execution group, create the configurable service with the same name as the execution group.
   b. Set the **Queue prefix** property to the required value.

   For example, create a Timer configurable service that uses a queue prefixed with SYSTEM.BROKER.TIMEOUT.SET1:

   ```
   mqsicreateconfigurableservice MB7BROKER -c Timer -o myTimer
   -n queuePrefix -v SET1
   ```

   You can use the **mqsideleteconfigurableservice** command to delete the Timer configurable service. However, the storage queue is not deleted automatically when the configurable service is deleted, so you must delete it separately.For more information, see "Configurable services properties" on page 3766.
3. In the TimeoutNotification and TimeoutControl nodes:
   a. Ensure that the name of the Timer configurable service is the same as the name specified in the `Unique Identifier` property on the **Basic** tab; for example, `myTimer`. If there is no Timer configurable service with the same name as the `Unique Identifier`, and if there is a configurable service with the same name as the execution group, that configurable service is used instead.
   b. Optional: Use the **mqsichangeproperties** and **mqsireportproperties** commands to change or view the properties of the configurable service. Alternatively, you can use the WebSphere Message Broker Explorer to view or modify a configurable service. For more information about working with

configurable services, see "Using the WebSphere Message Broker Explorer to work with configurable services" on page 644.

What to do next

The properties for the configurable service are not used by the broker until you restart or redeploy the message flow, or restart the broker.

Related concepts:

"Using the WebSphere Message Broker Explorer to work with configurable services" on page 644
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view, add, modify and delete configurable services.

"Configuring timeout flows" on page 2809
Use the TimeoutControl and TimeoutNotification nodes in message flows to process timeout requests or to generate timeout notifications at specified intervals.

"Sending timeout request messages" on page 2810
To set a controlled timeout, send a message with a set of elements with well known names to a TimeoutControl node. These elements control the properties of the timeout to be created or deleted.

Related tasks:

"Viewing configurable services" on page 647
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view the properties of configurable services defined on your broker.

"Modifying an IBM defined configurable service" on page 648
You can create a new configurable service for an external service on which the broker relies, by modifying predefined configurable services provided by IBM. Use the WebSphere Message Broker Explorer to view and modify existing configurable services.

"Modifying a configurable service" on page 649
Use the WebSphere Message Broker Explorer to view and modify existing configurable services.

"Deleting a configurable service" on page 652
Use the WebSphere Message Broker Explorer to delete custom configurable services.

Related reference:

"TimeoutNotification node" on page 4936
Use the TimeoutNotification node to manage timeout-dependent message flows.

"TimeoutControl node" on page 4932
Use the TimeoutControl node to process an input message that contains a timeout request.

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

Considering performance for timeout flows

When you design timeout flows, the decisions that you make can affect the performance of your brokers and applications.

You can use the timeout nodes TimeoutControl and TimeoutNotification in your message flows to control the way in which your message flows operate:

- Set the Operation Mode property of the TimeoutNotification node to Automatic. This setting causes a flow to be invoked at the interval that you specify in the Timeout Value property. If the downstream processing is intensive, and the flow is still busy when the next timeout occurs, the flow is not started for that timeout instance. The flow is notified to start again only if it is free when a particular timeout occurs.

  The value of the Additional Instances property of the message flow is ignored downstream of a TimeoutNotification node, and you cannot use this property to change the behavior of the flow.
- Use two associated flows to perform user-defined timeout processing. Set a timeout with a TimeoutControl node, and notify the flow using a TimeoutNotification node (which behaves like an input node to start a new message flow thread). If the downstream processing from the TimeoutNotification node is significant, requests that are set up in the TimeoutControl node can build up. You can specify that the timeout messages are generated only when the flow that starts with the TimeoutNotification node becomes free again.

  You cannot increase the Additional Instances property of the message flow if it starts with a TimeoutNotification node, therefore you cannot apply more threads to increase the capacity of the flow.

Although you can use a TimeoutNotification node to cause nodes in a message flow to poll for the next item of work, this approach forces a delay between each transaction, and typically does not provide an efficient solution. If you want to periodically check a resource for the next piece of work, and process it immediately, consider one or more of the following alternative solutions:

- Use a built-in input node.
- Write your own input node by using the user-defined node API (in Java or C).
- Consider purchasing an IBM or vendor-provided adapter which polls the subsystem you want, and triggers the flow.

A message flow that uses these options can process more work overall than it can if you implement a timeout solution, and incurs lower CPU cost, although your initial development costs might be slightly higher.

Related concepts:

"Sending timeout request messages" on page 2810
To set a controlled timeout, send a message with a set of elements with well known names to a TimeoutControl node. These elements control the properties of the timeout to be created or deleted.

Related tasks:

"Optimizing message flow throughput" on page 587
Each message flow that you design must provide a complete set of processing for messages received from a certain source. This design might result in very complex message flows that include large numbers of nodes that can cause a performance overhead, and might create potential bottlenecks. You can increase the number of message flows that process your messages to provide the opportunity for parallel processing and therefore improved throughput.

"Using more than one input node" on page 1473
You can include more than one input node in a single message flow.

"Configuring the storage of events for timeout nodes" on page 760
You can use a Timer configurable service to control the storage of events for TimeoutNotification and TimeoutControl nodes.

Related reference:

"Sending a message after a timed interval" on page 2813
Use TimeoutControl and TimeoutNotification nodes to send a message into a message flow after a timed interval.

"Sending a message multiple times after a specified start time" on page 2815
Use TimeoutControl and TimeoutNotification nodes to send a message into a message flow multiple times after a specified start time.

"Automatically generating messages to drive a flow" on page 2817
Using the TimeoutNotification node to automatically send a message into a message flow.

"TimeoutControl node" on page 4932
Use the TimeoutControl node to process an input message that contains a timeout request.

"TimeoutNotification node" on page 4936
Use the TimeoutNotification node to manage timeout-dependent message flows.

Handling errors in message flows

The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

About this task

For example, you might design a message flow that expects certain errors that you want to process in a particular way. Or perhaps your flow updates a database, and must roll back those updates if other processing does not complete successfully.

The options that you can use to do this are quite complex in some cases. The options that are provided for MQInput and TimeoutNotification nodes are extensive because these nodes deal with persistent messages and transactions. The MQInput node is also affected by configuration options for WebSphere MQ.

Because you can decide to handle different errors in different ways, there are no fixed procedures to describe. This section provides information about the principles of error handling, and the options that are available, and you must decide what combination of choices that you need in each situation based on the details that are provided in this section.

There are two general approaches to handling errors in a message flow:

- Failure checking

Wire the Failure terminal of a node to explicitly check for any errors that occur within that node. If errors occur, an exception list is propagated to the Failure terminal. The inflight message remains the same as it was before the node was invoked.

You can introduce more specialized error checking in nodes that can be customized by using ESQL. For example, you can create exit handlers within these nodes. For more information about using ESQL to create exit handlers, see "DECLARE HANDLER statement" on page 5124.

- Catching exceptions

  If you do not wire a Failure terminal, a failure in the node is converted into an exception which is thrown from the node. Any changes that were made to the inflight message before the exception was thrown are reversed. The exception might cause the current transaction to be rolled back which means that any updates to transactional resources are reversed.

  You can prevent the transaction from being rolled back, and control the extent to which message changes are reversed, by including a TryCatch node in your message flow. If an exception is thrown beyond the Try terminal of the TryCatch node, then an exception list is propagated to the node's Catch terminal. The inflight message reverts to the state it was in before it reached the TryCatch node.

  Other nodes apart from the TryCatch node have a Catch terminal. These nodes are typically at the start of a transaction, where an uncaught exception would cause a rollback. In these nodes, the Catch terminal behaves as though a TryCatch node was wired directly to the Out terminal. Use the Catch terminal to handle any exceptions that are thrown beyond the node in the message flow. Wire the Failure terminal to handle errors within the node itself.

You can choose one or more of these options in your message flows:

- Connect the Failure terminal of the node to a sequence of nodes that processes the node's internal exception (the fail flow).
- Connect the Catch terminal of the node to a sequence of nodes that processes exceptions that are generated beyond it (the catch flow).
- Insert one or more TryCatch nodes at specific points in the message flow to catch and process exceptions that are generated by the flow connected to the Try terminal.
- Include a Throw node, or code an ESQL THROW statement, to generate an exception.
- Ensure that all the messages received by an MQInput node are processed in a transaction, or are not processed in a transaction.
- Ensure that all the messages received by an MQInput node are persistent, or are not persistent.

If you include user-defined nodes in your message flow, you must see the information provided with the node to understand how you might handle errors with these nodes. The descriptions in this section cover only the built-in nodes.

When you design your error handling approach, consider the following factors:

- Most of the built-in nodes have Failure terminals. The exceptions are the AggregateControl, AggregateRequest, Input, Label, Output, Passthrough, Publication, Trace, and TryCatch nodes.

  When an exception is detected in a node, the message and the exception information are propagated to the node's Failure terminal. If the node does not have a Failure terminal, or it is not connected, the broker throws an exception

and returns control to the closest upstream node that can process it. This node might be a TryCatch node, an AggregateReply node, or the input node.

If an MQInput node detects an internal error, its behavior is slightly different; if the Failure terminal is not connected, it attempts to put the message to the input queue's backout requeue queue, or (if that is not defined) to the dead letter queue of the broker's queue manager.

For more information, see "Handling MQInput errors" on page 2829.

- Many built-in nodes have Catch terminals. These nodes are typically at the start of a transaction. For example:
  - Input nodes: FileInput, HTTPInput, JMSInput, MQInput, PeopleSoftInput, SAPInput, SCAInput, SiebelInput, SOAPInput, TCPIPClientInput, TCPIPServerInput, TwineballInput
  - Output and response nodes: SCAAsyncResponse, SOAPAsyncResponse
  - Routing nodes: AggregateReply, Collector, Resequence
  - Construction nodes: TryCatch
  - Timer nodes: TimeoutNotification

  A message is propagated to a Catch terminal only if it has first been propagated beyond the node (for example, to the nodes connected to the Out terminal).
- When a message is propagated to the Failure or Catch terminal, the node creates and populates a new exception list tree with an exception that represents the error that has occurred. The exception list is propagated as part of the message tree. When a message is propagated to the Failure terminal because of a problem that occurred in the Out or Catch flows (for example, repeated parsing errors that caused the backout threshold to be met), the original parsing errors are not in the exception list; the exception list contains the exception that indicates that the backout threshold has been met.
- The MQInput and TimeoutNotification nodes have additional processing for transactional messages (other input nodes do not handle transactional messages).

  For more information, see "Handling MQInput errors" on page 2829 and "Handling timeout notification errors" on page 2833.
- If you include a Trace node that specifies `$Root` or `$Body`, the complete message is parsed. This might generate parser errors that are not otherwise detected.

The general principles of error handling are:

- If you connect the Catch terminal of the input node, you are indicating that the flow handles all the exceptions that are generated anywhere in the out flow. The broker performs no rollback, and takes no action, unless there is an exception on the catch flow. If you want any rollback action after an exception has been raised and caught, you must provide this in the catch flow.
- If you do not connect the Catch terminal of the input node, you can connect the Failure terminal and provide a fail flow to handle exceptions generated by the node. The fail flow is started immediately when an exception occurs in the node.

  The fail flow is also started if an exception is generated beyond the MQInput node (in either out or catch flows), the message is transactional, and the reinstatement of the message on the input queue causes the backout count to reach the backout threshold.

  The HTTPInput node does not propagate the message to the Failure terminal if an exception is generated beyond the node and you have not connected the Catch terminal.

- If a node propagates a message to a catch flow, and another exception occurs that returns control to the same node again, the node handles the message as though the Catch terminal is not connected.
- If you do not connect either the Failure or Catch terminals of the input node, the broker provides default processing (which varies with the type of input node).
- If you need a more comprehensive error and recovery approach, include one or more TryCatch nodes to provide more localized areas of error handling.
- If you have a common procedure for handling particular errors, you might find it appropriate to create a subflow that includes the sequence of nodes required. Include this subflow wherever you need that action to be taken.

For more information, see "Connecting failure terminals" on page 2827, "Managing errors in the input node" on page 2828, and "Catching exceptions in a TryCatch node" on page 2835.

If your message flows include database updates, the way in which you configure the nodes that interact with those databases can also affect the way that errors are handled:

- You can specify whether updates are committed or rolled back. You can set the node property `Transaction` to specify whether database updates are committed or rolled back with the message flow (option Automatic), or are committed or rolled back when the node itself terminates (option Commit). You must ensure that the combination of these property settings and the message flow error processing give the correct result.
- You can specify how database errors are handled. You can set the properties `Treat warnings as errors` and `Throw exception on database error` to change the default behavior of database error handling.

For more information about coordinated database updates, see "Configuring transactionality for message flows" on page 1290.

Message flows for aggregation involve additional factors that are not discussed in this topic. For information about message flows for aggregation, see "Handling exceptions in aggregation flows" on page 2749.

The following sample demonstrates how to use an error handling routine to trap information about errors and to store that information in a database. The error handling routine is a subflow that you can add, unchanged, to your message flows. The sample also demonstrates how to configure message flows to control transactionality; in particular, the use of globally coordinated transactions to ensure overall data integrity.

- Error Handler

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Related concepts:

"Subflows" on page 1030
You can include subflows in your message flows in the same way as you include built-in or user-defined nodes. You can also connect subflows to other nodes in the same way.

Related tasks:

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

"Handling exceptions in aggregation flows" on page 2749
When you use aggregation flows, exceptions might occur.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"WebSphere MQ Enterprise Transport" on page 1542
WebSphere MQ Enterprise Transport is a service that connects applications to messaging middleware.

Connecting failure terminals

When a node that has a failure terminal detects an internal error, it propagates the message to that terminal. If it does not have a failure terminal, or if you have not connected the failure terminal, the broker generates an exception.

About this task

The nodes sometimes generate errors that you can predict, and it is in these cases that you might want to consider connecting the failure terminal to a sequence of nodes that can take sensible actions in response to the expected errors.

Examples of expected errors are:

- Temporary errors when the input node retrieves the message.
- Validation errors detected by an MQInput, Compute, or Mapping node.
- Messages with an internal or format error that cannot be recognized or processed by the input node.
- Acceptable errors when a node accesses a database, and you choose not to configure the node to handle those errors.
- ESQL errors during message flow development (some ESQL errors cannot be detected by the editor, but are recognized only by the broker; these cause an exception if you have not connected the failure terminal. You can remove the fail flow when you have completely tested the runtime ESQL code).

You can also connect the failure terminal if you do not want WebSphere MQ to try a message again or put it to a backout or dead letter queue.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Packaging and deployment overview" on page 3210
Deployment is the process of transferring data to an execution group on a broker so that it can take effect in the broker. Message flows and associated resources are packaged in broker archive (BAR) files for deployment.

"Subflows" on page 1030
You can include subflows in your message flows in the same way as you include built-in or user-defined nodes. You can also connect subflows to other nodes in the same way.

Related tasks:

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"WebSphere MQ Enterprise Transport" on page 1542
WebSphere MQ Enterprise Transport is a service that connects applications to messaging middleware.

Managing errors in the input node

When you design your message flow, consider which terminals on the input node to connect.

About this task

- If the node detects an internal error before the message is propagated to the Out terminal, the node always propagates the message and an exception list to the Failure terminal if the node has a Failure terminal and if you have connected a fail flow. If the Failure terminal is not connected or an exception occurs downstream of the failure terminal, the transaction is rolled back.
- If you connect the Catch terminal (if the node has one), this indicates that you want to handle all exceptions that are generated in the out flow. If you do not connect the Catch terminal, or the node does not have a Catch terminal, or an exception occurs downstream of the Catch terminal the current transaction is rolled back.
- Any internal exceptions that occur in the node after the message has been propagated to the Out terminal cause a rollback of the transaction. This situation is rare but could happen for some input nodes.

Refer to the documentation for each input node to determine the effect of rolling back the transaction and also for the retry capabilities provided by some input nodes.

Related concepts:

"Subflows" on page 1030
You can include subflows in your message flows in the same way as you include built-in or user-defined nodes. You can also connect subflows to other nodes in the same way.

Related tasks:

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can

create one or more message flows and deploy them to one or more brokers.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

"Handling exceptions in aggregation flows" on page 2749
When you use aggregation flows, exceptions might occur.

Related reference:

"WebSphere MQ Enterprise Transport" on page 1542
WebSphere MQ Enterprise Transport is a service that connects applications to messaging middleware.

Handling MQInput errors

The MQInput node takes certain actions when handling errors with persistent and transactional messages. The node attempts retry processing when a transactional message is rolled back to the input queue. Non-transactional messages are not rolled back to the input queue when an exception occurs.

About this task

Errors encountered with non-transactional messages are handled as described in "Managing errors in the input node" on page 2828.

- The MQInput node detects an internal error in the following situations:
  - An exception occurs when the associated message parser is initialized. If the `Parse timing` property is set to `Immediate` or `Complete`, the parser parses the input message after initialization. This parsing can cause a parsing or validation error, which is treated as an internal error.
  - A warning is received on an MQGET call.
  - The backout threshold is reached when the message is rolled back to the input queue.
- If the MQInput node detects an internal error, one of the following actions occur:
  - If you have not connected the Failure terminal, the MQInput node attempts to put the message to the input queue's backout requeue queue, or (if that is not defined) to the dead letter queue of the broker's queue manager. If the put attempt fails, the message is rolled back to the input queue. The MQInput node writes the original error and the MQPUT error to the local error log. The MQInput node now invokes the retry logic, described in "Handling retry processing" on page 2830.
  - If you have connected the Failure terminal, you are responsible for handling the error in the flow connected to the Failure terminal. The broker creates a new exception list to represent the error and this is propagated to the Failure terminal as part of the message tree, but neither the MQInput node nor the broker provide any further failure processing.
- If the MQInput node has successfully propagated the message to the out terminal and an exception is thrown in the out flow, the message is returned to the MQInput node:
  - If you have not connected the Catch terminal, the message is rolled back to the input queue. The MQInput node writes the error to the local error log and invokes the retry logic, described in "Handling retry processing" on page 2830.

  - If you have connected the Catch terminal, you are responsible for handling the error in the flow connected to the Catch terminal. The broker creates a new exception list to represent the error and this list is propagated to the Catch terminal as part of the message tree, but neither the MQInput node nor the broker provide any further failure processing.
- If the MQInput node has already propagated the message to the Catch terminal and an exception is thrown in the flow connected to the Catch terminal, the message is returned to the MQInput node:
  - The MQInput node writes the error to the local error log.
  - The message is rolled back to the input queue.
- If the MQInput node has already propagated the message to the Failure terminal and an exception is thrown in the flow connected to the Failure terminal, the message is returned to the MQInput node and rolled back to the input queue. The MQInput node writes the error to the local error log and invokes the retry logic, described in "Handling retry processing." The message is not propagated to the Catch terminal, even if that is connected.

This action is summarized in the following table.

Error event	Failure terminal connected	Failure terminal not connected	Catch terminal connected	Catch terminal not connected
Node detects internal error	Flow connected to the Failure terminal handles the error	Message put to alternative queue; node retries if the put fails	Not applicable	Not applicable
Node propagates message to out terminal, exception occurs in out flow	Not applicable	Not applicable	Flow connected to the Catch terminal handles the error	Node retries
Node propagates message to Catch terminal, exception occurs in flow connected to the Catch terminal	Error logged, message rolled back	Error logged, message rolled back	Not applicable	Not applicable
Node propagates message to Failure terminal, exception occurs in flow connected to the Failure terminal	Not applicable	Not applicable	Node retries	Node retries

Handling retry processing:
About this task

The node attempts retry processing when a message is rolled back to the input queue. The node checks whether the message has been backed out before, and if it has, whether the backout count has reached (equalled) the backout threshold. The backout count for each message is maintained by WebSphere MQ in the MQMD.

You specify (or allow to default to zero(0)) the backout threshold attribute *BOTHRESH* when you create the queue. If you accept the default value of 0, the node increases this value to one (1). The node also sets the value to 1 if it cannot detect the current value. If the backout threshold is set to 1, the message is processed once but it is not retried through the Out terminal of the MQInput node. To retry the message at least once, set the backout threshold to 2.

1. If the node has propagated a message to the Out terminal many times following repeated failed attempts in the output flow, and the number of retries has reached the backout threshold limit, the node attempts to propagate the message through the Failure terminal if that is connected. If you have not connected the Failure terminal, the node attempts to put the message to another queue. When a message is propagated through the Failure terminal, the exception list does not contain the exceptions that occurred in the flows connected to the Out or Catch terminals; the exception list contains new exceptions that cover the reason that the message has gone through the Failure terminal (for example, the backout threshold was reached).

   If a failure occurs beyond the Failure terminal, further retries are made until the backout count field in the MQMD reaches twice the backout threshold set for the input queue. When this limit is reached, the node attempts to put the message to another queue.
2. If the backout threshold has not been reached, the node gets the message from the queue again. If this fails, this is handled as an internal error (described above). If it succeeds, the node propagates the message to the out flow.
3. If the backout threshold has been reached:
   - If you have connected the Failure terminal, the node propagates the message to that terminal. You must handle the error on the flow connected to the Failure terminal.
   - If you have not connected the Failure terminal, the node attempts to put the message on an available queue, in order of preference:
     a. The message is put on the input queue's backout requeue name (queue attribute *BOQNAME*), if one is defined.
     b. If the backout queue is not defined, or it cannot be identified by the node, the message is put on the dead letter queue (DLQ), if one is defined. (If the broker's queue manager has been defined by the **mqsicreatebroker** command, a DLQ with a default name of SYSTEM.DEAD.LETTER.QUEUE has been defined and is enabled for this queue manager.) The MQDLH **PutApplName** property is set to `WebSphereMQIntegrator` and appended with the broker major version number, for example: `WebSphereMQIntegrator9`
     c. If the message cannot be put on either of these queues because there is an MQPUT error (including queue does not exist), or because they cannot be identified by the node, it cannot be handled safely without risk of loss.

        The message cannot be discarded, therefore the message flow continues to attempt to backout the message. It records the error situation by writing errors to the local error log. A second indication of this error is the continual incrementing of the *BackoutCount* of the message in the input queue.

        If this situation has occurred because neither queue exists, you can define one of the backout queues mentioned above. If the condition preventing the message from being processed has cleared, you can temporarily increase the value of the *BOTHRESH* attribute. This forces the message through normal processing.

4. If twice the backout threshold has been reached or exceeded, the node attempts to put the message on an available queue, in order of preference, as defined in the previous step.

Handling transactions, event monitoring, and backout processing:
About this task

When the MQInput node performs retry processing, it can backout the transaction by putting the message to the backout requeue queue or the dead letter queue. The backout count is checked when the message is received from the input queue. If the backout threshold is exceeded, the message is backed out immediately with no further processing performed on the message. The backout operation occurs in a separate transaction from previous processing failures, and the message is not parsed or validated during the backout transaction. The backout transaction generates its own set of monitoring events. Therefore, information that is obtained through message parsing, such as the *exceptionList*, might not be available.

Handling message group errors:
About this task

WebSphere MQ supports message groups. You can specify that a message belongs to a group and its processing is then completed with reference to the other messages in the group (that is, either all messages are committed or all messages are rolled back). When you send grouped messages to a broker, this condition is upheld if you have configured the message flow correctly, and errors do not occur during group message processing.

To configure the message flow to handle grouped messages correctly, follow the actions described in the "MQInput node" on page 4594. However, correct processing of the message group cannot be guaranteed if an error occurs while one of the messages is being processed.

If you have configured the MQInput node as described, under normal circumstances all messages in the group are processed in a single unit of work which is committed when the last message in the group has been successfully processed. However, if an error occurs before the last message in the group is processed, the unit of work that includes the messages up to and including the message that generates the error is subject to the error handling defined by the rules documented here, which might result in the unit of work being backed out.

However, any of the remaining messages in the group might be successfully read and processed by the message flow, and therefore are handled and committed in a new unit of work. A commit is issued when the last message is encountered and processed. Therefore if an error occurs in a group, but not on the first or last message, it is possible that part of the group is backed out and another part committed.

If your message processing requirements demand that this situation is handled in a particular way, you must provide additional error handling to handle errors in message groups. For example, you could record the failure of the message group in a database, and include a check on the database when you retrieve each message, forcing a rollback if the current group has already encountered an error. This would ensure that the whole group of messages is backed out and not processed unless all are successful.

Related concepts:

"Subflows" on page 1030
You can include subflows in your message flows in the same way as you include built-in or user-defined nodes. You can also connect subflows to other nodes in the same way.

Related tasks:

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Managing errors in the input node" on page 2828
When you design your message flow, consider which terminals on the input node to connect.

"Catching exceptions in a TryCatch node" on page 2835
You can design a message flow to catch exceptions before they are returned to the input node. You can include one or more TryCatch nodes in a flow to provide a single point of failure for a sequence of nodes. With this technique, you can provide specific error processing and recovery.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

"WebSphere MQ Enterprise Transport" on page 1542
WebSphere MQ Enterprise Transport is a service that connects applications to messaging middleware.

Handling timeout notification errors

The TimeoutNotification node takes various actions when it handles errors with transactional messages, depending on whether the Failure and Catch terminals are connected.

Errors that are encountered with non-transactional messages are handled as described in "Managing errors in the input node" on page 2828.

- If the TimeoutNotification node detects an internal error, one of the following actions occur:
  - If you have not connected the Failure terminal:
    1. The TimeoutNotification node writes the error to the local error log.
    2. The TimeoutNotification node repeatedly tries to process the request until the problem has been resolved.
  - If you have connected the Failure terminal, you are responsible for handling the error in the flow connected to the Failure terminal. The broker creates an exception list to represent the error, which is propagated to the Failure

terminal as part of the message tree. The TimeoutNotification node and the broker do not provide further failure processing. The message is written to the Failure terminal as part of the same transaction, and if the failure flow handles the error successfully the transaction is committed.

- If the TimeoutNotification node has successfully propagated the message to the Out terminal and an exception is thrown in the flow connected to the Out terminal, the message is returned to the TimeoutNotification node. The TimeoutNotification node writes the error to the local error log and does one of the following:
  - If you have not connected the Catch terminal, the TimeoutNotification node tries to process the message again until the problem is resolved.
  - If you have connected the Catch terminal, you are responsible for handling the error in the flow connected to the Catch terminal. The broker creates an exception list to represent the error, which is propagated to the Catch terminal as part of the message tree. The TimeoutNotification node and the broker do not provide further failure processing. The message is written to the Catch terminal as part of the same transaction, and if the flow connected to the Catch terminal handles the error successfully the transaction is committed.
- If the TimeoutNotification node has already propagated the message to the Catch terminal and an exception is thrown in the flow connected to the Catch terminal, the message is returned to the TimeoutNotification node. The TimeoutNotification node writes the error to the local error log and tries to process the message again.
- If the TimeoutNotification node has already propagated the message to the Failure terminal and an exception is thrown in the flow connected to the Failure terminal, the message is returned to the TimeoutNotification node. The TimeoutNotification node writes the error to the local error log, and tries to process the message again. The message is not propagated to the Catch terminal, even if that is connected.

This action is summarized in the following table.

Error event	Failure terminal connected	Failure terminal not connected	Catch terminal connected	Catch terminal not connected
The node detects an internal error.	The flow that is connected to the Failure terminal handles the error.	The error is logged and the node retries the operation.	Not applicable	Not applicable
The node propagates a message to the Out terminal and an exception occurs in the output flow.	Not applicable	Not applicable	The flow that is connected to the Catch terminal handles the error.	The error is logged and the node retries the operation.
The node propagates a message to the Catch terminal and an exception occurs in the flow that is connected to the Catch terminal.	The error is logged and the node retries the operation.	The error is logged and the node retries the operation.	Not applicable	Not applicable

Error event	Failure terminal connected	Failure terminal not connected	Catch terminal connected	Catch terminal not connected
The node propagates a message to the Failure terminal and an exception occurs in the flow that is connected to the Failure terminal.	Not applicable	Not applicable	The error is logged and the node retries the operation.	The error is logged and the node retries the operation.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Subflows" on page 1030
You can include subflows in your message flows in the same way as you include built-in or user-defined nodes. You can also connect subflows to other nodes in the same way.

Related tasks:

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Managing errors in the input node" on page 2828
When you design your message flow, consider which terminals on the input node to connect.

"Catching exceptions in a TryCatch node"
You can design a message flow to catch exceptions before they are returned to the input node. You can include one or more TryCatch nodes in a flow to provide a single point of failure for a sequence of nodes. With this technique, you can provide specific error processing and recovery.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"TimeoutNotification node" on page 4936
Use the TimeoutNotification node to manage timeout-dependent message flows.

Catching exceptions in a TryCatch node

You can design a message flow to catch exceptions before they are returned to the input node. You can include one or more TryCatch nodes in a flow to provide a single point of failure for a sequence of nodes. With this technique, you can provide specific error processing and recovery.

About this task

A TryCatch node does not process a message in any way, it represents only a decision point in a message flow. When the TryCatch node receives a message, it propagates it to the Try terminal. The broker passes control to the sequence of nodes that are connected to that terminal (the try flow).

If an exception is thrown in the try flow, the broker returns control to the TryCatch node. The node writes the current contents of the exception list tree to the local error log, then writes the information for the current exception to the exception list tree, overwriting the information that is stored there.

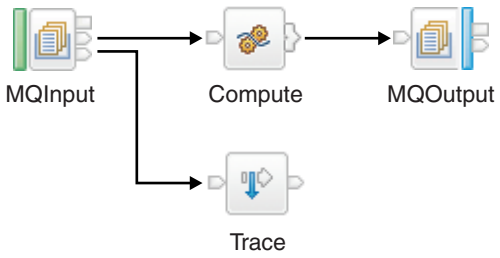
The node propagates the message to the sequence of nodes that are connected to the Catch terminal (the catch flow). The content of the message tree that is propagated is identical to the content that was propagated to the Try terminal, which is the content of the tree when the TryCatch node first received it. The node enhances the message tree with the new exception information that it wrote to the exception list tree. Any modifications or additions that the nodes in the try flow made to the message tree are not present in the message tree that is propagated to the catch flow.

However, if the try flow completes processing that involves updates to external databases, these updates are not lost. The updates persist while the message is processed by the catch flow, and the decision about whether the updates are committed or rolled back is made on the configuration of your message flow and the individual nodes that interact with the databases. If the updates are committed because of the configuration that you set, you must include logic in your catch flow that rolls back the changes that were made.

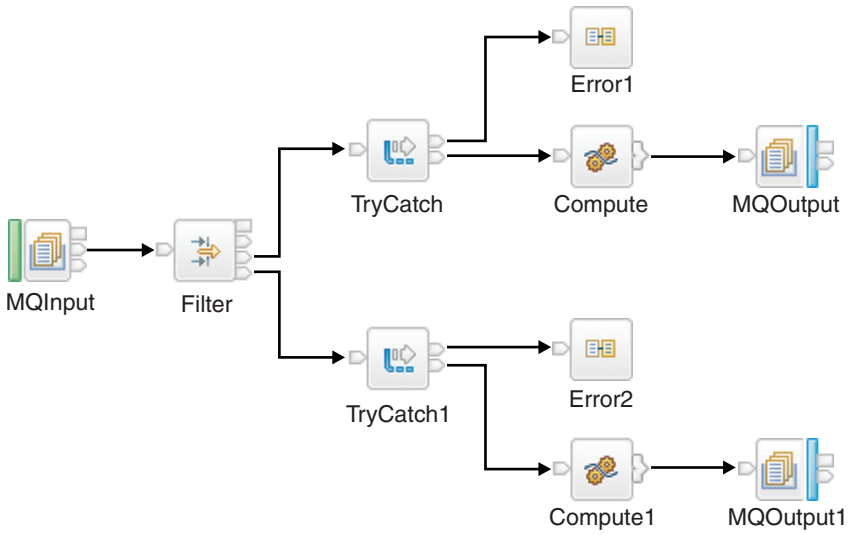
To review the options for configuration, see “Configuring transactionality for message flows” on page 1290.

If an exception is thrown in the catch flow of the TryCatch node (for example, if you include a Throw node, or code an ESQL THROW statement, or if the broker generates the exception), then the broker returns control to the next catch point in the message flow, such as another TryCatch node.

The following example shows how you can configure the flow to catch exceptions in the input node. The MQInput node's Catch terminal is connected to a Trace node to record the error.



In the following example, the message flow has two separate processing flows connected to the Filter node's True and False terminals. Here a TryCatch node is included on each of the two routes that the message can take. The Catch terminal of both TryCatch nodes is connected to a common error processing subflow.



If the input node in your message flow does not have a Catch terminal, and you want to process errors in the flow, you must include a TryCatch node.

Related concepts:

"Subflows" on page 1030
You can include subflows in your message flows in the same way as you include built-in or user-defined nodes. You can also connect subflows to other nodes in the same way.

Related tasks:

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

"Configuring transactionality for message flows" on page 1290
A message flow runs in a single transaction, which is started when data is received by an input node, and can be committed or rolled back when all processing has completed.

"Handling exceptions in aggregation flows" on page 2749
When you use aggregation flows, exceptions might occur.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"WebSphere MQ Enterprise Transport" on page 1542
WebSphere MQ Enterprise Transport is a service that connects applications to messaging middleware.

Constructing message models

This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

About this task

If you are unfamiliar with message models, read the topics that describe the concepts, starting with “Message modeling” on page 1154. These topics explain when you might want to model messages, and describe the message modeling objects that you can use, such as message sets and message definition files.

The WebSphere Message Broker message model is based on XML Schema. For more information about XML Schema, see XML Schema Part 0: Primer.

The following tasks are provided for developing message models:

- “Working with a message set project”
- “Working with a message set” on page 2840
- “Working with a message definition file” on page 2863
- “Working with MRM message model objects” on page 2870
- “Creating a multipart message” on page 2919
- “Linking from one message definition file to another” on page 2921
- “Working with a message category file” on page 2923
- “Working with data structures” on page 2930
- “Generating documentation from message sets and message flows” on page 2962
- “Generating XML Schemas” on page 2963
- “Generating a Broker SCA definition from a message set” on page 2967
- “Generating a WSDL definition from a message set” on page 2968

Tip: The WebSphere Message Broker Toolkit provides a set of toolbar icons that invoke wizards that you can use to create many of the resources that are associated with message models; for example, a new message set. Hold the mouse pointer over a toolbar icon to see its function.

The WebSphere Message Broker Toolkit lets you open resource files with other editors. However, use only the WebSphere Message Broker Toolkit to edit resource files that are associated with message models because this editor correctly validates all changes that you make to these files.

Related concepts:

“Message modeling” on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

“Why model messages?” on page 1158
WebSphere Message Broker supplies a range of parsers to parse and write message formats. Some message formats are self-defining and can be parsed without reference to a model. Most message formats, however, are not self-defining, and the parser must have access to a predefined model that describes the message, if it is to parse it correctly.

Working with a message set project

Creating and deleting a message set.

About this task

Before you begin to develop your message model, you must create a message set. A message set project is automatically created when you create a message set.

This topic area describes the tasks that are involved in working with a message set.

- "Creating a message set" on page 2842
- "Deleting a message set project"

Related concepts:

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

"Message set projects" on page 1161
A *message set project* is a specialized container in which you create and maintain all the resources associated with one message set.

Deleting a message set project

Delete a message set project and, optionally, the contents of the associated project directory.

Before you begin

Before you start:

You must have completed the following task:

- "Creating a message set" on page 2842

Tip: Close all open windows within the WebSphere Message Broker Toolkit that relate to the message set project or associated files that you want to delete. If you do not do this, errors might occur when you try to process objects that no longer exist your workspace.

About this task

This task topic describes how to delete a message set project and, optionally, the contents of the associated project directory.

To delete a message set project:

Procedure

1. Switch to the Broker Application Development perspective.
2. In the Broker Development view, right-click the message set project that you want to delete, then click **Delete** on the pop-up menu. The Confirm Project Delete window opens.
3. Choose whether to delete or retain the contents of the project directory. By default, project directory contents are not deleted. To delete the contents of the project directory, click **Also delete contents**; all files and directories that are associated with the project are deleted.
4. Click **Yes** to delete the message set project. Alternatively, click **No** or press **Esc** to cancel the deletion.

Results

Related concepts:

"Message set projects" on page 1161
A *message set project* is a specialized container in which you create and maintain all the resources associated with one message set.

Related tasks:

"Creating a message set" on page 2842
Use the New Message Set wizard to create a message set.

Working with a message set

Complete a variety of tasks that are involved in working with a message set.

About this task

- "Configuring message set preferences"
- "Opening an existing message set" on page 2841
- "Creating a message set" on page 2842
- "Configuring logical properties: Message sets" on page 2846
- "Working with physical formats" on page 2847
- "Configuring documentation properties: Message sets" on page 2860
- "Deleting a message set" on page 2862
- "Applying a Quick Fix to a task list error" on page 2862

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

"Message sets overview" on page 1162
A message set is a container for grouping messages and associated message resources (elements, types, groups).

"Namespaces in the message model" on page 1201
Use namespaces to qualify message model object names.

Related tasks:

"Working with a message set project" on page 2838
Creating and deleting a message set.

"Generating documentation from message sets and message flows" on page 2962
You can generate documentation from your message sets, message flows, message definition files, message maps, Java files, ESQL files, and deployable WSDL files.

Configuring message set preferences

This task topic explains how to make changes to preferences that relate to message set processing. These preferences are for message set editors, message set model validation, and importing XML Schema.

About this task

To configure message set preferences:

Procedure

1. Open the Preferences window by clicking **Window > Preferences**.
2. In the left hand pane, expand **Broker Development > Message Sets** by clicking **+**. This displays the following options:
   - Editors
   - Validation
   - XML Schema Importer

3. View or make any necessary changes to the preferences for message set processing. These preferences are shown in the right hand area of the window.
4. When you have finished, click **Apply**. Alternatively, click **Restore Defaults** to return to the default settings for the displayed fields.
5. Close the Preferences window by clicking **OK**.

Results

Related reference:

"Message set preferences" on page 5366
Preferences for message sets.

"Message Set Editor and Message Definition Editor preferences" on page 5367
While looking at a large message set that contains a number of message definition files that have different namespaces, or multiple message definition files that have the same namespace, you might want to view the information in alternative ways to make it easier for you to visualize the structure of the message set. If you double-click the global construct, you open the message definition file in which the global construct is defined.

"Validation of the message model" on page 5369
You can customize some of the warning messages that are generated by message set validation. Use the Message Set Validation Preference page to do this.

"XML Schema Importer" on page 5370
Preferences for the message set XML Schema Importer.

Opening an existing message set

Open an existing message set in the Message Set editor so that you can view or edit its contents.

Before you begin

Before you start:

Create a message set by following the instructions in "Creating a message set" on page 2842.

Tip: Although you can open resource files with other editors you are advised to only use the WebSphere Message Broker Toolkit Message Set editor to work with message set files because this editor correctly validates changes made to the `messageSet.mset` files when they are saved. Other editors might not do this.

About this task

To open a message set so that you can view or edit its contents:

Procedure

1. Switch to the Broker Application Development perspective.
2. In the Broker Development view, right-click the `messageSet.mset` file of the message set that you are opening then click **Open** on the pop-up menu. This opens the Message Set editor for the selected file.

Results

You can now view or edit the file as required.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

"Message sets overview" on page 1162
A message set is a container for grouping messages and associated message resources (elements, types, groups).

"Namespaces in the message model" on page 1201
Use namespaces to qualify message model object names.

Related tasks:

"Working with a message set" on page 2840
Complete a variety of tasks that are involved in working with a message set.

"Deleting a message set" on page 2862
If you want to delete a message set from your message model, you must delete the message set project that contains the message set.

"Generating documentation from message sets and message flows" on page 2962
You can generate documentation from your message sets, message flows, message definition files, message maps, Java files, ESQL files, and deployable WSDL files.

Related reference:

"Message set editor" on page 6819
The Message set editor is the default editor provided by the Broker Application Development perspective for editing message set (`messageSet.mset`) files.

"Message set properties" on page 5371
Message sets have properties that you can set to define their characteristics and the way in which they are processed.

"Custom Wire Format message set properties" on page 5375
The tables define the properties that you can set for a Custom Wire Format message set.

"XML Wire Format message set properties" on page 5400
The following tables define the properties for the XML Wire Format for the message set.

"MRM XML: In-line DTDs and the DOCTYPE text property" on page 5407
You can include in-line DTDs in your messages, and you can specify additional information by setting the property *`DOCTYPE Text`*. The parser takes certain actions when constructing an output message.

"TDS Format message set properties" on page 5381
The following tables show the properties that you can set for a TDS format message set.

"TDS Mnemonics" on page 5391
The Tagged/Delimited String Format (TDS) uses mnemonics for a number of properties for a message set, complex type, or both.

"Default TDS message set properties" on page 5394
The following tables define the defaults for the message set properties for the TDS Format for each of the industry standard messages that you can define.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

Creating a message set

Use the New Message Set wizard to create a message set.

About this task

The New Message Set wizard also creates a new message set project.

Note: You can also use a Quick Start wizard to create a message set, a message set project, and other resource files that you need to create a new application.

The New Message Set wizard allows you to select what kinds of message format you want to model in your message set. The message domain and the physical format created is inferred from the selection that you make. Note, however, that you can change the inferred domain using the message set editor.

The options are:

- XML documents (general)
- Web services (SOAP)
- Binary data (for example, C or COBOL structures)
- Text data (for example, CSV, SWIFT, or HL7)
- MIME documents other than Web services
- Data for WebSphere Adapters
- Database records

The default value is `XML documents (general)`.

Below the list of message formats there are check boxes corresponding to each of the message formats. The check box corresponding to the message format that you selected is not available, but you can select any of the other check boxes to add other message formats to your message set.

If you later select a different default message domain, the checked state for the domain that you originally selected as the default does not change, but the check box is enabled.

As you can now select more than one message domain you can, for example, use the default value of **XML documents (general)** together with **Binary data (for example, C or COBOL structures)** and **Text data (for example, CSV, SWIFT or HL7)**. This results in the selection of the XMLNSC and MRM domains (to handle non-XML documents) within the same message set if you require this functionality.

The mapping between the selection, the domain, and the wire format created is described in the following table:

Selection	Inferred message domain	Physical format created
XML documents (general)	XMLNSC	XML
Web services (SOAP)	SOAP and XMLNSC	XML
Binary data (for example, C or COBOL structures)	MRM	CWF
Text data (for example, CSV, SWIFT, or HL7)	MRM	TDS
MIME documents other than Web services	MIME	None
Data for WebSphere Adapters	DataObject	None
Data for CORBA	DataObject	None
Database records	DataObject	None

Depending on your selection, an appropriate IBM supplied message will be imported into the message set.

Note: The XML physical format is created only in case the user switches to MRM XML.

If you click **Finish** on the second page of the New Message Set wizard, the message set that is created has the following default property values:

Property	Default value
Message Domain	XML documents (general)
Physical Format	XML Wire Format (XML1)
Namespace support	Enabled

To create a new message set:

Procedure

1. Switch to the Broker Application Development perspective.
2. Open the New Message Set wizard. To do this, right-click anywhere in the Broker Development view, then click **New** > **Message Set**.
3. In the **Message set name** field, type the name for the message set that you are creating. The name that you type is also displayed in the **Message set project name** field.
4. Optional: You can choose a different message set project name; to do this, type this name into the **Message set project name** field.
5. Optional: You can specify a directory in which you want to store the project contents. If you do not specify a directory, the default workspace is used. To specify a directory, first clear the **Use default** check box, then either type into the **Directory** field the location of the directory, or click **Browse** to see a list of the folders that you can choose from for the location of the directory.
6. Optional: If you want to create a new message set whose definition is based on existing message set, click **Message Set** in the **Copy message set contents from another message set** pane and choose from the list of message set definitions that are shown; then click **Finish**. The new message set (and the message set project that contains it) is created immediately and the New Message Set wizard automatically closes.
7. Optional: If you want to create a message set whose definition is not based on an existing message set, click **Next**. You are presented with the next pane which allows you to choose the type of message data that you want to process.
   a. Expand the list shown under **Select the type of message data that you will be working with most often** and choose a value from the list shown. The value that you choose determines the default message domain of the message set. If you choose `XML Documents (general)`, the default message domain XMLNSC is used.
   b. Optional: You can now select additional types of message data. Under **Select any other types of message data that you will be working with** there are check boxes for each of the following message data types:
      - XML documents (general)
      - Web services SOAP
      - Binary data (for example, C or COBOL structures)
      - Text data (for example, CSV, SWIFT or HL7)

- MIME documents other than Web services
- Data for WebSphere Adapters

Note: These check boxes correspond to the list of data types from which you chose the data type that you will be working with most often, but the check box that corresponds to the data type that you chose from that list is not available.
By default, all these check boxes are cleared. You can select any, or all of these check boxes, to add the corresponding data types to your message set.

If you select the check box for text data, either for the type of message data that you will be working with most often or as another type of message data that you will be working with, you can additionally choose from the displayed list of text messaging standards. This list is the same as that given in the description of the `Messaging Standard` property in "TDS Format message set properties" on page 5381.

c. Click **Next**. A new panel is displayed that summarizes some information about the message set that you have created. Specifically, it lists:

   Supported message domains

   Physical formats to be created

   IBM supplied messages to be imported

8. Click **Finish** on this page to create the message set, and the message set project that contains it. The New Message Set wizard closes.

Results

After the New Message Set wizard finishes, the message set editor is opened.

What to do next

You can now create some message definitions in the new message set. You can either create new message definitions from scratch, or create them based on existing artifacts such as WSDL, XSD, DTD, C, COBOL files, or EIS metadata. Use the Message Definition File wizard and the Message Definition File From wizard to help you with this.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

"Message set projects" on page 1161
A *message set project* is a specialized container in which you create and maintain all the resources associated with one message set.

"Message sets overview" on page 1162
A message set is a container for grouping messages and associated message resources (elements, types, groups).

"Namespaces in the message model" on page 1201
Use namespaces to qualify message model object names.

"Quick Start wizards overview" on page 1409
You can use a Quick Start wizard to set up the basic resources that are required to develop a message flow application. The wizard sets up and gives names to containers for the resources that you need when you develop a message flow.

Related tasks:

"Working with a message set project" on page 2838
Creating and deleting a message set.

"Deleting a message set project" on page 2839
Delete a message set project and, optionally, the contents of the associated project directory.

"Working with a message set" on page 2840
Complete a variety of tasks that are involved in working with a message set.

"Deleting a message set" on page 2862
If you want to delete a message set from your message model, you must delete the message set project that contains the message set.

"Generating documentation from message sets and message flows" on page 2962
You can generate documentation from your message sets, message flows, message definition files, message maps, Java files, ESQL files, and deployable WSDL files.

Related reference:

"Message set editor" on page 6819
The Message set editor is the default editor provided by the Broker Application Development perspective for editing message set (`messageSet.mset`) files.

"Message set properties" on page 5371
Message sets have properties that you can set to define their characteristics and the way in which they are processed.

"Custom Wire Format message set properties" on page 5375
The tables define the properties that you can set for a Custom Wire Format message set.

"XML Wire Format message set properties" on page 5400
The following tables define the properties for the XML Wire Format for the message set.

"TDS Format message set properties" on page 5381
The following tables show the properties that you can set for a TDS format message set.

"TDS Mnemonics" on page 5391
The Tagged/Delimited String Format (TDS) uses mnemonics for a number of properties for a message set, complex type, or both.

Configuring logical properties: Message sets

Configure the logical properties of a message set using the Message Set editor.

Before you begin

Before you start:

You must have completed the following task:

- "Creating a message set" on page 2842

If the `messageSet.mset` file for the appropriate message set is not already open in the Message Set editor, you must first open it as described in "Opening an existing message set" on page 2841.

About this task

This task topic describes how to configure the logical properties of a message set using the Message Set editor.

To configure the logical properties of a message set:

Procedure

1. Switch to the Broker Application Development perspective.
2. In the Message Set editor, in the Properties Hierarchy, click **Message Set**. This displays the logical properties of the selected message set in the Details view.
3. Configure to your requirements the logical properties that are shown in the Details view.

   Note: Property fields that are disabled cannot be altered. For example, the **Message Set ID** field is disabled because the message set ID is generated automatically when the message set is created; the **Message Set ID** must not then be altered.
4. Save your changes by clicking **File> Save** or by pressing Ctrl+S. Alternatively click **File> Save All** or press Ctrl+Shift+S.

Results

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

"Message sets overview" on page 1162
A message set is a container for grouping messages and associated message resources (elements, types, groups).

"Namespaces in the message model" on page 1201
Use namespaces to qualify message model object names.

"Message domains and parsers" on page 1159
WebSphere Message Broker supplies a range of *parsers* to parse and write message formats.

"Message set version and keywords" on page 1169
When you develop a message set, you can define the version of the message set, and other key information that you want to be associated with it.

Related tasks:

"Working with a message set" on page 2840
Complete a variety of tasks that are involved in working with a message set.

"Deleting a message set" on page 2862
If you want to delete a message set from your message model, you must delete the message set project that contains the message set.

Related reference:

"Message set editor" on page 6819
The Message set editor is the default editor provided by the Broker Application Development perspective for editing message set (`messageSet.mset`) files.

"Message set properties" on page 5371
Message sets have properties that you can set to define their characteristics and the way in which they are processed.

Working with physical formats

If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

Before you begin

Before you start:

You must have completed the following tasks:
- "Creating a message set" on page 2842
- "Opening an existing message set" on page 2841

About this task

This topic area covers the following tasks that relate to working with the physical properties of a message set:
- "Adding a Custom Wire Format (CWF)"
- "Configuring Custom Wire Format (CWF) properties: Message sets" on page 2849
- "Adding a TDS physical format" on page 2851
- "Configuring TDS properties: Message sets" on page 2852
- "Adding an XML wire format" on page 2853
- "Configuring XML Wire Format properties: Message sets" on page 2855
- "Renaming a physical format" on page 2856
- "Applying default physical format settings: Message sets" on page 2857
- "Removing a physical format" on page 2858
- "Observing 2007 U.S. changes to daylight saving time" on page 2859

Related concepts:

"Message sets overview" on page 1162
A message set is a container for grouping messages and associated message resources (elements, types, groups).

"Namespaces in the message model" on page 1201
Use namespaces to qualify message model object names.

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

Adding a Custom Wire Format (CWF):

You can add a Custom Wire Format (CWF) physical format layer to a message set by using the Message Set editor.

Before you begin

Before you start:

You must have completed the following tasks:
- "Creating a message set" on page 2842
- "Opening an existing message set" on page 2841

About this task

To add a CWF physical format layer to a message set:

Procedure

1. Switch to the Broker Application Development perspective.
2. In the Message Set editor, in the Properties Hierarchy, open the Add Custom Wire Format window by right-clicking **Custom Wire Formats**, then clicking **Add Custom Wire Format**.
3. On the Add Custom Wire Format window, specify the name that you want to use for the new CWF physical format. The default name is 'Binary1'.

   Tip: Physical format names must be unique across a message set. You are recommended to start the name of your new CWF physical format with 'CWF' or 'Binary', because this clearly identifies the type of the physical format that you are adding in relation to any of the other types.
4. Click **OK** to add the physical format layer to the message set. Alternatively, if you decide to cancel the process, click **Cancel** or press Esc on the keyboard.

   Tip: The new physical format layer is added with the relevant default property values. You can configure the message set properties according to your requirements, using the appropriate editor.
5. Save your changes either by clicking **File> Save** or by pressing Ctrl+S.

Related concepts:

"Message sets overview" on page 1162
A message set is a container for grouping messages and associated message resources (elements, types, groups).

"MRM Custom Wire Format" on page 1214
Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not separated by delimiters.

Related tasks:

"Configuring Custom Wire Format (CWF) properties: Message sets"
Configure the Custom Wire Format (CWF) properties of a message set using the Message Set editor.

"Renaming a physical format" on page 2856
Rename a physical format using the Message Set editor.

"Applying default physical format settings: Message sets" on page 2857
Apply the default settings to a physical format layer that has previously been added to a message set.

"Removing a physical format" on page 2858
You can remove a physical format layer from your message set.

Related reference:

"Message set editor" on page 6819
The Message set editor is the default editor provided by the Broker Application Development perspective for editing message set (`messageSet.mset`) files.

"Custom Wire Format message set properties" on page 5375
The tables define the properties that you can set for a Custom Wire Format message set.

Configuring Custom Wire Format (CWF) properties: Message sets:

Configure the Custom Wire Format (CWF) properties of a message set using the Message Set editor.

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 2842
- "Opening an existing message set" on page 2841
- "Adding a Custom Wire Format (CWF)" on page 2848

About this task

To configure the CWF properties of a message set:

Procedure

1. Switch to the Broker Application Development perspective.
2. In the Message Set editor, the **Custom Wire Formats** node of the Properties Hierarchy shows the name of each of the CWF physical formats that have been added to the message set. If the physical format names are not in view, expand **Custom Wire Formats** by clicking **+**.
3. Click the chosen CWF physical format so that the properties of this format appear in the Details view of the Message Set editor.
4. Configure the CWF properties shown in the Details view according to your requirements.
5. Save your changes either by clicking **File> Save** or by pressing Ctrl+S. Alternatively click **File> Save All** or press Ctrl+Shift+S.

Results

Related concepts:

"Message sets overview" on page 1162
A message set is a container for grouping messages and associated message resources (elements, types, groups).

"MRM Custom Wire Format" on page 1214
Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not separated by delimiters.

Related tasks:

"Adding a Custom Wire Format (CWF)" on page 2848
You can add a Custom Wire Format (CWF) physical format layer to a message set by using the Message Set editor.

"Renaming a physical format" on page 2856
Rename a physical format using the Message Set editor.

"Applying default physical format settings: Message sets" on page 2857
Apply the default settings to a physical format layer that has previously been added to a message set.

"Removing a physical format" on page 2858
You can remove a physical format layer from your message set.

"Configuring documentation properties: Message sets" on page 2860
Document a message set in the WebSphere Message Broker Toolkit.

Related reference:

"Message set editor" on page 6819
The Message set editor is the default editor provided by the Broker Application

Development perspective for editing message set (`messageSet.mset`) files.

"Custom Wire Format message set properties" on page 5375
The tables define the properties that you can set for a Custom Wire Format message set.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

Adding a TDS physical format:

Add a Tagged/Delimited String (TDS) physical format layer to a message set using the Message Set editor.

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 2842
- "Opening an existing message set" on page 2841

About this task

This task topic describes how to add a Tagged/Delimited String (TDS) physical format layer to a message set using the Message Set editor.

To add a TDS physical format layer to a message set:

Procedure

1. Switch to the Broker Application Development perspective.
2. In the Message Set editor, in the Properties Hierarchy, open the Add Tagged/Delimited String Format window by right-clicking **Tagged/Delimited String Formats** then clicking **Add Tagged/Delimited String Format** on the pop-up menu.
3. In the Add Tagged/Delimited String Format window, specify the name that you want to use for the new TDS format. The default name is 'Text1'.

   Tip: Physical format names must be unique across a message set. You are recommended to start the name of your new TDS physical format with 'TDS' or 'Text', because this clearly identifies the type of the physical format that you are adding in relation to any of the other types.
4. Click **OK** to add the physical format to the message set. Alternatively, if you decide to cancel the process, click **Cancel** or press **Esc** on the keyboard.

   Tip: The new physical format layer is added with the relevant default property values. You can configure the message set properties according to your requirements, using the Message Set editor.
5. Save your changes either by clicking **File> Save** or by pressing Ctrl+S.

Related concepts:

"Message sets overview" on page 1162
A message set is a container for grouping messages and associated message resources (elements, types, groups).

"MRM TDS format" on page 1221
The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

Related tasks:

"Configuring TDS properties: Message sets"
Configure the Tagged/Delimited String (TDS) format properties of a message set using the Message Set editor.

"Renaming a physical format" on page 2856
Rename a physical format using the Message Set editor.

"Applying default physical format settings: Message sets" on page 2857
Apply the default settings to a physical format layer that has previously been added to a message set.

"Removing a physical format" on page 2858
You can remove a physical format layer from your message set.

Related reference:

"Message set editor" on page 6819
The Message set editor is the default editor provided by the Broker Application Development perspective for editing message set (`messageSet.mset`) files.

"TDS Format message set properties" on page 5381
The following tables show the properties that you can set for a TDS format message set.

"TDS Mnemonics" on page 5391
The Tagged/Delimited String Format (TDS) uses mnemonics for a number of properties for a message set, complex type, or both.

"Default TDS message set properties" on page 5394
The following tables define the defaults for the message set properties for the TDS Format for each of the industry standard messages that you can define.

Configuring TDS properties: Message sets:

Configure the Tagged/Delimited String (TDS) format properties of a message set using the Message Set editor.

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 2842
- "Opening an existing message set" on page 2841
- "Adding a TDS physical format" on page 2851

About this task

To configure the *TDS* physical format properties of a message set, do the following:

Procedure

1. Switch to the Broker Application Development perspective.
2. In the Message Set editor, the **Tagged/Delimited String Formats** node of the Properties Hierarchy shows the name of each of the TDS physical formats that have been added to the message set. If the physical format names are not in view, expand **Tagged/Delimited String Formats** by clicking **+**.

3. Click the chosen TDS physical format so that the properties of this format appear in the Details view of the Message Set editor.
4. Configure the TDS properties shown in the Details view according to your requirements.
5. Save your changes either by clicking **File> Save** or by pressing Ctrl+S. Alternatively click **File> Save All** or press Ctrl+Shift+S.

Results

Related concepts:

"Message sets overview" on page 1162
A message set is a container for grouping messages and associated message resources (elements, types, groups).

"MRM TDS format" on page 1221
The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

Related tasks:

"Adding a TDS physical format" on page 2851
Add a Tagged/Delimited String (TDS) physical format layer to a message set using the Message Set editor.

"Renaming a physical format" on page 2856
Rename a physical format using the Message Set editor.

"Applying default physical format settings: Message sets" on page 2857
Apply the default settings to a physical format layer that has previously been added to a message set.

"Removing a physical format" on page 2858
You can remove a physical format layer from your message set.

"Configuring documentation properties: Message sets" on page 2860
Document a message set in the WebSphere Message Broker Toolkit.

Related reference:

"Message set editor" on page 6819
The Message set editor is the default editor provided by the Broker Application Development perspective for editing message set (`messageSet.mset`) files.

"TDS Format message set properties" on page 5381
The following tables show the properties that you can set for a TDS format message set.

"TDS Mnemonics" on page 5391
The Tagged/Delimited String Format (TDS) uses mnemonics for a number of properties for a message set, complex type, or both.

"Default TDS message set properties" on page 5394
The following tables define the defaults for the message set properties for the TDS Format for each of the industry standard messages that you can define.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"TDS Industry standard formats" on page 6265
WebSphere Message Broker supports the ACORD AL3, CSV, EDIFACT, FIX, HL7, SWIFT, TLOG, and X12 standards.

Adding an XML wire format:

You can add an XML wire format physical format layer to a message set by using the Message Set editor.

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 2842
- "Opening an existing message set" on page 2841

About this task

This task topic describes how to add an XML wire format physical format layer to a message set using the Message Set editor.

To add an XML physical format layer to a message set:

Procedure

1. Switch to the Broker Application Development perspective.
2. In the Message Set editor, in the Properties Hierarchy, open the Add XML Wire Format window by right-clicking **XML Wire Formats**, then clicking **Add XML Wire Format** on the pop-up menu.
3. On the Add XML Wire Format window, specify the name that you want to use for the new XML wire format. The default name is 'XML1'.

   Tip: Physical format names must be unique across a message set. You are recommended to start the name of your new XML physical format with 'XML', because this clearly identifies the type of the physical format that you are adding in relation to any of the other types.
4. Click **OK** to add the physical format layer. Alternatively, if you decide to cancel the process, click **Cancel** or press Esc on the keyboard.

   Tip: The new physical format layer is added with the relevant default property values. You can configure the message set properties according to your requirements, using the appropriate editor.
5. Save your changes either by clicking **File> Save** or by pressing Ctrl+S.

Related concepts:

"Message sets overview" on page 1162
A message set is a container for grouping messages and associated message resources (elements, types, groups).

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

Related tasks:

"Configuring XML Wire Format properties: Message sets" on page 2855
Configure the XML Wire Format properties of a message set using the Message Set editor.

"Renaming a physical format" on page 2856
Rename a physical format using the Message Set editor.

"Applying default physical format settings: Message sets" on page 2857
Apply the default settings to a physical format layer that has previously been added to a message set.

"Removing a physical format" on page 2858
You can remove a physical format layer from your message set.

Related reference:

"Message set editor" on page 6819
The Message set editor is the default editor provided by the Broker Application Development perspective for editing message set (`messageSet.mset`) files.

"XML Wire Format message set properties" on page 5400
The following tables define the properties for the XML Wire Format for the message set.

"MRM XML: In-line DTDs and the DOCTYPE text property" on page 5407
You can include in-line DTDs in your messages, and you can specify additional information by setting the property *DOCTYPE Text*. The parser takes certain actions when constructing an output message.

Configuring XML Wire Format properties: Message sets:

Configure the XML Wire Format properties of a message set using the Message Set editor.

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 2842
- "Opening an existing message set" on page 2841
- "Adding an XML wire format" on page 2853

About this task

To configure the XML wire format properties of a message set:

Procedure

1. Switch to the Broker Application Development perspective.
2. In the Message Set editor, the **XML Wire Formats** node of the Properties Hierarchy shows the name of each of the XML physical formats that have been added to the message set. If the physical format names are not in view, expand **XML Wire Formats** by clicking **+**.
3. Click the chosen XML physical format so that the properties of this format appear in the Details view of the Message Set editor.
4. Configure the XML wire format properties shown in the Details view according to your requirements.
5. Save your changes either by clicking **File> Save** or by pressing Ctrl+S. Alternatively click **File> Save All** or press Ctrl+Shift+S.

Results

Related concepts:

"Message sets overview" on page 1162
A message set is a container for grouping messages and associated message resources (elements, types, groups).

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

Related tasks:

"Adding an XML wire format" on page 2853
You can add an XML wire format physical format layer to a message set by using the Message Set editor.

"Renaming a physical format"
Rename a physical format using the Message Set editor.

"Applying default physical format settings: Message sets" on page 2857
Apply the default settings to a physical format layer that has previously been added to a message set.

"Removing a physical format" on page 2858
You can remove a physical format layer from your message set.

"Configuring documentation properties: Message sets" on page 2860
Document a message set in the WebSphere Message Broker Toolkit.

Related reference:

"Message set editor" on page 6819
The Message set editor is the default editor provided by the Broker Application Development perspective for editing message set (`messageSet.mset`) files.

"XML Wire Format message set properties" on page 5400
The following tables define the properties for the XML Wire Format for the message set.

"MRM XML: In-line DTDs and the DOCTYPE text property" on page 5407
You can include in-line DTDs in your messages, and you can specify additional information by setting the property *`DOCTYPE Text`*. The parser takes certain actions when constructing an output message.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

Renaming a physical format:

Rename a physical format using the Message Set editor.

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 2842
- "Opening an existing message set" on page 2841

This task assumes that you have added one or more physical formats to the message set that you are working with. For further information see "Adding a Custom Wire Format (CWF)" on page 2848 or "Adding an XML wire format" on page 2853 or "Adding a TDS physical format" on page 2851.

About this task

This task topic describes how to rename a physical format using the Message Set editor.

To rename a physical format previously added to the message model:

Procedure

1. Close all message definition (`.mxsd`) files that are currently open in the Message Definition editor, otherwise you will not be able to rename the physical format.
2. Switch to the Broker Application Development perspective.
3. In the Message Set editor, the Properties Hierarchy shows the name of each of the physical formats that have been added to the message set. If the physical format names are not in view, expand **XML Wire Formats**, **Custom Wire Formats**, or **Tagged/Delimited String Formats** by clicking **+**.
4. Right-click the physical format that you want to rename then click **Rename** on the pop-up menu to open the "Rename wire format" window.
5. In the "Rename wire format" window, type the new name for the physical format. The renaming operation modifies all of the message definition files in the message set and saves the amended message set file.
6. Click **Finish** to rename the physical format and save the message set file.

Results

Related concepts:

"Message sets overview" on page 1162
A message set is a container for grouping messages and associated message resources (elements, types, groups).

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Removing a physical format" on page 2858
You can remove a physical format layer from your message set.

Related reference:

"Message set editor" on page 6819
The Message set editor is the default editor provided by the Broker Application Development perspective for editing message set (`messageSet.mset`) files.

Applying default physical format settings: Message sets:

Apply the default settings to a physical format layer that has previously been added to a message set.

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 2842
- "Opening an existing message set" on page 2841

The tasks in this topic area assume that you have added one or more physical formats to the relevant message set. For further information see "Adding a Custom

Wire Format (CWF)" on page 2848 or "Adding an XML wire format" on page 2853 or "Adding a TDS physical format" on page 2851.

About this task

To apply the default settings to a physical format that has previously been added to a message set:

Procedure

1. Switch to the Broker Application Development perspective.
2. In the Message Set editor, in the Properties Hierarchy, right-click the physical format to which you want to apply the default settings then click **Apply default physical format settings** on the pop-up menu.

Results

The default settings are applied to the physical format that you have selected. No warning appears beforehand.

Related concepts:

"Message sets overview" on page 1162
A message set is a container for grouping messages and associated message resources (elements, types, groups).

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Removing a physical format"
You can remove a physical format layer from your message set.

Related reference:

"Message set editor" on page 6819
The Message set editor is the default editor provided by the Broker Application Development perspective for editing message set (`messageSet.mset`) files.

Removing a physical format:

You can remove a physical format layer from your message set.

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 2842
- "Opening an existing message set" on page 2841

The tasks in this topic area assume that you have added one or more physical formats to a message set. For further information see "Adding a Custom Wire

Format (CWF)" on page 2848 or "Adding an XML wire format" on page 2853 or "Adding a TDS physical format" on page 2851.

About this task

To remove a physical format layer from your message set:

Procedure

1. Close any message definition files that are currently open in the Message Definition editor, otherwise you will not be able to remove the physical format.
2. Switch to the Broker Application Development perspective.
3. In the Message Set editor, the Properties Hierarchy shows the name of each of the physical formats that have been added to the message set. If the physical format names are not in view, expand **XML Wire Formats**, **Custom Wire Formats**, or **Tagged/Delimited Wire Formats**, by clicking **+**.
4. Right-click the physical format that you want to remove, then click **Delete** on the pop-up menu.

   Tip: If you decide to proceed with deleting the physical format, all of the message definition files under the current message set are modified and the amended message set file is saved.
5. Click **Finish** to remove the physical format, or click **Cancel** to return to the Broker Application Development perspective without making any changes. Pressing Esc also returns you to the Broker Application Development perspective without making any changes.

Results

Related concepts:

"Message sets overview" on page 1162
A message set is a container for grouping messages and associated message resources (elements, types, groups).

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

Related reference:

"Message set editor" on page 6819
The Message set editor is the default editor provided by the Broker Application Development perspective for editing message set (`messageSet.mset`) files.

Observing 2007 U.S. changes to daylight saving time

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 2842

- "Opening an existing message set" on page 2841

This task assumes that you have added and configured one or more physical formats to existing message sets. For further information see: "Working with physical formats" on page 2847.

About this task

This task describes how to ensure that the message sets observe daylight saving time (DST) in line with the 2007 U.S. changes.

If your message sets use a named time zone that is *not* changing DST in line with the 2007 U.S. changes, you do not need to do anything.

If you are using a GMT-04:00, GMT-05:00, GMT-06:00, GMT-07:00, or GMT-08:00 named time zone with DST, that must observe DST in line with the 2007 U.S. changes, do the following steps on every computer on which the broker is running:

1. Set the environment variable MQSI_USE_NEW_US_DST to an initial value: Y, for example.
2. Restart the broker to use the changed DST.

Related concepts:

"Message sets overview" on page 1162
A message set is a container for grouping messages and associated message resources (elements, types, groups).

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Starting and stopping a broker" on page 921
Run the appropriate command to start or stop a broker.

Related reference:

"Message set editor" on page 6819
The Message set editor is the default editor provided by the Broker Application Development perspective for editing message set (`messageSet.mset`) files.

"Custom Wire Format message set properties" on page 5375
The tables define the properties that you can set for a Custom Wire Format message set.

"XML Wire Format message set properties" on page 5400
The following tables define the properties for the XML Wire Format for the message set.

"TDS Format message set properties" on page 5381
The following tables show the properties that you can set for a TDS format message set.

Configuring documentation properties: Message sets

Document a message set in the WebSphere Message Broker Toolkit.

Before you begin

Before you start:

Complete the following tasks:

- "Creating a message set" on page 2842
- "Opening an existing message set" on page 2841

About this task

To configure the documentation for a message set:

Procedure

1. In the Message Set editor **Properties Hierarchy**, click **Message set**. The documentation text field appears in the Details view, with all the other logical properties of the message set.
2. Configure the documentation properties shown in the Details view to your requirements.

   You can use the `Documentation` property to set user-defined keywords and their value. These keywords are propagated to the broker when you deploy the message set in a BAR file. These keywords are used to give additional information about the message set when you display deployed message set properties in the WebSphere Message Broker Toolkit. See "Message set version and keywords" on page 1169 for more information.
3. Save your changes by clicking **File** > **Save**, or by pressing Ctrl+S. Alternatively, click **File** > **Save All**, or press Ctrl+Shift+S.

Results

Related concepts:

"Message sets overview" on page 1162
A message set is a container for grouping messages and associated message resources (elements, types, groups).

"Message set version and keywords" on page 1169
When you develop a message set, you can define the version of the message set, and other key information that you want to be associated with it.

Related tasks:

"Working with a message set" on page 2840
Complete a variety of tasks that are involved in working with a message set.

"Generating documentation from message sets and message flows" on page 2962
You can generate documentation from your message sets, message flows, message definition files, message maps, Java files, ESQL files, and deployable WSDL files.

Related reference:

"Message set editor" on page 6819
The Message set editor is the default editor provided by the Broker Application Development perspective for editing message set (`messageSet.mset`) files.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Guidance for defining keywords" on page 4017
You can add extra information to an object in the form of one or more keywords.

Deleting a message set

If you want to delete a message set from your message model, you must delete the message set project that contains the message set.

Procedure

1. Switch to the Broker Application Development perspective.
2. In the Broker Development view, right-click the message set project folder that contains the message set that you want to delete and click **Delete** on the pop-up menu. This opens the Confirm Project Delete window, which asks whether you want to delete the message set project that you have specified.
3. Click **Also delete contents .....** to delete the contents of the message set project, or click **Do not delete contents** to cancel the deletion of the message set project. Pressing the Esc key on your keyboard also cancels the deletion of the message set project.

Results

Important: When you delete a message set project, the action cannot be undone after you have confirmed the deletion. All folders and associated files for the message set project are deleted.

Related concepts:

"Message sets overview" on page 1162
A message set is a container for grouping messages and associated message resources (elements, types, groups).

"Namespaces in the message model" on page 1201
Use namespaces to qualify message model object names.

Applying a Quick Fix to a task list error

During the creation, migration and manipulation of message models, warnings or errors might occur; these are listed in the Problems view of the Broker Application Development perspective. Some of these warnings or errors can be cleared by applying a Quick Fix.

About this task

The types of warnings or errors that can be cleared using a Quick Fix are those where the construct has a broken link, where the construct has a facet that is not legal, or where the construct has been imported, and where a warning or error has occurred, but has been kept to ensure the integrity of structure that is being imported. This allows you to fix the problem in the most appropriate way.

To apply a Quick Fix:

Procedure

1. Switch to the Broker Application Development perspective.
2. Ensure that the Problems view is visible in the Broker Application Development perspective of the WebSphere Message Broker Toolkit. If the Problems view is not visible, from the WebSphere Message Broker Toolkit menu, click **Window** > **Show View** > **Problems**.
3. In the Problems view, right-click the task list warning or error that you want to apply the Quick Fix to, then click **Quick Fix**. Note that **Quick Fix** might not be available for the problem that you are trying to fix.

4. Step through the windows that are displayed, making the selections that are required to ensure that the fix is applied in the appropriate way.

Results

When the Quick Fix has successfully been applied to the task list warning or error, it is removed from the Problems view.

Related concepts:

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

"Message model integrity" on page 1210
When you create your message model, it is important that it is internally consistent.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Message model task list errors that have a quick fix" on page 6336
You can apply a quick fix to some message modeling task list warnings or errors to correct them.

Working with a message definition file

Create, open, and delete a message definition file.

Before you begin

Before you start:

You must have completed the following task:

- "Creating a message set" on page 2842

About this task

This topic area describes the tasks that are involved in working with a message definition file:

- "Opening an existing message definition file" on page 2864
- "Creating a message definition file" on page 2865
- "Deleting a message definition file" on page 2869

Related concepts:

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

"Message definition files" on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

"XML Schema" on page 1172
XML Schema is an international standard from the World Wide Web Consortium (W3C) that defines a language for describing the structure of XML documents.

"XML Schema restrictions in message sets" on page 1172
Some XML Schema 1.0 features are not supported when message models are contained in message sets.

"XML Schema extensions in message sets" on page 1173
WebSphere Message Broker provides some additional facilities that are not specified in the XML Schema 1.0 specification. When using a message set, further extensions are provided.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Opening an existing message definition file

This task topic describes how to open an existing message definition file in the Message Definition editor; you can then view and edit the contents of the file.

About this task

To open an existing message definition file:

Procedure

1. Switch to the Broker Application Development perspective.
2. In the Broker Development view, right-click the message definition file (file extension `*.mxsd`) that you want to open, and select **Open**. This opens the Message Definition editor for the message definition file that you have specified.

   Tip: The Eclipse framework lets you open resource files with other editors. However, you are advised to use only the WebSphere Message Broker Toolkit Message Definition editor to work with message definition files, because this editor correctly validates any changes that are made to the message definition files. Other editors might not do this.
3. View or edit the data in the file as required.

Results

Related concepts:

"Message definition files" on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

"XML Schema" on page 1172
XML Schema is an international standard from the World Wide Web Consortium (W3C) that defines a language for describing the structure of XML documents.

"XML Schema restrictions in message sets" on page 1172
Some XML Schema 1.0 features are not supported when message models are contained in message sets.

"XML Schema extensions in message sets" on page 1173
WebSphere Message Broker provides some additional facilities that are not specified in the XML Schema 1.0 specification. When using a message set, further extensions are provided.

Related tasks:

"Working with a message definition file" on page 2863
Create, open, and delete a message definition file.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message Definition editor" on page 6804
The Message Definition editor is the default editor provided by the Broker Application Development perspective for editing message definition (`.mxsd`) files.

"Message definition file properties" on page 5409
The properties of a message definition file.

Creating a message definition file

Creating an empty message definition file to contain your message model objects.

Before you begin

Before you start:

You must have completed the following task:

- "Creating a message set" on page 2842

About this task

You must create a message definition file before you can create the message model objects. The message definition file contains the logical and physical model definitions of the objects in XML Schema form.

You can create the message definition file in one of the following ways:

- Create the message definition file from scratch, see "Creating a message definition file from scratch" on page 2866.
- Base your message definition file on an existing resource (for example, an XML Schema file, an IBM supplied message, an XML DTD file, a C header file, a COBOL file, or a WSDL file), see "Creating a message definition file from an existing resource" on page 2867.
- Copy a message definition file from one message set to another.

  If you do copy a message definition file from one message set to another, you must check that the source and target message sets have identical physical formats, and that namespaces are enabled.

  If the source and target namespaces do not have identical formats, the physical format of the message definition file might be replaced by the default information applied to the target message set.

Related concepts:

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

"Message definition files" on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

Related tasks:

"Adding MRM message model objects" on page 2870
Various tasks are involved in adding message model objects to a message definition file.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

"Working with data structures" on page 2930
You can create a message definition file in a message set by importing from XML Schema, XML DTD, SCA import or export components, IBM supplied messages, WSDL definitions, IDL files, C header files, and COBOL copybooks. This topic area describes how to import from these data structures using the command line or the WebSphere Message Broker Toolkit.

"Importing file systems into the WebSphere Message Broker Toolkit" on page 2931
You can import file systems into the WebSphere Message Broker Toolkit by using the Import wizard, by dragging, or by copying.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message definition file properties" on page 5409
The properties of a message definition file.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"New message definition file wizard: Create a new message definition file from scratch" on page 6361
Create a new message definition file by using the New message definition file wizard.

Creating a message definition file from scratch:

Create an empty message definition file to contain message model objects.

Before you begin

Before you start:

You must have completed the following task:

- "Creating a message set" on page 2842

About this task

You must create a message definition file before you can create the message model objects. The message definition file contains the logical and physical model definitions of the objects in XML Schema form.

To create an empty message definition file from scratch:

Procedure

1. Switch to the Broker Application Development perspective.
2. Open the New Message Definition File wizard.

   To do this, right-click the message set project in the Broker Development view that you are adding the message definition file to, and click **New> Message Definition File** on the pop-up menu. The **Message Definition File** panel of the wizard is displayed. The target message set list is filtered to only show artifacts in the active working set.
3. Click the message set, type a name into the File name field, and click **Next**.
4. Step through the remainder of the wizard, completing the details as required.

Results

The new empty message definition file, with the name that you have specified and a file extension of `*.mxsd`, opens in the Message Definition editor; you can use the editor to create your own message definitions. If you have chosen to use a target namespace, a directory structure that is based on the URI that you have supplied is created. The new message definition file is placed within this directory structure, which appears in the Broker Development view.

Related concepts:

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

"Message definition files" on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

Related tasks:

"Adding MRM message model objects" on page 2870
Various tasks are involved in adding message model objects to a message definition file.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

"Working with data structures" on page 2930
You can create a message definition file in a message set by importing from XML Schema, XML DTD, SCA import or export components, IBM supplied messages, WSDL definitions, IDL files, C header files, and COBOL copybooks. This topic area describes how to import from these data structures using the command line or the WebSphere Message Broker Toolkit.

"Importing file systems into the WebSphere Message Broker Toolkit" on page 2931
You can import file systems into the WebSphere Message Broker Toolkit by using the Import wizard, by dragging, or by copying.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message definition file properties" on page 5409
The properties of a message definition file.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"New message definition file wizard: Create a new message definition file from scratch" on page 6361
Create a new message definition file by using the New message definition file wizard.

Creating a message definition file from an existing resource:

You must create a message definition file before you can create the message model objects. The message definition file contains the logical and physical model definitions of the objects in XML schema form.

Before you begin

You must have completed the following task:

- "Creating a message set" on page 2842

About this task

To create a new message definition file that is based on an existing resource:

Procedure

1. Switch to the Broker Application Development perspective.
2. Open the appropriate New Message Definition File From wizard.

   Right-click the message set project in the Broker Development view that you are adding the message definition file to, and click **New> Message Definition File From**. A submenu shows the list of resources from which you can choose.
3. Choose the resource on which to base your new message definition. Click one of the following resources:
   - **C Header File**
   - **COBOL File**
   - **CORBA IDL File**
   - **Database Definition File**
   - **IBM Supplied Message**
   - **SCA Import or Export**
   - **WSDL File**
   - **XML DTD File**
   - **XML Schema File**

   The first panel of the corresponding wizard is displayed.
4. Step through the remainder of the wizard supplying the details as required. For more information about using the New Message Definition File wizards, see "New message definition file wizards" on page 6360.

Results

The new message definition file, with the name that you have specified and a file extension of `.mxsd`, opens in the Message Definition editor. You can use the editor to create your own message definitions. If you have chosen to use a target namespace, a directory structure that is based on the URI that you have supplied is created. The new message definition file is placed within this directory structure, which is shown in the Broker Development view.

Related concepts:

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

"Message definition files" on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

Related tasks:

"Adding MRM message model objects" on page 2870
Various tasks are involved in adding message model objects to a message definition file.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

"Working with data structures" on page 2930
You can create a message definition file in a message set by importing from XML Schema, XML DTD, SCA import or export components, IBM supplied messages, WSDL definitions, IDL files, C header files, and COBOL copybooks. This topic area describes how to import from these data structures using the command line or the WebSphere Message Broker Toolkit.

"Importing file systems into the WebSphere Message Broker Toolkit" on page 2931
You can import file systems into the WebSphere Message Broker Toolkit by using the Import wizard, by dragging, or by copying.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message definition file properties" on page 5409
The properties of a message definition file.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"New message definition file wizard: Create a new message definition file from scratch" on page 6361
Create a new message definition file by using the New message definition file wizard.

Deleting a message definition file

You can delete a message definition file from your message model.

Before you begin

Before you start:

Before you delete a message definition file, you can list resources that refer to that file and would therefore be affected by the deletion; for more information, see "Showing resource references" on page 1447.

About this task

To delete a message definition file from your message model:

Procedure

1. Switch to the Broker Application Development perspective.
2. In the Broker Development view, right-click the message definition file (file extension `*.mxsd`) that you want to delete, then click **Delete**. Alternatively, select the message definition file that you want to delete in the Broker Development view, then, from the menu bar, click **Edit** > **Delete**, or press the Delete key.
3. In the Confirm Resource Delete window, click **Yes** to delete the message definition file. Click **No**, or press the Esc key, to cancel the deletion of the message definition file.

Results

Important: All files and objects that are associated with the message definition file are deleted. This action cannot be undone.

Related concepts:

"Message definition files" on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

Related reference:

"Message definition file properties" on page 5409
The properties of a message definition file.

Working with MRM message model objects

Add, configure, and delete objects.

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 2842
- "Creating a message definition file" on page 2865

About this task

This topic area describes the tasks that are involved in working with message model objects:

- "Adding MRM message model objects"
- "Configuring MRM message model objects" on page 2896
- "Deleting objects" on page 2919

Related concepts:

"Message definition files" on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

Related tasks:

"Working with a message definition file" on page 2863
Create, open, and delete a message definition file.

Related reference:

"Message Definition editor" on page 6804
The Message Definition editor is the default editor provided by the Broker Application Development perspective for editing message definition (`.mxsd`) files.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

Adding MRM message model objects

Various tasks are involved in adding message model objects to a message definition file.

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 2842
- "Creating a message definition file" on page 2865

Before starting any of the tasks in this topic area, you must first open the message definition file to which you want to add message model objects in the Message Definition editor. See "Opening an existing message definition file" on page 2864 for further details.

About this task

This topic area describes the tasks that are involved in adding message model objects to a message definition file:

- "Adding a message" on page 2872
- "Adding a message from a global element" on page 2873
- "Adding a global element" on page 2876
- "Adding a local element" on page 2877
- "Adding an element reference" on page 2878
- "Adding a wildcard element" on page 2880
- "Adding a global attribute" on page 2881
- "Adding a local attribute" on page 2882
- "Adding an attribute reference" on page 2884
- "Adding a wildcard attribute" on page 2885
- "Adding a simple type" on page 2886
- "Adding a complex type" on page 2889
- "Adding a simple type to an element or attribute" on page 2904
- "Adding a complex type to an element" on page 2905
- "Adding a global group" on page 2890
- "Adding an attribute group" on page 2892
- "Adding a group reference" on page 2894

Related concepts:

"Message definition files" on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

"Message model object identification" on page 1191
Objects in a message model (elements, attributes, types, groups) are identified by their name only.

Related tasks:

"Working with a message definition file" on page 2863
Create, open, and delete a message definition file.

Related reference:

"Message Definition editor" on page 6804
The Message Definition editor is the default editor provided by the Broker Application Development perspective for editing message definition (`.mxsd`) files.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

Adding a message:

Add a message to a message model.

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 2842
- "Creating a message definition file" on page 2865
- "Opening an existing message definition file" on page 2864

About this task

To add a message to your message definition file:

Procedure

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the WebSphere Message Broker Toolkit. If the Outline view is not visible, from the WebSphere Message Broker Toolkit menu, click **Window > Show View> Outline**.
3. In the Outline view, right-click **Messages** then click **Add Message** on the pop-up menu. A simple message is immediately added to your message model and is assigned a default name.
4. Either type a new name for the message or press **Enter** to accept the default.

   Tip: When you add a message to your message model, an associated complex type and global element with the same name as the message are also created. The global element and the message cannot have different names and changing the name of one changes the names of both. The complex type can be renamed.

Results

You can now configure the properties of the message to your exact requirements. For further information about configuring message model objects see "Configuring MRM message model objects" on page 2896.

Related concepts:

"Message definition files" on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

"Message model objects: messages" on page 1175
A *message* describes the structure and content of a set of data that is passed from one application to another.

"Multipart messages" on page 1191
A *multipart message* contains one or more other messages within its structure. The

contained message is sometimes referred to as an *embedded message*.

Related tasks:

"Adding MRM message model objects" on page 2870
Various tasks are involved in adding message model objects to a message definition file.

"Configuring MRM message model objects" on page 2896
Various tasks are involved in configuring MRM message model objects

"Deleting objects" on page 2919
Delete an object from your message model.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Message logical properties" on page 5449
This section describes the logical properties of a message.

"Message CWF properties" on page 5473
There are no properties to show.

"Message XML properties" on page 5495
The following tables describe the XML properties of a message.

"Message TDS properties" on page 5531
Message TDS properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

Adding a message from a global element:

Add a message from a global element to a message model.

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 2842
- "Creating a message definition file" on page 2865
- "Opening an existing message definition file" on page 2864
- "Adding a global element" on page 2876 (This must be a global element of complex type)

About this task

To add a message from a global element to your message model:

Procedure

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the WebSphere Message Broker Toolkit. If the Outline view is not visible, from the WebSphere Message Broker Toolkit menu, click **Window > Show View> Outline**.

3. In the Outline view, right-click **Messages** then click **Add Message From Global Element** on the pop-up menu to open the Add Message From Global Element window. This window lists all the global elements of a complex type that are not already associated with a message.
4. In the **Select a global element of complex type that is not already used for a message** list, click the global element that you want to use to create your message.
5. Click **OK**. This immediately adds a message with the same name as the selected global element to your message model.

Results

You can now configure the properties of the message to your exact requirements. For further information on configuring message model objects see "Configuring MRM message model objects" on page 2896.

Related concepts:

"Message definition files" on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

"Message model objects: messages" on page 1175
A *message* describes the structure and content of a set of data that is passed from one application to another.

"Multipart messages" on page 1191
A *multipart message* contains one or more other messages within its structure. The contained message is sometimes referred to as an *embedded message*.

"Message model objects: elements" on page 1176
An *element* is a named piece of information (a field) within a message, with a meaning that is agreed by the applications that create and process the message.

Related tasks:

"Adding MRM message model objects" on page 2870
Various tasks are involved in adding message model objects to a message definition file.

"Configuring MRM message model objects" on page 2896
Various tasks are involved in configuring MRM message model objects

"Deleting objects" on page 2919
Delete an object from your message model.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Message logical properties" on page 5449
This section describes the logical properties of a message.

"Message CWF properties" on page 5473
There are no properties to show.

"Message XML properties" on page 5495
The following tables describe the XML properties of a message.

"Message TDS properties" on page 5531
Message TDS properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

Adding a message from a global type:

Add a message from a global type to your message model.

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 2842
- "Creating a message definition file" on page 2865
- "Opening an existing message definition file" on page 2864
- "Adding a global element" on page 2876 (This must be a global element of complex type)

About this task

To add a message from a global type to your message model:

Procedure

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the WebSphere Message Broker Toolkit. If the Outline view is not visible, from the WebSphere Message Broker Toolkit menu, click **Window > Show View> Outline**.
3. In the Outline view, right-click **Messages** then click **Add Message From Global Type** on the pop-up menu to open the Add Message From Global Type window. This window lists all the global complex types that are not already associated with a message.
4. In the **Select a global complex type** list, click the global complex type that you want to use to create your message.
5. Click **OK**. This immediately adds a message with the same name as the selected global complex type to your message model.

Results

You can now configure the properties of the message to your exact requirements. For further information on configuring message model objects see "Configuring MRM message model objects" on page 2896.

Related concepts:

"Message definition files" on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

"Message model objects: messages" on page 1175
A *message* describes the structure and content of a set of data that is passed from one application to another.

"Multipart messages" on page 1191
A *multipart message* contains one or more other messages within its structure. The contained message is sometimes referred to as an *embedded message*.

"Message model objects: types" on page 1177
Types describe the data content of elements.

Related tasks:

"Adding MRM message model objects" on page 2870
Various tasks are involved in adding message model objects to a message definition file.

"Configuring MRM message model objects" on page 2896
Various tasks are involved in configuring MRM message model objects

"Deleting objects" on page 2919
Delete an object from your message model.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Message logical properties" on page 5449
This section describes the logical properties of a message.

"Message CWF properties" on page 5473
There are no properties to show.

"Message XML properties" on page 5495
The following tables describe the XML properties of a message.

"Message TDS properties" on page 5531
Message TDS properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

Adding a global element:

Add a global element to a message model.

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 2842
- "Creating a message definition file" on page 2865
- "Opening an existing message definition file" on page 2864

About this task

To add a global element to your message model:

Procedure

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the WebSphere Message Broker Toolkit. If the Outline view is not visible, from the WebSphere Message Broker Toolkit menu, click **Window > Show View> Outline**.
3. In the Outline view, right-click **Elements and Attributes** then click **Add Global Element** on the pop-up menu. This adds a global element of type string to your message model, and assigns a default name.
4. Either type a new name for the global element or press Enter to accept the default.

Results

You can now configure the global element to your requirements. For further information on configuring message model objects see "Configuring MRM message model objects" on page 2896.

Related concepts:

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

"Message model objects: elements" on page 1176
An *element* is a named piece of information (a field) within a message, with a meaning that is agreed by the applications that create and process the message.

Related tasks:

"Configuring MRM message model objects" on page 2896
Various tasks are involved in configuring MRM message model objects

"Deleting objects" on page 2919
Delete an object from your message model.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Global element logical properties" on page 5430
The logical properties of a global element.

"Global element CWF properties" on page 5463
There are no CWF properties to show for a global element.

"Global element XML properties" on page 5486
The properties, and their permissible values, vary according to the type of object.

"Global element TDS properties" on page 5513
The properties, and their permissible values, vary according to the object type.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

Adding a local element:

Add a local element to a message, type, group, or complex element.

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 2842
- "Creating a message definition file" on page 2865
- "Opening an existing message definition file" on page 2864

This task assumes that you have previously added the relevant message, type, group, or complex element to your message model.

About this task

To add a local element to a message, type, group, or complex element:

Procedure

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the WebSphere Message Broker Toolkit. If the Outline view is not visible, from the WebSphere Message Broker Toolkit menu, click **Window > Show View> Outline**.
3. In the Outline view, right-click the object (message, complex type, group, or complex element) to which you are adding a local element then click **Add Local Element** on the pop-up menu. A local element of type string is added to the message model and is assigned a default name.
4. Either type a new name for the local element or press **Enter** to accept the default.

Results

You can now configure the local element to your exact requirements. For further information about configuring message model objects see "Configuring MRM message model objects" on page 2896.

Related concepts:

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

"Message model objects: elements" on page 1176
An *element* is a named piece of information (a field) within a message, with a meaning that is agreed by the applications that create and process the message.

Related tasks:

"Configuring MRM message model objects" on page 2896
Various tasks are involved in configuring MRM message model objects

"Deleting objects" on page 2919
Delete an object from your message model.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Local element logical properties" on page 5442
The logical properties of a local element include properties that specify the number of occurrences and value of the local element.

"Local element CWF properties" on page 5469
The properties, and their permissible values, vary according to the object type.

"Local element XML properties" on page 5492
The properties, and their permissible values, vary according to the object type.

"Local element TDS properties" on page 5524
The properties, and their permissible values, vary according to the object type.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

Adding an element reference:

Add an element reference to a message, type, group, or complex element.

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 2842
- "Creating a message definition file" on page 2865
- "Opening an existing message definition file" on page 2864

This task assumes that you have previously added the relevant message, type, global group, or complex element to your message model.

About this task

To add an element reference to a message, type, global group, or complex element:

Procedure

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the WebSphere Message Broker Toolkit. If the Outline view is not visible, from the WebSphere Message Broker Toolkit menu, click **Window > Show View> Outline**.
3. In the Outline view, right-click the object (message, complex type, group, or complex element) to which you are adding the element reference, then click **Add Element Reference** on the pop-up menu. This adds a default element reference to the message model object that points to an existing global element. This existing global element might be in the current message definition file or in a message definition file defined under **Includes** or **Imports** for the current message definition file. For further information about Imports and Includes, see "Linking from one message definition file to another" on page 2921.

Results

You can now configure the element reference to your exact requirements. For further information about configuring message model objects see "Configuring MRM message model objects" on page 2896.

Related concepts:

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

"Message model objects: elements" on page 1176
An *element* is a named piece of information (a field) within a message, with a meaning that is agreed by the applications that create and process the message.

Related tasks:

"Configuring MRM message model objects" on page 2896
Various tasks are involved in configuring MRM message model objects

"Deleting objects" on page 2919
Delete an object from your message model.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Element reference logical properties" on page 5423
The logical properties of an element reference include properties that specify the number of occurrences of the element reference.

"Element reference CWF properties" on page 5460
The properties, and permissible values, vary according to the type of object.

"Element reference XML properties" on page 5481
The properties, and their permissible values, vary according to the type of object.

"Element reference TDS properties" on page 5509
The properties, and their permissible values, vary according to the object type.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

Adding a wildcard element:

Add a wildcard element to a message, type, group, or complex element in a message model.

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 2842
- "Creating a message definition file" on page 2865
- "Opening an existing message definition file" on page 2864

You can add a wildcard element to a message, type, group or complex element. This task assumes that you have previously added the relevant message, type, group or complex element to your message model.

About this task

To add a wildcard element to a message, type, group or complex element:

Procedure

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the WebSphere Message Broker Toolkit. If the Outline view is not visible, from the WebSphere Message Broker Toolkit menu, click **Window > Show View> Outline**.
3. In the Outline view, right-click the message model object (message, complex type, group or complex element) to which you are adding the wildcard element then click **Add Wildcard Element** on the pop-up menu. A wildcard element is added and appears in the Outline view.

Results

You can now configure the wildcard element to your exact requirements. For further information on configuring message model objects see "Configuring MRM message model objects" on page 2896.

Related concepts:

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

"Message model objects: elements" on page 1176
An *element* is a named piece of information (a field) within a message, with a meaning that is agreed by the applications that create and process the message.

Related tasks:

"Configuring MRM message model objects" on page 2896
Various tasks are involved in configuring MRM message model objects

"Deleting objects" on page 2919
Delete an object from your message model.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Wildcard element logical properties" on page 5453
The logical properties of a wildcard element include properties that specify the number of occurrences of the wildcard element.

"Wildcard element CWF properties" on page 5475
There are no properties to show.

"Wildcard element XML properties" on page 5500
There are no properties to show.

"Wildcard element TDS properties" on page 5534
There are no properties to show.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

Adding a global attribute:

Add a global attribute to your message model.

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 2842
- "Creating a message definition file" on page 2865
- "Opening an existing message definition file" on page 2864

About this task

To add a global attribute to your message model:

Procedure

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the WebSphere Message Broker Toolkit. If the Outline view is not visible, from the WebSphere Message Broker Toolkit menu, click **Window > Show View> Outline**.

3. In the Outline view, right-click **Elements and attributes** then click **Add Global Attribute** on the pop-up menu. A global attribute of type string is immediately added and is assigned a default name.
4. Either type a new name for the global attribute or press Enter to accept the default.

Results

You can now configure the global attribute to your requirements. For more information on configuring message model objects see "Configuring MRM message model objects" on page 2896.

Related concepts:

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

"Message model objects: attributes" on page 1185
An *attribute* describes an XML attribute. They are used only when the data is XML.

Related tasks:

"Configuring MRM message model objects" on page 2896
Various tasks are involved in configuring MRM message model objects

"Deleting objects" on page 2919
Delete an object from your message model.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Global attribute logical properties" on page 5425
The logical properties of a global attribute.

"Global attribute CWF properties" on page 5462
There are no properties to show.

"Global attribute XML properties" on page 5483
The properties, and their permissible values, vary according to the object type.

"Global attribute TDS properties" on page 5511
The properties, and their permissible values, vary according to the object type.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

Adding a local attribute:

Add a local attribute to a message, complex type, or complex element.

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 2842
- "Creating a message definition file" on page 2865
- "Opening an existing message definition file" on page 2864

You can add a local attribute to a message, complex type, or complex element. This task assumes that you have previously added the relevant message, complex type, or complex element to your message model.

About this task

To add a local attribute to a message, complex type or complex element:

Procedure

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the WebSphere Message Broker Toolkit. If the Outline view is not visible, from the WebSphere Message Broker Toolkit menu, click **Window > Show View> Outline**.
3. In the Outline view, right-click the object (message, complex type, complex element, or attribute group) to which you are adding the local attribute then click **Add Local Attribute** on the pop-up menu. A local attribute of type string is immediately added to the message model object and is assigned a default name.
4. Either type a new name for the local attribute or press **Enter** to accept the default.

Results

You can now configure the local attribute to your requirements. For further information about configuring message model objects see "Configuring MRM message model objects" on page 2896.

Related concepts:

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

"Message model objects: attributes" on page 1185
An *attribute* describes an XML attribute. They are used only when the data is XML.

Related tasks:

"Configuring MRM message model objects" on page 2896
Various tasks are involved in configuring MRM message model objects

"Deleting objects" on page 2919
Delete an object from your message model.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Local attribute logical properties" on page 5438
The logical properties of a local attribute.

"Local attribute CWF properties" on page 5467
The properties, and their permissible values, vary according to the object type.

"Local attribute XML properties" on page 5490
The properties, and their permissible values, vary according to the object type.

"Local attribute TDS properties" on page 5522
The properties, and their permissible values, vary according to the object type.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

Adding an attribute reference:

Add an attribute reference to a message, complex type, or complex element.

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 2842
- "Creating a message definition file" on page 2865
- "Opening an existing message definition file" on page 2864

You can add an attribute reference to a message, complex type, or complex element. This task assumes that you have previously added the relevant message, complex type, or complex element to your message model.

About this task

To add an attribute reference to a message, complex type or complex element:

Procedure

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the WebSphere Message Broker Toolkit. If the Outline view is not visible, from the WebSphere Message Broker Toolkit menu, click **Window > Show View> Outline**.
3. In the Outline view, right-click the message model object (message, complex type, complex element, or attribute group) to which you are adding the attribute reference then click **Add Attribute Reference** on the pop-up menu. This action adds a default attribute reference to the message model object that points to an existing global attribute.

Results

You can now configure the attribute reference to your exact requirements. For further information about configuring message model objects see "Configuring MRM message model objects" on page 2896.

Related concepts:

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

"Message model objects: attributes" on page 1185
An *attribute* describes an XML attribute. They are used only when the data is XML.

Related tasks:

"Configuring MRM message model objects" on page 2896
Various tasks are involved in configuring MRM message model objects

"Deleting objects" on page 2919
Delete an object from your message model.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Attribute reference logical properties" on page 5418
The logical properties of an attribute reference.

"Attribute reference CWF properties" on page 5457
The properties, and their permissible values, vary according to the object type.

"Attribute reference XML properties" on page 5478
The properties, and their permissible values, vary according to the object type.

"Attribute reference TDS properties" on page 5503
The properties, and their permissible values, vary according to the object type.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Attribute reference properties" on page 5541
Different types of properties are available for an attribute reference.

Adding a wildcard attribute:

Add a wildcard attribute to a message, complex type, or complex element.

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 2842
- "Creating a message definition file" on page 2865
- "Opening an existing message definition file" on page 2864

You can add a wildcard attribute to a message, complex type, or complex element. This task assumes that you have previously added the relevant message, complex type, or complex element to your message model.

About this task

To add a wildcard attribute to a message, complex type or complex element:

Procedure

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the WebSphere Message Broker Toolkit. If the Outline view is not visible, from the WebSphere Message Broker Toolkit menu, click **Window > Show View> Outline**.
3. In the Outline view, right-click the object (message, complex type, complex element, or attribute group) to which you are adding the wildcard attribute then click **Add Wildcard Attribute** on the pop-up menu. A wildcard attribute of type string is immediately added to the message model object and is assigned a default name.

4. Either type a new name for the wildcard attribute or press **Enter** to accept the default.

Results

You can now configure the wildcard attribute to your requirements. For further information about configuring message model objects see "Configuring MRM message model objects" on page 2896.

Related concepts:

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

"Message model objects: attributes" on page 1185
An *attribute* describes an XML attribute. They are used only when the data is XML.

Related tasks:

"Configuring MRM message model objects" on page 2896
Various tasks are involved in configuring MRM message model objects

"Deleting objects" on page 2919
Delete an object from your message model.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Wildcard attribute logical properties" on page 5453
The logical properties of a wildcard attribute.

"Wildcard attribute CWF properties" on page 5475
There are no properties to show.

"Wildcard attribute XML properties" on page 5499
There are no properties to show.

"Wildcard attribute TDS properties" on page 5534
There are no properties to show.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

Adding a simple type:

Add a simple type to your message model.

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 2842
- "Creating a message definition file" on page 2865
- "Opening an existing message definition file" on page 2864

About this task

To add a simple type to your message model:

Procedure

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the WebSphere Message Broker Toolkit. If the Outline view is not visible, from the WebSphere Message Broker Toolkit menu, click **Window > Show View> Outline**.
3. In the Outline view, right-click **Types** then click either **Add Simple Type Restriction**, **Add Simple Type List**, or **Add Simple Type Union** on the pop-up menu.
   - For a restriction, a simple type of base type string is added, and assigned a default name.
   - For a list, a simple type of item type string is added, and assigned a default name.
   - For a union, a simple type with a single member type of string is added, and assigned a default name.
4. Either type a new name for the simple type or press **Enter** to accept the default.

Results

You can now configure the simple type to your exact requirements.

If the simple type is a restriction:

- You can change the base type using the editor Properties view.
- You can set the value constraints associated with the simple type. See "Setting value constraints" on page 2907.
- You can replace the base type with a new local simple type. In the Outline view right-click simple type and click one of:
  - **Add Simple Type Restriction**. This option replaces the base type with a new simple type restriction, with a base type of string. You can configure the restriction as described in 'If the simple type is a restriction'. The result is that the original simple type becomes a restriction of a restriction.
  - **Add Simple Type List**. This option replaces the base type with a new simple type list, with an item type of string. You can configure the list as described in 'If the simple type is a list'. The result is that the original simple type becomes a restriction of a list. It appears as a list in the editor, because a restriction of a list is itself a list, but you can also set certain value constraints.

If the simple type is a list:

- You can change the item type using the editor Properties view.
- You can replace the item type with a new local simple type. In the Outline view right-click the simple type and click one of:
  - **Add Simple Type Restriction**. This option replaces the item type with a new simple type restriction, with a base type of string. You can configure the restriction as described in 'If the simple type is a restriction'. The result is that the original simple type becomes a list of a restriction.
  - **Add Simple Type Union**. This option replaces the item type with a new simple type union, with a single member type of string. You can configure the union as described in 'If the simple type is a union'. The result is that the original simple type becomes a list of a union.

If the simple type is a union:

- If the member type of string is not required, in the Outline view right-click the string and click Delete.
- You can add further members to the union. In the Outline view right-click the simple type and click one of:
  - **Add Union Member Type**. This option adds a union member that is an existing simple type. Use the type selection dialog to select the simple type required.
  - **Add Local Member Type Restriction**. This option adds a union member that is a new simple type restriction, with a base type of string. You can configure the restriction as described in 'If the simple type is a restriction' referred to earlier.
  - **Add Local Member Type List**. This option adds a union member that is a new simple type list, with an item type of string. You can configure the list as described in 'If the simple type is a list' referred to earlier.
  - **Add Local Member Type Union**. This option adds a union member that is a new simple type union, with a single member type of string. You can configure the new union as described in 'If the simple type is a union'.
- New members are added to the end of the union. To change the order of a member, in the Outline view select the member and drag it to the required position in the union. All union members that are existing simple types must occur ahead of all members that are local restrictions, lists, and unions, so reordering is subject to this rule.

For further information about configuring message model objects see “Configuring MRM message model objects” on page 2896.

Related concepts:

“Message model objects” on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

“Message model objects: types” on page 1177
Types describe the data content of elements.

“Message model objects: simple types” on page 1180
A *simple type* is an abstract definition of an item of data such as a number, a string, or a date.

Related tasks:

“Configuring MRM message model objects” on page 2896
Various tasks are involved in configuring MRM message model objects

“Deleting objects” on page 2919
Delete an object from your message model.

Related reference:

“Message model object properties” on page 5416
Access property information by property kind, or by object.

“Simple type logical properties” on page 5450
The logical properties of a simple type.

“Simple type CWF properties” on page 5474
There are no properties to show.

“Simple type XML properties” on page 5498
There are no properties to show.

“Simple type TDS properties” on page 5532
There are no properties to show.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

Adding a complex type:

Add a complex type to your message model.

Before you begin

Before you start:

You must already have completed the following tasks:

- "Creating a message set" on page 2842
- "Creating a message definition file" on page 2865
- "Opening an existing message definition file" on page 2864

About this task

To add a complex type to your message model:

Procedure

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the WebSphere Message Broker Toolkit. If the Outline view is not visible, from the WebSphere Message Broker Toolkit menu, click **Window > Show View> Outline**.
3. In the Outline view, right-click **Types** then click **Add Complex Type** on the pop-up menu. A complex type is added and is assigned a default name.
4. Either type a new name for the complex type or press Enter to accept the default.

Results

You can now configure the complex type to your requirements. For further information on configuring message model objects see "Configuring MRM message model objects" on page 2896.

Related concepts:

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

"Message model objects: types" on page 1177
Types describe the data content of elements.

"Message model objects: complex types" on page 1178
A *complex type* describes the structure of one or more complex elements.

Related tasks:

"Configuring MRM message model objects" on page 2896
Various tasks are involved in configuring MRM message model objects

"Deleting objects" on page 2919
Delete an object from your message model.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Complex type logical properties" on page 5419
The logical properties of a complex type include properties that describe content and substitution settings.

"Complex type CWF properties" on page 5459
There are no properties to show.

"Complex type XML properties" on page 5480
There are no properties to show.

"Complex type TDS properties" on page 5505
The TDS properties of a complex type.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

Adding a global group:

Add a global group to your message model.

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 2842
- "Creating a message definition file" on page 2865
- "Opening an existing message definition file" on page 2864

About this task

To add a global group to your message model:

Procedure

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the WebSphere Message Broker Toolkit. If the Outline view is not visible, from the WebSphere Message Broker Toolkit menu, click **Window > Show View> Outline**.
3. In the Outline view, right-click **Groups** then click **Add Group** on the pop-up menu. A global group is added to your message model and is assigned a default name.
4. Either type a new name for the global group or press **Enter** to accept the default.

Results

You can now configure the global group to your requirements. For further information about configuring the properties of message model objects see "Configuring MRM message model objects" on page 2896.

Related concepts:

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere

Message Broker extension to XML Schema.

"Message model objects: groups" on page 1183
A *group* is a list of elements that defines how those elements appear in a message. Groups define the composition and content validation of a complex type.

Related tasks:

"Configuring MRM message model objects" on page 2896
Various tasks are involved in configuring MRM message model objects

"Deleting objects" on page 2919
Delete an object from your message model.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Global group logical properties" on page 5433
The logical properties of a global group.

"Global group CWF properties" on page 5464
There are no properties to show.

"Global group XML properties" on page 5488
There are no properties to show.

"Global group TDS properties" on page 5516
The TDS properties of a global group.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

Adding a local group:

Add a local group to a message, type, global group, or complex element.

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 2842
- "Creating a message definition file" on page 2865
- "Opening an existing message definition file" on page 2864

You can add a local group to a message, complex type, group, or complex element. This task assumes that you have previously added the relevant message, complex type, group, or complex element to your message model.

About this task

To add a local group to a message, complex type, group, or complex element:

Procedure

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the WebSphere Message Broker Toolkit. If the Outline view is not visible, from the WebSphere Message Broker Toolkit menu, click **Window > Show View> Outline**.

3. In the Outline view, right-click the object (message, complex type, group, or complex element) to which you are adding the local group then click **Add Local Group** on the pop-up menu. A local group is immediately added to the message model with type composition set to *sequence* by default.

Results

You can now configure the local group to your requirements. For further information about configuring message model objects, see "Configuring MRM message model objects" on page 2896.

Related concepts:

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

"Message model objects: groups" on page 1183
A *group* is a list of elements that defines how those elements appear in a message. Groups define the composition and content validation of a complex type.

Related tasks:

"Configuring MRM message model objects" on page 2896
Various tasks are involved in configuring MRM message model objects

"Deleting objects" on page 2919
Delete an object from your message model.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Local group logical properties" on page 5446
The logical properties of a local group include properties that specify the number of occurrences of the local group.

"Local group CWF properties" on page 5471
The CWF properties of a local group are described in the following tables.

"Local group XML properties" on page 5494
There are no properties to show.

"Local group TDS properties" on page 5526
TDS properties of a local group.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

Adding an attribute group:

Add an attribute group to your message model.

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 2842
- "Creating a message definition file" on page 2865
- "Opening an existing message definition file" on page 2864

About this task

To add an attribute group to your message model:

Procedure

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the WebSphere Message Broker Toolkit. If the Outline view is not visible, from the WebSphere Message Broker Toolkit menu, click **Window > Show View> Outline**.
3. In the Outline view, right-click **Groups** then click **Add Attribute Group** on the pop-up menu. A global attribute group is added to your message model and is assigned a default name.
4. Either type a new name for the attribute group or press **Enter** to accept the default.

Results

You can now configure the attribute group to your requirements. For further information about configuring the properties of message model objects see "Configuring MRM message model objects" on page 2896.

Related concepts:

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

"Message model objects: groups" on page 1183
A *group* is a list of elements that defines how those elements appear in a message. Groups define the composition and content validation of a complex type.

Related tasks:

"Configuring MRM message model objects" on page 2896
Various tasks are involved in configuring MRM message model objects

"Deleting objects" on page 2919
Delete an object from your message model.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Global attribute group logical properties" on page 5429
The logical properties of an attribute group.

"Global attribute group CWF properties" on page 5463
There are no properties to show.

"Global attribute group XML properties" on page 5485
There are no properties to show.

"Global attribute group TDS properties" on page 5513
The TDS properties of a global attribute group.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Global attribute group properties" on page 5741
Different types of properties are available for a global attribute group.

Adding a group reference:

You can add a group reference to a message, type, global group, or complex element.

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 2842
- "Creating a message definition file" on page 2865
- "Opening an existing message definition file" on page 2864

You can add a group reference to a message, complex type, group, or complex element. This task assumes that you have previously added the relevant message, complex type, group, or complex element to your message model.

About this task

To add a group reference to a message, complex type, group or complex element:

Procedure

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the WebSphere Message Broker Toolkit. If the Outline view is not visible, from the WebSphere Message Broker Toolkit menu, click **Window > Show View> Outline**.
3. In the Outline view, right-click the object (message, complex type, group, or complex element) to which you want to add a group reference then click **Add Group Reference** on the pop-up menu. A group reference is immediately added to your message model.

Results

You can now configure the group reference to your requirements. For further information on configuring the properties of message model objects see "Configuring MRM message model objects" on page 2896.

Related concepts:

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

"Message model objects: groups" on page 1183
A *group* is a list of elements that defines how those elements appear in a message. Groups define the composition and content validation of a complex type.

Related tasks:

"Configuring MRM message model objects" on page 2896
Various tasks are involved in configuring MRM message model objects

"Deleting objects" on page 2919
Delete an object from your message model.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Group reference CWF properties" on page 5465
The CWF properties of a group reference.

"Group reference XML properties" on page 5488
The XML properties of a group reference.

"Group reference TDS properties" on page 5520
The following tables describe the TDS properties of a group reference.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

Adding an attribute group:

Add an attribute group reference to a message model.

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 2842
- "Creating a message definition file" on page 2865
- "Opening an existing message definition file" on page 2864

About this task

To add an attribute group to your message model:

Procedure

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the WebSphere Message Broker Toolkit. If the Outline view is not visible, from the WebSphere Message Broker Toolkit menu, click **Window > Show View> Outline**.
3. In the Outline view, right-click **Groups** then click **Add Attribute Group** on the pop-up menu. An attribute group is added to your message model and is assigned a default name.
4. Either type a new name for the attribute group reference or press Enter to accept the default.

Results

You can now configure the attribute group to your requirements using the Message Definition editor. For further information on configuring the properties of message model objects see "Configuring MRM message model objects" on page 2896.

Related concepts:

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

"Message model objects: groups" on page 1183
A *group* is a list of elements that defines how those elements appear in a message.

Groups define the composition and content validation of a complex type.

Related tasks:

"Configuring MRM message model objects"
Various tasks are involved in configuring MRM message model objects

"Deleting objects" on page 2919
Delete an object from your message model.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Attribute group reference logical properties" on page 5417
The logical properties of an attribute group reference.

"Attribute group reference CWF properties" on page 5456
There are no properties to show.

"Attribute group reference XML properties" on page 5477
The XML properties of an attribute group reference.

"Attribute group reference TDS properties" on page 5502
There are no properties to show.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Attribute group reference properties" on page 5537
Different types of properties are available for an attribute group reference.

Configuring MRM message model objects

Various tasks are involved in configuring MRM message model objects

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 2842
- "Creating a message definition file" on page 2865
- "Adding MRM message model objects" on page 2870 (You must have added one or more objects to your message model)

Before starting any of the tasks in this topic area, you must first open the message definition file for which you want to configure message model objects in the Message Definition editor. See "Opening an existing message definition file" on page 2864 for further details.

About this task

This topic area describes the tasks that are involved in configuring message model objects:

- "Analyzing planned changes to message model objects" on page 2897
- "Renaming objects" on page 2898
- "Reordering objects" on page 2900
- "Copying objects" on page 2901
- "Pasting objects" on page 2901
- "Changing the type of an element or attribute" on page 2903

- "Setting value constraints" on page 2907
- "Configuring logical properties: Message model objects" on page 2909
- "Configuring documentation properties: Message model objects" on page 2910
- "Configuring physical properties" on page 2912
  - "Configuring Custom Wire Format (CWF) properties: Message model objects" on page 2913
  - "Configuring XML Wire Format properties: Message model objects" on page 2916
  - "Configuring TDS properties: Message model objects" on page 2914
  - "Applying default physical format settings: Message model objects" on page 2917
- "Deleting objects" on page 2919

Related concepts:

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

Analyzing planned changes to message model objects:

Use impact analysis to analyze the effect of renaming message model objects.

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 2842
- "Creating a message definition file" on page 2865
- "Opening an existing message definition file" on page 2864
- "Adding MRM message model objects" on page 2870 (You must have added one or more objects to your message model)
- "Enabling and disabling indexing" on page 1454

About this task

This information covers the following task:

- "Renaming message model objects"

Renaming message model objects:
About this task

You can analyze the effect of renaming the following objects:

- Complex and simple types
- Model Groups
- Elements
- Attributes

- Attribute Groups

Procedure

1. In the Broker Development view, right-click the object that you want to rename, then click **Impact Analysis** > **Rename**.
2. In the Impact Analysis - Rename Artifact window, type the new name of the object, then click **Analyze Impact**.

   The Rename Artifact dialog box shows the results of impact analysis, listing primary and secondary impacts.

   You can view the results of impact analysis in the Impact Analysis dialog box, or the Impact Analysis view of the WebSphere Message Broker Toolkit. For more information about how to view information about selected resources, mark changes as complete, copy results to an external application, and view previous results, see "Impact Analysis view" on page 6801.

:

Related concepts:

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

"Impact analysis: analyzing the effects of planned changes to your applications" on page 1150
When developing an application, you might want to change the names of some artifacts, or move them. Impact analysis reports the artifacts that are likely to be affected by a particular change.

Related tasks:

"Enabling and disabling indexing" on page 1454
Enable indexing to support impact analysis.

"Analyzing planned changes to ESQL objects" on page 2403
Use impact analysis to analyze the effect of renaming or moving ESQL objects, or moving or changing the broker schema of an ESQL file.

"Analyzing the impact of changes to message maps" on page 2234
Use impact analysis to analyze the effect of renaming or moving message maps.

"Analyzing planned changes to message flows" on page 1436
Use impact analysis to analyze the effect of renaming or moving message flows, including subflows.

"Analyzing planned changes to message set resources" on page 1165
Use impact analysis to analyze the effect of renaming message definition or deployable WSDL files.

Related reference:

"Impact analysis: reference" on page 4174
Some artifacts are excluded from secondary analysis.

"Impact Analysis view" on page 6801
The Impact Analysis view shows information about selected resources, which changes have been marked as complete, and previous results. You can also use this view to copy results to the clipboard.

Renaming objects:

You can rename message model objects in the WebSphere Message Broker Toolkit.

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 2842
- "Creating a message definition file" on page 2865
- "Opening an existing message definition file" on page 2864
- "Adding MRM message model objects" on page 2870 (You must have added one or more objects to your message model)

Also read "Analyzing planned changes to message model objects" on page 2897 for information on the effects of the intended changes.

About this task

Objects in the WebSphere Message Broker Toolkit such as files, messages and elements can have different physical representations. Eclipse handles renaming differently depending on the object.

Tip: Not all objects can be renamed. For example, you cannot rename wildcards, local groups, or local types, because they do not have a name.

If an object can be renamed the usual way to do it is as follows:

Procedure

1. Switch to the Broker Application Development perspective.
2. In the Outline view, right-click the object that you want to rename then click **Rename** on the pop-up menu. Alternatively, right-click the object in the Message Definition editor **Overview** tab then click **Rename** on the pop-up menu. In both cases, depending on the object, either a renaming dialog opens or you will now be able to edit the name of the object directly.
3. Type the new name for the object.
4. If the renaming dialog is open, either press Enter or click **OK**.

Results

Related concepts:

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

Related tasks:

"Deleting objects" on page 2919
Delete an object from your message model.

"Renaming a physical format" on page 2856
Rename a physical format using the Message Set editor.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

Reordering objects:

Reorder message model objects within a message definition file.

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 2842
- "Creating a message definition file" on page 2865
- "Opening an existing message definition file" on page 2864
- "Adding MRM message model objects" on page 2870 (You must have added one or more objects to your message model)

About this task

To reorder objects within a message definition file:

Procedure

1. Switch to the Broker Application Development perspective.
2. Click the object that you want to move. For example, you could select a local element within a message in either the Outline view or Properties Hierarchy.
3. Use the mouse to drag the object to its new location.

   Tip: As you drag the object and the mouse cursor passes between objects, a black line appears, showing where the current focus is. If you try to drag the object to a location that it cannot be moved to (including objects that are highlighted as the cursor passes over them), the object remains in its original position when you release it.

Results

Related concepts:

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

Related tasks:

"Copying objects" on page 2901
You can copy an object in a message definition file, such as a message for a local element object, or types for a complex type object.

"Deleting objects" on page 2919
Delete an object from your message model.

"Pasting objects" on page 2901
Paste objects that you have previously copied within the same message definition file

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

Copying objects:

You can copy an object in a message definition file, such as a message for a local element object, or types for a complex type object.

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 2842
- "Creating a message definition file" on page 2865
- "Opening an existing message definition file" on page 2864
- "Adding MRM message model objects" on page 2870 (You must have added one or more objects to your message model)

About this task

To copy an object:

Procedure

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective. If the Outline view is not visible, from the WebSphere Message Broker Toolkit menu, click **Window** > **Show View** > **Outline**.
3. In the Outline view, right-click the message model object that you want to copy, then click **Copy**. Alternatively, right-click the object in the Message Definition editor **Overview** tab, then click **Copy**.

Results

The object is now copied.

Related concepts:

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

Related tasks:

"Pasting objects"
Paste objects that you have previously copied within the same message definition file

"Reordering objects" on page 2900
Reorder message model objects within a message definition file.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

Pasting objects:

Paste objects that you have previously copied within the same message definition file

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 2842
- "Creating a message definition file" on page 2865
- "Opening an existing message definition file" on page 2864
- "Adding MRM message model objects" on page 2870 (You must have added one or more objects to your message model)
- "Copying objects" on page 2901

About this task

You can paste objects that you have previously copied within the same message definition file.

You can only copy and paste an object within the same message definition. You cannot copy an object and paste it into another message definition, either within the same message set or in a different message set.

To paste an object in the message definition from which you copied it:

Procedure

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the WebSphere Message Broker Toolkit. If the Outline view is not visible, from the WebSphere Message Broker Toolkit menu, click **Window > Show View> Outline**.
3. In the Outline view, right-click the location where you are going to paste the object then click **Paste** on the pop-up menu. Alternatively, right-click the object in the Message Definition editor **Overview** tab then click **Paste** on the pop-up menu. The object appears in the new location with a default name which you can change if you want to.
4. Either type a new name for the object or press **Enter** to accept the default.

What to do next

Note: When you copy and paste objects within the message set, where physical properties exist for that object, these settings are not pasted, but are set to default values.

Tip: If you cannot select **Paste** from either menu, you might be trying to paste to a location that is not valid. For example, you cannot paste a complex type into a local element.

Related concepts:

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

Related tasks:

"Copying objects" on page 2901
You can copy an object in a message definition file, such as a message for a local

element object, or types for a complex type object.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

Changing the type of an element or attribute:

You can change the type to a local element, global element, local attribute, or global attribute.

Before you begin

Before you start:

You must already completed the following tasks:

- "Creating a message set" on page 2842
- "Creating a message definition file" on page 2865
- "Opening an existing message definition file" on page 2864
- "Adding MRM message model objects" on page 2870 (You must have added one or more objects to your message model)

About this task

You can change the type of an element or attribute in your message model to another existing type, or you can create a new simple type or a new complex type.

To change the type of an element or attribute to an existing type:

Procedure

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the WebSphere Message Broker Toolkit. If the Outline view is not visible, from the WebSphere Message Broker Toolkit menu, click **Window** > **Show View** > **Outline**.
3. In the Outline view, click the element for which you want to change the type.
4. Display the **Properties** tab of the Message Definition editor by clicking **Properties** in the lower-left corner of the editor area.
5. In the Properties Hierarchy click **Logical Properties** > **Global Element** (or **Logical Properties** > **Local Element**, **Logical Properties** > **Global Attribute**, or **Logical Properties** > **Local Attribute**). If necessary, expand **Logical Properties** by clicking **+**.
6. In the Details view, in the **Type** property, click the new type that you require.

   Tip: If the type you require is not displayed, you can find it by clicking **(More...)** in the list. This displays the Type Selection window with additional options. If you know which type you require, specify the first letter in the text box at the top of the Type Selection window. Matching types are then displayed, making the selection process easier.
7. When you have selected the type that you require, click **OK**.

Results

The change to the type is applied wherever the element or attribute occurs.

The task above explains how to switch to an existing type. If you want to create a new simple type or a new complex type, select **(New Simple Type Restriction)**, **(New Simple Type List)**, **(New Simple Type Union)**, or **(New Complex Type)** in the **Type** list (see step 6 above). For information on how to create a new simple type or a new complex type see "Adding a simple type to an element or attribute" or "Adding a complex type to an element" on page 2905.

Related concepts:

"Message model objects: elements" on page 1176
An *element* is a named piece of information (a field) within a message, with a meaning that is agreed by the applications that create and process the message.

Related tasks:

"Adding a simple type to an element or attribute"
You can add a simple type to a local element, global element, local attribute, or global attribute.

"Adding a complex type to an element" on page 2905
You can add a complex type to a local element, global element, local attribute, or global attribute.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

Adding a simple type to an element or attribute:

You can add a simple type to a local element, global element, local attribute, or global attribute.

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 2842
- "Creating a message definition file" on page 2865
- "Opening an existing message definition file" on page 2864

This task assumes that you have previously added the relevant element or attribute to your message model.

About this task

To add a new simple type:

Procedure

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the WebSphere Message Broker Toolkit. If the Outline view is not visible, from the WebSphere Message Broker Toolkit menu, click **Window > Show View> Outline**.
3. In the Outline view, click the element to which you want to add a new simple type.
4. In the Message Definition editor, click the **Properties** tab.

5. In the Properties Hierarchy, click **Logical Properties > Global Element** (or **Logical Properties > Local ElementLogical Properties > Global Attribute**, or **Logical Properties > Local Attribute**).
6. In the Details view, in the **Type** property, select **(New Simple Type Restriction)**, **(New Simple Type List)**, or **(New Simple Type Union)** to open the relevant New Simple Type window to create a simple type of the type that you specify.
7. In the New Simple Type window, in the **Base Type** list, click the type that you want to use.
8. Optional: If you want to add the new simple type as a global simple type, select the **Create as Global Simple Type** check box and specify the name for your new simple type in the **Name** field.
9. Click **OK**. A simple type is immediately added to your message model.

Results

Any changes that you make are reflected throughout where the element to which you have added a new simple type occurs.

Related concepts:

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

"Message model objects: types" on page 1177
Types describe the data content of elements.

"Message model objects: simple types" on page 1180
A *simple type* is an abstract definition of an item of data such as a number, a string, or a date.

Related tasks:

"Configuring MRM message model objects" on page 2896
Various tasks are involved in configuring MRM message model objects

"Deleting objects" on page 2919
Delete an object from your message model.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Simple type logical properties" on page 5450
The logical properties of a simple type.

"Simple type CWF properties" on page 5474
There are no properties to show.

"Simple type XML properties" on page 5498
There are no properties to show.

"Simple type TDS properties" on page 5532
There are no properties to show.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

Adding a complex type to an element:

You can add a complex type to a local element, global element, local attribute, or global attribute.

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 2842
- "Creating a message definition file" on page 2865
- "Opening an existing message definition file" on page 2864

This task assumes that you have previously added the relevant element or attribute to your message model.

About this task

When you add a complex type to an element or attribute, you can either create a new complex type or derive a new complex type from an existing base type.

To add a new complex type:

Procedure

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the WebSphere Message Broker Toolkit. If the Outline view is not visible, from the WebSphere Message Broker Toolkit menu, click **Window > Show View> Outline**.
3. In the Outline view, click the element to which you want to add a new complex type.
4. In the Message Definition editor, click the **Properties** tab.
5. In the Properties Hierarchy, click **Logical Properties > Global Element** (or **Logical Properties > Local ElementLogical Properties > Global Attribute**, or **Logical Properties > Local Attribute**).
6. In the Details view, in the **Type** list, click **(New Complex Type)** to display the New Complex Type window.
7. If you want to create a new local complex type, click **Create a Local Complex Type**, in the **Composition** list, click the option that you require.
8. If you want to derive a new local complex type from an existing base type:
   a. Click **Derive a new Local Complex Type from existing base type**.
   b. In the **Base Type** list, click the base type that you want to use. Depending on the base type you select, the **Composition** and **Derived By** lists might become active.
   c. If the **Composition** and **Derived By** lists are active, click the options that you require in both lists.
9. If you want to add the new complex type as a global complex type, select the **Create as Global Complex Type** check box, and specify a name for your new complex type in the **Name** field.
10. Click **OK** to close the New Complex Type window and add the new complex type to your message model.

Results

Any changes that you make are reflected throughout where the element to which you are adding the complex type occurs.

Related concepts:

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

"Message model objects: types" on page 1177
Types describe the data content of elements.

"Message model objects: complex types" on page 1178
A *complex type* describes the structure of one or more complex elements.

Related tasks:

"Configuring MRM message model objects" on page 2896
Various tasks are involved in configuring MRM message model objects

"Deleting objects" on page 2919
Delete an object from your message model.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Complex type logical properties" on page 5419
The logical properties of a complex type include properties that describe content and substitution settings.

"Complex type CWF properties" on page 5459
There are no properties to show.

"Complex type XML properties" on page 5480
There are no properties to show.

"Complex type TDS properties" on page 5505
The TDS properties of a complex type.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

Setting value constraints:

You can set the value constraints associated with a simple type.

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 2842
- "Creating a message definition file" on page 2865
- "Opening an existing message definition file" on page 2864
- "Adding a simple type" on page 2886 or "Adding a simple type to an element or attribute" on page 2904 (You must have added one or more global or local simple types to your message model)

About this task

Value constraints are usually associated with a simple type; they refine a simple type by defining limits on the values which the simple type can represent. To set the value constraints associated with a simple type:

Procedure

1. Switch to the Broker Application Development perspective.
2. In the Outline view, click the simple type you are updating. If the Outline view is not visible, from the WebSphere Message Broker Toolkit menu, click **Window** > **Show View** > **Outline**.
3. Display the **Properties** tab of the Message Definition Editor by clicking **Properties** in the lower-left corner of the editor area. The Properties Hierarchy displays the following nodes:
   - Logical Properties
   - Physical Properties
   - Documentation
4. In the Properties Hierarchy under **Logical Properties** click **Value Constraints**. This displays the current value constraints settings for the selected simple type in the Details pane.

   Tip: If **Value Constraints** is not in view, expand **Logical Properties** by clicking **+**.
5. Set the value constraints for the selected simple type by making the appropriate changes to the information shown in the Details pane.

Results

Setting an enumeration:
About this task

An enumeration restricts which values can be set for the value constraint. For example, "ABC" and "123". Use this section to create a list of fixed values that the associated type must match.

To set an enumeration:

Procedure

1. Click **Add** to the right of the **Enumerations** field. This adds an enumeration that has a default enumeration (for example *enumeration1*).
2. Type the data that you want to set for this value constraint.
3. Press Enter on your keyboard.
4. Repeat the above steps for each enumeration that you are adding.

Setting a pattern:
About this task

Set a pattern to indicate that the value constraint defines a string used as a regular expression that must be matched by the data in the associated type. The regular expression syntax supported is XML Schema regular expressions.

See “Regular expression syntax” on page 6304 for a list of supported regular expression syntax elements.

To set a pattern:

Procedure

1. Select **Add** to the right of the **Patterns** field. This adds a pattern that has a default pattern (for example *pattern1*) and is in update mode.

2. Type the data that you want to set for this value constraint.
3. Press Enter on your keyboard.
4. Repeat the above steps for each pattern that you are adding.

Related concepts:

"Message model objects: simple type value constraints" on page 1188
Value constraints, also known as *facets* in XML Schema, refine a simple type by defining limits on the values that it can represent.

Related tasks:

"Deleting objects" on page 2919
Delete an object from your message model.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Simple type logical value constraints" on page 5450
The properties, and their permissible values, vary according to the object type.

Configuring logical properties: Message model objects:

You can configure the logical properties of an object that has previously been added to the message model.

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 2842
- "Creating a message definition file" on page 2865
- "Opening an existing message definition file" on page 2864
- "Adding MRM message model objects" on page 2870 (You must have added one or more objects to your message model)

About this task

To configure the logical properties of an object that is part of the message model:

Procedure

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the WebSphere Message Broker Toolkit and is displaying the following information:
   - The name of the message definition file
   - Messages
   - Types
   - Groups
   - Elements and Attributes

   If the Outline view is not visible, from the WebSphere Message Broker Toolkit menu, click **Window** > **Show View** > **Outline**. If the information listed above is not displayed, ensure that the message definition file is open in the Message Definition editor. Message definition files have an `.mxsd` file extension.

3. In the Outline view, select the message model object for which you want to configure the logical properties:
   a. Depending on the type of the object that you are selecting, expand **Messages**, **Types**, **Groups** or **Elements and Attributes** as appropriate by clicking **+**.
   b. Click the object that you want to select within the expanded node.
4. Display the **Properties** tab of the Message Definition editor by clicking **Properties** in the lower-left corner of the editor area. The Properties Hierarchy displays the following nodes:
   - Logical Properties
   - Physical Properties
   - Documentation

   The type (for example, Local Element or Global Element) of the message model object that you selected in the Outline view is displayed under each of these nodes.

   If the items under **Logical Properties** are not in view, expand **Logical Properties** by clicking **+**.
5. Display the logical properties of the selected object in the Details view of the Message Definition editor, by clicking the appropriate item under **Logical Properties**.
6. Configure the logical properties of the selected item to your requirements by making the appropriate changes to the information shown in the Details view.
7. Save your changes by clicking **File** > **Save message_definition_file.mxsd** or by pressing Ctrl+S. Alternatively click **FileSave All** or press Ctrl+Shift+S.

Results

Related concepts:

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

Related tasks:

"Deleting objects" on page 2919
Delete an object from your message model.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

Configuring documentation properties: Message model objects:

You can configure the documentation properties of an object that has previously been added to the message model.

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 2842
- "Creating a message definition file" on page 2865
- "Opening an existing message definition file" on page 2864

- "Adding MRM message model objects" on page 2870 (You must have added one or more objects to your message model)

About this task

To configure the documentation properties of an object contained within a message definition file:

Procedure

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the WebSphere Message Broker Toolkit and is displaying the following information:
   - The name of the message definition file
   - Messages
   - Types
   - Groups
   - Elements and Attributes

   If the Outline view is not visible, from the WebSphere Message Broker Toolkit menu, click **Window** > **Show View** > **Outline**. If the information listed above is not displayed, ensure that the message definition file is open in the Message Definition editor. Message definition files have an `.mxsd` file extension.
3. In the Outline view, select the message model object for which you want to configure the documentation properties by taking the following steps:
   a. Depending on the type of the object that you are selecting, expand **Messages**, **Types**, **Groups** or **Elements and Attributes** as appropriate by clicking **+**.
   b. Click the object you want to select within the expanded node.
4. Display the **Properties** tab of the Message Definition editor by clicking **Properties** in the lower-left corner of the editor area. The Properties Hierarchy displays the following nodes:
   - Logical Properties
   - Physical Properties
   - Documentation

   The type (for example, Local Element or Global Element) of the message model object that you selected in the Outline view is displayed under each of these nodes.

   Tip: If the items under **Documentation** are not in view, expand **Documentation** by clicking **+**.
5. Display the logical properties of the selected object in the Details view by clicking the appropriate item under **Documentation**.
6. Configure the documentation properties of the selected item to your requirements by typing text into the **Documentation** field. Right-clicking in the field allows you to select options for undoing changes you have made, cutting or copying text from the field, pasting text into the field, deleting highlighted text or selecting all text in the field.
7. Save your changes by clicking **File** > **Save message_definition_file.mxsd** from the menu or pressing Ctrl+S. Alternatively, from the menu, click **File** > **Save All** or press Ctrl+Shift+S.

Results

Related concepts:

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

Configuring physical properties:

Working with the physical properties of message model objects.

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 2842
- "Creating a message definition file" on page 2865
- "Opening an existing message definition file" on page 2864
- "Adding MRM message model objects" on page 2870 (You must have added one or more objects to your message model)

The tasks in this topic area assume that you have added one or more physical formats to a message set. For further information see "Adding a Custom Wire Format (CWF)" on page 2848 or "Adding an XML wire format" on page 2853 or "Adding a TDS physical format" on page 2851.

About this task

When you have added objects to your message model it is likely that you will want to configure the physical properties of these objects. The following tasks relate to configuring the physical properties of message model objects:

- "Configuring Custom Wire Format (CWF) properties: Message model objects" on page 2913
- "Configuring XML Wire Format properties: Message model objects" on page 2916
- "Configuring TDS properties: Message model objects" on page 2914
- "Applying default physical format settings: Message model objects" on page 2917

Related concepts:

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

Configuring Custom Wire Format (CWF) properties: Message model objects:

You can configure the Custom Wire Format (CWF) properties of a message model object by using the Message Definition editor

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 2842
- "Creating a message definition file" on page 2865
- "Opening an existing message definition file" on page 2864
- "Adding a Custom Wire Format (CWF)" on page 2848
- "Adding MRM message model objects" on page 2870 (You must have added one or more objects to your message model)

About this task

To configure the *CWF* properties of a message model object:

Procedure

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the WebSphere Message Broker Toolkit and is displaying the following information:
   - The name of the message definition file
   - Messages
   - Types
   - Groups
   - Elements and Attributes

   If the Outline view is not visible, from the WebSphere Message Broker Toolkit menu, click **Window** > **Show View** > **Outline**. If the above hierarchy is not displayed, ensure that the message definition file is open in the Message Definition editor. Message definition files have an `.mxsd` file extension.
3. In the Outline view, select the object for which you want to configure the CWF properties by doing the following.
   a. Depending on the type of the object that you are selecting, expand **Messages**, **Types**, **Groups** or **Elements and Attributes** by clicking **+**.
   b. Click the object you that want to select within the expanded node.
4. Display the **Properties** tab of the Message Definition editor by clicking **Properties** in the lower-left corner of the editor area. In the Message Definition editor, in the Properties Hierarchy, the name of each of the physical formats that have been added to the message set appears under **Physical Properties**. The object type (for example, Local Element or Global Element) of the message model object that you selected in the Outline view is displayed under each physical format shown.

Tip: If the physical formats are not in view in the Properties Hierarchy, expand **Physical Properties** by clicking **+**. By default the CWF physical format is called Binary1 but could have a user defined name instead.

5. Under **Physical Properties**, click the object type for the message model object that you have chosen to configure under the CWF physical format. The CWF properties of your selected message model object appear in the Details view.
6. Configure the CWF properties of the selected object to your requirements by making the appropriate changes to the information shown in the Details view.

   Note: It is not possible to configure disabled fields.
7. Save your changes by clicking **File** > **Save message_definition_file.mxsd** or pressing Ctrl+S. Alternatively click **FileSave All** or press Ctrl+Shift+S.

Results

Related concepts:

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

Related tasks:

"Configuring Custom Wire Format (CWF) properties: Message sets" on page 2849
Configure the Custom Wire Format (CWF) properties of a message set using the Message Set editor.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

Configuring TDS properties: Message model objects:

You can configure the Tagged/Delimited String (TDS) properties of a message model object.

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 2842
- "Creating a message definition file" on page 2865
- "Opening an existing message definition file" on page 2864
- "Adding a TDS physical format" on page 2851
- "Adding MRM message model objects" on page 2870 (Adding one or more objects to your message model)

About this task

To configure the *TDS* properties of a message model object:

Procedure

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the WebSphere Message Broker Toolkit and is displaying the following information:
   - The name of the message definition file
   - Messages
   - Types
   - Groups
   - Elements and Attributes

   If the Outline view is not visible, from the WebSphere Message Broker Toolkit menu, click **Window** > **Show View** > **Outline**. If the above hierarchy is not displayed, ensure that the message definition file is open in the Message Definition editor. Message definition files have an `.mxsd` file extension.
3. In the Outline view, select the object for which you want to configure the TDS properties by taking the following steps:
   a. Depending on the type of the object that you are selecting, expand **Messages**, **Types**, **Groups** or **Elements and Attributes** by clicking **+**.
   b. Click the object that you want to select within the expanded node.
4. Display the **Properties** tab of the Message Definition editor by clicking **Properties** in the lower-left corner of the editor area. In the Message Definition editor, in the Properties Hierarchy, the name of each of the physical formats that have been added to the message set appears under **Physical Properties**. The object type (for example, Local Element or Global Element) of the message model object that you selected in the Outline view is displayed under each physical format shown.

   Tip: If the physical formats are not in view in the Properties Hierarchy, expand **Physical Properties** by clicking **+**. By default the TDS physical format is called Text1 but could have a user defined name instead.
5. Select the **Properties** tab located in the lower-left corner of the Message Definition editor.
6. Under **Physical Properties**, under the TDS physical format, click the object type for the message model object that you have chosen to configure. The TDS physical format properties of your selected message model object appear in the Details view.
7. Configure the TDS physical format properties of the selected object to your requirements by making the appropriate changes to the information shown in the Details view.

   Note: It is not possible to configure disabled fields.
8. Save your changes by selecting **File** > **Save message_definition_file.mxsd** from the menu or press Ctrl+S. Alternatively, from the menu, select **File** > **Save All**, or press Ctrl+Shift+S.

Results

Related concepts:

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

Related tasks:

"Configuring TDS properties: Message sets" on page 2852
Configure the Tagged/Delimited String (TDS) format properties of a message set using the Message Set editor.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

Configuring XML Wire Format properties: Message model objects:

You can configure the XML Wire Format properties of a message model object.

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 2842
- "Creating a message definition file" on page 2865
- "Opening an existing message definition file" on page 2864
- "Adding an XML wire format" on page 2853
- "Adding MRM message model objects" on page 2870 (You must have added one or more objects to your message model)

About this task

To configure the *XML Wire Format* properties of a message model object:

Procedure

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the WebSphere Message Broker Toolkit and is displaying the following information:
   - The name of the message definition file
   - Messages
   - Types
   - Groups
   - Elements and Attributes

   If the Outline view is not visible, from the WebSphere Message Broker Toolkit menu, click **Window** > **Show View** > **Outline**. If the above hierarchy is not displayed, ensure that the message definition file is open in the Message Definition Editor. Message definition files have an `.mxsd` file extension.
3. In the Outline view, select the object for which you want to configure the XML Wire Format properties by taking the following steps:
   a. Depending on the type of the object that you are selecting, expand **Messages**, **Types**, **Groups** or **Elements and Attributes** by clicking **+**.
   b. Click the object that you want to select within the expanded node.

4. Display the **Properties** tab of the Message Definition editor by clicking **Properties** in the lower-left corner of the editor area. In the Message Definition editor, in the Properties Hierarchy, the name of each of the physical formats that have been added to the message set appears under **Physical Properties**. The object type (for example, Local Element or Global Element) of the message model object that you selected in the Outline view is displayed under each physical format shown.

   Tip: If the physical formats are not in view in the Properties Hierarchy, expand **Physical Properties** by clicking **+**. By default the XML Wire Format is called XML1 but could have a user defined name instead.
5. Under **Physical Properties**, under the XML Wire Format, click the object type for the message model object that you have chosen to configure. The XML Wire Format properties of your selected message model object appear in the Details view of the Message Definition editor.
6. Configure the XML Wire Format properties of the selected object to your requirements by making the appropriate changes to the information shown in the Details view.

   Note: It is not possible to configure disabled fields.
7. Save your changes by clicking **File** > **Save message_definition_file.mxsd** or pressing Ctrl+S. Alternatively select **File** > **Save All** from the menu or press Ctrl+Shift+S.

Results

Related concepts:

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

Related tasks:

"Configuring XML Wire Format properties: Message sets" on page 2855
Configure the XML Wire Format properties of a message set using the Message Set editor.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

Applying default physical format settings: Message model objects:

You can apply the default physical format settings to a message model object that is contained in a message definition file.

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 2842

- "Creating a message definition file" on page 2865
- "Opening an existing message definition file" on page 2864
- "Adding MRM message model objects" on page 2870 (You must have added one or more objects to your message model)

This task assumes that you have added one or more physical formats to the relevant message set. For further information see "Adding a Custom Wire Format (CWF)" on page 2848 or "Adding an XML wire format" on page 2853 or "Adding a TDS physical format" on page 2851.

About this task

To apply the default physical format setting to a message model object previously added to a message definition file:

Procedure

1. Switch to the Broker Application Development perspective.
2. In the Outline view, click the object to which you want to apply default physical format settings.
3. Click the **Properties** tab located in the lower-left corner of the Message Definition editor.
4. Check that the Message Definition editor Properties Hierarchy displays the following information:
   - Logical Properties
   - Physical Properties (For each of the physical formats that have been added to the message set, the name of the physical format appears under **Physical Properties**. Under each physical format the type of message model object that you selected is displayed as a child.)
   - Documentation

   Ensure that **Physical Properties** in the Properties Hierarchy is fully expanded by clicking **+**.
5. Right-click on the message model object type underneath the physical format to which you want to apply the default settings then click **Apply default physical format settings**. The default physical format settings for the message model object that you selected are applied without warning.
6. Save your changes by clicking **File** > **Save message_definition_file.mxsd** from the menu or pressing Ctrl+S. Alternatively, from the menu, click **File** > **Save All**, or press Ctrl+Shift+S.

Results

Related concepts:

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

Deleting objects

Delete an object from your message model.

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 2842
- "Creating a message definition file" on page 2865
- "Opening an existing message definition file" on page 2864
- "Adding MRM message model objects" on page 2870 (You must have added one or more objects to your message model)

About this task

To *remove* objects contained within a message definition file:

Procedure

1. Ensure that the Outline view is visible in the Broker Application Development perspective of the WebSphere Message Broker Toolkit. If the Outline view is not visible, from the WebSphere Message Broker Toolkit menu, click **Window > Show View> Outline**.
2. In the Outline view, right-click the object that you want to remove then click **Delete** on the pop-up menu. Alternatively right-click the object in the Message Definition editor **Overview** tab, and then click **Delete**.

   The type of object and the relationship that it has with other objects determines whether the object is now deleted without a confirmation window appearing, or whether a confirmation window opens with a list of all the objects that will be deleted along with the one that you have selected.
3. If a confirmation window opens, click **OK** to delete the objects.

   Tip: You can undo a deletion by selecting **Edit> Undo**, provided that you have not saved the changes that you have made.

Results

Related concepts:

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

Creating a multipart message

A multipart message occurs when you embed a message in another message.

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 2842
- "Creating a message definition file" on page 2865
- "Opening an existing message definition file" on page 2864

About this task

To create a multipart (embedded) message:

Procedure

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the WebSphere Message Broker Toolkit. If the Outline view is not visible, from the WebSphere Message Broker Toolkit menu, click **Window** > **Show View** > **Outline**.
3. In the Outline view, add one of the following objects to your message model:
   - A complex type (for further information on completing this task see "Adding a complex type" on page 2889)
   - A global group (for further information on completing this task see "Adding a global group" on page 2890)
   - A local group (for further information on completing this task see "Adding a local group" on page 2891)

   Tip: You can also use a local `ANONYMOUS` complex type when creating a multipart message. For further information see "Adding a complex type to an element" on page 2905.
4. Display the **Properties** tab of the Message Definition editor by clicking **Properties** in the lower-left corner of the editor area.
5. In the Properties Hierarchy, under **Logical properties**, click one of the following items, depending on which of these you added in step 3:
   - **Complex Type**
   - **Global Group**
   - **Local Group**
6. In the Details view, make the following changes to the displayed logical properties:
   a. In the **Composition** drop-down list, click **message**.
   b. In the **Content validation** drop-down list, click **Open**, **Closed** or **Open Defined**, depending on your requirements. Note that if the embedded message is defined in a different message set, you must click **Open**. For further information about using these three options, see "MRM content validation" on page 5422.

Results

Note: There are a number of different ways for the parser to identify an embedded message within a message bit stream. For further information on identifying a message within another message refer to the following concept topics.

Related Concepts

"Multipart messages" on page 1191
A *multipart message* contains one or more other messages within its structure. The contained message is sometimes referred to as an *embedded message*.

Related Tasks

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

"Configuring MRM message model objects" on page 2896
Various tasks are involved in configuring MRM message model objects

"Accessing embedded messages in the MRM domain" on page 2594
If you have defined a multipart message, you have at least one message embedded within another. Within the overall complex type that represents the outer messages, you can model the inner message in two ways.

Related References

"Complex type logical properties" on page 5419
The logical properties of a complex type include properties that describe content and substitution settings.

"Global group logical properties" on page 5433
The logical properties of a global group.

Linking from one message definition file to another

Add an 'include', or an 'import' to the file that you want to reference.

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 2842
- "Creating a message definition file" on page 2865
- "Opening an existing message definition file" on page 2864

About this task

There are two ways to link one message definition file to another: either you can add an 'include', or you can add an 'import', for the file that you want to reference.

To check whether a message definition file currently includes or imports other files:

1. Open the message definition file in the Message Definition editor.
2. In the Outline view, in the displayed hierarchy, select the `.mxsd` file.
3. In the Properties Hierarchy, expand **Imports** or **Includes** as appropriate to display a list of the other files that the currently selected file includes or imports.

Include

About this task

Use the include option if you want to link to a message definition file with the same namespace, or if you want to link to a message definition file with no target namespace from a message definition file with a target namespace (chameleon

behavior). You must also add an include rather than an import if you want to link a message definition file with no target namespace to another message definition file that also has no target namespace.

Note: A message definition file can only reference objects in another message definition file if this other file has been included directly, so you might have a problem if you try to use the include option to include message definition files that are themselves included within other message definition files. For information about ways of resolving this situation, see "Resolving problems when developing message models" on page 3459.

This task assumes that you have opened an existing message definition file.

To add an include to a message definition file:

Procedure

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the WebSphere Message Broker Toolkit. If the Outline view is not visible, from the WebSphere Message Broker Toolkit menu, click **Window > Show View> Outline**.
3. In the Outline view, click the message definition (`.mxsd`) file name.
4. Display the **Properties** tab of the Message Definition Editor by clicking **Properties** in the lower-left corner of the editor area.
5. In the Properties Hierarchy, right-click **Includes** then click **Add Include** on the pop-up menu. The "Select Message Definition file to include" window opens.
6. In the Message Sets pane, select the message definition file that you want to include. If the message definition files within your project are not visible in this pane, expand the project hierarchy by clicking **+**.
7. Click **Finish**. The message definition file that you selected in step 4 is included within the message definition file that you opened before beginning this task.

Import

About this task

You use the import option if you want to link a message definition file to another message definition file in a different namespace. You cannot add an import from the same namespace. This restriction includes linking from a message definition file with no target namespace to another message definition file with no target namespace.

To add an import to a message definition file:

Procedure

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the WebSphere Message Broker Toolkit. If the Outline view is not visible, from the WebSphere Message Broker Toolkit menu, click **Window** > **Show View** > **Outline**.
3. In the Outline view, click the message definition (`.mxsd`) file name.
4. Display the **Properties** tab of the Message Definition editor by clicking **Properties** in the lower-left corner of the editor area.

5. In the Properties Hierarchy, right-click **Imports** then click **Add Import**. The "Select Message Definition file to import" window opens.
6. In the Message Sets pane, select the message definition file that you want to import from the workspace. If the message definition files within your project are not visible in this pane, expand the project hierarchy by clicking **+**.
7. Click **Finish**. The message definition file that you selected in step 4 is imported into the schema of the message definition file that you opened before beginning this task.

Related concepts:

"Namespaces in the message model" on page 1201
Use namespaces to qualify message model object names.

"Message definition files" on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

Related tasks:

"Generating documentation from message sets and message flows" on page 2962
You can generate documentation from your message sets, message flows, message definition files, message maps, Java files, ESQL files, and deployable WSDL files.

Related reference:

"Message definition file properties" on page 5409
The properties of a message definition file.

"Message definition file includes properties" on page 5410
The location of each message definition file that has been included in this message definition file is displayed.

"Message definition file imports properties" on page 5411
The file imports properties of a message definition.

"Message Definition editor" on page 6804
The Message Definition editor is the default editor provided by the Broker Application Development perspective for editing message definition (`.mxsd`) files.

Working with a message category file

This topic area lists the tasks that are involved when working with a message category file.

About this task

- "Creating a message category file" on page 2924
- "Opening an existing message category file" on page 2925
- "Adding a message to a message category" on page 2926
- "Deleting a message from a message category" on page 2928
- "Viewing or configuring message category file properties" on page 2928
- "Deleting a message category file" on page 2930

Related concepts:

"Message categories" on page 1200
Message category files have the suffix `.category`. These files are optional. You can have many message category files in a message set.

Related tasks:

"Adding MRM message model objects" on page 2870
Various tasks are involved in adding message model objects to a message definition file.

"Deleting objects" on page 2919
Delete an object from your message model.

"Configuring MRM message model objects" on page 2896
Various tasks are involved in configuring MRM message model objects

Related reference:

"Message category properties" on page 5413
A message category provides a way of grouping your messages.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

Creating a message category file

Create a message category file to add categories that you can use to group different message sets.

Before you begin

Before you start:

Complete the following task:

- "Creating a message set" on page 2842

About this task

To create a message category file, complete the following steps.

Procedure

1. Right-click in the Broker Development view, then select **New** > **Message Category File** to open the New Message Category File wizard.

   Tip: To preselect the message set when the wizard opens, either right-click the message set to which you are adding the message category file, or select the message set, before you open the wizard.
2. In the first pane, select the `Category Kind` for the type of category that you are creating.
   - other. This value indicates that this message category represents a generic grouping of messages. The Category Usage field is disabled.
   - wsdl. This value indicates that this message category represents a WSDL operation. The specified category name is used as the WSDL operation name.
3. Optional: If you set `Category Kind` to wsdl, specify the WSDL operation type by selecting one of the following values for the `Category Usage` field, then click **Next**.
   - wsdl:request-response
   - wsdl:solicit-response
   - wsdl:one-way
   - wsdl:notification
4. Select a folder under the target message set for the new message category file to be saved. The message set folder view is filtered to show only resources in the active working set.
5. In the **File name** field, type a name for the new message category file, then click **Next**. The file is automatically given the file extension of `.category`.
6. Select all messages that you want to add to the new category. Use **Shift-click** to select a range of messages, and **Ctrl-click** to select or clear individual messages.

You cannot complete the creation of the category file without adding one or more messages, and setting the **Role Type** and **Role Usage** values of each message correctly.

7. Click **Finish**.

Results

The message category file is created in the message set folder that you selected. The new message category file opens in the Message Category editor, so that you can view and edit it as required.

Related concepts:

"Message categories" on page 1200
Message category files have the suffix `.category`. These files are optional. You can have many message category files in a message set.

Related tasks:

"Adding MRM message model objects" on page 2870
Various tasks are involved in adding message model objects to a message definition file.

"Deleting objects" on page 2919
Delete an object from your message model.

"Configuring MRM message model objects" on page 2896
Various tasks are involved in configuring MRM message model objects

Related reference:

"Message Category editor" on page 6802
The Message Category editor is the default editor provided by the Broker Application Development perspective for working with message category (`.category`) files in a message set.

"Message category properties" on page 5413
A message category provides a way of grouping your messages.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

Opening an existing message category file

This describes how to open an existing message category file in the Message Category editor so that you can view or edit it.

Before you begin

Before you start:

To complete this task, you must have completed the following task:

- "Creating a message category file" on page 2924

About this task

To open an existing message category file:

Procedure

1. Switch to the Broker Application Development perspective.
2. In the Broker Development view, right-click the message category file (with a file extension of `.category`) that you want to open, then click **Open** on the pop-up menu. This opens the message category file that you have selected in the Message Category editor.

3. View and edit the message category file as required.

Results

Tip: The Eclipse framework lets you open resource files with other editors. You are advised to only use the WebSphere Message Broker Toolkit Message Category editor to work with the message category files because this editor correctly validates changes made to the files. Other editors might not do this.

Related concepts:

"Message categories" on page 1200
Message category files have the suffix `.category`. These files are optional. You can have many message category files in a message set.

Related tasks:

"Adding MRM message model objects" on page 2870
Various tasks are involved in adding message model objects to a message definition file.

"Deleting objects" on page 2919
Delete an object from your message model.

"Configuring MRM message model objects" on page 2896
Various tasks are involved in configuring MRM message model objects

Related reference:

"Message Category editor" on page 6802
The Message Category editor is the default editor provided by the Broker Application Development perspective for working with message category (`.category`) files in a message set.

"Message category properties" on page 5413
A message category provides a way of grouping your messages.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

Adding a message to a message category

You can add a message to a message category file by using the Message Category editor.

Before you begin

Before you start:

You must have completed the following tasks:

- "Adding MRM message model objects" on page 2870 (to create at least one message)
- "Creating a message category file" on page 2924
- "Opening an existing message category file" on page 2925

About this task

To add a message to a message category file:

Procedure

1. Open the Message Category editor.
2. In the Properties Hierarchy, open the Add Messages window by right-clicking **Message Category**, then clicking **Add Messages**. The Add Messages window

lists all the messages that are available for adding to the message category file. All messages that are in the message set but have not already been added to the category are displayed.

3. Select the message or messages that you would like to add. Use **Shift-click** to select a range of messages and **Ctrl-click** to select or clear individual messages.
4. Click **OK**. The selected message or messages are added to the message category and now appear in the Properties Hierarchy.

   Tip: Until you save the message category file, you can undo all additions that you make. To undo a change, right-click **Message Category** in the Properties Hierarchy, then click **Undo**. If you have added multiple messages, this action removes all the messages that you have added. If you want to remove a single message, right-click this message, then click **Undo**. To redo an addition after undoing it, use the **Redo** option.
5. Save and validate the additions that you have made to the message category file by clicking **File** > **Save**, or by pressing Ctrl+S.

Results

When you have saved the message category file after adding a message, you can no longer undo the addition of this message by using the **Undo** option. To remove a message after saving your changes, delete the message from the message category file.

When you have added a message to a message category file, you can configure its properties, according to your requirements, in the Message Category editor Details view.

Related concepts:

"Message categories" on page 1200
Message category files have the suffix `.category`. These files are optional. You can have many message category files in a message set.

Related tasks:

"Adding MRM message model objects" on page 2870
Various tasks are involved in adding message model objects to a message definition file.

"Deleting objects" on page 2919
Delete an object from your message model.

"Configuring MRM message model objects" on page 2896
Various tasks are involved in configuring MRM message model objects

"Viewing or configuring message category file properties" on page 2928
This topic describes how to view or configure the properties of a message category file and associated messages using the Message Category editor.

"Deleting a message from a message category" on page 2928
Delete a message from a message category file.

Related reference:

"Message Category editor" on page 6802
The Message Category editor is the default editor provided by the Broker Application Development perspective for working with message category (`.category`) files in a message set.

"Message category properties" on page 5413
A message category provides a way of grouping your messages.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

Deleting a message from a message category

Delete a message from a message category file.

Before you begin

Before you start:

To complete this task, you must have completed the following tasks:

- "Creating a message category file" on page 2924
- "Opening an existing message category file" on page 2925
- "Adding a message to a message category" on page 2926

About this task

To delete a message from a message category file:

Procedure

In the Message Category editor, in the Properties Hierarchy, right-click the message that you want to delete, then click **Delete** on the pop-up menu.

Tip: The message is deleted from the message category file immediately, without a warning appearing first.

Results

Related concepts:

"Message categories" on page 1200
Message category files have the suffix `.category`. These files are optional. You can have many message category files in a message set.

Related tasks:

"Adding MRM message model objects" on page 2870
Various tasks are involved in adding message model objects to a message definition file.

"Deleting objects" on page 2919
Delete an object from your message model.

"Configuring MRM message model objects" on page 2896
Various tasks are involved in configuring MRM message model objects

Related reference:

"Message Category editor" on page 6802
The Message Category editor is the default editor provided by the Broker Application Development perspective for working with message category (`.category`) files in a message set.

"Message category properties" on page 5413
A message category provides a way of grouping your messages.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

Viewing or configuring message category file properties

This topic describes how to view or configure the properties of a message category file and associated messages using the Message Category editor.

Before you begin

Before you start:

To complete this task, you must have completed the following tasks:

- "Creating a message category file" on page 2924
- "Opening an existing message category file" on page 2925
- "Adding a message to a message category" on page 2926 (You must have added one or more messages to your message category file)

About this task

To configure the properties of a message category file:

Procedure

1. Switch to the Message Category editor in the Broker Application Development perspective.
2. To view or configure the properties of a message category, click **Message Category** in the Properties Hierarchy. From the Details section of the Message Category editor you can now view the properties of the message category and make any changes to the properties that are necessary.
3. To view or configure the properties of a message in the message category file, click the name of the message in the Properties Hierarchy. From the Details section of the Message Category editor you can now view the properties of the message and make any changes to the properties that are necessary.
4. If you have changed any of the properties in the message category or messages, you can save those changes by selecting **File** > **Save** from the menu.

What to do next

Note: Note that some combinations of `Message Category Usage`, `Role Type` and `Role Usage` are not valid for WSDL and will result in task list errors being generated.

Related concepts:

"Message categories" on page 1200
Message category files have the suffix `.category`. These files are optional. You can have many message category files in a message set.

Related tasks:

"Adding MRM message model objects" on page 2870
Various tasks are involved in adding message model objects to a message definition file.

"Deleting objects" on page 2919
Delete an object from your message model.

"Configuring MRM message model objects" on page 2896
Various tasks are involved in configuring MRM message model objects

Related reference:

"Message Category editor" on page 6802
The Message Category editor is the default editor provided by the Broker Application Development perspective for working with message category (`.category`) files in a message set.

"Message category properties" on page 5413
A message category provides a way of grouping your messages.

"Message category member properties" on page 5414
This describes the properties that are associated with a message category member.

Deleting a message category file

You can delete a message category file from your message model.

Before you begin

Before you start:

To complete this task, you must have completed the following task:

- "Creating a message category file" on page 2924

About this task

To delete a message category file:

Procedure

1. Switch to the Broker Application Development perspective.
2. In the Broker Development view, right-click the message category file (`*.category` file extension) that you want to delete, then click **Delete**. Alternatively select the message category file in the Broker Development view, then either click **Edit** > **Delete**, or press the Delete key.
3. On the Confirm Resource Delete window, click **Yes** to delete the message category file. Alternatively, to cancel the message category file deletion, either click **No** or press the Esc key.

Results

Tip: After you have deleted a message category file, the action cannot be undone.

Related concepts:

"Message categories" on page 1200
Message category files have the suffix `.category`. These files are optional. You can have many message category files in a message set.

Related tasks:

"Adding MRM message model objects" on page 2870
Various tasks are involved in adding message model objects to a message definition file.

"Deleting objects" on page 2919
Delete an object from your message model.

"Configuring MRM message model objects" on page 2896
Various tasks are involved in configuring MRM message model objects

Related reference:

"Message category properties" on page 5413
A message category provides a way of grouping your messages.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

Working with data structures

You can create a message definition file in a message set by importing from XML Schema, XML DTD, SCA import or export components, IBM supplied messages,

WSDL definitions, IDL files, C header files, and COBOL copybooks. This topic area describes how to import from these data structures using the command line or the WebSphere Message Broker Toolkit.

About this task

Before you attempt to create a message definition from a data structure, by using the WebSphere Message Broker Toolkit, you are advised to read "Importing file systems into the WebSphere Message Broker Toolkit."

The following tasks topics relate to importing by using the WebSphere Message Broker Toolkit:

- "Importing from C" on page 2934
- "Importing from COBOL copybooks" on page 2937
- "Importing from IBM supplied messages" on page 2942
- "Importing SCA import or SCA export components" on page 2943
- "Generating a Broker SCA definition from a message set" on page 2967
- "Exporting an SCA Import or Export from a Broker SCA definition" on page 2945
- "Importing from WSDL" on page 2946
- "Importing an IDL file" on page 2952
- "Importing from XML DTD" on page 2954
- "Importing from XML Schema" on page 2957

The following tasks relate to importing by using the command line:

- "Importing from the command line" on page 2936 for C header files, COBOL Copybooks, XML DTDs and XML Schemas.
- "Importing WSDL definitions from the command line" on page 2948

Related concepts:

"Ways to create message definitions" on page 1253
When you have created a message set, you must populate the message set with message definitions.

"Importing from other model representations to create message definitions" on page 1254
You can add message definitions to your message set by importing application message formats that already exist.

Related tasks:

"Working with a message set project" on page 2838
Creating and deleting a message set.

Related reference:

"Import formats" on page 6346
Supported features of formats that have been imported from C, COBOL, SCA Export, SCA Import, WSDL, or XML Schema.

Importing file systems into the WebSphere Message Broker Toolkit

You can import file systems into the WebSphere Message Broker Toolkit by using the Import wizard, by dragging, or by copying.

About this task

Use one of the following options to import files for use by your selected message set project:

- "Using the Import wizard"
- "Dragging and dropping" on page 2933
- "Copying and pasting" on page 2933

You can then select the imported file in the New Message Definition File wizard to create a message definition that is based on the contents of this file.

Using the Import wizard:
About this task

Use the Import wizard to import all the files, or a selection of files, from the specified source.

To import files using the Import wizard:

Procedure

1. In the Broker Development view, click the project folder into which you are going to import the files.
2. Open the Import wizard by clicking **File** > **Import**.
3. On the Select page of the Import wizard, click either **File System** or **Archive file**, depending on the type of resource that you are importing.
4. Click **Next**.
5. On the File System page, in the **Directory** field, specify the import source. Either type the source name in the field, or click **Browse** and select the parent directory, or compressed file that contains the file or files that you want to import. Then click **OK** (directory) or **Open** (compressed file).

   Tip: Directories from which you recently imported files are shown in the list in the **Directory** field.
6. Using the left and right panes that are displayed under the **Directory** field, specify the folders or files, or both, that you want to import. Consider the following points when you are making your selections:
   - To import the entire contents of a folder, select the check box for this folder in the left pane. To view secondary folders within a folder, expand the folder by clicking the plus sign (+).
   - To import a specific file or files within a folder, use the right pane to select the individual files that you want to import. If you select a file or files in the right pane, the check box for the folder that contains these files is disabled in the left pane to indicate that only some of the files in the folder will be imported.
   - To restrict the type of files that you are importing, click **Filter Types**, then, on the Select Types window, select the check boxes for the file types that you want to include, and click **OK**. If you want to include files with extensions that are not shown in the list, type these extensions in the **Other Extensions** field.
   - To select all the folders and files that are shown on the File System page, click **Select All**.
   - To clear all the folders and files that are currently selected on the File System page, click **Deselect All**.

The **Select the destination for imported resources** field is already set to the name of the project folder that you selected in step 2.

7. Optional: To change the destination project or folder, click **Browse** to open the Folder Selection window. Select an alternative project folder by clicking the folder, then clicking **OK**.
8. Optional: To overwrite existing resources and not have a warning displayed, select the **Overwrite existing resources without warning** check box. This check box applies to both compressed files and file systems.
9. File system import only: Select one of the following options, depending on the folder structure that you want to create:
   - **Create complete folder structure**
   - **Create selected folders only**
10. Click **Finish**.

Results

The files that you selected are imported and are shown in the Broker Development view under the project folder that you selected.

Dragging and dropping:
About this task

You can use the drag-and-drop method to import files from your file system into the WebSphere Message Broker Toolkit. Drag the resources that you are importing to the exact location in the Broker Development view where you want the resources to be. Do not drag them onto a blank area in the Broker Development view.

To import files by dragging:

Procedure

1. In your file system, locate the file or folder that you want to import into the WebSphere Message Broker Toolkit.
2. Drag the file or folder to a specific location in the Broker Development view. When you are dragging resources into the Broker Development view, the project or folder into which you are trying to drop the resource is selected.
3. Ensure that the file or folder is copied into the WebSphere Message Broker Toolkit.

Results

Copying and pasting:
About this task

You can use the copy and paste function of your operating system as a method of importing a file system into the WebSphere Message Broker Toolkit.

To import files by copying and pasting:

Procedure

1. Locate the file or directory that you want to import into the WebSphere Message Broker Toolkit.

2. Using the copy and paste function in the operating system, copy the file or directory to your system clipboard.
3. Select the destination for the file or directory in the Broker Development view.
4. From the WebSphere Message Broker Toolkit menu, click **Edit** > **Paste**.

Results

The files or directories are copied into the WebSphere Message Broker Toolkit, and placed into the location that you selected.

Related concepts:

"Ways to create message definitions" on page 1253
When you have created a message set, you must populate the message set with message definitions.

"Importing from other model representations to create message definitions" on page 1254
You can add message definitions to your message set by importing application message formats that already exist.

Related tasks:

"Working with a message set project" on page 2838
Creating and deleting a message set.

Related reference:

"Import formats" on page 6346
Supported features of formats that have been imported from C, COBOL, SCA Export, SCA Import, WSDL, or XML Schema.

Importing from C

Create a message definition file from a C header file for use in the MRM and IDOC domains.

Before you begin

Before you start:

Complete the following tasks:

- "Creating a message set" on page 2842
- "Importing file systems into the WebSphere Message Broker Toolkit" on page 2931

Be aware of the following points:

- The wizard can import C header files with `.h`, `.c` and `.css` extensions. If your source file has a different extension you must rename it before attempting to import it.
- If the message set to which you are adding the new message definition file *does not* have an Custom Wire Format (CWF) layer only the logical information appears in the model. You can add the physical layer to the message set before or after importing a C header file, but you should add the physical layer *before* importing it to ensure that it is populated with settings from the C header file.
- You can import a C header file from the command line using **`mqsicreatemsgdefs`**.

About this task

The following steps cover both creating a completely new message definition file and overwriting the contents of an existing file.

To create a message definition file from a C header file:

Procedure

1. Switch to the Broker Application Development perspective.
2. Open the New Message Definition File wizard by clicking **File** > **New** > **Message Definition File** from the WebSphere Message Broker Toolkit menu. Alternatively, you can open the wizard by right-clicking a C header file previously imported into the WebSphere Message Broker Toolkit and clicking **New** > **Message Definition File** on the menu.
3. In the displayed list of options, click **C header file** then click **Next**.
4. Step through the remainder of the wizard completing the details as required.

Results

When you have completed importing the C header file using the wizard:

- Carefully check for any errors in the report that is created when the file is imported. You can find this report in the `log` directory within the project containing the message definition that you have attempted to create. The report has a `.c.report.txt` file extension, prefixed with the name that you specified for the new message definition file.
- Review the messages shown in the WebSphere Message Broker Toolkit task list to check whether any new warnings or errors have appeared.

Related concepts:

"Ways to create message definitions" on page 1253
When you have created a message set, you must populate the message set with message definitions.

"Importing from other model representations to create message definitions" on page 1254
You can add message definitions to your message set by importing application message formats that already exist.

Related tasks:

"Working with a message set project" on page 2838
Creating and deleting a message set.

"Importing from the command line" on page 2936
This describes how to use the command line importer **mqsicreatemsgdefs** to import C, COBOL copybooks, XML DTD or XML Schema in order to populate a message set with message definitions.

"Adding a Custom Wire Format (CWF)" on page 2848
You can add a Custom Wire Format (CWF) physical format layer to a message set by using the Message Set editor.

Related reference:

"Import formats" on page 6346
Supported features of formats that have been imported from C, COBOL, SCA Export, SCA Import, WSDL, or XML Schema.

"Importing from C (MRM): supported features" on page 6347
The C importer uses default values when mapping C data types to message model elements.

"New message definition file wizard: Create a new message definition file from a C header file" on page 6362
You can create a new message definition file from a C header file.

Importing from the command line:

This describes how to use the command line importer **mqsicreatemsgdefs** to import C, COBOL copybooks, XML DTD or XML Schema in order to populate a message set with message definitions.

Before you begin

Before you start:

Before you attempt this task, you should read the following information:

- "**mqsicreatemsgdefs** command" on page 3702

The command line importer allows you to create a new message set, into which the message definition files will be placed. When you create a new message set from the command line, only the logical information is created by default. However, the command line importer allows you to create a new message set based on an existing message set. The physical format information from the base message set is also created in the new message set. If you want physical format information to be created as well, you must do the following before you invoke the **mqsicreatemsgdefs** command:

1. Using the WebSphere Message Broker Toolkit, create a message set in your workspace that is to be used as a base message set.
2. To this base message set, add the physical formats that you want to be created in your new message set.

About this task

To import C, COBOL copybooks, XML DTD or XML Schema using the command line:

Procedure

1. Close the WebSphere Message Broker Toolkit. This must not be running when you use the command line importer.
2. Invoke the **mqsicreatemsgdefs** command from a command prompt specifying the message set project name, path name of the source files folder, and any other optional parameters that you require. If you want to add physical formats to the new message set that the **mqsicreatemsgdefs** command creates, specify the base message set that contains these physical formats as the `-base` parameter on the import command line.
3. When the command has completed, open `mqsicreatemsgdefs.report.txt`. This report is created when you invoke the **mqsicreatemsgdefs** command and by default is written to the directory from which you invoked the command. The report provides you with the following information:
   - Details of the parameters that were used when **mqsicreatemsgdefs** was invoked.
   - The message set level action.
   - The name of the file or files that have been imported.
   - Details of the import process (for example, any warnings that have been generated and message model objects that have been created).

- The number of files imported.

4. Start the WebSphere Message Broker Toolkit and switch to the Broker Application Development perspective. The message definition file that was created when you invoked **mqsicreatemsgdefs** is visible in the project that you specified.

Results

If an error occurs during the import of a C, COBOL copybook, XML DTD, or XML Schema file, carefully check any errors that the importer reports. By default, all errors are written to the screen and to the log file described above. To gather additional information about the import, specify the `-v` (Verbose) command line parameter. This parameter displays more detailed information as the import proceeds.

Related concepts:

"Ways to create message definitions" on page 1253
When you have created a message set, you must populate the message set with message definitions.

"Importing from other model representations to create message definitions" on page 1254
You can add message definitions to your message set by importing application message formats that already exist.

Related tasks:

"Working with a message set project" on page 2838
Creating and deleting a message set.

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Creating a message set" on page 2842
Use the New Message Set wizard to create a message set.

Related reference:

"**mqsicreatemsgdefs** command" on page 3702
Use the **mqsicreatemsgdefs** command to create message definition files.

"Import formats" on page 6346
Supported features of formats that have been imported from C, COBOL, SCA Export, SCA Import, WSDL, or XML Schema.

Importing from COBOL copybooks

This topic describes how to create a new message definition from a COBOL data structure using the New Message Definition File wizard in the WebSphere Message Broker Toolkit.

Before you begin

Before you start:

Complete the following tasks:

- "Creating a message set" on page 2842
- "Importing file systems into the WebSphere Message Broker Toolkit" on page 2931

Be aware of the following points:

- The wizard can import COBOL files with `.cbl`, `.ccp`, `.cob` and `.cpy` extensions. If your source file has a different extension, you must rename it before attempting to import it.
- If the message set to which you are adding the new message definition file *does not* have a Custom Wire Format (CWF) layer, or a Tagged/Delimited String (TDS) format layer, only the logical information appears in the model.

  You can add the physical layer to the message set before or after importing a COBOL data structure but ensure that you add the physical layer *before* you import the data structure to ensure that it is populated with settings from the COBOL copybook.
- You can import a COBOL data structure from the command line using **`mqsicreatemsgdefs`**.
- The copybook must not contain field names that are COBOL reserved keywords.

About this task

The following steps cover creating a new message definition file and overwriting the contents of an existing file.

To create a message definition file from a COBOL data structure:

Procedure

1. Switch to the Broker Application Development perspective.
2. Open the New Message Definition File wizard by clicking **File** > **New** > **Message Definition File** from the WebSphere Message Broker Toolkit menu. Alternatively, you can open the wizard by right-clicking a C header file previously imported into the WebSphere Message Broker Toolkit and clicking **New** > **Message Definition File** on the menu.
3. Click **COBOL file**, then click **Next**.
4. Step through the remainder of the wizard supplying the details as required.

   For more information, see "New message definition file wizard: Create a new message definition file from a COBOL file" on page 6363.

Results

When you have completed importing the COBOL file using the wizard:

- Carefully check for any errors in the report that is created when the file is imported. You can find this report in the `log` directory within the project containing the message definition that you have attempted to create. The report has a `.cobol.report.txt` file extension, prefixed with the name that you specified for the new message definition file.
- Review the messages shown in the WebSphere Message Broker Toolkit task list to check whether any new warnings or errors have appeared.

Related concepts:

"Ways to create message definitions" on page 1253
When you have created a message set, you must populate the message set with message definitions.

"Importing from other model representations to create message definitions" on page 1254
You can add message definitions to your message set by importing application message formats that already exist.

Related tasks:

"Working with a message set project" on page 2838
Creating and deleting a message set.

"Importing from the command line" on page 2936
This describes how to use the command line importer **mqsicreatemsgdefs** to import C, COBOL copybooks, XML DTD or XML Schema in order to populate a message set with message definitions.

"Adding a Custom Wire Format (CWF)" on page 2848
You can add a Custom Wire Format (CWF) physical format layer to a message set by using the Message Set editor.

"Developing SCA applications for non-XML data" on page 2103
The SCA nodes allow non-XML data to be sent and received from WebSphere Process Server by using the WebSphere MQ binding. For example, create a message model from a COBOL copybook, and use that message model to parse messages received from WebSphere Process Server.

Related reference:

"Import formats" on page 6346
Supported features of formats that have been imported from C, COBOL, SCA Export, SCA Import, WSDL, or XML Schema.

"Importing from COBOL: supported features" on page 6350
The COBOL importer uses a set of default values and behaviors when mapping COBOL data types to message model elements.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"New message definition file wizard: Create a new message definition file from a COBOL file" on page 6363
You can create a new message definition file from a COBOL file.

Importing from the command line:

This describes how to use the command line importer **mqsicreatemsgdefs** to import C, COBOL copybooks, XML DTD or XML Schema in order to populate a message set with message definitions.

Before you begin

Before you start:

Before you attempt this task, you should read the following information:

- "**mqsicreatemsgdefs** command" on page 3702

The command line importer allows you to create a new message set, into which the message definition files will be placed. When you create a new message set from the command line, only the logical information is created by default. However, the command line importer allows you to create a new message set based on an existing message set. The physical format information from the base message set is also created in the new message set. If you want physical format information to be created as well, you must do the following before you invoke the **mqsicreatemsgdefs** command:

1. Using the WebSphere Message Broker Toolkit, create a message set in your workspace that is to be used as a base message set.

2. To this base message set, add the physical formats that you want to be created in your new message set.

About this task

To import C, COBOL copybooks, XML DTD or XML Schema using the command line:

Procedure

1. Close the WebSphere Message Broker Toolkit. This must not be running when you use the command line importer.
2. Invoke the **mqsicreatemsgdefs** command from a command prompt specifying the message set project name, path name of the source files folder, and any other optional parameters that you require. If you want to add physical formats to the new message set that the **mqsicreatemsgdefs** command creates, specify the base message set that contains these physical formats as the `-base` parameter on the import command line.
3. When the command has completed, open `mqsicreatemsgdefs.report.txt`. This report is created when you invoke the **mqsicreatemsgdefs** command and by default is written to the directory from which you invoked the command. The report provides you with the following information:
   - Details of the parameters that were used when **mqsicreatemsgdefs** was invoked.
   - The message set level action.
   - The name of the file or files that have been imported.
   - Details of the import process (for example, any warnings that have been generated and message model objects that have been created).
   - The number of files imported.
4. Start the WebSphere Message Broker Toolkit and switch to the Broker Application Development perspective. The message definition file that was created when you invoked **mqsicreatemsgdefs** is visible in the project that you specified.

Results

If an error occurs during the import of a C, COBOL copybook, XML DTD, or XML Schema file, carefully check any errors that the importer reports. By default, all errors are written to the screen and to the log file described above. To gather additional information about the import, specify the `-v` (Verbose) command line parameter. This parameter displays more detailed information as the import proceeds.

Related concepts:

"Ways to create message definitions" on page 1253
When you have created a message set, you must populate the message set with message definitions.

"Importing from other model representations to create message definitions" on page 1254
You can add message definitions to your message set by importing application message formats that already exist.

Related tasks:

"Working with a message set project" on page 2838
Creating and deleting a message set.

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Creating a message set" on page 2842
Use the New Message Set wizard to create a message set.

Related reference:

"**mqsicreatemsgdefs** command" on page 3702
Use the **mqsicreatemsgdefs** command to create message definition files.

"Import formats" on page 6346
Supported features of formats that have been imported from C, COBOL, SCA Export, SCA Import, WSDL, or XML Schema.

Creating a message definition from a database definition

You can create a new message definition from a database definition file (`.dbm`) by using the New Message Definition File wizard in the WebSphere Message Broker Toolkit.

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 2842
- "Adding database definitions to the WebSphere Message Broker Toolkit" on page 2278

About this task

The following steps cover both creating a completely new message definition and overwriting the contents of an existing definition.

To create a message definition from a database definition:

Procedure

1. Switch to the Broker Application Development perspective.
2. Open the New Message Definition File wizard by clicking **File> New> Message Definition File** from the WebSphere Message Broker Toolkit menu.

   Tip: You can create a message definition by right-clicking a `.dbm` file.
3. In the displayed list of options, click **Database Definition File** to select it, then click **Next**.
4. Step through the remainder of the wizard completing the details as required. The final page of the wizard reports any errors, for example:
   - Unknown user-defined data types.
   - The database name, schema name, or table name contain characters. The non-NCName characters are replaced by an underscore.

Results

When you have completed creating the database definitions using the wizard:

- Carefully check for any errors in the report that is created when the file is imported. You can find this report in the `log` directory within the project containing the message definition that you have attempted to create. The report has a `.dbm.report.txt` file extension, prefixed with the name that you specified for the new message definition file.
- Review the messages shown in the WebSphere Message Broker Toolkit task list to check whether any new warnings or errors have appeared.

Related concepts:

"Ways to create message definitions" on page 1253
When you have created a message set, you must populate the message set with message definitions.

"Importing from other model representations to create message definitions" on page 1254
You can add message definitions to your message set by importing application message formats that already exist.

Related tasks:

"Working with a message set project" on page 2838
Creating and deleting a message set.

"Importing from the command line" on page 2936
This describes how to use the command line importer **mqsicreatemsgdefs** to import C, COBOL copybooks, XML DTD or XML Schema in order to populate a message set with message definitions.

Related reference:

"Import formats" on page 6346
Supported features of formats that have been imported from C, COBOL, SCA Export, SCA Import, WSDL, or XML Schema.

"New message definition file wizard: Create a new message definition file from a database definition" on page 6365
You can create a new message definition from a database definition file (`.dbm`) by using the New Message Definition File wizard in the WebSphere Message Broker Toolkit.

Importing from IBM supplied messages

You can create a new message definition file from an IBM supplied message.

Before you begin

Before you start:

You must have completed the following task:

- "Creating a message set" on page 2842

About this task

The following steps describe how to create a new message definition file, and how to overwrite the contents of an existing file.

To create a message definition from an IBM supplied message:

Procedure

1. Switch to the Broker Application Development perspective.

2. Open the New Message Definition File wizard by clicking **File** > **New** > **Message Definition File From** on the WebSphere Message Broker Toolkit menu.
3. In the displayed list of options, select **IBM supplied message** and click **Next**.
4. Complete the fields of the panel that is displayed by the wizard. See "New message definition file wizard: IBM supplied message" on page 6366.

Results

When you have finished the import of the IBM supplied message:

- Carefully check for any errors in the report that is created when the file is imported. You can find this report in the `log` directory within the project that contains the message definition that you have created. The report has a `.xsd.report.txt` file extension, prefixed with the name that you specified for the new message definition file.
- Review the messages shown in the WebSphere Message Broker Toolkit task list to check whether any new warnings or errors are displayed.

Related concepts:

"Ways to create message definitions" on page 1253
When you have created a message set, you must populate the message set with message definitions.

"Importing from other model representations to create message definitions" on page 1254
You can add message definitions to your message set by importing application message formats that already exist.

Related tasks:

"Working with a message set project" on page 2838
Creating and deleting a message set.

"Creating a message set" on page 2842
Use the New Message Set wizard to create a message set.

Related reference:

"Import formats" on page 6346
Supported features of formats that have been imported from C, COBOL, SCA Export, SCA Import, WSDL, or XML Schema.

"Message set preferences" on page 5366
Preferences for message sets.

"New message definition file wizard: IBM supplied message" on page 6366
You can create a new message definition file from an IBM supplied message.

"IBM supplied messages that you can import" on page 6367
You can import IBM supplied messages to create a new message definition file.

Importing SCA import or SCA export components

You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to import SCA import or SCA export components from WebSphere Integration Developer. You must import an SCA import or SCA export into the workspace to provide a broker SCA definition for use in configuring the SCA nodes.

Before you begin

Before you start:

There are two methods for importing from SCA import or export:

- Create a message set and use the New Message Definition File wizard. This method is described in this topic.
- Use the Start from SCA Import or Export Quick Start wizard. See "Creating an application based on SCA import or export files" on page 1422.

If you choose the first of these options, before you start you must have completed the following tasks:

- "Creating a message set" on page 2842
- "Importing file systems into the WebSphere Message Broker Toolkit" on page 2931

About this task

Ensure that the SCA import or SCA export components that you import from WebSphere Integration Developer use SOAP 1.1 bindings; a validation error occurs if you attempt to import SCA Import or SCA Export components that have been generated with SOAP 1.2 bindings.

The following steps are required to create a new message definition file, or to overwrite the contents of an existing file.

To create a message definition from an SCA import or SCA export:

Procedure

1. Switch to the Broker Application Development perspective.
2. Open the New Message Definition File From wizard by clicking **File** > **New** > **Message Definition File From ...** on the WebSphere Message Broker Toolkit menu.
3. In the displayed list of options, select **SCA Import or Export** and click **Next**. Alternatively, open the wizard by right-clicking a file with extension `.zip` that was previously imported into the WebSphere Message Broker Toolkit and clicking **New> Message Definition File From ...** on the menu.
4. Step through the remainder of the wizard supplying the details as required.

   You must choose whether the SCA import or SCA export that you want to import is in the current workspace in the WebSphere Message Broker Toolkit or is outside the workspace.

   Check boxes provide options to:

   - Copy the source file (or files) into the 'importFiles' directory of the message set project. By default, this check box is cleared.
   - Add appropriate supported domains if they do not exist. Selecting this check box adds the XMLNSC domain for all binding types; the SOAP domain is also added for Web service bindings.
   - Create an appropriate physical format if one does not exist. By default, this check box is selected.

   Note:

- The panels and options that are available in the wizard are dependent on the settings that you select.
- Some fields in the wizard might not be available perhaps because the field has a mandatory setting, or because the field has only one possible value, or because the field is not being used as a result of other settings that have been made.

Related tasks:

"Developing SCA applications for non-XML data" on page 2103
The SCA nodes allow non-XML data to be sent and received from WebSphere Process Server by using the WebSphere MQ binding. For example, create a message model from a COBOL copybook, and use that message model to parse messages received from WebSphere Process Server.

Related reference:

"New message definition file wizard: Create a new message definition file from an SCA Import or Export" on page 6369
You can create a new message definition file from an SCA import or export.

Exporting an SCA Import or Export from a Broker SCA definition

You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to export SCA import or SCA export components. You must export an SCA import or SCA export if the Broker SCA definition is to be used by WebSphere Integration Developer.

Before you begin

You must have completed the following task:

- "Generating a Broker SCA definition from a message set" on page 2967

About this task

The following steps are required to export an SCA Import or Export from a Broker SCA definition:

Procedure

1. Switch to the Broker Application Development perspective.
2. In the Broker Development view, right-click the folder that contains the SCA definition from which you want to export an SCA import or SCA export and select **Export** to start the Export wizard.
3. In the displayed list of options, select **Message Sets** > **SCA Import or Export from Broker SCA Definition** and click **Next**. This action starts the SCA Import or Export from Broker SCA Definition wizard.
4. Step through the remainder of the wizard and provide the details as required.

   Check boxes provide options to:

   - Overwrite existing files without warning. By default, this check box is cleared.
   - Apply working set filtering to artifact selections on this page. By default, this check box is selected.

   Note:

   - The panels and options that are available in the wizard are dependent on the settings that you select.

- Some fields in the wizard might not be available. This might be because the field has a mandatory setting, or because the field has only one possible value, or because the field is not being used as a result of other settings that have been made.

Results

The wizard creates a folder on the file system that contains the individual files from the broker SCA definition (.import or .export, .wsdl and .xsd files).

What to do next

In WebSphere Integration Developer, import the SCA import or export component and WSDL that have been exported from WebSphere Message Broker. Do not select the XSD files. See *Importing WSDL files* in the WebSphere Integration Developer Information Center.

Related tasks:

"Developing SCA applications for non-XML data" on page 2103
The SCA nodes allow non-XML data to be sent and received from WebSphere Process Server by using the WebSphere MQ binding. For example, create a message model from a COBOL copybook, and use that message model to parse messages received from WebSphere Process Server.

Related reference:

"Export SCA Import or Export from Broker SCA Definition wizard" on page 6380
The Export SCA Import or Export from Broker SCA Definition wizard exports an SCA import or export from a Broker SCA definition in a message set.

Importing from WSDL

You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to create a new message definition from WSDL.

Before you begin

There are two methods for importing from WSDL:

- Create a message set and use the New Message Definition File wizard. This method is described here.
- Use the Start from WSDL and/or XSD files Quick Start wizard. See "Creating an application based on WSDL or XSD files" on page 1413.

If you choose the first of these options, before you start you must have completed the following tasks:

- "Creating a message set" on page 2842
- "Importing file systems into the WebSphere Message Broker Toolkit" on page 2931

About this task

The following steps are required to create a completely new message definition file, or to overwrite the contents of an existing file.

To create a message definition from a WSDL file (or files):

Procedure

1. Switch to the Broker Application Development perspective.

2. Open the New Message Definition File From wizard by clicking **File** > **New** > **Message Definition File From** on the WebSphere Message Broker Toolkit menu.
3. In the displayed list of options, select**WSDL file** and click **Next**. Alternatively, open the wizard by right-clicking a `.wsdl` file that was previously imported into the WebSphere Message Broker Toolkit and clicking **New** > **Message Definition File From** on the menu.
4. Step through the remainder of the wizard completing the details as required.

   You must choose whether the WSDL file, or files, that you want to import are in the current workspace in the WebSphere Message Broker Toolkit or are outside the workspace.

   Check boxes provide options to:

   - Copy the source file (or files) into a directory of the message set project. By default, this check box is cleared.
   - Add the SOAP and XMLNSC domains to your message set so that you can use the SOAP nodes. By default, this check box is selected.

   Note:

   - The panels and options available in the wizard are dependant on the settings that you select.
   - Some fields in the wizard might not be available. This might be because the field has a mandatory setting, or because the field has only one possible value, or because the field is not being used as a result of other settings that have been made.

Results

When you have finished importing the WSDL file (or files) using the wizard:

- Check carefully for any errors in the report that is created when the file is imported. You can find this report in the `log` directory within the project that contains the message definition that you have tried to create. The report has a `<wsdl-file-name>.wsdl.report.txt` file descriptor, where `<wsdl-file-name>` is the name of the WSDL definition that you are importing.
- Review the messages that are shown in the WebSphere Message Broker Toolkit task list to check whether any new warnings or errors have appeared.

Note: Any required SOAP Envelope and SOAP encoding message definitions are automatically added to your message set during the import. If required, you can also import these manually using the New Message Definition File wizard by selecting the new option **IBM supplied message**.

Related concepts:

"Ways to create message definitions" on page 1253
When you have created a message set, you must populate the message set with message definitions.

"Importing from other model representations to create message definitions" on page 1254
You can add message definitions to your message set by importing application message formats that already exist.

"Importing WSDL files to create message definitions" on page 1267
Import WSDL files by using the **New Message Definition File From WSDL file** wizard, the **Start from WSDL and/or XSD files** Quick Start wizard, or the **`mqsicreatemsgdefsfromwsdl`** command.

"Importing XML Schema into message sets with namespaces disabled" on page 1259
You can import an XML Schema file with a target namespace even if the message set does not have namespaces enabled.

"Namespaces in the message model" on page 1201
Use namespaces to qualify message model object names.

Related tasks:

"Working with a message set project" on page 2838
Creating and deleting a message set.

"Importing from the command line" on page 2936
This describes how to use the command line importer **mqsicreatemsgdefs** to import C, COBOL copybooks, XML DTD or XML Schema in order to populate a message set with message definitions.

"Configuring message set preferences" on page 2840
This task topic explains how to make changes to preferences that relate to message set processing. These preferences are for message set editors, message set model validation, and importing XML Schema.

"Creating an application based on WSDL or XSD files" on page 1413
You can use existing WSDL or XSD files as the basis for your solution.

Related reference:

"Import formats" on page 6346
Supported features of formats that have been imported from C, COBOL, SCA Export, SCA Import, WSDL, or XML Schema.

"Message set preferences" on page 5366
Preferences for message sets.

"New message definition file wizard: Create a new message definition file from a WSDL file" on page 6370
You can create a new message definition file from a WSDL file.

Importing WSDL definitions from the command line:

WSDL definitions can be imported using the (**mqsicreatemsgdefsfromwsdl**) command.

Before you begin

Before you start:

Before you attempt this task, read the following information:

- "**mqsicreatemsgdefsfromwsdl** command" on page 3712.

The WSDL command line importer allows you to create a new namespace enabled message set into which the message definition files will be placed. It also allows you to add message definition files to an existing message set that is namespace enabled.

If you are adding new message definition files to an existing message set, the message set must have an XML physical format layer. To improve Web services interoperability, avoid unnecessary customization of the XML physical format layer for messages that participate in Web services processes.

When you create a new message set from the command line, only the logical information is created by default. If you require physical formats in the message set you have two options:

- Create a new message set based on an existing message set. The physical format information from the base message set is also created in the new message set.
- Use the WebSphere Message Broker Toolkit to create or open the message set and directly add the physical formats to the message set prior to importing the WSDL definitions into it.

Before starting the import, the **mqsicreatemsgdefsfromwsdl** command copies the WSDL files that it needs into the workspace. These are the top level WSDL file and any further files that might be imported by it. The files are copied under the specified message set in a folder called importFiles and are not removed after the import finishes. This allows you to update them, or run validation on them, in the WebSphere Message Broker Toolkit at a later time.

About this task

To import WSDL definitions using the command line:

Procedure

1. Close the WebSphere Message Broker Toolkit. The WebSphere Message Broker Toolkit must not be running when you use the command line importer.
2. Invoke the **mqsicreatemsgdefsfromwsdl** command from a command prompt; you must specify the message set project name, the path name of the directory where the top level WSDL file is located, the name of that file, the location of the workspace, and any other optional parameters that you require. If you want to add physical formats to the new message set that the **mqsicreatemsgdefsfromwsdl** command creates, specify the base message set that contains these physical formats as the `-base` parameter on the import command line.
3. When the command has completed, check the log file. The name of the log file is the name that you specified in the command, and it has the file extension `*.wsdl.report.txt`. This report is created when you invoke the **mqsicreatemsgdefsfromwsdl** command and, by default, it is written to the directory from which you invoked the command. The report provides you with the following information:
   - Details of the parameters that were used when **mqsicreatemsgdefsfromwsdl** was invoked.
   - The name of the file that has been imported.
   - Details of the import process (for example, any warnings that have been generated and message model objects that have been created).
4. Start the WebSphere Message Broker Toolkit and switch to the Broker Application Development perspective. The message definition file that was created by the **mqsicreatemsgdefsfromwsdl** command is visible in the project that you specified.

Results

If an error occurs during the import of a WSDL definition, carefully check any errors that are reported. By default, all errors are written both to the screen and to the file described above. To gather additional information about the import, specify the `-v` (Verbose) command line parameter. This parameter displays more detailed information as the import proceeds.

Related concepts:

"Ways to create message definitions" on page 1253
When you have created a message set, you must populate the message set with message definitions.

"Importing from other model representations to create message definitions" on page 1254
You can add message definitions to your message set by importing application message formats that already exist.

Related tasks:

"Working with a message set project" on page 2838
Creating and deleting a message set.

"Creating a message set" on page 2842
Use the New Message Set wizard to create a message set.

Related reference:

"**mqsicreatemsgdefsfromwsdl** command" on page 3712
Use the **mqsicreatemsgdefsfromwsdl** command to import a single WSDL definition.

"**mqsicreatemsgdefs** command" on page 3702
Use the **mqsicreatemsgdefs** command to create message definition files.

"Import formats" on page 6346
Supported features of formats that have been imported from C, COBOL, SCA Export, SCA Import, WSDL, or XML Schema.

Accepting self-signed certificates when importing WSDL:

You can import WSDLs that reference schemas on self-signed secure HTTPS servers, by adding security certificates to the Java Virtual Machine JVM.

About this task

The following procedure enables you to add certificates from the SSL server to each instance of your JVM.

If you are using Windows Vista or Windows 7, you must enter the commands from a console that has administration privileges.

Procedure

1. Obtain the certificate from the server (it is a `.cer` file) and copy it into your filesystem. somewhere.

   This example uses `D:\mb.cer`
2. Open a command prompt and navigate to your Java runtime environment (JRE) bin directory that is located in your WebSphere Message Broker install directory, for example, `C:\Program Files\IBM\WMBT700\jdk\jre\bin`.
3. Type in `keytool -printcert -file D:\mb.cer`
4. You obtain some output, and the important parameter to check is the `CN=value`.

   The value should be the same as the server name from which the WSDL is requesting files.
5. Input the certificate into a new keystore file.
   a. This procedure assumes that you can store your keystore file in `D:\mb.keystore`

Note, that the alias must be the same name as the server and the name can be anything you require.

For example, the name can be of the form `<userID>.<servername>.ibm.com` or `subdomain.messagebroker.com`

The example within this topic uses the form `<userID>.<servername>.ibm.com`

b. Type in:

```
keytool -import -alias <userID>.<servername>.ibm.com -file
 D:\mb.cer -keystore D:\mb.keystore
```

c. Import the certificate into a keystore file.

You are either asked for a password, or you need to create a password when the system requests one. This is the password used in Step 7, and the example within this topic uses the word `broker`.

d. Select `Yes` to trust the certificate.

6. Add the keystore as an argument when you start WebSphere Message Broker.

You must do this so that you can use the certificates you have just added to the keystore.

a. Go back three directories to: `C:\Program Files\IBM\WMBT700`.

7. Type in:

```
mb -vmargs -Djavax.net.ssl.trustStore=d:\\mb.keystore
 -Djavax.net.ssl.trustStorePassword=broker
```

8. Validate and import the WSDL

Results

You obtain a console output that is of the following format:

```
C:\Program
Files\IBM\WMBT700\jdk\jre\bin>keytool -printcert -file d:\
mb.cerOwner: EMAILADDRESS=jdoe@xx.ibm.com, CN=<userID>.
<servername>.ibm.com, OU=Message Broker Toolkit, O=IBM,
ST=<anystate>, C=<anycountry>Issuer: EMAILADDRESS=
jdoe@xx.ibm.com, CN=<userID>.<servername>.ibm.com,
OU=Message Broker Toolkit, O=IBM, ST=<anystate>, C=<anycountry>
Serial number: e1cabb1486f2bc7f
Valid from: 9/27/10 12:33 PM until: 9/27/11 12:33 PM
Certificate fingerprints:
         MD5:  ED:9B:BD:1C:C7:B5:8D:6E:F3:21:B7:92:26:25:52:9B
         SHA1: 5C:DE:70:CF:A5:64:96:16:C3:ED:4E:2C:A2:6E:EA:D3:A5:4B:69:BC

C:\Program
Files\IBM\WMBT700\jdk\jre\bin>keytool -import -alias <userID>
.<servername>.ibm.com -file d:\mb.cer -keystore  d:\mb.keystore
Enter keystore password:
Re-enter new password:
Owner: EMAILADDRESS=jdoe@xx.ibm.com, CN=<userID>.<servername>.ibm.com,
 OU=Message Broker Toolkit, O=IBM, ST=<anystate>, C=<anycountry>
Issuer: EMAILADDRESS=jdoe@xx.ibm.com, CN=<userID>.<servername>.ibm.com,
OU=Message Broker Toolkit, O=IBM, ST=<anystate>, C=<anycountry>
Serial number: e1cabb1486f2bc7f
Valid from: 9/27/10 12:33 PM until: 9/27/11 12:33 PM
Certificate fingerprints:
         MD5:  ED:9B:BD:1C:C7:B5:8D:6E:F3:21:B7:92:26:25:52:9B
         SHA1: 5C:DE:70:CF:A5:64:96:16:C3:ED:4E:2C:A2:6E:EA:D3:A5:4B:69:BC
Trust this certificate? [no]:  yes
Certificate was added to keystore

C:\Program
Files\IBM\WMBT700\jdk\jre\bin>cd ..
```

```
C:\Program
Files\IBM\WMBT700\jdk\jre>cd ..

C:\Program
Files\IBM\WMBT700\jdk>cd ..

C:\Program
Files\IBM\WMBT700>mb -vmargs -Djavax.net.ssl.trustStore=d:\\mb.keysto
re -Djavax.net.ssl.trustStorePassword=broker

C:\Program
Files\IBM\WMBT700>
```

If an error occurs during the import of a WSDL definition, carefully check any errors that are reported. By default, all errors are written both to the screen and to the file that has the format `*.wsdl.report.txt`.

Related tasks:

"Importing WSDL definitions from the command line" on page 2948
WSDL definitions can be imported using the (**mqsicreatemsgdefsfromwsdl**) command.

"Working with a message set project" on page 2838
Creating and deleting a message set.

"Creating a message set" on page 2842
Use the New Message Set wizard to create a message set.

Related reference:

"Import formats" on page 6346
Supported features of formats that have been imported from C, COBOL, SCA Export, SCA Import, WSDL, or XML Schema.

Importing an IDL file

You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to create a message definition from an IDL file.

Before you begin

Before you start:

Complete the following tasks:

- "Creating a message set" on page 2842
- "Importing file systems into the WebSphere Message Broker Toolkit" on page 2931
- Ensure that you have a valid IDL file. If you select an IDL file that is not valid, you see an error message and you cannot complete the wizard. When you import an IDL file, supported and unsupported operations are listed. You can import an IDL file that contains operations with types that are not supported by WebSphere Message Broker, but if you try to call an unsupported operation, you see an error message. The CORBA IDL file must contain at least one interface that has one operation. For more information about the IDL operations that are supported, see "CORBA support" on page 2149.
- For more information about how IDL types correspond to XML schema types, see "IDL data types" on page 2150.

About this task

The following steps describe how to use an IDL file to create a message definition file or overwrite the contents of an existing file.

Procedure

1. Right-click the message set, then click **New** > **Message Definition File From** > **CORBA IDL File** to open the New Message Definition File From wizard.
2. Complete the wizard by following the on-screen instructions.
   a. Select an IDL file either from the list of files in your workspace, or by using **Browse** to search outside your workspace. If you have imported an IDL file that contains includes, select the top-level IDL file.
   b. Ensure that the check box to add the DataObject domain to the message set is selected. By default, this check box is selected.
   c. Optional: You can provide a target namespace.
   d. By default, the message definition file name is the same as the name of the IDL file. You can change the name of the message definition file.
   e. If the IDL or message definition file exists, click **Next**. To rename or overwrite the existing files, select them.
3. Click **Finish**.
4. After you have imported the IDL file, check for errors.
   - Check for errors in the report that is created when the file is imported. You can find this report in the `log` directory of the project that contains the new message definition. The report is named `<idl-file-name>.idl.report.txt`, where `<idl-file-name>` is the name of the IDL file that you are importing.
   - Check for errors in the WebSphere Message Broker Toolkit task list.

Results

When you have finished importing the IDL file, the message definition opens. A read-only copy of the IDL file is stored in the `CORBA IDLs` folder.

For each IDL file, a single message definition is created. (If you have imported an IDL file that contains includes, all the elements and types in the IDL files are generated into a single message definition.) In the message definition, two messages are created for each operation in the IDL file (one message for the request, and one for the response), and a message is created for each user-defined exception. The request has a child element for each in and inout parameter; the response has a child element for each inout and out parameter, and a child element named "_return" for the return type of the operation.

The name of these elements is based on the interface name and operation name; for example, for the operation *sayHello* in the Interface *Hello*, the request element is called *Hello.sayHello*, and the response element is called *Hello.sayHelloResponse*. If the interface is contained in a module, the request and response element names are qualified with the names of the modules. For example, if the operation *sayHello* in the Interface *Hello* is contained in *ModuleB*, which in turn is contained in ModuleA, the response element would be called *ModuleA.ModuleB.Hello.sayHelloResponse*.

Another message definition is created with a message for each CORBA system exception.

What to do next

Next: Develop a message flow, as described in "Developing a message flow with a CORBARequest node" on page 2161.

Related concepts:

"CORBA nodes" on page 2147
Use CORBA nodes to connect WebSphere Message Broker with CORBA Internet Inter-Orb Protocol (IIOP) applications.

"CORBA support" on page 2149
The CORBA nodes in WebSphere Message Broker support a set of types and operations in imported IDL files.

"Ways to create message definitions" on page 1253
When you have created a message set, you must populate the message set with message definitions.

"Importing from other model representations to create message definitions" on page 1254
You can add message definitions to your message set by importing application message formats that already exist.

"Namespaces in the message model" on page 1201
Use namespaces to qualify message model object names.

Related tasks:

"Connecting to an external CORBA application" on page 2159
Connecting to an external CORBA application involves importing an IDL file, creating a message flow, building a message, and processing the response from the CORBARequest node.

"Resolving problems when you use CORBA nodes" on page 3396
Advice for dealing with common problems that can arise when you develop message flows that contain CORBA nodes.

Related reference:

"CORBARequest node" on page 4349
Use the CORBARequest node to call an external CORBA application over Internet Inter-Orb Protocol (IIOP).

"New message definition file wizard: Create a new message definition file from a CORBA IDL file" on page 6364
You can create a new message definition file from a CORBA IDL file.

Importing from XML DTD

You can create a new message definition from an XML DTD by using the New Message Definition File wizard in the WebSphere Message Broker Toolkit.

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 2842
- "Importing file systems into the WebSphere Message Broker Toolkit" on page 2931

Before you begin this task, you should be aware of the following points:

- If the message set to which you are adding the new message definition file *does not* have an XML wire format (XML) layer only the logical information appears

in the model. You can add the physical layer to the message set before or after importing from a XML DTD, but you should add the physical layer *before* importing it to ensure that it is populated with settings from the XML DTD.

- It is also possible to import an XML DTD from the command line using **mqsicreatemsgdefs**.
- The file extension must be `.dtd` in lowercase.

About this task

The following steps cover both creating a completely new message definition file and overwriting the contents of an existing file.

To create a message definition from an XML DTD:

Procedure

1. Switch to the Broker Application Development perspective.
2. Open the New Message Definition File wizard by clicking **File> New> Message Definition File** from the WebSphere Message Broker Toolkit menu.
3. In the displayed list of options, click **XML DTD file** to select it then click **Next**.
4. Step through the remainder of the wizard completing the details as required.

Results

When you have completed importing the XML DTD using the wizard:

- Carefully check for any errors in the report that is created when the file is imported. You can find this report in the `log` directory within the project containing the message definition that you have attempted to create. The report has a `.dtd.report.txt` file extension, prefixed with the name that you specified for the new message definition file.
- Review the messages shown in the WebSphere Message Broker Toolkit task list to check whether any new warnings or errors have appeared.

The message definition file is created from the XML DTD and is opened in the Message Definition editor so that you can check the imported information and make any required changes. While you are checking the newly created message definition file, review any messages that appear in the WebSphere Message Broker Toolkit task list to see whether you need to make any corrections to resolve errors or warnings relating to the new file.

Related concepts:

"Ways to create message definitions" on page 1253
When you have created a message set, you must populate the message set with message definitions.

"Importing from other model representations to create message definitions" on page 1254
You can add message definitions to your message set by importing application message formats that already exist.

Related tasks:

"Working with a message set project" on page 2838
Creating and deleting a message set.

"Importing from the command line" on page 2936
This describes how to use the command line importer **mqsicreatemsgdefs** to import C, COBOL copybooks, XML DTD or XML Schema in order to populate a

message set with message definitions.

"Adding an XML wire format" on page 2853
You can add an XML wire format physical format layer to a message set by using the Message Set editor.

Related reference:

"Import formats" on page 6346
Supported features of formats that have been imported from C, COBOL, SCA Export, SCA Import, WSDL, or XML Schema.

"New message definition file wizard: Create a new message definition file from an XML DTD file" on page 6371
You can create a new message definition file from an XML DTD file.

Importing from the command line:

This describes how to use the command line importer **mqsicreatemsgdefs** to import C, COBOL copybooks, XML DTD or XML Schema in order to populate a message set with message definitions.

Before you begin

Before you start:

Before you attempt this task, you should read the following information:

- "**mqsicreatemsgdefs** command" on page 3702

The command line importer allows you to create a new message set, into which the message definition files will be placed. When you create a new message set from the command line, only the logical information is created by default. However, the command line importer allows you to create a new message set based on an existing message set. The physical format information from the base message set is also created in the new message set. If you want physical format information to be created as well, you must do the following before you invoke the **mqsicreatemsgdefs** command:

1. Using the WebSphere Message Broker Toolkit, create a message set in your workspace that is to be used as a base message set.
2. To this base message set, add the physical formats that you want to be created in your new message set.

About this task

To import C, COBOL copybooks, XML DTD or XML Schema using the command line:

Procedure

1. Close the WebSphere Message Broker Toolkit. This must not be running when you use the command line importer.
2. Invoke the **mqsicreatemsgdefs** command from a command prompt specifying the message set project name, path name of the source files folder, and any other optional parameters that you require. If you want to add physical formats to the new message set that the **mqsicreatemsgdefs** command creates, specify the base message set that contains these physical formats as the `-base` parameter on the import command line.
3. When the command has completed, open `mqsicreatemsgdefs.report.txt`. This report is created when you invoke the **mqsicreatemsgdefs** command and by

default is written to the directory from which you invoked the command. The report provides you with the following information:

- Details of the parameters that were used when **mqsicreatemsgdefs** was invoked.
- The message set level action.
- The name of the file or files that have been imported.
- Details of the import process (for example, any warnings that have been generated and message model objects that have been created).
- The number of files imported.

4. Start the WebSphere Message Broker Toolkit and switch to the Broker Application Development perspective. The message definition file that was created when you invoked **mqsicreatemsgdefs** is visible in the project that you specified.

Results

If an error occurs during the import of a C, COBOL copybook, XML DTD, or XML Schema file, carefully check any errors that the importer reports. By default, all errors are written to the screen and to the log file described above. To gather additional information about the import, specify the **-v** (Verbose) command line parameter. This parameter displays more detailed information as the import proceeds.

Related concepts:

"Ways to create message definitions" on page 1253
When you have created a message set, you must populate the message set with message definitions.

"Importing from other model representations to create message definitions" on page 1254
You can add message definitions to your message set by importing application message formats that already exist.

Related tasks:

"Working with a message set project" on page 2838
Creating and deleting a message set.

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Creating a message set" on page 2842
Use the New Message Set wizard to create a message set.

Related reference:

"**mqsicreatemsgdefs** command" on page 3702
Use the **mqsicreatemsgdefs** command to create message definition files.

"Import formats" on page 6346
Supported features of formats that have been imported from C, COBOL, SCA Export, SCA Import, WSDL, or XML Schema.

Importing from XML Schema

You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to create a new message definition from an XML Schema

Before you begin

Before you start you must have completed the following tasks:

- "Creating a message set" on page 2842
- "Importing file systems into the WebSphere Message Broker Toolkit" on page 2931

Before you begin this task, you should be aware of the following points:

- If you are importing a collection of related XML Schema files, you are advised to use the **mqsicreatemsgdefs** command. This imports all the XML Schema files in a single operation, and automatically adjusts the import and include paths.
- If the message set to which you are adding the new message definition file *does* have an XML wire format layer, but *no* namespace support, the imported schema is modified to remove namespaces. For this reason, you should enable namespace support before importing a schema.
- If the message set to which you are adding the new message definition file *does not* have an XML wire format layer, but *does* have namespace support, only the logical information appears in the model. For this reason, you should add the physical layer to the message set before importing the schema. This ensures that the message set is populated with the settings and values from the schema. The XML Schema is not modified to remove namespaces.
- If the message set to which you are adding the new message definition file *does not* have an XML wire format layer, and *does not* have namespace support, only the logical information appears in the model and the imported schema is modified to remove namespaces.
- If you are working with a message set that does not have namespace support, you must specify the preferences that apply when you import a schema into the message set. These preferences allow you to specify how the importer treats certain individual schema constructs. You can either reject the schema if any occurrences of the construct are encountered or modify occurrences of the construct. If you choose modify, the importer modifies all occurrences of the construct.
- The extension to the XML Schema file must be `.xsd` in lowercase.

About this task

The following steps create a completely new message definition file or overwrite the contents of an existing file.

To create a message definition from an XML Schema file:

Procedure

1. Switch to the Broker Application Development perspective.
2. Open the New Message Definition File wizard by clicking **File> New> Message Definition File** on the WebSphere Message Broker Toolkit menu. Alternatively, you can open the wizard by right-clicking an `*.xsd` file that was previously imported into the WebSphere Message Broker Toolkit and clicking **New> Message Definition File** on the menu.
3. In the displayed list of options, click **XML Schema file** to select it, and then click **Next**.
4. Step through the remainder of the wizard, competing the details as required. The processing time for importing the XML Schema varies according to the size

and complexity of that schema. In a large and complex schema, it can take some time to import the file, generate the log file and display any task list warnings or errors.

Results

When you have finished importing the XML Schema using the wizard:

- Carefully check the log file for any warnings or errors in the report that is created when the file is imported. These warnings and error messages give information about whether the schema failed to import or needed to be modified to enable it to be successfully imported. You can find this report in the `log` directory structure within the project that contains the message definition that you have tried to create. The report has a `.xsd.report.txt` file extension, prefixed with the name that you specified for the new message definition file.
- Review the messages that are shown in the WebSphere Message Broker Toolkit task list to check whether any new warnings or error messages have appeared. Although you might have imported a perfectly valid schema, the task list will display warnings or error messages for any errors that exist in the message definition file. The following situations are examples where messages appear:
  - If the XML Schema that you are importing contains xsd:key, xsd:keyref and xsd:unique constructs, warning messages appear in the task list to tell you that these constructs are unsupported and will be ignored by the broker. If you prefer to delete these constructs, open the message definition file in the Message Definition editor, and delete the constructs as described in "Deleting objects" on page 2919. Deleting the constructs also removes the warning messages from the task list. If you decide not to delete the constructs, they remain in the message model but are not be deployed to the broker, or used for any other purpose. The warning messages remain in the task list, but you can use the message model normally.
  - If the XML Schema that you are importing contains xsd:redefine constructs, error messages appear in the task list to tell you that this construct is unsupported. If you right-click the error messages and select **Quick Fix**, you can choose to convert the xsd:redefine constructs into xsd:include constructs. This also removes the error messages.
  - If the XML Schema that you are importing contains xsd:attribute constructs that contain both a fixed value and a default value, error messages appear in the task list to tell you that this construct is unsupported. However, the schema is still imported and the fixed value is used, not the default value. The error messages can be ignored.
  - If you are importing a collection of related XML Schema files and the Message Definition Editor cannot resolve the links between two of the imported files, messages appear in the task list to say that referenced types or other objects cannot be found. If this occurs, refer to "Resolving problems when developing message models" on page 3459 for more information.

Related concepts:

"Ways to create message definitions" on page 1253
When you have created a message set, you must populate the message set with message definitions.

"Importing from other model representations to create message definitions" on page 1254
You can add message definitions to your message set by importing application message formats that already exist.

"Importing XML Schema into message sets with namespaces disabled" on page 1259
You can import an XML Schema file with a target namespace even if the message set does not have namespaces enabled.

"Namespaces in the message model" on page 1201
Use namespaces to qualify message model object names.

"XML Schema" on page 1172
XML Schema is an international standard from the World Wide Web Consortium (W3C) that defines a language for describing the structure of XML documents.

"XML Schema restrictions in message sets" on page 1172
Some XML Schema 1.0 features are not supported when message models are contained in message sets.

Related tasks:

"Working with a message set project" on page 2838
Creating and deleting a message set.

"Creating a message set" on page 2842
Use the New Message Set wizard to create a message set.

"Importing from the command line" on page 2936
This describes how to use the command line importer **mqsicreatemsgdefs** to import C, COBOL copybooks, XML DTD or XML Schema in order to populate a message set with message definitions.

"Configuring message set preferences" on page 2840
This task topic explains how to make changes to preferences that relate to message set processing. These preferences are for message set editors, message set model validation, and importing XML Schema.

"Adding an XML wire format" on page 2853
You can add an XML wire format physical format layer to a message set by using the Message Set editor.

"Deleting objects" on page 2919
Delete an object from your message model.

Related reference:

"Import formats" on page 6346
Supported features of formats that have been imported from C, COBOL, SCA Export, SCA Import, WSDL, or XML Schema.

"Message set preferences" on page 5366
Preferences for message sets.

"XML Schema Importer" on page 5370
Preferences for the message set XML Schema Importer.

"XSD options file for the **mqsicreatemsgdefs** command" on page 3709
Specify the options for the **mqsicreatemsgdefs** command when you import an XML Schema file.

Importing from the command line:

This describes how to use the command line importer **mqsicreatemsgdefs** to import C, COBOL copybooks, XML DTD or XML Schema in order to populate a message set with message definitions.

Before you begin

Before you start:

Before you attempt this task, you should read the following information:

- "**mqsicreatemsgdefs** command" on page 3702

The command line importer allows you to create a new message set, into which the message definition files will be placed. When you create a new message set from the command line, only the logical information is created by default. However, the command line importer allows you to create a new message set based on an existing message set. The physical format information from the base message set is also created in the new message set. If you want physical format information to be created as well, you must do the following before you invoke the **mqsicreatemsgdefs** command:

1. Using the WebSphere Message Broker Toolkit, create a message set in your workspace that is to be used as a base message set.
2. To this base message set, add the physical formats that you want to be created in your new message set.

About this task

To import C, COBOL copybooks, XML DTD or XML Schema using the command line:

Procedure

1. Close the WebSphere Message Broker Toolkit. This must not be running when you use the command line importer.
2. Invoke the **mqsicreatemsgdefs** command from a command prompt specifying the message set project name, path name of the source files folder, and any other optional parameters that you require. If you want to add physical formats to the new message set that the **mqsicreatemsgdefs** command creates, specify the base message set that contains these physical formats as the `-base` parameter on the import command line.
3. When the command has completed, open `mqsicreatemsgdefs.report.txt`. This report is created when you invoke the **mqsicreatemsgdefs** command and by default is written to the directory from which you invoked the command. The report provides you with the following information:
   - Details of the parameters that were used when **mqsicreatemsgdefs** was invoked.
   - The message set level action.
   - The name of the file or files that have been imported.
   - Details of the import process (for example, any warnings that have been generated and message model objects that have been created).
   - The number of files imported.
4. Start the WebSphere Message Broker Toolkit and switch to the Broker Application Development perspective. The message definition file that was created when you invoked **mqsicreatemsgdefs** is visible in the project that you specified.

Results

If an error occurs during the import of a C, COBOL copybook, XML DTD, or XML Schema file, carefully check any errors that the importer reports. By default, all errors are written to the screen and to the log file described above. To gather additional information about the import, specify the `-v` (Verbose) command line parameter. This parameter displays more detailed information as the import proceeds.

Related concepts:

"Ways to create message definitions" on page 1253
When you have created a message set, you must populate the message set with message definitions.

"Importing from other model representations to create message definitions" on page 1254
You can add message definitions to your message set by importing application message formats that already exist.

Related tasks:

"Working with a message set project" on page 2838
Creating and deleting a message set.

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Creating a message set" on page 2842
Use the New Message Set wizard to create a message set.

Related reference:

"**mqsicreatemsgdefs** command" on page 3702
Use the **mqsicreatemsgdefs** command to create message definition files.

"Import formats" on page 6346
Supported features of formats that have been imported from C, COBOL, SCA Export, SCA Import, WSDL, or XML Schema.

Generating documentation from message sets and message flows

You can generate documentation from your message sets, message flows, message definition files, message maps, Java files, ESQL files, and deployable WSDL files.

About this task

To generate documentation that describes your message sets, message flows, message definition files, message maps, Java files, ESQL files, and deployable WSDL files, complete the following steps.

Bidirectional text is not fully supported in generated documentation. For example, if you enter text in the WebSphere Message Broker Toolkit that has a right-to-left orientation, the text is displayed in the generated documentation with a left-to-right orientation.

Procedure

1. Switch to the Broker Application Development perspective.

2. In the pop-up menu of the Broker Development view, right-click a message set project, a message set, a message flow, a message definition file, a Java file, an ESQL file, or a deployable WSDL file, and select the action **Generate Documentation**. The Documentation Generation wizard opens.
3. Provide the information that is requested to describe the documentation report that you want, and click **Next** to move to the next panel of the wizard.
4. Step through the wizard, clicking **Next** to move to a new panel, and clicking **Finish** when you have described all the information that you want your report to document.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

"Generate model representations" on page 1270
After you have created and populated a message set, you can generate a message model in several different representations for use by a broker, a parser, or your applications.

Related tasks:

"Working with a message set" on page 2840
Complete a variety of tasks that are involved in working with a message set.

"Working with a message definition file" on page 2863
Create, open, and delete a message definition file.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

"Working with a message category file" on page 2923
This topic area lists the tasks that are involved when working with a message category file.

Generating XML Schemas

You can generate either a single XML Schema from a message definition file, or multiple XML Schemas from a message set.

About this task

To generate a single XML Schema from a message definition file, see "Generating an XML Schema" on page 2965.

To generate multiple XML Schemas (one from each message definition file in a message set) see "Generating multiple XML Schemas."

Generating multiple XML Schemas

You can generate an XML Schema for each message definition file in a message set.

Before you begin

You must have completed the following tasks:

- "Creating a message set" on page 2842
- "Working with a message definition file" on page 2863
- "Working with MRM message model objects" on page 2870

Note: WebSphere Message Broker uses XML Schema 1.0 to describe the logical structure of messages.

Tip: You should replace any deprecated constructs before you generate XML Schema representations of your models.

About this task

Procedure

1. Switch to the Broker Application Development perspective.
2. In the Broker Development view, right-click the message set folder from which you want to generate XML Schemas, and click **Generate> XML Schemas**.
3. The Generate XML Schemas window is displayed, and you must put into the **Zip file name** field the name of the compressed file (`*.zip` file extension) that you want to contain the generated XML Schemas.
4. Select a destination folder for this compressed file. You can choose a location either inside or outside the workspace:
   - Click **Create in a workspace directory** and select the required destination folder from the expanded workspace directory. The contents of the folder that you select are overwritten.

     If you want to create a new folder:

     a. Click the desired location.

     b. Click **Create New Folder**.

     c. Click OK
   - Click **Export to an external directory** and click **Browse** to expand the directory. Select a folder from the expanded directory. The contents of the folder that you select are overwritten.

     If you want to create a new folder:

     a. Click the desired location.

     b. Click **Make New Folder** and type the name of the new folder into the directory tree.

     c. Click OK
5. Optional: Choose from the list given in the **XML Wire Format** field an XML wire format that you want to use to generate the XML Schemas.

   Tip: You must have previously added one or more XML Wire Format layers to the message set if you want to use an XML physical format when you generate XML Schemas. For further information see "Adding an XML wire format" on page 2853.
6. If you do not want strict generation of an XML Schema, clear the **Strict generation** check box at the bottom of the Generate XML Schemas page. By default, this check box is selected.

   Tip: For further information on strict and lax generation of an XML Schema, see "Generate XML schema" on page 1272.
7. Click **Finish**. The compressed file that contains your generated XML Schemas is created.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

"Generate model representations" on page 1270
After you have created and populated a message set, you can generate a message model in several different representations for use by a broker, a parser, or your applications.

Related tasks:

"Working with a message set" on page 2840
Complete a variety of tasks that are involved in working with a message set.

"Working with a message definition file" on page 2863
Create, open, and delete a message definition file.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

"Working with a message category file" on page 2923
This topic area lists the tasks that are involved when working with a message category file.

Related reference:

"WSDL generation" on page 6340
Files and other objects are created by the WSDL Generator.

Generating an XML Schema

You can generate an XML schema from a message definition file.

Before you begin

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 2842
- "Working with a message definition file" on page 2863
- "Working with MRM message model objects" on page 2870

Note: WebSphere Message Broker uses XML Schema 1.0 to describe the logical structure of messages.

Tip: You should replace any deprecated constructs before generating XML Schema representations of your models.

About this task

This task topic describes how to generate an XML Schema from a message definition file:

Procedure

1. Switch to the Broker Application Development perspective.
2. In the Broker Development view, right-click the message definition file (`*.msxd` file extension) from which you want to generate an XML Schema, then click **Generate** > **XML Schema** on the menu.
3. The Generate XML Schema window is displayed, and the message definition file that you selected is highlighted. The message definition file list is filtered to only show artifacts in the active working set. If this is not the message definition file from which you want to generate an XML Schema, select the correct message definition file.
4. Optional: From the drop down list at the bottom of the Generate XML Schema window, select the XML Wire Format that you want to use to generate the XML Schema.

Tip: You must have previously added one or more XML Wire Format layers to a message set if you want to use an XML physical format when you generate XML Schema. For further information see "Adding an XML wire format" on page 2853.

5. If you do not want strict generation of an XML Schema, clear the **Strict generation** check box at the bottom of the Generate XML Schema page. By default, this check box is selected.

   Tip: For further information on strict and lax generation of XML Schema, see "Generate XML schema" on page 1272.
6. Click **Next** to move to the next page of the wizard.
7. Select a destination folder for the XML Schema. You can choose a location either inside or outside the workspace:
   - Click **Create in a workspace directory** and select the required destination folder from the expanded workspace directory. The contents of the folder that you select are overwritten.

     If you want to create a new folder:

     a. Click the desired location.
     b. Click **Create New Folder**.
     c. Click OK
   - Click **Export to an external directory** and click **Browse** to expand the directory. Select a folder from the expanded directory. The contents of the folder that you select are overwritten.

     If you want to create a new folder:

     a. Click the desired location.
     b. Click **Make New Folder** and type the name of the new folder into the directory tree.
     c. Click OK
8. Click **Finish**. Your XML Schema is generated.
9. Use the Broker Development view to locate the destination folder that you specified for the generated XML Schema. This folder contains a file with exactly the same name as your message definition file with the file extension `*.xsd`. This is the generated XML Schema. To view this file, right-click it, then click **Open** on the menu. This opens the schema editor.

   Tip: The **Design**, **Source** or **Graph** tabs located in lower-left corner of the schema editor provide you with different views of generated XML Schema.

Results

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

"Generate model representations" on page 1270
After you have created and populated a message set, you can generate a message model in several different representations for use by a broker, a parser, or your applications.

Related tasks:

"Working with a message set" on page 2840
Complete a variety of tasks that are involved in working with a message set.

"Working with a message definition file" on page 2863
Create, open, and delete a message definition file.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

"Working with a message category file" on page 2923
This topic area lists the tasks that are involved when working with a message category file.

Related reference:

"WSDL generation" on page 6340
Files and other objects are created by the WSDL Generator.

Generating a Broker SCA definition from a message set

WebSphere Message Broker creates a Broker SCA definition from existing message definitions.

Before you begin

Before you start: You must have created a message set that contains message definitions. The wizard gives an error if the set of message definitions is empty.

About this task

To generate a Broker SCA definition:

Procedure

1. Switch to the Broker Application Development perspective.
2. In the Broker Development view, right-click the folder that contains the message set file from which you want to generate a Broker SCA definition, and select **Generate** > **Broker SCA definition**. This action starts the Generate Broker SCA Definition wizard.
3. Step through the wizard providing the details as required. Some of the panels and options are subject to settings that you make within the wizard and might not always be shown. Also, some fields in the wizard might be grayed out. This happens when a field has a mandatory setting, or when the field is not used because of settings that have already been made in other fields.

   By default, the wizard creates the Broker SCA definition in the message set project.

Results

On completion of the Generate Broker SCA Definition wizard, you have generated either a `.insca` or a `.outsca` Broker SCA definition that is stored in the SCA category under the message set project.

A `.insca` Broker SCA definition is a compressed file that contains an SCA import and all the XSDs and WSDLs that are referenced directly or indirectly by the SCA import and a `.outsca` Broker SCA definition is a compressed file that contains an SCA export and all the XSDs and WSDLs that are referenced directly or indirectly by the SCA export.

Specifically, the Broker SCA definition contains:

- A WSDL interface that contains one or more operations.
- XSDs that correspond to the messages that are used in the operations defined in the interface.

- If you have selected a Web services binding, the WSDL contains service and binding information that defines the endpoint.
- If you have not selected a Web services binding, the SCA Import or Export must contain binding information for another binding that is supported by WebSphere Message Broker.

Related tasks:

"Creating a message set" on page 2842
Use the New Message Set wizard to create a message set.

"Developing SCA applications for non-XML data" on page 2103
The SCA nodes allow non-XML data to be sent and received from WebSphere Process Server by using the WebSphere MQ binding. For example, create a message model from a COBOL copybook, and use that message model to parse messages received from WebSphere Process Server.

"Exporting an SCA Import or Export from a Broker SCA definition" on page 2945
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to export SCA import or SCA export components. You must export an SCA import or SCA export if the Broker SCA definition is to be used by WebSphere Integration Developer.

Related reference:

"Broker SCA definition generation" on page 6340
The Broker SCA Definition wizard creates a Broker SCA definition that includes objects that define the binding, interface, and message format information to permit interoperation between WebSphere Message Broker and WebSphere Process Server.

"Generate Broker SCA Definition wizard" on page 6372
The Generate Broker SCA Definition wizard creates a Broker SCA definition from a message set.

Generating a WSDL definition from a message set

To ensure the highest interoperability of your Web services, use the document style of WSDL whenever possible. If rpc style WSDL is necessary, use literal encoding.

Before you begin

Before you start you must already have completed the following tasks:

- "Creating a message set" on page 2842

Replace any deprecated constructs before generating WSDL representations of your message models.

About this task

To generate a WSDL definition:

Procedure

1. Switch to the Broker Application Development perspective.
2. In the Broker Development view, right-click the folder that contains the message set file from which you want to generate a Web service definition, and select **Generate** > **WSDL Definition**. This starts the Generate WSDL wizard.
3. Step through the wizard completing the details as required. Some of the panels and options are subject to settings that you make within the wizard and might not always be shown. Also, some fields in the wizard might be greyed out. This

happens when a field has a mandatory setting, or when the field is not used because of settings that have already been made in other fields.

By default, the wizard creates the WSDL in the message set project. If you are going to use the WSDL to configure a SOAP node, create the WSDL in the message set, not the message set project.

Results

On completion of the Generate WSDL wizard, you have generated a WSDL definition. The file extension for WSDL files is .wsdl, and the file extension for any imported schema files in multi-file mode (where the WSDL definition is split over a number of files) is .xsd.

This following is an example of the WSDL that is generated for a JMS binding:

```
<wsdl:service name='HTTP'>
  <wsdl:port binding='tns:JMSSoapBinding' name='HTTP'>
    <wsdlsoap:address
       location='jms:/queue?destination=jms/MyQueue&amp;
                 connectionFactory=jms/MyCF&amp;
                 priority=5&amp;
                 targetService=GetQuote'/>
  </wsdl:port>
</wsdl:service>
```

Note: The various parts of the location string are broken over separate lines for clarity, but are generated as a continuous string without additional white space.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

"Generate model representations" on page 1270
After you have created and populated a message set, you can generate a message model in several different representations for use by a broker, a parser, or your applications.

Related tasks:

"Working with a message set" on page 2840
Complete a variety of tasks that are involved in working with a message set.

"Working with a message definition file" on page 2863
Create, open, and delete a message definition file.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

"Working with a message category file" on page 2923
This topic area lists the tasks that are involved when working with a message category file.

Related reference:

"WSDL generation" on page 6340
Files and other objects are created by the WSDL Generator.

"Generate WSDL wizard" on page 6382
The Generate WSDL wizard creates a WSDL definition from a message set.

Developing user-defined extensions

A user-defined extension is a component that you design and implement to extend the function of WebSphere Message Broker.

You can create and implement the following types of user-defined extension:

- User-defined nodes
- User-defined parsers
- User-defined exits

The user-defined nodes, parsers, and exits that you create can be used with the nodes and parsers that are supplied with the product, and with nodes and parsers that are supplied by independent software vendors.

For information about each type of user-defined extension that you can create, see the relevant topics listed under the related links. These topics help you to understand how your user-defined extension interacts with other components of WebSphere Message Broker, such as message flows and their associated execution groups. A good understanding of the broker architecture helps you to plan and construct user-defined extensions more effectively.

Related concepts:

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

"Why use a user-defined extension?" on page 2972
Use a user-defined node or parser when the built-in resources do not provide the required functions.

"Which type of user-defined extension to use" on page 2988
The types of user-defined extension that you can write include user-defined nodes, user-defined parsers, user-defined exits, and using a subflow as a user-defined node.

Related tasks:

"Implementing the supplied user-defined extension samples" on page 3017
WebSphere Message Broker provides some sample code to help you understand how to write user-defined nodes and parsers.

"Implementing user-defined extensions" on page 3019
Create the resources for your user-defined extension. You can write user-defined nodes in C or Java, or you can use a subflow to create a node. You can write user-defined parsers and exits only in C.

"Packaging and distributing user-defined extensions" on page 3117
When you have created and tested a user-defined extension, you can package and distribute it.

"Testing a user-defined node" on page 3094
When you have created and installed the required resources, you can test your user-defined node.

"Packaging and distributing a user-defined node project" on page 3121
Export the user-defined node project to make it available for other users.

Related reference:

"User-defined extensions" on page 6411
Reference material that supports the creation and management of your user-defined extensions.

User-defined extensions overview

A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

You can create the following types of user-defined extension:

- Input nodes
- Message processing nodes
- Output nodes
- Parsers
- User exits

The user-defined nodes and parsers that you create can be used with the nodes and parsers supplied with the product, and with nodes and parsers that are supplied by other vendors. You can configure a user-defined node to use a user-defined parser.

You can write user-defined exits and parsers only in the C programming language. You can write user-defined nodes in the C or the Java programming languages, or you can use a subflow as a user-defined node. You must compile user-defined nodes and parsers that are written in C into a loadable implementation library (LIL), and user exits that are written in C into a loadable exit library (LEL): that is, a shared library on Linux and UNIX systems, or a dynamic link library (DLL) on Windows systems. You must package user-defined nodes that are written in Java as a JAR file. You must import any user-defined nodes that you create into the WebSphere Message Broker Toolkit before you can use them.

The samples on the start screen of the WebSphere Message Broker Toolkit has examples of user-defined nodes and parsers. Look at the following sample for an example of how a node is created and used.

- User-defined Extension

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

To achieve platform independence, use the ANSI standard C or Java programming languages, and avoid platform-specific code in your user-defined extension.

The related links help you to understand how your user-defined extensions interact with other components of WebSphere Message Broker, such as message flows and execution groups. A good understanding of the broker architecture helps you to plan and construct user-defined extensions more effectively.

Related concepts:

"Why use a user-defined extension?" on page 2972
Use a user-defined node or parser when the built-in resources do not provide the required functions.

"Why use a user exit?" on page 2984
Use a user exit to intercept the progress of messages through message flows without having to redesign the message flow.

"Which type of user-defined extension to use" on page 2988
The types of user-defined extension that you can write include user-defined nodes, user-defined parsers, user-defined exits, and using a subflow as a user-defined

node.

"Which language to use to implement a user-defined extension" on page 3016
You can use Java or C to implement a user-defined extension.

"User-defined extensions in the runtime environment" on page 2980
Before you design and implement user-defined extensions, familiarize yourself with the core components. Ensure that you also understand the basic WebSphere Message Broker runtime architecture.

"Node and parser factory behavior" on page 2982
The node factory and the parser factory assume roles in declaring a node to the broker or defining a parser.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"C language user-defined node API" on page 6416
Learn about the different types of call provided by the API.

Related information:

Java user-defined extensions API

Why use a user-defined extension?

Use a user-defined node or parser when the built-in resources do not provide the required functions.

Before you start to create your user-defined extension, be clear about its purpose. Most tasks can be performed by using the functions already provided with WebSphere Message Broker, but you might need a user-defined extension for your particular task.

To write user-defined extensions you need to be a skilled programmer, with some knowledge of WebSphere Message Broker and its architecture, therefore make sure that you have the skills and knowledge required. You also need the time to test and debug your user-defined node or parser, and a safe environment in which to do this.

Also note that the maintenance and servicing of your own user-defined extensions is your responsibility. You should ensure that there will be someone available who can perform future updates or fixes.

A user-defined extension might be appropriate in the following situations:

- When you cannot manipulate the supplied nodes or parsers to perform the function you require. For example, you might want to connect to another software component in your message flow outside of WebSphere MQ. If there is no supplied node for doing this, you must create your own.
- When you can improve performance, ease of use, or reliability by using your own user-defined extensions in place of the supplied nodes or parsers.
- If the available choices are not appropriate for your requirement. You can create user-defined extensions to handle internal, customer-specific, or generic commercial messages formats.

Consider the following design factors when you are planning or writing a user-defined node or parser. You should be familiar with the concepts covered in the following topics before designing a user-defined extension.

- "Errors and exception handling" on page 2973

- "Storage management in user-defined nodes" on page 2976
- "String handling in user-defined nodes" on page 2977
- "Threading considerations for user-defined extensions" on page 2978
- "ODBC restrictions for user-defined nodes" on page 2979
- "User-defined extensions in the runtime environment" on page 2980
- "Node and parser factory behavior" on page 2982

Related concepts:

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

"Which type of user-defined extension to use" on page 2988
The types of user-defined extension that you can write include user-defined nodes, user-defined parsers, user-defined exits, and using a subflow as a user-defined node.

"Which language to use to implement a user-defined extension" on page 3016
You can use Java or C to implement a user-defined extension.

Related tasks:

"Implementing user-defined extensions" on page 3019
Create the resources for your user-defined extension. You can write user-defined nodes in C or Java, or you can use a subflow to create a node. You can write user-defined parsers and exits only in C.

Related reference:

"User-defined extensions" on page 6411
Reference material that supports the creation and management of your user-defined extensions.

Errors and exception handling:

Correct handling of errors and exceptions is important for correct broker operation. You must consider how and when your user-defined extension must handle errors and exceptions.

The errors and exception handling here describes factors that you must consider when you develop user-defined extensions for WebSphere Message Broker in the C programming language. If you are developing user-defined extensions using the Java programming language, you can use standard Java error and exception handling methods. If, for example, WebSphere Message Broker throws an exception internally, a Java exception of class **`MbException`** is made available.

The broker generates C++ exceptions to handle error conditions. These exceptions are caught in the relevant software layers in the broker, and are handled accordingly. However, programs written in C cannot catch C++ exceptions, and all exceptions thrown, by default, bypass all C user-defined extension code and are caught in a higher layer of the broker.

Utility functions, by convention, typically use the return value to pass back requested data; for example, the address or handle of a broker object. The return value sometimes indicates that a failure has occurred. For example, if the address or handle of a broker object could not be retrieved, zero (CCI_NULL_ADDR) is returned. Additionally, the reason for an error condition is stored in the return code output parameter, which is, by convention, part of the function prototype of all utility functions. If the utility function completed successfully and `returnCode`

was not null, `returnCode` contains CCI_SUCCESS. Otherwise, it contains one of the return codes described here. You can test the value of `returnCode` to determine whether a utility function was successful.

If the call to a utility function causes the broker to generate an exception, the error is visible to the user-defined extension only if it specified a value for the `returnCode` parameter to that utility function. If a null value was specified for `returnCode`, and an exception occurs:

- The user-defined extension is not be aware of that exception
- The utility function does not return to the user-defined extension
- Execution control passes to higher layers in the broker stack to process the exception

Therefore, a user-defined extension cannot perform its own error recovery. If, however, the `returnCode` parameter is specified, and an exception occurs, a return code of CCI_EXCEPTION is returned. In this case, cciGetLastExceptionData or cciGetLastExceptionDataW (the difference being that cciGetLastExceptionDataW returns a CCI_EXCEPTION_WIDE_ST which can contain Unicode trace text) can be used to obtain diagnostic information on the type of exception that occurred. The data is returned in the CCI_EXCEPTION_ST or CCI_EXCEPTION_WIDE_ST structure.

If there are no resources to be released, do not set the returnCode argument in your user-defined extension. Not setting this argument allows exceptions to bypass your user-defined extensions. These exceptions can then be handled higher up the WebSphere Message Broker stack, by the broker.

Message inserts can be returned in the CCI_STRING_ST members of the CCI_EXCEPTION_ST structure. The CCI_STRING_ST allows the user-defined extension to provide a buffer to receive all required inserts. The broker copies the data into this buffer, and returns the number of bytes output and the actual length of the data. If the buffer is not large enough, no data is copied and the "dataLength" member can be used to increase the size of the buffer, if required.

The user-defined extension can set a non-null value for `returnCode` and provide its own error recovery, if required. The utility function calls return to the user-defined extension and pass their status through `returnCode`. All exceptions that occur in a utility function must be passed back to the broker for additional error recovery to be performed; that is, when CCI_EXCEPTION is returned in `returnCode`. You do this by calling cciRethrowLastException, after the user-defined extension has completed its own error processing. Calling cciRethrowLastException causes the C interface to re-throw the last exception so that it can be handled by other layers in the broker. In the same way as the C **`exit`** call, **`cciRethrowLastException`** does not return in this case.

If an exception occurs and is caught by a user-defined extension, the extension must not call utility functions except cciGetLastExceptionData, cciGetLastExceptionDataW, or cciRethrowLastException. An attempt to call other utility functions results in unpredictable behavior that can compromise the integrity of the broker.

If a user-defined extension encounters a serious error, cciThrowException or cciThrowExceptionW can be used to generate an exception that is processed by the broker in the correct manner. The generation of such an exception causes the

supplied information to be written to the system log (syslog or Eventviewer) if the exception is not handled. The information is also written to trace (if trace is active).

Types of exception and broker behavior: The broker generates a set of exceptions that can be passed to a user-defined extension. These exceptions can also be generated by a user-defined extension when an error condition is encountered. The exception classes are:

Fatal Fatal exceptions are generated when a condition occurs that prevents the broker process from continuing execution safely, or where it is broker policy to terminate the process. Examples of fatal exceptions are a failure to acquire a critical system resource, or an internally-caught severe software error. The broker process terminates following the throwing of a fatal exception.

Recoverable
These exceptions are generated for errors which, although not terminal in nature, mean that the processing of the current message flow has to be ended. Examples of recoverable exceptions are invalid data in the content of a message, or a failure to write a message to an output node. When a recoverable exception is thrown, the processing of the current message is canceled on that thread, but the thread recommences execution at its input node.

Configuration
Configuration exceptions are generated when a configuration request fails. This can be because of an error in the format of the configuration request, or an error in the data. When a configuration exception is thrown, the request is rejected and an error response message is returned.

Parser These exceptions are generated by message parsers for errors that prevent the parsing of the message content or creating a bit stream. A parser exception is treated as a recoverable exception by the broker.

Conversion
These exceptions are generated by the broker character conversion functions if invalid data is found when trying to convert to another data type. A conversion exception is treated as a recoverable exception by the broker.

User These exceptions are generated when a Throw node throws a user-defined exception.

Database
These exceptions are generated when a database management system reports an error during broker operation. A database exception is treated as a recoverable exception by the broker.

Related concepts:

"Flow debugger overview" on page 3158
Use the flow debugger in the WebSphere Message Broker Toolkit to track messages through your message flows.

Related tasks:

"Throwing an exception" on page 2511
If you detect an error or other situation in your message flow in which you want message processing to be ended, you can throw an exception in a message flow in two ways.

Chapter 13, “Troubleshooting and support,” on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

Related reference:

“cciGetLastExceptionData” on page 6647
Gets diagnostic information about the last exception generated. Information about the last exception generated on the current thread is returned in a CCI_EXCEPTION_ST output structure. The user-defined extension can use this function to determine whether any recovery is required when a utility function returns an error code.

“cciLog” on page 6651
Use cciLog to write an error, warning, or informational event.

“cciRethrowLastException” on page 6659
This function re-throws the last exception that has been generated on the current thread. It is used to pass the exception back to the broker for further handling. The function, like a C **exit** call, does not return to the caller.

“cciThrowException” on page 6666
Use this function to throw an exception. The exception is thrown by the broker interface, and includes the specified arguments as exception data.

“cciGetLastExceptionDataW” on page 6649
Gets diagnostic information about the last exception generated. Information about the last exception generated on the current thread is returned in a CCI_EXCEPTION_WIDE_ST output structure. The user-defined extension uses this function to determine whether any recovery is required when a utility function returns an error code.

“cciLogW” on page 6653
cciLogW logs an error, warning, or informational event. The event is logged by the broker interface and uses the specified arguments as log data.

“cciThrowExceptionW” on page 6668
The cciThrowExceptionW exception is thrown by the broker interface and uses the specified arguments as exception data.

Storage management in user-defined nodes:

Consider issues that relate to storage management when you develop user-defined extensions in the C programming language.

If you are developing user-defined extensions using the Java programming language, you can use standard Java string handling methods.

All memory that is allocated by a user-defined extension must be released by the user-defined extension. The construction of a node at run time causes the cniCreateNodeContext function to be invoked, which allows the user-defined extension to allocate node instance specific data areas to store a context. The address of the context is returned to the broker, and is passed back from the broker when an internal method causes a user-defined extension function to be invoked; thus, the C user-defined extension can locate and use the correct context for the function processing.

The broker passes addresses of C++ objects to the user-defined extension, which are used as handles to be passed back on subsequent function calls. Your C

user-defined extension must not manipulate or use these pointers in any way, for example, by trying to release storage using the free function. Such actions cause unpredictable behavior in the broker.

The cniCreateNodeContext implementation function is invoked whenever the underlying node object has been constructed internally. It is called when a broker is defined with a message flow that uses a user-defined node. This activity is not necessarily the same as creating (or reusing) a thread to execute a message flow instance that contains the node. The cniCreateNodeContext function is called only once, during the configuration of the message flow, regardless of how many threads are executing the message flow.

Similar considerations apply to user-defined parsers, and the corresponding implementation function cpiCreateContext.

Related concepts:

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

Related reference:

"cpiCreateContext" on page 6553
This function creates a user-defined extension context associated with a parser object. It is called by the broker when an instance of a parser object is constructed or allocated. This action occurs when a message flow causes the message data to be parsed; the broker constructs or allocates a parser object to acquire the appropriate section of the message data.

"cniCreateNodeContext" on page 6447
This function creates a context for an instance of a node object. It is called by the broker whenever an instance of a node object is constructed. Nodes are constructed when a message flow is deployed by the broker, or when the execution group is started.

String handling in user-defined nodes:

Consider issues that relate to string handling when you develop user-defined extensions in the C programming language.

If you are developing user-defined extensions using the Java programming language, you can use standard Java string handling methods.

To enable a broker to handle messages in all languages at the same time, text processing within the broker is done in UCS-2 Unicode. UCS-2 Unicode character strings are also used across the Java and C language user-defined extension APIs to pass and return character data. Attributes are received in XML configuration messages as character strings, regardless of data type. If the true data type of an attribute is not a string, the cniSetAttribute function must perform the necessary verification and conversion before storing the attribute value. Similarly, when an attribute value is retrieved using cniGetAttribute2, conversion must be performed to a UCS-2 Unicode character string before returning the result.

`CciChar` defines a 16-bit character with UCS-2 Unicode representation. A `CciChar*` is a string of such characters terminated with a `CciChar` of 0. By default, a `CciChar` is represented by type `wchar_t`. However, some platforms do not have a convenient way of representing UCS-2 constants in source code, typically because of 4-byte `wchar_t` or EBCDIC representation. For example, a source-code constant such as L"ABC" expands to 12 bytes on Solaris.

For this reason, WebSphere Message Broker provides the utility functions cciMbsToUcs and cciUcsToMbs. Use these functions, where appropriate, to ensure portability of your user-defined nodes.

Related concepts:

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

Related reference:

"cniSetAttribute" on page 6511
This function sets the value of an attribute on a specific node instance. It is called by the broker when a configuration request is received that attempts to set the value of a node attribute, or during initialization of the node.

"cniGetAttribute2" on page 6484
Use this function to get the value of an attribute on a specific node instance. It is called by the broker after all the attributes that the user deploys are set.

"cciMbsToUcs" on page 6655
Use this function to convert multibyte string data to Universal Character Set (UCS).

"cciUcsToMbs" on page 6672
Use this function to convert Universal Character Set (UCS) data to multibyte string data. This function is, typically, used only for formatting diagnostic messages. Normal processing is best done in UCS-2, which can represent all characters from all languages.

Threading considerations for user-defined extensions:

Message processing nodes and parsers must work in a multi-instance, multithreaded environment. Many node objects or parser objects are available, each with several syntax elements, and many threads can be executing methods on these objects.

An instance of a message flow processing node is shared and used by all the threads that service the message flow in which the node is defined. Parsers are invoked on the same thread as the nodes, therefore, if the flow is using multiple threads, the parsers are as well.

A user-defined extension must use this model. If a user-defined node requires global data or resources, you must protect the global data or resources by using semaphores to serialize access across threads. However, such serialization can result in performance bottlenecks. Avoid using global data and resources to create a more scalable solution.

The functions implemented by user-defined extensions must be reentrant, and any functions that they invoke must also be reentrant. All user-defined extension utility functions are fully reentrant.

Although a user-defined extension can create additional threads if required, all C utility functions and Java methods must be invoked on the same thread that called the cniEvaluate function in C or the evaluate method in Java, as appropriate for the language in which the node is written. If the same thread is not used, your code might compromise the integrity of the broker and cause unpredictable behavior. Any additional threads must not call the user-defined extension API. The API must only be used from the main thread that is invoked by the Broker.

For information about the cniEvaluate function see "cniEvaluate" on page 6475.

Related concepts:

"User-defined input nodes" on page 2990
A user-defined input node is an extension to the broker that provides a new input node in addition to the nodes supplied with the product.

"Execution and threading models in a message flow" on page 1279
The execution model is the system used to start message flows which process messages through a series of nodes.

"User-defined extensions execution model" on page 2981
The execution model is the system used to start message flows through a series of nodes.

Related tasks:

"Creating an input node in C" on page 3027
Create a user-defined input node in C to receive messages into a message flow.

"Extending the capability of a C input node" on page 3034
When you have created a user-defined node, you can extend its capability.

"Creating an input node in Java" on page 3055
An input node is used to receive a message into a message flow, typically from a source that is not supported by the built-in input nodes.

Related reference:

"cniDispatchThread" on page 6456
Use this function to dispatch a new message flow thread to call another thread instance to run the user-defined message flow input node.

Related information:

Java user-defined extensions API

ODBC restrictions for user-defined nodes:

The ODBC environment cannot be accessed using the Java or C language user-defined extension API.

Database access must be performed using the supplied processing nodes, or by using the following implementation functions supplied for that purpose:

- "cniSqlCreateStatement" on page 6527
- "cniSqlExecute" on page 6531
- "cniSqlSelect" on page 6535
- "cniSqlDeleteStatement" on page 6530

Java Database Connectivity

Types 2 and 4 JDBC drivers are supported, but are not provided with the broker.

Related concepts:

"Why use a user-defined extension?" on page 2972
Use a user-defined node or parser when the built-in resources do not provide the required functions.

"Errors and exception handling" on page 2973
Correct handling of errors and exceptions is important for correct broker operation. You must consider how and when your user-defined extension must handle errors and exceptions.

"Storage management in user-defined nodes" on page 2976
Consider issues that relate to storage management when you develop user-defined extensions in the C programming language.

"String handling in user-defined nodes" on page 2977
Consider issues that relate to string handling when you develop user-defined extensions in the C programming language.

Related reference:

"C language user-defined node API" on page 6416
Learn about the different types of call provided by the API.

Related information:

Java user-defined extensions API

User-defined extensions in the runtime environment:

Before you design and implement user-defined extensions, familiarize yourself with the core components. Ensure that you also understand the basic WebSphere Message Broker runtime architecture.

Ensure that you are familiar with the following runtime components and concepts:

- "The broker environment" on page 46
- "Execution groups" on page 53
- "User-defined extensions execution model" on page 2981

Also make sure that you understand the following concepts:

- "Message flows overview" on page 1022

When you have gained an understanding of the runtime environment, read the following topics to help you understand how your user-defined extension interacts with the runtime components.

- "C user-defined input node life cycle" on page 2991
- "Java user-defined input node life cycle" on page 2993
- "C user-defined message processing nodes life cycle" on page 2997
- "Java user-defined message processing nodes life cycle" on page 3000
- "User-defined output node life cycle" on page 3007
- "User-defined parser life cycle" on page 3011

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

"User-defined extensions execution model"
The execution model is the system used to start message flows through a series of nodes.

"Developing user-defined extensions" on page 2970
A user-defined extension is a component that you design and implement to extend the function of WebSphere Message Broker.

User-defined extensions execution model:

The execution model is the system used to start message flows through a series of nodes.

When an execution group is initialized, the appropriate loadable implementation library (LIL) files and Plug-in Archive (PAR) files are made available to the runtime environment. The execution group runtime process starts, and creates a dedicated configuration thread. You are responsible for ensuring that a user-defined node is thread-safe. If a node updates a variable across multiple threads, appropriate locking must be in place. Do not compromise this threading model in your implementation of user-defined nodes. Consider the following points:

- An input message sent to a message flow is processed only by the thread that received it.
- A single instance of a user-defined extension might be invoked on several threads concurrently.
- The message flow execution environment is conceptually like procedural programming. Nodes that you insert into a message flow are like subroutines called using a function call interface. However, rather than a call-return interface, in which parameters are passed in the form of input message data, the execution model is referred to as a propagation-and-return model.

As an example, consider a message flow in which you use both user-defined nodes and parsers. You use a user-defined node to process messages, and a user-defined parser to parse messages; both the node and parser contain implementation functions. The broker calls the implementation functions, or callback functions, when certain events occur:

- When an input message is received by the message flow and is propagated to the user-defined node:
  - For C nodes, the broker calls the cniEvaluate function for the user-defined node. See "cniEvaluate" on page 6475.
  - For Java nodes, the broker calls the evaluate method that is implemented by the user-defined node.
- If the user-defined node wants to query the message to decide what to do with it, the node calls a C utility function or a Java method, as appropriate for the language in which the node is written.

The broker invokes the user-defined parser on one of its implementation functions, for example cpiParseFirstChild. This function instructs the parser to build the parse tree. The parser builds the tree by invoking utility functions that create elements in the parse tree, for example cpiCreateElement. The parser can be called many times by the broker.

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

"Execution groups" on page 53
An execution group is a named grouping of message flows that have been assigned to a broker. The broker enforces a degree of isolation between message flows in distinct execution groups by ensuring that they run in separate address spaces, or as unique processes.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Execution and threading models in a message flow" on page 1279
The execution model is the system used to start message flows which process messages through a series of nodes.

"Threading considerations for user-defined extensions" on page 2978
Message processing nodes and parsers must work in a multi-instance, multithreaded environment. Many node objects or parser objects are available, each with several syntax elements, and many threads can be executing methods on these objects.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

Related reference:

"C language user-defined node API" on page 6416
Learn about the different types of call provided by the API.

"C language user-defined parser API" on page 6538
The C language user-defined parser API consists of two complementary sets of functions that you can use to define the function of your parser.

Related information:

Java user-defined extensions API

Node and parser factory behavior:

The node factory and the parser factory assume roles in declaring a node to the broker or defining a parser.

Each loadable implementation library (LIL) has one node factory, or one parser factory, or has both. A node factory can identify many nodes, and a parser factory can identify many parsers.

When the broker loads the LIL, it calls the following functions:

- **`bipGetMessageflowNodeFactory`**

  After the operating system has loaded and initialized the LIL, the broker calls initialization function `bipGetMessageflowNodeFactory`. The `bipGetMessageflowNodeFactory` function calls the utility function `cniCreateNodeFactory`, which passes back a factory name (or group name) for all the nodes that your LIL supports.

- **`bipgetparserfactory`**

  After the operating system has loaded and initialized the LIL, the broker calls initialization function `bipgetparserfactory`. The `bipgetparserfactory` function defines the name of the factory that the user-defined parser supports, and the classes of objects, or shared object, that the factory supports. The initialization

function `bipgetparserfactory` calls the utility function `cpiCreateParserFactory`, which passes back a factory name (or group name) for all the parsers that your LIL supports.

Before the node factory is returned, the broker calls the following functions:

1. **cniCreateNodeFactory**

   This function creates a single instance of the node factory in the broker.

2. **cndDefineNodeClass**

   This function defines the name of a node class that a node factory supports, and identifies the nodes that the node factory can create.

Before the parser factory is returned, the broker calls the following functions:

1. **cpiCreateParserFactory**

   This function creates a single instance of the named parser factory in the message broker.

2. **cpiDefineParserClass**

   This function defines the name of a parser class that a parser factory supports, and identifies the parsers that the factory can create.

See the following topics for information on these functions:

- "cniCreateNodeFactory" on page 6449
- "cpiCreateParserFactory" on page 6555
- "cniDefineNodeClass" on page 6451
- "cpiDefineParserClass" on page 6557

Related concepts:

"Why use a user-defined extension?" on page 2972
Use a user-defined node or parser when the built-in resources do not provide the required functions.

"Planning user-defined input nodes" on page 2995
Before you develop a user-defined input node, plan and design its content and purpose.

"Planning user-defined message processing nodes" on page 3002
Plan how to write your message processing node or output node, and how to navigate the message within the node.

"Planning user-defined output nodes" on page 3008
A user-defined output node generates an output bit stream from a message tree.

"Planning user-defined parsers" on page 3013
Read about the concepts that you should consider before you develop a user-defined parser.

"User-defined extensions in the runtime environment" on page 2980
Before you design and implement user-defined extensions, familiarize yourself with the core components. Ensure that you also understand the basic WebSphere Message Broker runtime architecture.

"C user-defined input node life cycle" on page 2991
A user-defined input node that is written in the C programming language progresses through several stages during its lifetime.

"Java user-defined input node life cycle" on page 2993
A user-defined input node that is written in the Java programming language progresses through several stages during its lifetime.

"User-defined input nodes" on page 2990
A user-defined input node is an extension to the broker that provides a new input node in addition to the nodes supplied with the product.

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

Related tasks:

"Creating an input node in C" on page 3027
Create a user-defined input node in C to receive messages into a message flow.

"Creating an input node in Java" on page 3055
An input node is used to receive a message into a message flow, typically from a source that is not supported by the built-in input nodes.

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

"Creating a message processing or output node in Java" on page 3062
A message processing node is used to process a message, and an output node is used to output a message as a bit stream.

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Why use a user exit?

Use a user exit to intercept the progress of messages through message flows without having to redesign the message flow.

User exits provide a mechanism to apply actions (such as monitoring, message tracking, and auditing) operationally to deployed message flows at run time.

You can use user exits to call (by using callbacks) your custom C code, which is provided in a loadable exit library (LEL), at key points in message transactions in deployed message flows. These user exits can use utility functions from the user-defined extensions APIs to extract details of the broker, execution group, message flow, node, and message assembly. In addition, the user exits can use utility functions from the user-defined extensions APIs to modify parts of the message assembly.

To write user exits, you must be a skilled programmer with an understanding of WebSphere Message Broker and its architecture. Testing and debugging user exits can be time-consuming, and must be done in a safe environment. You must also maintain and service your own user exit.

Consider the following design factors when you plan and write a user exit:

- The effect on performance

User exit callbacks are run inline with the current message transaction; that is, progress of the transaction is blocked until the return from the callback is received. Updating the message in a user exit callback can affect performance, particularly if the input message would not otherwise be changed in the message flow.

- Message parse timing

  On-demand parsing, referred to as partial parsing, is used to parse a message bit stream only as far as is necessary to satisfy the current reference in the message assembly. A user exit can navigate the message at each of its callback points, which can mean that the parse timing of the message flow is changed when you enable the user exit.

- Error handling

  To ensure that the error handling that is provided by the designer of a message flow that is being intercepted by a user exit continues to operate as designed, you must program the user exit in the following way:

  – All internal errors must be handled within the user exit, and the normal return from the callback must enable the message flow transaction to complete as normal.

  – All exception condition that is encountered when the user exit calls utility functions in the user-defined extensions APIs must be returned to the flow for normal error processing. This behavior is achieved by calling cciRethrowLastException() to cut short the callback processing.

Related concepts:

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

"User exits" on page 3015
A user exit is user-provided custom software, written in C, to track data passing through message flows.

Related tasks:

"Exploiting user exits"
Your message flows can benefit from user exits.

"Implementing a user-defined exit" on page 3113
You can develop and deploy a user-defined exit.

"Developing a user exit" on page 3114
Develop a user exit by declaring it, implementing its behavior, then compiling it.

"Deploying a user exit" on page 3116
Deploy your user exit to the broker.

Related reference:

"User-defined extensions" on page 6411
Reference material that supports the creation and management of your user-defined extensions.

Exploiting user exits:

Your message flows can benefit from user exits.

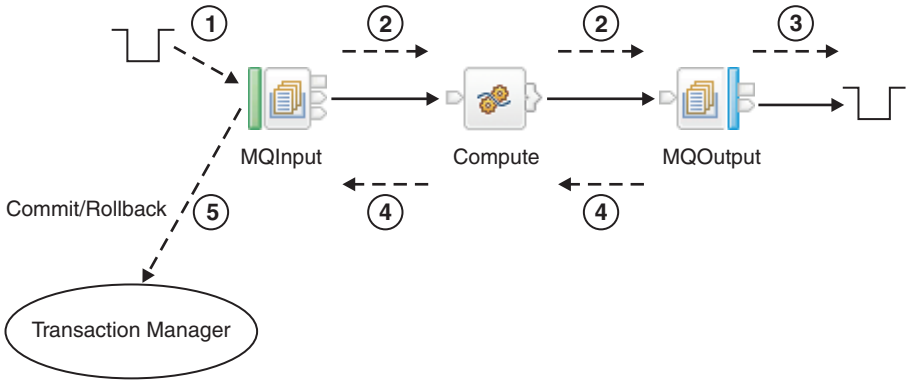
Before you begin

Before you start:

- Read "User exits" on page 3015.
- Read "Why use a user exit?" on page 2984

About this task

The following diagram illustrates how a user exit works. The numbered events are described after the diagram. The MQInput node is used as an example, but the function applies to all input nodes, including user-defined input nodes. Similarly, the Compute and MQOutput nodes could be replaced by any equivalent nodes.



1. (cciInputMessageCallback) The message is dequeued from the input source (read into the flow).

   Built-in nodes and user-defined nodes differ slightly in the way in which user exits are called. For built-in input nodes, the user exit is called as soon as possible after the data has been read from the external source. For user-defined input nodes, the user exit is called just before the node propagates the message.
2. (cciPropagatedMessageCallback) The message is propagated to the node for processing.
3. (cciOutputMessageCallback). A request message is sent to the output node's transport, and transport-specific destination information is written to WrittenDestination in the LocalEnvironment (for example, this information includes the queueName and msgId for an MQ message). The call is made when a node successfully puts a message to a transport, from either an output or a request node. The outputMessageEvent is called by built-in nodes only. The topic for each node that supports WrittenDestination information contains details about the data that it contains.
4. (cciNodeCompletionCallback) Node processing completes.
5. (cciTransactionEventCallback) The user exit is called after the transaction has completed, so that user exit processing is not part of that transaction. The user exit is invoked even if no transactional processing is completed by the flow.

   Where the message flow property `Commit Count` is greater than one, many-to-one ratios exist between events 1 and 5. This ratio also exists for some scenarios that are specific to the particular input node; for example, when an MQInput node is configured with the `Commit by Message Group` property selected.

You can write a user exit to track any number of these events. For each of these events, the following data is available to the user exit. All access is read-only, unless stated otherwise:

- The message is dequeued:
  - Bit stream
  - Input node

  - Environment tree (read and write)
- The message is propagated to the node:
  - Message tree (body element read and write)
  - LocalEnvironment tree (read and write)
  - Exception list
  - Environment tree (read and write)
  - Source node
  - Target node
- A message is sent to a transport:
  - Message tree (body element read and write)
  - LocalEnvironment tree (read and write)
  - Exception list
  - Environment tree (read and write)
  - Output or request node
- Node processing completes:
  - Message tree (body element read and write)
  - LocalEnvironment tree (read and write)
  - Exception list
  - Environment tree (read and write)
  - Node
  - Upstream node
  - Exception (if any)
- The end of the transaction:
  - Input node
  - Exception (if any)
  - Environment tree (read and write)

You can register multiple user exits, and, if they are registered, they are invoked in a defined order (see "**mqsichangeflowuserexits** command" on page 3751). Any changes that are made to the message assembly (the message and environment) by a user exit are visible to subsequent user exits.

When the user exit is invoked, it can query the following information:

- Message flow information:
  - Message flow name
  - Broker name
  - Broker's queue manager name
  - Execution group name
  - Message flow's commit count property
  - Message flow's commit interval property
  - Message flow's coordinated transaction property
- Node information:
  - Node name
  - Node type
  - Terminal name
  - Node properties

The user exit can also perform the following tasks:

- Navigate and read the message assembly (Message, LocalEnvironment, ExceptionList, Environment)
- Navigate and write the Message body, LocalEnvironment, and Environment tree

You can register the user exits on a dynamic basis, without needing to redeploy the configuration.

Related concepts:

"User exits" on page 3015
A user exit is user-provided custom software, written in C, to track data passing through message flows.

"Why use a user exit?" on page 2984
Use a user exit to intercept the progress of messages through message flows without having to redesign the message flow.

Related tasks:

"Implementing a user-defined exit" on page 3113
You can develop and deploy a user-defined exit.

"Deploying a user exit" on page 3116
Deploy your user exit to the broker.

"Developing a user exit" on page 3114
Develop a user exit by declaring it, implementing its behavior, then compiling it.

Related reference:

"**mqsichangeflowuserexits** command" on page 3751
Use the **mqsichangeflowuserexits** command to set the list of active or inactive user exits. A list of active and a list of inactive user exits is maintained for each execution group and message flow. The effective state of user exits for a given flow is decided when the flow starts.

"User-defined extensions" on page 6411
Reference material that supports the creation and management of your user-defined extensions.

Which type of user-defined extension to use

The types of user-defined extension that you can write include user-defined nodes, user-defined parsers, user-defined exits, and using a subflow as a user-defined node.

The following topics describe the different types of user-defined extension in more detail:

- "User-defined nodes" on page 2989
- "User-defined parsers" on page 3010
- "User exits" on page 3015
- "Using a subflow as a user-defined node" on page 3008

Related concepts:

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

"Why use a user-defined extension?" on page 2972
Use a user-defined node or parser when the built-in resources do not provide the required functions.

"Which language to use to implement a user-defined extension" on page 3016
You can use Java or C to implement a user-defined extension.

Related tasks:

"Implementing user-defined extensions" on page 3019
Create the resources for your user-defined extension. You can write user-defined nodes in C or Java, or you can use a subflow to create a node. You can write user-defined parsers and exits only in C.

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"User-defined extensions" on page 6411
Reference material that supports the creation and management of your user-defined extensions.

User-defined nodes:

User-defined nodes are the main mechanism for extending the functions of WebSphere Message Broker.

The most common uses for a user-defined node are:

- Calling an external system for which WebSphere Message Broker does not provide nodes
- Calling already defined program libraries that perform a transformation or calculation that is required in the design of a message flow
- Packaging a subflow

Before you consider constructing a user-defined node, make sure that no built-in node is available to perform the required actions. For example, you might have considered creating a user-defined node to perform the following tasks, but you can use a JavaCompute node instead:

- Allowing programming languages other than ESQL to be used for coding message flow functions
- Performance advantages in performing some actions in compiled code
- Complex functions that are not available in ESQL, such as the large number of classes provided in JS2E

The following topics describe the different types of user-defined node in more detail:

- "User-defined input nodes" on page 2990
- "User-defined message processing nodes" on page 2996
- "User-defined output nodes" on page 3006
- "Using a subflow as a user-defined node" on page 3008

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User exits" on page 3015
A user exit is user-provided custom software, written in C, to track data passing through message flows.

"Why use a user-defined extension?" on page 2972
Use a user-defined node or parser when the built-in resources do not provide the required functions.

"Which language to use to implement a user-defined extension" on page 3016
You can use Java or C to implement a user-defined extension.

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing user-defined extensions" on page 3019
Create the resources for your user-defined extension. You can write user-defined nodes in C or Java, or you can use a subflow to create a node. You can write user-defined parsers and exits only in C.

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"User-defined extensions" on page 6411
Reference material that supports the creation and management of your user-defined extensions.

User-defined input nodes:

A user-defined input node is an extension to the broker that provides a new input node in addition to the nodes supplied with the product.

You create user-defined input nodes by using the C or Java programming language, or from a subflow, to provide message input to a message flow from a message queue when you want your broker to accept messages from a transport protocol other than WebSphere MQ.

You can use a user-defined input node to receive data from an external data source and to allow that data to be processed in a broker. In this way, you can complement the primitive input node types provided by WebSphere Message Broker.

You cannot use a user-defined input node to provide the In terminal to a message subflow. If you want to provide the In terminal to a subflow, you must use the supplied Input node.

Before writing a user-defined node, make sure that you are familiar with the concepts that are introduced in "Why use a user-defined extension?" on page 2972 and "User-defined extensions in the runtime environment" on page 2980.

Related concepts:

"User-defined extensions in the runtime environment" on page 2980
Before you design and implement user-defined extensions, familiarize yourself with the core components. Ensure that you also understand the basic WebSphere Message Broker runtime architecture.

"Planning user-defined input nodes" on page 2995
Before you develop a user-defined input node, plan and design its content and purpose.

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"Why use a user-defined extension?" on page 2972
Use a user-defined node or parser when the built-in resources do not provide the

required functions.

"Using a subflow as a user-defined node" on page 3008
You can develop a user-defined node that packages a subflow from scratch, in the same way that you can create any other user-defined node that has its implementation based on Java, or base it on an existing subflow.

C user-defined input node life cycle:

A user-defined input node that is written in the C programming language progresses through several stages during its lifetime.

The stages of the life cycle are:
- Registration
- Instantiation
- Processing
- Destruction

Registration: During the registration phase, the broker discovers which resources are available and which LILs can provide them. In this instance, the resources available are nodes. The phase starts when an execution group starts. The LILs are loaded on the startup of an execution group, and the broker queries them to find out what resources they can provide.

A CciFactory structure is created during the registration phase, when the user-defined node calls **cniCreateNodeFactory**.

The following APIs are called by the broker during this stage:
- biGetMessageflowNodeFactory
- bipGetParserFactory

The following API is called by the user-defined node during this stage:
- cniCreateNodeFactory

Instantiation: An instance of a user-defined input node is created when the **mqsistart** command starts or restarts the execution group process, or when a message flow that is associated with the node is deployed.

The following APIs are called during this phase:
- **cniCreateNodeContext**. This API allocates memory for the instantiation of the user-defined node to hold the values for configured attributes. This API is called once for each message flow that is using the user-defined Input node.
- **cniCreateInputTerminal**. This API is invoked within the cniCreateNodeContext API, and is used to tell the broker what input terminals, if any, your user-defined input node has.

  Your user-defined input node only has input terminals if it is also acting as a message processing node. If this is the case, it is typically better to use a separate user-defined message processing node to perform the message processing, rather than combine both operations in one, more complex, node.
- **cniCreateOutputTerminal**. This API is invoked within the cniCreateNodeContext API, and is used to tell the broker what output terminals your user-defined input node has.
- **cniSetAttribute**. This API is called by the broker to establish the values for the configured attributes of the user-defined node.

During this phase, a CciTerminal structure is created when cniCreateTerminal is called.

Processing: The processing phase begins when the cniRun function is called by the broker. The broker uses the cniRun function to determine how to process a message, including determining the domain in which a message is defined, and invoking the relevant parser for that domain.

A thread is demanded from the message flow's thread pool, and is started in the run method of the input node. The thread connects to the broker's queue manager, and retains this connection for its lifetime. When a thread has been allocated, the node enters a message processing loop while it waits to receive a message. It remains in the loop until a message is received. If the message flow is configured to use multiple threads, thread dispatching is activated.

The message data can now be propagated downstream.

The following APIs are called by the broker during this phase:

- **cniRun**. This function is called by the broker to determine how to process the input message.
- **cniSetInputBuffer**. This function provides an input buffer, or tells the broker where the input buffer is, and associates it with a message object.

Destruction: A user-defined input node is destroyed when the message flow is redeployed, or when the **mqsistop** command is used to stop the execution group process. You can destroy the node by implementing the cniDeleteNodeContext function.

When a user-defined input node is destroyed in one of these ways, you should free any memory used by the node, and release any held resources, such as sockets.

The following APIs are called by the broker during this phase:

- cniDeleteNodeContext. This function is called by the broker to destroy the instance of the input node.

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

"Creating an input node in C" on page 3027
Create a user-defined input node in C to receive messages into a message flow.

Related reference:

"C language user-defined node API" on page 6416
Learn about the different types of call provided by the API.

Java user-defined input node life cycle:

A user-defined input node that is written in the Java programming language progresses through several stages during its lifetime.

The stages of the life cycle are:
- Registration
- Instantiation
- Processing
- Destruction

Registration: During the registration phase a user-defined input node written in Java makes itself known to the broker. The node is registered with the broker through the static getNodeName method. Whenever a broker starts, it loads all the relevant Java classes. The static method getNodeName is called at this point, and the broker registers the input node with the node name specified in the getNodeName method. If you do not specify a node name, the broker automatically creates a name for the node based on the package in which it is contained.

Using a static method here means that the method can be called by the broker before the node itself is instantiated.

Instantiation: A Java user-defined input node is instantiated when a broker deploys a message flow containing the user-defined input node. When the node is instantiated, the broker calls the constructor of the input node's class.

When a node is instantiated, any terminals that you have specified are created. A message processing node can have any number of input and output terminals associated with it. You must include the createInputTerminal and createOutputTerminal methods in your node constructor to declare these terminals.

To handle exceptions that are passed back to your input node, use createOutputTerminal to create a catch terminal for your input node. When the input node catches an error, the catch terminal processes it in the same way as an MQInput node would. You can allow most exceptions, such as exceptions that are caused by deployment problems, to pass back to the broker, and the broker will warn the user of any possible configuration errors.

As a minimum, your constructor class needs only to create these output terminals on your input node. However, if you need to initialize attribute values, such as defining the parser that will initially parse a message passed from the input node, you should also include that code at this point in your input node.

Processing: Message processing for an input node begins when the broker calls the run method. The run method creates the input message, and should contain the processing function for the input node.

The run method is defined in MbInputNodeInterface, which is the interface used in a user-defined node that defines it as an input node. You must include a run method in your node. If you do not include a run method in your user-defined input node, the node source code will not compile.

When a message flow containing a user-defined input node is deployed successfully, the broker calls the node's run implementation method, and continues to call this method while it waits for messages to process.

When a message flow starts, a single thread is dispatched by the broker, and is called into the input node's run method. If the dispatchThread() method is called, further threads can also be created in the same run method. These new threads immediately call into the input node's run method, and can be treated the same as the original thread. The number of new threads that can be created is defined by the additionalInstances property. Make sure that threads are dispatched after a message has been created, and before it is propagated, to ensure that only one thread at a time is waiting for a new message.

The user-defined input node can choose a different threading model and is responsible for implementing the chosen model. If the input node supports the additionalInstances property, and dispatchThread() is called, the code must be fully re-entrant, and any functions that are invoked by the node should also be re-entrant. If the input node forces single threading, that is, it does not call dispatchThread(), the node documentation must state that setting the additionalInstances property has no effect on the input node.

For more information on the threading model for user-defined input nodes, see "Threading considerations for user-defined extensions" on page 2978.

Destruction: A Java user-defined input node is destroyed when the node is deleted or the broker is shut down. You do not need to include anything in your code that specifies the node should be physically deleted, because this can be handled by the garbage collector.

However, if you want notification that a node is about to be deleted, you can use the **onDelete** method. You might want to do this if there are resources that you want to delete, other than those that will be garbage collected. For example, if you have opened a socket, this will not be properly closed when the node is automatically deleted. You can include this instruction in your **onDelete** method to ensure that the socket is closed properly.

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

"Threading considerations for user-defined extensions" on page 2978
Message processing nodes and parsers must work in a multi-instance, multithreaded environment. Many node objects or parser objects are available, each with several syntax elements, and many threads can be executing methods on these objects.

Related tasks:

"Creating an input node in Java" on page 3055
An input node is used to receive a message into a message flow, typically from a source that is not supported by the built-in input nodes.

"Creating a message processing or output node in Java" on page 3062
A message processing node is used to process a message, and an output node is used to output a message as a bit stream.

Related information:

Java user-defined extensions API

Planning user-defined input nodes:

Before you develop a user-defined input node, plan and design its content and purpose.

Analysis: Before you develop a user-defined input node, ask yourself the following questions:

- Do you need to create a custom input node?

  You must include at least one input node in a message flow. Which one you choose depends on the source of the input messages:

  – If the messages arrive at the broker on a WebSphere MQ queue, use the MQInput node.
  – If SOAP messages are received over HTTP, use the SOAPInput node.
  – If other messages are received over HTTP, use the HTTPInput node.
  – If the messages are received from a JMS source, use the JMSInput node.
  – If the messages are received from an EIS, use the PeopleSoftInput, SAPInput, or SiebelInput node.
  – If the messages are retrieved from files, use the FileInput node.
  – If the message source is any other, you must use a user-defined input node.

  For information about using more than one input node in a message flow, see "Using more than one input node" on page 1473.
- To successfully input the data concerned, does the input node have to interface with vendor software? If so, does the API that enables access to this software break your threading model?
- Do you need a new user-defined parser to interpret the body (payload) of the message generated by this input node, or can it be parsed by a standard built-in parser?
- Do you need the user-defined input node to operate the message flow instance in which it resides under transactional control as a globally-coordinated transaction?
- Do you need the new user-defined input node to offer configuration options?
- Do you need messages propagated by this input node to be processed by the following primitives?

  – All primitive output nodes
  – ResetContentDescriptor nodes

Design considerations: Before developing and implementing your input node, decide on the following factors:

- Which message parser will initially parse the input message.
- Whether to override the default message parser attribute values for this input node.
- Which threading model is appropriate for the input node.
- What kind of message processing and transaction support will the node support.
- Which configuration attributes required by the input node should be externalized for alteration by the message flow designer.
- What optional node APIs will the user-defined node provide.
- How you will handle general development issues:

  – "Threading considerations for user-defined extensions" on page 2978
  – "Storage management in user-defined nodes" on page 2976

- "String handling in user-defined nodes" on page 2977
- "Errors and exception handling" on page 2973
- Expected message formats for primitive nodes that expect specific header folders.

When you design nodes to be used as extensions to WebSphere Event Broker, the following restrictions apply:

- User-defined input nodes can support only XML, BLOB, and the WebSphere MQ parsers, because the MRM parser is not shipped with WebSphere Event Broker and user-defined parsers are not supported.
- User-defined nodes must not allow users to evaluate user ESQL code, because the use of ESQL in WebSphere Event Broker is not supported. For example, nodes that expose the input to **`MbSQLStatement`** as a node attribute are effectively emulating a Compute node.

Related concepts:

"Parsers" on page 1072
A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from the internal message tree representation.

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Creating an input node in Java" on page 3055
An input node is used to receive a message into a message flow, typically from a source that is not supported by the built-in input nodes.

"Creating an input node in C" on page 3027
Create a user-defined input node in C to receive messages into a message flow.

Related reference:

"C language user-defined node API" on page 6416
Learn about the different types of call provided by the API.

Related information:

Java user-defined extensions API

User-defined message processing nodes:

A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

You might consider the use of a user-defined message processing node in the following situations:

- Your messages need transformations that the built-in nodes do not provide. For example, you might need a currency converter node.
- You want to reuse a subflow.
- You want to hide a message flow implementation by packaging it in a user-defined node.

Combine your user-defined nodes with the built-in nodes to create message flows that meet your exact business requirements.

Related concepts:

"Message flow nodes" on page 1024
A message flow node is a processing step in a message flow. It can be a built-in node, a user-defined node, or a subflow node.

"C user-defined message processing nodes life cycle"
A user-defined message processing node for the C programming language goes through various stages.

"Java user-defined message processing nodes life cycle" on page 3000
During the lifecycle of the user-defined nodes that you create in Java, objects are created and destroyed, and different methods and classes called.

"Planning user-defined message processing nodes" on page 3002
Plan how to write your message processing node or output node, and how to navigate the message within the node.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

"Creating a message processing or output node in Java" on page 3062
A message processing node is used to process a message, and an output node is used to output a message as a bit stream.

"Creating a user-defined node from a subflow" on page 3076
Create user-defined nodes either from scratch, or by using an existing subflow.

Related reference:

"C language user-defined node API" on page 6416
Learn about the different types of call provided by the API.

Related information:

Java user-defined extensions API

C user-defined message processing nodes life cycle:

A user-defined message processing node for the C programming language goes through various stages.

This topic covers the objects that are created and destroyed, and the implementation functions and classes that are called in the following stages:

- Registration
- Instantiation
- Processing
- Destruction

The information in this topic applies to both output nodes and message processing nodes. Both of these node types can be considered together, because although a message processing node is typically used to process a message, and an output node is used to provide an output in the form of a bit stream, you can use either type of node to perform either of these functions.

Registration: A user-defined message processing node is registered with the broker when the LIL that contains the node has been loaded and initialized by the operating system.

The broker calls **bipGetMessageflowNodeFactory** to establish the function of the LIL, and how the LIL should be called.

The **bipGetMessageflowNodeFactory** function in turn calls the **cniCreateNodeFactory** function, which returns a factory or group name for all of the nodes that are supported by your LIL.

The LIL should then call the utility function **cniDefineNodeClass** to pass both the name of each node and a virtual function table of the function pointers of the implementation functions.

Instantiation: During the instantiation phase, an instance of a user-defined message processing node is created. The phase starts when the broker creates a message flow and calls the **cniCreateNodeContext** function for each instantiation of the user-defined node in that message flow. The **cniCreateNodeContext** function is that which is specified in the **iFpCreateNodeContext** field of the CNI_VFT struct passed to **cniDefineNodeClass** for that node type. This function should allocate the resources required for that node, including memory such that the instantiation of the user-defined node can hold the values for the configured attributes.

The broker will create a node instance and call **cniCreateNodeContext** on the following occasions:

- Message flow is created:
  - Broker is being started (user has run mqsistart). Any message flows previously deployed are re-created when the broker starts.
  - Execution group is being reloaded (user has run mqsireload). Any message flows that have been deployed previously are re-created when the execution group reloads.
  - A severe error has occurred within the execution group which results in the execution group being restarted.
- Message flow is redeployed. When a message flow is changed and redeployed, the broker processes redeploy by deleting all nodes in the flow and then re-creating them with the new configuration.

Note: A message flow is not created when starting an execution group. Stopping an execution group simply stops all flows and does not delete the flow or bring the process down. Restarting an execution group, starts the message flows but does not re-create the message flows.

Within **cniCreateContext**, the user-defined extension calls the two functions **cniCreateInputTerminal** and **cniCreateOutputTerminal** in order to establish what input and output terminals the message processing node has.

Processing: During the processing phase of the life cycle of a user-defined message processing node, the message is transformed in some way, when some processing operation takes place on the input message.

When the broker retrieves a message from the queue and that message arrives at the input terminal of your user-defined node, the broker calls the implementation function **cniEvaluate**. This function is used to decide what to do with the message.

You can use a range of node utility functions in your user-defined message processing node to perform a range of message processing functions, such as accessing the message data, accessing ESQL, transforming a message object, and

propagating a message. You should include the node utility functions you are going to use to process the message within the **cniEvaluate** function.

This interface does not automatically generate a properties subtree for a message. It is not a requirement for a message to have a properties subtree, although you might find it useful to create one to provide a consistent message tree structure regardless of input node. If you want a properties subtree to be created in a message, and you are also using a user-defined input node, you must do this yourself

Destruction: When a user-defined message processing node has processed a message, you should ensure that it is destroyed, to release any system resources that it used, and to release any data areas specific to the node instance, such as context, that were acquired when the message was constructed or processed.

An instance of a user-defined message processing node is destroyed when the broker calls the **cniDeleteNodeContext** function.

The broker calls **cniDeleteNodeContext** when the instance of the node is deleted. The following events can cause a node to be deleted:

- Controlled termination of the execution group process:
  - Broker is being stopped (user has run mqsistop)
  - Execution group is being reloaded (user has run mqsireload)
  - A severe error has occurred within the execution group, which results in the execution group being restarted.

  Note: This does NOT include stopping an execution group. Stopping an execution group simply stops all flows, and does not delete the flow or bring the process down.
- Message flow is deleted.
- Message flow is redeployed. When a message flow is changed and redeployed, the broker processes redeploy by deleting all nodes in the flow and then re-creating them with the new configuration.

Related concepts:

"Message flow nodes" on page 1024
A message flow node is a processing step in a message flow. It can be a built-in node, a user-defined node, or a subflow node.

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"Planning user-defined message processing nodes" on page 3002
Plan how to write your message processing node or output node, and how to navigate the message within the node.

"Java user-defined message processing nodes life cycle" on page 3000
During the lifecycle of the user-defined nodes that you create in Java, objects are created and destroyed, and different methods and classes called.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

"Creating a message processing or output node in Java" on page 3062
A message processing node is used to process a message, and an output node is used to output a message as a bit stream.

Related reference:

"cniCreateInputTerminal" on page 6444
Use this function to create an input terminal on an instance of a node object and return the address of the terminal object that was created.

"cniCreateNodeFactory" on page 6449
Use this function to create a node factory in the broker. A single instance of the named message flow node factory is created.

"cniDeleteNodeContext" on page 6454
This function deletes any context for an instance of a user-defined node object. It is called by the broker whenever an instance of a node object is destroyed, when a message flow is deleted, or when a configuration is redeployed.

"cniDefineNodeClass" on page 6451
Use this function to define a node class, as specified by the *name* parameter, which is supported by the node factory specified as the*factoryObject* parameter.

"cniEvaluate" on page 6475
This function performs node processing. The broker calls this function when a message is received on one of the input terminals of an instance of a node object.

Java user-defined message processing nodes life cycle:

During the lifecycle of the user-defined nodes that you create in Java, objects are created and destroyed, and different methods and classes called.

Every node goes through the following stages:

- Registration
- Instantiation
- Processing
- Destruction

The information here applies to both output nodes and message processing nodes. Both of these node types can be considered together, because although a message processing node is typically used to process a message, and an output node is used to provide an output, in the form of a bit stream, from a message, you can use both types of node to perform either of these functions.

Registration: The registration phase occurs when a user-defined message processing node that is written in Java contacts the broker, or registers with the broker.

Whenever a broker starts, it loads all relevant LIL files and Java classes. To ensure that a message processing node is registered with the broker, you must provide the broker with a class that implements the MbNodeInterface interface and is contained in the classpath used by the broker.

Instantiation: A Java user-defined message processing node is instantiated when a broker deploys a message flow that contains the user-defined message processing node. When the node is instantiated, the constructor of the message processing node class is called.

When a node is instantiated, all terminals that you have specified are created. A message processing node can have an unlimited number of input and output terminals associated with it. You must include the createInputTerminal and createOutputTerminal methods in your node constructor to declare these terminals.

Output terminals include out, failure, and catch terminals. Use the createOutputTerminal class within the node class constructor in order to create as many output terminals as you require.

As a minimum, you must create only these output terminals by using your constructor class. However, if you need to initialize attribute values, you must also include that code at this point in your message processing node.

If you want to handle exceptions that are passed back to your message processing node, it is good practice to do this by creating a failure terminal for your user-defined message processing node, by using the createOutputTerminal method. It is sensible to use the failure terminal for this process because that is the terminal to which errors are propagated.

Make sure that all exceptions that are caught by the message processing node are dealt with properly. If you do not include a failure terminal, the message processing node does not attempt to handle the exception. If your message flow does not contain a method of exception handling, all exceptions thrown are passed back to the input node, where the input node deals with the exceptions.

If you do catch exceptions, make sure that you rethrow all exceptions that the message processing node cannot deal with. This action causes the exception to be passed back to the input node for handling; for example, when you want to rollback a transaction.

Processing: During the processing phase of the life cycle of a user-defined message processing node, the message processing node takes the logical hierarchy of the message and processes it in some way.

Destruction: A Java user-defined message processing node is destroyed when the node is deleted, or the broker is shut down. You do not have to include anything in your code to specify that the node is physically deleted, because this process can be handled by the garbage collector.

However, if you want notification that a node is about to be deleted, you can use the onDelete method. You might want to receive notification if the node has resources that you want to delete, other than those that will be garbage collected. For example, if you have opened a socket, it is not properly closed when the node is automatically deleted. You can include this instruction in your onDelete method to ensure that the socket is closed properly.

Related concepts:

"Message flow nodes" on page 1024
A message flow node is a processing step in a message flow. It can be a built-in node, a user-defined node, or a subflow node.

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"Planning user-defined message processing nodes" on page 3002
Plan how to write your message processing node or output node, and how to navigate the message within the node.

"C user-defined message processing nodes life cycle" on page 2997
A user-defined message processing node for the C programming language goes through various stages.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

"Creating a message processing or output node in Java" on page 3062
A message processing node is used to process a message, and an output node is used to output a message as a bit stream.

Related reference:

"cniCreateInputTerminal" on page 6444
Use this function to create an input terminal on an instance of a node object and return the address of the terminal object that was created.

"cniCreateNodeFactory" on page 6449
Use this function to create a node factory in the broker. A single instance of the named message flow node factory is created.

"cniDeleteNodeContext" on page 6454
This function deletes any context for an instance of a user-defined node object. It is called by the broker whenever an instance of a node object is destroyed, when a message flow is deleted, or when a configuration is redeployed.

"cniDefineNodeClass" on page 6451
Use this function to define a node class, as specified by the *name* parameter, which is supported by the node factory specified as the*factoryObject* parameter.

"cniEvaluate" on page 6475
This function performs node processing. The broker calls this function when a message is received on one of the input terminals of an instance of a node object.

Planning user-defined message processing nodes:

Plan how to write your message processing node or output node, and how to navigate the message within the node.

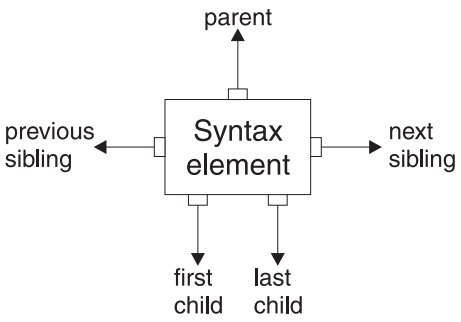
Design factors: Before developing and implementing your message processing node, consider the following points:

- Which parser will parse messages.
- Whether to override the default message parser attribute values for this message processing node.
- What is the appropriate threading model for the message processing node.
- How to implement the end of message processing and transaction support that the node must support.
- What configuration properties required by the message processing node must be externalized for alteration by the message flow designer.
- What optional node APIs will the user-defined node provide.
- General development issues:
  - "Threading considerations for user-defined extensions" on page 2978
  - "Storage management in user-defined nodes" on page 2976
  - "String handling in user-defined nodes" on page 2977
  - "Errors and exception handling" on page 2973
  - Expected message formats for built-in nodes that expect specific header folders, see "Element definitions for message parsers" on page 4237

Syntax element navigation: The broker provides functions that your node can call to traverse the tree representation of the message, as well as functions and methods that support navigation from the current element to other elements:

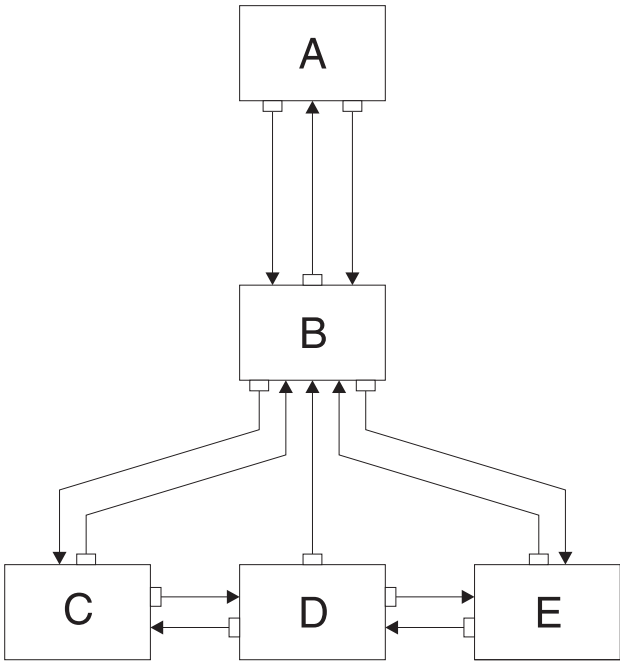
- Parent
- First child
- Last child
- Previous (or left) sibling
- Next (or right) sibling

These relationships are shown in the following diagram.



Other functions and methods support the manipulation of the elements themselves, with functions and methods to create elements, to set or query their values, to insert new elements into the tree, and to remove elements from the tree. See “C node utility functions” on page 6419 and “C parser utility functions” on page 6539, or the Javadoc information for more details.

The following diagram describes a simple syntax element tree that shows a full range of interconnections between the elements.

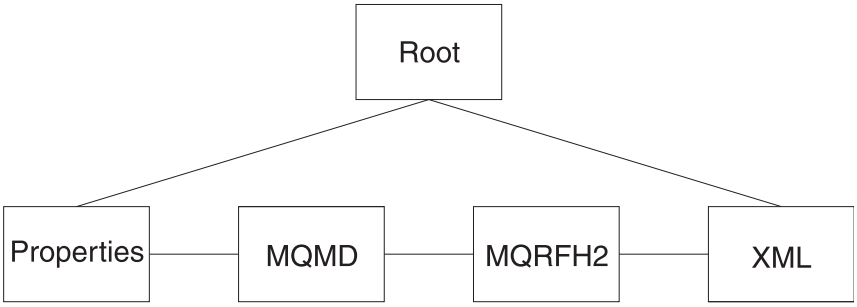


The element **A** is the root element of the tree. It has no parent because it is the root. It has a first child of element **B**. Because **A** has no other children, element **B** is also the last child of **A**.

Element **B** has three children: elements **C**, **D**, and **E**. Element **C** is the first child of **B**; element **E** is the last child of **B**.

Element **C** has two siblings: elements **D** and **E**. The next sibling of element **C** is element **D**. The next sibling of element **D** is element **E**. The previous sibling of element **E** is element **D**. The previous sibling of element **D** is element **C**.

The following diagram shows the first generation of syntax elements of a typical WebSphere MQ message received by a broker. (Not all messages have an MQRFH2 header.)



These elements at the first generation are often referred to as folders, in which syntax elements that represent message headers and message content data are stored. In this example, the first child of root is the Properties folder. The next sibling of Properties is the folder for the MQMD header. The next sibling is the folder for the MQRFH2 header. The last folder represents the message content, which (in this example) is an XML message.

The previous figure includes an MQMD and an MQRFH2 header. All messages that are received by a processing node that handles WebSphere MQ include an MQMD header; a number of other headers can also be included.

Navigating an XML message: Consider the following XML message:

```
<Business>
  <Product type='messaging'></Product>
  <Company>
    <Title>IBM</Title>
    <Location>Hursley</Location>
    <Department>WebSphere MQ</Department>
  </Company>
</Business>
```

In this example, the elements are of the following types:

Name element
Business, Product, Company, Title, Location, Department

Value element
IBM, Hursley, WebSphere MQ

Name-value element
type='messaging'

Use supplied node utility functions and methods (or the similar parser utility functions) to navigate through a message. Using the XML message shown, you must call cniRootElement first, with the message received by the node as input to this function. In Java, you must call getRootElement on the incoming MbMessage

object. This call returns an MbElement that represents the root of the element. Do not modify this root element in the user-defined node.

The figure of the first generation of the syntax elements of a typical message that is received by the broker, shows that the last child of the root element is the folder containing the XML parse tree. Navigate to this folder by calling cniLastChild (with the output of the previous call as input to this function) in a C node, or by calling the method getLastChild on the root element, in a Java node.

Only one element (`<Business>`) is at the top level of the message, therefore call cniFirstChild (in C) or getFirstChild (in Java) to move to this point in the tree. Use cniElementType or getType to get its type (which is `name`), followed by cniElementName or getName to return the name itself (`Business`).

The element `<Business>` has two children, `<Product>` and `<Company>`. Use cniFirstChild or getFirstChild followed by cniNextSibling or getNextSibling to navigate to each child in turn.

The element `<Product>` has an attribute (`type='messaging'`), which is a child element. Use cniFirstChild or getFirstChild to navigate to this element, and cniElementType or getType to return its type (which is `name-value`). Use cniElementName or getName to get the name. To get the value, call cniElementValueType to return the type, followed by the appropriate function in the cniElementValue group, in this example it is cniElementCharacterValue. In Java use the method getValue, which returns a Java object representing the element value.

The element `<Company>` has three children, each one having a child that is a value element (`IBM`, `Hursley`, and `WebSphere MQ`). Use the functions already described to navigate to them and access their values.

Other functions are available to copy the element tree (or part of it). The copy can then be modified by adding or removing elements, and changing their names and values, to create an output message. See “C node utility functions” on page 6419 and “C parser utility functions” on page 6539, or the Java user-defined node API, for more information.

Related concepts:

“Why use a user-defined extension?” on page 2972
Use a user-defined node or parser when the built-in resources do not provide the required functions.

“Message flow nodes” on page 1024
A message flow node is a processing step in a message flow. It can be a built-in node, a user-defined node, or a subflow node.

“User-defined message processing nodes” on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

“C user-defined message processing nodes life cycle” on page 2997
A user-defined message processing node for the C programming language goes through various stages.

“Java user-defined message processing nodes life cycle” on page 3000
During the lifecycle of the user-defined nodes that you create in Java, objects are created and destroyed, and different methods and classes called.

“Planning user-defined output nodes” on page 3008
A user-defined output node generates an output bit stream from a message tree.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

"Creating a message processing or output node in Java" on page 3062
A message processing node is used to process a message, and an output node is used to output a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

Related information:

Java user-defined extensions API

User-defined output nodes:

A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

If you want your message flow to send messages by using a protocol that is not supported by WebSphere Message Broker, you can create your own output node.

WebSphere Message Broker provides the following output nodes:

- MQOutput - delivers an output message from a message flow to a WebSphere MQ queue
- MQReply - sends a response to the originator of the input message.
- SCADAOutput - sends a message to a client connecting using the MQIsdp protocol
- Publication - filters output messages from a message flow and transmit them to subscribers who have registered an interest in a particular set of topics.
- JMSOutput - sends a message to a JMS destination
- EmailOutput - sends an email message to one or more recipients
- FileOutput - writes a message to a file

If the target application expects to receive message in any other way, you must use a user-defined output node.

User-defined output nodes can be considered together with user-defined message processing nodes. Conceptually, these two kinds of user-defined nodes are the same. Although a message processing node is typically used to process a message, and an output node is used to provide an output, in the form of a bit stream, from a message, you construct output nodes and message processing nodes in a similar way, and you can use either type of node to perform either function.

You can also create a user-defined output node from a subflow, see "Using a subflow as a user-defined node" on page 3008

For more information about user-defined output nodes, read the topics that cover user-defined message processing nodes, see "User-defined message processing nodes" on page 2996.

Related concepts:

"User-defined output node life cycle"
The life cycle of a user-defined output node follows the same pattern as the life cycle of user-defined message processing nodes.

"Planning user-defined output nodes" on page 3008
A user-defined output node generates an output bit stream from a message tree.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

"Creating a message processing or output node in Java" on page 3062
A message processing node is used to process a message, and an output node is used to output a message as a bit stream.

Related reference:

"User-defined extensions" on page 6411
Reference material that supports the creation and management of your user-defined extensions.

User-defined output node life cycle:

The life cycle of a user-defined output node follows the same pattern as the life cycle of user-defined message processing nodes.

The following topics describe the life cycle of user-defined message processing nodes; read the topic that corresponds to your type of output node:

- "C user-defined message processing nodes life cycle" on page 2997
- "Java user-defined message processing nodes life cycle" on page 3000

Although a message processing node is typically used to process a message, and an output node is used to provide an output in the form of a bit stream, you can use both types of node to perform either of these functions.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"C user-defined message processing nodes life cycle" on page 2997
A user-defined message processing node for the C programming language goes through various stages.

"Java user-defined message processing nodes life cycle" on page 3000
During the lifecycle of the user-defined nodes that you create in Java, objects are created and destroyed, and different methods and classes called.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

"Creating a message processing or output node in Java" on page 3062
A message processing node is used to process a message, and an output node is used to output a message as a bit stream.

Related reference:

"Output node" on page 4626
Use the Output node as an out terminal for an embedded message flow (a subflow).

"User-defined extensions" on page 6411
Reference material that supports the creation and management of your user-defined extensions.

Planning user-defined output nodes:

A user-defined output node generates an output bit stream from a message tree.

Optionally, you can connect the node to another node and propagate the message tree for further processing. User-defined output nodes and message processing nodes are, therefore, structured in the same way. All relevant information for output nodes is included in "Planning user-defined message processing nodes" on page 3002.

Related concepts:

"Planning user-defined message processing nodes" on page 3002
Plan how to write your message processing node or output node, and how to navigate the message within the node.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

"Creating a message processing or output node in Java" on page 3062
A message processing node is used to process a message, and an output node is used to output a message as a bit stream.

Related reference:

"Output node" on page 4626
Use the Output node as an out terminal for an embedded message flow (a subflow).

"C language user-defined node API" on page 6416
Learn about the different types of call provided by the API.

Related information:

Java user-defined extensions API

Using a subflow as a user-defined node:

You can develop a user-defined node that packages a subflow from scratch, in the same way that you can create any other user-defined node that has its implementation based on Java, or base it on an existing subflow.

The project that contains user-defined nodes can be exported as a plug-in that is installed in the development environment of the user. The nodes that are packaged in the plug-in are displayed in the palette in the Message Flow editor, and can be used in a message flow in the same way as a built-in node.

Packaging a subflow as a user-defined node provides all the benefits of a subflow, such as reusability and maintainability, as well as the following benefits:

- The user-defined node can be distributed to other developers as a plug-in.

- The user-defined node hides the implementation details of the subflow from developers who reuse the subflow.
- The user-defined node prevents developers who reuse the subflow from modifying it.
- The subflow is displayed in the palette in the Message Flow editor.

Limitations

- All the flow resources (Maps, ESQL, XSL, or other external resources), except Java code and message sets, that are referenced in the subflow, must be located in the user-defined node project.
- A user-defined node can reference another user-defined node in the same or different user-defined node project, but it must not reference anything from a regular message flow project.
- The user-defined node project can have references to other projects, such as message set and Java projects.
- If the user-defined node references a message set, you must deploy the message set to the runtime separately. You can copy the message set to your workspace and deploy it through the broker archive (BAR) file.
- A subflow implementation of a user-defined node can contain other subflows, but all the subflows must be contained in the user-defined node project.
- Promoted properties from the nodes within the subflow are supported. Configurable promoted properties from nodes in the subflow are displayed as configurable node properties in the Broker Archive editor.
- User-defined properties (UDP) on the subflow are supported. If you create multiple instances of user-defined nodes in your flow, each type of user-defined property that you define must have the same value in each instance.
- You can use the same subflow more than once to construct a flow of your own.
- You can use the following node types that have named correlators to create a user-defined node, but you must not use more than one instance of the user-defined node within a flow, an execution group, or a broker. For example, you cannot use asynchronous request and asynchronous response nodes, aggregate nodes, TimeoutControl and TimeoutNotification pairs of nodes, or label nodes.

  If you do use one of these node types, the resultant message flow is invalid. If you deploy flows that contain asynchronous request, asynchronous response, or label nodes, you receive an error message. However, you do not receive an error message if you deploy flows that contain TimeoutControl, TimeoutNotification, or aggregate nodes.
- Resources in the plug-in space are visible to all projects in the workspace. Keep user-defined nodes and their associated flows, maps, ESQL, and other similar resources, in appropriately named broker schemas. Do not put such resources in a default schema, or schemas with special names, for example, `mqsi`.

Related concepts:

"User-defined nodes" on page 2989
User-defined nodes are the main mechanism for extending the functions of WebSphere Message Broker.

Related tasks:

"Creating a user-defined node from a subflow" on page 3076
Create user-defined nodes either from scratch, or by using an existing subflow.

"Choosing the location of a user-defined node in the palette" on page 3092
Use the Palette editor to edit palette-specific information for user-defined nodes,

add and delete separators, and rearrange user-defined nodes in the palette.

"Testing a subflow user-defined node project" on page 3097
Test how the user-defined node is displayed, and how it behaves in your environment.

"Packaging and distributing a user-defined node project" on page 3121
Export the user-defined node project to make it available for other users.

"Installing a user-defined node" on page 1496
Develop message flows that use a user-defined node.

User-defined parsers:

A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

Create a user-defined parser when the WebSphere Message Broker parsers are not sufficient to parse user-defined messages.

Do not use user-defined parsers to provide connectivity or transformation functions. In most cases, the MRM or other IBM supplied parsers are capable of passing most standard type of format. You can also parse a message and construct a message tree in a user-defined node without the need to write a parser. For example, a user-defined node that reads emails from a POP3 server can parse the email and construct a message tree without the need to write a user-defined parser.

If the parser is going to be used only in a user-defined node, you do not need to use a user-defined parser. However, consider a user-defined parser if the parser will be called from other message flow nodes.

Related concepts:

"User-defined parser life cycle" on page 3011
Various stages exist in the life of a user-defined message flow parser.

"Planning user-defined parsers" on page 3013
Read about the concepts that you should consider before you develop a user-defined parser.

"Specific types used by parsers" on page 3015
Specific types are used when a parser needs additional information that is associated with some or all the elements in a tree in order to generate the bit stream.

"Parsers" on page 1072
A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from the internal message tree representation.

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

User-defined parser life cycle:

Various stages exist in the life of a user-defined message flow parser.

These stages are involved:
- Registration
- Instantiation
- Processing
- Destruction

This topic describes the interactions that take place between WebSphere Message Broker components when you run a user-defined parser. It explains each stage in terms of the events that start each stage, and the events that occur during and after each stage, and the APIs that are called. Understanding the concepts here help you to design and develop your parser more effectively.

Registration: The first phase in the user-defined parser's life cycle is the registration phase. The purpose of the registration phase is to register the user-defined parser with the broker. This phase starts when the execution group starts.

Instantiation: The parser is created during the instantiation phase of the parser life cycle. When an input message is received, or an output message is built in a Compute node, the relevant parser is identified, and parser requirements are taken from the message header, such as the MQMD. The broker starts and loads the Loadable Implementation Library (LIL) and the parser factory. Before the cpiCreateContext function is called, the broker creates a name element as the effective root element for the parser. However, this element is not named. The parser should name this element in the cpiSetElementName function. The execution group process creates an instance of the parser, and the broker makes a call to cpiCreateContext to allow the parser object to acquire the appropriate section of the message.

The broker then makes a call to cpiParseBuffer. cpiParseBuffer performs any necessary initialization, and returns the length of the message content that the parser is taking ownership of. The parser assesses how much of the message data to parse, and claims the appropriate number of bytes.

Whenever an instance of a user-defined parser object is created, the context creation implementation function cpiCreateContext is also invoked by the broker. This call allows the parser to allocate instance data associated with the parser. A cpiDeleteContext function to delete the context of the parser object is also required.

Processing: During the processing phase, the parser manipulates, alters, and references elements within the message object. The message flow processing phase begins when any message processing activity occurs, such as navigation, that requires access to an element within a message that does not exist in the broker's internal model representation of the message concerned.

During the message flow processing phase, the parser is invoked in response to attempts to navigate into the message tree. The parser examines the buffer that was allocated when cpiParseBuffer was called, and creates any necessary message elements.

The parser can then navigate through the message elements, using any or all of the following parser implementation functions:

- cpiParseFirstChild
- cpiParseLastChild
- cpiParsePreviousSibling
- cpiParseNextSibling

These functions are invoked when any form of navigation is made (such as a filter expression that specifies a message field) into the part of the syntax element tree that logically represents the data for a message format supported by a user-defined parser. This navigation occurs when an operation within the broker requires a syntax element tree to be built or extended.

Consider the following points when deciding how best to navigate the syntax element tree:

- A Syntax element has five pointers to its parents, siblings, and first and last children, so that a finite set of navigations is available.
- The same internal classes are used to perform all of these navigations.
- The parser does not control the navigation. The ESQL or a user-defined node makes the decision about the direction in which to navigate, and the order in which the navigational parser implementation functions are invoked. The user-defined parser has no control over the direction and order, and needs to respond correctly to the chosen navigation scheme; for example, parsing right to left, as well as left to right.
- When writing a user-defined parser, place the parser code in a parseNextItem function. This function should build the syntax element tree one element at a time, setting names, values and complete flags appropriately. How you implement this function depends on the nature of the bit stream to be parsed. The supplied sample parser demonstrates this behavior.

When the parser has finished parsing the relevant parts of the syntax element tree, it calls cpiWriteBuffer. This function appends its portion of the syntax element tree to the bit stream in the message buffer that is associated with the parser object, and creates the output message.

Destruction: The Destruction phase is the final phase in the user-defined parser life cycle. When the parser has written its portion of the syntax element tree to the bit stream and created the output message, the system resources that were created by the broker for the parser to use need to be released.

The destruction phase begins when the **mqsistop** command is used to stop the execution process.

Related concepts:

"Parsers" on page 1072
A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from the internal message tree representation.

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to

extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"C language user-defined parser API" on page 6538
The C language user-defined parser API consists of two complementary sets of functions that you can use to define the function of your parser.

"**mqsistop** command" on page 3972
Use the **mqsistop** command to stop the specified component.

Planning user-defined parsers:

Read about the concepts that you should consider before you develop a user-defined parser.

When you have considered the information provided here, and are ready to develop your own parser, use the instructions in "Implementing a user-defined parser" on page 3099 to construct your parser.

Analysis: Before you start to create your own parser, be clear about its purpose. You can perform most tasks using the functions that are provided with WebSphere Message Broker, so you might not need to create a user-defined parser for your particular task.

Before you construct and implement a user-defined parser, consider the following questions:

- Do you need to create a user-defined parser?

  If the available parsers in WebSphere Message Broker are not appropriate for your needs, define your own parser to parse internal, customer-specific, or generic commercial message formats.
- Does WebSphere Message Broker already provide a parser for the domain or message header?

  See "Parsers" on page 1072 for details of message domains for which the supplied parsers can accept input messages, and message headers with which the supplied parsers can work.
- Does the syntax of the in-house or commercial message dictate a format that can be parsed?
- To parse the message successfully, does the parser need to interact with vendor software? If so, does the API that enables access to this software break your threading model?
- Do you need to process multi-part, multi-format messages?

  WebSphere Message Broker does not support multi-part, multi-format messages. A multi-part MRM message must consist of messages that are all in the same format.
- What type of parsing strategy will provide best performance?

  WebSphere Message Broker supports partial parsing, which allows your parser to parse only relevant fields in a message. Using partial parsing can save system resources.

Partial and full parsing: WebSphere Message Broker supports *partial parsing*. If an individual message contains hundreds or even thousands of individual fields, the parsing operation requires considerable memory and processor resources to complete. An individual message flow might reference only a few of these fields, or none at all, so it is inefficient to parse every input message completely. For this reason, WebSphere Message Broker allows parsing of messages on an as-needed basis. (This ability does not prevent a parser from processing the entire message in one step, and some parsers are written to process the entire message in this way.)

Each syntax element in a logical message has two bits that indicate whether all the elements on either side of an element are complete, and whether its children are complete. Parsing is typically completed in a bottom-to-top, left-to-right manner. When a parser has parsed the siblings of a particular element that precede the given element and the first child, it sets the first completion bit to one. Similarly, when the pointer to the next sibling of an element is complete, as well as its last child pointer, the other completion bit is set to one.

In partial parsing, the broker waits until a part of the message is referenced, and invokes the parser to parse that part of the message. Message processing nodes refer to fields within a message using hierarchical names. The name begins at the root of the message and proceeds down the message tree until the particular element is located. If an element is encountered without its completion bits set, and further navigation from this element is required, the appropriate parser entry point is called to parse the necessary part of the message. The relevant part of the message is parsed, appropriate elements are added to the logical message tree, and the element in question is marked as complete.

If you do not need to parse the full bit stream, you can use partial parsing. During partial parsing, a parser is called recursively until the requested element is returned, or until the message tree has been marked as complete, and the requested element is known not to exist.

Whether you choose to perform a full or partial parse depends on how the message will be processed. If most field elements within the message are likely to be accessed during processing, performing a full parse of the message when an attempt is made to access it is typically more efficient, particularly for smaller messages.

However, if most field elements within the message are not likely to be accessed during processing, performing a partial parse of the message when an attempt is made to access a specific field is typically more efficient, particularly when the message size grows.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"Parsing on demand" on page 4173
On-demand parsing, referred to as partial parsing, is used to parse an input message bit stream only as far as is necessary to satisfy the current reference. The parsers that can perform partial parsing of input messages are the JSON, MRM, XML, XMLNS, and XMLNSC parsers.

Specific types used by parsers:

Specific types are used when a parser needs additional information that is associated with some or all the elements in a tree in order to generate the bit stream.

For the XML parser, the specific type information is used to mark special elements such as components, processing instructions, and CDATA sections. The methods getSpecificType and setSpecificType are used by user-defined nodes to query this information and to generate message trees that use these special types.

Developers of user-defined parsers can generate their own specific type values to control special handling characteristics in their parser code using the existing C user-defined parser interface. The getSpecificType and setSpecificType methods enable Java user-defined nodes to fully use this parser capability.

Related tasks:

"Getting and setting the specific type of an Mb element" on page 3073
Two methods are provided for handling the specific type of an Mb syntax element.

"Creating a user-defined extension in Java" on page 3054
You must complete a series of tasks to create user-defined nodes that use the Java language.

"Creating a user-defined extension in C" on page 3026
You must complete a series of tasks to create user-defined extensions that use the C language.

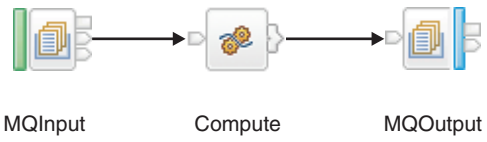
User exits:

A user exit is user-provided custom software, written in C, to track data passing through message flows.

User-provided functions can be invoked at specific points during the life cycle of a message while it passes through the message flow, and can invoke utility functions to query information about the point in the flow, and the contents of the message assembly. The utility function can also modify certain parts of the message assembly. For more information about using user exits, see "Why use a user exit?" on page 2984.

The user exits can be invoked when one or more of the following events occur:

- The end of a unit-of-work (UOW) or transaction (COMMIT or ROLLBACK).
- A message passes between two nodes.
- A message is successfully enqueued or sent to a transport in an output, reply, or request node.
- A message is dequeued or received in an input, response, or TimeoutNotification node.



In the basic message flow shown here, you can track messages at three levels:

- Transaction level
- Node level

- Input or output level

At the transaction level, you can track the following events:
- Messages being read into the flow
- Completion of the transaction

At the node level, you can track the following events:
- A message passing from one node to another
- Completion of processing for one node

At the message input or output level, you can track the following events:
- Messages being read into the flow
- Messages being written from the flow

Therefore, you can track five different types of event, which occur in the following sequence:

1. A message is dequeued from the input source (read into the flow).
2. A message is propagated to the node for processing.
3. A request message is sent to the output node's transport, and transport-specific destination information is written to "WrittenDestination" in the LocalEnvironment.
4. Node processing is completed.
5. The transaction ends.

Related concepts:

"Why use a user exit?" on page 2984
Use a user exit to intercept the progress of messages through message flows without having to redesign the message flow.

Related tasks:

"Exploiting user exits" on page 2985
Your message flows can benefit from user exits.

"Implementing a user-defined exit" on page 3113
You can develop and deploy a user-defined exit.

"Developing a user exit" on page 3114
Develop a user exit by declaring it, implementing its behavior, then compiling it.

"Deploying a user exit" on page 3116
Deploy your user exit to the broker.

Related reference:

"User-defined extensions" on page 6411
Reference material that supports the creation and management of your user-defined extensions.

Which language to use to implement a user-defined extension

You can use Java or C to implement a user-defined extension.

You can use C to implement all types of user-defined extension, but you can use Java to implement only user-defined nodes.

If you can, use Java for user-defined nodes, and use C for everything else.

You must compile user-defined nodes, parsers, and exits that are written in C into a loadable implementation library (LIL): that is, a shared library on Linux and UNIX systems, or a dynamic link library (DLL) on Windows systems. You must package user-defined nodes that are written in Java as a JAR file.

To achieve platform independence, use the ANSI standard C or Java programming languages, and avoid platform-specific code in your user-defined extension.

Related concepts:

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

"Why use a user-defined extension?" on page 2972
Use a user-defined node or parser when the built-in resources do not provide the required functions.

"Which type of user-defined extension to use" on page 2988
The types of user-defined extension that you can write include user-defined nodes, user-defined parsers, user-defined exits, and using a subflow as a user-defined node.

Related tasks:

"Implementing user-defined extensions" on page 3019
Create the resources for your user-defined extension. You can write user-defined nodes in C or Java, or you can use a subflow to create a node. You can write user-defined parsers and exits only in C.

"Compiling a C user-defined extension" on page 3047
Compile user-defined extensions in C for all supported operating systems.

"Compiling a Java user-defined node" on page 3074
When you have created the code for your Java user-defined node, you must compile it for your operating system.

Related reference:

"User-defined extensions" on page 6411
Reference material that supports the creation and management of your user-defined extensions.

Implementing the supplied user-defined extension samples

WebSphere Message Broker provides some sample code to help you understand how to write user-defined nodes and parsers.

About this task

You can see sample code for user-defined extensions in the following ways:

- By using the sample that is supplied in the WebSphere Message Broker Toolkit. To use the sample, see User-defined Extension.

  You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.
- By using sample code that is installed with WebSphere Message Broker. To use the sample code, read the following information in this topic.

The sample code consists of a sample parser, and the following sample nodes:

Node	Description
Switch	A node, implemented in both C and Java versions, that propagates an input message to one of several output terminals depending on the message content.

Node	Description
Transform	A node, implemented in both C and Java versions, that performs a simple message transformation.

Each sample node consists of the source files and some files that you can use to test each node. For the sample parser there are only source files. See “Sample node files” on page 6412 and “Sample parser files” on page 6414 for details of the sample files and where to find them.

To implement the supplied samples from the sample code that is installed with WebSphere Message Broker:

1. Compile the samples. For information about how to compile a Java node, see “Compiling a Java user-defined node” on page 3074. For information about how to compile a C node or parser, see “Compiling a C user-defined extension” on page 3047.
2. Install the user-defined extension in a broker domain. For instructions on completing this step, see “Installing user-defined extension runtime files on a broker” on page 3125.
3. Copy and extract the `SampleNodesProject.zip` file by competing the following steps:
   a. From a computer that has WebSphere Message Broker installed, copy the `SampleNodesProject.zip` file to a computer with WebSphere Message Broker Toolkit installed. The `.zip` file is in the `sample` subdirectory. For example, on Windows, the `SampleNodesProject.zip` is in the *`install_dir`*`\sample\extensions\nodes\com.ibm.samples.nodes` directory, where *install_dir* is the home directory of your WebSphere Message Broker installation.
   b. On the computer that has WebSphere Message Broker Toolkit installed, extract the `SampleNodesProject.zip` file, and copy the extracted files to a directory from which the WebSphere Message Broker Toolkit can access them. For more information about where to copy the files, see “Installing a user-defined extension to current and past versions of WebSphere Message Broker” on page 3128.
4. Open the WebSphere Message Broker Toolkit and switch to the Broker Application Development perspective. The category called "Sample nodes" is now visible in the palette, and the sample nodes are shown below it. Documentation about the sample nodes is also visible in the help system under "Samples".
5. Include the sample nodes in a message flow (see “Adding a message flow node” on page 1494).
6. Deploy the message flow; see Chapter 11, “Packaging and deploying,” on page 3209.
7. For the Switch and Transform nodes, you can put a message to the input queue of the message flow and observe the results, as follows:
   a. Make sure that the message flow that contains the sample node is deployed successfully; see “Checking the results of deployment” on page 3243.
   b. Use the Enqueue message function to put the sample input messages (provided in `.xml` files) to the input queue named on the input node of the message flow; see “Debug: putting a test message on an input queue” on page 3162.

   You can also use a Trace node or the Flow debugger to see what is happening in your message flow.

Related concepts:

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

"Developing user-defined extensions" on page 2970
A user-defined extension is a component that you design and implement to extend the function of WebSphere Message Broker.

Related tasks:

"Adding a message flow node" on page 1494
When you have created a message flow, add nodes to define its function.

Chapter 10, "Testing and debugging message flow applications," on page 3143
Use the flow debugger to track messages through your message flows and check for errors, or use the Test Client to test your message flows in a safe environment before they are used on a production system.

"Compiling a Java user-defined node" on page 3074
When you have created the code for your Java user-defined node, you must compile it for your operating system.

"Compiling a C user-defined extension" on page 3047
Compile user-defined extensions in C for all supported operating systems.

"Installing a user-defined extension to current and past versions of WebSphere Message Broker" on page 3128
Install user-defined extensions that you have developed yourself, or have acquired from an independent software vendor, with the minimum of user intervention.

"Checking the results of deployment" on page 3243
After you have made a deployment, check that the operation has completed successfully.

Chapter 13, "Troubleshooting and support," on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

Related reference:

"Trace node" on page 4942
Use the Trace node to generate trace records that you can use to monitor the behavior of a message flow.

"User-defined extensions" on page 6411
Reference material that supports the creation and management of your user-defined extensions.

Implementing user-defined extensions

Create the resources for your user-defined extension. You can write user-defined nodes in C or Java, or you can use a subflow to create a node. You can write user-defined parsers and exits only in C.

Before you begin

Before you start:

Read the following topics:

- "User-defined extensions overview" on page 2971
- "Which type of user-defined extension to use" on page 2988

About this task

To create a user-defined extension, follow the instructions in the appropriate topic:

- "Implementing a user-defined node" on page 3022
- "Implementing a user-defined parser" on page 3099
- "Implementing a user-defined exit" on page 3113

For user-defined nodes only, you must create a WebSphere Message Broker Toolkit Eclipse plug-in. For user-defined nodes created in Java and C, you must also create the run time `.lil` or `.jar` file. The WebSphere Message Broker Toolkit plug-in adds the user-defined node to the node palette in the Message Flow editor, so that you can include the new node in message flows. This additional task is described in "Creating the user interface representation of a user-defined node in the WebSphere Message Broker Toolkit" on page 3079. This step is not required for user-defined parsers or exits.

The following table shows the tasks required for creating the different types of user-defined extension.

Action	Topics to view
To use one of the Java sample nodes	1. "Compiling a Java user-defined node" on page 3074 2. "Installing user-defined extension runtime files on a broker" on page 3125 3. "Creating the user interface representation of a user-defined node in the WebSphere Message Broker Toolkit" on page 3079 4. "Testing a user-defined node" on page 3094
To use one of the C sample nodes	1. "Compiling a C user-defined extension" on page 3047 2. "Installing user-defined extension runtime files on a broker" on page 3125 3. "Creating the user interface representation of a user-defined node in the WebSphere Message Broker Toolkit" on page 3079 4. "Testing a user-defined node" on page 3094
To use the sample parser	1. "Compiling a C user-defined extension" on page 3047 2. "Installing user-defined extension runtime files on a broker" on page 3125
To create your own Java node by using the WebSphere Message Broker Toolkit	1. "Creating an input node in Java" on page 3055 or "Creating a message processing or output node in Java" on page 3062 2. "Using error logging from a user-defined extension" on page 3137 3. "Compiling a Java user-defined node" on page 3074 4. "Testing a user-defined node" on page 3094 5. "Packaging and distributing a user-defined node project" on page 3121 6. "Installing a user-defined extension to current and past versions of WebSphere Message Broker" on page 3128

Action	Topics to view
To create your own C node	1. "Creating an input node in C" on page 3027 or "Creating a message processing or output node in C" on page 3036 2. "Using error logging from a user-defined extension" on page 3137 3. "Compiling a C user-defined extension" on page 3047 4. "Installing user-defined extension runtime files on a broker" on page 3125 5. "Creating the user interface representation of a user-defined node in the WebSphere Message Broker Toolkit" on page 3079 6. "Testing a user-defined node" on page 3094 7. "Packaging and distributing a user-defined node project" on page 3121 8. "Installing a user-defined extension to current and past versions of WebSphere Message Broker" on page 3128
To create your own node from a subflow from scratch	1. "Creating a user-defined node from a subflow from scratch" on page 3076 2. "Creating the user interface representation of a user-defined node in the WebSphere Message Broker Toolkit" on page 3079 3. "Testing a subflow user-defined node project" on page 3097 4. "Packaging and distributing a user-defined node project" on page 3121 5. "Installing a user-defined node" on page 1496
To create your own node from an existing subflow	1. "Creating a user-defined node from an existing subflow" on page 3078 2. "Testing a subflow user-defined node project" on page 3097 3. "Packaging and distributing a user-defined node project" on page 3121 4. "Installing a user-defined node" on page 1496
To create your own parser	1. "Implementing a user-defined parser" on page 3099 2. "Using error logging from a user-defined extension" on page 3137 3. "Compiling a C user-defined extension" on page 3047 4. "Installing user-defined extension runtime files on a broker" on page 3125
To create a user exit	1. "Developing a user exit" on page 3114 2. "Compiling a C user-defined extension" on page 3047 3. "Deploying a user exit" on page 3116

Related concepts:

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

"Which type of user-defined extension to use" on page 2988
The types of user-defined extension that you can write include user-defined nodes, user-defined parsers, user-defined exits, and using a subflow as a user-defined

node.

Related tasks:

"Implementing a user-defined node"
You can implement a user-defined node to extend the function of WebSphere Message Broker.

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

"Implementing a user-defined exit" on page 3113
You can develop and deploy a user-defined exit.

"Creating the user interface representation of a user-defined node in the WebSphere Message Broker Toolkit" on page 3079
When you are developing a user-defined node in Java or C only, you must create the user interface representation of the node in the WebSphere Message Broker Toolkit.

Related reference:

"User-defined extensions" on page 6411
Reference material that supports the creation and management of your user-defined extensions.

Implementing a user-defined node

You can implement a user-defined node to extend the function of WebSphere Message Broker.

Before you begin

Before you start:

Read the following topics:

- "User-defined extensions overview" on page 2971
- "Why use a user-defined extension?" on page 2972
- "User-defined nodes" on page 2989
- "Which type of user-defined extension to use" on page 2988

About this task

Consider the following restrictions and factors when developing user-defined nodes:

- Interfacing a C user-defined node to Java and providing a JNI wrapper is not supported. This restriction exists because the broker internally initializes a JVM, which is unavailable through the user-defined extension interface. The JVM initializes with various parameters that are specific to the requirements of the broker. Because only one JVM exists in a process, whoever initializes it first specifies these parameters. If a user-defined node uses Java, and the broker is initialized first, these parameters might not be suitable for the user-defined node. If the user-defined node creates the JVM before the broker starts, the broker might not function correctly.
- User-defined input nodes can support only XML, BLOB, and the WebSphere MQ parsers.
- Avoid using functions that are specific to an operating system. If you code in this way, your user-defined extensions can work on various operating systems without requiring changes to the source code.

- Always put a user-defined node into a non-default schema because a user-defined node in a broker schema is known to other message flows by its schema qualified name. For example, if a user-defined node is named `ErrorHandler` and it is in broker schema `com.ibm.mb.toolkit`, it is referenced as `com.ibm.mb.toolkit.ErrorHandler`. If a second provider also has an error handler named `ErrorHandler` and it is in broker schema `com.xxx.product`, it is referenced as `com.xxx.product.ErrorHandler`. A user-defined node in a default schema is addressed by its name only. Therefore, if two different providers develop two unrelated error handlers and both are named `ErrorHandler` and both are placed in a default schema, when both user-defined nodes are in the plug-in space for a third user, the reference to `ErrorHandler` is ambiguous.
- If you want to use a subflow to create a user-defined node, read the limitations section in the following topic: "Using a subflow as a user-defined node" on page 3008.

To implement a user-defined node, complete the following tasks in the specified order:

Procedure

1. "Designing a user-defined node" on page 3024
2. "Creating a user-defined node" on page 3025
3. "Packaging and distributing user-defined extensions" on page 3117
4. "Testing a user-defined node" on page 3094
5. "Packaging and distributing a user-defined node project" on page 3121

Related concepts:

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

"Why use a user-defined extension?" on page 2972
Use a user-defined node or parser when the built-in resources do not provide the required functions.

"User-defined nodes" on page 2989
User-defined nodes are the main mechanism for extending the functions of WebSphere Message Broker.

"Which type of user-defined extension to use" on page 2988
The types of user-defined extension that you can write include user-defined nodes, user-defined parsers, user-defined exits, and using a subflow as a user-defined node.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

"Implementing a user-defined exit" on page 3113
You can develop and deploy a user-defined exit.

Related reference:

"User-defined extensions" on page 6411
Reference material that supports the creation and management of your user-defined extensions.

Designing a user-defined node:

Decide what type of node you need to implement the functions that are required by your application.

Before you begin

Before you start:

Read "Deciding which nodes to use" on page 1457 to understand the different types of node. You might need more than one node to implement all the functions that you require.

About this task

The functions that you require might not be satisfied by a template that already exists for several reasons:

- The functions that you require do not relate to interacting with external systems. Most of the node design pattern concentrates on communication with external systems, which is the most likely requirement for a user-defined node.
- The functions that are required are not well suited to the WebSphere Message Broker architecture, so you should implement them in an end application, or an application server.
- The functions require complex control and state information, which you should not implement as a plug-in.

Related concepts:

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

"Why use a user-defined extension?" on page 2972
Use a user-defined node or parser when the built-in resources do not provide the required functions.

"User-defined nodes" on page 2989
User-defined nodes are the main mechanism for extending the functions of WebSphere Message Broker.

"Which type of user-defined extension to use" on page 2988
The types of user-defined extension that you can write include user-defined nodes, user-defined parsers, user-defined exits, and using a subflow as a user-defined node.

Related tasks:

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

"Implementing a user-defined exit" on page 3113
You can develop and deploy a user-defined exit.

Related reference:

"User-defined extensions" on page 6411
Reference material that supports the creation and management of your user-defined extensions.

Creating a user-defined node:

You can write user-defined nodes in C, Java, or from a subflow.

Before you begin

Before you start:

Read "Designing a user-defined node" on page 3024.

About this task

When you have created a user-defined node, you can test it, as described in "Testing a user-defined node" on page 3094. If you want to test or use user-defined nodes or parsers on multiple computers, follow the instructions given in "Packaging and distributing user-defined extensions" on page 3117.

Decide whether you want to create a user-defined node in C, Java, or from a subflow, then follow the instructions in the appropriate topic.

- "Creating a user-defined extension in C" on page 3026
- "Creating a user-defined extension in Java" on page 3054
- "Creating a user-defined node from a subflow" on page 3076
- "Creating the user interface representation of a user-defined node in the WebSphere Message Broker Toolkit" on page 3079

The following table shows the tasks that are involved in creating the different types of user-defined node. For more information about using user-defined nodes, see the following sample: User-defined Extension.

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Objective	Tasks to complete
To create your own Java node by using the WebSphere Message Broker Toolkit	1. "Creating an input node in Java" on page 3055 or "Creating a message processing or output node in Java" on page 3062 2. "Using error logging from a user-defined extension" on page 3137 3. "Compiling a Java user-defined node" on page 3074 4. "Testing a user-defined node" on page 3094 5. "Packaging and distributing a user-defined node project" on page 3121 6. "Installing a user-defined extension to current and past versions of WebSphere Message Broker" on page 3128

Objective	Tasks to complete
To create your own C node:	1. "Creating an input node in C" on page 3027 or "Creating a message processing or output node in C" on page 3036 2. "Using error logging from a user-defined extension" on page 3137 3. "Compiling a C user-defined extension" on page 3047 4. "Installing user-defined extension runtime files on a broker" on page 3125 5. "Creating the user interface representation of a user-defined node in the WebSphere Message Broker Toolkit" on page 3079 6. "Testing a user-defined node" on page 3094 7. "Packaging and distributing a user-defined node project" on page 3121 8. "Installing a user-defined extension to current and past versions of WebSphere Message Broker" on page 3128
To create your own node from a subflow	1. "Creating a user-defined node from a subflow" on page 3076 2. "Creating the user interface representation of a user-defined node in the WebSphere Message Broker Toolkit" on page 3079 3. "Testing a subflow user-defined node project" on page 3097 4. "Packaging and distributing a user-defined node project" on page 3121 5. "Installing a user-defined node" on page 1496

Related concepts:

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Designing a user-defined node" on page 3024
Decide what type of node you need to implement the functions that are required by your application.

"Testing a user-defined node" on page 3094
When you have created and installed the required resources, you can test your user-defined node.

"Packaging and distributing user-defined extensions" on page 3117
When you have created and tested a user-defined extension, you can package and distribute it.

Related reference:

"User-defined extensions" on page 6411
Reference material that supports the creation and management of your user-defined extensions.

Creating a user-defined extension in C:

You must complete a series of tasks to create user-defined extensions that use the C language.

About this task

You can write user-defined nodes and user-defined parsers in C.

Complete the appropriate tasks from the following list:

- "Creating an input node in C"
- "Creating a message processing or output node in C" on page 3036
- "Implementing a user-defined parser" on page 3099
- "Compiling a C user-defined extension" on page 3047

What to do next

When you have completed this set of tasks, continue with the following tasks:

- If you have compiled a user-defined node, "Creating the user interface representation of a user-defined node in the WebSphere Message Broker Toolkit" on page 3079
- "Testing a user-defined node" on page 3094
- "Packaging and distributing user-defined extensions" on page 3117

Related concepts:

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related reference:

"User-defined extensions" on page 6411
Reference material that supports the creation and management of your user-defined extensions.

Creating an input node in C:

Create a user-defined input node in C to receive messages into a message flow.

Before you begin

Before you start

Read the following topics:

- "Why use a user-defined extension?" on page 2972
- "User-defined input nodes" on page 2990

About this task

A loadable implementation library, or *LIL*, is the implementation module for a C node. A LIL is implemented as a shared or dynamic link library (DLL), but has the file extension `.lil` not `.dll`.

The implementation functions that you write for the node are listed in "C node implementation functions" on page 6417. You can call utility functions, implemented in the runtime broker, to help with the node operation; these functions are listed in "C node utility functions" on page 6419.

WebSphere Message Broker provides the source for two sample user-defined nodes called SwitchNode and TransformNode. You can use these nodes in their current state, or you can modify them.

To create an input node in C:

1. "Declaring and defining the node"
2. "Creating an instance of the node" on page 3029
3. "Setting attributes" on page 3030
4. "Implementing the node functionality" on page 3031
5. "Overriding the default message parser attributes (optional)" on page 3031
6. "Deleting an instance of the node" on page 3032

Declaring and defining the node:
About this task

To declare and define a user-defined node to the broker, include an initialization function, bipGetMessageflowNodeFactory, in your LIL. The following steps outline how the broker calls your initialization function, and how your initialization function declares and defines the user-defined node:

Procedure

1. The initialization function, bipGetMessageflowNodeFactory, is called by the broker after the operating system has loaded and initialized the LIL. The broker calls this function to understand what your LIL can do and how the broker can call the LIL. For example:

   ```
   CciFactory LilFactoryExportPrefix * LilFactoryExportSuffix
   bipGetMessageflowNodeFactory()
   ```

2. The bipGetMessageflowNodeFactory function must call the utility function cniCreateNodeFactory. This function passes back a unique factory name (or group name) for all the nodes that your LIL supports. Every factory name (or group name) that is passed back must be unique throughout all the LILs in a single runtime broker.
3. The LIL must call the utility function cniDefineNodeClass to pass the unique name of each node, and a virtual function table of the addresses of the implementation functions.

   For example, the following code declares and defines a single node called InputxNode:

   ```
   {
    CciFactory* factoryObject;
    int rc = 0;
    CciChar factoryName[] = L"MyNodeFactory";
    CCI_EXCEPTION_ST exception_st;

    /* Create the Node Factory for this node */
    factoryObject = cniCreateNodeFactory(0, factoryName);
    if (factoryObject == CCI_NULL_ADDR) {

     /* Any local error handling can go here */
    }
    else {
     /* Define the nodes supported by this factory */
     static CNI_VFT vftable = {CNI_VFT_DEFAULT};

    /* Setup function table with pointers to node implementation functions */
    vftable.iFpCreateNodeContext = _createNodeContext;
    vftable.iFpDeleteNodeContext = _deleteNodeContext;
    vftable.iFpGetAttributeName2 = _getAttributeName2;
    vftable.iFpSetAttribute      = _setAttribute;
    vftable.iFpGetAttribute2     = _getAttribute2;
    vftable.iFpRun               = _run;
   ```

```
 cniDefineNodeClass(0, factoryObject, L"InputxNode", &vftable);
 }

 /* Return address of this factory object to the broker */
 return(factoryObject);
}
```

A user-defined node identifies itself as providing the features of an input node by implementing the cniRun implementation function.

User-defined nodes have to implement either a cniRun or a cniEvaluate implementation function. If they do not, the broker does not load the user-defined node, and the cniDefineNodeClass utility function fails, returning CCI_MISSING_IMPL_FUNCTION.

For example:

```
int cniRun(
  CciContext* context,
  CciMessage* localEnvironment,
  CciMessage* exceptionList,
  CciMessage* message
){
  ...
  /* Get data from external source */
  return CCI_SUCCESS_CONTINUE;
}
```

Use the return value periodically to return control to the broker.

When a message flow that contains a user-defined input node is deployed successfully, the cniRun function of the node is called repeatedly to enable the node to retrieve messages and propagate them to the rest of the message flow. For the minimum code required to compile a C user-defined node, see the "C skeleton code" on page 6683.

Creating an instance of the node:
About this task

To instantiate your node:

Procedure

1. When the broker has received the table of function pointers, it calls the function cniCreateNodeContext for each instantiation of the user-defined node. For example, if three message flows are using your user-defined node, your cniCreateNodeContext function is called for each of them. This function must allocate memory for that instantiation of the user-defined node to hold the values for the configured attributes. For example:
   a. Call the cniCreateNodeContext function:

```
CciContext* _createNodeContext(
  CciFactory* factoryObject,
  CciChar*    nodeName,
  CciNode*    nodeObject
){
  static char* functionName = (char *)"_createNodeContext()";
  NODE_CONTEXT_ST* p;
  CciChar          buffer[256];
```

   b. Allocate a pointer to the local context and clear the context area:

```
  p = (NODE_CONTEXT_ST *)malloc(sizeof(NODE_CONTEXT_ST));

  if (p) {
     memset(p, 0, sizeof(NODE_CONTEXT_ST));
```

   c. Save the node object pointer in the context:

```
p->nodeObject = nodeObject;
```

d. Save the node name:

```
CciCharNCpy((CciChar*)&p->nodeName, nodeName, MAX_NODE_NAME_LEN);
```

e. Return the node context:

```
return (CciContext*) p;
```

2. An input node has a number of output terminals associated with it, but typically does not have any input terminals. Use the utility function cniCreateOutputTerminal to add output terminals to an input node when the node is instantiated. These functions must be invoked within the cniCreateNodeContext implementation function. For example, to define an input node with three output terminals:

```
{
   const CciChar* ucsOut = CciString("out", BIP_DEF_COMP_CCSID) ;
   insOutputTerminalListEntry(p, (CciChar*)ucsOut);
   free((void *)ucsOut) ;
 }
 {
   const CciChar* ucsFailure = CciString("failure", BIP_DEF_COMP_CCSID) ;
   insOutputTerminalListEntry(p, (CciChar*)ucsFailure);
   free((void *)ucsFailure) ;
 }
 {
   const CciChar* ucsCatch = CciString("catch", BIP_DEF_COMP_CCSID) ;
   insOutputTerminalListEntry(p, (CciChar*)ucsCatch);
   free((void *)ucsCatch) ;    }
```

For the minimum code required to compile a C user-defined node, see "C skeleton code" on page 6683.

Setting attributes:

About this task

Attributes are set whenever you start the broker, or when you redeploy the message flow with new values.

Following the creation of output terminals, the broker calls the cniSetAttribute function to pass the values for the configured attributes of the user-defined node. For example:

```
{
  const CciChar* ucsAttr = CciString("nodeTraceSetting", BIP_DEF_COMP_CCSID) ;
  insAttrTblEntry(p, (CciChar*)ucsAttr, CNI_TYPE_INTEGER);
  _setAttribute(p, (CciChar*)ucsAttr, (CciChar*)constZero);
  free((void *)ucsAttr) ;
}
{
  const CciChar* ucsAttr = CciString("nodeTraceOutfile", BIP_DEF_COMP_CCSID) ;
  insAttrTblEntry(p, (CciChar*)ucsAttr, CNI_TYPE_STRING);
  _setAttribute(p, (CciChar*)ucsAttr, (CciChar*)constSwitchTraceLocation);
  free((void *)ucsAttr) ;
}
```

The number of configuration attributes that a node can have is unlimited. However, a user-defined node must not implement an attribute that is already implemented as a base configuration attribute. The base attributes are:

- label
- userTraceLevel
- traceLevel
- userTraceFilter

- traceFilter

Implementing the node functionality:
About this task

When the broker knows that it has an input node, it calls the cniRun function of this node at regular intervals. The cniRun function must then decide what course of action it must take. If data is available for processing, the cniRun function can propagate the message. If no data is available for processing, the cniRun function must return with CCI_TIMEOUT so that the broker can continue configuration changes.

For example, to configure the node to call cniDispatchThread and process the message, or return with CCI_TIMEOUT:

```
If ( anything to do )
 CniDispatchThread;

   /* do the work */

 If ( work done O.K.)
  Return CCI_SUCCESS_CONTINUE;
 Else
  Return CCI_FAILURE_CONTINUE;
Else
  Return CCI_TIMEOUT;
```

Overriding the default message parser attributes (optional):
About this task

An input node implementation typically determines what message parser initially parses an input message. For example, the MQInput node dictates that an MQMD parser is required to parse the MQMD header. A user-defined input node can select an appropriate header or message parser, and the mode in which the parsing is controlled, by using or overriding the following attributes that are included as default:

rootParserClassName
Defines the name of the root parser that parses message formats that are supported by the user-defined input node. It defaults to `GenericRoot`, a supplied root parser that causes the broker to allocate and chain parsers together. It is unlikely that a node would need to modify this attribute value.

firstParserClassName
Defines the name of the first parser, in what might be a chain of parsers that are responsible for parsing the bit stream. It defaults to `XML`.

messageDomainProperty
An optional attribute that defines the name of the message parser that is required to parse the input message. The supported values are the same as the values that are supported by the MQInput node. (See "MQInput node" on page 4594 for more information.)

messageSetProperty
An optional attribute that defines the message set identifier, or the message set name, in the `Message Set` field, only if the MRM parser was specified by the `messageDomainProperty` attribute.

messageTypeProperty
An optional attribute that defines the identifier of the message in the `MessageType` field, only if the MRM parser was specified by the `messageDomainProperty` attribute.

messageFormatProperty
An optional attribute that defines the format of the message in the `MessageFormat` field, only if the MRM parser was specified by the `messageDomainProperty` attribute.

If you have written a user-defined input node that always begins with data of a known structure, you can ensure that a specific parser handles the start of the data. For example, the MQInput node reads data only from WebSphere MQ queues, therefore this data always has an MQMD at the beginning, and the MQInput node sets firstParserClassName to MQHMD. If your user-defined node always handles data that begins with a structure that can be parsed by a specific parser, for example "MYPARSER", set firstParserClassName to MYPARSER in the following way::

1. Declare the implementation functions:

```
CciFactory LilFactoryExportPrefix * LilFactoryExportSuffix bipGetMessageflowNodeFactory()
{
  ....
  CciFactory*      factoryObject;
  ....
  factoryObject = cniCreateNodeFactory(0, (unsigned short *)constPluginNodeFactory);
  ...
  vftable.iFpCreateNodeContext = _createNodeContext;
  vftable.iFpSetAttribute      = _setAttribute;
  vftable.iFpGetAttribute      = _getAttribute;
  ...
  cniDefineNodeClass(&rc, factoryObject, (CciChar*)constSwitchNode, &vftable);
  ...
  return(factoryObject);
}
```

2. Set the attribute in the cniCreateNodeContext implementation function:

```
CciContext* _createNodeContext(
  CciFactory* factoryObject,
  CciChar*    nodeName,
  CciNode*    nodeObject
){
  NODE_CONTEXT_ST* p;
  ...

    /* Allocate a pointer to the local context */
    p = (NODE_CONTEXT_ST *)malloc(sizeof(NODE_CONTEXT_ST));
    /* Create attributes and set default values */
    {
      const CciChar* ucsAttrName  = CciString("firstParserClassName", BIP_DEF_COMP_CCSID);
      const CciChar* ucsAttrValue = CciString("MYPARSER", BIP_DEF_COMP_CCSID) ;
      insAttrTblEntry(p, (CciChar*)ucsAttrName, CNI_TYPE_INTEGER);
      /*see sample BipSampPluginNode.c for implementation of insAttrTblEntry*/

      _setAttribute(p, (CciChar*)ucsAttrName, (CciChar*)ucsAttrValue);
      free((void *)ucsAttrName) ;
      free((void *)ucsAttrValue) ;
    }
```

In the code example above, the insAttrTblEntry method is called. This method is declared in the SwitchNode and TransformNode sample user-defined nodes.

Deleting an instance of the node:

About this task

Nodes are destroyed when a message flow is redeployed, or when the execution group process is stopped (using the **mqsistop** command). When a node is destroyed, you must call the cniDeleteNodeContext function to free all used memory and release all held resources. For example:

```
void _deleteNodeContext(
  CciContext* context
){
  static char* functionName = (char *)"_deleteNodeContext()";

  return;
}
```

Related concepts:

"User-defined extensions in the runtime environment" on page 2980
Before you design and implement user-defined extensions, familiarize yourself with the core components. Ensure that you also understand the basic WebSphere Message Broker runtime architecture.

"User-defined input nodes" on page 2990
A user-defined input node is an extension to the broker that provides a new input node in addition to the nodes supplied with the product.

"Developing user-defined extensions" on page 2970
A user-defined extension is a component that you design and implement to extend the function of WebSphere Message Broker.

"Why use a user-defined extension?" on page 2972
Use a user-defined node or parser when the built-in resources do not provide the required functions.

Related tasks:

"Extending the capability of a C input node" on page 3034
When you have created a user-defined node, you can extend its capability.

"Implementing the supplied user-defined extension samples" on page 3017
WebSphere Message Broker provides some sample code to help you understand how to write user-defined nodes and parsers.

"Compiling a C user-defined extension" on page 3047
Compile user-defined extensions in C for all supported operating systems.

Related reference:

"C skeleton code" on page 6683
Use the skeleton code that is supplied as guidance for your C user-defined node. The code has the minimum content that is required to compile a user-defined node successfully.

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

"Element definitions for message parsers" on page 4237

"C language user-defined node API" on page 6416
Learn about the different types of call provided by the API.

"cniCreateNodeFactory" on page 6449
Use this function to create a node factory in the broker. A single instance of the named message flow node factory is created.

"cniDefineNodeClass" on page 6451
Use this function to define a node class, as specified by the *name* parameter, which is supported by the node factory specified as the*factoryObject* parameter.

Extending the capability of a C input node:

When you have created a user-defined node, you can extend its capability.

Before you begin

Before you start

Read "Creating an input node in C" on page 3027.

About this task

After you have created a user-defined node, the following options are available:

1. "Receiving external data into a buffer"
2. "Controlling threading and transactions"
3. "Propagating the message" on page 3035

Receiving external data into a buffer:
About this task

An input node can receive data from any type of external source, such as a file system or FTP connection, provided that the output from the node is in the correct format. For connections to queues or databases, use the built-in nodes and the API calls supplied, principally because the built-in nodes are already set up for error handling. Do not use the MQGET or MQPUT calls for direct access to WebSphere MQ queues.

You must provide an input buffer (or bit stream) to contain input data, and associate it with a message object. In the C API, the buffer is attached to the CciMessage object that represents the input message by using the cniSetInputBuffer utility function. For example:

```
{
  static char* functionName = (char *)"_Input_run()";
  void*        buffer;
  CciTerminal* terminalObject;
  int          buflen = 4096;
  int          rc = CCI_SUCCESS;
  int          rcDispatch = CCI_SUCCESS;

  buffer = readFromDevice(&buflen);
  cniSetInputBuffer(&rc, message, buffer, buflen);
}
/*propagate etc*/
```

Controlling threading and transactions:
About this task

An input node must perform appropriate end-of-message processing when a message has been propagated through a message flow. Specifically, the input node needs to cause any transactions to be committed or rolled back, and return threads to the thread pool.

Each message flow thread is allocated from a pool of threads that is maintained for each message flow, and starts execution in the cniRun function. You determine the behavior of a thread using the cniDispatchThread utility function, together with the appropriate return value.

From the cniRun function, you can call the cniDispatchThread utility function to cause another thread to start executing the cniRun function. The most appropriate time to execute another thread is directly after you have established that data could be available for the function to process on the new thread.

The term *transaction* is used generically to describe either a globally coordinated transaction, or a broker-controlled transaction. Globally coordinated transactions are coordinated by either WebSphere MQ as an XA-compliant Transaction Manager, or Resource Recovery Service (RRS) on z/OS. WebSphere Message Broker controls transactions by committing (or rolling back) any database resources, and then committing any WebSphere MQ units of work. However, if a user-defined node is used, the broker cannot automatically commit any resource updates. The user-defined node uses return values to indicate whether a transaction has been successful, and to control whether transactions are committed or rolled-back. The broker infrastructure catches any unhandled exceptions, and rolls back the transaction.

The following table describes each of the supported return values, the effect that each one has on any transactions, and what the broker does with the current thread.

Return value	Affect on transaction	Broker action on the thread
CCI_SUCCESS_CONTINUE	Committed	Calls the same thread again in the cniRun function.
CCI_SUCCESS_RETURN	Committed	Returns the thread to the thread pool.
CCI_FAILURE_CONTINUE	Rolled back	Calls the same thread again in the cniRun function.
CCI_FAILURE_RETURN	Rolled back	Returns the thread to the thread pool.
CCI_TIMEOUT	Not applicable. The function periodically times out while it is waiting for an input message.	Calls the same thread again in the cniRun function.

The following code is an example of using the SUCCESS_RETURN return code with the cniDispatchThread function:

```
{
  ...
  cniDispatchThread(&rcDispatch, ((NODE_CONTEXT_ST *)context)->nodeObject);
  ...
  if (rcDispatch == CCI_NO_THREADS_AVAILABLE) return CCI_SUCCESS_CONTINUE;
  else return CCI_SUCCESS_RETURN;
}
```

Propagating the message:
About this task

Before you propagate a message, decide what message flow data you want to propagate, and which terminal is to receive the data.

The terminalObject is derived from a list that the user-defined node maintains.

For example, to propagate the message to the output terminal, use the cniPropagate function:

```
if (terminalObject) {
  if (cniIsTerminalAttached(&rc, terminalObject)) {
    if (rc == CCI_SUCCESS) {
      cniPropagate(&rc, terminalObject, localEnvironment, exceptionList, message);
    }
  }
```

In the above example, the cniIsTerminalAttached function is used to test whether the message can be propagated to the specified terminal. If you do not use the cniIsTerminalAttached function, and the terminal is not attached to another node by a connector, the message is not propagated and no warning message is returned. Use the cniIsTerminalAttached function to prevent this error occurring.

Related concepts:

"User-defined extensions in the runtime environment" on page 2980
Before you design and implement user-defined extensions, familiarize yourself with the core components. Ensure that you also understand the basic WebSphere Message Broker runtime architecture.

"User-defined input nodes" on page 2990
A user-defined input node is an extension to the broker that provides a new input node in addition to the nodes supplied with the product.

"Developing user-defined extensions" on page 2970
A user-defined extension is a component that you design and implement to extend the function of WebSphere Message Broker.

"Why use a user-defined extension?" on page 2972
Use a user-defined node or parser when the built-in resources do not provide the required functions.

Related tasks:

"Implementing the supplied user-defined extension samples" on page 3017
WebSphere Message Broker provides some sample code to help you understand how to write user-defined nodes and parsers.

"Compiling a C user-defined extension" on page 3047
Compile user-defined extensions in C for all supported operating systems.

Related reference:

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

"C language user-defined node API" on page 6416
Learn about the different types of call provided by the API.

"cniSetInputBuffer" on page 6520
Use this function to supply a buffer. It is used only by input nodes. The address is specified by the `source` parameter as an input bit stream of the input message to the broker.

Creating a message processing or output node in C:

A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Before you begin

Before you start

Read the following topics:

- "Why use a user-defined extension?" on page 2972
- "User-defined message processing nodes" on page 2996
- "User-defined output nodes" on page 3006

About this task

When you code a message processing node or an output node, the nodes provide essentially the same services. You can perform message processing in an output node, and you can send a message to a bit stream by using a message processing node. For simplicity, this topic refers mainly to the node as a message processing node but it does also contain information about the functions of both types of node.

A loadable implementation library (LIL), is the implementation module for a C node. A LIL is implemented as a shared or dynamic link library (DLL), but has the file extension `.lil` not `.dll`.

For more information about the C node implementation functions that you write for the node, see "C node implementation functions" on page 6417. You can call C node utility functions, implemented in the runtime broker, to help with the node operation; see "C node utility functions" on page 6419.

WebSphere Message Broker provides the source for two sample user-defined nodes called SwitchNode and TransformNode. You can use these nodes in their current state, or you can modify them. In addition, you can view the following sample which demonstrates the use of user-defined nodes, including a message processing node written in C.

- User-defined Extension

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

To create either type of node complete the following tasks:

1. "Declaring and defining your node"
2. "Creating an instance of the node" on page 3039
3. "Setting attributes" on page 3041
4. "Implementing the node functionality" on page 3041
5. "Deleting an instance of the node" on page 3042

Declaring and defining your node:
About this task

To declare and define a user-defined node to the broker, include an initialization function, bipGetMessageflowNodeFactory, in your LIL. The following steps take place on the configuration thread and outline how the broker calls your initialization function and how your initialization function declares and defines the user-defined node:

Procedure

1. The broker calls the initialization function bipGetMessageflowNodeFactory after the operating system has loaded and initialized the LIL. The broker calls this function to understand what your LIL can do and how the broker can call the LIL. For example:

   ```
   CciFactory LilFactoryExportPrefix * LilFactoryExportSuffix
   bipGetMessageflowNodeFactory()
   ```

2. The bipGetMessageflowNodeFactory function must call the utility function cniCreateNodeFactory. This function passes back a factory name (or group name) for all the nodes that your LIL supports. The factory name (or group name) must be unique throughout all the LILs in a single runtime broker.

3. The LIL must call the utility function cniDefineNodeClass to pass the unique name of each node and a virtual function table of the addresses of the implementation functions.

   For example, the following code declares and defines a single node called MessageProcessingxNode:

   ```
   {
    CciFactory* factoryObject;
    int rc = 0;
    CciChar factoryName[] = L"MyNodeFactory";
    CCI_EXCEPTION_ST exception_st;

    /* Create the Node Factory for this node */
    factoryObject = cniCreateNodeFactory(0, factoryName);
    if (factoryObject == CCI_NULL_ADDR) {
     /* Any local error handling can go here */
    }
    else {
     /* Define the nodes supported by this factory */
    static CNI_VFT vftable = {CNI_VFT_DEFAULT};

    /* Setup function table with pointers to node implementation functions */
    vftable.iFpCreateNodeContext = _createNodeContext;
    vftable.iFpDeleteNodeContext = _deleteNodeContext;
    vftable.iFpGetAttributeName2 = _getAttributeName2;
    vftable.iFpSetAttribute      = _setAttribute;
    vftable.iFpGetAttribute2     = _getAttribute2;
    vftable.iFpEvaluate          = _evaluate;

    cniDefineNodeClass(0, factoryObject, L"MessageProcessingxNode", &vftable);

    }

    /* Return address of this factory object to the broker */
    return(factoryObject);
   }
   ```

   A user-defined node identifies itself as a message processing or output node by implementing the cniEvaluate function. User-defined nodes must implement either a cniEvaluate or a cniRun implementation function, otherwise the broker does not load the user-defined node, and the cniDefineNodeClass utility function fails, returning CCI_MISSING_IMPL_FUNCTION.

   When a message flow containing a user-defined message processing node is deployed successfully, the node's cniEvaluate function is called for each message propagated to the node.

   Message flow data is received at the input terminal of the node, that is, the message, Environment, LocalEnvironment, and ExceptionList.

   For example:

```
void cniEvaluate(
  CciContext* context,
  CciMessage* localEnvironment,
  CciMessage* exceptionList,
  CciMessage* message
){
  ...
}
```

For the minimum code required to compile a C user-defined node, see "C skeleton code" on page 6683.

Creating an instance of the node:
About this task

To instantiate your node:

Procedure

1. When the broker has received the table of function pointers, it calls the function cniCreateNodeContext for each instantiation of the user-defined node. For example, if three message flows are using your user-defined node, your cniCreateNodeContext function is called for each of them. This function allocates memory for that instantiation of the user-defined node to hold the values for the configured attributes. For example:
   a. The user function cniCreateNodeContext is called:
   ```
   CciContext* _Switch_createNodeContext(
     CciFactory* factoryObject,
     CciChar*    nodeName,
     CciNode*    nodeObject
   ){
     static char* functionName = (char *)"_Switch_createNodeContext()";
     NODE_CONTEXT_ST* p;
     CciChar          buffer[256];
   ```
   b. Allocate a pointer to the local context and clear the context area:
   ```
     p = (NODE_CONTEXT_ST *)malloc(sizeof(NODE_CONTEXT_ST));

     if (p) {
        memset(p, 0, sizeof(NODE_CONTEXT_ST));
   ```
   c. Save the node object pointer in the context:
   ```
      p->nodeObject = nodeObject;
   ```
   d. Save the node name:
   ```
    CciCharNCpy((CciChar*)&p->nodeName, nodeName, MAX_NODE_NAME_LEN);
   ```
   e. Return the node context:
   ```
   return (CciContext*) p;
   ```
2. The broker calls the appropriate utility functions to find out about the node's input terminals and output terminals. A node has a number of input terminals and output terminals associated with it. Within the user function cniCreateNodeContext, calls must be made to cniCreateInputTerminal and cniCreateOutputTerminal to define the user node's terminals. These functions must be started within the cniCreateNodeContext implementation function. For example, to define a node with one input terminal and two output terminals:
   ```
   {
     const CciChar* ucsIn = CciString("in", BIP_DEF_COMP_CCSID) ;
     insInputTerminalListEntry(p, (CciChar*)ucsIn);
     free((void *)ucsIn) ;
   }
   {
   ```

```
    const CciChar* ucsOut = CciString("out", BIP_DEF_COMP_CCSID) ;
    insOutputTerminalListEntry(p, (CciChar*)ucsOut);
    free((void *)ucsOut) ;
  }
  {
    const CciChar* ucsFailure = CciString("failure", BIP_DEF_COMP_CCSID) ;
    insOutputTerminalListEntry(p, (CciChar*)ucsFailure);
    free((void *)ucsFailure) ;
  }
```

The previous code starts the insInputTerminalListEntry and insOutputTerminalListEntry functions. You can find these functions in the sample code `Common.c`; see "Sample node files" on page 6412. These functions define the terminals to the broker and store handles to the terminals. Handles are stored in the structure referenced by the value returned in CciContext*. The node can then access the terminal handles from within the other implementation functions (for example CciEvaluate) because CciContext is passed to those implementation functions.

The following code shows the definition of insInputTerminalListEntry:

```
TERMINAL_LIST_ENTRY *insInputTerminalListEntry(
  NODE_CONTEXT_ST* context,
  CciChar*         terminalName
){
  static char* functionName = (char *)"insInputTerminalListEntry()";
  TERMINAL_LIST_ENTRY* entry;
  int                  rc;

  entry = (TERMINAL_LIST_ENTRY *)malloc(sizeof(TERMINAL_LIST_ENTRY));
  if (entry) {

    /* This entry is the current end of the list */
    entry->next = 0;

    /* Store the terminal name */
    CciCharCpy(entry->name, terminalName);

    /* Create terminal and save its handle */
    entry->handle = cniCreateInputTerminal(&rc, context->nodeObject, (CciChar*)terminalName);

    /* Link an existing previous element to this one */
    if (context->inputTerminalListPrevious) context->inputTerminalListPrevious->next = entry;
    else if ((context->inputTerminalListHead) == 0) context->inputTerminalListHead = entry;

    /* Save the pointer to the previous element */
    context->inputTerminalListPrevious = entry;
  }
  else {
    /* Error: Unable to allocate memory */
  }

  return(entry);
}
```

The following example shows the code for insOutputTerminalListEntry:

```
TERMINAL_LIST_ENTRY *insOutputTerminalListEntry(
  NODE_CONTEXT_ST* context,
  CciChar*         terminalName
){
  static char* functionName = (char *)"insOutputTerminalListEntry()";
  TERMINAL_LIST_ENTRY* entry;
  int                  rc;

  entry = (TERMINAL_LIST_ENTRY *)malloc(sizeof(TERMINAL_LIST_ENTRY));
  if (entry) {
```

```
    /* This entry is the current end of the list */
    entry->next = 0;

    /* Store the terminal name */
    CciCharCpy(entry->name, terminalName);

    /* Create terminal and save its handle */
    entry->handle = cniCreateOutputTerminal(&rc, context->nodeObject, (CciChar*)terminalName);

    /* Link an existing previous element to this one */
    if (context->outputTerminalListPrevious) context->outputTerminalListPrevious->next = entry;
    else if ((context->outputTerminalListHead) == 0) context->outputTerminalListHead = entry;

    /* Save the pointer to the previous element */
    context->outputTerminalListPrevious = entry;
  }
  else {
    /* Error: Unable to allocate memory */
  }

  return(entry);
}
```

For the minimum code required to compile a C user-defined node, see "C skeleton code" on page 6683.

Setting attributes:
About this task

Attributes are set whenever you start the broker, or when you redeploy a message flow with new values. Attributes are set by the broker calling user code on the configuration thread. Your code needs to store these attributes in its node context area, for later use when processing messages.

Following the creation of input and output terminals, the broker calls the cniSetAttribute function to pass the values for the configured attributes for this instantiation of the user-defined node. For example:

```
{
  const CciChar* ucsAttr = CciString("nodeTraceSetting", BIP_DEF_COMP_CCSID) ;
  insAttrTblEntry(p, (CciChar*)ucsAttr, CNI_TYPE_INTEGER);
  _setAttribute(p, (CciChar*)ucsAttr, (CciChar*)constZero);
  free((void *)ucsAttr) ;
}
{
  const CciChar* ucsAttr = CciString("nodeTraceOutfile", BIP_DEF_COMP_CCSID) ;
  insAttrTblEntry(p, (CciChar*)ucsAttr, CNI_TYPE_STRING);
  _setAttribute(p, (CciChar*)ucsAttr, (CciChar*)constSwitchTraceLocation);
  free((void *)ucsAttr) ;
}
```

The number of configuration attributes that a node can have is unlimited. However, a node must not implement an attribute that is already implemented as a base configuration attribute. The following list shows base attributes:

- label
- userTraceLevel
- traceLevel
- userTraceFilter
- traceFilter

Implementing the node functionality:

About this task

When the broker retrieves a message from the queue, and that message arrives at the input terminal of your user-defined message processing or output node, the broker calls the implementation function cniEvaluate. This function is called on the message processing thread and it must decide what to do with the message. This function might be called on multiple threads, especially if additional instances are used.

Deleting an instance of the node:
About this task

If a node is deleted, the broker calls the cniDeleteNodeContext function. This function is started on the same thread as cniCreateNodeContext. Use this function to release resources used by your user-defined node. For example:

```
void _deleteNodeContext(
  CciContext* context
){
  static char* functionName = (char *)"_deleteNodeContext()";
  free ((void*) context);
  return;
}
```

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"User-defined extensions in the runtime environment" on page 2980
Before you design and implement user-defined extensions, familiarize yourself with the core components. Ensure that you also understand the basic WebSphere Message Broker runtime architecture.

"User-defined input nodes" on page 2990
A user-defined input node is an extension to the broker that provides a new input node in addition to the nodes supplied with the product.

"Developing user-defined extensions" on page 2970
A user-defined extension is a component that you design and implement to extend the function of WebSphere Message Broker.

"Why use a user-defined extension?" on page 2972
Use a user-defined node or parser when the built-in resources do not provide the required functions.

Related tasks:

"Extending the capability of a C message processing or output node" on page 3043
When you have created a user-defined message processing or output node in C, you can extend its capability.

"Implementing the supplied user-defined extension samples" on page 3017
WebSphere Message Broker provides some sample code to help you understand how to write user-defined nodes and parsers.

"Compiling a C user-defined extension" on page 3047
Compile user-defined extensions in C for all supported operating systems.

Related reference:

"C language user-defined node API" on page 6416
Learn about the different types of call provided by the API.

"cniCreateNodeFactory" on page 6449
Use this function to create a node factory in the broker. A single instance of the

named message flow node factory is created.

"cniDefineNodeClass" on page 6451
Use this function to define a node class, as specified by the *name* parameter, which is supported by the node factory specified as the*factoryObject* parameter.

"cniCreateNodeContext" on page 6447
This function creates a context for an instance of a node object. It is called by the broker whenever an instance of a node object is constructed. Nodes are constructed when a message flow is deployed by the broker, or when the execution group is started.

"cniDeleteNodeContext" on page 6454
This function deletes any context for an instance of a user-defined node object. It is called by the broker whenever an instance of a node object is destroyed, when a message flow is deleted, or when a configuration is redeployed.

"C node implementation functions" on page 6417
The user-defined node implements a function interface for the broker to call during runtime operation. This interface includes functions to create a local context whenever a node instance is created, functions to set and retrieve attribute values, the function to perform the processing of the node itself, and functions to examine messages.

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"C skeleton code" on page 6683
Use the skeleton code that is supplied as guidance for your C user-defined node. The code has the minimum content that is required to compile a user-defined node successfully.

"Sample node files" on page 6412
Several sample node files are provided on all platforms.

Extending the capability of a C message processing or output node:

When you have created a user-defined message processing or output node in C, you can extend its capability.

Before you begin

Before you start

Read the topic "Creating a message processing or output node in C" on page 3036.

About this task

After you have created a user-defined node, the following options are available:

1. "Accessing message data"
2. "Transforming a message object" on page 3044
3. "Accessing ESQL" on page 3045
4. "Propagating a message" on page 3045
5. "Writing to an output device" on page 3046

Accessing message data:

About this task

In many cases, the user-defined node must access the contents of the message that is received on its input terminal. The message is represented as a tree of syntax elements. Groups of utility functions are provided for message management, message buffer access, syntax element navigation, and syntax element access. (See "C node utility functions" on page 6419 for details of the utility functions.)

The types of query that you are likely to want to perform include:

- Obtaining the root element of the required message object
- Accessing the bit stream representation of an element tree
- Navigating or querying the tree by asking for child or sibling elements by name
- Getting the type of the element
- Getting the value of the element

For example, to query the name and type of the first child of body:

```
void cniEvaluate( ...
){
  ...
/* Navigate to the target element */
  rootElement = cniRootElement(&rc, message);
  bodyElement = cniLastChild(&rc, rootElement);
  bodyFirstChild = cniFirstChild(&rc, bodyElement);

/* Query the name and value of the target element */
  cniElementName(&rc, bodyFirstChild, (CciChar*)&elementname, sizeof(elementName));
  bytes = cniElementCharacterValue(
  &rc, bodyfirstChild, (CciChar*)&eValue, sizeof(eValue));
  ...
}
```

To access the bit stream representation of an element tree you can use the cniElementAsBitstream function. Using this function, you can obtain the bit stream representation of any element in a message. See "cniElementAsBitstream" on page 6458 for details of how to use this function, and sample code.

Transforming a message object:
About this task

The received input message is read-only, therefore before a message can be transformed, you must write it to a new output message using the cniCreateMessage function. You can copy elements from the input message, or you can create new elements and attach them to the message. New elements are typically in a parser's domain.

For example:

1. To write the incoming message to a new message:

   ```
   {
     ...
     context = cniGetMessageContext(&rc, message));
     outMsg = cniCreateMessage(&rc, context));
     ...
   }
   ```

2. To make a copy of the new message:

   ```
   cniCopyElementTree(&rc, sourceElement, targetElement);
   ```

3. To modify the value of a target element:

```
cniSetElementIntegerValue(&rc, targetElement, L"newValue", 8);
```

4. After finalizing and propagating the message, you must delete the output message using the cniDeleteMessage function:

```
cniDeleteMessage(&rc, outMsg);
```

As part of the transformation, you might want to create a new message body. To create a new message body, use one of the following functions, which assign a parser to a message tree folder:

```
cniCreateElementAsFirstChildUsingParser
cniCreateElementAsLastChildUsingParser
cniCreateElementAfterUsingParser
cniCreateElementBeforeUsingParser
```

When creating a message body, do not use the following functions because they do not associate an owning parser with the folder:

```
cniCreateElementAsFirstChild
cniCreateElementAsLastChild
cniCreateElementAfter
cniCreateElementBefore
```

Accessing ESQL:
About this task

Nodes can invoke ESQL expressions using Compute node ESQL syntax. You can create and modify the components of the message using ESQL expressions, and you can refer to elements of both the input message and data from an external database using the cniSqlCreateStatement, cniSqlSelect, cniSqlDeleteStatement, and cniSqlExecute functions.

For example, to populate the Result element from the contents of a column in a database table:

```
{
  ...
  sqlExpr = cniSqlCreateStatement(&rc,
   (NODE_CONTEXT_ST *)context->nodeObject,
   L"DB", CCI_SQL_TRANSACTION_AUTO,
   L"SET OutputRoot.XMLNS.Result[] = (SELECT T.C1 AS Col1 FROM Database.TABLE AS T;");
  ...
  cniSqlSelect(&rc, sqlExpr, localEnvironment, exceptionList, message, outMsg);
  cniSqlDeleteStatement(&rc, sqlExpr);
  ...
}
```

For more information about ESQL, see “ESQL overview” on page 2371.

If your user-defined node primarily uses ESQL, consider using a “Compute node” on page 4340.

Propagating a message:
About this task

Before you propagate a message, decide what message flow data you want to propagate, and which terminal is to receive the data.

1. If the message has changed, finalize the message before you propagate it using the cniFinalize function. For example:

```
cniFinalize(&rc, outMsg, CCI_FINALIZE_NONE);
```

2. The terminalObject is derived from a list that the user-defined node maintains itself. To propagate the message to the output terminal, use the cniPropagate function:

```
if (terminalObject) {
  if (cniIsTerminalAttached(&rc, terminalObject)) {
    if (rc == CCI_SUCCESS) {
      cniPropagate(&rc, terminalObject, localEnvironment, exceptionList, outMsg);
    }
  }
```

   In the above example, the cniIsTerminalAttached function is used to test whether the message can be propagated to the specified terminal. If you do not use the cniIsTerminalAttached function and the terminal is not attached to another node by a connector, the message is not propagated and no warning message is returned. Use the cniIsTerminalAttached function to prevent this error occurring.

3. If you created a new output message using cniCreateMessage, after propagating the message, delete the output message using the cniDeleteMessage function:

```
cniDeleteMessage(&rc, outMsg);
```

Writing to an output device:
About this task

A transformed message must be serialized to a bit stream; a message can be serialized only once.

The bit stream can then be accessed and written to an output device. Write the message to a bit stream using the cniWriteBuffer function. For example:

```
{
  ...
  cniWriteBuffer(&rc, message);
  writeToDevice(cniBufferPointer(&rc, message), cniBufferSize(&rc, message));
  ...
}
```

In this example, the method writeToDevice is a user-written method which writes a bit stream to an output device.

Do not write a user-defined output node to write messages to WebSphere MQ queues; use the supplied MQOutput node in this scenario. The broker internally maintains a WebSphere MQ connection and open queue handles on a thread-by-thread basis, and these are cached to optimize performance. In addition, the broker handles recovery scenarios when certain WebSphere MQ events occur; this recovery would be adversely affected if WebSphere MQ MQI calls are used in a user-defined output node.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"User-defined extensions in the runtime environment" on page 2980
Before you design and implement user-defined extensions, familiarize yourself with the core components. Ensure that you also understand the basic WebSphere Message Broker runtime architecture.

"User-defined input nodes" on page 2990
A user-defined input node is an extension to the broker that provides a new input

node in addition to the nodes supplied with the product.

"Developing user-defined extensions" on page 2970
A user-defined extension is a component that you design and implement to extend the function of WebSphere Message Broker.

"Why use a user-defined extension?" on page 2972
Use a user-defined node or parser when the built-in resources do not provide the required functions.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

"Implementing the supplied user-defined extension samples" on page 3017
WebSphere Message Broker provides some sample code to help you understand how to write user-defined nodes and parsers.

"Compiling a C user-defined extension"
Compile user-defined extensions in C for all supported operating systems.

Related reference:

"C language user-defined node API" on page 6416
Learn about the different types of call provided by the API.

Compiling a C user-defined extension:

Compile user-defined extensions in C for all supported operating systems.

Before you begin

Before you start:

If you create your own user-defined nodes, parsers, and user exits in C, compile them on the operating system on which the target broker is running. Samples are provided for both nodes and parsers, and are described in "Sample node files" on page 6412 and "Sample parser files" on page 6414. Use the instructions here to compile the samples. If you want to create your own extensions, see the following topics:

- "Creating a user-defined extension in C" on page 3026
- "Implementing a user-defined parser" on page 3099
- "Implementing a user-defined exit" on page 3113

About this task

These instructions use the file names of the supplied samples. If you are compiling your own user-defined extensions, substitute your own file names.

When you compile a user-defined extension that is written in C, you need a compatible compiler. For details of supported compilers, see "Optional software support" on page 3603.

Header files:
About this task

The following header files define the C interfaces:

BipCni.h
: Message processing nodes

BipCpi.h
Message parsers

BipCci.h
Interfaces common to both nodes and parsers

BipCos.h
Platform-specific definitions

Compiling:
About this task

Compile the source for your user-defined extension on each of the supported operating systems to create the executable file that the broker calls to implement your user-defined extension. On Linux, UNIX, and z/OS systems, this file is a loadable implementation library (LIL) file; on Windows systems, it is a dynamic load library (DLL) file.

The libraries that you build to contain user-defined nodes or parsers must have the extension `.lil` on all operating systems so that the broker can load them. Libraries that contain user exits must have the extension `.lel` on all operating systems. The examples in this topic show libraries with the extension `.lil`

Refer to the documentation for the compiler that you are using for full details of available compile and link options that might be required for your programs.

Navigate to the directory where your user-defined extension source code is located, and follow the instructions for your operating system:
- AIX
- HP-Itanium
- Linux
- Solaris
- Windows
- z/OS

Compiling on AIX:
About this task

When you compile a user-defined extension that is written in C, use a supported compiler.

The following instructions are for compiling an extension for a 64-bit execution group; 32-bit execution groups are not supported.

```
xlc_r -q64 \
   -I. \
   -I/install_dir/include/plugin \
   -c SwitchNode.c \
   -o SwitchNode.o

xlc_r -q64 \
   -I. \
   -I/install_dir/include/plugin \
   -c TransformNode.c \
   -o TransformNode.o

xlc_r -q64 \
   -I. \
   -I/install_dir/include/plugin \
   -c BipSampPluginUtil.c \
   -o BipSampPluginUtil.o
```

```
xlc_r -q64 \
   -I. \
   -I/install_dir/include/plugin \
   -c Common.c \
   -o Common.o

xlc_r -q64 \
   -I. \
   -I/install_dir/include/plugin \
   -c NodeFactory.c \
   -o NodeFactory.o

xlc_r -q64 \
      -qmkshrobj \
      -bM:SRE \
      -bexpall \
      -bnoentry \
      -o SwitchNode.lil SwitchNode.o \
         BipSampPluginUtil.o Common.o NodeFactory.o \
      -L /install_dir/lib \
      -l imbdfplg

chmod a+r SwitchNode.lil
```

Compiling on HP-Itanium:
About this task

When you compile a user-defined extension that is written in C, use a supported compiler.

The following instructions are for compiling an extension for a 64-bit execution group; 32-bit execution groups are not supported.

```
cc +z +e +DD64 -D_HPUX_SOURCE -DTHREADS -D_REENTRANT -Ae \
   -I. \
   -I/install_dir/include \
   -I/install_dir/include/plugin \
   -c BipSampPluginUtil.c \
   -o output_dir/BipSampPluginUtil.o

cc +z +e +DD64 -D_HPUX_SOURCE -DTHREADS -D_REENTRANT -Ae \
   -I. \
   -I/install_dir/include \
   -I/install_dir/include/plugin \
   -c Common.c \
   -o output_dir/Common.o

cc +z +e +DD64 -D_HPUX_SOURCE -DTHREADS -D_REENTRANT -Ae \
   -I. \
   -I/install_dir/include \
   -I/install_dir/include/plugin \
   -c NodeFactory.c \
   -o output_dir/NodeFactory.o

cc +z +e +DD64 -D_HPUX_SOURCE -DTHREADS -D_REENTRANT -Ae \
   -I. \
   -I/install_dir/include \
   -I/install_dir/include/plugin \
   -c SwitchNode.c \
   -o output_dir/SwitchNode.o

cc +z +e +DD64 -D_HPUX_SOURCE -DTHREADS -D_REENTRANT -Ae \
   -I. \
   -I/install_dir/include \
   -I/install_dir/include/plugin \
```

```
   -c TransformNode.c \
   -o output_dir/TransformNode.o


ld -b \
   -o output_dir/SwitchNode.lil \
   output_dir/BipSampPluginUtil.o \
   output_dir/Common.o \
   output_dir/NodeFactory.o \
   output_dir/SwitchNode.o \
   output_dir/TransformNode.o \
   -L install_dir/lib \
   -L install_dir/xml4c/lib \
   -L install_dir/merant/lib \
   -L install_dir/jre16/lib/IA64N\
   -L install_dir/jre16/lib/IA64N/server \
   -l imbdfplg

chmod a+r output_dir/SwitchNode.lil
```

Compiling on Linux:
About this task

When you compile a user-defined extension that is written in C, use a supported compiler.

When you compile programs on Linux on POWER, replace the option `-fpic` with `-fPIC` if you want to use dynamic linking and avoid any limit on the size of the global offset table.

The following instructions are for compiling an extension for a 64-bit execution group on Linux on x86-64, Linux on POWER, and Linux on IBM z Systems. 32-bit execution groups are not supported on those platforms. To compile the extension for a 32-bit execution group on Linux on x86, replace `-m64` with `-m32` in the compile and link examples.

```
g++ -c -O2 -m64 -Wall -Wno-format-y2k -fpic \
   -I. \
   -I/install_dir/include \
   -I/install_dir/include/plugin \
   -D_LINUX -D_THREADS -D_POSIX_PTHREAD_SEMANTICS -D_REENTRANT \
   TransformNode.c

g++ -c -O2 -m64 -Wall -Wno-format-y2k -fpic \
   -I. \
   -I/install_dir/include \
   -I/install_dir/include/plugin \
   -D_LINUX -D_THREADS -D_POSIX_PTHREAD_SEMANTICS -D_REENTRANT \
   SwitchNode.c

g++ -c -O2 -m64 -Wall -Wno-format-y2k -fpic \
   -I. \
   -I/install_dir/include \
   -I/install_dir/include/plugin \
   -D_LINUX -D_THREADS -D_POSIX_PTHREAD_SEMANTICS -D_REENTRANT \
   BipSampPluginUtil.c

g++ -c -O2 -m64 -Wall -Wno-format-y2k -fpic \
   -I. \
   -I/install_dir/include \
   -I/install_dir/include/plugin \
   -D_LINUX -D_THREADS -D_POSIX_PTHREAD_SEMANTICS -D_REENTRANT \
   Common.c

g++ -c -O2 -m64 -Wall -Wno-format-y2k -fpic \
```

```
   -I. \
   -I/install_dir/include \
   -I/install_dir/include/plugin \
   -D_LINUX -D_THREADS -D_POSIX_PTHREAD_SEMANTICS -D_REENTRANT \
   NodeFactory.c

g++ -m64 -O2 -o samples.lil \
   TransformNode.o \
   SwitchNode.o \
   BipSampPluginUtil.o \
   Common.o NodeFactory.o \
   -shared -lc -lnsl -ldl \
   -L/install_dir/lib -limbdfplg
```

These commands create the file `samples.lil` that provides TransformNode and SwitchNode objects.

Compiling on Solaris:
About this task

When you compile a user-defined extension that is written in C, use a supported compiler.

The following instructions are for compiling an extension for a 64-bit execution group on Solaris on SPARC; 32-bit execution groups are not supported. To compile the extension for a default 64-bit execution group on Solaris on x86-64, replace `-xarch=v9` with `-xarch=amd64` in the compile examples; 32-bit execution groups are not supported.

```
cc -xarch=v9 -mt \
   -I. \
   -I/install_dir/include \
   -I/install_dir/include/plugin \
   -c SwitchNode.c \
   -o output_dir/SwitchNode.o

cc -xarch=v9 -mt \
   -I. \
   -I/install_dir/include \
   -I/install_dir/include/plugin \
   -c TransformNode.c \
   -o output_dir/TransformNode.o

cc -xarch=v9 -mt \
   -I. \
   -I/install_dir/include \
   -I/install_dir/include/plugin \
   -c BipSampPluginUtil.c \
   -o output_dir/BipSampPluginUtil.o

cc -xarch=v9 -mt \
   -I. \
   -I/install_dir/include \
   -I/install_dir/include/plugin \
   -c NodeFactory.c \
   -o output_dir/NodeFactory.o

cc -xarch=v9 -mt \
   -I. \
   -I/install_dir/include \
   -I/install_dir/include/plugin \
   -c Common.c \
   -o output_dir/Common.o

cc -xarch=v9  -xcode=pic32 -mt -G \
```

```
   -o output_dir/SwitchNode.lil \
      output_dir/SwitchNode.o \
      output_dir/BipSampPluginUtil.o \
      output_dir/NodeFactory.o \
      output_dir/Common.o \
   -L /install_dir/lib \
   -l imbdfplg

chmod a+r output_dir/SwitchNode.lil
```

Compiling on Windows:
About this task

When you compile a user-defined extension that is written in C, use a supported compiler.

Ensure that you include a space between `SwitchNode.c` and `BipSampPluginUtil.c`, and also between `-link` and `/DLL`.

Enter the command as a single line of input; in the following example the lines are split to improve readability.

```
cl /VERBOSE /LD /MD /Zi /EHsc /I.
   /Iinstall_dir\include\plugin
   SwitchNode.c BipSampPluginUtil.c Common.c
   NodeFactory.c TransformNode.c
   -link /DLL install_dir\lib\imbdfplg.lib
   /OUT:SwitchNode.lil
```

If you have correctly set the *LIB* environment variable, you do not have to specify the full paths to the LIB files.

Compiling on z/OS:
About this task

When you compile a user-defined extension that is written in C, use a supported compiler.

Force your link to use prelinker or linker by setting the `_CC_STEPS` variable to `-1`:

```
export _CC_STEPS=-1
```

Alternatively, add these two lines to your makefile to export it:

```
_CC_STEPS=-1
.EXPORT : _CC_STEPS
```

To create optimized builds, use **-2** in place of **-g** in the following commands:

```
cc -c \
 -Wc,LP64 -g -W0,langlvl\(extended\),EXPORTALL,float\ieee\) \
 -Wc,xplink \
 -W0,LIST\(./SwitchNode.lst\) \
 -I. -I${install_dir}/include \
 -I${install_dir}/include/plugin \
 -I${install_dir}/sample/include \
 -I${install_dir}/sample/plugin \
 -o ./SwitchNode.o ./SwitchNode.c

cc -c \
 -Wc,LP64 -g -W0,langlvl\(extended\),EXPORTALL,float\ieee\) \
 -Wc,xplink \
 -W0,LIST\(./TransformNode.lst\) \
 -I. -I${install_dir}/include \
```

```
 -I${install_dir}/include/plugin \
 -I${install_dir}/sample/include \
 -I${install_dir}/sample/plugin \
 -o ./SwitchNode.o ./TransformNode.c

cc -c \
 -Wc,LP64 -g -W0,langlvl\(extended\),EXPORTALL,float\ieee\) \
 -Wc,xplink \
 -W0,LIST\(./BipSampPluginUtil.lst\) \
 -I. -I${install_dir}/include \
 -I${install_dir}/include/plugin \
 -I${install_dir}/sample/include \
 -I${install_dir}/sample/plugin \
 -o ./BipSampPluginUtil.o ./BipSampPluginUtil.c

cc -c \
 -Wc,LP64 -g -W0,langlvl\(extended\),EXPORTALL,float\ieee\) \
 -Wc,xplink \
 -W0,LIST\(./Common.lst\) \
 -I. -I${install_dir}/include \
 -I${install_dir}/include/plugin \
 -I${install_dir}/sample/include \
 -I${install_dir}/sample/plugin \
 -o ./Common.o ./Common.c

cc -c \
  -Wc,LP64 -g -W0,langlvl\(extended\),EXPORTALL,float\ieee\) \
 -Wc,xplink \
 -W0,LIST\(./NodeFactory.lst\) \
 -I. -I${install_dir}/include \
 -I${install_dir}/include/plugin \
 -I${install_dir}/sample/include \
 -I${install_dir}/sample/plugin \
 -o ./NodeFactory.o ./NodeFactory.c

cc \
 -Wl,DLL,LP64 -g -Wl,p,map -Wl,LIST=ALL,MAP,XREF,REUS=RENT \
 -Wl,xplink \
 -o ./SwitchNode.lil ./SwitchNode.o ./BipSampPluginUtil.o \
 ./Common.o ./NodeFactory.o \
 ${install_dir}/lib/libimbdfplg.x
```

Run the following command to set the file permissions of the user-defined extension to `group read` and to be executable:

```
chmod a+rx {output_dir}/SwitchNode.lil
```

Related concepts:

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

"Developing user-defined extensions" on page 2970
A user-defined extension is a component that you design and implement to extend the function of WebSphere Message Broker.

Related tasks:

"Creating a user-defined extension in C" on page 3026
You must complete a series of tasks to create user-defined extensions that use the C language.

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

"Implementing a user-defined exit" on page 3113
You can develop and deploy a user-defined exit.

"Creating message catalogs" on page 3138
Create your own message catalogs to write tailored entries to the local error log.

"Creating the user interface representation of a user-defined node in the WebSphere Message Broker Toolkit" on page 3079
When you are developing a user-defined node in Java or C only, you must create the user interface representation of the node in the WebSphere Message Broker Toolkit.

"Testing a user-defined node" on page 3094
When you have created and installed the required resources, you can test your user-defined node.

"Packaging and distributing a user-defined node project" on page 3121
Export the user-defined node project to make it available for other users.

Related reference:

"User-defined extensions" on page 6411
Reference material that supports the creation and management of your user-defined extensions.

"Support for 32-bit and 64-bit platforms" on page 3589
WebSphere Message Broker operates in 32-bit mode or 64-bit mode, on supported operating systems.

"Optional software support" on page 3603
The products listed here are not required, but might be useful. Except where stated, these products are not supplied with WebSphere Message Broker.

Creating a user-defined extension in Java:

You must complete a series of tasks to create user-defined nodes that use the Java language.

About this task

Complete the appropriate tasks from the following list:

- "Creating an input node in Java" on page 3055
- "Creating a message processing or output node in Java" on page 3062
- "Compiling a Java user-defined node" on page 3074
- "Packaging a Java user-defined node" on page 3118

You can write only user-defined nodes in Java: you must write user-defined parsers in C.

What to do next

When you have completed this set of tasks, continue with the following tasks:

- "Creating the user interface representation of a user-defined node in the WebSphere Message Broker Toolkit" on page 3079
- "Testing a user-defined node" on page 3094
- "Packaging and distributing user-defined extensions" on page 3117

Related concepts:

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related reference:

"User-defined extensions" on page 6411
Reference material that supports the creation and management of your user-defined extensions.

Restrictions when creating Java nodes:

In Java user-defined nodes and the JavaCompute node, calling the System.exit(...) method is not supported. Calling this method results in a `SecurityException`.

Related concepts:

"User-defined extensions in the runtime environment" on page 2980
Before you design and implement user-defined extensions, familiarize yourself with the core components. Ensure that you also understand the basic WebSphere Message Broker runtime architecture.

"User-defined input nodes" on page 2990
A user-defined input node is an extension to the broker that provides a new input node in addition to the nodes supplied with the product.

"Developing user-defined extensions" on page 2970
A user-defined extension is a component that you design and implement to extend the function of WebSphere Message Broker.

"Why use a user-defined extension?" on page 2972
Use a user-defined node or parser when the built-in resources do not provide the required functions.

Related tasks:

"Creating an input node in Java"
An input node is used to receive a message into a message flow, typically from a source that is not supported by the built-in input nodes.

"Creating a message processing or output node in Java" on page 3062
A message processing node is used to process a message, and an output node is used to output a message as a bit stream.

Related reference:

"Exception list structure" on page 4224
An exception list contains information about exceptions, such as error numbers, the name of the node that generated the exception, and the reason for the exception.

Related information:

Java user-defined extensions API

Creating an input node in Java:

An input node is used to receive a message into a message flow, typically from a source that is not supported by the built-in input nodes.

Before you begin

Before you start

Read the following topics:

- "Why use a user-defined extension?" on page 2972
- "User-defined input nodes" on page 2990

About this task

To create an input node in the Java language:

1. "Creating a Java project" on page 3056
2. "Declaring the input node class" on page 3056
3. "Defining the node constructor" on page 3057
4. "Receiving external data into a buffer" on page 3057

5. "Propagating the message" on page 3058
6. "Controlling threading and transactionality" on page 3058
7. "Declaring the node name" on page 3059
8. "Declaring attributes" on page 3060
9. "Implementing the node functionality" on page 3060
10. "Overriding default message parser attributes (optional)" on page 3060
11. "Deleting an instance of the node" on page 3061

A Java user-defined node is distributed as a `.jar` file.

Creating a Java project:
About this task

Before you can create Java nodes in the WebSphere Message Broker Toolkit, you must create a Java project:

Procedure

1. Click **File** > **New** > **Project**, select **Java Project** then click **Next**.
2. Give the project a name, then click **Next**.
3. On the Java Settings pane, select the **Libraries** tab, and click **Add External JARs**.
4. Select *install_dir*`\classes\jplugin2.jar`, where *install_dir* is the home directory of your WebSphere Message Broker installation.
5. Follow the prompts on the other tabs to define any other build settings.
6. Click **Finish**.

What to do next

You can now develop the source for your Java node in this project.

Declaring the input node class:
About this task

Every class that implements MbInputNodeInterface and is contained in the broker LIL path is registered with the broker as an input node. When you implement MbInputNodeInterface, you must also implement a run method for this class. The run method represents the start of the message flow, contains the data that formulates the message, and propagates it down the flow. The broker calls the run method when threads become available in accordance with your specified threading model.

The class name must end with the word "Node". For example, if the name is BasicInput in the WebSphere Message Broker Toolkit, the class name must be BasicInputNode.

For example, to declare the input node class:

```
package com.ibm.jplugins;

import com.ibm.broker.plugin.*;

public class BasicInputNode extends MbInputNode implements MbInputNodeInterface
{
...
```

Follow these steps to complete this action in the WebSphere Message Broker Toolkit:

Procedure

1. Click **File** > **New** > **Class**.
2. Set the package and class name fields to appropriate values.
3. Delete the text in the **Superclass** text field and click the **Browse** button.
4. Select **MbInputNode**.
5. Click the **Add** button next to Interfaces text field, and select **MbInputNodeInterface**.
6. Click **Finish**.

Defining the node constructor:
About this task

When the node is instantiated, the constructor of the node class is called. This class is where you create the terminals of the node, and initialize the default values for the attributes.

An input node has a number of output terminals associated with it, but does not typically have any input terminals. Use the createOutputTerminal method to add output terminals to a node when the node is instantiated. For example, to create a node with three output terminals:

```
public BasicInputNode() throws MbException
{
 createOutputTerminal ("out");
 createOutputTerminal ("failure");
 createOutputTerminal ("catch");
 setAttribute ("firstParserClassName","myParser");
 attributeVariable  = "none";
}
```

Receiving external data into a buffer:
About this task

An input node can receive data from any type of external source, such as a file system, a queue, or a database, in the same way as all other Java programs, if the output from the node is in the correct format.

Provide an input buffer (or bit stream) to contain input data, and associate it with a message object. Create a message from a byte array by using the createMessage method of the **MbInputNode** class, and then generate a valid message assembly from this message. For example, to read the input data from a file:

Procedure

1. Create an input stream to read from the file:
   ```
   FileInputStream inputStream = new FileInputStream("myfile.msg");
   ```
2. Create a byte array the size of the input file:
   ```
   byte[] buffer = new byte[inputStream.available()];
   ```
3. Read from the file into the byte array:
   ```
   inputStream.read(buffer);
   ```
4. Close the input stream:
   ```
   inputStream.close();
   ```
5. Create a message to put on the queue:

```
MbMessage msg = createMessage(buffer);
```

6. Create a message assembly to hold this message:

```
msg.finalizeMessage(MbMessage.FINALIZE_VALIDATE);
MbMessageAssembly newAssembly =
     new MbMessageAssembly(assembly, msg);
```

Propagating the message:
About this task

After creating a message assembly, you can propagate it to one of the output terminals that are defined on the node.

For example, to propagate the message assembly to the terminal named **out**:

```
MbOutputTerminal out = getOutputTerminal("out");
out.propagate(newAssembly);
```

To delete the message:

```
msg.clearMessage();
```

To clear the memory that is allocated for the message tree, call the clearMessage() function within the finally block of `try/catch`.

Controlling threading and transactionality:
About this task

The broker infrastructure handles transaction issues such as controlling the commit of a WebSphere MQ or database unit of work when message processing has completed. However, resources modified from within a user-defined node are not necessarily under the transactional control of the broker.

Each message flow thread is allocated from a pool of threads maintained for each message flow, and starts in the run method.

The user-defined node uses return values to indicate whether a transaction is successful, to control whether transactions are committed or rolled back, and to control when the thread is returned to the pool. The broker infrastructure catches all unhandled exceptions, and rolls back the transaction.

You determine the behavior of transactions and threads by using the appropriate return value:

MbInputNode.SUCCESS_CONTINUE
The transaction is committed and the broker calls the run method again by using the same thread.

MbInputNode.SUCCESS_RETURN
The transaction is committed and the thread is returned to the thread pool, assuming that it is not the only thread for this message flow.

MbInputNode.FAILURE_CONTINUE
The transaction is rolled back and the broker calls the run method again by using the same thread.

MbInputNode.FAILURE_RETURN
The transaction is rolled back and the thread is returned to the thread pool, assuming that it is not the only thread for this message flow.

MbInputNode.TIMEOUT
The run method must not block indefinitely while waiting for input data to arrive. While the flow is blocked by user code, you cannot shut down or reconfigure the broker. The run method must yield control to the broker periodically by returning from the run method. If input data is not received after a certain period (for example, 5 seconds), the method must return with the TIMEOUT return code. Assuming that the broker does not need to reconfigure or shut down, the run method of the input node gets called again immediately.

To create multithreaded message flows, you call the dispatchThread method after a message is created, but before the message is propagated to an output terminal. This action ensures that only one thread is waiting for data while other threads are processing the message. New threads are obtained from the thread pool up to the maximum limit specified by the `Additional Instances` property of the message flow. For example:

```
public int run( MbMessageAssembly assembly ) throws MbException
{
  byte[] data = getDataWithTimeout();  // user supplied method
                                       // returns null if timeout
  if( data == null )
    return TIMEOUT;

  MbMessage msg = createMessage( data );
  msg.finalizeMessage( MbMessage.FINALIZE_VALIDATE );
  MbMessageAssembly newAssembly =
       new MbMessageAssembly( assembly, msg );

  dispatchThread();

  getOutputTerminal( "out" ).propagate( newAssembly );

  return SUCCESS_RETURN;
}
```

Declaring the node name:
About this task

You must declare the name of the node for use and identification by the WebSphere Message Broker Toolkit. All node names must end with the characters "Node". Declare the name by using the following method:

```
public static String getNodeName()
{
   return "BasicInputNode";
}
```

If this method is not declared, the Java API framework creates a default node name by using the following rules:

- The class name is appended to the package name.
- The periods are removed, and the first letter of each part of the package and class name is capitalized.

For example, by default, the following class is assigned the node name "ComIbmPluginsamplesBasicInputNode":

```
package com.ibm.pluginsamples;
public class BasicInputNode extends MbInputNode implements MbInputNodeInterface
{
   ...
```

Declaring attributes:
About this task

Declare node attributes by using the same method that you use for Java bean properties. You are responsible for writing get and set methods for the attributes; the API framework infers the attribute names by using the Java bean introspection rules. For example, if you declare the following two methods:

```
private String attributeVariable;

public String getFirstAttribute()
{
  return attributeVariable;
}

public void setFirstAttribute(String value)
{
  attributeVariable = value;
}
```

The broker infers that this node has an attribute called firstAttribute. This name is derived from the names of the get or set methods, not from the variable names of any internal class members. Attributes can be exposed only as strings, so convert numeric types to and from strings in the get or set methods. For example, the following method defines an attribute called timeInSeconds:

```
int seconds;

public String getTimeInSeconds()
{
  return Integer.toString(seconds);
}

public void setTimeInSeconds(String value)
{
  seconds = Integer.parseInt(value);
}
```

Implementing the node functionality:
About this task

As already described, the run method is called by the broker to create the input message. This method must provide all the processing function for the input node.

Overriding default message parser attributes (optional):
About this task

An input node implementation normally determines which message parser initially parses an input message. For example, the built-in MQInput node dictates that an MQMD parser is required to parse the MQMD header. A user-defined input node can select an appropriate header or message parser, and the mode in which the parsing is controlled, by using the following default attributes that are included, which you can override:

rootParserClassName
Defines the name of the root parser that parses message formats supported by the user-defined input node. It defaults to `GenericRoot`, a supplied root parser that causes the broker to allocate and chain parsers together. It is unlikely that a node would have to modify this attribute value.

firstParserClassName
Defines the name of the first parser, in what might be a chain of parsers that are responsible for parsing the bit stream. It defaults to `XML`.

messageDomainProperty
An optional attribute that defines the name of the message parser required to parse the input message. The supported values are the same as the values that are supported by the MQInput node.

messageSetProperty
An optional attribute that defines the message set identifier, or the message set name, in the `Message Set` field, only if the MRM parser was specified by the `messageDomainProperty` attribute.

messageTypeProperty
An optional attribute that defines the identifier of the message in the `MessageType` field, only if the MRM parser was specified by the `messageDomainProperty` attribute.

messageFormatProperty
An optional attribute that defines the format of the message in the `Message Format` field, only if the MRM parser was specified by the `messageDomainProperty` attribute.

Deleting an instance of the node:
About this task

An instance of the node is deleted when either:

- You shut down the broker.
- You remove the node or the message flow that contains the node, and redeploy the configuration.

When the node is deleted, it can perform cleanup operations, such as closing sockets, if it implements the optional onDelete method. This method, if present, is called by the broker just before the node is deleted.

Implement the onDelete method as follows:

```
public void onDelete()
{
  // perform node cleanup if necessary
}
```

Related concepts:

"User-defined extensions in the runtime environment" on page 2980
Before you design and implement user-defined extensions, familiarize yourself with the core components. Ensure that you also understand the basic WebSphere Message Broker runtime architecture.

"User-defined input nodes" on page 2990
A user-defined input node is an extension to the broker that provides a new input node in addition to the nodes supplied with the product.

"Developing user-defined extensions" on page 2970
A user-defined extension is a component that you design and implement to extend the function of WebSphere Message Broker.

"Why use a user-defined extension?" on page 2972
Use a user-defined node or parser when the built-in resources do not provide the required functions.

Related tasks:

"Restrictions when creating Java nodes" on page 3055
In Java user-defined nodes and the JavaCompute node, calling the System.exit(...) method is not supported. Calling this method results in a `SecurityException`.

"Implementing the supplied user-defined extension samples" on page 3017
WebSphere Message Broker provides some sample code to help you understand how to write user-defined nodes and parsers.

"Compiling a Java user-defined node" on page 3074
When you have created the code for your Java user-defined node, you must compile it for your operating system.

Related reference:

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

"Exception list structure" on page 4224
An exception list contains information about exceptions, such as error numbers, the name of the node that generated the exception, and the reason for the exception.

Related information:

Java user-defined extensions API

Creating a message processing or output node in Java:

A message processing node is used to process a message, and an output node is used to output a message as a bit stream.

Before you begin

Before you start

Read the following topics:

- "Why use a user-defined extension?" on page 2972
- "User-defined message processing nodes" on page 2996
- "User-defined output nodes" on page 3006
- "Restrictions when creating Java nodes" on page 3055

About this task

WebSphere Message Broker provides the source for two sample user-defined nodes called SwitchNode and TransformNode. You can use these nodes in their current state, or you can modify them.

When you code a message processing node or an output node, the two types provide essentially the same functions. You can perform message processing within an output node, and likewise you can propagate an output message to a bit stream from a message processing node. For simplicity, this topic refers mainly to the node as a message processing node, but it does discuss the functionality of both types of node.

Complete the following steps:

1. "Creating a new Java project" on page 3063
2. "Declaring the message processing node class" on page 3063
3. "Defining the node constructor" on page 3064

4. “Accessing message data” on page 3064
5. “Transforming a message object” on page 3064
6. “Propagating the message” on page 3066
7. “Declaring the node name” on page 3066
8. “Declaring attributes” on page 3067
9. “Implementing the node functionality” on page 3067
10. “Deleting an instance of the node” on page 3068

A Java user-defined node is distributed as a `.jar` file.

Creating a new Java project:
About this task

Before you can create Java nodes in the WebSphere Message Broker Toolkit, you must create a new Java project:

1. Click **File** > **New** > **Project**. Select **Java** and click **Next**.
2. In the **Project name** field, enter a project a name, then click **Next**.
3. On the Java Settings pane, select the **Libraries** tab, and click **Add External JARs**.
4. Select install_dir\classes\jplugin2.jar.
5. Follow the prompts on the other tabs to define any other build settings.
6. Click **Finish**.

You can now develop the source for your Java node within this project.

Declaring the message processing node class:
About this task

Any class that implements MbNodeInterface, and is contained in the broker's LIL path, is registered with the broker as a message processing node. When you implement MbNodeInterface, you must also implement an evaluate method for this class. The evaluate method is called by the broker for each message that passes through the flow.

The class name must end with the word "Node". For example, if you have assigned the name as Basic in the WebSphere Message Broker Toolkit, the class name must be BasicNode.

For example, to declare the message processing node class:

```
package com.ibm.jplugins;

import com.ibm.broker.plugin.*;

public class BasicNode extends MbNode implements MbNodeInterface
```

Declare the class in the WebSphere Message Broker Toolkit:

1. Click **File** > **New** > **Class**.
2. Set the package and class name fields to appropriate values.
3. Delete the text in the Superclass text field and click **Browse**.
4. Select **MbNode** and click **OK**.
5. Click the **Add** button next to Interfaces text field, and select **MbNodeInterface**.
6. Click **Finish**.

Defining the node constructor:
About this task

When the node is instantiated, the constructor of the user's node class is called. Create the terminals of the node, and initialize any default values for attributes in this constructor.

A message processing node has a number of input terminals and output terminals that are associated with it. Use the methods createInputTerminal and createOutputTerminal to add terminals to a node when the node is instantiated.

For example, to create a node with one input terminal and two output terminals:

```
public MyNode() throws MbException
{
  // create terminals here
  createInputTerminal ("in");
  createOutputTerminal ("out");
  createOutputTerminal ("failure");
}
```

Accessing message data:
About this task

In many cases, the user-defined node needs to access the contents of the message received on its input terminal. The message is represented as a tree of syntax elements. Use the supplied utility function to evaluate methods for message management, message buffer access, syntax element navigation, and syntax element access.

The MbElement class provides the interface to the syntax elements.

For example:

Procedure

1. To navigate to the relevant syntax element in the XML message:

```
MbElement rootElement = assembly.getMessage().getRootElement();
MbElement switchElement =
rootElement.getLastChild().getFirstChild().getFirstChild();
```

2. To select the terminal indicated by the value of this element:

```
String terminalName;
String elementValue = (String)switchElement.getValue();
if(elementValue.equals("add"))
  terminalName = "add";
else if(elementValue.equals("change"))
  terminalName = "change";
else if(elementValue.equals("delete"))
  terminalName = "delete";
else if(elementValue.equals("hold"))
  terminalName = "hold";
else
  terminalName = "failure";

MbOutputTerminal out = getOutputTerminal(terminalName);
```

Results

Transforming a message object:

About this task

The received input message is read-only, so before you can transform a message, you must write it to a new output message. You can copy elements from the input message, or you can create new elements in the output message.

The MbMessage class provides the copy constructors, and the methods to get the root element of the message. The MbElement class provides the interface to the syntax elements.

For example, if you have an incoming message assembly with embedded messages, you could have the following code in the evaluate method of your user-defined node:

1. To create a new copy of the message assembly and its embedded messages:

```
MbMessage newMsg = new MbMessage(assembly.getMessage());
MbMessageAssembly newAssembly = new MbMessageAssembly(assembly, newMsg);
```

2. To navigate to the relevant syntax element in the XML message:

```
MbElement rootElement = newAssembly.getMessage().getRootElement();
MbElement switchElement =
rootElement.getFirstElementByPath("/XML/data/action");
```

3. To change the value of an existing element:

```
String elementValue = (String)switchElement.getValue();
  if(elementValue.equals("add"))
    switchElement.setValue("change");
  else if(elementValue.equals("change"))
    switchElement.setValue("delete");
  else if(elementValue.equals("delete"))
    switchElement.setValue("hold");
  else
    switchElement.setValue("failure");
```

4. To add a new tag as a child of the switch tag:

```
MbElement tag = switchElement.createElementAsLastChild(MbElement.TYPE_NAME,
                                                        "PreviousValue",
                                                        elementValue);
```

5. To add an attribute to this new tag:

```
tag.createElementAsFirstChild(MbElement.TYPE_NAME_VALUE,
                               "NewValue",
                               switchElement.getValue());

MbOutputTerminal out = getOutputTerminal("out");
```

As part of the transformation, you might need to create a new message body. To create a new message body, use one of the following methods, which specifically assigns a parser to a message tree folder:

```
createElementAfter(String)
createElementAsFirstChild(String)
createElementAsLastChild(String)
createElementBefore(String)
createElementAsLastChildFromBitstream(byte[], String, String, String, String, int, int, int)
```

Do not use the following methods, which do not associate an owning parser with the folder:

```
createElementAfter(int)
createElementAfter(int, String, Object)
createElementAsFirstChild(int)
createElementAsFirstChild(int, String, Object)
```

```
createElementAsLastChild(int)
createElementAsLastChild(int, String, Object)
createElementBefore(int)
createElementBefore(int, String, Object)
```

Propagating the message:
About this task

Before you propagate a message, decide what message flow data you want to propagate, and whether to propagate to a node terminal, or to a Label node.

For example:

1. To propagate the message to the output terminal "out":
   ```
   MbOutputTerminal out = getOutputTerminal("out");
          out.propagate(newAssembly);
   ```
2. To propagate the message to a Label node:
   ```
   MbRoute label1 = getRoute ("label1");
   Label1.propagate(newAssembly);
   ```

Call the clearMessage() function within the finally block of try/catch to clear the memory that is allocated for the message tree.

To propagate the same MbMessage object multiple times, call the finalizeMessage() method on the MBMessage object, so that any changes made to the message are reflected in the bit stream that is generated downstream of the Java node; for example:

```
MbMessage newMsg = new MbMessage(assembly.getMessage());
MbMessageAssembly newAssembly = new MbMessageAssembly(assembly, newMsg);
...
newMsg.finalizeMessage(MbMessage.FINALIZE_NONE);
out.propagate(newAssembly;
...
newMsg.finalizeMessage(MbMessage.FINALIZE_NONE);
out.propagate(newAssembly);
```

Declaring the node name:
About this task

The name of the node must be the same as the one that is used in the WebSphere Message Broker Toolkit. All node names must end with "Node". Declare the name using the following method:

```
public static String getNodeName()
{
   return "BasicNode";
}
```

If this method is not declared, the Java API framework creates a default node name using the following rules:

- The class name is appended to the package name.
- The dots are removed, and the first letter of each part of the package and class name are capitalized.

For example, by default, the following class is assigned the node name "ComIbmPluginsamplesBasicNode":

```
package com.ibm.pluginsamples;
public class BasicNode extends MbNode implements MbNodeInterface
{
   ...
```

Declaring attributes:
About this task

Declare node attributes in the same way as Java Bean properties. You must write getter and setter methods for the attributes. The API framework infers the attribute names using the Java Bean introspection rules. For example, if you declare the following two methods:

```
private String attributeVariable;

public String getFirstAttribute()
{
  return attributeVariable;
}

public void setFirstAttribute(String value)
{
  attributeVariable = value;
}
```

the broker infers that this node has an attribute called firstAttribute. This name is derived from the names of the get or set methods, not from any internal class member variable names. Attributes can only be exposed as strings, therefore, you must convert any numeric types to and from strings in the get or set methods. For example, the following method defines an attribute called timeInSeconds:

```
int seconds;

public String getTimeInSeconds()
{
  return Integer.toString(seconds);
}

public void setTimeInSeconds(String value)
{
  seconds = Integer.parseInt(value);
}
```

Implementing the node functionality:
About this task

The evaluate method, defined in MbNodeInterface, is called by the broker to process the message. All the processing function for the node is included in this method.

The evaluate method has two parameters that are passed in by the broker:

1. The MbMessageAssembly, which contains the following objects that are accessed using the appropriate methods:
   - The incoming message
   - The LocalEnvironment
   - The global Environment
   - The ExceptionList
2. The input terminal on which the message has arrived.

For example, the following code extract shows how you might write the evaluate method:

```
public void evaluate(MbMessageAssembly assembly, MbInputTerminal inTerm) throws MbException
  {
    // add message processing code here

    getOutputTerminal("out").propagate(assembly);
  }
```

The message flow data, which consists of the message, Environment, LocalEnvironment, and ExceptionList, is received at the input terminal of the node.

Deleting an instance of the node:
About this task

An instance of the node is deleted when either:

- You shut down the broker.
- You remove the node or the message flow that contains the node, and redeploy the configuration.

If you want the node to perform any cleanup operations, for example closing sockets, include an implementation of the onDelete method:

```
public void onDelete()
{
  // perform node cleanup if necessary
}
```

This method is called by the broker immediately before it deletes the node.

Related concepts:

"User-defined extensions in the runtime environment" on page 2980
Before you design and implement user-defined extensions, familiarize yourself with the core components. Ensure that you also understand the basic WebSphere Message Broker runtime architecture.

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Developing user-defined extensions" on page 2970
A user-defined extension is a component that you design and implement to extend the function of WebSphere Message Broker.

"Why use a user-defined extension?" on page 2972
Use a user-defined node or parser when the built-in resources do not provide the required functions.

Related tasks:

"Extending the capability of a Java message processing or output node" on page 3069
Within a message processing or output node, you can add extended functions to your Java node.

"Restrictions when creating Java nodes" on page 3055
In Java user-defined nodes and the JavaCompute node, calling the System.exit(...) method is not supported. Calling this method results in a `SecurityException`.

"Implementing the supplied user-defined extension samples" on page 3017
WebSphere Message Broker provides some sample code to help you understand how to write user-defined nodes and parsers.

"Compiling a Java user-defined node" on page 3074
When you have created the code for your Java user-defined node, you must compile it for your operating system.

Related reference:

"Exception list structure" on page 4224
An exception list contains information about exceptions, such as error numbers, the name of the node that generated the exception, and the reason for the exception.

"Label node" on page 4569
Use the Label node to process a message that is propagated by a RouteToLabel node to dynamically determine the route that the message takes through the message flow.

Related information:

Java user-defined extensions API

Extending the capability of a Java message processing or output node:

Within a message processing or output node, you can add extended functions to your Java node.

Before you begin

Before you start

Read "Creating a message processing or output node in Java" on page 3062.

About this task

You can add one or more of the following functions:

- "Accessing ESQL"
- "Interacting with databases" on page 3070
- "Handling exceptions" on page 3070
- "Writing to an output device" on page 3072

Accessing ESQL:
About this task

Nodes can invoke ESQL expressions using Compute node ESQL syntax. You can create and modify the components of the message using ESQL expressions, and you can refer to elements of both the input message and data from an external database.

The following procedure demonstrates how to use ESQL to control transactions from the evaluate method in your user-defined node:

1. Set the name of the ODBC data source to use. For example:

   ```
   String dataSourceName = "myDataSource";
   ```

2. Set the ESQL statement to run:

   ```
   String statement =
      "SET OutputRoot.XMLNS.data =
             (SELECT Field2 FROM Database.Table1 WHERE Field1 = 1);";
   ```

   Or, if you want to run a statement that returns no result:

```
String statement = "PASSTHRU(
                         'INSERT INTO Database.Table1 VALUES(
                              InputRoot.XMLNS.DataField1,
                              InputRoot.XMLNS.DataField2)');";
```

3. Select the transaction you want from the following types:

   MbSQLStatement.SQL_TRANSACTION_COMMIT
   Immediately commit the transaction after the ESQL statement has completed.

   MbSQLStatement.SQL_TRANSACTION_AUTO
   Commit the transaction when the message flow has completed. (Rollbacks are performed if necessary.)

   For example:

   ```
   int transactionType = MbSQLStatement.SQL_TRANSACTION_AUTO;
   ```

4. Get the ESQL statement. For example:

   ```
   MbSQLStatement sql =
          createSQLStatement(dataSourceName, statement, transactionType);
   ```

   You can use the method createSQLStatement(data source, statement to default the transaction type to MbSQLStatement.SQL_TRANSACTION_AUTO).

5. Create the new message assembly to be propagated:

   ```
   MbMessageAssembly newAssembly =
          new MbMessageAssembly(assembly, assembly.getMessage());
   ```

6. Run the ESQL statement:

   ```
   sql.select(assembly, newAssembly);
   ```

   Or, if you want to run an ESQL statement that returns no result:

   ```
   sql.execute(assembly);
   ```

Interacting with databases:
About this task

You can interact with databases from the Java code in your message processing node. The support that is provided is identical to the support for Java code that you write for the JavaCompute node; for details of the available options, and the advantages and restrictions that apply, see "Interacting with databases by using the JavaCompute node" on page 2661.

Handling exceptions:
About this task

Use the MbException class to catch and access exceptions. The MbException class returns an array of exception objects that represent the children of an exception in the broker exception list. Each element returned specifies its exception type. An empty array is returned if an exception has no children. The following code sample shows an example of how you might use the MbException class in the evaluate method of your user-defined node.

```
public void evaluate(MbMessageAssembly assembly, MbInputTerminal inTerm) throws MbException
  {
    try
      {

        // plug-in functionality

      }
    catch(MbException ex)
```

```
    {
      traverse(ex, 0);

      throw ex; // if re-throwing, it must be the original exception that was caught
    }
  }

  void traverse(MbException ex, int level)
  {
    if(ex != null)
      {
        // Do whatever action here
        System.out.println("Level: " + level);
        System.out.println(ex.toString());
        System.out.println("traceText:  " + ex.getTraceText());

        // traverse the hierarchy
        MbException e[] = ex.getNestedExceptions();
        int size = e.length;
        for(int i = 0; i < size; i++)
          {
            traverse(e[i], level + 1);
          }
      }
  }
```

You can develop a user-defined message-processing or output node in such a way that it can access all current exceptions. For example, to catch database exceptions you can use the MbSQLStatement class. This class sets the value of the 'throwExceptionOnDatabaseError' attribute, which determines broker behavior when it encounters a database error. When it is set to true, if an exception is thrown it can be caught and handled by the evaluate method in your user-defined extension.

The following code sample shows an example of how to use the MbSQLStatement class.

```
public void evaluate(MbMessageAssembly assembly, MbInputTerminal inTerm) throws MbException
  {
    MbMessage newMsg = new MbMessage(assembly.getMessage());
    MbMessageAssembly newAssembly = new MbMessageAssembly(assembly, newMsg);

    String table =
       assembly.getMessage().getRootElement().getLastChild().getFirstChild().getName();

    MbSQLStatement state = createSQLStatement( "dbName",
       "SET OutputRoot.XMLNS.integer[] = PASSTHRU('SELECT * FROM " + table + "');" );

    state.setThrowExceptionOnDatabaseError(false);
    state.setTreatWarningsAsErrors(true);

    state.select( assembly, newAssembly );

    int sqlCode = state.getSQLCode();
    if(sqlCode != 0)
      {
        // Do error handling here

        System.out.println("sqlCode = " + sqlCode);
        System.out.println("sqlNativeError = " + state.getSQLNativeError());
        System.out.println("sqlState = " + state.getSQLState());
        System.out.println("sqlErrorText = " + state.getSQLErrorText());
      }

    getOutputTerminal("out").propagate(newAssembly);
  }
```

Writing to an output device:
About this task

To write to an output device, the logical (hierarchical) message must be converted back into a bit stream in your evaluate method. Use the getBuffer method in **MbMessage** to perform this task:

```
public void evaluate( MbMessageAssembly assembly, MbInputTerminal in)
                                                        throws MbException
{
  MbMessage msg = assembly.getMessage();
  byte[] bitstream = msg.getBuffer();

  // write the bitstream out somewhere
  writeBitstream( bitstream );   // user method

 }
```

Typically, for an output node the message is not propagated to any output terminal, therefore you can just return at this point.

You must use the supplied MQOutput node when writing to WebSphere MQ queues, because the broker internally maintains a WebSphere MQ connection and the open queue handles on a thread-by-thread basis. These handles are cached to optimize performance. In addition, the broker handles exception scenarios when certain WebSphere MQ events occur, and this recovery is adversely affected if WebSphere MQ MQI calls are used in a user-defined output node.

Related concepts:

"User-defined extensions in the runtime environment" on page 2980
Before you design and implement user-defined extensions, familiarize yourself with the core components. Ensure that you also understand the basic WebSphere Message Broker runtime architecture.

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Developing user-defined extensions" on page 2970
A user-defined extension is a component that you design and implement to extend the function of WebSphere Message Broker.

"Why use a user-defined extension?" on page 2972
Use a user-defined node or parser when the built-in resources do not provide the required functions.

Related tasks:

"Creating a message processing or output node in Java" on page 3062
A message processing node is used to process a message, and an output node is used to output a message as a bit stream.

"Implementing the supplied user-defined extension samples" on page 3017
WebSphere Message Broker provides some sample code to help you understand how to write user-defined nodes and parsers.

"Compiling a Java user-defined node" on page 3074
When you have created the code for your Java user-defined node, you must compile it for your operating system.

"Enabling JDBC connections to the databases" on page 683
Configure connections to a user database through a JDBCProvider service.

"Interacting with databases by using the JavaCompute node" on page 2661
Access databases from Java code included in the JavaCompute node.

Related reference:

"Exception list structure" on page 4224
An exception list contains information about exceptions, such as error numbers, the name of the node that generated the exception, and the reason for the exception.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

Related information:

Java user-defined extensions API

Getting and setting the specific type of an Mb element:

Two methods are provided for handling the specific type of an Mb syntax element.

About this task

The following methods are available:
- getSpecificType
- setSpecificType

Use these methods to access or set the specific type of an XML element. For example, to update the current value:

Procedure

1. Call getSpecificType on the syntax element.

   The getSpecificType method does not take any parameters, but returns the specific type of the element as an int value.
2. Call setSpecificType on the syntax element.

   The setSpecificType method takes one parameter of the type int, which is the specific type that you want the Mb element to be. This method has no return value.

Results

Specific type values for the XML and MRM parsers are listed in "XML, MRM, and XMLNSC parser constants" on page 6691.

Related concepts:

"Specific types used by parsers" on page 3015
Specific types are used when a parser needs additional information that is associated with some or all the elements in a tree in order to generate the bit stream.

Related reference:

"XML, MRM, and XMLNSC parser constants" on page 6691
The names of the XML and MRM parser constants, together with their corresponding values, and a link to the XMLNSC constants.

Compiling a Java user-defined node:

When you have created the code for your Java user-defined node, you must compile it for your operating system.

Before you begin

Before you start

You must have a user-defined node written in Java. This node can be one of the provided sample nodes that are described in "Sample node files" on page 6412, or a node that you have created yourself by using the instructions in either "Creating a message processing or output node in Java" on page 3062 or "Creating an input node in Java" on page 3055.

About this task

You can compile a Java user-defined node either from the command line, or from within the project in the WebSphere Message Broker Toolkit. Both options are described later in this section.

When you compile a Java user-defined node from the command line, you must have a compatible IBM Software Developer Kit for Java on the current operating system. For details of supported Java versions, see "Additional software requirements" on page 3598.

Compiling a Java user-defined node from the WebSphere Message Broker Toolkit:
About this task

Use the following procedure to compile your Java user-defined node from the WebSphere Message Broker Toolkit:

Procedure

1. In the Package Explorer, select the `/src` directory inside your node project, and click **File** > **Export**.
2. From the list displayed, select **JAR file**. Click **Next**. The resources that are available for you to export as a JAR file are listed.
3. Verify that **Export generated class files and resources** is selected.
4. Specify a name and location for your JAR file. Place the file inside the root directory of your node project, and give the file the same name as the name of the project (with a `.jar` extension). You can use the default values for the rest of the options. Click **Finish**.

Results

The `.jar` file that you have created is displayed in your node project, and is ready for you to install on one or more brokers (see "Installing user-defined extension runtime files on a broker" on page 3125), or to package for distribution (see "Packaging and distributing a user-defined node project" on page 3121).

Compiling a Java user-defined node from the command line:
About this task

Use the following procedure to compile your Java user-defined node from the command line:

Procedure

1. Add the location of `jplugin2.jar` to the CLASSPATH for your current platform:

   Windows *install_dir*`\classes\jplugin2.jar`

   Linux *install_dir*`/classes/jplugin2.jar`

   UNIX *install_dir*`/classes/jplugin2.jar`

   z/OS *install_dir*`/classes/jplugin2.jar`
2. Put your Java user-defined node class into the following directory:

   Windows *install_dir*`\sample\extensions\nodes`

   Linux *install_dir*`/sample/extensions/nodes`

   UNIX *install_dir*`/sample/extensions/nodes`

   z/OS *install_dir*`/sample/extensions/nodes`
3. Change to the directory that now contains your user-defined node class.
4. Compile the `.java` file by using the **`javac`** command; for example:

   `javac` *`nodename`*`.java`
5. Package the resulting `.class` file into a `.par` file. See "Packaging a Java user-defined node" on page 3118.

Results

The `.par` file that you have created is ready for you to install on one or more brokers (see "Installing user-defined extension runtime files on a broker" on page 3125), or to package for distribution (see "Packaging and distributing a user-defined node project" on page 3121).

Related concepts:

"User-defined input nodes" on page 2990
A user-defined input node is an extension to the broker that provides a new input node in addition to the nodes supplied with the product.

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

"Developing user-defined extensions" on page 2970
A user-defined extension is a component that you design and implement to extend

the function of WebSphere Message Broker.

"Why use a user-defined extension?" on page 2972
Use a user-defined node or parser when the built-in resources do not provide the required functions.

"User-defined node class loading" on page 3120
Details the Java classes packaging options and loading order precedence for user-defined nodes.

Related tasks:

"Installing user-defined extension runtime files on a broker" on page 3125
Install the compiled runtime files for your user-defined extension on the broker on which you want to test its function.

"Testing a user-defined node" on page 3094
When you have created and installed the required resources, you can test your user-defined node.

"Packaging and distributing a user-defined node project" on page 3121
Export the user-defined node project to make it available for other users.

"Packaging a Java user-defined node" on page 3118
How to package a Java user-defined node.

Related information:

Java user-defined extensions API

Creating a user-defined node from a subflow:

Create user-defined nodes either from scratch, or by using an existing subflow.

About this task

Complete the appropriate tasks from the following list:

- "Creating a user-defined node from a subflow from scratch"
- "Creating a user-defined node from an existing subflow" on page 3078
- "Choosing the location of a user-defined node in the palette" on page 3092

Related concepts:

"User-defined nodes" on page 2989
User-defined nodes are the main mechanism for extending the functions of WebSphere Message Broker.

"Using a subflow as a user-defined node" on page 3008
You can develop a user-defined node that packages a subflow from scratch, in the same way that you can create any other user-defined node that has its implementation based on Java, or base it on an existing subflow.

Related tasks:

"Testing a subflow user-defined node project" on page 3097
Test how the user-defined node is displayed, and how it behaves in your environment.

"Packaging and distributing a user-defined node project" on page 3121
Export the user-defined node project to make it available for other users.

Creating a user-defined node from a subflow from scratch:

Create a user-defined node that packages a subflow by creating a user-defined node project, then creating the user-defined node.

About this task

After you have created a user-defined node, you can edit its properties in the Palette editor.

You can define as many user-defined nodes as you want in the same user-defined node project.

To create any type of user-defined node, use the following steps:

Procedure

1. Create a user-defined node project:
   a. To open the New User-defined Node Project wizard, click **File** > **New** > **New User-defined Node Project**, then click **Next**.
   b. Either choose an existing category, or create a category by clicking **Create a new category**. Your user-defined nodes are displayed in the palette in the Message Flow editor under the category you specify for the project. You can have only one category in each user-defined node project.
   c. Enter the details for the remainder of the fields as required, then click **Finish**.

   A user-defined node project is created.
2. Create a user-defined node:
   a. To open the New User-defined Node wizard, click **File** > **New** > **User-defined Node**.
   b. Select the user-defined node project that you created previously.
   c. Select the schema location that you want to host the user-defined node. Use a fully qualified schema. Do not use a default or `mqsi` schema.
   d. Enter the details for the remainder of the fields as required.
   e. Select **Implemented as a subflow**. The Message Flow editor opens.
   f. Click **Finish**.
3. Decide on the location of your user-defined node in the palette. See "Choosing the location of a user-defined node in the palette" on page 3092.

What to do next

You can now define the reusable logic and the terminals and properties of the user-defined node.

Related concepts:

"User-defined nodes" on page 2989
User-defined nodes are the main mechanism for extending the functions of WebSphere Message Broker.

"Using a subflow as a user-defined node" on page 3008
You can develop a user-defined node that packages a subflow from scratch, in the same way that you can create any other user-defined node that has its implementation based on Java, or base it on an existing subflow.

Related tasks:

"Testing a subflow user-defined node project" on page 3097
Test how the user-defined node is displayed, and how it behaves in your environment.

"Packaging and distributing a user-defined node project" on page 3121
Export the user-defined node project to make it available for other users.

"Creating a user-defined node from an existing subflow"
Create a user-defined node by converting the project for an existing subflow into a user-defined node project, and then creating the user-defined node.

"Installing a user-defined node" on page 1496
Develop message flows that use a user-defined node.

"Choosing the location of a user-defined node in the palette" on page 3092
Use the Palette editor to edit palette-specific information for user-defined nodes, add and delete separators, and rearrange user-defined nodes in the palette.

Creating a user-defined node from an existing subflow:

Create a user-defined node by converting the project for an existing subflow into a user-defined node project, and then creating the user-defined node.

About this task

To create a user-defined node from an existing subflow, use the following steps:

Procedure

1. To convert existing subflows into user-defined nodes, you must convert the project for the subflow into a user-defined node project:
   a. Right-click the existing project, click **Convert to User-defined Node Project**. The Convert to User-defined Node Project wizard is displayed.
   b. Choose a category from the following options. Your user-defined nodes are displayed in the palette in the Message Flow editor under the category you specify for the project.
      - To use an existing category, select a category from the list provided.
      - To create a new category, select **Create a new category**, enter a name for the category for the nodes that you want to create in the new project.
   c. Enter the details for the remainder of the fields as required.
   d. Click **Finish**.
2. Select the subflow that you want to convert to a user-defined node:
   a. Right-click the subflow to display the menu.
   b. Click **Convert to User-defined Node**. The Convert to User-defined Node wizard opens.
   c. Enter the details for the remainder of the fields as required.
   d. Click **Finish**.
3. Decide on the location of your user-defined node in the palette. See "Choosing the location of a user-defined node in the palette" on page 3092.

Related concepts:

"User-defined nodes" on page 2989
User-defined nodes are the main mechanism for extending the functions of WebSphere Message Broker.

"Using a subflow as a user-defined node" on page 3008
You can develop a user-defined node that packages a subflow from scratch, in the same way that you can create any other user-defined node that has its implementation based on Java, or base it on an existing subflow.

Related tasks:

"Testing a subflow user-defined node project" on page 3097
Test how the user-defined node is displayed, and how it behaves in your environment.

"Packaging and distributing a user-defined node project" on page 3121
Export the user-defined node project to make it available for other users.

"Creating a user-defined node from a subflow from scratch" on page 3076
Create a user-defined node that packages a subflow by creating a user-defined node project, then creating the user-defined node.

"Installing a user-defined node" on page 1496
Develop message flows that use a user-defined node.

"Choosing the location of a user-defined node in the palette" on page 3092
Use the Palette editor to edit palette-specific information for user-defined nodes, add and delete separators, and rearrange user-defined nodes in the palette.

Creating the user interface representation of a user-defined node in the WebSphere Message Broker Toolkit:

When you are developing a user-defined node in Java or C only, you must create the user interface representation of the node in the WebSphere Message Broker Toolkit.

Before you begin

Before you start:

- Perform the steps in "Creating a user-defined node" on page 3025.

About this task

The following topics describe the steps that you must complete:

1. "Creating a user-defined node project" on page 3080
2. "Creating a user-defined node in the WebSphere Message Broker Toolkit" on page 3081

What to do next

When you have created the WebSphere Message Broker Toolkit representation, test your user-defined node, see "Testing a user-defined node" on page 3094.

Related concepts:

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

"Developing user-defined extensions" on page 2970
A user-defined extension is a component that you design and implement to extend the function of WebSphere Message Broker.

"Why use a user-defined extension?" on page 2972
Use a user-defined node or parser when the built-in resources do not provide the required functions.

Related tasks:

"Testing a user-defined node" on page 3094
When you have created and installed the required resources, you can test your user-defined node.

"Packaging and distributing a user-defined node project" on page 3121
Export the user-defined node project to make it available for other users.

Related reference:

"User-defined extensions" on page 6411
Reference material that supports the creation and management of your user-defined extensions.

Creating a user-defined node project:

When you create user-defined nodes, you must first create a user-defined node project to contain the nodes and their supporting files.

About this task

To create a new project for your user-defined node, complete the following steps:

Procedure

1. Click **File** > **New** > **User-defined Node Project**. Click **Next** to start the User-defined Node Project wizard. The New User-defined Node Project window opens.
2. Either select an existing category, or select **Create a new category** and specify the name of a new category for the nodes that you are creating. Your user-defined nodes are displayed in the palette in the Message Flow editor under the category you specify for the project.
3. Specify a name for your project.
4. Specify a name for your plug-in identifier. To be consistent with the supplied nodes, and to avoid conflict with the names of node projects that are supplied by other independent software vendors, use your Internet domain name for your organization as part of the name and plug-in identifier. For example, the name and plug-in identifier must be of the form `com.xyz.nodegroup`, where *com.xyz* is the company Internet domain name.

   You can save any number of nodes in a single project.
5. Accept all default values and click **Finish**.
6. If warnings are displayed in the Problems view, that are associated with the newly created project, perform the following steps:
   a. Click **Window** > **Preferences**.
   b. Expand **Plug-In Development** and click **Target Platform**.
   c. Click **Add required Plug-ins** to select all loaded plug-ins and click **OK**.
   d. Select your user-defined node project in the Package Explorer view, and click **Project** > **Clean**. A dialog box opens in which you can select either of the following options:
      - **Clean all projects**.
      - **Clean projects selected below**. If you choose this option, select the projects that you want to clean.

Results

A project folder that contains all the supporting files that are required for your user-defined node is displayed in the Package Explorer view. The project is stored in the default workspace directory.

What to do next

Next:

Create the user-defined node plug-ins, see “Creating a user-defined node in the WebSphere Message Broker Toolkit.”

Related concepts:

“User-defined input nodes” on page 2990
A user-defined input node is an extension to the broker that provides a new input node in addition to the nodes supplied with the product.

“User-defined message processing nodes” on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

“User-defined output nodes” on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

“Developing user-defined extensions” on page 2970
A user-defined extension is a component that you design and implement to extend the function of WebSphere Message Broker.

“Why use a user-defined extension?” on page 2972
Use a user-defined node or parser when the built-in resources do not provide the required functions.

“Message flow node palette” on page 1027
The palette in the Message Flow editor contains all the built-in nodes, which are organized into categories, or drawers. A drawer is a container for a list of icons, such as the **Favorites** drawer.

Related tasks:

“Creating a user-defined node in the WebSphere Message Broker Toolkit”
Create the representation of a user-defined node created in Java and C only, in the WebSphere Message Broker Toolkit.

“Testing a user-defined node” on page 3094
When you have created and installed the required resources, you can test your user-defined node.

“Packaging and distributing a user-defined node project” on page 3121
Export the user-defined node project to make it available for other users.

Related reference:

“User-defined extensions” on page 6411
Reference material that supports the creation and management of your user-defined extensions.

Creating a user-defined node in the WebSphere Message Broker Toolkit:

Create the representation of a user-defined node created in Java and C only, in the WebSphere Message Broker Toolkit.

Before you begin

Before you start:

Complete the following task: “Creating a user-defined node project” on page 3080.

About this task

To create the visual representation of your user-defined node in the WebSphere Message Broker Toolkit, complete the following tasks:

1. “Creating the user-defined node plug-in files” on page 3082

2. “Defining the node properties” on page 3083
3. Optional: “Adding help to the node” on page 3088
4. Optional: “Creating node icons” on page 3090
5. Optional: “Adding a property editor or compiler” on page 3091

When you have created the representation of the node, you cannot move it to another folder.

Related concepts:

“User-defined input nodes” on page 2990
A user-defined input node is an extension to the broker that provides a new input node in addition to the nodes supplied with the product.

“User-defined message processing nodes” on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

“User-defined output nodes” on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

“Developing user-defined extensions” on page 2970
A user-defined extension is a component that you design and implement to extend the function of WebSphere Message Broker.

Related tasks:

“Creating a user-defined node project” on page 3080
When you create user-defined nodes, you must first create a user-defined node project to contain the nodes and their supporting files.

“Implementing the supplied user-defined extension samples” on page 3017
WebSphere Message Broker provides some sample code to help you understand how to write user-defined nodes and parsers.

“Testing a user-defined node” on page 3094
When you have created and installed the required resources, you can test your user-defined node.

“Packaging and distributing a user-defined node project” on page 3121
Export the user-defined node project to make it available for other users.

Related reference:

“User-defined extensions” on page 6411
Reference material that supports the creation and management of your user-defined extensions.

Creating the user-defined node plug-in files:

Create the files that contain the message processing logic, for user-defined nodes created in Java and C only.

Before you begin

Before you start:

Complete the task following task: “Creating a user-defined node project” on page 3080.

About this task

Procedure

1. Launch the wizard by clicking **File** > **New** > **User-defined Node**. The **New User-defined Node** window opens.
2. Select the parent folder for the node from the list of names that are displayed. This folder is the project that you created to contain this node.
3. Specify a schema for this node. You must not use the default schema or any other common schema, such as `mqsi`.
4. Specify a name for the node. The name must be the name of the node, excluding the `Node` at the end. For example, if you have created a node called `BasicNode`, the node name must be `Basic`.
5. Click **Finish**. A `.msgnode` file for the new node is created and is added to the project in the Broker Development view. The `.msgnode` file is opened in the Message Node editor.

What to do next

Next:

When you have completed this task, define the node properties, see "Defining the node properties."

Related concepts:

"User-defined input nodes" on page 2990
A user-defined input node is an extension to the broker that provides a new input node in addition to the nodes supplied with the product.

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a user-defined node project" on page 3080
When you create user-defined nodes, you must first create a user-defined node project to contain the nodes and their supporting files.

"Defining the node properties"
Define the properties for a user-defined node created in Java or C only, and add input and output terminals so that you can connect it to other nodes in a message flow.

Related reference:

"User-defined extensions" on page 6411
Reference material that supports the creation and management of your user-defined extensions.

Defining the node properties:

Define the properties for a user-defined node created in Java or C only, and add input and output terminals so that you can connect it to other nodes in a message flow.

Before you begin

Before you start:

Complete the following tasks:
1. "Creating a user-defined node project" on page 3080
2. "Creating the user-defined node plug-in files" on page 3082

About this task

When you complete the task described in "Creating the user-defined node plug-in files" on page 3082, a `.msgnode` file is created for the new node, and is opened in the Message Node editor of the Broker Application Development perspective. You can now add terminals and properties to the node.

Adding terminals to the node:
Procedure

1. If the Terminals page is not already displayed, click the **Terminals** tab at the bottom of the Message Node editor.
2. Click **Add** to the right of the **In Terminals** or **Out Terminals** fields to add an input or output terminal.

   You must define at least one input terminal, but output terminals are optional.
3. To rename a terminal, click the terminal name so that it is highlighted and shows a flashing cursor after the name, and enter a new name.
4. If your node supports dynamic input or output terminals, select the appropriate check box.

   Dynamic terminals are terminals that you can add to certain nodes after you have added them to a message flow in the Message Flow editor. For more information, see "Message flow node terminals" on page 1034.

Defining properties for the node:
Procedure

1. Click the **Properties** tab at the bottom of the Message Node editor.

   On the Properties page, you can set the properties for the node, for example, a database name, a host server name, or a password. The properties that you set here must match the properties that you specified in the user-defined node itself by using the get and set methods.
2. If the node is an input node, click the node name in the hierarchy to highlight it, and select **Input node**. Select **Use broker default values** if you want the node to initialize with the default values for the broker.
3. By default, all properties are grouped under the Basic group. You can add new groups in which to place properties. When your custom node is selected in the WebSphere Message Broker Toolkit, each group of properties is rendered as a separate tab in the Properties view. To create additional groups of properties, click **Add Property Group**  .
4. To add a simple property, click the name of a property group in the hierarchy to highlight it, and click **Add Simple Property** 

The new property is added to the hierarchy as a child of the property group. Its name is highlighted so that you can change it. A number of fields are displayed in the Details section, where you can configure the property.

a. Select the correct attribute type: one of the built-in types, or a type to match the list of values that the property can have.
b. Enter any default values, which are shown in the Properties view when the node is included in a message flow.
c. Optional: If you want to use a custom property compiler, in the **Custom Compiler Class** field specify the class you want to use for your custom compiler. The class must implement the IPropertyCompiler interface. For more information, see "Adding a property editor or compiler" on page 3091.
d. Optional: If you want to use a custom property editor, in the **Custom Editor Class** field specify the class you want to use for your custom editor. The class must implement the IPropertyEditor interface. For more information, see "Adding a property editor or compiler" on page 3091.
e. Specify the system property for each attribute that you define:

   Hidden
   The property is not displayed in the Properties view or the Promote Property dialog box.

   Read only
   The property is displayed, but cannot be changed.

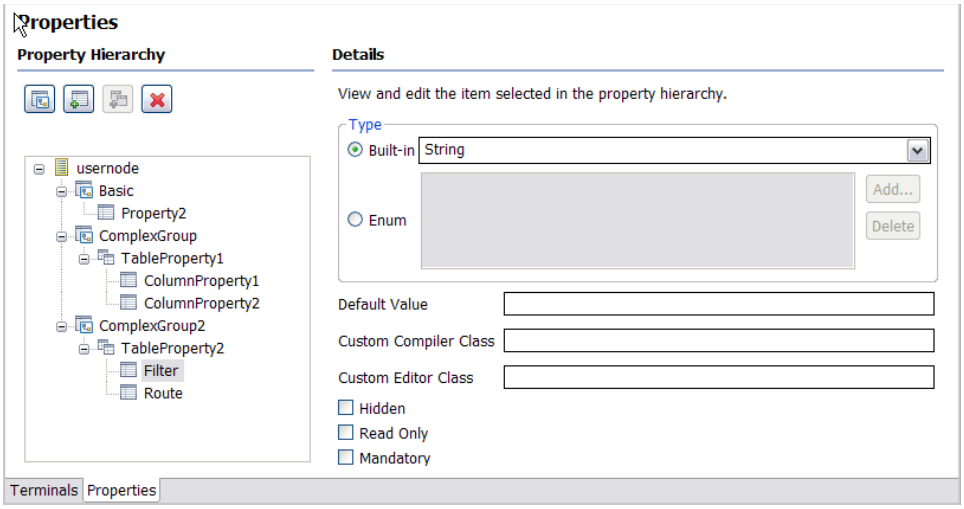
   Mandatory
   A value is required. The field cannot be left blank. Boolean and enum properties are always mandatory.

   Configurable
   The property can be configured at deployment time

5. To add a table property, select the name of a property group in the Properties view and click **Add Table Property**  .

   In addition to simple properties, a node can also have complex properties. A table property represents a repeating property of a complex type. The new property is added to the hierarchy as a child of the property group. A number of fields are displayed in the Details section where you can configure the table property.
6. To add a column to an existing table, select the name of the table property in the hierarchy, and click **Add Simple Property**  .

   For example, the following figure shows the Property Hierarchy of the usernode, where Filter and Route columns have been added.



A number of fields are displayed in the Details section where you can configure the properties of the column. Define table columns, where each column is assigned to a type.

a. Select the correct attribute type in the **Type** field: one of the built-in types, or enumeration.

b. Enter the default value, in the **Default Value** field. This value is shown in the Properties view when the node is included in a message flow.

c. Specify a qualified class name in the **Custom Compiler Class** field for a property compiler. To create a custom compiler, use the IPropertyCompiler interface. For more information about custom property editors and property compilers, see "Adding a property editor or compiler" on page 3091.

d. Specify a qualified class name in the **Custom Editor Class** field to generate a custom property editor. The property editor specified in this field implements the IColumnPropertyEditor interface responsible for cell editing behavior. Leaving the **Custom Editor Class** field blank means that a property editor matching-column type is used. Specify your own IColumnPropertyEditor only if you need custom cell editing behavior.

e. Specify the following attributes for each column that you define:

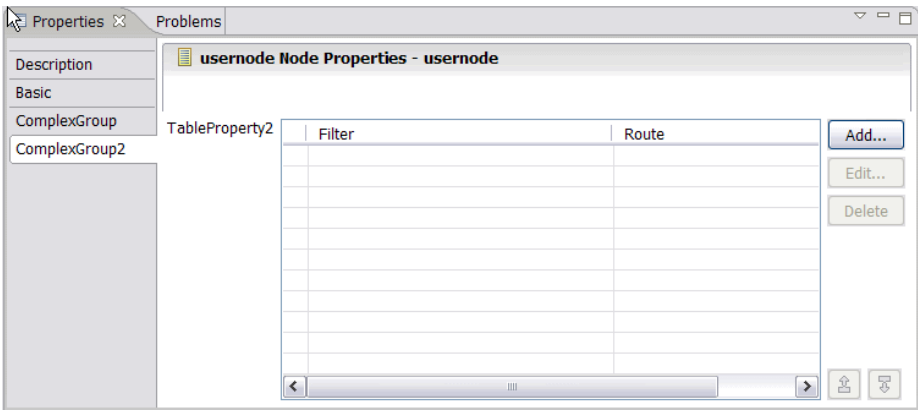
Hidden
Use a hidden column when you want to store, for each row, metadata that is not being displayed.

Read only
The column is displayed, but cannot be changed.

Mandatory
A value is required. The field cannot be left blank. Boolean and enum properties are always mandatory.

Leave the **Custom Editor Class** field of the Details section of a table property blank, unless you want to overwrite the behavior of the entire table. For example, if the table becomes disabled in response to a change in another property editor.

The following figure shows how the Table properties are rendered as a table in the Properties view, where you can add, edit, and delete values, and change the order of the values in the table.



7. Optional: Drag the properties in the properties hierarchy to change the order in which they are listed on the properties page.
8. Close the *nodename*.msgnode file.
9. Optional: You can customize the text that is displayed in the node properties view for each property. To set the text, open the *nodename*.properties file and edit the line:

   ```
   Property.propertyName = your descriptive text
   ```

What to do next

Next:

The following tasks are optional:

- "Adding help to the node" on page 3088
- "Creating node icons" on page 3090
- "Adding a property editor or compiler" on page 3091

You can now test your node, see "Testing a user-defined node" on page 3094.

Related concepts:

"User-defined input nodes" on page 2990
A user-defined input node is an extension to the broker that provides a new input node in addition to the nodes supplied with the product.

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

"Message flow node terminals" on page 1034
A terminal is the point at which one node in a message flow is connected to another node.

"Broker properties" on page 1144
For each broker, WebSphere Message Broker maintains a set of properties. You can access some of these properties from your ESQL programs. A subset of the properties is also accessible from Java programs. It can be useful, at run time, to have real-time access to details of a specific node, flow, or broker.

Related tasks:

"Creating a user-defined node project" on page 3080
When you create user-defined nodes, you must first create a user-defined node project to contain the nodes and their supporting files.

"Creating the user-defined node plug-in files" on page 3082
Create the files that contain the message processing logic, for user-defined nodes created in Java and C only.

"Promoting a property" on page 1298
You can promote a node property to the message flow level to simplify the maintenance of the message flow and its nodes, and to provide common values for multiple nodes in the flow by converging promoted properties.

Related reference:

"User-defined extensions" on page 6411
Reference material that supports the creation and management of your user-defined extensions.

Related information:

Property editor API

Adding help to the node:

Add help information by adding an HTML file to the node that you have created.

Before you begin

Before you start:

Complete the following tasks:

1. "Creating a user-defined node project" on page 3080
2. "Creating the user-defined node plug-in files" on page 3082
3. "Defining the node properties" on page 3083

About this task

Add help information for the node that you have created to explain why and when to use the node, and how it must be configured:

- Topic information that is displayed in the information center
- Context-sensitive help that is displayed when you press F1
- Hover help that is displayed when you hold the mouse pointer over the node

All three forms of help are optional; you can create any one or more of the three resources described in the following section.

Procedure

1. Create a `help.html` file in the project to contain the online help that explains what the node does and how you can use it. If you have several files, create a separate doc subdirectory in the plug-in project, and store the online help files in that directory.

   You can make the online help for the node look like it is integrated with the product-supplied information center, under the leaf node called "User-defined nodes", which you can find at **Reference** > **Message flow development** > **User-defined extensions** > **User-defined nodes**. To make the online help for your node show at that point, complete the following steps:

a. Modify the `plugin.xml` file to include the following extension point to the information center:

```
<extension point="org.eclipse.help.toc">
<toc file="toc.xml"/>
</extension>
```

b. Create a `toc.xml` file in your user-defined node project, and modify the `link_to` attribute to link to the "UDNodes" anchor that is already defined in the information center contents views:

```
<toc label="My Plugin Node" topic="my_node.htm"
 link_to="../com.ibm.etools.mft.doc/toc.xml#UDNodes">
<topic label="Mytopic 1" href="topic1.htm">
</toc>
```

Your help topic is now displayed in the contents view under **Reference** > **Message flow development** > **User-defined extensions** > **User-defined nodes**.

The sample nodes that are provided with the product demonstrate this option.

For further explanation of extension points and how to use them, see the PDE guide at Eclipse documentation.

2. Add context sensitive (F1) help to the node. Context-sensitive help is displayed when you click a node in the Broker Application Development perspective and press F1.

   When a node is created, a `HelpContexts.xml` file is created. This file assigns a context identifier based on the name of the node. Modify the `HelpContexts.xml` file for your node by changing the text in the description field. The name of the `HelpContexts.xml` file must be unique within the project, but can contain multiple context entries; for example, if you have several nodes within a single project, each node can have its context-sensitive help in the file.

   Context-sensitive help is limited in length. A useful way of providing more help to the user is to link from the F1 help to an HTML file that contains further information; for example, to the online help for the node, described previously. Use the following code for the link:

   ```
   <topic href"../plug-in directory/html file" label="Link title">
   ```

3. Add hover help (known as ToolTip help on Windows) to the node. When you create a user-defined node, a `palette.properties` file is created. Modify this file to contain the hover help for your node, which shows the node name when the palette is not wide enough to display it all.

What to do next

You can add another optional feature, a node icon or a property editor or compiler, or you can test your node, see “Creating node icons” on page 3090, “Adding a property editor or compiler” on page 3091, and “Testing a user-defined node” on page 3094.

Related concepts:

“User-defined input nodes” on page 2990
A user-defined input node is an extension to the broker that provides a new input node in addition to the nodes supplied with the product.

“User-defined message processing nodes” on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

“User-defined output nodes” on page 3006
A user-defined output node is an extension to the broker that provides a new

message flow output node in addition to the nodes supplied with the product.

"Developing user-defined extensions" on page 2970
A user-defined extension is a component that you design and implement to extend the function of WebSphere Message Broker.

Related tasks:

"Implementing the supplied user-defined extension samples" on page 3017
WebSphere Message Broker provides some sample code to help you understand how to write user-defined nodes and parsers.

"Testing a user-defined node" on page 3094
When you have created and installed the required resources, you can test your user-defined node.

"Packaging and distributing a user-defined node project" on page 3121
Export the user-defined node project to make it available for other users.

Related reference:

"User-defined extensions" on page 6411
Reference material that supports the creation and management of your user-defined extensions.

Creating node icons:

Create the icons that are displayed in the WebSphere Message Broker Toolkit when your user-defined node is present.

Before you begin

Before you start:

You must complete the following tasks:

1. "Creating a user-defined node project" on page 3080
2. "Creating the user-defined node plug-in files" on page 3082
3. "Defining the node properties" on page 3083

About this task

You can choose the icon when you create your user-defined node, or you can use the Palette editor to change the icon, see "Choosing the location of a user-defined node in the palette" on page 3092.

What to do next

You can add help, a property editor or compiler, or you can test your node, see "Adding help to the node" on page 3088 or "Adding a property editor or compiler" on page 3091, and "Testing a user-defined node" on page 3094.

Related concepts:

"User-defined input nodes" on page 2990
A user-defined input node is an extension to the broker that provides a new input node in addition to the nodes supplied with the product.

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Choosing the location of a user-defined node in the palette" on page 3092
Use the Palette editor to edit palette-specific information for user-defined nodes, add and delete separators, and rearrange user-defined nodes in the palette.

"Creating a user-defined node project" on page 3080
When you create user-defined nodes, you must first create a user-defined node project to contain the nodes and their supporting files.

Related reference:

"User-defined extensions" on page 6411
Reference material that supports the creation and management of your user-defined extensions.

Adding a property editor or compiler:

Create a property editor by using the IPropertyEditor interface to control how the properties of your user-defined node created in Java or C only, are displayed in the WebSphere Message Broker Toolkit. Create a custom compiler by using the IPropertyCompiler interface; for example, to encrypt a value before sending it to the server.

Before you begin

Before you start:

You must complete the following tasks:

1. "Creating a user-defined node project" on page 3080
2. "Creating the user-defined node plug-in files" on page 3082
3. "Defining the node properties" on page 3083

About this task

The IPropertyEditor interface is used as the basis for all the node property editors in the WebSphere Message Broker Toolkit. You can customize the property editor to contain different kinds of controls, such as text fields and lists. See the IPropertyEditor and IPropertyCompiler interfaces in the Property editor API.

Creating a Java class:
About this task

To create a Java class for your property editor or compiler, complete the following steps.

Procedure

1. Switch to the Java perspective.
2. Right-click your user-defined node project and click **New** > **Class**. If **Class** is not shown, click **Other**, select **Class** and click **Next**.
3. Type a name for your class in the **Name** field.
4. Complete the following steps, according to whether you are creating a property editor or a property compiler:
   - If you are creating a property editor:
   a. Delete any text in the **Superclass** text field, and click **Browse**.
   b. In the **Choose a type** field, type `AbstractPropertyEditor` and click **OK**. AbstractPropertyEditor implements the IPropertyEditor interface.

- If you are creating a property compiler:

a. Click **Add** next to the **Interfaces** field.

b. In the **Choose interfaces** field, type `IPropertyCompiler` and click **OK**.

5. Click **Finish**.

Testing your property editor or compiler:
About this task

To test your property editor, see "Testing a user-defined node" on page 3094.

To test your property compiler, deploy to a broker the flow that contains your user-defined node.

A custom property editor can use Rational Application Developer or Eclipse APIs. When you migrate to a new version of WebSphere Message Broker, your custom property editor might not work if the Rational Application Developer or Eclipse APIs change. Update your property editor code to comply with the changed API.

Related concepts:

"User-defined input nodes" on page 2990
A user-defined input node is an extension to the broker that provides a new input node in addition to the nodes supplied with the product.

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a user-defined node project" on page 3080
When you create user-defined nodes, you must first create a user-defined node project to contain the nodes and their supporting files.

Related reference:

"User-defined extensions" on page 6411
Reference material that supports the creation and management of your user-defined extensions.

Related information:

Property editor API

Choosing the location of a user-defined node in the palette:

Use the Palette editor to edit palette-specific information for user-defined nodes, add and delete separators, and rearrange user-defined nodes in the palette.

About this task

- If you change the details for a user-defined node in the Palette editor, save the changes before you add or remove another user-defined node or the changed details will be lost.
- Do not confuse the Palette editor and Customize functions:

  Palette editor
  The Palette editor is started from the Navigator when you select a user-defined node project or the user-defined node virtual folder. The

Palette editor is used to set the category, node names, and icons that you want to export so that the user can see the modified node.

Customize action
The Customize action is started from the palette in the Message Flow editor when you select a node. The Customize action is used to change the way that the nodes are displayed in the workspace of the current user.

Procedure

1. To open the Palette editor use one of the following options:
   - In the Broker Development view, under the user-defined node project, select **User Defined Nodes**, click **Edit and Arrange Palette**.
   - In the Broker Development view, in the user-defined node project, right-click **palette.xmi**. Click **Open With** and select **Palette Editor**.
2. To edit palette-specific information:
   a. In the left side of the Palette editor under the Category folder, click the user-defined node that you want to edit. The details about the user-defined node are displayed in the right side of the Palette editor.
   b. You can change the details in the **Display Name**, **Tooltip on palette**, and **Node Icons** fields, but you cannot change the **Node Implementation** section.
   c. You can change the category by selecting the Category folder on the left side of the Palette editor, and on the right side either choose an existing category, or rename your own category.
3. To group the user-defined nodes you can add or delete a separator:
   - To add a separator in a user-defined node:
     a. In the left side of the Palette editor, right-click the node.
     b. Select **Add Separator Below**. A separator is added below the node in the left side of the Palette editor.
   - To delete a separator:
     a. In the left side of the Palette editor, right-click **Separator**.
     b. Click **Delete Separator**.
4. To rearrange your user-defined nodes, in the left side of the Palette editor either drag the node to its new position, or right-click the node and select **Move Up** or **Move Down**.

Related concepts:

"User-defined nodes" on page 2989
User-defined nodes are the main mechanism for extending the functions of WebSphere Message Broker.

"Using a subflow as a user-defined node" on page 3008
You can develop a user-defined node that packages a subflow from scratch, in the same way that you can create any other user-defined node that has its implementation based on Java, or base it on an existing subflow.

Related tasks:

"Testing a subflow user-defined node project" on page 3097
Test how the user-defined node is displayed, and how it behaves in your environment.

"Packaging and distributing a user-defined node project" on page 3121
Export the user-defined node project to make it available for other users.

"Creating a user-defined node from a subflow from scratch" on page 3076
Create a user-defined node that packages a subflow by creating a user-defined node project, then creating the user-defined node.

"Creating a user-defined node from an existing subflow" on page 3078
Create a user-defined node by converting the project for an existing subflow into a user-defined node project, and then creating the user-defined node.

"Using the node palette" on page 1489
The node palette contains all the built-in nodes, which are organized into categories.

Testing a user-defined node:

When you have created and installed the required resources, you can test your user-defined node.

Before you begin

Before you start

Complete the following tasks:

- "Creating a user-defined extension in C" on page 3026 or "Creating a user-defined extension in Java" on page 3054
- "Creating the user interface representation of a user-defined node in the WebSphere Message Broker Toolkit" on page 3079
- "Installing user-defined extension runtime files on a broker" on page 3125

This topic is for user-defined nodes created in Java and C only.

For user-defined nodes created from a subflow, see "Testing a subflow user-defined node project" on page 3097.

Procedure

Use the following steps to test your user-defined node:

1. To test these projects and resources you must launch a second WebSphere Message Broker Toolkit by using the following steps:
   a. Right-click the user-defined node project. The menu opens.
   b. Click **Run in New Workbench**.
      - If the project contains errors, you receive an error message that gives you the option to continue, or to cancel the action.
      - If the project is error-free, the Workspace Launcher window opens.
        1) You can choose to use a new or an existing workspace. To choose an existing workspace, click **Browse**. The second workspace opens.
        2) You can now test by using the user-defined nodes in all error-free user-defined node projects in your first workbench.
2. Open the Message Flow editor in the second instance of the WebSphere Message Broker Toolkit. Your new nodes are displayed in the node palette.
3. Create a message flow that includes your node. See "Adding a message flow node" on page 1494.
4. Deploy the message flow to a broker. See "Deploying resources" on page 3234.

5. Send a test message through the flow and look for the results that you expect (for example, a message put to a target queue). You might have to write an application to send the test message to the message flow.
6. Use the diagnostic tools that are provided to determine whether your node works, or if not, what went wrong:
   a. For a description of some common problems and their solutions. See "Resolving problems with user-defined extensions" on page 3511.
   b. Check the Administration log. See "Administration Log view" on page 6840.
   c. Write entries to the Administration log from your node. See "Using error logging from a user-defined extension" on page 3137.
   d. Switch on user trace at debug level. See "Using trace" on page 3533.

      The following debug messages are generated by a user trace to help you to understand the execution of your user-defined nodes and parsers:

      - BIP2233 and BIP2234: A pair of messages traced before and after a user-defined extension implementation function is started. These messages report the input parameters and the returned value.

        In these messages, an "implementation function" can be interpreted as either a C implementation function or a Java implementation method.
      - BIP3904: A message traced before starting the Java evaluate() method of a user-defined node.
      - BIP3905: A message traced before starting the C cniEvaluate() implementation function (iFpEvaluate member of CNI_VFT) of a user-defined node.
      - BIP4142: A debug message that is traced when starting a user-defined node utility function, where the utility function alters the state of a syntax element. All utility functions that start with cniSetElement*, where * represents all nodes with that stem, are included.
      - BIP4144 and BIP4145: A pair of messages traced by certain implementation functions that, when started by a user-defined extension, can modify the internal state of an object in the broker. Possible broker objects include syntax element, node, and parser.

        In these messages, an "implementation function" can be interpreted as either a C implementation function or a Java implementation method.
      - BIP4146: A debug message that is traced when starting a user-defined parser utility function, where the utility function alters the state of a syntax element. All utility functions that start with `cpiSetElement*`, where * represents all nodes with that stem, are included.
      - BIP4147: An error message that is traced when a user-defined extension passes an invalid input object to a user-defined extension utility API function.
      - BIP4148: An error message that is traced when a user-defined extension damages an object in a broker.
      - BIP4149: An error message that is traced when a user-defined extension passes an invalid input data pointer to a user-defined extension utility API function.
      - BIP4150: An error message that is traced when a user-defined extension passes invalid input data to a user-defined extension utility API function.
      - BIP4151: A debug message that is traced when cniGetAttribute2 or cniGetAttributeName2 sets the return code to an unexpected value. Expected values are CCI_SUCCESS, CCI_ATTRIBUTE_UNKNOWN, and CCI_BUFFER_TOO_SMALL. Any other value is an unexpected value.

- BIP4152: A debug message that is traced in the following situations:
  1) cniGetAttribute2 or cniGetAttributeName2 sets the return code to CCI_BUFFER_TOO_SMALL.
  2) cniGetAttribute2 or cniGetAttributeName2 is called again with the correct size buffer, however the return code is set to CCI_BUFFER_TOO_SMALL.

e. Add a Trace node to your message flow, and check the output that is generated.

f. Use the flow debugger to debug the flow that contains your node. See Chapter 10, "Testing and debugging message flow applications," on page 3143.

What to do next

When your node behavior is complete and correct, add the new node into your normal palette of nodes in the Message Flow editor, see "Packaging and distributing a user-defined node project" on page 3121. Until you complete this task, the new nodes are available only in your test WebSphere Message Broker Toolkit session on your local system.

Related concepts:

"User-defined input nodes" on page 2990
A user-defined input node is an extension to the broker that provides a new input node in addition to the nodes supplied with the product.

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

"Developing user-defined extensions" on page 2970
A user-defined extension is a component that you design and implement to extend the function of WebSphere Message Broker.

Related tasks:

"Adding a message flow node" on page 1494
When you have created a message flow, add nodes to define its function.

Chapter 10, "Testing and debugging message flow applications," on page 3143
Use the flow debugger to track messages through your message flows and check for errors, or use the Test Client to test your message flows in a safe environment before they are used on a production system.

"Resolving problems with user-defined extensions" on page 3511
Advice for dealing with some common problems that can arise when you work with user-defined extensions

"Packaging and distributing a user-defined node project" on page 3121
Export the user-defined node project to make it available for other users.

"Testing a subflow user-defined node project" on page 3097
Test how the user-defined node is displayed, and how it behaves in your environment.

Related reference:

"Administration Log view" on page 6840
The Administration Log view shows administration requests and their results, changes made to objects, and the result of deployment actions on brokers. The Administration Log view can also describe configuration changes that have been automatically applied as a result of moving the broker from one version to another; for example, after applying maintenance.

"Trace node" on page 4942
Use the Trace node to generate trace records that you can use to monitor the behavior of a message flow.

"User-defined extensions" on page 6411
Reference material that supports the creation and management of your user-defined extensions.

"C language user-defined node API" on page 6416
Learn about the different types of call provided by the API.

"C language user-defined parser API" on page 6538
The C language user-defined parser API consists of two complementary sets of functions that you can use to define the function of your parser.

Related information:

Java user-defined extensions API

Testing a subflow user-defined node project:

Test how the user-defined node is displayed, and how it behaves in your environment.

About this task

You can test how the user-defined node is displayed, and how it behaves in your environment by simulating the node. The user-defined node is displayed in the palette in the Message Flow editor, and any change to the plug-in during the simulation is effective immediately; you are not required to restart the simulation.

- To start the simulation of a user-defined node:
  1. Right-click the user-defined node project, click **Start Simulation**. This action adds all the user-defined nodes that are in the project to the palette in the Message Flow editor under the category that you defined for the project.
  2. You can drag these nodes from the palette onto the Message Flow editor canvas in the same way as the built-in nodes.
- To stop the simulation of a user-defined node:
  1. Right-click the user-defined node project, click **Stop Simulation**.
  2. If any message flows are using the user-defined node, an error is generated because the node is no longer available, and the following message is displayed:

     ```
     User-defined nodes that are contributed by the user-defined node
     project Varname will be removed from the Message Flow editor palette.
     If a copy of these nodes is not available from the plug-in space,
     any message flows that use these nodes might get unresolved node
     errors.
     ```
- If you are using custom editors, compilers, or Java code, these resources cannot be tested in simulation mode. To test and debug these projects and resources you must launch a second WebSphere Message Broker Toolkit by using the following steps:
  1. Right-click the user-defined node project. The menu opens.
  2. Click **Run in New Workbench**.

- If the project contains errors, you receive an error message that gives you the option to continue, or to cancel the action.
- If the project is error-free, the Workspace Launcher window opens.
  a. You can choose to use a new or an existing workspace. To choose an existing workspace, click **Browse**. The second workspace opens.
  b. You can now test and debug by using the user-defined nodes in all error-free user-defined node projects in your first workbench.

- In previous versions of the WebSphere Message Broker Toolkit, you were able to use launch configurations to test custom editors, compilers, or Java code. However, launch configurations are not supported with user-defined nodes that are created from a subflow. To test custom editors, compilers, and Java code with user-defined nodes that are created from a subflow, right-click the user-defined node project and click **Run in New Workbench**.

Related concepts:

"User-defined nodes" on page 2989
User-defined nodes are the main mechanism for extending the functions of WebSphere Message Broker.

"Using a subflow as a user-defined node" on page 3008
You can develop a user-defined node that packages a subflow from scratch, in the same way that you can create any other user-defined node that has its implementation based on Java, or base it on an existing subflow.

Related tasks:

"Debugging the message flow in simulation mode"
Compile, deploy, test, and debug the message flow that includes your user-defined node.

"Packaging and distributing a user-defined node project" on page 3121
Export the user-defined node project to make it available for other users.

"Creating a user-defined node from a subflow" on page 3076
Create user-defined nodes either from scratch, or by using an existing subflow.

Debugging the message flow in simulation mode:

Compile, deploy, test, and debug the message flow that includes your user-defined node.

About this task

You can compile, deploy, test, and debug the message flow that includes your user-defined node in the same way that you test any regular subflow. However, you must set a breakpoint so that the debugger stops at that point, see "Working with breakpoints in the flow debugger" on page 3166.

You can debug the message flow only if the following constraints are met:

- The user-defined node project must be in the workspace.
- The user-defined node project must be in the simulation mode.
- The subflow cannot be debugged after it has been exported as a plug-in, unless the source code is in your workspace.

Related concepts:

"Managing message flow resources" on page 1424
Manage your message flows and associated resources in the WebSphere Message Broker Toolkit.

"User-defined nodes" on page 2989
User-defined nodes are the main mechanism for extending the functions of WebSphere Message Broker.

"Using a subflow as a user-defined node" on page 3008
You can develop a user-defined node that packages a subflow from scratch, in the same way that you can create any other user-defined node that has its implementation based on Java, or base it on an existing subflow.

"Subflows" on page 1030
You can include subflows in your message flows in the same way as you include built-in or user-defined nodes. You can also connect subflows to other nodes in the same way.

Related tasks:

Chapter 11, "Packaging and deploying," on page 3209
Package resources that you create in the WebSphere Message Broker Toolkit, such as message flows, and deploy them to execution groups on brokers.

"Testing a message flow" on page 3146
You can test your message flows using the Test Client.

"Debugging a message flow" on page 3157
Use the tasks described in this section of the documentation to manage and work with the flow debugger.

"Working with breakpoints in the flow debugger" on page 3166
When you have started a debugging session by attaching the debugger to an execution group, you can set breakpoints to control where the message flow will pause.

"Testing a subflow user-defined node project" on page 3097
Test how the user-defined node is displayed, and how it behaves in your environment.

"Packaging and distributing a user-defined node project" on page 3121
Export the user-defined node project to make it available for other users.

"Creating a user-defined node from a subflow" on page 3076
Create user-defined nodes either from scratch, or by using an existing subflow.

Implementing a user-defined parser

Create a user-defined parser to interpret messages with a different format and structure.

Before you begin

Before you start

Read the following topics:

- "Why use a user-defined extension?" on page 2972
- "User-defined parsers" on page 3010

About this task

A loadable implementation library, or a *LIL*, is the implementation module for a C parser (or node). A LIL is a Linux or UNIX shared object or Windows dynamic link library (DLL), that does not have the file extension `.dll` but `.lil`.

The implementation functions that you must write are listed in “C parser implementation functions” on page 6538. The utility functions that are provided by WebSphere Message Broker to help you are listed in “C parser utility functions” on page 6539.

WebSphere Message Broker provides the source for a sample user-defined parser called BipSampPluginParser.c. This example is a simple pseudo-XML parser that you can use in its current state, or you can modify.

The task of writing a parser varies considerably according to the complexity of the bit stream to be parsed. Only the basic steps are described here:

1. “Declaring and defining the parser”
2. “Creating an instance of the parser” on page 3101
3. “Deleting an instance of the parser” on page 3102

Declaring and defining the parser:
About this task

To declare and define a user-defined parser to the broker, you must include an initialization function, bipGetParserFactory, in your LIL. The following steps outline how the broker calls your initialization function and how your initialization function declares and defines the user-defined parser:

The following procedure shows you how to declare and define your parser to the broker:

Procedure

1. The initialization function, bipGetParserFactory, is called by the broker after the LIL has been loaded and initialized by the operating system. The broker calls this function to understand what your LIL is able to do, and how it must be called. For example:

   ```
   CciFactory LilFactoryExportPrefix * LilFactoryExportSuffix
   bipGetParserFactory()
   ```

2. The bipGetParserFactory function calls the utility function cpiCreateParserFactory. This function passes back a unique factory name (or group name) for all the parsers that your LIL supports. Every factory name (or group name) passed back must be unique throughout all the LILs in the broker.
3. The LIL calls the utility function cpiDefineParserClass to pass the unique name of each parser, and a virtual function table of the addresses of the implementation functions.

   For example, the following code declares and defines a single parser called InputxParser:

   ```
   {
    CciFactory* factoryObject;
    int rc = 0;
    CciChar factoryName[] = L"MyParserFactory";
    CCI_EXCEPTION_ST exception_st;

    /* Create the Parser Factory for this parser */
    factoryObject = cpiCreateParserFactory(0, factoryName);
    if (factoryObject == CCI_NULL_ADDR) {

     /* Any local error handling can go here */
    }
    else {
     /* Define the parsers supported by this factory */
   ```

```
 static CNI_VFT vftable = {CNI_VFT_DEFAULT};

/* Setup function table with pointers to parser implementation functions */
vftable.iFpCreateContext            = cpiCreateContext;
vftable.iFpParseBufferEncoded       = cpiParseBufferEncoded;
vftable.iFpParseFirstChild          = cpiParseFirstChild;
vftable.iFpParseLastChild           = cpiParseLastChild;
vftable.iFpParsePreviousSibling     = cpiParsePreviousSibling;
vftable.iFpParseNextSibling         = cpiParseNextSibling;
vftable.iFpWriteBufferEncoded       = cpiWriteBufferEncoded;
vftable.iFpDeleteContext            = cpiDeleteContext;
vftable.iFpSetElementValue          = cpiSetElementValue;
vftable.iFpElementValue             = cpiElementValue;
vftable.iFpNextParserClassName      = cpiNextParserClassName;
vftable.iFpSetNextParserClassName   = cpiSetNextParserClassName;
vftable.iFpNextParserEncoding       = cpiNextParserEncoding;
vftable.iFpNextParserCodedCharSetId = cpiNextParserCodedCharSetId;

cpiDefineParserClass(0, factoryObject, L"InputxParser", &vftable);
}

/* Return address of this factory object to the broker */
return(factoryObject);
}
```

The initialization function must create a parser factory by starting cpiCreateParserFactory. The parser classes supported by the factory are defined by calling cpiDefineParserClass. The address of the factory object (returned by cpiCreateParserFactory) must be returned to the broker as the return value from the initialization function.

For example:

a. Create the parser factory using the cpiCreateParserFactory function:

```
factoryObject = cpiCreateParserFactory(&rc, constParserFactory);
```

b. Define the classes of message supported by the factory using the cpiDefineParserClass function:

```
if (factoryObject) {
   cpiDefineParserClass(&rc, factoryObject, constPXML, &vftable);
  }
else {
    /* Error: Unable to create parser factory */
  }
```

c. Return the address of this factory object to the broker:

```
  return(factoryObject);
}
```

Creating an instance of the parser:
About this task

When the broker has received the table of function pointers, it calls the function cpiCreateContext for each instantiation of the user-defined parser. If you have three message flows that use your user-defined parser, your cpiCreateContext function is called for each of them. This function allocates memory for that instantiation of the user-defined parser to hold the values for the configured attributes. For example:

Procedure

1. Call the cpiCreateContext function:

```
CciContext* _createContext(
  CciFactory* factoryObject,
  CciChar*    parserName,
  CciNode*    parserObject
```

```
){
  static char* functionName = (char *)"_createContext()";
  PARSER_CONTEXT_ST* p;
  CciChar          buffer[256];
```

2. Allocate a pointer to the local context and clear the context area:

```
p = (PARSER_CONTEXT_ST *)malloc(sizeof(PARSER_CONTEXT_ST));

if (p) {
   memset(p, 0, sizeof(PARSER_CONTEXT_ST));
```

3. Save the parser object pointer in the context:

```
p->parserObject = parserObject;
```

4. Save the parser name:

```
CciCharNCpy((CciChar*)&p->parserName, parserName, MAX_NODE_NAME_LEN);
```

5. Return the parser context:

```
return (CciContext*) p;
```

Deleting an instance of the parser:
About this task

Parsers are destroyed when a message flow is deleted or redeployed, or when the execution group process is stopped (using the **mqsistop** command). When a parser is destroyed, it must free any used memory and release any held resources using the cpiDeleteContext function. For example:

```
void cpiDeleteContext(
  CciParser*  parser,
  CciContext* context
){
  PARSER_CONTEXT_ST* pc = (PARSER_CONTEXT_ST *)context ;
  int                rc = 0;

  return;
}
```

Related concepts:

"User-defined extensions in the runtime environment" on page 2980
Before you design and implement user-defined extensions, familiarize yourself with the core components. Ensure that you also understand the basic WebSphere Message Broker runtime architecture.

"Developing user-defined extensions" on page 2970
A user-defined extension is a component that you design and implement to extend the function of WebSphere Message Broker.

"Why use a user-defined extension?" on page 2972
Use a user-defined node or parser when the built-in resources do not provide the required functions.

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

Related tasks:

"Implementing the supplied user-defined extension samples" on page 3017
WebSphere Message Broker provides some sample code to help you understand how to write user-defined nodes and parsers.

"Compiling a C user-defined extension" on page 3047
Compile user-defined extensions in C for all supported operating systems.

Related reference:

"C language user-defined parser API" on page 6538
The C language user-defined parser API consists of two complementary sets of functions that you can use to define the function of your parser.

Extending the capability of a C user-defined parser:

When you have created a C parser, you can extend its capability.

Before you begin

Before you start

Ensure that you have read and understood the following topic:

- "Implementing a user-defined parser" on page 3099

About this task

You can extend the capability of a C parser in the following ways:

- "Implementing the parser functionality"
- "Implementing input functions"
- "Implementing parse functions" on page 3104
- "Implementing output functions" on page 3105
- "Implementing a message header parser" on page 3105

Implementing the parser functionality:
About this task

A parser needs to implement the following types of implementation function:

1. Input functions
2. Parse functions
3. Output functions

Implementing input functions:
About this task

Your parser must implement one, and only one, of the following input functions:

- "cpiParseBuffer" on page 6580
- "cpiParseBufferEncoded" on page 6582
- "cpiParseBufferFormatted" on page 6583

The broker invokes the *input* function when your user-defined parser is required to parse an input message. The parser must tell the broker how much of the input bitstream buffer that it claims to own. In the case of a fixed-size header, the parser claims the size of the header. If the parser is intended to handle the whole message, it claims the remainder of the buffer.

For example:

1. The broker invokes the cpiParseBufferEncoded input function:

```
int cpiParseBufferEncoded(
  CciParser*  parser,
  CciContext* context,
  int         encoding,
```

```
  int         ccsid
){
  PARSER_CONTEXT_ST* pc = (PARSER_CONTEXT_ST *)context ;
  int                rc;
```

2. Get a pointer to the message buffer and set the offset using the cpiBufferPointer utility function:

```
pc->iBuffer = (void *)cpiBufferPointer(&rc, parser);
pc->iIndex = 0;
```

3. Save the format of the buffer:

```
pc->iEncoding = encoding;
pc->iCcsid = ccsid;
```

4. Save the size of the buffer using the cpiBufferSize utility function:

```
pc->iSize = cpiBufferSize(&rc, parser);
```

5. Prime the first byte in the stream using the cpiBufferByte utility function:

```
pc->iCurrentCharacter = cpiBufferByte(&rc, parser, pc->iIndex);
```

6. Set the current element to the root element using the cpiRootElement utility function:

```
pc->iCurrentElement = cpiRootElement(&rc, parser);
```

7. Reset the flag to ensure parsing is reset correctly:

```
pc->iInTag = 0;
```

8. Claim ownership of the remainder of the buffer:

```
  return(pc->iSize);
}
```

Implementing parse functions:
About this task

General parse functions (for example, **cpiParseFirstChild**) are those invoked by the broker when the syntax element tree needs to be created in order to evaluate an ESQL or Java expression. For example, a Filter node uses an ESQL field reference in an ESQL expression. This field reference must be resolved in order to evaluate the expression. Your parser's general parse function is called, perhaps repeatedly, until the requested element is either created, or is known by the parser not to exist.

For example:

```
void cpiParseFirstChild(
  CciParser*  parser,
  CciContext* context,
  CciElement* element
){
  PARSER_CONTEXT_ST* pc = (PARSER_CONTEXT_ST *)context ;
  int                rc;

  if ((!cpiElementCompleteNext(&rc, element)) &&
      (cpiElementType(&rc, element) == CCI_ELEMENT_TYPE_NAME)) {

    while ((!cpiElementCompleteNext(&rc, element))     &&
           (!cpiFirstChild(&rc, element)) &&
           (pc->iCurrentElement))
    {
      pc->iCurrentElement = parseNextItem(parser, context, pc->iCurrentElement);
    }
  }
  return;
}
```

Implementing output functions:
About this task

Your parser must implement one, and only one, of the following output functions:

- "cpiWriteBuffer" on page 6610
- "cpiWriteBufferEncoded" on page 6612
- "cpiWriteBufferFormatted" on page 6613

The broker invokes the *output* function when your user-defined parser is required to serialize a syntax element tree to an output bit stream. For example, a Compute node might have created a tree in the domain of your user-defined parser. When a node, such as an MQOutput node, needs to serialize this tree, the parser is responsible for appending the output bitstream buffer with data that represents the tree that has been built.

For example:

```
int cpiWriteBufferEncoded(
  CciParser*  parser,
  CciContext* context,
  int         encoding,
  int         ccsid
){
  PARSER_CONTEXT_ST* pc = (PARSER_CONTEXT_ST *)context ;
  int                initialSize = 0;
  int                rc = 0;
  const void* a;
  CciByte b;

  initialSize = cpiBufferSize(&rc, parser);
  a = cpiBufferPointer(&rc, parser);
  b = cpiBufferByte(&rc, parser, 0);

  cpiAppendToBuffer(&rc, parser, (char *)"Some test data", 14);

  return cpiBufferSize(0, parser) - initialSize;
}
```

Implementing a message header parser:
About this task

Typically, the incoming message data is of a single message format, therefore one parser is responsible for parsing the entire contents of the message. The class name of the parser that is needed is defined in the `Format` field in the MQMD or the MQRFH2 header of the input message.

However, the message might consist of multiple formats, for example where there is a header in one format followed by data in another format. In this case, the first parser has to identify the class name of the parser that is responsible for the next format in the chain, and so on. In a user-defined parser, the implementation function cpiNextParserClassName is invoked by the broker when it navigates down a chain of parser classes for a message that is composed of multiple message formats.

If your user-defined parser supports parsing a message format that is part of a multiple message format, the user-defined parser *must* implement the cpiNextParserClassName function.

For example:

1. Call the **cpiNextParserClassName** function:

```
void cpiNextParserClassName(
  CciParser*  parser,
  CciContext* context,
  CciChar*    buffer,
  int         size
){
  PARSER_CONTEXT_ST* pc = (PARSER_CONTEXT_ST *)context ;
  int                rc = 0;
```

2. Copy the name of the next parser class to the broker:

```
  CciCharNCpy(buffer, pc->iNextParserClassName, size);

  return;
}
```

Related concepts:

"User-defined extensions in the runtime environment" on page 2980
Before you design and implement user-defined extensions, familiarize yourself with the core components. Ensure that you also understand the basic WebSphere Message Broker runtime architecture.

"Developing user-defined extensions" on page 2970
A user-defined extension is a component that you design and implement to extend the function of WebSphere Message Broker.

"Why use a user-defined extension?" on page 2972
Use a user-defined node or parser when the built-in resources do not provide the required functions.

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

"Implementing the supplied user-defined extension samples" on page 3017
WebSphere Message Broker provides some sample code to help you understand how to write user-defined nodes and parsers.

"Compiling a C user-defined extension" on page 3047
Compile user-defined extensions in C for all supported operating systems.

Related reference:

"C language user-defined parser API" on page 6538
The C language user-defined parser API consists of two complementary sets of functions that you can use to define the function of your parser.

Compiling a C user-defined extension:

Compile user-defined extensions in C for all supported operating systems.

Before you begin

Before you start:

If you create your own user-defined nodes, parsers, and user exits in C, compile them on the operating system on which the target broker is running. Samples are

provided for both nodes and parsers, and are described in “Sample node files” on page 6412 and “Sample parser files” on page 6414. Use the instructions here to compile the samples. If you want to create your own extensions, see the following topics:

- “Creating a user-defined extension in C” on page 3026
- “Implementing a user-defined parser” on page 3099
- “Implementing a user-defined exit” on page 3113

About this task

These instructions use the file names of the supplied samples. If you are compiling your own user-defined extensions, substitute your own file names.

When you compile a user-defined extension that is written in C, you need a compatible compiler. For details of supported compilers, see “Optional software support” on page 3603.

Header files:
About this task

The following header files define the C interfaces:

BipCni.h
Message processing nodes

BipCpi.h
Message parsers

BipCci.h
Interfaces common to both nodes and parsers

BipCos.h
Platform-specific definitions

Compiling:
About this task

Compile the source for your user-defined extension on each of the supported operating systems to create the executable file that the broker calls to implement your user-defined extension. On Linux, UNIX, and z/OS systems, this file is a loadable implementation library (LIL) file; on Windows systems, it is a dynamic load library (DLL) file.

The libraries that you build to contain user-defined nodes or parsers must have the extension `.lil` on all operating systems so that the broker can load them. Libraries that contain user exits must have the extension `.lel` on all operating systems. The examples in this topic show libraries with the extension `.lil`

Refer to the documentation for the compiler that you are using for full details of available compile and link options that might be required for your programs.

Navigate to the directory where your user-defined extension source code is located, and follow the instructions for your operating system:

- AIX
- HP-Itanium
- Linux
- Solaris

- Windows
- z/OS

Compiling on AIX:
About this task

When you compile a user-defined extension that is written in C, use a supported compiler.

The following instructions are for compiling an extension for a 64-bit execution group; 32-bit execution groups are not supported.

```
xlc_r -q64 \
   -I. \
   -I/install_dir/include/plugin \
   -c SwitchNode.c \
   -o SwitchNode.o

xlc_r -q64 \
   -I. \
   -I/install_dir/include/plugin \
   -c TransformNode.c \
   -o TransformNode.o

xlc_r -q64 \
   -I. \
   -I/install_dir/include/plugin \
   -c BipSampPluginUtil.c \
   -o BipSampPluginUtil.o

xlc_r -q64 \
   -I. \
   -I/install_dir/include/plugin \
   -c Common.c \
   -o Common.o

xlc_r -q64 \
   -I. \
   -I/install_dir/include/plugin \
   -c NodeFactory.c \
   -o NodeFactory.o

xlc_r -q64 \
      -qmkshrobj \
      -bM:SRE \
      -bexpall \
      -bnoentry \
      -o SwitchNode.lil SwitchNode.o \
         BipSampPluginUtil.o Common.o NodeFactory.o \
      -L /install_dir/lib \
      -l imbdfplg

chmod a+r SwitchNode.lil
```

Compiling on HP-Itanium:
About this task

When you compile a user-defined extension that is written in C, use a supported compiler.

The following instructions are for compiling an extension for a 64-bit execution group; 32-bit execution groups are not supported.

```
cc +z +e +DD64 -D_HPUX_SOURCE -DTHREADS -D_REENTRANT -Ae \
   -I. \
   -I/install_dir/include \
   -I/install_dir/include/plugin \
   -c BipSampPluginUtil.c \
   -o output_dir/BipSampPluginUtil.o

cc +z +e +DD64 -D_HPUX_SOURCE -DTHREADS -D_REENTRANT -Ae \
   -I. \
   -I/install_dir/include \
   -I/install_dir/include/plugin \
   -c Common.c \
   -o output_dir/Common.o

cc +z +e +DD64 -D_HPUX_SOURCE -DTHREADS -D_REENTRANT -Ae \
   -I. \
   -I/install_dir/include \
   -I/install_dir/include/plugin \
   -c NodeFactory.c \
   -o output_dir/NodeFactory.o

cc +z +e +DD64 -D_HPUX_SOURCE -DTHREADS -D_REENTRANT -Ae \
   -I. \
   -I/install_dir/include \
   -I/install_dir/include/plugin \
   -c SwitchNode.c \
   -o output_dir/SwitchNode.o

cc +z +e +DD64 -D_HPUX_SOURCE -DTHREADS -D_REENTRANT -Ae \
   -I. \
   -I/install_dir/include \
   -I/install_dir/include/plugin \
   -c TransformNode.c \
   -o output_dir/TransformNode.o


ld -b \
   -o output_dir/SwitchNode.lil \
   output_dir/BipSampPluginUtil.o \
   output_dir/Common.o \
   output_dir/NodeFactory.o \
   output_dir/SwitchNode.o \
   output_dir/TransformNode.o \
   -L install_dir/lib \
   -L install_dir/xml4c/lib \
   -L install_dir/merant/lib \
   -L install_dir/jre16/lib/IA64N\
   -L install_dir/jre16/lib/IA64N/server \
   -l imbdfplg

chmod a+r output_dir/SwitchNode.lil
```

Compiling on Linux:
About this task

When you compile a user-defined extension that is written in C, use a supported compiler.

When you compile programs on Linux on POWER, replace the option `-fpic` with `-fPIC` if you want to use dynamic linking and avoid any limit on the size of the global offset table.

The following instructions are for compiling an extension for a 64-bit execution group on Linux on x86-64, Linux on POWER, and Linux on IBM z Systems. 32-bit

execution groups are not supported on those platforms. To compile the extension for a 32-bit execution group on Linux on x86, replace `-m64` with `-m32` in the compile and link examples.

```
g++ -c -O2 -m64 -Wall -Wno-format-y2k -fpic \
   -I. \
   -I/install_dir/include \
   -I/install_dir/include/plugin \
   -D_LINUX -D_THREADS -D_POSIX_PTHREAD_SEMANTICS -D_REENTRANT \
   TransformNode.c

g++ -c -O2 -m64 -Wall -Wno-format-y2k -fpic \
   -I. \
   -I/install_dir/include \
   -I/install_dir/include/plugin \
   -D_LINUX -D_THREADS -D_POSIX_PTHREAD_SEMANTICS -D_REENTRANT \
   SwitchNode.c

g++ -c -O2 -m64 -Wall -Wno-format-y2k -fpic \
   -I. \
   -I/install_dir/include \
   -I/install_dir/include/plugin \
   -D_LINUX -D_THREADS -D_POSIX_PTHREAD_SEMANTICS -D_REENTRANT \
   BipSampPluginUtil.c

g++ -c -O2 -m64 -Wall -Wno-format-y2k -fpic \
   -I. \
   -I/install_dir/include \
   -I/install_dir/include/plugin \
   -D_LINUX -D_THREADS -D_POSIX_PTHREAD_SEMANTICS -D_REENTRANT \
   Common.c

g++ -c -O2 -m64 -Wall -Wno-format-y2k -fpic \
   -I. \
   -I/install_dir/include \
   -I/install_dir/include/plugin \
   -D_LINUX -D_THREADS -D_POSIX_PTHREAD_SEMANTICS -D_REENTRANT \
   NodeFactory.c

g++ -m64 -O2 -o samples.lil \
   TransformNode.o \
   SwitchNode.o \
   BipSampPluginUtil.o \
   Common.o NodeFactory.o \
   -shared -lc -lnsl -ldl \
   -L/install_dir/lib -limbdfplg
```

These commands create the file `samples.lil` that provides TransformNode and SwitchNode objects.

Compiling on Solaris:
About this task

When you compile a user-defined extension that is written in C, use a supported compiler.

The following instructions are for compiling an extension for a 64-bit execution group on Solaris on SPARC; 32-bit execution groups are not supported. To compile the extension for a default 64-bit execution group on Solaris on x86-64, replace `-xarch=v9` with `-xarch=amd64` in the compile examples; 32-bit execution groups are not supported.

```
cc -xarch=v9 -mt \
   -I. \
   -I/install_dir/include \
```

```
   -I/install_dir/include/plugin \
   -c SwitchNode.c \
   -o output_dir/SwitchNode.o

cc -xarch=v9 -mt \
   -I. \
   -I/install_dir/include \
   -I/install_dir/include/plugin \
   -c TransformNode.c \
   -o output_dir/TransformNode.o

cc -xarch=v9 -mt \
   -I. \
   -I/install_dir/include \
   -I/install_dir/include/plugin \
   -c BipSampPluginUtil.c \
   -o output_dir/BipSampPluginUtil.o

cc -xarch=v9 -mt \
   -I. \
   -I/install_dir/include \
   -I/install_dir/include/plugin \
   -c NodeFactory.c \
   -o output_dir/NodeFactory.o

cc -xarch=v9 -mt \
   -I. \
   -I/install_dir/include \
   -I/install_dir/include/plugin \
   -c Common.c \
   -o output_dir/Common.o

cc -xarch=v9  -xcode=pic32 -mt -G \
   -o output_dir/SwitchNode.lil \
      output_dir/SwitchNode.o \
      output_dir/BipSampPluginUtil.o \
      output_dir/NodeFactory.o \
      output_dir/Common.o \
   -L /install_dir/lib \
   -l imbdfplg

chmod a+r output_dir/SwitchNode.lil
```

Compiling on Windows:
About this task

When you compile a user-defined extension that is written in C, use a supported compiler.

Ensure that you include a space between `SwitchNode.c` and `BipSampPluginUtil.c`, and also between `-link` and `/DLL`.

Enter the command as a single line of input; in the following example the lines are split to improve readability.

```
cl /VERBOSE /LD /MD /Zi /EHsc /I.
   /Iinstall_dir\include\plugin
   SwitchNode.c BipSampPluginUtil.c Common.c
   NodeFactory.c TransformNode.c
   -link /DLL install_dir\lib\imbdfplg.lib
   /OUT:SwitchNode.lil
```

If you have correctly set the *LIB* environment variable, you do not have to specify the full paths to the LIB files.

Compiling on z/OS:
About this task

When you compile a user-defined extension that is written in C, use a supported compiler.

Force your link to use prelinker or linker by setting the _CC_STEPS variable to -1:

```
export _CC_STEPS=-1
```

Alternatively, add these two lines to your makefile to export it:

```
_CC_STEPS=-1
.EXPORT : _CC_STEPS
```

To create optimized builds, use **-2** in place of **-g** in the following commands:

```
cc -c \
 -Wc,LP64 -g -W0,langlvl\(extended\),EXPORTALL,float\ieee\) \
 -Wc,xplink \
 -W0,LIST\(./SwitchNode.lst\) \
 -I. -I${install_dir}/include \
 -I${install_dir}/include/plugin \
 -I${install_dir}/sample/include \
 -I${install_dir}/sample/plugin \
 -o ./SwitchNode.o ./SwitchNode.c
```

```
cc -c \
 -Wc,LP64 -g -W0,langlvl\(extended\),EXPORTALL,float\ieee\) \
 -Wc,xplink \
 -W0,LIST\(./TransformNode.lst\) \
 -I. -I${install_dir}/include \
 -I${install_dir}/include/plugin \
 -I${install_dir}/sample/include \
 -I${install_dir}/sample/plugin \
 -o ./SwitchNode.o ./TransformNode.c
```

```
cc -c \
 -Wc,LP64 -g -W0,langlvl\(extended\),EXPORTALL,float\ieee\) \
 -Wc,xplink \
 -W0,LIST\(./BipSampPluginUtil.lst\) \
 -I. -I${install_dir}/include \
 -I${install_dir}/include/plugin \
 -I${install_dir}/sample/include \
 -I${install_dir}/sample/plugin \
 -o ./BipSampPluginUtil.o ./BipSampPluginUtil.c
```

```
cc -c \
 -Wc,LP64 -g -W0,langlvl\(extended\),EXPORTALL,float\ieee\) \
 -Wc,xplink \
 -W0,LIST\(./Common.lst\) \
 -I. -I${install_dir}/include \
 -I${install_dir}/include/plugin \
 -I${install_dir}/sample/include \
 -I${install_dir}/sample/plugin \
 -o ./Common.o ./Common.c
```

```
cc -c \
  -Wc,LP64 -g -W0,langlvl\(extended\),EXPORTALL,float\ieee\) \
 -Wc,xplink \
 -W0,LIST\(./NodeFactory.lst\) \
 -I. -I${install_dir}/include \
 -I${install_dir}/include/plugin \
 -I${install_dir}/sample/include \
 -I${install_dir}/sample/plugin \
 -o ./NodeFactory.o ./NodeFactory.c
```

```
cc \
 -Wl,DLL,LP64 -g -Wl,p,map -Wl,LIST=ALL,MAP,XREF,REUS=RENT \
 -Wl,xplink \
 -o ./SwitchNode.lil ./SwitchNode.o ./BipSampPluginUtil.o \
 ./Common.o ./NodeFactory.o \
 ${install_dir}/lib/libimbdfplg.x
```

Run the following command to set the file permissions of the user-defined extension to `group read` and to be executable:

```
chmod a+rx {output_dir}/SwitchNode.lil
```

Related concepts:

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

"Developing user-defined extensions" on page 2970
A user-defined extension is a component that you design and implement to extend the function of WebSphere Message Broker.

Related tasks:

"Creating a user-defined extension in C" on page 3026
You must complete a series of tasks to create user-defined extensions that use the C language.

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

"Implementing a user-defined exit"
You can develop and deploy a user-defined exit.

"Creating message catalogs" on page 3138
Create your own message catalogs to write tailored entries to the local error log.

"Creating the user interface representation of a user-defined node in the WebSphere Message Broker Toolkit" on page 3079
When you are developing a user-defined node in Java or C only, you must create the user interface representation of the node in the WebSphere Message Broker Toolkit.

"Testing a user-defined node" on page 3094
When you have created and installed the required resources, you can test your user-defined node.

"Packaging and distributing a user-defined node project" on page 3121
Export the user-defined node project to make it available for other users.

Related reference:

"User-defined extensions" on page 6411
Reference material that supports the creation and management of your user-defined extensions.

"Support for 32-bit and 64-bit platforms" on page 3589
WebSphere Message Broker operates in 32-bit mode or 64-bit mode, on supported operating systems.

"Optional software support" on page 3603
The products listed here are not required, but might be useful. Except where stated, these products are not supplied with WebSphere Message Broker.

Implementing a user-defined exit

You can develop and deploy a user-defined exit.

Before you begin

Before you start:

- Read "User exits" on page 3015.

About this task

The following topics describe the steps required to develop and deploy a user-defined exit.

Procedure

1. "Developing a user exit"
2. "Deploying a user exit" on page 3116

Related concepts:

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

"User exits" on page 3015
A user exit is user-provided custom software, written in C, to track data passing through message flows.

"Why use a user-defined extension?" on page 2972
Use a user-defined node or parser when the built-in resources do not provide the required functions.

Related tasks:

"Developing a user exit"
Develop a user exit by declaring it, implementing its behavior, then compiling it.

"Deploying a user exit" on page 3116
Deploy your user exit to the broker.

Related reference:

"User-defined extensions" on page 6411
Reference material that supports the creation and management of your user-defined extensions.

Developing a user exit:

Develop a user exit by declaring it, implementing its behavior, then compiling it.

About this task

To develop a user exit, follow these steps.

Procedure

1. Declare the user exit.

   Declare a user exit by using the bipInitializeUserExits function to specify the following properties:

   a. Name (used to register and control the active state of the exit)
   b. User context storage
   c. A function to be invoked (for one or more Event Types)
2. Implement the user exit behavior.

When the user exit is declared, a set of functions is registered, and these functions are invoked when specific events occur. The behavior of the user exit is provided by implementing these functions. The following table lists the events and their associated functions:

Event	Function
A message is dequeued from the input source	cciInputMessageCallback
A message is propagated to the node for processing	cciPropagatedMessageCallback
A request message is sent to the output node's transport, and transport-specific destination information is written to "WrittenDestination" in the LocalEnvironment	cciOutputMessageCallback
The node completes processing	cciNodeCompletionCallback
The transaction ends	cciTransactionEventCallback

3. Your user exit code must implement the cleanup function.

   The user exit library must implement the bipTerminateUserExits function. This function is invoked as the ExecutionGroup's process is ending, and your user exit must clear up any resources allocated during the bipInitializeUserExits function.

4. Compile.

   Use your existing process for your environment to compile your user exit. The supported C compilers are shown in “Optional software support” on page 3603. See “Compiling a C user-defined extension” on page 3047 for more details.

5. Link the compiled code to a library with the extension `.lel` that exports the bipInitializeUserExits and bipTerminateUserExits functions.

Related concepts:

“User exits” on page 3015
A user exit is user-provided custom software, written in C, to track data passing through message flows.

Related tasks:

“Deploying a user exit” on page 3116
Deploy your user exit to the broker.

“Exploiting user exits” on page 2985
Your message flows can benefit from user exits.

Related reference:

“**mqsicreatebroker** command” on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

“**mqsichangebroker** command” on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

“**mqsireportflowuserexits** command” on page 3933
Use the **mqsireportflowuserexits** command to report the list of active and inactive user exits for the specified broker, execution group, or message flow.

“**mqsichangeflowuserexits** command” on page 3751
Use the **mqsichangeflowuserexits** command to set the list of active or inactive user exits. A list of active and a list of inactive user exits is maintained for each execution group and message flow. The effective state of user exits for a given flow is decided when the flow starts.

Deploying a user exit:

Deploy your user exit to the broker.

Before you begin

Before you start:

- Write and compile the user exit code. See "Developing a user exit" on page 3114.
- Ensure that the exit:
  1. Is in a library that has the extension `.lel`
  2. Exports the functions bipInitializeUserExits and bipTerminateUserExits

About this task

You can set the state of the user exit dynamically to active, or inactive, on a per-message flow basis without restarting the broker.

To deploy the user exit:

Procedure

1. Install the user exit code on a broker.

   The library containing the user exit code must be installed on a file system that can be accessed by the broker. For example, the file must have read and execute authority for the user ID under which the broker runs. The broker looks in the following places for libraries that contain user exits:

   - The broker property `UserExitPath` defines a list of directories separated by colons (semicolons on Windows). Use the `-x` flag on the **mqsicreatebroker** or **mqsichangebroker** command to set this property for execution groups for each broker.
   - Alternatively, you can append the directory containing the directory that holds the extension files to the environment variable MQSI_USER_EXIT_PATH associated with the environment in which the broker is running.

   If both are set, the environment variable takes precedence. All the directories in the environment variable are searched in the order in which they appear in the variable, then all the directories in the broker property are searched in the order in which they appear in the property.
2. Load the user exit library into the broker's processes.

   When the user exit library has been installed on the broker, you must load it in one of the following ways:

   - Stop and restart the broker.
   - Run the **mqsireload** command to restart the execution group processes.
3. Activate the user exit.

   User exits can be active or inactive, and are inactive by default. You can change the state of a user exit dynamically by using the **mqsichangeflowuserexits** command on a per-flow basis, without having to restart the broker. You can also change the default state for a set of user exits to active on a per-broker basis by using the **mqsichangebroker** command; in this case, you do have to restart the broker.

   To set the default user exit state for a broker:

   a. Stop the broker.

b. Set the `activeUserExits` property of the broker by using the **mqsichangebroker** command.

c. Start the broker and check the system log to ensure that all execution groups start without error. If any invalid user exit names are specified (that is, the user exit is not provided by any library loaded by the execution group), a BIP2314 message is written to the system log and all flows in the execution groups fail to start unless you take one of the following actions:

   - Provide a library in the user exit path that implements the exit; then run the **mqsireload** command, or restart the broker, to load an exit from the library.
   - Run the **mqsichangeflowuserexits** command to remove the exit from both the active and inactive lists.

You can also override the default user exit state for a broker. You can use the **mqsichangeflowuserexits** command to activate, or deactivate, user exits on a per-execution group or per-message flow basis, with the order of precedence being message flow then execution group. When multiple exits are active for a flow, the broker starts them in the order that is defined by the **mqsichangeflowuserexits** command.

Results

Related concepts:

"User exits" on page 3015
A user exit is user-provided custom software, written in C, to track data passing through message flows.

Related tasks:

"Developing a user exit" on page 3114
Develop a user exit by declaring it, implementing its behavior, then compiling it.

"Exploiting user exits" on page 2985
Your message flows can benefit from user exits.

Related reference:

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

"**mqsireportflowuserexits** command" on page 3933
Use the **mqsireportflowuserexits** command to report the list of active and inactive user exits for the specified broker, execution group, or message flow.

"**mqsichangeflowuserexits** command" on page 3751
Use the **mqsichangeflowuserexits** command to set the list of active or inactive user exits. A list of active and a list of inactive user exits is maintained for each execution group and message flow. The effective state of user exits for a given flow is decided when the flow starts.

"**mqsireload** command" on page 3909
Use the **mqsireload** command to request the broker to stop and restart execution groups.

Packaging and distributing user-defined extensions

When you have created and tested a user-defined extension, you can package and distribute it.

Before you begin

Before you start:

Complete the following tasks:

- "Implementing user-defined extensions" on page 3019
- "Testing a user-defined node" on page 3094

About this task

When you have created and tested your user-defined extension, you can distribute these resources to other computers.

For user-defined extensions created in Java or C:

1. Package and install the user-defined extensions. To package and install the resources that make up the workbench representation of your user-defined node, see "Packaging and distributing a user-defined node project" on page 3121.
2. Copy the files generated by the compilation step to all the computers on which you have created brokers that might need these resources. See "Installing user-defined extension runtime files on a broker" on page 3125. For a more automated approach, see "Installing a user-defined extension to current and past versions of WebSphere Message Broker" on page 3128.

For user-defined nodes created from subflows:

- Package and install the user-defined nodes implemented as a subflow, see "Packaging and distributing a user-defined node project" on page 3121.

Related concepts:

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Installing a user-defined node" on page 1496
Develop message flows that use a user-defined node.

Related reference:

"User-defined extensions" on page 6411
Reference material that supports the creation and management of your user-defined extensions.

Packaging a Java user-defined node:

How to package a Java user-defined node.

Before you begin

Before you start

You must have a user-defined node written in Java. This node can be one of the provided sample nodes that are described in "Sample node files" on page 6412, or a node that you have created yourself by using the instructions in either "Creating a message processing or output node in Java" on page 3062 or "Creating an input node in Java" on page 3055.

About this task

You can package a user-defined node in two ways:

- **PAR**

  A Plug-in Archive (PAR) is the deployment unit for Java user-defined nodes. The PAR contains the user-defined node classes and, if required as dependencies, can contain JAR files. A PAR file is a compressed file with a `.par` file extension. The directory structure in the `.par` file has the following format:

  – `/classes`

    The user-defined node classes are stored in this location.

  – `/lib`

    JAR files that are required by the user-defined node are stored in this location. This directory is optional because it might not be necessary to include JAR files.

  The following procedure describes how to package an example user-defined node, *parexamplenode*. In this example, the PAR is to be contained in *par.example.parexamplenode.class* with a JAR file dependency *dependency.jar*.

  1. Create the directory structure; for example:
     – `/classes/par/example/parexamplenode.class`
     – `/lib/dep.jar`
  2. Issue a file compression command to create the PAR; for example:

     ```
     jar cvf parexample.par classes lib
     ```

  The PAR must be placed in the LIL path that is specified in "Installing user-defined extension runtime files on a broker" on page 3125.

- **JAR**

  User-defined nodes can be packaged by using a simple JAR. For example, if your node is defined in `example/jarexamplenode.class`, create the JAR by using the `jar cvf jarexample.jar example` command.

  The preferred way to package a Java user-defined node is to use a PAR file, because all dependencies can be packaged with the node, and each node is loaded in a separate class loader. For information about loading classes, see "User-defined node class loading" on page 3120.

  The JAR must be placed in the LIL path that is specified in "Installing user-defined extension runtime files on a broker" on page 3125.

Deployment dependencies:
About this task

If a user-defined node requires an external package, the package can be deployed in one of following ways:

- The external packages can be added to the `/lib` directory in the deployed PAR.
- For external packages that are shared between several node types, the packages can be added to one of the following locations:
  – One of the `shared-classes` directories. For more details of these directories, see "Java shared classloader" on page 2637.
  – The `CLASSPATH` environment variable, where all user-defined nodes that are in the broker installation can access the packages

Related concepts:

"User-defined node class loading" on page 3120
Details the Java classes packaging options and loading order precedence for

user-defined nodes.

Related tasks:

"Compiling a Java user-defined node" on page 3074
When you have created the code for your Java user-defined node, you must compile it for your operating system.

"Java shared classloader" on page 2637
Loads all the JAR files located within the shared-classes directories. The precedence order of loading is dictated by the directories the JAR files are located in.

User-defined node class loading:

Details the Java classes packaging options and loading order precedence for user-defined nodes.

Java user-defined node classes can be packaged and loaded in two ways:

- Plug-in Archive (PAR) file; a broker defined format - Each PAR file is given it's own class loader which isolates any classes it loads from any other part of the WebSphere Message Broker, including nodes in other PAR files.
- Standard Java archive (JAR) file - All JAR files containing plug-in nodes are loaded into the same classloader and can access classes in JAR files containing other user-defined nodes.

For both packaging mechanisms, if the classloader cannot find a required class within the package it defers to the shared class loader to find the required class. The shared classloader looks in a set of directories on the broker machine and loads any JAR files found. It can be used to install any required JAR files that do not need to be repeatedly deployed, such as client libraries that the Java compute nodes need to use. For more details, see "Java shared classloader" on page 2637.

If the required class cannot be found in any of the deployed JAR files, or in the JAR files installed in the shared classes directories, a classloader containing all of the broker supplied classes is checked (for example: this classloader contains the `jplugin2.jar`), followed by the classpath, and then finally the Java virtual machine (JVM) system classloader.

Two key points must be considered when deciding which of the above mechanisms are used to load a class:

- Isolation between different applications (for example: adding classes to the classpath makes them available to every part of WebSphere Message Broker and can cause conflicts).
- Delegation from one classloader to another can only occur in one direction. If a class is resolved in the shared classloader, then it cannot directly create classes in the PAR classloader.

User-defined nodes class loading search paths:
User-defined nodes package in a PAR file

The broker uses the following search path to find user-defined node classes:

1. `/classes` to locate classes in the deployed PAR file.
2. `/lib` to locate any JAR files in the deployed PAR file.
3. *`workpath`*`/config/`*`<my_broker_name>`*`/`*`<my_eg_label>`*`/shared-classes` to locate any JAR files in the execution group `shared-classes` directory.

4. *workpath*/shared-classes/ to locate any JAR files in the top level shared-classes directory.
5. CLASSPATH environment variable.

User-defined nodes package in a JAR file

The broker uses the following search path to find user-defined node classes:

1. The deployed JAR file.
2. *workpath*/config/<*my_broker_name*>/<*my_eg_label*>/shared-classes to locate any JAR files in the execution group shared-classes directory.
3. *workpath*/shared-classes/ to locate any JAR files in the top level shared-classes directory.

Related tasks:

"Compiling a Java user-defined node" on page 3074
When you have created the code for your Java user-defined node, you must compile it for your operating system.

"Packaging a Java user-defined node" on page 3118
How to package a Java user-defined node.

"Java shared classloader" on page 2637
Loads all the JAR files located within the shared-classes directories. The precedence order of loading is dictated by the directories the JAR files are located in.

Packaging and distributing a user-defined node project:

Export the user-defined node project to make it available for other users.

About this task

To export the user-defined node project, you can package the user-defined node project as plug-in JAR files or as an update site:

- "Packaging as plug-in JAR files" on page 3122
- "Packaging as an update site" on page 3122

What to do next

You can now install the plug-in on all the computers on which your WebSphere Message Broker Toolkit users might want to use them, following the instructions in "Installing a user-defined node" on page 1496.

Related concepts:

"User-defined nodes" on page 2989
User-defined nodes are the main mechanism for extending the functions of WebSphere Message Broker.

"Using a subflow as a user-defined node" on page 3008
You can develop a user-defined node that packages a subflow from scratch, in the same way that you can create any other user-defined node that has its implementation based on Java, or base it on an existing subflow.

Related tasks:

"Debugging the message flow in simulation mode" on page 3098
Compile, deploy, test, and debug the message flow that includes your user-defined node.

"Creating a user-defined node from a subflow" on page 3076
Create user-defined nodes either from scratch, or by using an existing subflow.

"Testing a subflow user-defined node project" on page 3097
Test how the user-defined node is displayed, and how it behaves in your environment.

"Message sets packaged with a user-defined node" on page 1497
Adding message sets to an update site.

Packaging as plug-in JAR files:

Export the user-defined node project as plug-in JAR files to make it available for other users.

About this task

If you use this option, the user-defined node user must copy the user-defined node plug-ins into the plug-ins folder in the WebSphere Message Broker Toolkit. However, the user-defined node user can write a script to automate the copying process, so that the user-defined nodes are deployed automatically.

To package your user-defined node projects so that they are available in the environment of the user, complete the following tasks:

Procedure

1. Right-click the project that you want to package, select **Package**. The Package and Distribute User-Defined Nodes wizard opens.
2. Select **Plug-in jars**. Click **Next**.
3. Select the plug-ins and fragments that you want to use. Projects upon which the subflow user-defined node depends, such as message set projects, are automatically included and are not shown in the list.
4. Navigate through the wizard, completing the fields as required. Click **Finish**.

Related concepts:

"User-defined nodes" on page 2989
User-defined nodes are the main mechanism for extending the functions of WebSphere Message Broker.

Related tasks:

"Packaging as an update site"
Create an installation site for Web page distribution of the user-defined node projects that includes or references other projects or plug-ins. You can either create a new update site, or use an existing one.

"Packaging and distributing a user-defined node project" on page 3121
Export the user-defined node project to make it available for other users.

"Message sets packaged with a user-defined node" on page 1497
Adding message sets to an update site.

"Installing a user-defined node" on page 1496
Develop message flows that use a user-defined node.

Packaging as an update site:

Create an installation site for Web page distribution of the user-defined node projects that includes or references other projects or plug-ins. You can either create a new update site, or use an existing one.

About this task

To package your user-defined node projects so that they are available in the environment of the user, complete the following tasks:

Procedure

1. Right-click the project that you want to package, select **Package**. The Package and Distribute User-Defined Nodes wizard opens.
2. Select **Installation package**. You can either create a new installation site or use an existing installation site in the workspace.
   - To create a new installation site:
     a. Select **Create a new package**, and in the **Create a new package** field, enter a name for the package. If a project with that name exists in the workspace, you are asked if you want to delete the project so that you can continue with packaging.
     b. Click **Next**.
     c. Optional: Select any projects and plug-ins that you want to include in your package:
        1) The top section shows the user-defined node projects that are available in your workspace. Select the projects that you want to include in your package.
        2) The bottom section shows all the Eclipse plug-ins that the selected user-defined node projects require. WebSphere Message Broker Toolkit built-in plug-ins that have a plug-in identifier that starts with `com.ibm.etools.mft` are not shown. The Eclipse plug-ins that are required are a combination of the following plug-ins:
           – All Eclipse plug-ins that are listed as the dependency plug-ins in the plug-in manifest for the user-defined node project
           – All Eclipse plug-ins that contributed user-defined nodes that are being used by one of your selected user-defined node projects
        3) Click **Next**.
     d. Optional: Further customize your user-defined node package by adding in any detailed information that you want to include. Click **Next**.
     e. Click **Finish**. When you click **Finish**, the selected user-defined node project and its dependent user-defined node projects in the workspace are packaged into an Eclipse feature, and an Eclipse Update Site is built. The Eclipse feature and Eclipse update site project use the name that you have given to the package appended with `.feature` or `.site`. Any space in the package name is replaced by a dot (.).
     f. The Distribution window opens. To save the instructions to the clipboard, click **Save the instruction above to clip board**.
     g. To select a destination for your user-defined node package, in the "Web distribution" section, click **Copy project *Your project name*.** . The "Copy package" window opens. Select the location, click **OK**.
   - To use an existing installation site you must have already created an installation site.
     a. Select **Update user-defined nodes in an existing package**, select an installation site from the list. Click **Finish**. When you click **Finish**, the selected user-defined node project and its dependent user-defined node projects in the workspace are packaged into an Eclipse feature, and an Eclipse Update Site is built. The Eclipse feature and Eclipse update site

project use the name that you have given to the package appended with `.feature` or `.site`. Any space in the package name is replaced by a dot (.).

b. The Distribution window opens. To save the instructions to the clipboard, click **Save the instruction above to clip board**.

c. To select a destination for your user-defined node package, in the "Web distribution" section, click **Copy project *Your project name.*** . The "Copy package" window opens. Select the location, click **OK**.

Related concepts:

"User-defined nodes" on page 2989
User-defined nodes are the main mechanism for extending the functions of WebSphere Message Broker.

Related tasks:

"Packaging as plug-in JAR files" on page 3122
Export the user-defined node project as plug-in JAR files to make it available for other users.

"Packaging and distributing a user-defined node project" on page 3121
Export the user-defined node project to make it available for other users.

"Message sets packaged with a user-defined node" on page 1497
Adding message sets to an update site.

Installing from an update site:

If you have packaged your user-defined nodes into an update site, use the software update mechanism to install the user-defined nodes.

About this task

To install your user-defined nodes from an update site, use the following steps:

Procedure

1. Click **Help** > **Software Updates**. The Software Updates and Add-ons window opens.
2. Click the **Available Software** tab. A list of available sites is displayed. If the update site that you want to use is not listed, click **Add Site**. The Add Site window opens. Select the location that you want to use, click **OK**.
3. In the **Available Software** tab, select the site that you want to use, click **Install**.

Related concepts:

"User-defined nodes" on page 2989
User-defined nodes are the main mechanism for extending the functions of WebSphere Message Broker.

Related tasks:

"Packaging and distributing a user-defined node project" on page 3121
Export the user-defined node project to make it available for other users.

"Updating and uninstalling from an update site"
You can update, uninstall, or revert the configuration of any user-defined node plug-ins that have been previously installed from an update site.

Updating and uninstalling from an update site:

You can update, uninstall, or revert the configuration of any user-defined node plug-ins that have been previously installed from an update site.

About this task

To update, uninstall, or revert the configuration of any user-defined node plug-ins, complete the following steps:

Procedure

1. Click **Help** > **Software Updates**. The Software Updates and Add-ons window opens.
2. Click the **Installed Software** tab.
3. Select the previously installed user-defined node plug-in.
4. Click **Update**, **Uninstall**, or **Revert Configuration** depending on the action you want to perform.

Related concepts:

"User-defined nodes" on page 2989
User-defined nodes are the main mechanism for extending the functions of WebSphere Message Broker.

"Using a subflow as a user-defined node" on page 3008
You can develop a user-defined node that packages a subflow from scratch, in the same way that you can create any other user-defined node that has its implementation based on Java, or base it on an existing subflow.

Related tasks:

"Debugging the message flow in simulation mode" on page 3098
Compile, deploy, test, and debug the message flow that includes your user-defined node.

"Creating a user-defined node from a subflow" on page 3076
Create user-defined nodes either from scratch, or by using an existing subflow.

"Testing a subflow user-defined node project" on page 3097
Test how the user-defined node is displayed, and how it behaves in your environment.

"Installing from an update site" on page 3124
If you have packaged your user-defined nodes into an update site, use the software update mechanism to install the user-defined nodes.

"Packaging and distributing a user-defined node project" on page 3121
Export the user-defined node project to make it available for other users.

Installing user-defined extension runtime files on a broker:

Install the compiled runtime files for your user-defined extension on the broker on which you want to test its function.

Before you begin

Before you start

- Create and compile your user-defined extension using the procedure described in "Compiling a Java user-defined node" on page 3074 or "Compiling a C user-defined extension" on page 3047.
  - The files that have been created for extension created in C depend on the underlying broker operating system:

    Windows A dynamic link library (DLL), named with a file type of `.lil`.

    Linux A shared object, again with a file type of `.lil`.

    UNIX A shared object, again with a file type of `.lil`.

z/OS A shared object, with a file type of `.lil`.

– For Java nodes, a Java Archive file (JAR), with a file type of `.jar` (on all operating systems).

- If you have created a user-defined node, you must also complete the task "Creating the user interface representation of a user-defined node in the WebSphere Message Broker Toolkit" on page 3079.

About this task

This task instructs you to stop and restart brokers. This action is required in all but the two circumstances described in step 4 later in this section, although if you do stop and restart the broker, you can ensure that anyone with an interest in a particular execution group is made aware that recent changes have been made.

This task is applicable to user-defined nodes written in Java or C only.

To install runtime files on the broker:

Procedure

1. Stop the broker on which you want to install your compiled or packaged user-defined extension file (files with extension `.lil`, `.jar`, `.par`, `.pdb`, or `.lel`)
2. Create a directory if you do not already have one for this purpose. Add the directory to the LILPATH by using the **mqsichangebroker** command.

   CAUTION:
   Do not put the `.lil`, `.jar`, `.par`, `.pdb`, or `.lel` files in the WebSphere Message Broker installation directory, because they might be overwritten by the broker.
3. Put your user-defined file in the directory, and make sure that the broker has access to it. For example, on Linux or UNIX, use the **chmod 755*** command on the file.
4. Stop and restart the broker to implement the change and to ensure that the existence of the new file is detected. A broker restart is not necessary in the following circumstances:
   - If you have created an execution group in the WebSphere Message Broker Toolkit, and nothing is yet deployed to it, you can add the `.lil`, `.pdb`, `.jar`, `.par`, or `.lel` file to your selected directory.
   - If something has already been deployed to the execution group that you want to use, add the `.lil`, `.pdb`, `.jar`, `.par`, or `.lel` file to your selected directory, and issue the **mqsireload** command to restart the group. You cannot overwrite an existing file on the Windows system when the broker is running, because of the file lock that is put in place by the operating system.

   Use these two approaches with care, because any execution group that is connected to the same broker also detects the new `.lil`, `.pdb`, `.jar`, `.par`, or `.lel` files when that execution group restarts, or when something is first deployed to that execution group.
5. Repeat the previous steps for every broker that needs the user-defined extension file and user-defined node plug-in file. If all of your brokers are on the same operating system type, you can build the user-defined extension file once and distribute it to each of your systems.

   If you have a cluster, for example, that includes one AIX, one Solaris, and one Windows broker, you must build the user-defined extension files separately on each operating system type.

Windows On Windows, the `.pdb` file provides symbolic information that is used when stack diagnostic information is displayed in the event of access violations or other software malfunctions.

6. For C user-defined extensions, store the `.pdb` file in the same directory as the `.lil` file to which it corresponds.
7. Use either the **mqsichangebroker** command or the **mqsicreatebroker** command, as appropriate, to specify to the broker the directory that contains the user-defined extension file.

   When you have installed a user-defined extension, it is referred to by its schema and name, just like a message flow.

Results

The broker loads the user-defined extension files during initialization. After loading the files, the broker calls the registration functions in the user-defined extension and records what nodes or parsers the user-defined extension supports.

A C user-defined extension implements a node or parser factory that can support multiple nodes or parser types. For more information, see "Node and parser factory behavior" on page 2982. Java users are not required to write a node factory.

Related concepts:

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

"User-defined extensions in the runtime environment" on page 2980
Before you design and implement user-defined extensions, familiarize yourself with the core components. Ensure that you also understand the basic WebSphere Message Broker runtime architecture.

"Developing user-defined extensions" on page 2970
A user-defined extension is a component that you design and implement to extend the function of WebSphere Message Broker.

Related tasks:

"Testing a user-defined node" on page 3094
When you have created and installed the required resources, you can test your user-defined node.

Related reference:

"User-defined extensions" on page 6411
Reference material that supports the creation and management of your user-defined extensions.

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsireload** command" on page 3909
Use the **mqsireload** command to request the broker to stop and restart execution groups.

Installing a user-defined extension to current and past versions of WebSphere Message Broker:

Install user-defined extensions that you have developed yourself, or have acquired from an independent software vendor, with the minimum of user intervention.

Before you begin

Before you start

Complete the following tasks:

1. "Compiling a Java user-defined node" on page 3074, "Compiling a C user-defined extension" on page 3047, or "Creating a user-defined node from a subflow" on page 3076
2. "Creating the user interface representation of a user-defined node in the WebSphere Message Broker Toolkit" on page 3079
3. "Testing a user-defined node" on page 3094
4. "Packaging and distributing a user-defined node project" on page 3121

About this task

You must install user-defined extensions on all appropriate WebSphere Message Broker computers, and, if the extension is a user-defined node, on the WebSphere Message Broker Toolkit computers (user-defined parsers have no WebSphere Message Broker Toolkit component). Components can be installed separately, or as part of one process. The components can be on different systems, therefore check that the installations are completed on all affected systems.

If an extension writes messages to user trace, you must update the environment variable MQSI_CONSOLE_NLSPATH (Windows systems), or NLSPATH (all other systems), so that the **mqsiformatlog** command can find the message catalog.

The WebSphere Message Broker Toolkit installation:
Before you begin

Before installing a user-defined node, check the version of the WebSphere Message Broker Toolkit to which you are installing, because a specific version of the WebSphere Message Broker Toolkit might be a prerequisite of the user-defined extension, or it might require specific files to run.

To determine the version, see "Detecting installed versions of WebSphere Message Broker" on page 3132.

Procedure

1. Copy your files to a directory that the WebSphere Message Broker Toolkit can access, so that you can view your user-defined node in the WebSphere Message Broker Toolkit session. Choose one of the following options:
   - Add your new plug-in JAR file directly into the `dropins` directory where the WebSphere Message Broker Toolkit is installed. For example, on Windows 32-bit, add the file to `C:\Program Files\IBM\WMBT700\dropins`.

     If you choose this option, you might find it difficult to manage your plug-ins files safely if you later remove or replace them. Also, your system administrator might want to control security and access on computers that

are used by more than one user, and might not set permissions for users to write to the primary installation directories.

- Create Eclipse link files to the directories in which you maintain your plug-in files.

  For details about how to create link files, see the developerWorks article about using Eclipse features (the section entitled "Using link files to manage an Eclipse install").

  a. Linux On Linux: Delete any existing `.eclipse` directory in your home directory, and restart the WebSphere Message Broker Toolkit. For example, remove the following directory, where *userid* is your user identification: `/home/userid/.eclipse`

  b. Create a directory called `eclipse` in a suitable location in your file system; the directory structure that contains the `eclipse` directory is not significant.

  c. Within the `eclipse` directory, create directories named `features` and `plugins`.

  d. Windows On Windows: Under `C:\Program Files\IBM\WMBT700` create a new links folder, and create a file called *name*`.link`, where *name* is a name that you have chosen. Type the following text `path=C:/path/to/your/dir/` into the *name*`.link` file, where *path/to/your/dir/* is the path to your directory.

     Example: If you put your plug-ins into `C:\Temp\MyPlugins\eclipse\plugins\`, the content of the link file is `path=C:/Temp/MyPlugins`.

2. Restart your WebSphere Message Broker Toolkit session for the changes to take effect.

Broker installations:
About this task

You might be required to detect the versions of WebSphere Message Broker that are installed, to ensure that the correct LIL file is loaded by the correct level of the broker. See “Detecting installed versions of WebSphere Message Broker” on page 3132.

To add `.jar` or `.lil` files to broker installations on WebSphere Message Broker Version 6.0 and later, see “Installing user-defined extension runtime files on a broker” on page 3125.

Installing a user-defined extension for single broker:
About this task

Version 7.0 and Version 8.0

- To make an extension accessible from only one broker on the system, modify the *UserLilPath* setting for the broker by specifying the `-l` parameter on the **mqsicreatebroker** or **mqsichangebroker** command.

Version 6.1 and earlier

- To make a 32-bit extension accessible from only one broker on the system, modify the *UserLilPath* setting for the broker by specifying the `-l` parameter on the **mqsicreatebroker** or **mqsichangebroker** command.
- To make a 64-bit extension accessible from only one broker on the system, modify the *UserLilPath64* setting for the broker by specifying the `-r` parameter on the **mqsicreatebroker** or **mqsichangebroker** command.

For more information, see "**mqsicreatebroker** command" on page 3831 and "**mqsichangebroker** command" on page 3723.

Installing a user-defined extension for multiple brokers:
About this task

Version 7.0 and Version 8.0

- To affect all brokers on a system, you modify the system LILPATH. Append the directory containing the directory that holds the extension files to the environment variable MQSI_LILPATH. MQSI_LILPATH64 is not valid at this version.

Version 6.1 and earlier

- To affect all brokers on a system, you modify the system LILPATH. Append the directory containing the directory that holds the extension files to the environment variable MQSI_LILPATH (for 32-bit extensions) or MQSI_LILPATH64 (for 64-bit extensions).

Make this change by creating a custom environment script in the working directory:

- Linux UNIX On Linux and UNIX systems: `/var/mqsi/common/profiles`
- Windows On Windows: `%ALLUSERSPROFILE%\Application Data\IBM\MQSI\common\profiles` where `%ALLUSERSPROFILE%` is the environment variable that defines the system working directory. The default directory depends on the operating system:
  - On Windows XP and Windows Server 2003: `C:\Documents and Settings\All Users\Application Data\IBM\MQSI\common\profiles`
  - On Windows 7, Windows Vista and Windows Server 2008: `C:\ProgramData\IBM\MQSI\common\profiles`

  The actual value might be different on your computer.

You can give the environment script any name, but the file extension must be `.cmd` on Windows and `.sh` on all other platforms. The script can perform all the operations of a shell script, but you must limit the scope to only appending the following variables:

MQSI_LILPATH
Defines the directories to search for plug-ins

CLASSPATH
Defines the locations that Java searches for additional classes

MQSI_CONSOLE_NLSPATH
On Windows: Defines the location of message catalogs (DLL files)

NLSPATH
On Linux and UNIX: Defines the location of message catalogs (CAT files)

PATH
Defines the location of executable files. On Windows: This variable also defines the location of dependent libraries.

LIBPATH / SHLIB_PATH / LD_LIBRARY_PATH
On UNIX and Linux: Defines the location of dependent libraries.

Example scripts:

About this task

Windows On Windows: This example shows the environment profile for MyExtension, which is installed in `C:\Program Files\MyExtensions` on Windows 32-bit, or in `C:\Program Files(x86)\MyExtensions` on Windows 64-bit.

The script is called `MyExtension.cmd` and is stored in the working directory. The default location is `%ALLUSERSPROFILE%\Application Data\IBM\MQSI\common\profiles` where the default setting for the environment variable`%ALLUSERSPROFILE%` depends on the operating system:

- On Windows XP and Windows Server 2003: `C:\Documents and Settings\All Users\`.
- On Windows Vista and Windows Server 2008: `C:\ProgramData\`.

The actual value might be different on your computer.

The Windows script contains the following content:

```
REM Added by MyExtension install, do not modify
set MQSI_LILPATH=%MQSI_LILPATH%;"C:\Program Files\MyExtension\bin"
```

Linux UNIX On Linux and UNIX: This example shows the environment profile for MyExtension, which is installed in `/opt/MyExtension`. The script is called `MyExtension.sh` and is stored in the working directory `/var/mqsi/common/profiles/`.

The Linux script contains the following content:

```
#!/bin/ksh
# Added by MyExtension install, do not modify
export MQSI_LILPATH=/opt/MyExtension/lil${MQSI_LILPATH:+":"${MQSI_LILPATH}}
```

You can test the following variables in the profile script, for example if you want to ensure that a user-defined extension runs only on a specific version of the broker:

MQSI_FILEPATH
The full path to the installed file for WebSphere Message Broker

MQSI_WORKPATH
The full path to the configuration data for WebSphere Message Broker

MQSI_VERSION
WebSphere Message Broker version, in the form `version.release.modification.fix`

MQSI_VERSION_V
The value of WebSphere Message Broker major version

MQSI_VERSION_R
The value of WebSphere Message Broker release

MQSI_VERSION_M
The value of WebSphere Message Broker modification number

MQSI_VERSION_F
The value of WebSphere Message Broker fix level

Related concepts:

"User-defined input nodes" on page 2990
A user-defined input node is an extension to the broker that provides a new input

node in addition to the nodes supplied with the product.

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

"Developing user-defined extensions" on page 2970
A user-defined extension is a component that you design and implement to extend the function of WebSphere Message Broker.

Related tasks:

"Creating the user interface representation of a user-defined node in the WebSphere Message Broker Toolkit" on page 3079
When you are developing a user-defined node in Java or C only, you must create the user interface representation of the node in the WebSphere Message Broker Toolkit.

"Detecting installed versions of WebSphere Message Broker"
A user-defined extension can detect which version of WebSphere Message Broker is installed.

"Installing user-defined extension runtime files on a broker" on page 3125
Install the compiled runtime files for your user-defined extension on the broker on which you want to test its function.

"Creating message catalogs" on page 3138
Create your own message catalogs to write tailored entries to the local error log.

"Testing a user-defined node" on page 3094
When you have created and installed the required resources, you can test your user-defined node.

Related reference:

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

Detecting installed versions of WebSphere Message Broker:

A user-defined extension can detect which version of WebSphere Message Broker is installed.

About this task

Use the conditions described here to test for particular version or versions. If expected conditions are not met, a component might not have installed correctly, or might have become corrupted. Check the status of the installed component and the local logs to identify and resolve any errors.

Detecting installed versions on Windows:
About this task

Use the following instructions in your installer scripts on Windows to test for the following versions. To detect each version, look for the registry key given for each version. In the examples shown, x can be any integer.

WebSphere Message Broker Version 6.0 toolkit

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\
WMBT60\DisplayVersion = 6.x.x.x
```

WebSphere Message Broker (and later) toolkit

Check for the presence of the files `\IBM\Installation Manager\installed.xml` and `\IBM\Installation Manager\installRegistry.xml` in the working directory.

The default working directory is `%ALLUSERSPROFILE%\Application Data\IBM\MQSI` where `%ALLUSERSPROFILE%` is the environment variable that defines the system working directory. The default directory depends on the operating system:

- On Windows XP and Windows Server 2003: `C:\Documents and Settings\All Users\Application Data\IBM\MQSI`
- On Windows Vista and Windows Server 2008: `C:\ProgramData\IBM\MQSI`

The actual value might be different on your computer; use `%ALLUSERSPROFILE%` to ensure that you access the correct location.

WebSphere Message Broker Version 6.0 (and later) runtime components

Open the file `install.properties` in the working directory.

The default working directory is `%ALLUSERSPROFILE%\Application Data\IBM\MQSI` where `%ALLUSERSPROFILE%` is the environment variable that defines the system working directory. The default directory depends on the operating system:

- On Windows XP and Windows Server 2003: `C:\Documents and Settings\All Users\Application Data\IBM\MQSI`
- On Windows Vista and Windows Server 2008: `C:\ProgramData\IBM\MQSI`

The actual value might be different on your computer; use `%ALLUSERSPROFILE%` to ensure that you access the correct location.

Each line in the file is of the following format:

```
PATH_TO_INSTALLATION = VERSION_OF_INSTALLATION
```

For example, if you have installed Version 7.0 GA level in the default installation location, the line contains the following text:

```
C\:\\ Program\ Files\\IBM\\MQSI=7.0.0.0
```

The backslash character \ is interpreted as an escape character. It is inserted before each non-alphabetic and non-numeric character in the string to preserve the character. A colon, a space, and several backslash characters are escaped in this example.

Parse each line of the file to detect all the installed versions and the directory paths for the runtime components, and ignore all duplicates and non-existent directories.

Detecting installed versions on Linux and UNIX systems:

About this task

Linux and UNIX systems do not have a common packaging method: you must check which files are present in the file system. Look for the following files for each version of WebSphere Message Broker that you want to detect.

WebSphere Message Broker Version 6.0 toolkit

To detect Version 6 and later toolkits, look for the existence of /etc/IBM/WebSphereMessageBrokersToolkit/products/com.ibm.wbmt.

To determine the version, use the following code example. Shell-script notation is used in this code: '-e' means if file exists.

```
if [ -e /etc/IBM/WebSphereMessageBrokersToolkit/products/com.ibm.webt  ]
  # Event Broker installed
      if [ -e `grep location /etc/IBM/WebSphereMessageBrokersToolkit/products/
    com.ibm.webt | sed 's/location=//'`/webt_prod/version.txt` ]
        # it is FP1 or greater
        get version from version.txt
      else
        #version is 6.0
      fi
    fi
  if [ -e /etc/IBM/WebSphereMessageBrokersToolkit/products/com.ibm.wmbt ]
    # Message Broker installed
    if [ -e `grep location /etc/IBM/WebSphereMessageBrokersToolkit/products/
    com.ibm.wmbt | sed 's/location=//'`/wmbt_prod/version.txt` ]
      #It is FP1 or greater
      get version from version.txt
    else
      #version is 6.0
    fi
  fi
```

WebSphere Message Broker (and later) toolkit

Check for the presence of the files /var/ibm/InstallationManager/installed.xml and /var/ibm/InstallationManager/installRegistry.xml.

WebSphere Message Broker Version 6.0 (and later) runtime components

To detect Version 6.0 and later runtime components, look for the file /var/mqsi/install.properties. Each line in this file contains an installation path and V.R.M.F version information.

:

Related concepts:

"User-defined input nodes" on page 2990
A user-defined input node is an extension to the broker that provides a new input node in addition to the nodes supplied with the product.

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

"Developing user-defined extensions" on page 2970
A user-defined extension is a component that you design and implement to extend the function of WebSphere Message Broker.

Related tasks:

"Creating the user interface representation of a user-defined node in the WebSphere Message Broker Toolkit" on page 3079
When you are developing a user-defined node in Java or C only, you must create the user interface representation of the node in the WebSphere Message Broker Toolkit.

"Testing a user-defined node" on page 3094
When you have created and installed the required resources, you can test your user-defined node.

"Installing a user-defined extension to current and past versions of WebSphere Message Broker" on page 3128
Install user-defined extensions that you have developed yourself, or have acquired from an independent software vendor, with the minimum of user intervention.

Updating a user-defined extension:

On all systems, you can change a user-defined extension file.

About this task

You must stop and restart the broker for your changes to show, However, you are not required to stop and restart the broker in the following two scenarios:

- If you have created an execution group in the WebSphere Message Broker Toolkit, but have not yet deployed to it, you can add the `.lil`, `.pdb`, and `.jar` files to your chosen directory.
- If an object has already been deployed to the execution group that you want to use, add the `.lil`, `.pdb`, and `.jar` files to your chosen directory and use the **mqsireload** command to restart the group. You cannot overwrite an existing file on the Windows system when the broker is running because a file lock is put in place by the operating system.

These two scenarios must be used with caution, because any execution group that is connected to the same broker also detects the new `.lil`, `.pdb`, and `.jar` files when that execution group is restarted, or when an object is first deployed to it. If you restart the broker, you must ensure that anyone with an interest in a particular execution group is made aware that recent changes have been made to the broker.

These two scenarios assume that you have used either the **mqsichangebroker** command or the **mqsicreatebroker** command to notify the broker of the directory in which the user-defined extension files have been placed.

Procedure

To change a user-defined extension file:

1. Stop the broker by using the **mqsistop** command.
2. Update or overwrite the `.lil` or `.jar` file.
3. Restart the broker by using the **mqsistart** command.

Related concepts:

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

"Developing user-defined extensions" on page 2970
A user-defined extension is a component that you design and implement to extend the function of WebSphere Message Broker.

Related tasks:

"Creating the user interface representation of a user-defined node in the WebSphere Message Broker Toolkit" on page 3079
When you are developing a user-defined node in Java or C only, you must create the user interface representation of the node in the WebSphere Message Broker Toolkit.

"Testing a user-defined node" on page 3094
When you have created and installed the required resources, you can test your

user-defined node.

"Packaging and distributing a user-defined node project" on page 3121
Export the user-defined node project to make it available for other users.

Related reference:

"**mqsireload** command" on page 3909
Use the **mqsireload** command to request the broker to stop and restart execution groups.

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsistop** command" on page 3972
Use the **mqsistop** command to stop the specified component.

"**mqsistart** command" on page 3965
Use the **mqsistart** command to start the specified broker if all initial verification tests complete successfully.

"User-defined extensions" on page 6411
Reference material that supports the creation and management of your user-defined extensions.

Uninstalling a user-defined extension from the broker:

Remove a user-defined extension file from the broker.

About this task

1. Stop the broker by using the **mqsistop** command.
2. Delete the `.lil` or `.jar` file from the directory you created when installing the user-defined extension. See "Installing user-defined extension runtime files on a broker" on page 3125.
3. Restart the broker by using the **mqsistart** command.

Related concepts:

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

"Developing user-defined extensions" on page 2970
A user-defined extension is a component that you design and implement to extend the function of WebSphere Message Broker.

Related tasks:

"Installing user-defined extension runtime files on a broker" on page 3125
Install the compiled runtime files for your user-defined extension on the broker on which you want to test its function.

"Creating the user interface representation of a user-defined node in the WebSphere Message Broker Toolkit" on page 3079
When you are developing a user-defined node in Java or C only, you must create the user interface representation of the node in the WebSphere Message Broker Toolkit.

"Testing a user-defined node" on page 3094
When you have created and installed the required resources, you can test your user-defined node.

"Packaging and distributing a user-defined node project" on page 3121
Export the user-defined node project to make it available for other users.

Related reference:

"User-defined extensions" on page 6411
Reference material that supports the creation and management of your user-defined extensions.

Using error logging from a user-defined extension:

Program user-defined extensions to write entries in the local error log.

About this task

In most circumstances, user-defined extensions should use exceptions to report errors. However, you can provide information about significant events, error or otherwise, for problem determination and operational purposes. The details that you supply are included in predefined message text that is extracted from a message source or catalog.

- In C code, use the utility function CciLog or CciLogW to report events. Two of the arguments that you pass to this function, `messageSource` and `messageNumber`, define the event source (catalog) and the integer representation of a message within that source, respectively.

  You can also write trace information, using CciUserTrace, CciUserTraceW, CciUserDebugTrace, and CciUserDebugTraceW when tracing and debugging is active.
- In Java code, use the class `MbService`, which provides static methods to log information to the event log. To log messages to the event log, package your messages into a standard Java resource bundle. You can use one of the three logging methods, passing in the resource bundle name and the message key. The message is fully resolved, and is then inserted as a single insert into the appropriate broker message as shown:
  - logInformation(...) - BIP4360 Java user-defined node information: *user message*
  - logWarning(...) - BIP4361 Java user-defined node warning: *user message*
  - logError(...) - BIP4362 Java user-defined node error: *user message*

You can write messages that are defined in the product message catalog (`BIPmsgs`), to which you can add your own text as an argument. If you prefer, you can create your own message catalog, so that you can create more complex messages, or share a message catalog with other applications. If you want to create your own message catalog, see "Creating message catalogs" on page 3138.

- Windows On Windows systems, messages are written to the Windows event log.
- Linux UNIX z/OS On Linux, UNIX, and z/OS systems, messages are written to the SYSLOG facility.

The description here covers exceptions that are raised during normal message flow processing. You must also provide for exceptions that are raised when you deploy and configure a message flow. Messages that result from these configuration exceptions are reported back to the WebSphere Message Broker Toolkit for display to the WebSphere Message Broker Toolkit user. Create an appropriately-named Java properties file to contain your messages, then copy the file to each computer on which you are running the WebSphere Message Broker Toolkit, so that your messages can be displayed.

Related concepts:

"User-defined input nodes" on page 2990
A user-defined input node is an extension to the broker that provides a new input node in addition to the nodes supplied with the product.

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"Developing user-defined extensions" on page 2970
A user-defined extension is a component that you design and implement to extend the function of WebSphere Message Broker.

Related tasks:

"Creating message catalogs"
Create your own message catalogs to write tailored entries to the local error log.

Creating message catalogs:

Create your own message catalogs to write tailored entries to the local error log.

About this task

In some error and other situations, you might choose to write information to the error log so that you can track what is happening in a message flow. You can use the Throw and Trace built-in nodes to generate entries in the log, or you can create your own nodes and user exits, and write entries in the log from your user-defined extensions.

You can write either or both of the following sets of messages:

- A fixed set of messages that are provided in the product message catalog. This set provides a range of numbers for Throw nodes (BIP3001 to BIP3001), and a second range for Trace nodes (BIP3051 to BIP3001). A third range (BIP2951 to BIP2999) is provided for the ESQL statements LOG and THROW.

  When you use these messages, you can also provide additional text that is displayed in the message text.
- Your own messages, created in your own message catalog. You can use this additional catalog to define specialized message content, and you can include variables or inserts that are determined by the code that generates the message. You can also share your own message catalog with other applications that are not associated with WebSphere Message Broker.

When you throw an exception from ESQL by using a THROW statement, the ESQL code adds an extra leading insert that contains the name of the current component. The rest of the inserts that are provided by the ESQL script follow this leading insert. Therefore, you must consider this insert when you are writing your own message catalog.

The instructions in this topic describe how to create message catalogs for C programs. If you want to create a Java resource bundle, refer to the documentation for the Java 2 Platform, Standard Edition.

Read the section appropriate to your broker operating system:

- "Building and installing a Windows message source"
- "Creating an XPG/4 catalog for Linux, UNIX, and z/OS" on page 3140

Building and installing a Windows message source:
About this task

On Windows, you must create your additional message catalog as a DLL file. The DLL file contains definitions of your event messages, which the event viewer can display in a readable format, based on the event message written by your application. When you compile a message catalog, a header file is created that defines symbolic values for each event message number you have created. You must include the header file in your application.

To create an event source for the Windows Event Log Service:

Procedure

1. Create a message compiler input (`.mc`) file with the source for your event messages. Refer to the Microsoft Developer Network Web site, and search on `.mc file` for details on the format of this input file.
2. Compile the message file to create a resource compiler input file:

   ```
   mc -v -w -s -h c:\mymessages -r c:\mymessages mymsg.mc
   ```

   where `c:\mymessages` is the location of the output files and *mymsg.mc* is the name of the input file.

   The message compiler produces an output header (`.h`) file that contains symbolic `#defines` that map to each message number that is coded in the input `.mc` file. Include this header file when you compile a user-defined extension source file that uses a utility function (for example, CciLog) to write an event message that you have defined. The `messageNumber` argument to utility function must use the appropriate value that is hash-defined in the output header file.
3. Compile the output file (`.rc`) from the message compiler to create a resource file (`.res`):

   ```
   RC /v output_file.rc
   ```
4. Create a resource DLL file from the `.res` file:

   ```
   LINK /DLL /NOENTRY resource_file.res
   ```
5. Append the location of the resource DLL file to the MQSI_CONSOLE_NLSPATH environment variable, for example:

   ```
   set MQSI_CONSOLE_NLSPATH=%MQSI_CONSOLE_NSPATH%;c:\messages
   ```

   You can do this by creating a custom environment script in your working directory. The default location is `%ALLUSERSPROFILE%\Application Data\IBM\MQSI\common\profiles` where `%ALLUSERSPROFILE%` is the environment variable that defines the system working directory. The default directory depends on the operating system:

   - On Windows XP and Windows Server 2003: `C:\Documents and Settings\All Users\Application Data\IBM\MQSI\common\profiles`
   - On Windows Vista and Windows Server 2008: `C:\ProgramData\IBM\MQSI\common\profiles`

The actual value might be different on your computer.

6. Install the event source into the Windows Event Log Service:
   a. Start the Windows Registry Editor:
      ```
      regedit
      ```
   b. Create a new registry subkey for your user-defined extension under the existing structure:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\EventLog\Application
```

Right-click **Application** and select **New** > **Key**. The new key is created immediately under the Application key (not under the WebSphere Message Broker key). You must give the key the name that you specify for the `messageSource` on a utility function in your user-defined extension (for example, CciLog) or as the property of the built-in node that you have included in your message flow.

Create the following values for this entry:

`EventMessageFile`
Set the value of this string to contain the fully qualified path for the DLL file that you have created to contain your messages. This entry represents the message catalog.

`TypesSupported`
Set the DWORD value to "7".

Creating an XPG/4 catalog for Linux, UNIX, and z/OS:
About this task

On Linux, UNIX, and z/OS systems, messages are written to the SYSLOG facility. If you want to use your own message catalog, you must create an XPG/4 message catalog.

The process for creating a message catalog (a `.cat` file) depends on the operating system on which you are creating it. The commands that you use are typically **`gencat`** (create or modify a message catalog) and **`dspcat`** (to display all or part of a message catalog). The **`gencat`** command merges text files that contain your message text, to create or modify a formatted catalog. The text files typically have a file extension of `.msg`.

You must append the location of the message catalog to the NLSPATH environment variable. You can use %L and %N to represent the locale and the catalog name, for example:

```
export NLSPATH=${NLSPATH}:${MY_INST_PATH}/messages/%L/%N:${MY_INST_PATH}/messages/En_US/%N
```

In this example, the English version is hardcoded later in the search path, ensuring that messages are displayed even in locales for which no `.cat` file exists.

The messages that you define in the `.msg` files can include variables that are substituted at run time. Such variables must be of the format *{number}*, where *{number}* is the message insert number, surrounded by braces. The first message insert is numbered 0. For example:

```
1234 "MSG1234E: \
Syntax Error. \n
The value '{0}' is not valid for property '{1}'.\n
Correct it and then reissue the command.\n"
```

If you create a message catalog on one operating system, you cannot port it to another operating system because the catalogs are binary-encoded. However, you can use the same `.msg` files as input to the **gencat** command on another system.

See the relevant information in the documentation for your operating system. For example:

- AIX For AIX, see the *Commands Reference* in the information center.
- z/OS For z/OS, see the *UNIX System Services Command Reference* in the z/OS V1R8.0 LibraryCenter.

You must also check the information about additional supported locales, if you want to use messages in locales other than US English.

Related concepts:

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

"Developing user-defined extensions" on page 2970
A user-defined extension is a component that you design and implement to extend the function of WebSphere Message Broker.

Related tasks:

"Using error logging from a user-defined extension" on page 3137
Program user-defined extensions to write entries in the local error log.

Related information:

z/OS V1R8.0 LibraryCenter

pSeries and AIX Information Center

Chapter 10. Testing and debugging message flow applications

Use the flow debugger to track messages through your message flows and check for errors, or use the Test Client to test your message flows in a safe environment before they are used on a production system.

Before you begin

Before you start:

To use the flow debugger effectively, you must have a basic understanding of message flows and their representation in the WebSphere Message Broker Toolkit. See "Message flows overview" on page 1022.

About this task

The Test Client and the flow debugger are provided as part of the WebSphere Message Broker Toolkit.

- "Testing message flows by using the Test Client" on page 3144

  Use the Test Client to monitor the output nodes in the message flow, and provide information about the path that a test message takes through a message flow.
- "Flow debugger overview" on page 3158

  Learn about the function provided by the flow debugger, and why you might want to use it.
- "Debugging a message flow" on page 3157

  Start the flow debugger and set options to test and debug the message flow.
- "Debugging by using trace" on page 3194

  Use trace in various ways to debug messages flows.

Related concepts:

"Flow debugger overview" on page 3158
Use the flow debugger in the WebSphere Message Broker Toolkit to track messages through your message flows.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Debug perspective" on page 6789

Related tasks:

"Debugging a message flow" on page 3157
Use the tasks described in this section of the documentation to manage and work with the flow debugger.

Related reference:

"Flow debugger shortcuts" on page 6719

"Flow debugger icons and symbols" on page 6720
The Debug perspective uses various debugger icons and symbols.

Testing message flows by using the Test Client

You can test message flows in a safe environment before they are used on a production system by using the Test Client.

You can use the Test Client to send test messages to message flows that use WebSphere MQ, JMS, SOAP, or HTTP input nodes. The Test Client monitors the output nodes in the message flow, and can provide information about the path that a test message takes through a message flow. The Test Client can also provide information about errors that are generated by the message flow.

You can complete the following tasks by using the Test Client:

- "Testing a message flow" on page 3146
- "Configuring the test settings" on page 3148
- "Creating and editing a test message" on page 3152
- "Using the Test Client in trace and debug mode" on page 3155

Related concepts:

"Test Client overview"
Use the Test Client to test message flows in a safe environment before they are used in a production system.

Related reference:

"Test Client" on page 6708
Use the Test Client to test message flow applications.

"Enqueue" on page 6712
Enqueue is the term that is used to describe the process of putting a message on to a WebSphere MQ queue.

"Dequeue" on page 6712
Dequeue is the term that is used to describe the process of removing a message from a WebSphere MQ queue.

Test Client overview

Use the Test Client to test message flows in a safe environment before they are used in a production system.

You can use the Test Client to send test messages to message flows that use any of the following input nodes:

- WebSphere MQ
- JMS
- HTTP
- SOAP
- SCA

Configuring the input message

You can use the Test Client to change the content of test messages that are sent to a message flow, to help you determine if the message flow is working as expected.

WebSphere MQ queues
If your message flow uses WebSphere MQ queues, the Test Client clears the queues before your test messages are sent to the message flow.

XML messages
If the input node in the message flow that you select expects an XML

message from an associated message set, the message structure is provided, and it can be edited to produce the appropriate test message. Alternatively, you can create a new test message, or import an existing message from your file system.

WebSphere MQ and JMS messages
If the message format is WebSphere MQ or JMS, you can also configure an appropriate header for the test message.

Monitoring a flow with the Test Client

The Test Client monitors output nodes in the message flow so that you can see which nodes output messages are received on. When an error message is produced as the message passes along the flow, or when a message is received on an output node, a test event is recorded in the Test Client.

You can view the content of the output message, and view error messages. The details of the test configuration and the test events can be saved as a `.mbtest` file. You can use this file to repeat the test or to review the results later.

Deploying message flows when you use the Test Client

If you change your message flow, you can use the same test configuration to test the changes. The default behavior of the Test Client is to deploy the message flow that you want to test automatically to an execution group, whenever a change is made to the message flow. You can therefore change a message flow, and quickly test the result using the Test Client, without the need to manually deploy your message flows.

The first time that you send a test message to an input node, you configure the execution group to deploy the message flow by using the Deployment location wizard. You can configure the deployment options to override the default behavior of the Test Client to deploy the message flow manually, or to deploy the message flow every time that you pass a test message to the message flow.

Stopping the Test Client

The default behavior of the Test Client is to stop the test when the first output message is received. You can configure the Test Client to wait for multiple output messages to be received. In this case, you stop the test manually. Stopping the test disconnects the monitors that are running, but does not stop the message flow.

Synchronous tests
A synchronous test, such as when the message flow is invoked from an HTTPInput node, is stopped automatically when a reply message is received.

Asynchronous tests
You can stop an asynchronous test, such as when the message flow is invoked from an MQInput node, manually depending on the monitor setting in the configuration panel.

All test events are stopped when the Test Client is closed, and all test monitors removed.

Using The flow debugger with the Test Client

You can run the Test Client using the trace and debug mode to view more information about the path that the message takes through the message flow. A test event is produced when the message passes from one node to the next node in the message flow. The structure of the message is recorded as it leaves each node in the message flow. The flow debugger is launched in the trace and debug mode so that the test message stops at breakpoints that are configured in the message flow.

Related tasks:

"Testing a message flow"
You can test your message flows using the Test Client.

"Configuring the test settings" on page 3148
You can configure the settings in the Test Client to control how your tests are run.

"Creating and editing a test message" on page 3152
To use the Test Client, you must create or edit a test message to send to your message flow.

"Using the Test Client in trace and debug mode" on page 3155
You can run the Test Client in trace and debug mode to trace the path of a test message through a message flow.

"Debug: putting a test message on an input queue" on page 3162
You can put a message on an input queue to test a message flow that you are debugging.

"Debug: getting a test message from an output queue" on page 3164
You can get a message from an output queue to test a message flow that you are debugging.

Related reference:

"Test Client" on page 6708
Use the Test Client to test message flow applications.

Testing a message flow

You can test your message flows using the Test Client.

Before you begin

Before you start:

You must have a broker running. If you do not have an existing broker, create one using the Default Configuration wizard; see "Creating a default configuration" on page 564.

About this task

To test a message flow, complete the following tasks:

Procedure

1. "Opening the Test Client editor" on page 3147
2. "Configuring the test settings" on page 3148
3. "Creating and editing a test message" on page 3152
4. "Selecting the deployment location for the message flow" on page 3154.

Results

The test message is put to the selected input node. The Test Client monitors the output nodes in the message flow and events are generated as the message passes through the message flow. You might need to stop the test manually, depending on the nodes in the message flow and the settings that you have configured in the Test Client.

What to do next

Next:

To test the message flow again, right-click **Invoke Message Flow** in the **Message Flow Test Events** pane and click **Invoke** to start a new test; or click **Duplicate** or **Re-run** to re-run the test using the same message.

Related concepts:

"Test Client overview" on page 3144
Use the Test Client to test message flows in a safe environment before they are used in a production system.

Related tasks:

"Using the Test Client in trace and debug mode" on page 3155
You can run the Test Client in trace and debug mode to trace the path of a test message through a message flow.

Related reference:

"Test Client **Events** tab" on page 6709
You can use the **Events** tab in the Test Client to edit the properties and content of your test messages. You also view test events in the **Events** tab when you run the test.

Opening the Test Client editor

You can open the Test Client editor by using a menu, by right-clicking a message flow, or by right-clicking an input node.

Before you begin

Before you start:

Before you open your message flow in the Test Client, ensure that the flow contains no errors. Errors are shown in the Problems view; for more details, see "Problems view" on page 6787.

About this task

To begin testing your message flow with the Test Client editor, complete the following steps.

Procedure

Open the Test Client editor by using one of the following methods:

- In the Broker Development pane, right-click the message flow that you want to test and click **Test Message Flow**.
- Open the message flow that you want to test, right-click the input node, and click **Test**.

- Open the message flow that you want to test and click **Flow** > **Test**. (This menu option is disabled if there are no input nodes that can be tested.)

Results

The Test Client editor is opened with settings from the message flow.

What to do next

Next:

You can now configure the Test Client settings. For more details, see "Configuring the test settings."

Related concepts:

"Test Client overview" on page 3144
Use the Test Client to test message flows in a safe environment before they are used in a production system.

Related tasks:

"Testing a message flow" on page 3146
You can test your message flows using the Test Client.

"Configuring the test settings"
You can configure the settings in the Test Client to control how your tests are run.

"Creating and editing a test message" on page 3152
To use the Test Client, you must create or edit a test message to send to your message flow.

"Selecting the deployment location for the message flow" on page 3154
You can specify the execution group to which to deploy a message flow by using the Deployment Location wizard within the Test Client.

Configuring the test settings

You can configure the settings in the Test Client to control how your tests are run.

Before you begin

Before you start:

You must complete the following tasks before you can configure test settings:

- "Creating a message flow" on page 1431
- "Opening the Test Client editor" on page 3147

About this task

Use the following topics to help you to configure the settings on the Test Client:

- "Testing a message flow that has WebSphere MQ nodes" on page 3149
- "Testing a message flow that uses JMS nodes" on page 3150
- "Test Client Configuration tab" on page 6713

You can modify settings that relate to all your test configurations using the Test Client preferences; see "Test Client preferences" on page 6716.

Related concepts:

"Test Client overview" on page 3144
Use the Test Client to test message flows in a safe environment before they are used in a production system.

Related tasks:

"Testing a message flow" on page 3146
You can test your message flows using the Test Client.

"Creating and editing a test message" on page 3152
To use the Test Client, you must create or edit a test message to send to your message flow.

"Selecting the deployment location for the message flow" on page 3154
You can specify the execution group to which to deploy a message flow by using the Deployment Location wizard within the Test Client.

Related reference:

"Test Client **Events** tab" on page 6709
You can use the **Events** tab in the Test Client to edit the properties and content of your test messages. You also view test events in the **Events** tab when you run the test.

Testing a message flow that has WebSphere MQ nodes:

You can configure settings in the Test Client for testing message flows that have WebSphere MQ nodes.

Before you begin

Before you start:

Open the Test Client by following the instructions in "Opening the Test Client editor" on page 3147.

About this task

To test a message flow that uses WebSphere MQ nodes, complete the following steps.

Procedure

1. Right-click a message flow and click **Test Message Flow**. The Test Client opens with the settings from the selected message flow.
2. Click the **Configuration** tab to display the Test Client configuration settings.
3. Click **MQ Settings** and select the appropriate options for your test.
4. Click **MQ Message Header "Default Header"** to view the settings for the message header that is used for the test message. You can edit the options for the default header, or you can create a new header to edit by completing the following steps.
   a. Click **MQ Message Headers**.
   b. Click **Add** and enter a unique name for the header.
   c. Optional: Select **Include RFH V2 Header** to define an MQRFH2 header; for further information, see "Test Client Configuration tab" on page 6713.
   d. Edit the header settings.
   e. Click the **Events** tab, and select the appropriate header for your message from the **Header** list.
5. You can use the Test Client to create WebSphere MQ queues that are used in nodes in your message flow. To configure the Test Client to create the queues, complete the following steps.
   a. Click **Window** > **Preferences**.

b. Expand **Broker Development** and click **Message Broker Test Client**.
c. Ensure that **Create queues of input and output nodes of message flows when host name is localhost** is selected and click **OK**.

6. Optional: Save the Test Client configuration in a `.mbtest` file by completing the following steps.
   a. Click **File** > **Save**. The Save Execution Trace wizard opens.
   b. Enter a name for the file, and select a project in which to save the file.
   c. Click **Finish** to save the file.
7. Create a test message to test your message flow by following the instructions in "Creating and editing a test message" on page 3152.

Related concepts:

"Test Client overview" on page 3144
Use the Test Client to test message flows in a safe environment before they are used in a production system.

Related tasks:

"Testing a message flow" on page 3146
You can test your message flows using the Test Client.

"Configuring the test settings" on page 3148
You can configure the settings in the Test Client to control how your tests are run.

Related reference:

"Test Client Configuration tab" on page 6713
Configure your test environment in the **Configuration** tab in the Test Client.

Testing a message flow that uses JMS nodes:

You can configure settings in the Test Client for testing message flows that uses JMS nodes.

Before you begin

Before you start:

Open the Test Client by following the instructions in "Opening the Test Client editor" on page 3147.

About this task

To test a message flow that uses JMS nodes, complete the following steps.

Procedure

1. Right-click a message flow and click **Test Message Flow**. The Test Client opens with the settings from the selected message flow.
2. Click the **Configuration** tab to display the Test Client configuration settings.
3. Click **JMS Settings** and select the appropriate options for your test. You can add references to the client JAR files used to create the JMS connection.

   To add a reference to these JAR files into your test configurations, complete the following steps.
   a. Click **Configure preference settings**. The Test Client preferences are displayed.
   b. Click **Add** and locate the JAR files in your file system.
   c. Click **OK** to add the reference to the JAR files.

d. Ensure that **Use preference settings** is selected on the **Configuration** tab.

4. To create a JMS header, click **JMS Message Headers**, then complete the following steps.
   a. Click **Add** and enter a unique name for the header.
   b. Edit the header settings.
   c. Click the **Events** tab and select the appropriate header for your message from the **Header** list.
5. Optional: Save the Test Client configuration in a `.mbtest` file by completing the following steps.
   a. Click **File** > **Save**. The Save Execution Trace wizard opens.
   b. Enter a name for the file, and select a project in which to save the file.
   c. Click **Finish** to save the file.
6. Create a test message to test your message flow by following the instructions in "Creating and editing a test message" on page 3152.

Related concepts:

"Test Client overview" on page 3144
Use the Test Client to test message flows in a safe environment before they are used in a production system.

Related tasks:

"Testing a message flow" on page 3146
You can test your message flows using the Test Client.

"Configuring the test settings" on page 3148
You can configure the settings in the Test Client to control how your tests are run.

Related reference:

"Test Client Configuration tab" on page 6713
Configure your test environment in the **Configuration** tab in the Test Client.

"JMS events in the Test Client" on page 6717
Use the information in this topic to help you to understand JMS events in the Test Client.

Testing a message flow that has SCAInput nodes:

You can configure settings in the Test Client for testing message flows that have SCAInput nodes that use the MQ Binding type.

Before you begin

Before you start:

Open the Test Client by following the instructions in "Opening the Test Client editor" on page 3147.

About this task

To test a message flow that uses SCAInput nodes:

Procedure

1. Right-click a message flow and click **Test Message Flow**. The Test Client opens with the settings from the selected message flow. If the Binding type is Web services, skip to step 7 on page 3152, otherwise continue with step2 on page 3152

2. Click the **Configuration** tab to display the Test Client configuration settings.
3. Click **MQ Settings** and select the appropriate options for your test.
4. Click **MQ Message Header "Default Header"** to view the settings for the message header that is used for the test message. You can edit the options for the default header, or you can create a new header to edit by completing the following steps.
   a. Click **MQ Message Headers**.
   b. Click **Add** and enter a unique name for the header.
   c. Edit the header settings. If you expect a response, specify the **Reply to queue name** and **Reply to queue manager name**.
   d. Click the **Events** tab and select the appropriate header for your message from the **Header** list.
5. You can use the Test Client to create WebSphere MQ queues that are used in nodes in your message flow. To configure the Test Client to create the queues, complete the following steps.
   a. Click **Window** > **Preferences**.
   b. Expand **Broker Development** and click **Message Broker Test Client**.
   c. Ensure that **Create queues of input and output nodes of message flows when host name is localhost** is selected and click **OK**.
6. Optional: Save the Test Client configuration in a `.mbtest` file by completing the following steps.
   a. Click **File** > **Save**. The Save Execution Trace wizard opens.
   b. Enter a name for the file, and select a project in which to save the file.
   c. Click **Finish** to save the file.
7. Create a test message to test your message flow by following the instructions in "Creating and editing a test message."

Related concepts:

"Test Client overview" on page 3144
Use the Test Client to test message flows in a safe environment before they are used in a production system.

Related tasks:

"Testing a message flow" on page 3146
You can test your message flows using the Test Client.

"Configuring the test settings" on page 3148
You can configure the settings in the Test Client to control how your tests are run.

Related reference:

"Test Client Configuration tab" on page 6713
Configure your test environment in the **Configuration** tab in the Test Client.

Creating and editing a test message

To use the Test Client, you must create or edit a test message to send to your message flow.

Before you begin

Before you start:

Complete these tasks before you edit your test message:

1. "Opening the Test Client editor" on page 3147
2. "Configuring the test settings" on page 3148

About this task

A number of editors are available in the Test Client for creating a test message. The most appropriate editor to use depends upon the type of test message you want to send to your message flow.

- If the input node that you want to send the message to for the test expects an XML message, and the message flow is associated with a message definition, the **Edit as XML structure** editor is available.
- If you want to send an XML message, but do not have a message definition defined, or you want to create a test message that is not in XML format, you can use the **Edit as text** editor.
- If you want to use an existing test message from a workspace resource or file system file, you can use the **Import from external file** editor.
- Alternatively you can import an existing test message into the **Edit as text** editor, or take the generated source from the **Edit as XML structure** editor and paste it into the **Edit as text** editor.

The Test Client provides limited support for the generation of XSD and WSDL test messages. If the Test Client cannot generate the test message automatically, you can create the test message by using either the Edit as text or Import from external file method described in this topic.

Select from the following options to create and edit a test message:

Procedure

- Edit as XML structure:
  1. In the Events tab of the Test Client, select **Edit as XML structure** from the **Body** list.
  2. Edit the entries in the **Value** column for each field to change the content of the test message.
  3. Right-click the fields in the **Edit as XML structure** editor to see additional options for defining the content of the test message. These options include adding message parts and elements, for example if your message has repeating fields.
  4. You can save your file with the updated test message by clicking **File** > **Save**.
  5. To view and copy the generated test message, click **Show Generated Source**.
- Edit as text:
  1. In the Events tab of the Test Client, select **Edit as text** from the **Body** list.
  2. Enter the text content for your test message. You can copy content into the editor from another source by right-clicking in the editor and selecting **Paste**, or import content from an existing test message by clicking **Import Source**.
  3. You can save your file with the updated test message by clicking **File** > **Save**.
- Import from external file:
  1. In the Events tab of the Test Client:
     - If the input node expects an XML message, select **Import from external file** from the **Body** list.
     - If the input node does not expect an XML message, select **Import from external file (Binary and Text)** from the **Body** list.
  2. Select from the following options to locate the file containing your test message:
     - Workspace resource

a. Click **Workspace**. You can then locate your existing test message from your workspace.
b. Click **OK** to use the selected resource.
c. Click **Edit** to open the default editor associated with the resource.

– File system file
a. Click **File system**. You can then locate your existing test message from your file system.
b. Click **Open** to use the selected file.

3. You can save your file with the updated test message file details by clicking **File** > **Save**.

What to do next

Next:

Ensure that you have selected the correct input node to send the test message to in your message flow. Click **Send Message** to send your test message to the selected input node. If this is the first time you have sent a message using this Test Client file, the Deployment Location wizard opens. See "Selecting the deployment location for the message flow."

Related concepts:

"Test Client overview" on page 3144
Use the Test Client to test message flows in a safe environment before they are used in a production system.

Related tasks:

"Testing a message flow" on page 3146
You can test your message flows using the Test Client.

Related reference:

"Test Client **Events** tab" on page 6709
You can use the **Events** tab in the Test Client to edit the properties and content of your test messages. You also view test events in the **Events** tab when you run the test.

Selecting the deployment location for the message flow

You can specify the execution group to which to deploy a message flow by using the Deployment Location wizard within the Test Client.

Before you begin

Before you start:

Before you can test your message flow, you must have configured a broker. The broker must be running. If you do not have an existing broker, you can create one using the Deployment Location wizard within the Test Client by selecting **New local broker**. Alternatively select **Add remote broker** to import a connection to a remote broker.

About this task

When you first send a test message to a message flow using the Test Client, the **Deployment Location** wizard is opened. You can use the wizard to select an execution group to which to deploy the message flow.

Procedure

1. In the Test Client, click **Send Message**, to open the **Deployment Location** wizard.
2. If your broker is not connected, click **Connect**.
3. From the list in the wizard, select the execution group to which to deploy your message flow. You can also create a new execution group from the **Deployment Location** wizard by selecting a broker, and clicking **New Exec Group**.
4. Select **Trace and debug** to display information about each node that the message passes through in the message flow and to run the flow debugger. You can select an option for the trace to **Stop at the beginning of the flow during debugging**. For more information see Using the Test Client in trace and debug mode.
5. Click **Next**.
6. Modify the test settings as required.
7. Click **Finish** to save the settings and deploy the message flow.

What to do next

Next:

You can change the deployment location settings from the **Configuration** tab.

1. Click **Configuration** in the Test Client.
2. Click **Deployment** to display the deployment settings.
3. Click **Change** to open the **Deployment Location** wizard.

Related concepts:

"Test Client overview" on page 3144
Use the Test Client to test message flows in a safe environment before they are used in a production system.

Related tasks:

"Testing a message flow" on page 3146
You can test your message flows using the Test Client.

"Using the Test Client in trace and debug mode"
You can run the Test Client in trace and debug mode to trace the path of a test message through a message flow.

"Creating a broker for a development environment" on page 569
Create a broker by using the WebSphere Message Broker Toolkit on Linux on x86 or Windows.

"Importing broker definitions into the WebSphere Message Broker Explorer" on page 906
Import broker connection details that have been created by another user into your session of the WebSphere Message Broker Explorer or WebSphere Message Broker Toolkit, to use the broker.

Related reference:

"Deployment Location wizard" on page 6716
Use the Deployment Location wizard to set the execution group to which the test message flow is deployed. You can also use the wizard to create a broker, a connection to a remote broker, and a new execution group.

Using the Test Client in trace and debug mode

You can run the Test Client in trace and debug mode to trace the path of a test message through a message flow.

Before you begin

Before you start:

Before you can test your message flow, you must have configured a broker. The broker must be running. You must also have created an execution group to which to deploy your message flows.

About this task

You can use the trace and debug mode to complete the following actions:

- Stop the test message at breakpoints in the message flow by using the flow debugger.
- Trace the message nodes and terminals that the test message passes through.
- See the test message change as it passes through the message flow.
- View a message node, where an exception occurs, and the associated exception message and trace details.

Note: If the message flow does not contain an MQOutput, HTTPReply, JMSOutput, or SOAPReply node, the Test Client stops the test after the test message is sent to the broker. Therefore, you might not see trace events when the flow is run.

All output from the trace and debug mode is written to the "Message flow test events" section on the **Message flow Test Events** tab.

To use the Test Client in trace and debug mode, complete the following steps:

Procedure

1. Configure the flow debug port for the execution group in the WebSphere Message Broker Toolkit:
   a. In the Brokers view, right-click the execution group with which you want to work.
   b. Click **Launch Debugger**.
   c. To set the Flow Debug Port, click **Configure**, and enter a number for the port.
   d. Click **OK**.

   The debugger is now enabled on the selected execution group. Click **Terminate Debugger** to stop the debugger.
2. In the Test Client, click **Send Message** to open the Deployment Location wizard.
3. If your broker is not connected, click **Connect**. From the list in the wizard, select the execution group to which you want to deploy your message flow.
4. Click **Trace and debug**.
5. Optional: If you want the test message to stop at a breakpoint after the input node, click **Stop at the beginning of the flow during debug**.
6. Click **Next** and modify the test settings as required.
7. Click **Finish** to save the settings and deploy the message flow.

What to do next

Next:

You can modify the deployment location settings from run mode to trace and debug mode by using the Deployment Location wizard:

1. Click **Configuration** in the Test Client.
2. Click **Deployment** to display the deployment settings.
3. Click **Change** to open the Deployment Location wizard.

Related concepts:

"Test Client overview" on page 3144
Use the Test Client to test message flows in a safe environment before they are used in a production system.

Related tasks:

"Testing a message flow" on page 3146
You can test your message flows using the Test Client.

Related reference:

"Deployment Location wizard" on page 6716
Use the Deployment Location wizard to set the execution group to which the test message flow is deployed. You can also use the wizard to create a broker, a connection to a remote broker, and a new execution group.

Debugging a message flow

Use the tasks described in this section of the documentation to manage and work with the flow debugger.

Before you begin

Before you start

If you are new to debugging, see: "Flow debugger overview" on page 3158.

Deploy your message flow to an execution group in a broker and make sure that the broker is running. See: "Deploying resources" on page 3234.

About this task

To debug a message flow, perform the following tasks. You might want to vary the tasks you perfom and repeat certain tasks, depending on your particular debugging requirements.

Procedure

1. Start the flow debugger.

   Set the required preferences, then start debugging by attaching the flow debugger to an execution group. You can then send test messages along the flow. See: "Starting the flow debugger" on page 3160.
2. Work with breakpoints.

   Add and manipulate breakpoints in your message flow. See: "Working with breakpoints in the flow debugger" on page 3166.
3. Follow the progress of a test message.

   Use breakpoints to pause the progress of a test message so that you can observe its behavior. See: "Stepping through message flow instances in the debugger" on page 3172.
4. View message data.

View (and change) data in messages, ESQL code, Java code or mappings as debugging progresses. See: “Debugging data” on page 3180.

5. Manage message flows.

   During a debugging session, there are various administrative tasks you might need to do. When you have finished debugging, detach the debugger from the execution group. See: “Managing flows and flow instances during debugging” on page 3187.

Related concepts:

“Flow debugger overview”
Use the flow debugger in the WebSphere Message Broker Toolkit to track messages through your message flows.

“Message flows overview” on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

“Debug perspective” on page 6789

Related tasks:

Chapter 11, “Packaging and deploying,” on page 3209
Package resources that you create in the WebSphere Message Broker Toolkit, such as message flows, and deploy them to execution groups on brokers.

Related reference:

“Flow debugger shortcuts” on page 6719

“Flow debugger icons and symbols” on page 6720
The Debug perspective uses various debugger icons and symbols.

Flow debugger overview

Use the flow debugger in the WebSphere Message Broker Toolkit to track messages through your message flows.

Use the Debug perspective in the WebSphere Message Broker Toolkit to use the flow debugger. For an introduction to the Debug perspective and the views it presents, see “Debug perspective” on page 6789.

You can set breakpoints in a message flow, then step through the flow. While you are stepping through, you can examine and change the message variables and the variables used by ESQL code, Java code, and mappings. You can debug a wide variety of error conditions in flows, including the following:

- Nodes that are wired incorrectly (for example, outputs that are connected to the wrong inputs)
- Incorrect conditional branching in transition conditions
- Unintended infinite loops in flow

From a single WebSphere Message Broker Toolkit, you can attach the debugger to one or more execution groups, and debug multiple flows in different execution groups (and therefore multiple messages) at the same time. However, an execution group can be debugged by only one user at a time. Therefore, if you attach your debugger to an execution group, another user cannot attach a debugger to that same execution group until you have ended your debugging session.

When you debug message flows, use a broker that is not being used in a production environment. Debugging might degrade the performance of all

message flows in the same execution group and those in other execution groups that share the same broker because they might be affected by potential resource contention.

Debugging code and mappings in message flow nodes

You can use the flow debugger to examine the behavior of code and mappings in message flow nodes.

After you have deployed a message flow, you can set a breakpoint just before one of the nodes listed in this section so that, when the flow pauses at the breakpoint, you can step through the code or mappings line by line. This allows you to examine the logic, and check the actions taken and their results. You can set additional breakpoints and you can also examine and change variables.

The following nodes can contain ESQL code modules:
- Compute node
- Filter node
- Database node

The following nodes can contain Java code modules:
- User-defined nodes
- JavaCompute node

The following nodes can contain mappings:
- Mapping node
- DataInsert node
- DataUpdate node
- DataDelete node
- Extract node
- Warehouse node

Restrictions

The following restrictions apply when you debug a message flow:

- You must use the same version of the broker and the WebSphere Message Broker Toolkit; for example, you cannot use the WebSphere Message Broker Toolkit Version 6.1 to debug a message flow that you have deployed to a broker at an earlier version.
- You should not debug message flows over the Internet, because there might be security issues.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Debug perspective" on page 6789

Related tasks:

"Debugging a message flow" on page 3157
Use the tasks described in this section of the documentation to manage and work with the flow debugger.

Chapter 11, "Packaging and deploying," on page 3209
Package resources that you create in the WebSphere Message Broker Toolkit, such as message flows, and deploy them to execution groups on brokers.

Related reference:

"Flow debugger shortcuts" on page 6719

"Flow debugger icons and symbols" on page 6720
The Debug perspective uses various debugger icons and symbols.

Starting the flow debugger

To start the flow debugger, you must attach it to an execution group. When the flow debugger is started, you can introduce test messages to your message flow.

About this task

Complete the following tasks to start the debugger:

Procedure

1. "Attaching the flow debugger to an execution group for debugging"
2. Optional: "Debug: putting a test message on an input queue" on page 3162
3. Optional: "Debug: getting a test message from an output queue" on page 3164

Related concepts:

"Flow debugger overview" on page 3158
Use the flow debugger in the WebSphere Message Broker Toolkit to track messages through your message flows.

Related tasks:

"Debugging a message flow" on page 3157
Use the tasks described in this section of the documentation to manage and work with the flow debugger.

Related reference:

"Flow debugger shortcuts" on page 6719

"Flow debugger icons and symbols" on page 6720
The Debug perspective uses various debugger icons and symbols.

Attaching the flow debugger to an execution group for debugging

Before you can debug your message flow, you must attach the flow debugger to the execution group where your flow is deployed, then start a debugging session.

Before you begin

Before you start:

- If authorisation permissions are required, see: "Authorizing users for broker administration" on page 371
- Create a message flow. See: Chapter 9, "Developing message flow applications," on page 1019
- Deploy your message flow to a broker execution group. See: "Deploying resources" on page 3234
- Start the broker. See: "Starting and stopping a broker" on page 921

About this task

From a single WebSphere Message Broker Toolkit, you can attach the flow debugger to multiple execution groups that are running on the same or on different host computers, and debug their flows (and therefore multiple messages) simultaneously.

An execution group can be debugged by only one user at a time. If you attach your debugger to an execution group, another user cannot attach a debugger to that same execution group until you have ended your debugging session.

To attach the debugger to an execution group:

Procedure

1. Set a Flow Debug port number. To configure the broker JVM with a debug port number, use one of the following methods:
   - In the Brokers view in the WebSphere Message Broker Toolkit, right-click the execution group with which you want to work, and click **Launch Debugger**. Click **Configure**, and enter a port number. Click **OK** to enable debugging on the selected port, and attach the debugger to the selected execution group.
   - In the WebSphere Message Broker Explorer, right-click the execution group with which you want to work, and click **Properties**. Enter a port number in the Flow Debug Port field on the Extended page, and click **OK** to modify the Flow Debug port number. You must right-click the execution group and click **Flow Debug Port** > **Enable**. to enable debugging on the selected port.
   - Set the Java debug port by running the **mqsichangeproperties** command (all on one line) in the Command Console:

     ```
     mqsichangeproperties broker_name -e execution_group_name
     -o ComIbmJVMManager -n jvmDebugPort -v port_number
     ```

     For example:

     ```
     mqsichangeproperties TEST -e default
     -o ComIbmJVMManager -n jvmDebugPort -v 3920
     ```

     When this command has completed, restart the broker. See: “Starting and stopping a broker” on page 921.

   The Java JIT (just-in-time) compiler is disabled if the jvmDebugPort parameter is set to an integer greater than zero. If you are not debugging a message flow, reset the jvmDebugPort parameter to zero to maximize performance.
2. If you used the **mqsichangeproperties** command or the WebSphere Message Broker Explorer to configure the Flow Debug Port, you must use the Brokers view in the WebSphere Message Broker Toolkit to attach the flow debugger to the execution group. Right-click the execution group with which you want to work, and click **Launch Debugger**, and click **OK** to attach the debugger to the selected execution group.
3. Open the message flow that you want to debug in the Message Flow editor by double-clicking its name in the Broker Development view.
4. Add a breakpoint to a connection that leads out of the input node to ensure that the message flow does not run to completion before you can begin to debug it.

   The breakpoint appears as  . For information about adding a breakpoint, see “Working with breakpoints in the flow debugger” on page 3166.
5. Switch to the Debug perspective.
6. Right-click the Message Broker Launch Configuration in the Debug view, and click **Edit Source Lookup**. You can use Edit Source Lookup Path to tell the debugger where to look for your source files for message flows, and related resources such as ESQL, message maps and Java during debugging.

7. Click **Add**, and select the type of source to add to the lookup path. The lookup path can be an Eclipse project name, an external folder, or a compressed (`.zip`) file. You can specify multiple locations, but the debugger always looks first in the message flow project that you specify in the Edit Source Lookup Path dialog.
8. Select the resources to include in the lookup path, and click **OK**.
9. Click **Add** to include more resources in the lookup path, click **Up**, or **Down** to modify the order of the resources.
10. Click **OK** to exit the Edit Source Lookup Path dialog, and save your changes.
11. When the next message comes into your flow and arrives at breakpoint you added after the input node, the flow pauses, the breakpoint icon is highlighted:  , and you can start debugging.
12. In the Debug view, double-click the message flow that you want to debug. The message flow opens in the Message Flow editor. You can now add more breakpoints, start stepping over the flow, and so on.

What to do next

Next:

Continue with one of the following tasks:

- Optional: "Debug: putting a test message on an input queue" and "Debug: getting a test message from an output queue" on page 3164. These tasks involve putting messages to, and taking messages from, WebSphere MQ queues and are therefore useful only if your message flow includes MQInput and MQOutput nodes.
- "Working with breakpoints in the flow debugger" on page 3166.

Related concepts:

"Flow debugger overview" on page 3158
Use the flow debugger in the WebSphere Message Broker Toolkit to track messages through your message flows.

Related tasks:

Chapter 11, "Packaging and deploying," on page 3209
Package resources that you create in the WebSphere Message Broker Toolkit, such as message flows, and deploy them to execution groups on brokers.

"Debugging a message flow" on page 3157
Use the tasks described in this section of the documentation to manage and work with the flow debugger.

"Debug: ending a session" on page 3191
Finish debugging by detaching the flow debugger from the execution group to which your message flows are deployed.

Related reference:

"Flow debugger shortcuts" on page 6719

"Flow debugger icons and symbols" on page 6720
The Debug perspective uses various debugger icons and symbols.

Debug: putting a test message on an input queue

You can put a message on an input queue to test a message flow that you are debugging.

Before you begin

Before you start

Complete the steps described in "Attaching the flow debugger to an execution group for debugging" on page 3160.

About this task

If your message flow includes MQInput and MQOutput nodes, you can test the flow by putting a message on the input queue of your first MQInput node.

You can use the command line interfaces or WebSphere MQ Explorer to put a message to a queue.

You can also use the Test Client as a repeatable alternative. To use the Test Client, complete the steps described in the following sections:

- "Using enqueue in the Test Client"
- "Adding data to your message"
- "Optional: Using a file of sample data" on page 3164

If the message is processed by the message flow and is put on an output queue, you can retrieve it from that queue. See: "Debug: getting a test message from an output queue" on page 3164.

Using enqueue in the Test Client:
About this task

To configure an enqueue event in the Test Client so that you can use it to send a test message:

Procedure

1. Switch to the Broker Application Development perspective.
2. Click **File** > **New** > **Other**. The **New** dialog opens.
3. Select **Message Broker Test Client** in the Message Brokers category and click **Next**. The wizard opens and displays its first panel.
4. Select the project in which you want to create the Test Client file.
5. Enter a name for the Test Client file and click **Finish**. The Test Client editor opens.
6. On the toolbar at the upper right of the Test Client editor, under Message Flow Test Events, click the **Put a message onto a queue** icon  .
7. Under Detailed Properties, enter the names of the queue manager and the queue for the input node for this flow. Queue manager names are case-sensitive; check that you enter the name correctly.

   If you are putting a message onto an input queue that is on a remote computer, ensure that the queue manager of the associated broker has a server-connection channel called SYSTEM.BKR.CONFIG.
8. If you are putting a message onto a remote queue, enter values to identify the host and port of the computer that is hosting the queue.
9. Click **File** > **Save** to save the file.

Adding data to your message:

About this task

If you want to add just a small amount of test data in your test message, type the data into the **Source** section of the **Message** pane:

Procedure

1. Open your Test Client file.
2. Type your test data directly into the **Source** section of the Message pane.
3. Put the test message by clicking **Send Message**.

Optional: Using a file of sample data:
About this task

If you want your test message to contain a larger quantity of sample data (for example some structured XML), you can import a file containing that data into the Test Client:

Procedure

1. In the **Events** tab of your Test Client file, click **Import Source**.
2. Select the file you want to use as the content for the test message, and click **Open**. The contents of the selected file is added to the **Source** pane.
3. Click **File** > **Save** when you have finished.
4. Put the test message by clicking the **Send Message** button.

Related concepts:

"Flow debugger overview" on page 3158
Use the flow debugger in the WebSphere Message Broker Toolkit to track messages through your message flows.

Related tasks:

"Debug: getting a test message from an output queue"
You can get a message from an output queue to test a message flow that you are debugging.

"Working with breakpoints in the flow debugger" on page 3166
When you have started a debugging session by attaching the debugger to an execution group, you can set breakpoints to control where the message flow will pause.

Related reference:

"Flow debugger shortcuts" on page 6719

"Flow debugger icons and symbols" on page 6720
The Debug perspective uses various debugger icons and symbols.

Related information:

WebSphere MQ Version 7 Information Center online

Debug: getting a test message from an output queue

You can get a message from an output queue to test a message flow that you are debugging.

Before you begin

Before you start

Completed the following tasks:

- Chapter 9, “Developing message flow applications,” on page 1019
- Chapter 11, “Packaging and deploying,” on page 3209
- “Attaching the flow debugger to an execution group for debugging” on page 3160
- “Debug: putting a test message on an input queue” on page 3162

About this task

If your message flow includes MQInput and MQOutput nodes, you can test the flow by putting a message on the input queue of your first MQInput node and retrieving it from an MQOutput node.

You can use the command line interfaces or WebSphere MQ Explorer to get a message from an output queue.

You can also use the Test Client as a repeatable alternative. To use the Test Client, complete the following steps:

Procedure

1. Switch to the Broker Application Development perspective.
2. Click **File** > **New** > **Other**. The **New** dialog opens.
3. Select **Message Broker Test Client** in the Message Brokers category and click **Next**. The wizard opens and displays its first panel.
4. Select the project in which you want to create the Test Client file.
5. Enter a name for the Test Client file and click **Finish**. The Test Client editor opens.
6. On the toolbar at the upper right of the Test Client editor, under **Message Flow Test Events**, click the **Get a message from a Queue** icon  .
7. Under **Detailed Properties**, enter the name of the queue manager and output node queue.
8. Click **Get Message** to read a message from the queue.

Results

When a message is available on an output queue, you can see it in the Test Client editor.

Related concepts:

“Flow debugger overview” on page 3158
Use the flow debugger in the WebSphere Message Broker Toolkit to track messages through your message flows.

Related tasks:

“Debug: putting a test message on an input queue” on page 3162
You can put a message on an input queue to test a message flow that you are debugging.

“Working with breakpoints in the flow debugger” on page 3166
When you have started a debugging session by attaching the debugger to an execution group, you can set breakpoints to control where the message flow will pause.

Related reference:

“Flow debugger shortcuts” on page 6719

"Flow debugger icons and symbols" on page 6720
The Debug perspective uses various debugger icons and symbols.

Working with breakpoints in the flow debugger

When you have started a debugging session by attaching the debugger to an execution group, you can set breakpoints to control where the message flow will pause.

About this task

Use the following tasks to manage breakpoints:

- "Adding breakpoints in the flow debugger"
- "Restricting breakpoints in the flow debugger to specific flow instances" on page 3168
- "Enabling and disabling breakpoints in the flow debugger" on page 3169
- "Removing breakpoints in the flow debugger" on page 3170

What to do next

Next:

After you have set one or more breakpoints in the message flow, continue your debugging session by stepping through the message flow, pausing at each active breakpoint. See: "Stepping through message flow instances in the debugger" on page 3172.

You can also examine message data, code, and mappings at appropriate points. See: "Debugging data" on page 3180.

Related concepts:

"Flow debugger overview" on page 3158
Use the flow debugger in the WebSphere Message Broker Toolkit to track messages through your message flows.

"Debug perspective" on page 6789

Related tasks:

"Attaching the flow debugger to an execution group for debugging" on page 3160
Before you can debug your message flow, you must attach the flow debugger to the execution group where your flow is deployed, then start a debugging session.

Related reference:

"Flow debugger shortcuts" on page 6719

"Flow debugger icons and symbols" on page 6720
The Debug perspective uses various debugger icons and symbols.

Adding breakpoints in the flow debugger

Add breakpoints to connections in your message flow to control where flow processing will pause.

Before you begin

Before you start:

Attach the flow debugger to the execution group where your flow is deployed. See: "Attaching the flow debugger to an execution group for debugging" on page 3160.

About this task

You can add breakpoints to the connections of a message flow that is open in the Message Flow editor. Each breakpoint that you add to a flow is also automatically added to all other instances of the flow and you do not need to restart any of the instances.

Every breakpoint is automatically enabled when you add it to a connection and the connection is flagged with the enabled breakpoint symbol  .

Manually set a breakpoint after the collector node or any other multithreaded node. When you use the Debug perspective on the node, you see that the thread has been ended.

To add breakpoints to the connections of a message flow:

Procedure

1. Switch to the Debug perspective.
2. Add breakpoints to the appropriate connections. Use any of the following methods:

Option	Method
Add breakpoints individually to selected connections.	1. In the Message Flow editor, right-click the connection where you want to set the breakpoint. 2. Click **Add Breakpoint**.
Add breakpoints simultaneously to all connections *entering* a selected node.	1. In the Message Flow editor, right-click the node before which you want to set breakpoints. 2. Click **Add Breakpoints Before Node**.
Add breakpoints simultaneously to all connections *leaving* a selected node.	1. In the Message Flow editor, right-click the node after which you want to set breakpoints. 2. Click **Add Breakpoints After Node**.

What to do next

Next:

After you have set one or more breakpoints in the message flow, step through the flow, pausing at each active breakpoint. See: “Stepping through message flow instances in the debugger” on page 3172.

You can also examine message data, code, and mappings at appropriate points. See: “Debugging data” on page 3180.

Related concepts:

“Flow debugger overview” on page 3158
Use the flow debugger in the WebSphere Message Broker Toolkit to track messages through your message flows.

Related tasks:

"Restricting breakpoints in the flow debugger to specific flow instances"
Breakpoints can be applied to particular flow instances, instead of all instances, which is the default behavior.

"Enabling and disabling breakpoints in the flow debugger" on page 3169
You can disable breakpoints that are currently enabled, and vice versa.

"Removing breakpoints in the flow debugger" on page 3170
Remove breakpoints that are no longer required from connections in your message flow.

"Stepping through message flow instances in the debugger" on page 3172
After you have added one or more breakpoints to a message flow in the debugger, you can step through the flow, pausing as required.

"Debugging data" on page 3180
You can view (and change) data in messages, ESQL code, Java code or mappings as debugging progresses.

Related reference:

"Flow debugger shortcuts" on page 6719

"Flow debugger icons and symbols" on page 6720
The Debug perspective uses various debugger icons and symbols.

Restricting breakpoints in the flow debugger to specific flow instances

Breakpoints can be applied to particular flow instances, instead of all instances, which is the default behavior.

Before you begin

Before you start:

Add one or more breakpoints to your message flow. See: "Adding breakpoints in the flow debugger" on page 3166.

About this task

When you add a breakpoint to a message flow in the Message Flow editor, the breakpoint automatically applies to all instances of the flow. However, you can restrict a breakpoint to one or more instances of a flow. This enables you to work more easily with just those instances that you are currently interested in, rather than with all instances.

To restrict a breakpoint to one or more flow instances:

Procedure

1. Switch to the Debug perspective.
2. In the Breakpoints view, right-click the breakpoint that you want to restrict, then click **Properties** to open the Flow Breakpoints Properties window.
3. In the **Restrict to Selected Flow Instance(s)** list box, select those instances to which you want to restrict the breakpoint.
   - You must have at least one instance active; if not, the **Restrict to Selected Flow Instance(s)** list box is empty.
   - If any instance is currently paused at the breakpoint, all check boxes in the **Restrict to Selected Flow Instance(s)** list box are disabled and you cannot select them.

4. Click **OK**.

What to do next

Next:

Now you can add additional breakpoints (if needed), step through the flow instance, and work with data:

- "Adding breakpoints in the flow debugger" on page 3166
- "Stepping through message flow instances in the debugger" on page 3172
- "Debugging data" on page 3180

Related concepts:

"Flow debugger overview" on page 3158
Use the flow debugger in the WebSphere Message Broker Toolkit to track messages through your message flows.

Related tasks:

"Adding breakpoints in the flow debugger" on page 3166
Add breakpoints to connections in your message flow to control where flow processing will pause.

"Stepping through message flow instances in the debugger" on page 3172
After you have added one or more breakpoints to a message flow in the debugger, you can step through the flow, pausing as required.

"Debugging data" on page 3180
You can view (and change) data in messages, ESQL code, Java code or mappings as debugging progresses.

Related reference:

"Flow debugger shortcuts" on page 6719

"Flow debugger icons and symbols" on page 6720
The Debug perspective uses various debugger icons and symbols.

Enabling and disabling breakpoints in the flow debugger

You can disable breakpoints that are currently enabled, and vice versa.

Before you begin

Before you start:

Add one or more breakpoints to your message flow. See: "Adding breakpoints in the flow debugger" on page 3166.

About this task

Message flow processing pauses only at breakpoints that are enabled. By controlling which breakpoints are enabled and which are disabled, you can, for example, allow processing to continue to the part of a flow that you are interested in without having to continually add and remove breakpoints.

The following symbols identify breakpoints:

 Enabled breakpoint

 Disabled breakpoint

If you disable all the breakpoints in a message flow, you cannot perform any other debugging tasks until you add a new breakpoint, or enable an existing breakpoint.

To change the state of breakpoints:

Procedure

1. Switch to the Debug perspective.
2. In the Breakpoints view, select one or more breakpoints that you want to enable or disable.
3. Right-click the selected breakpoints and click **Enable** or **Disable**.
4. Optional: to change the state of a single breakpoint, right-click the breakpoint and click **Properties**. Select or clear the **Enabled** check box as required, then click **OK**.

Results

The state of breakpoints is changed in all instances of the message flow where they are set.

What to do next

Next:

If you have finished debugging, continue with: "Debug: ending a session" on page 3191.

Related concepts:

"Flow debugger overview" on page 3158
Use the flow debugger in the WebSphere Message Broker Toolkit to track messages through your message flows.

Related tasks:

"Enabling and disabling breakpoints in the flow debugger" on page 3169
You can disable breakpoints that are currently enabled, and vice versa.

"Removing breakpoints in the flow debugger"
Remove breakpoints that are no longer required from connections in your message flow.

Related reference:

"Flow debugger shortcuts" on page 6719

"Flow debugger icons and symbols" on page 6720
The Debug perspective uses various debugger icons and symbols.

Removing breakpoints in the flow debugger

Remove breakpoints that are no longer required from connections in your message flow.

Before you begin

Before you start:

Add one or more breakpoints to your message flow. See: "Adding breakpoints in the flow debugger" on page 3166.

About this task

The following symbols identify breakpoints:

 Enabled breakpoint

 Disabled breakpoint

If you remove a breakpoint from a message flow, it is automatically removed from all instances of the message flow where it is set.

If you remove all the breakpoints that you have added to your message flow, you cannot perform any other debugging tasks until you add a new breakpoint.

To remove breakpoints:

Procedure

1. Switch to the Debug perspective.
2. Remove the breakpoints. Use one of the following methods, depending on how many breakpoints you want to remove:

Option	Method
Remove individual breakpoints.	1. In the Message Flow editor, right-click the breakpoint that you want to remove, then click **Remove Breakpoint**.
Remove several breakpoints simultaneously.	1. Click the **Flow Breakpoints** tab to show the Breakpoints view. 2. Select one or more breakpoints that you want to remove. 3. Click the **Remove Selected Breakpoints** icon  on the toolbar, or right-click the selected breakpoints, then click  Remove.
Remove all breakpoints simultaneously.	1. Click the **Breakpoints** tab to show the Breakpoints view. 2. Click the **Remove All Breakpoints** icon  on the toolbar, or right-click any breakpoint, then click  Remove All.

What to do next

Next:

If you have finished debugging, continue with: “Debug: ending a session” on page 3191.

Related concepts:

“Flow debugger overview” on page 3158
Use the flow debugger in the WebSphere Message Broker Toolkit to track messages through your message flows.

Related tasks:

"Enabling and disabling breakpoints in the flow debugger" on page 3169
You can disable breakpoints that are currently enabled, and vice versa.

"Stepping through message flow instances in the debugger"
After you have added one or more breakpoints to a message flow in the debugger, you can step through the flow, pausing as required.

"Debugging data" on page 3180
You can view (and change) data in messages, ESQL code, Java code or mappings as debugging progresses.

Related reference:

"Flow debugger shortcuts" on page 6719

"Flow debugger icons and symbols" on page 6720
The Debug perspective uses various debugger icons and symbols.

Stepping through message flow instances in the debugger

After you have added one or more breakpoints to a message flow in the debugger, you can step through the flow, pausing as required.

Before you begin

Before you start:

Add one or more breakpoints to your message flow. See: "Adding breakpoints in the flow debugger" on page 3166.

About this task

The message flow debugger pauses flow processing at the first breakpoint it encounters. You can then continue with one or more of the following tasks, as appropriate:

- "Debug: resuming message flow processing" on page 3173
- "Debug: running to completion" on page 3174
- "Debug: stepping over nodes" on page 3175
- "Debug: stepping into subflows" on page 3176
- "Debug: stepping out of subflows" on page 3177
- "Debug: stepping through source code" on page 3178

What to do next

Next:

As you step through the message flow you can look at the processing data. When you have finished, end your debugging session:

- "Debugging data" on page 3180
- "Debug: ending a session" on page 3191

Related concepts:

"Flow debugger overview" on page 3158
Use the flow debugger in the WebSphere Message Broker Toolkit to track messages through your message flows.

"Debug perspective" on page 6789

Related tasks:

"Debugging data" on page 3180
You can view (and change) data in messages, ESQL code, Java code or mappings as debugging progresses.

"Working with breakpoints in the flow debugger" on page 3166
When you have started a debugging session by attaching the debugger to an execution group, you can set breakpoints to control where the message flow will pause.

"Managing flows and flow instances during debugging" on page 3187
During a debugging session, there are various administrative tasks that you might need to do, which include detaching the debugger from the execution group when you have finished.

Related reference:

"Flow debugger shortcuts" on page 6719

"Flow debugger icons and symbols" on page 6720
The Debug perspective uses various debugger icons and symbols.

Debug: resuming message flow processing

Each time message flow processing pauses at an active breakpoint, you can investigate the state of the flow, then resume processing.

Before you begin

Before you start:

Add one or more breakpoints to your message flow. See: "Adding breakpoints in the flow debugger" on page 3166.

About this task

When message flow processing has paused at a breakpoint, you can resume processing:

Procedure

1. Switch to the Debug perspective.
2. In the Debug view, take one of the following steps:
   - On the toolbar click **Resume Flow Execution**  .
   - Right-click the flow stack frame, then click **Resume**  .

Results

Message flow processing continues until the next breakpoint that is set in the logical processing of the current message. If there is no further enabled breakpoint at which the flow instance can pause, processing runs to completion and the flow instance is removed from the Debug view.

What to do next

Next:

If you have completed debugging this message flow, you can remove the breakpoints, or end the debugging session:

- "Removing breakpoints in the flow debugger" on page 3170

- "Debug: ending a session" on page 3191

Related concepts:

"Flow debugger overview" on page 3158
Use the flow debugger in the WebSphere Message Broker Toolkit to track messages through your message flows.

"Debug perspective" on page 6789

Related tasks:

"Debug: running to completion"
When message flow processing pauses at an active breakpoint, you can let it continue to the end of the flow, ignoring other breakpoints.

"Debug: stopping a message flow instance" on page 3188
While debugging, a message flow cannot be redeployed until it has been stopped.

Related reference:

"Flow debugger shortcuts" on page 6719

"Flow debugger icons and symbols" on page 6720
The Debug perspective uses various debugger icons and symbols.

Debug: running to completion

When message flow processing pauses at an active breakpoint, you can let it continue to the end of the flow, ignoring other breakpoints.

Before you begin

Before you start:

Add one or more breakpoints to your message flow. See: "Adding breakpoints in the flow debugger" on page 3166.

About this task

When message flow processing has paused at a breakpoint, you can restart processing so that the message flow runs to completion.

If you want the flow to continue processing, but you want to pause at the next enabled breakpoint instead of running to completion, see: "Debug: resuming message flow processing" on page 3173.

Procedure

1. Switch to the Debug perspective.
2. In the Debug view:
   - either, click **Run to completion** on the toolbar.
   - or, right-click the flow stack frame, then click **Run to completion** .

Results

The flow instance ignores all breakpoints and processing continues to the end. The flow instance is automatically removed from the Debug view.

What to do next

Next:

If you have finished debugging, you can remove the breakpoints, or end the debugging session:

- "Removing breakpoints in the flow debugger" on page 3170
- "Debug: ending a session" on page 3191

Related concepts:

"Flow debugger overview" on page 3158
Use the flow debugger in the WebSphere Message Broker Toolkit to track messages through your message flows.

"Debug perspective" on page 6789

Related tasks:

"Debug: resuming message flow processing" on page 3173
Each time message flow processing pauses at an active breakpoint, you can investigate the state of the flow, then resume processing.

"Working with breakpoints in the flow debugger" on page 3166
When you have started a debugging session by attaching the debugger to an execution group, you can set breakpoints to control where the message flow will pause.

"Debug: ending a session" on page 3191
Finish debugging by detaching the flow debugger from the execution group to which your message flows are deployed.

Related reference:

"Flow debugger shortcuts" on page 6719

"Flow debugger icons and symbols" on page 6720
The Debug perspective uses various debugger icons and symbols.

Debug: stepping over nodes

When message flow processing pauses at an active breakpoint, you can step over the node and continue processing until the next active breakpoint, ignoring breakpoints that might have been set in code within the node.

Before you begin

Before you start:

Add one or more breakpoints to your message flow. See: "Adding breakpoints in the flow debugger" on page 3166.

About this task

To step over the next node and continue message flow processing:

Procedure

1. Switch to the Debug perspective.
2. In the Debug view:
   - either, click **Step Over Node**  on the toolbar.
   - or, right-click the flow stack frame, then click **Step Over Node**  .

Results

Message flow processing continues until the next breakpoint that is set in the logical processing of the current message. If there is no further enabled breakpoint

at which the flow instance can pause, processing runs to completion and the flow instance is removed from the Debug view.

What to do next

Next:

If you have finished debugging, you can remove the breakpoints, or end the debugging session:

- "Removing breakpoints in the flow debugger" on page 3170
- "Debug: ending a session" on page 3191

Related concepts:

"Flow debugger overview" on page 3158
Use the flow debugger in the WebSphere Message Broker Toolkit to track messages through your message flows.

"Debug perspective" on page 6789

Related tasks:

"Debug: resuming message flow processing" on page 3173
Each time message flow processing pauses at an active breakpoint, you can investigate the state of the flow, then resume processing.

"Debug: running to completion" on page 3174
When message flow processing pauses at an active breakpoint, you can let it continue to the end of the flow, ignoring other breakpoints.

"Debug: stepping into subflows"
When message flow processing pauses at a breakpoint, you can step into the subflow that follows.

Related reference:

"Flow debugger shortcuts" on page 6719

"Flow debugger icons and symbols" on page 6720
The Debug perspective uses various debugger icons and symbols.

Debug: stepping into subflows

When message flow processing pauses at a breakpoint, you can step into the subflow that follows.

Before you begin

Before you start:

Add one or more breakpoints to your message flow. See: "Adding breakpoints in the flow debugger" on page 3166.

About this task

To step into a subflow:

Procedure

1. Switch to the Debug perspective.
2. In the Debug view:
   - either, click **Step Into Subflow** on the toolbar.
   - or, right-click the flow stack frame, then click **Step Into Subflow** .

Results

The subflow opens in the Message Flow editor and displaces the parent message flow. Message flow processing continues until the next breakpoint that is set in the logical processing of the current message. If there is no further enabled breakpoint at which the flow instance can pause, processing runs to completion and the flow instance is removed from the Debug view.

What to do next

Next:

If you have finished debugging, you can remove the breakpoints, or end the debugging session:

- "Removing breakpoints in the flow debugger" on page 3170
- "Debug: ending a session" on page 3191

Related concepts:

"Flow debugger overview" on page 3158
Use the flow debugger in the WebSphere Message Broker Toolkit to track messages through your message flows.

"Debug perspective" on page 6789

Related tasks:

"Debug: running to completion" on page 3174
When message flow processing pauses at an active breakpoint, you can let it continue to the end of the flow, ignoring other breakpoints.

"Debug: stepping out of subflows"
When message flow processing has paused at a breakpoint in a subflow, you can step out of the subflow.

Related reference:

"Flow debugger shortcuts" on page 6719

"Flow debugger icons and symbols" on page 6720
The Debug perspective uses various debugger icons and symbols.

Debug: stepping out of subflows

When message flow processing has paused at a breakpoint in a subflow, you can step out of the subflow.

Before you begin

Before you start

Complet the following tasks:

- "Adding breakpoints in the flow debugger" on page 3166
- "Debug: stepping into subflows" on page 3176

About this task

To step out of a subflow:

Procedure

1. Switch to the Debug perspective.
2. In the Debug view:

- either, click **Step Out of Subflow** on the toolbar.
- or, right-click the flow stack frame, then click **Step Out of Subflow**.

Results

The debugger continues processing until it reaches the connection from the output terminal of the subflow, where it pauses. The parent flow opens in the Message Flow editor, displacing the subflow.

What to do next

Next:

If you have finished debugging, you can remove the breakpoints, or end the debugging session:

- "Removing breakpoints in the flow debugger" on page 3170
- "Debug: ending a session" on page 3191

Related concepts:

"Flow debugger overview" on page 3158
Use the flow debugger in the WebSphere Message Broker Toolkit to track messages through your message flows.

"Debug perspective" on page 6789

Related tasks:

"Debug: stepping into subflows" on page 3176
When message flow processing pauses at a breakpoint, you can step into the subflow that follows.

"Debug: running to completion" on page 3174
When message flow processing pauses at an active breakpoint, you can let it continue to the end of the flow, ignoring other breakpoints.

Related reference:

"Flow debugger shortcuts" on page 6719

"Flow debugger icons and symbols" on page 6720
The Debug perspective uses various debugger icons and symbols.

Debug: stepping through source code

When message flow processing has paused at a breakpoint on entry to a node that contains ESQL code, Java code, or mappings, you can step through the code.

Before you begin

Before you start:

Add one or more breakpoints to your message flow. See "Adding breakpoints in the flow debugger" on page 3166.

About this task

The nodes that can contain ESQL code, Java code, or mappings are listed in: "Flow debugger overview" on page 3158. Add breakpoints as appropriate:

- ESQL code: add a breakpoint in the ESQL code.
- Java code: add a breakpoint in the Java code.

- Mappings: add a breakpoint to a map using the Map Script panel. Note that mapping routines are implemented in ESQL; you might choose to step through the ESQL code rather than the mappings.

To step through your source code:

Procedure

1. Switch to the Debug perspective.
2. Step into the source code. In the Debug view:
   - either, click **Step into Source Code** on the toolbar.
   - or, right-click the flow stack frame, then click **Step Into** .
3. When message flow processing has paused at a breakpoint within ESQL code, Java code, or mappings, you can step through the source code, line by line. Repeat this step as often as necessary. In the Debug view:
   - either, click **Step Over** on the toolbar.
   - or, right-click the flow stack frame, then click **Step Over** .

   A single line of source code runs and the flow pauses at the next line of code. What you can do depends on what type of code is contained within the node. See:
   - "Debugging ESQL" on page 3182
   - "Debugging Java" on page 3183
   - "Debugging mappings" on page 3185

   If the debugger is paused before the last line of code when you step over, the last line of code runs and message flow processing continues until the next breakpoint in the logical processing of the current message. If there is no further enabled breakpoint at which the flow instance can pause, processing runs to completion and the flow instance is removed from the Debug view.
4. If you have finished looking at the code or mappings before the last breakpoint, you can continue processing the message flow. In the Debug view:
   - either, click **Step Return** on the toolbar.
   - or, right-click the flow stack frame, then click **Step Return** .

   The source code runs to completion from the current breakpoint and message flow processing continues until the next breakpoint that is set in the logical processing of the current message. If there is no further enabled breakpoint at which the flow instance can pause, processing runs to completion and the flow instance is removed from the Debug view.

What to do next

Next:

If you have completed debugging this message flow, you can remove the breakpoints or end the debugging session:

- "Removing breakpoints in the flow debugger" on page 3170
- "Debug: ending a session" on page 3191

Related concepts:

"Flow debugger overview" on page 3158
Use the flow debugger in the WebSphere Message Broker Toolkit to track messages through your message flows.

"Debug perspective" on page 6789

Related tasks:

"Debug: running to completion" on page 3174
When message flow processing pauses at an active breakpoint, you can let it continue to the end of the flow, ignoring other breakpoints.

Related reference:

"Flow debugger shortcuts" on page 6719

"Flow debugger icons and symbols" on page 6720
The Debug perspective uses various debugger icons and symbols.

Debugging data

You can view (and change) data in messages, ESQL code, Java code or mappings as debugging progresses.

About this task

When you have added one or more breakpoints to a deployed message flow, the debugger stops the message flow processing at each breakpoint. Depending on the context of the breakpoint, you can do one of the following tasks:

- "Debugging messages" on page 3181
- "Debugging ESQL" on page 3182
- "Debugging Java" on page 3183
- "Debugging mappings" on page 3185

Results

When you have finished debugging a message flow, you can remove the breakpoints, or end the debugging session:

- "Removing breakpoints in the flow debugger" on page 3170
- "Debug: ending a session" on page 3191

Related concepts:

"Flow debugger overview" on page 3158
Use the flow debugger in the WebSphere Message Broker Toolkit to track messages through your message flows.

"Debug perspective" on page 6789

Related tasks:

"Working with breakpoints in the flow debugger" on page 3166
When you have started a debugging session by attaching the debugger to an execution group, you can set breakpoints to control where the message flow will pause.

"Stepping through message flow instances in the debugger" on page 3172
After you have added one or more breakpoints to a message flow in the debugger, you can step through the flow, pausing as required.

Related reference:

"Flow debugger shortcuts" on page 6719

"Flow debugger icons and symbols" on page 6720
The Debug perspective uses various debugger icons and symbols.

Debugging messages

When message flow processing has paused at a breakpoint in your message flow, you can examine and modify the message content.

Before you begin

Before you start

Add one or more breakpoints to your message flow. See: "Adding breakpoints in the flow debugger" on page 3166.

About this task

To examine and modify message data:

Procedure

1. Switch to the Debug perspective.
2. View the messages in the Variables view.

   The Breakpoints view and the Variables view share the same pane. Click the tab at the bottom to select the view that you want.
3. To alter a message, right-click it and select an option from the menu. You cannot alter the content of exceptions within a message.

Results

Message flow processing continues until the next breakpoint that is set in the logical processing of the current message. If there is no further enabled breakpoint at which the flow instance can pause, processing runs to completion and the flow instance is removed from the Debug view.

What to do next

Next:

If you have finished debugging this message flow, you can remove the breakpoints, or end the debugging session:

- "Removing breakpoints in the flow debugger" on page 3170
- "Debug: ending a session" on page 3191

Related concepts:

"Flow debugger overview" on page 3158
Use the flow debugger in the WebSphere Message Broker Toolkit to track messages through your message flows.

"Debug perspective" on page 6789

Related tasks:

"Working with breakpoints in the flow debugger" on page 3166
When you have started a debugging session by attaching the debugger to an execution group, you can set breakpoints to control where the message flow will pause.

"Stepping through message flow instances in the debugger" on page 3172
After you have added one or more breakpoints to a message flow in the debugger, you can step through the flow, pausing as required.

Related reference:

"Flow debugger shortcuts" on page 6719

"Flow debugger icons and symbols" on page 6720
The Debug perspective uses various debugger icons and symbols.

Debugging ESQL

When message flow processing has paused at a breakpoint that you have set in source code within a node that contains ESQL code, you can examine and modify the ESQL variables in the Flow Debugger.

Before you begin

Before you start

Complete the following tasks:

- "Adding breakpoints in the flow debugger" on page 3166
- "Debug: stepping through source code" on page 3178

About this task

You can browse ESQL variables in the Variables view in the Debug Perspective, and change their associated data values. You can also set breakpoints on lines in the ESQL code. See the following sections for further details:

- "Using breakpoints on ESQL code lines"
- "Working with ESQL variables"

Using breakpoints on ESQL code lines:
Procedure

1. Switch to the Debug perspective.
2. Open the ESQL editor.
3. Right-click a line where you want to set a breakpoint.

   You cannot set a breakpoint on a comment line or a blank line.
4. Select from the menu to create, delete, or restrict the breakpoint, in a similar way to normal debugger breakpoints, as described in "Working with breakpoints in the flow debugger" on page 3166.

Working with ESQL variables:
Procedure

1. Switch to the Debug perspective.
2. Open the Variables view. Variables are shown in a tree, using the symbol  .
3. To work with a variable, right-click it and select an option from the pop-up menu.

   You cannot update message trees, or REFERENCE variables.

   For example, if you have declared the following ESQL variables, you can change their values in the debugger:

   ```
   DECLARE myInt INT 0;
   DECLARE myFloat FLOAT 0.0e-1;
   DECLARE myDecimal DECIMAL 0.1;
   DECLARE myInterval INTERVAL DAY TO MONTH;
   ```

Results

Message flow processing continues until the next breakpoint that is set in the logical processing of the current message. If there is no further enabled breakpoint at which the flow instance can pause, processing runs to completion and the flow instance is removed from the Debug view.

What to do next

Next:

If you have finished debugging this message flow, you can remove the breakpoints, or end the debugging session:

- "Removing breakpoints in the flow debugger" on page 3170
- "Debug: ending a session" on page 3191

Related concepts:

"Flow debugger overview" on page 3158
Use the flow debugger in the WebSphere Message Broker Toolkit to track messages through your message flows.

"Debug perspective" on page 6789

Related tasks:

"Working with breakpoints in the flow debugger" on page 3166
When you have started a debugging session by attaching the debugger to an execution group, you can set breakpoints to control where the message flow will pause.

"Debug: stepping through source code" on page 3178
When message flow processing has paused at a breakpoint on entry to a node that contains ESQL code, Java code, or mappings, you can step through the code.

Related reference:

"Flow debugger shortcuts" on page 6719

"Flow debugger icons and symbols" on page 6720
The Debug perspective uses various debugger icons and symbols.

Debugging Java

When message flow processing has paused at a breakpoint that you have set in source code within a node that contains Java code, you can examine and modify the Java variables in the Flow Debugger.

About this task

If you notice that performance has degraded after following these steps, see "Resolving problems with performance" on page 3504.

Procedure

1. To open the Command Console, click **Start** > **Programs** > **IBM WebSphere Message Broker 7.0** > **Command Console**.
2. Start the broker by running the **mqsistart** command in the Command Console.
3. Set the Java debug port by running the **mqsichangeproperties** command (all on one line) in the Command Console:

   ```
   mqsichangeproperties broker_name -e execution_group_name
   -o ComIbmJVMManager -n jvmDebugPort -v port_number
   ```

For example:

```
mqsichangeproperties TEST -e default
-o ComIbmJVMManager -n jvmDebugPort -v 3920
```

4. Stop and restart the broker by running the **mqsistop** and **mqsistart** commands.
5. Open the message flow that you want to debug in the Message Flow editor by double-clicking its name in the Broker Development view.
6. Add a breakpoint where the Java method is called, by following the instructions in "Adding breakpoints in the flow debugger" on page 3166.
7. To step directly into the Java code during the debugging process, add a breakpoint in the Java code.
8. Deploy the broker archive (BAR) file that includes the JAR file that contains the Java code, by following the instructions in "Deploying a broker archive file" on page 3235.
9. Click **Run** > **Debug** to open the Debug wizard.
10. Right-click **Message Broker Debug** in the list of elements on the left and click **New**.
11. Set the **Java Debug Port** with the same value that you specified for the **-v** parameter on the **mqsichangeproperties** command, and click **Apply** to save your changes.
12. Click the **Source** tab, specify the source file location, and click **Apply** to save your changes.
13. Click **Debug** to start the debug process.

Working with Java variables:
About this task

When message flow processing has paused at a breakpoint in the source code within a node that contains Java code (a user-defined node or a JavaCompute node), you can browse Java variables in the Variables view on the Debug perspective, and change their associated data values.

Procedure

1. Switch to the Debug perspective.
2. Click the **Variables** tab to open the Variables view if it is not already open.

   Variables are shown in a tree, using the symbol  .
3. To work with a variable, right-click it and select an option from the menu.

Results

Message flow processing continues until the next breakpoint that is set in the logical processing of the current message. If there is no further enabled breakpoint at which the flow instance can pause, processing runs to completion and the flow instance is removed from the Debug view.

What to do next

Next:

When you have completed debugging the message flow, you can remove the breakpoints or end the debugging session:

- "Removing breakpoints in the flow debugger" on page 3170

- "Debug: ending a session" on page 3191

Related concepts:

"Flow debugger overview" on page 3158
Use the flow debugger in the WebSphere Message Broker Toolkit to track messages through your message flows.

"Debug perspective" on page 6789

Related tasks:

"Adding breakpoints in the flow debugger" on page 3166
Add breakpoints to connections in your message flow to control where flow processing will pause.

"Working with breakpoints in the flow debugger" on page 3166
When you have started a debugging session by attaching the debugger to an execution group, you can set breakpoints to control where the message flow will pause.

"Debug: stepping through source code" on page 3178
When message flow processing has paused at a breakpoint on entry to a node that contains ESQL code, Java code, or mappings, you can step through the code.

Related reference:

"Flow debugger shortcuts" on page 6719

"Flow debugger icons and symbols" on page 6720
The Debug perspective uses various debugger icons and symbols.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsistart** command" on page 3965
Use the **mqsistart** command to start the specified broker if all initial verification tests complete successfully.

Debugging mappings

When message flow processing has paused at a breakpoint that you have set in source code within a node that contains mappings, you can view the mapping routines and modify user-defined variables in the Flow Debugger.

Before you begin

Before you start

To complete this task, you must have completed the following tasks:

- "Adding breakpoints in the flow debugger" on page 3166
- "Debug: stepping through source code" on page 3178

About this task

Mapping routines are implemented in ESQL. If you step into the code, you can either step through the ESQL code, or step through the mappings.

Procedure

1. Switch to the Debug perspective.
2. In the Debug view, take one of the following steps:
   - Click **Step into Source Code**  on the toolbar.

- Right-click the flow stack frame, then click **Step into**  .

The Message Mapping editor opens with the mapping routine highlighted in both the Mapping editor and the Outline view.

3. To use breakpoints on mapping lines:
   a. In the Message Mapping Editor, select the line for the mapping command that you want to use, right-click the space beside it and select from the menu to add or disable a breakpoint. (Alternatively, double-click the same space to add or remove a breakpoint.)
   b. Select from the menu to create, delete, or restrict the breakpoint, in a similar way to normal debugger breakpoints, as described in: "Working with breakpoints in the flow debugger" on page 3166.

   You cannot set a breakpoint on a comment line or a blank line.
4. Check the mapping routines by stepping through the mappings.

   In the Debug view, the stack frame shows the list of mapping commands and the current command. The Variables view shows your user-defined mapping variables and the current message. You can change the values of user-defined variables.

Results

Message flow processing continues until the next breakpoint that is set in the logical processing of the current message. If there is no further enabled breakpoint at which the flow instance can pause, processing runs to completion and the flow instance is removed from the Debug view.

What to do next

Next:

If you have finished debugging, you can remove the breakpoints, or end the debugging session:

- "Removing breakpoints in the flow debugger" on page 3170
- "Debug: ending a session" on page 3191

Related concepts:

"Flow debugger overview" on page 3158
Use the flow debugger in the WebSphere Message Broker Toolkit to track messages through your message flows.

"Debug perspective" on page 6789

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

Related tasks:

"Working with breakpoints in the flow debugger" on page 3166
When you have started a debugging session by attaching the debugger to an execution group, you can set breakpoints to control where the message flow will pause.

"Debug: stepping through source code" on page 3178
When message flow processing has paused at a breakpoint on entry to a node that contains ESQL code, Java code, or mappings, you can step through the code.

Related reference:

"Flow debugger shortcuts" on page 6719

"Flow debugger icons and symbols" on page 6720
The Debug perspective uses various debugger icons and symbols.

Managing flows and flow instances during debugging

During a debugging session, there are various administrative tasks that you might need to do, which include detaching the debugger from the execution group when you have finished.

About this task

When you have started a session for message flow debugging, you might want to complete one or more of the following associated tasks:

- "Debug: querying a broker to find deployed flows"
- "Debug: stopping a message flow instance" on page 3188
- "Debug: redeploying a message flow" on page 3189
- "Debug: ending a session" on page 3191

Related concepts:

"Flow debugger overview" on page 3158
Use the flow debugger in the WebSphere Message Broker Toolkit to track messages through your message flows.

"Debug perspective" on page 6789

Related tasks:

"Working with breakpoints in the flow debugger" on page 3166
When you have started a debugging session by attaching the debugger to an execution group, you can set breakpoints to control where the message flow will pause.

"Stepping through message flow instances in the debugger" on page 3172
After you have added one or more breakpoints to a message flow in the debugger, you can step through the flow, pausing as required.

"Debugging data" on page 3180
You can view (and change) data in messages, ESQL code, Java code or mappings as debugging progresses.

Related reference:

"Flow debugger shortcuts" on page 6719

"Flow debugger icons and symbols" on page 6720
The Debug perspective uses various debugger icons and symbols.

Debug: querying a broker to find deployed flows

You can find the message flow that you want to work with in the Flow Debugger by refreshing the list of available flows.

About this task

During an active debugging session, you can query an execution group on a broker to find out what flows are currently deployed to it. The displayed list of message flows that are available in that execution group is updated. The updated list might include message flows that were not previously deployed, or that were not accessible because the flow was already being accessed by another developer.

To query an execution group for deployed flows:

Procedure

1. Switch to the Debug perspective.
2. In the Debug view, select the execution group that you want to query, then:
   - either, click **Refresh Selected Flow Engine to Get More Flow Types**  on the toolbar.
   - or, right-click the execution group, then click **Refresh**  .

Results

The Debug view is refreshed with the names of the flows that are currently deployed to the execution group and are available.

What to do next

Next:

You can continue your debugging session and debug one of the listed message flows, or end your debugging session:

- "Working with breakpoints in the flow debugger" on page 3166
- "Debug: ending a session" on page 3191

Related concepts:

"Flow debugger overview" on page 3158
Use the flow debugger in the WebSphere Message Broker Toolkit to track messages through your message flows.

"Debug perspective" on page 6789

Related tasks:

"Managing flows and flow instances during debugging" on page 3187
During a debugging session, there are various administrative tasks that you might need to do, which include detaching the debugger from the execution group when you have finished.

"Stepping through message flow instances in the debugger" on page 3172
After you have added one or more breakpoints to a message flow in the debugger, you can step through the flow, pausing as required.

Related reference:

"Flow debugger shortcuts" on page 6719

"Flow debugger icons and symbols" on page 6720
The Debug perspective uses various debugger icons and symbols.

Debug: stopping a message flow instance

While debugging, a message flow cannot be redeployed until it has been stopped.

About this task

While you are debugging, you might need to stop a message flow instance. For example, you might want to correct an error in your flow or source code. To do this, you must stop the flow and then redeploy it. See "Debug: redeploying a message flow" on page 3189.

To stop message flow processing, run it to completion:

Procedure

1. Switch to the Debug perspective.
2. In the Debug view:
   - either, click **Run to completion** on the toolbar.
   - or, right-click the flow stack frame, then click **Run to completion**  .

Results

The flow instance ignores all breakpoints and processing continues to the end. The flow instance is automatically removed from the Debug view.

What to do next

Next:

After stopping a flow instance, you can start to debug another message flow, or end your debugging session:

- "Attaching the flow debugger to an execution group for debugging" on page 3160
- "Debug: ending a session" on page 3191

Related concepts:

"Flow debugger overview" on page 3158
Use the flow debugger in the WebSphere Message Broker Toolkit to track messages through your message flows.

"Debug perspective" on page 6789

Related tasks:

"Debug: running to completion" on page 3174
When message flow processing pauses at an active breakpoint, you can let it continue to the end of the flow, ignoring other breakpoints.

"Stepping through message flow instances in the debugger" on page 3172
After you have added one or more breakpoints to a message flow in the debugger, you can step through the flow, pausing as required.

Related reference:

"Flow debugger shortcuts" on page 6719

"Flow debugger icons and symbols" on page 6720
The Debug perspective uses various debugger icons and symbols.

Debug: redeploying a message flow

If you want to change your message flow while you are debugging it, you must redeploy it to the execution group, then reattach the flow debugger.

Before you begin

Before you start

Stop the message flow before you redeploy it. See: "Debug: stopping a message flow instance" on page 3188

About this task

During your debugging session, you might find a problem in a message flow that you want to correct or see a behavior that you want to change. You can alter the flow to resolve the situation and redeploy the flow to the broker:

Procedure

1. Switch to the Debug perspective.
2. Detach the debugger from the execution group by clicking **Detach from the Selected Flow Engines**  on the toolbar.
3. Switch to the Broker Application Development perspective.
4. Edit the flow in the Message Flow editor and save your changes.
5. Double-click the broker archive (BAR) file that contains your flow. Remove the flow, then add your edited version and save your changes.

   See “Adding files to a broker archive” on page 3223.
6. Deploy your BAR file.

   Drag your BAR file from the Broker Development view to the execution group in the Brokers view. Check the Administration log to make sure that the deployment was successful.

   See: “Deploying a broker archive file” on page 3235.
7. Switch to the Debug perspective.
8. Reattach the debugger to the execution group.

   Click the down-arrow on the **Debug** icon  on the toolbar, and select **Debug** to invoke the **Debug (Create, manage, and run configurations)** wizard, and attach the flow engine again, following the instructions in “Attaching the flow debugger to an execution group for debugging” on page 3160.

Results

The modified message flow is now deployed to the broker, and the debugging session is ready for you to debug the new flow logic.

What to do next

Next: Continue to use these tasks to debug your message flow:

- “Working with breakpoints in the flow debugger” on page 3166
- “Stepping through message flow instances in the debugger” on page 3172

Related concepts:

“Flow debugger overview” on page 3158
Use the flow debugger in the WebSphere Message Broker Toolkit to track messages through your message flows.

“Debug perspective” on page 6789

Related tasks:

“Adding files to a broker archive” on page 3223
To deploy files to an execution group, include them in a broker archive (BAR) file.

“Attaching the flow debugger to an execution group for debugging” on page 3160
Before you can debug your message flow, you must attach the flow debugger to the execution group where your flow is deployed, then start a debugging session.

"Debug: ending a session"
Finish debugging by detaching the flow debugger from the execution group to which your message flows are deployed.

Related reference:

"Flow debugger shortcuts" on page 6719

"Flow debugger icons and symbols" on page 6720
The Debug perspective uses various debugger icons and symbols.

Debug: ending a session

Finish debugging by detaching the flow debugger from the execution group to which your message flows are deployed.

About this task

When you have finished debugging a flow, detach the flow debugger from the execution group. Other developers are then able to attach the debugger to the execution group. Detaching the flow debugger also restores the performance of your workbench environment, which might have been reduced by having the debugger attached.

To detach the flow debugger from an execution group:

Procedure

1. Switch to the Debug perspective.
2. In the Debug view, select the name of the execution group from which you want to detach the flow debugger, then take one of the following steps:
   - On the toolbar, click **Detach from the Selected Flow Engines**  .
   - Right-click the execution group, then click  **Detach**).

Results

All existing flow instances are automatically run to completion and the flow debugger is detached from the execution group. Your debugging session is now finished. You can start a new debugging session at any time.

Related concepts:

"Flow debugger overview" on page 3158
Use the flow debugger in the WebSphere Message Broker Toolkit to track messages through your message flows.

"Debug perspective" on page 6789

Related tasks:

"Attaching the flow debugger to an execution group for debugging" on page 3160
Before you can debug your message flow, you must attach the flow debugger to the execution group where your flow is deployed, then start a debugging session.

"Stepping through message flow instances in the debugger" on page 3172
After you have added one or more breakpoints to a message flow in the debugger, you can step through the flow, pausing as required.

Related reference:

"Flow debugger shortcuts" on page 6719

"Flow debugger icons and symbols" on page 6720
The Debug perspective uses various debugger icons and symbols.

Debugging message flows that contain WebSphere Adapters nodes

You can use various methods to monitor message flows that include WebSphere Adapters nodes.

About this task

Before you use any of the methods listed in this section, ensure that the appropriate JAR files and shared libraries are available to the WebSphere Adapters nodes. For more information, see "Preparing the environment for WebSphere Adapters nodes" on page 717.

Also, check for the latest information about WebSphere Adapters; see WebSphere Adapters technotes.

- **User and service trace:** You can use user and service trace to trace a message flow that contains WebSphere Adapters nodes. For more information, see "Using trace" on page 3533.
- **Flow debugger:** Use the flow debugger in the normal way to debug a message flow that contains WebSphere Adapters nodes. For more information, see "Debugging a message flow" on page 3157.
- **Adapter event table:** The WebSphere Adapters nodes use an event table to communicate the outcome of operations asynchronously to a calling application. For more information, see "Creating a custom event project in PeopleTools" on page 2083.

Handling exceptions that are raised by a WebSphere Adapters request node

The WebSphere Adapters request nodes raise exceptions that indicate the following Enterprise Information System (EIS) failures.

Message number	Exception type	Explanation
BIP3511	`RecordNotFound`	The requested record could not be found in the EIS.
BIP3512	`DuplicateRecord`	An attempt was made to create a record that already exists in the EIS.
BIP3513	`MultipleMatchingRecords`	A retrieve request matched more than one record. To retrieve multiple records, perform a `retrieveall` operation.
BIP3515	`MatchesExceededLimit`	A `retrieveall` exception returned more entries than the maximum allowed number.
BIP3516	`MissingData`	The message tree that was sent to the adapter request node does not have all the required fields set.

If an exception occurs that does not fit into the categories in the table, the node raises a general BIP3450 message that describes the problem.

You can use these exceptions to perform special processing when you do not want the exceptions to be treated as errors. For example:

- If a `create` operation fails because the record already exists, you could modify the request to an `update`.

- If a `retrieve` operation fails because the request matches more than one record, you could try a `retrieveall` operation instead.
- If a `retrieve` operation fails because the record could not be found, an empty record could be returned.

To handle these exceptions, you can connect a message routing node, Compute node, or JavaCompute node to the Failure terminal of the WebSphere Adapters request node, and route the exception to other processing nodes based on the exception message number.

XSD Schema Validation problem

When you configure an SAP adapter in the WebSphere Message Broker Toolkit, you might see the following warning, referring to an unresolvable IBM XML schema:

```
CTDX1101W : XSD: The location '' has not been resolved
example1.xsd /EAI_ESB_LIB line 2
example2.xsd /EAI_ESB_LIB line 2
XSD Schema Validation Problem
```

This warning is caused by the following namespace reference in an `<xsd:import>` element:

```
<xsd:import namespace="http://www.ibm.com/xmlns/prod/websphere/j2ca/sap/metadata"/>
```

No types or elements from this namespace are referenced in the logical structure of the XSD. An XML schema that references this namespace is in the referenced connector project, and is pointed to from the XML catalog. The XML catalog entry is populated when the SAP connector project is brought into the workspace.

This warning is benign and can be safely ignored, because it does not affect the structural content of your schema definition.

Tuning the SAP adapter for scalability and performance

You can monitor the performance of a message flow that contains SAP nodes by using user trace, and accounting and statistics data, then use that information to tune your flow by configuring an appropriate number of listeners and additional instances for the message flow. For more information, see “Tuning the SAP adapter for scalability and performance” on page 3278.

Related concepts:

“SAP adapter scalability and performance” on page 1949
You can improve performance by configuring the number of listeners on the adapter and the number of additional instances on the message flow to prevent delays when processing synchronous calls from SAP.

“Flow debugger overview” on page 3158
Use the flow debugger in the WebSphere Message Broker Toolkit to track messages through your message flows.

“WebSphere Adapters nodes” on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

Related tasks:

“Using trace” on page 3533
You can use different types of trace to help you with problem determination and troubleshooting.

"Debugging a message flow" on page 3157
Use the tasks described in this section of the documentation to manage and work with the flow debugger.

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

Related reference:

"**mqsichangetrace** command" on page 3822
Use the **mqsichangetrace** command to set the tracing characteristics for a broker.

"**mqsireadlog** command" on page 3905
Use the **mqsireadlog** command to retrieve trace records for the specified component.

"**mqsiformatlog** command" on page 3880
Use the **mqsiformatlog** command to process the XML log created by **mqsireadlog**. The command retrieves and formats any messages that the XML log contains into a form suitable for the locale of the user who runs the command.

WebSphere Adapters technotes

Debugging by using trace

You can use trace in several ways to analyze message flow behavior.

About this task

Enabling user trace shows the history of processing that is carried out in a particular message flow, but it shows only those parts of the messages that were accessed. You can use Trace nodes to write out your own debugging information at specific points in the message flow, including the full message tree at that point, provided that you have coded the flow to include it.

You can use both of these methods to review behavior only after a message has been processed.

Procedure

- "Debugging with user trace" on page 3195
- "Debugging by adding Trace nodes to a message flow" on page 3205

Related concepts:

"Trace" on page 6871
If you cannot get enough information about a particular problem from the entries that are available in the various logs, the next troubleshooting method to consider is using trace. Trace provides more details about what is happening while code runs. The information produced from trace is sent to a specified trace record, so that you or IBM support personnel can analyze it to discover the cause of your problem.

"Flow debugger overview" on page 3158
Use the flow debugger in the WebSphere Message Broker Toolkit to track messages through your message flows.

"Testing message flows by using the Test Client" on page 3144
You can test message flows in a safe environment before they are used on a

production system by using the Test Client.

Related tasks:

"Using trace" on page 3533
You can use different types of trace to help you with problem determination and troubleshooting.

Chapter 10, "Testing and debugging message flow applications," on page 3143
Use the flow debugger to track messages through your message flows and check for errors, or use the Test Client to test your message flows in a safe environment before they are used on a production system.

"Debugging a message flow" on page 3157
Use the tasks described in this section of the documentation to manage and work with the flow debugger.

Related reference:

"Trace node" on page 4942
Use the Trace node to generate trace records that you can use to monitor the behavior of a message flow.

"Message flow debugger" on page 6718
The flow debugger is a visual interface that supports the debugging of message flow applications in the WebSphere Message Broker Toolkit.

"Test Client" on page 6708
Use the Test Client to test message flow applications.

Debugging with user trace

Message flow nodes write messages to user trace when they are processing work. You can use these messages offline to review the activity in a message flow and show information such as which nodes were invoked, what code they ran, and through which terminals the messages was sent.

Before you begin

Before you start:

Before you start to trace a broker, or any of its execution groups or messages flows, the broker must be running, and you must have deployed the message flows by using the WebSphere Message Broker Toolkit.

About this task

If part of the message is parsed by the nodes, user trace shows the fields that are being navigated.

If an error occurs while a message is being processed, the exception is written to user trace. If the error is not caught in the message flow, it is also be written to the system log. Each entry in user trace is prefixed by "BIP". You can search for BIP messages in the information center. For information about the location of user trace log files on different operating systems, see "User trace" on page 6873.

When you start user tracing, you cause additional processing for every activity in the component that you are tracing. Large quantities of data are generated by the components. Expect to see some effect on performance while trace is active. You can limit this additional processing by being selective about what you trace, and by restricting the time during which trace is active.

Procedure

- Trace is inactive by default. Turn it on by following the instructions in "Starting user trace" on page 3197.
- If you need to check what tracing options are currently active for your brokers, use the **mqsireporttrace** command, as described in "Checking user trace options" on page 3199.
- To change user trace options, use the **mqsichangetrace** command, as described in "Changing user trace options" on page 3201.
- To retrieve user trace, use the **mqsireadlog** command, as described in "Retrieving user trace" on page 3204.
- To format the information that is generated by the **mqsireadlog** command, use the **mqsiformatlog** command, as described in "Formatting trace" on page 3543.
- For information about how to interpret the contents of user trace, see "Interpreting trace" on page 3546.
- To stop user trace, use the **mqsichangetrace** command, as described in "Stopping user trace" on page 3202.
- To clear old information from trace files, use the **mqsichangetrace** command, as described in "Clearing old information from trace files" on page 3548.
- Alternatively, you can include a Trace node in your message flows when you design them. Use a Trace node when you want to specify an alternative location for the trace contents. For more details, see "Debugging by adding Trace nodes to a message flow" on page 3205.

Related concepts:

"Trace" on page 6871
If you cannot get enough information about a particular problem from the entries that are available in the various logs, the next troubleshooting method to consider is using trace. Trace provides more details about what is happening while code runs. The information produced from trace is sent to a specified trace record, so that you or IBM support personnel can analyze it to discover the cause of your problem.

"Flow debugger overview" on page 3158
Use the flow debugger in the WebSphere Message Broker Toolkit to track messages through your message flows.

"Testing message flows by using the Test Client" on page 3144
You can test message flows in a safe environment before they are used on a production system by using the Test Client.

Related tasks:

"Using trace" on page 3533
You can use different types of trace to help you with problem determination and troubleshooting.

"Debugging by adding Trace nodes to a message flow" on page 3205
By adding a Trace node to a message flow, you can write debugging messages to a file, to user trace, or to the system log, and review those messages after the message flow has processed some data.

Chapter 10, "Testing and debugging message flow applications," on page 3143
Use the flow debugger to track messages through your message flows and check for errors, or use the Test Client to test your message flows in a safe environment before they are used on a production system.

"Debugging a message flow" on page 3157
Use the tasks described in this section of the documentation to manage and work with the flow debugger.

Related reference:

"Trace node" on page 4942
Use the Trace node to generate trace records that you can use to monitor the behavior of a message flow.

"Message flow debugger" on page 6718
The flow debugger is a visual interface that supports the debugging of message flow applications in the WebSphere Message Broker Toolkit.

"Test Client" on page 6708
Use the Test Client to test message flow applications.

Starting user trace

Use user trace for debugging your applications; you can trace brokers, execution groups, and deployed message flows. Start user trace facilities using the **mqsichangetrace** command or the WebSphere Message Broker Explorer.

Before you begin

Before you start:

Before you start to trace a broker, or one of its execution groups or message flows, the broker must be running, and you must have deployed the message flows by using the WebSphere Message Broker Explorer. Follow the instructions in the Chapter 11, "Packaging and deploying," on page 3209 topic.

About this task

To start a user trace:

Procedure

1. Start WebSphere Message Broker user trace facilities by using the **mqsichangetrace** command, or, for execution groups and assigned message flows, from the WebSphere Message Broker Explorer. You can select only one broker on each invocation of the command, but you can activate concurrent traces for more than one broker, by invoking the command more than once.
2. Specify an individual execution group or message flow within the specified broker to limit the scope of a trace. The events that are recorded when you select the message flow option include:
   - Sending a message from one Message Processing node to the next
   - Evaluating expressions in a Compute or Filter node
3. Start your trace. You can start trace at two levels:

   normal This tracks events that affect objects that you create and delete, such as nodes.

   debug This tracks the beginning and end of a process, as well as monitoring objects that are affected by that process.

Example: starting user trace for the default execution group:
About this task

To start normal level user tracing for the default execution group on a broker that you have created with the name MB7BROKER, on distributed systems, enter the command

```
mqsichangetrace MB7BROKER -u -e default -l normal
```

where:

-u specifies user trace
-e specifies the execution group (in this case, the default execution group)
-l specifies the level of trace (in this case, normal)

z/OS On z/OS, enter the command

```
F MQP1BRK,ct u=yes, e='default', l=normal
```

Example: starting user trace on all the message flows in the default execution group from the WebSphere Message Broker Explorer:
About this task

To start normal level user tracing on all the message flows in the default execution group from the WebSphere Message Broker Explorer:

Procedure

1. In the Navigator view, expand the Brokers folder and right-click the execution group with which you want to work.
2. Click **User Trace All Flows** > **Normal**

Example: starting user trace for a message flow from the WebSphere Message Broker Explorer:
About this task

To start normal level user tracing for one of your message flows from the WebSphere Message Broker Explorer:

Procedure

1. In the Navigator view, expand the Brokers folder and right-click the message flow with which you want to work.
2. Click **User Trace** > **Normal**

Results

An alert saying `Message Flow is tracing at level 'normal'` is displayed in the Alert Viewer.

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

"Trace" on page 6871
If you cannot get enough information about a particular problem from the entries that are available in the various logs, the next troubleshooting method to consider is using trace. Trace provides more details about what is happening while code runs. The information produced from trace is sent to a specified trace record, so that you or IBM support personnel can analyze it to discover the cause of your problem.

Related tasks:

Chapter 13, "Troubleshooting and support," on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

Chapter 11, “Packaging and deploying,” on page 3209
Package resources that you create in the WebSphere Message Broker Toolkit, such as message flows, and deploy them to execution groups on brokers.

“Using trace” on page 3533
You can use different types of trace to help you with problem determination and troubleshooting.

“Starting service trace” on page 3534
Service trace is used to get detailed information about your environment for use by your IBM Support Center.

“Stopping user trace” on page 3202
Use user trace for debugging your applications; you can trace brokers, execution groups, and deployed message flows. Stop user trace facilities by using the **mqsichangetrace** command or the WebSphere Message Broker Explorer.

“Resolving problems with user-defined extensions” on page 3511
Advice for dealing with some common problems that can arise when you work with user-defined extensions

Related reference:

“User trace” on page 6873
User trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Administration log. User trace is inactive by default; you must activate it explicitly by using a command, or by selecting options in the WebSphere Message Broker Toolkit.

“**mqsichangetrace** command” on page 3822
Use the **mqsichangetrace** command to set the tracing characteristics for a broker.

Checking user trace options

Use the **mqsireporttrace** command or the WebSphere Message Broker Explorer to check what tracing options are currently active for your brokers, execution groups and message flows.

About this task

If you use the **mqsireporttrace** command, you must specify the component for which the check is required, for example, the execution group or message flow. The command responds with the current trace status for the component that you have specified.

Example: checking user trace options for a message flow using the WebSphere Message Broker Explorer:
About this task

Follow these steps to check the user trace options for one of your message flows from the WebSphere Message Broker Explorer:

Procedure

1. In the Navigator view, expand the Brokers folder and right-click the message flow with which you want to work.
2. Click **User** to view user trace settings for your message flow.
3. Click **Trace nodes** to see if trace nodes are disabled or enabled for your message flow.
4. Click **Service** to view service trace settings for your message flow.

Example: checking user trace options for a broker:
About this task

To check what options are currently set for the broker `MB7BROKER` and its execution group `test`, on distributed systems, enter the command

```
mqsireporttrace MB7BROKER -u -e test
```

where:

-u specifies user trace
-e specifies the execution group (in this case, test)

z/OS On z/OS, enter the command

```
F MQP1BRK,reporttrace u=yes, e='test'
```

Results

If you have started tracing by following the example in "Starting user trace" on page 3197, the response to the **mqsireporttrace** command is:

```
BIP8098I: Trace level: normal, mode: safe, size: 1024 KB
BIP8071I: Successful command completion
```

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

"Trace" on page 6871
If you cannot get enough information about a particular problem from the entries that are available in the various logs, the next troubleshooting method to consider is using trace. Trace provides more details about what is happening while code runs. The information produced from trace is sent to a specified trace record, so that you or IBM support personnel can analyze it to discover the cause of your problem.

Related tasks:

Chapter 13, "Troubleshooting and support," on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

"Using trace" on page 3533
You can use different types of trace to help you with problem determination and troubleshooting.

"Starting user trace" on page 3197
Use user trace for debugging your applications; you can trace brokers, execution groups, and deployed message flows. Start user trace facilities using the **mqsichangetrace** command or the WebSphere Message Broker Explorer.

"Changing user trace options" on page 3201
Use the **mqsichangetrace** command to change the trace options that you have set. You can also use the WebSphere Message Broker Explorer to change the trace options for execution groups and assigned message flows.

"Checking service trace options" on page 3537
Use the **mqsireporttrace** command or the WebSphere Message Broker Explorer to check what tracing options are currently active for your brokers.

Related reference:

"User trace" on page 6873
User trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Administration log. User trace is inactive by default; you must activate it explicitly by using a command, or by selecting options in the WebSphere Message Broker Toolkit.

"**mqsireporttrace** command" on page 3947
Use the **mqsireporttrace** command to display the trace options currently in effect. Trace can be run only against a broker, or any of its resources.

Changing user trace options

Use the **mqsichangetrace** command to change the trace options that you have set. You can also use the WebSphere Message Broker Explorer to change the trace options for execution groups and assigned message flows.

Procedure

Example: changing user trace from normal to debug:
About this task

To change from a normal level of user trace to a debug level on the default execution group of a broker called MB7BROKER, on distributed systems, enter the command

```
mqsichangetrace MB7BROKER -u -e default -l debug
```

where:

-u specifies user trace
-e specifies the execution group (in this case, the default execution group)
-l specifies the level of trace (in this case, changing it to debug)

z/OS On z/OS, enter the command

```
F MQP1BRK,ct u=yes, e='default', l=debug
```

Example: changing user trace from normal to debug from the WebSphere Message Broker Explorer:
About this task

To change from a normal level of user trace to a debug level for one of your message flows from the WebSphere Message Broker Explorer:

Procedure

1. In the Navigator view, expand the Brokers folder and right-click the message flow with which you want to work.
2. Click **User trace** > **Debug**.

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

"Trace" on page 6871
If you cannot get enough information about a particular problem from the entries that are available in the various logs, the next troubleshooting method to consider is using trace. Trace provides more details about what is happening while code runs. The information produced from trace is sent to a specified trace record, so that you or IBM support personnel can analyze it to discover the cause of your problem.

Related tasks:

Chapter 13, "Troubleshooting and support," on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

"Using trace" on page 3533
You can use different types of trace to help you with problem determination and troubleshooting.

"Starting user trace" on page 3197
Use user trace for debugging your applications; you can trace brokers, execution groups, and deployed message flows. Start user trace facilities using the **mqsichangetrace** command or the WebSphere Message Broker Explorer.

"Checking user trace options" on page 3199
Use the **mqsireporttrace** command or the WebSphere Message Broker Explorer to check what tracing options are currently active for your brokers, execution groups and message flows.

"Changing service trace options" on page 3538
Use the **mqsichangetrace** command or the WebSphere Message Broker Explorer to change the service trace options that you have set.

Related reference:

"User trace" on page 6873
User trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Administration log. User trace is inactive by default; you must activate it explicitly by using a command, or by selecting options in the WebSphere Message Broker Toolkit.

"**mqsichangetrace** command" on page 3822
Use the **mqsichangetrace** command to set the tracing characteristics for a broker.

Stopping user trace

Use user trace for debugging your applications; you can trace brokers, execution groups, and deployed message flows. Stop user trace facilities by using the **mqsichangetrace** command or the WebSphere Message Broker Explorer.

About this task

Use the **mqsichangetrace** command with a trace level of `none` to stop an active trace. This action stops the trace activity for the component that you specify on the command. It does not affect active traces on other components. For example, if you stop tracing on the execution group `test`, an active trace on another execution group continues.

You can also stop an active trace for execution groups or assigned message flows by using the WebSphere Message Broker Explorer. If you redeploy a component from the WebSphere Message Broker Explorer, trace for that component is returned to its default setting of `none`.

Example: stopping user trace on the default execution group:
About this task

To stop the trace started by the command shown in "Starting user trace" on page 3197, on distributed systems, enter the following command:

```
mqsichangetrace MB7BROKER -u -e default -l none
```

where:

-u specifies user trace
-e specifies the execution group (in this case, the default execution group)
-l specifies the level of trace (in this case, none)

z/OS On z/OS, enter the following command:

```
F MQP1BRK,ct u=yes, e='default', l=none
```

Example: stopping user trace on a message flow from the WebSphere Message Broker Explorer:
About this task

You can stop user trace on individual message flows, or select an execution group and stop user trace on all the message flows in the execution group.

To stop trace for one of your message flows from the WebSphere Message Broker Explorer:

Procedure

1. In the Navigator view, expand the Brokers folder and right-click the message flow with which you want to work.
2. Click **User trace** > **None**.

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

"Trace" on page 6871
If you cannot get enough information about a particular problem from the entries that are available in the various logs, the next troubleshooting method to consider is using trace. Trace provides more details about what is happening while code runs. The information produced from trace is sent to a specified trace record, so that you or IBM support personnel can analyze it to discover the cause of your problem.

Related tasks:

Chapter 13, "Troubleshooting and support," on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

"Using trace" on page 3533
You can use different types of trace to help you with problem determination and troubleshooting.

"Starting user trace" on page 3197
Use user trace for debugging your applications; you can trace brokers, execution groups, and deployed message flows. Start user trace facilities using the **mqsichangetrace** command or the WebSphere Message Broker Explorer.

"Stopping service trace" on page 3540
Use the **mqsichangetrace** command or the WebSphere Message Broker Explorer to stop an active service trace.

"Clearing old information from trace files" on page 3548
If the component that you are tracing has stopped, you can delete its trace files from the log subdirectory of the WebSphere Message Broker home directory.

Related reference:

"User trace" on page 6873
User trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Administration log. User trace is inactive by default; you must activate it explicitly by using a command, or by selecting options in the WebSphere Message Broker Toolkit.

"**mqsichangetrace** command" on page 3822
Use the **mqsichangetrace** command to set the tracing characteristics for a broker.

Retrieving user trace

Use the **mqsireadlog** command to access the trace information that is recorded by the user trace facilities.

Procedure

This command retrieves the trace details according to parameters that you specify on the command, and writes the requested records to a file, or to the command line window, in XML format.

Example: retrieving user trace information in XML format:
Procedure

To retrieve information for the user trace activated with the **mqsichangetrace** command and write it to an output file, on distributed systems, enter the command

```
mqsireadlog MB7BROKER -u -e default -o trace.xml
```

where:

-u specifies user trace
-e specifies the execution group (in this case, the default execution group)
-o specifies the output file (in this case, trace.xml)

Results

This sends a log request to the broker to retrieve the user trace log, and stores the responses in the `trace.xml` file. You can view this file using a plain text editor.

Related concepts:

"Trace" on page 6871
If you cannot get enough information about a particular problem from the entries that are available in the various logs, the next troubleshooting method to consider is using trace. Trace provides more details about what is happening while code runs. The information produced from trace is sent to a specified trace record, so that you or IBM support personnel can analyze it to discover the cause of your problem.

Related tasks:

Chapter 13, “Troubleshooting and support,” on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

“Using trace” on page 3533
You can use different types of trace to help you with problem determination and troubleshooting.

“Starting user trace” on page 3197
Use user trace for debugging your applications; you can trace brokers, execution groups, and deployed message flows. Start user trace facilities using the **mqsichangetrace** command or the WebSphere Message Broker Explorer.

“Retrieving service trace” on page 3542
Use the **mqsireadlog** command to access the trace information recorded by the service trace facilities.

Related reference:

“User trace” on page 6873
User trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Administration log. User trace is inactive by default; you must activate it explicitly by using a command, or by selecting options in the WebSphere Message Broker Toolkit.

“**mqsichangetrace** command” on page 3822
Use the **mqsichangetrace** command to set the tracing characteristics for a broker.

“**mqsireadlog** command” on page 3905
Use the **mqsireadlog** command to retrieve trace records for the specified component.

Debugging by adding Trace nodes to a message flow

By adding a Trace node to a message flow, you can write debugging messages to a file, to user trace, or to the system log, and review those messages after the message flow has processed some data.

About this task

You must add a Trace node when the message flow is designed. “Viewing the logical message tree in trace output” on page 1481 explains how to view the structure of the logical message tree at any point in the message flow, and contains an example of the message content. You can turn the Trace node off when a message flow is promoted to production to improve performance, but you can turn the node on when required. Performance can be affected when Trace nodes are active. The extent to which performance is affected depends on the destination that you choose for the debugging messages; for example, writing to user trace is typically faster than writing to a file or to the system log.

The debug messages can include writing part or all of the logical message tree, but they can also include hard-coded strings to identify a particular point in the message flow (such as `PRINTF` in a C program). If you write the entire message tree in a Trace node, the behavior of the message flow might be changed. Typically, only the parts of the message that are referenced are parsed, rather than the entire message.

Procedure

- Add a Trace node to your message flow, then set the following properties on the node (as described in detail in "Trace node" on page 4942):
  - Set the destination of the trace record that is written by the node to `User Trace`, `Local Error Log`, or `File`.
  - If you choose `File`, set the file path of the file to which to write records.
  - Use the `Pattern` property to create an ESQL pattern that specifies the data to be included in the trace record.
  - Specify the message catalog from which the error text for the error number of the exception is extracted.
  - Specify the error number of the message that is written.
  -
- After you have added a Trace node to your message flow, you can turn it on or off, as described in "Switching Trace nodes on and off" on page 3555.
- To view the structure of the logical message tree at any point in the message flow, include a Trace node and write some or all of the message (including headers and all four message trees) to the trace output destination. The following topics describes how to view that output: "Viewing the logical message tree in trace output" on page 1481
- If you write debugging messages to user trace, the following topic describes how to retrieve user trace: "Retrieving user trace" on page 3204

Related concepts:

"Trace" on page 6871
If you cannot get enough information about a particular problem from the entries that are available in the various logs, the next troubleshooting method to consider is using trace. Trace provides more details about what is happening while code runs. The information produced from trace is sent to a specified trace record, so that you or IBM support personnel can analyze it to discover the cause of your problem.

"Flow debugger overview" on page 3158
Use the flow debugger in the WebSphere Message Broker Toolkit to track messages through your message flows.

"Testing message flows by using the Test Client" on page 3144
You can test message flows in a safe environment before they are used on a production system by using the Test Client.

Related tasks:

"Using trace" on page 3533
You can use different types of trace to help you with problem determination and troubleshooting.

"Debugging with user trace" on page 3195
Message flow nodes write messages to user trace when they are processing work. You can use these messages offline to review the activity in a message flow and show information such as which nodes were invoked, what code they ran, and through which terminals the messages was sent.

Chapter 10, "Testing and debugging message flow applications," on page 3143
Use the flow debugger to track messages through your message flows and check for errors, or use the Test Client to test your message flows in a safe environment before they are used on a production system.

"Debugging a message flow" on page 3157
Use the tasks described in this section of the documentation to manage and work with the flow debugger.

Related reference:

"Trace node" on page 4942
Use the Trace node to generate trace records that you can use to monitor the behavior of a message flow.

"Message flow debugger" on page 6718
The flow debugger is a visual interface that supports the debugging of message flow applications in the WebSphere Message Broker Toolkit.

"Test Client" on page 6708
Use the Test Client to test message flow applications.

Chapter 11. Packaging and deploying

Package resources that you create in the WebSphere Message Broker Toolkit, such as message flows, and deploy them to execution groups on brokers.

About this task

Read the overview section to learn about the different ways in which you can deploy resources, and the resources and properties associated with deployment:

- "Packaging and deployment overview" on page 3210
  - "Deployment methods" on page 3211
  - "Message flow application deployment" on page 3213

The tasks in this section assume that you have already set up your broker, and connected to it. For information about how to complete those tasks, and other related actions, see "Managing brokers" on page 900.

Refer to the following topics for information about how to package and deploy a message flow application:

- "Packaging resources" on page 3221
  - "Creating a broker archive" on page 3222
  - "Adding files to a broker archive" on page 3223
  - "Refreshing the contents of a broker archive" on page 3233
- "Deploying resources" on page 3234
  - "Deploying a broker archive file" on page 3235
  - "Deploying a message flow that uses WebSphere Adapters" on page 3240
  - "Importing a broker archive file to the WebSphere Message Broker Explorer" on page 3242

Further topics describe other deployment tasks:

- "Checking the results of deployment" on page 3243
- "Renaming objects that are deployed to execution groups" on page 3246
- "Removing a deployed object from an execution group" on page 3246

Related concepts:

"Execution groups" on page 53
An execution group is a named grouping of message flows that have been assigned to a broker. The broker enforces a degree of isolation between message flows in distinct execution groups by ensuring that they run in separate address spaces, or as unique processes.

"Broker archive" on page 3216
The unit of deployment to the broker is the *broker archive* or *BAR* file.

Related tasks:

"Managing brokers" on page 900
Work with your existing brokers to manage their connections and their active status by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer. You can use the Administration API (also known as the CMP API) to complete some of these actions.

"Deploying resources to a broker from a CMP application" on page 982
Deploy BAR files to the brokers in your broker network from a CMP application.

"Resolving problems when deploying message flows or message sets" on page 3436
Use the advice given here to help you to resolve common problems that can arise when you deploy message flows or message sets.

Related reference:

"**mqsideploy** command" on page 3872
Use the **mqsideploy** command to make a deployment request to the broker.

Packaging and deployment overview

Deployment is the process of transferring data to an execution group on a broker so that it can take effect in the broker. Message flows and associated resources are packaged in broker archive (BAR) files for deployment.

When you create application resources such as message flows in the WebSphere Message Broker Toolkit, you must distribute them to the brokers on which you want them to run. Data for message flows and associated resources is packaged in a broker archive (BAR) file before being sent to the broker.

You can package and deploy message flows and associated resources in one step by using the WebSphere Message Broker Toolkit. You can use this approach when you are developing your flows to make changes and quickly see the results. When you want to deploy to a production environment, you can separate the packaging and deployment steps by first packaging your resources into a BAR file and then deploying that file at a later time. You can create BAR files in the following ways:

- By using the WebSphere Message Broker Toolkit
- By using the **mqsicreatebar** command

For more information about packaging resources, see "Packaging resources" on page 3221.

After you create a BAR file, you can customize the file for deployment to different brokers; for example, to specify queue names or data source names. For more information about customizing your BAR file, see "Configurable properties of a broker archive" on page 3217.

After creating a BAR file and customizing it, you can deploy the BAR file in the following ways:

- From the WebSphere Message Broker Toolkit
- From the WebSphere Message Broker Explorer
- By using the **mqsideploy** command
- By using functions defined by the Administration API (also known as the CMP API)

Depending on your work patterns, you might use all these methods at different times. These options are described in "Deployment methods" on page 3211.

After you read these overview topics, find detailed instructions for the tasks that you want to complete in subsequent topics in this section.

Related tasks:

Chapter 11, "Packaging and deploying," on page 3209
Package resources that you create in the WebSphere Message Broker Toolkit, such

as message flows, and deploy them to execution groups on brokers.

"Packaging resources" on page 3221
Package message flow applications by adding required resources, optionally with their source files, to a broker archive (BAR) file.

"Deploying resources" on page 3234
Deploy message flow applications to execution groups by sending a broker archive (BAR) file to a broker, which unpacks and stores the contents ready for when your message flows are started.

"Checking the results of deployment" on page 3243
After you have made a deployment, check that the operation has completed successfully.

"Deploying resources to a broker from a CMP application" on page 982
Deploy BAR files to the brokers in your broker network from a CMP application.

Related reference:

"WebSphere Message Broker Toolkit" on page 6783
The WebSphere Message Broker Toolkit provides your application development environment on Windows and Linux on x86.

"**mqsideploy** command" on page 3872
Use the **mqsideploy** command to make a deployment request to the broker.

Deployment methods

Choose the appropriate method of deployment to suit the way in which you are working. You can use the WebSphere Message Broker Toolkit, the WebSphere Message Broker Explorer, the **mqsideploy** command, or functions described by the Administration API for WebSphere Message Broker (also known as the CMP API).

Using the WebSphere Message Broker Toolkit

The Brokers view of the WebSphere Message Broker Toolkit displays all the brokers on the local machine that you have defined, and any remote brokers to which you have defined connections. If you expand a broker, all the execution groups in that broker are displayed, as well as deployed message flows and their associated resources. You can drag a message flow or a broker archive (BAR) file from the Broker Development view onto an execution group to deploy it. Alternatively, you can right-click an execution group to select a message flow or BAR file to deploy to the selected execution group.

You would typically use the WebSphere Message Broker Toolkit if your primary role is as an application developer, or if you are new to WebSphere Message Broker.

Using the WebSphere Message Broker Explorer

The Navigator view of the WebSphere Message Broker Explorer displays all the brokers on the local machine that you have defined, and any remote brokers to which you have defined connections. If you expand a broker, all the execution groups in that broker are displayed, as well as deployed message flows and their associated resources. You can import BAR files into the WebSphere Message Broker Explorer. The BAR files are displayed in the Broker Archive Files folder in the WebSphere Message Broker Explorer. You can drag a BAR file from the Broker Archive Files folder onto an execution group to deploy the contents of the broker archive. Alternatively, you can right-click an execution group or BAR file and click **Deploy File** to deploy resources to an execution group. You can also drag BAR files from your file system directly onto an execution group to deploy the contents of the broker archive.

You would typically use the WebSphere Message Broker Explorer if your primary role is as a WebSphere Message Broker administrator.

Using the `mqsideploy` command

You can deploy from the command line by using the **`mqsideploy`** command. On the command line, specify the connection details and parameters that are specific to the deployment.

You would typically use the **`mqsideploy`** command in a script when you are more familiar with WebSphere Message Broker.

Using the CMP API

You can control deployment from a Java program by using functions described by the CMP API. You can also interrogate the responses from the broker and take appropriate action.

Java applications can also use the CMP API to control other objects, such as brokers and execution groups. Therefore, you can use the CMP API to create and manipulate all your brokers and associated resources programmatically.

Deployment results

Whichever deployment method you use, configuration changes are attempted immediately.

- If you are using the WebSphere Message Broker Toolkit, the Deployment Log view is updated to show the results when your deployment completes. If the deployment fails, the reason for the failure is displayed in this view.
- If you are using the WebSphere Message Broker Explorer, you can see the deployment status for all users connected to the broker. The Administration Queue shows all deployments that are currently being processed by the broker, and the Administration Log shows all recent configuration requests and changes made to the broker.
- If you use the **`mqsideploy`** command, the command completes when the broker has processed the deployment request, or when the wait time, defined by the **`-w`** parameter, has expired, whichever occurs first. The results of the deployment are displayed as output from the command.
- If you are using the CMP API, you can view the results of the deployment in the following ways:
  - Review the DeployResult object that is returned from the deployment methods.
  - Access the LogProxy object that represents the administration log.
  - Access the AdminQueueProxy object that represents the administration queue.

Related concepts:

"WebSphere Message Broker Toolkit" on page 31
The WebSphere Message Broker Toolkit is an integrated development environment and graphical user interface based on the Eclipse platform.

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

Related tasks:

Chapter 11, "Packaging and deploying," on page 3209
Package resources that you create in the WebSphere Message Broker Toolkit, such as message flows, and deploy them to execution groups on brokers.

"Developing applications that use the Administration API" on page 956
Develop Java applications that use the Administration API (also known as the CMP API) to communicate with, deploy to, and manage brokers and their associated resources.

"Deploying resources to a broker from a CMP application" on page 982
Deploy BAR files to the brokers in your broker network from a CMP application.

Related reference:

"WebSphere Message Broker Toolkit" on page 6783
The WebSphere Message Broker Toolkit provides your application development environment on Windows and Linux on x86.

"**mqsideploy** command" on page 3872
Use the **mqsideploy** command to make a deployment request to the broker.

Message flow application deployment

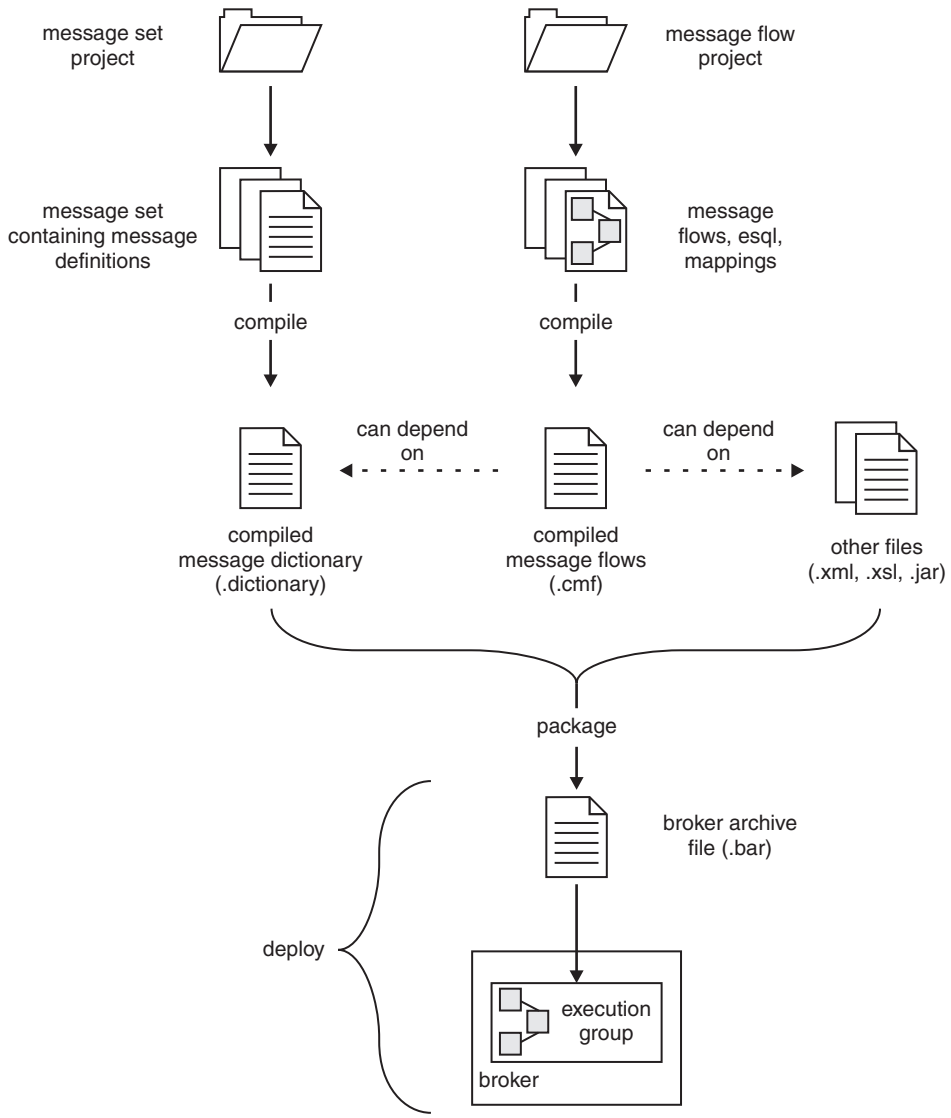
Package all the resources in your message flow into a broker archive (BAR) file for deployment.

When you add files to the broker archive, they are automatically compiled as part of the process. JAR files that are required by JavaCompute nodes in message flows are added automatically from your Java project.

The BAR file is a compressed file that is sent to the broker, where its contents are extracted and distributed to execution groups.

The mode in which your broker is working can affect the number of execution groups and message flows that you can deploy, and the types of node that you can use. See "Restrictions that apply in each operation mode" on page 3657.

The following diagram shows the flow of events when you deploy a message flow application.



The diagram illustrates the following sequence of events.

1. You create a broker archive.
2. You add files to the broker archive.
3. You deploy the BAR file by sending it to the broker, from where its contents are distributed to the execution groups.

You can deploy a BAR file in two ways:

- "Incremental BAR file deployment" on page 3215. Deployed files are added to the execution group. Files that exist in the execution group are replaced by the new version.
- "Complete BAR file deployment" on page 3215. Files that are already deployed to the execution group are removed before the entire contents of the BAR file are deployed. Therefore, nothing is left in the execution group from previous deployments.

Incremental BAR file deployment

If you run an incremental deployment of a BAR file, the broker extracts the contents of the BAR file and sends the contents to the specified execution group. The following conditions are applied when a file is deployed to the BAR file.

- If a file in the BAR file has the same name as an object that is already deployed to the execution group, the version that is already deployed is replaced by the version in the BAR file.
- If a file in the BAR file is of zero length, and a file of that name has already been deployed to the execution group, the deployed file is removed from the execution group.

Use incremental BAR file deployment to deploy message flows, message sets, or other deployable objects incrementally to an execution group.

To completely clear the contents of the execution group before the BAR file is deployed, use complete BAR file deployment instead.

Complete BAR file deployment

If you run a complete deployment of a BAR file, the broker extracts the deployable content of the BAR file and sends the contents to the specified execution group, first removing any existing deployed contents of the execution group.

Use a complete BAR file deployment to deploy message flows, message sets, or other deployable objects to an execution group.

To merge the existing contents of the execution group with the contents of the BAR file, use incremental BAR file deployment instead.

Related concepts:

"Packaging and deployment overview" on page 3210
Deployment is the process of transferring data to an execution group on a broker so that it can take effect in the broker. Message flows and associated resources are packaged in broker archive (BAR) files for deployment.

"Broker archive" on page 3216
The unit of deployment to the broker is the *broker archive* or *BAR* file.

Related tasks:

"Adding files to a broker archive" on page 3223
To deploy files to an execution group, include them in a broker archive (BAR) file.

"Deploying resources" on page 3234
Deploy message flow applications to execution groups by sending a broker archive (BAR) file to a broker, which unpacks and stores the contents ready for when your message flows are started.

"Deploying resources to a broker from a CMP application" on page 982
Deploy BAR files to the brokers in your broker network from a CMP application.

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

Related reference:

"**mqsideploy** command" on page 3872
Use the **mqsideploy** command to make a deployment request to the broker.

"JavaCompute node" on page 4514
Use the JavaCompute node to work with messages by using the Java language.

Broker archive

The unit of deployment to the broker is the *broker archive* or *BAR* file.

The BAR file is a compressed file that can contain a number of different files:

- A `.cmf` file for each message flow. This file is a compiled version of the message flow. You can have any number of these files within your BAR file.
- A `.dictionary` file for each message set dictionary. You can have any number of these files within your BAR file.
- One or more XSD compressed files (`.xsdzip`), if XML schema and WSDL are defined within a message set.
- A `broker.xml` file. This file is called the *broker deployment descriptor*. You can have only one of these files within your BAR file. This file, in XML format, is contained in the `META-INF` folder of the compressed file and can be modified by using a text editor or shell script.
- One or more XML files (`.xml`), style sheets (`.xsl`), and XSLT files (`.xlst`), if required by nodes in the message flows you have added to this BAR file. The XSLTransform node is one that might require these files.
- One or more JAR files, if required by JavaCompute nodes in the message flows you have added to this BAR file.
- One or more inbound or outbound adapter files (`.inadapter` or `.outadapter`), if required by WebSphere Adapter nodes (for example, the SiebelInput node) in the message flows you have added to this BAR file.
- One or more PHP script files (`.php`), if required by PHPCompute nodes in the message flows you have added to this BAR file.
- Other files that you might want to associate with this BAR file. For example, you might want to include Java source files, `.msgflow` files, or `.wsdl` files for future reference. BAR files can contain all files types.

Related concepts:

"Message flow application deployment" on page 3213
Package all the resources in your message flow into a broker archive (BAR) file for deployment.

Related tasks:

"Deploying resources" on page 3234
Deploy message flow applications to execution groups by sending a broker archive (BAR) file to a broker, which unpacks and stores the contents ready for when your message flows are started.

"Creating a broker archive" on page 3222
Create a separate broker archive (BAR) file for each configuration that you want to deploy to execution groups on your brokers.

"Adding files to a broker archive" on page 3223
To deploy files to an execution group, include them in a broker archive (BAR) file.

Related reference:

"**mqsideploy** command" on page 3872
Use the **mqsideploy** command to make a deployment request to the broker.

"XSLTransform node" on page 4968
Use the XSLTransform node to transform an XML message to another form of message, according to the rules provided by an XSL (Extensible Stylesheet Language) style sheet, and to set the `Message domain`, `Message set`, `Message type`, and `Message format` for the generated message.

"JavaCompute node" on page 4514
Use the JavaCompute node to work with messages by using the Java language.

Configurable properties of a broker archive

System objects that are defined in message flows can have properties that you can update in the broker archive (BAR) file before deployment.

An administrator can use configurable properties to update target-dependent properties, such as queue names, queue manager names, and database connections.

By changing configurable properties, you can customize a BAR file for a new broker (for example, a test system) without needing to edit and rebuild the message flows or the resources that they work with, such as message mappings, ESQL code, and Java code. Properties that you define are contained in the deployment descriptor, `META-INF/broker.xml` The deployment descriptor is parsed when the BAR file is deployed.

Edit the configurable properties by using either the Broker Archive editor or the **mqsiapplybaroverride** command. In the Broker Archive editor, click a resource on the **Manage** tab to see its properties in the Properties view.

Use the supplied editor and command to ensure that the BAR file contents are correct after the changes are applied. You can also edit the XML-format deployment descriptor manually by using an external text editor or shell script; in this case, you must ensure that you have not invalidated the XML content.

Related tasks:

Chapter 11, "Packaging and deploying," on page 3209
Package resources that you create in the WebSphere Message Broker Toolkit, such as message flows, and deploy them to execution groups on brokers.

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

Related reference:

"Configurable message flow properties" on page 4020
When you add a message flow to a broker archive (BAR) file in preparation for deploying it to a broker, you can set additional properties that influence its run time operation. These properties are available for review and update when you select the **Manage and Configure** tab for the broker archive file.

"**mqsiapplybaroverride** command" on page 3684
Use the **mqsiapplybaroverride** command to replace configurable values in the broker archive (BAR) deployment descriptor with new values that you specify in a properties file.

Version and keyword information for deployable objects

Use the Broker Archive editor to view the version and keyword information of deployable objects.

You can display properties of deployed objects, and can modify associated comments:

- "Object version in the Broker Archive editor" on page 1443
- "Version, deployment time, and keywords of deployed objects" on page 1443
- "Path and Comment columns" on page 1444

Object version in the Broker Archive editor

The Version column on the Manage page of the Broker Archive editor displays the version tag for the following objects that have a defined version.

- `.dictionary` files
- `.cmf` files
- Embedded JAR files with a version defined in a `META-INF/keywords.txt` file

You cannot edit the Version column.

You can use the **mqsireadbar** command to list the keywords that are defined for each deployable file within a deployable archive file. For more information, see "**mqsireadbar** command" on page 3697.

Version, deployment time, and keywords of deployed objects

In the WebSphere Message Broker Explorer, the Properties QuickView displays the following properties for all deployed objects:

- Version
- Deployment time
- All defined keywords

For example, you deploy a message flow with the following literal strings:

- `$MQSI_VERSION=v1.0 MQSI$`
- `$MQSI Author=fred MQSI$`
- `$MQSI Subflow 1 Version=v1.3.2 MQSI$`

The Properties view displays these properties:

Property	Description
Deployment Time	Date and time of deployment
Modification Time	Date and time of modification. **Note:** This property has no concept of time zone, therefore it is only meaningful if you know in which time zone it was last modified.
Version	`v1.0`
Author	`fred`
Subflow 1 Version	`v1.3.2`

If the keyword information is not available, a message is displayed in the Properties view to indicate the reason; for example, if keyword resolution has not been enabled at deployment time, the Properties view displays the message `Deployed with keyword search disabled`.

Path and Comment columns

If you add source files, the Path column on the Manage tab is populated automatically.

To add a comment, double-click the Comment column and type the text that you require.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Adding keywords to ESQL files" on page 2486
You can add keywords to ESQL files to contain information that you want to associate with a message flow.

"Keywords in subflows" on page 1447
You can embed keywords in each subflow that you use in a message flow.

Related tasks:

"Adding keywords to JAR files" on page 2660
If a BAR file contains JAR files, you can associate keywords with the JAR files.

Related reference:

"Description properties for a message flow" on page 4016
The description properties for a message flow include the `Version`, `Short Description` and `Long Description`. To view and edit the properties of a message flow click **Flow** > **Properties**.

"**mqsireadbar** command" on page 3697
Use the **mqsireadbar** command to read a deployable BAR file and identify its defined keywords.

"Broker Application Development perspective" on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

"Adding keywords to XSL style sheets" on page 4975
Embedded keywords in an XSL style sheet; their location is not restricted. You can also add a keyword as an XML comment.

"Broker Archive editor" on page 6794
Use the Broker Archive editor to create and manage broker archive (BAR) files.

"Guidance for defining keywords" on page 4017
You can add extra information to an object in the form of one or more keywords.

WebSphere Adapters deployment

When you have run the Adapter Connection wizard and created a message flow, you must deploy the resources that are generated by adding them to a broker archive (BAR) file.

To build a WebSphere Message Broker application that integrates with an Enterprise Information System (EIS), such as SAP, Siebel, JD Edwards, or PeopleSoft, you need the following resources:

- A message flow that contains at least one WebSphere Adapters node
- A message set (created by the Adapter Connection wizard) that describes the logical model of the data, as defined by the EIS
- An adapter, which includes two sets of information that are used by the WebSphere Adapters node in the message flow:
  - Connection information
  - Interface information

    The interface information contains a list of methods. For outbound adapters, methods define the operations or services that can be run on the EIS by the WebSphere Adapters request node. For inbound adapters, the method defines callout functions or events from the EIS that cause the WebSphere Adapters input node to propagate a message through the message flow.

    For each method, the information consists of the name of the method, and the name and namespaces of the message types that are used for input and output. To run the method successfully, the message types must be defined in the message set.

Iterative deployment

The message flow can be coded with knowledge of the logical model of the data that is exchanged with the EIS (for example, where Mapping nodes are used to transform the data), but it can also act as a gateway to the EIS, where no transformation of data takes place. When the flow acts as a gateway, you need to be able to run new operations or respond to new events in the EIS without changing or reloading the resources that are already deployed.

You do not need to stop the message flow when you deploy the secondary adapter. During deployment of the secondary adapter, the primary adapter provides connection information, and the secondary adapter supplies additional interface information.

You can use iterative deployment to deploy the resources that are required to support the new methods, without affecting any resources that are already deployed. Iterative deployment is possible by using primary and secondary adapters and message sets.

- The primary adapter for a WebSphere Adapters node contains its connection information and part of its interface; the secondary adapters contain the rest of the interface.
- The primary message set contains message model metadata for the parts of the interfaces that are supported by the node; the secondary message sets define the rest of the model.

You do not need to stop the message flow when you deploy the secondary message set and adapter. During deployment of the secondary adapter, the primary adapter provides connection information, and the secondary adapter supplies additional interface information.

Related tasks:

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

"Configuring WebSphere Adapters nodes for secondary adapters" on page 2040
You can deploy the resources that are required to support new methods in an Enterprise Information System (EIS), without affecting any resources that are already deployed, by using primary and secondary adapters and message sets. You must configure the WebSphere Adapters nodes in your message flows to use the secondary adapters.

"Calling new services from a WebSphere Adapters request node without changing existing deployed resources" on page 2042
If your message flow acts as a gateway to an Enterprise Information System (EIS), you can use it to call new services that did not exist when you developed the flow. Therefore, if a new service is provided by the EIS, you do not have to modify and retest the message flow.

"Handling new event types from a Websphere Adapters input node without changing existing deployed resources" on page 2044
You can create an event handler to an Enterprise Information System (EIS) to handle new event types that did not exist when you first developed your message flow. Therefore, if a new event is provided by the EIS, you do not have to modify and retest the message flow.

"Resolving problems that occur during deployment of message flows" on page 3440
Use the advice given here to help you to resolve problems that can arise during

deployment of message flows or message sets.

"Enhancing existing adapters with newly discovered objects" on page 2063
In WebSphere Message Broker Version 7.0, you can take an adapter component that was created by using the Adapter Connection wizard, and update it with newly discovered objects from the Enterprise Information System (EIS). This facility is known as *iterative discovery*. You can either add the new objects without modifying existing objects, or replace existing objects.

"Connecting to Enterprise Information Systems" on page 1912
Use WebSphere Adapters to communicate with Enterprise Information Systems (EIS) such as SAP, Siebel, PeopleSoft, and JD Edwards.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

Packaging resources

Package message flow applications by adding required resources, optionally with their source files, to a broker archive (BAR) file.

Before you begin

Before you start:

Develop and test the message flow applications that you want to package; see Chapter 9, "Developing message flow applications," on page 1019 and Chapter 10, "Testing and debugging message flow applications," on page 3143.

About this task

Before you can deploy your message flow applications to a broker, you must package them into BAR files. If you want to deploy a single message flow, the packaging step can be completed automatically when you deploy the flow; see "Deploying resources" on page 3234.

In most cases, you are working with one or more message flows, one or more message sets, and other objects that you want to deploy with the message flows. In these cases, you can deploy these resources together by packaging them in a BAR file yourself, and deploying just that file. For instructions about how to package resources into a BAR file, see the following topics:

1. "Creating a broker archive" on page 3222
2. "Adding files to a broker archive" on page 3223
3. "Refreshing the contents of a broker archive" on page 3233

You can edit configurable properties for the BAR file in the BAR file editor, or by using the **mqsiapplybaroverride** command. For more details, see "Editing configurable properties" on page 3227.

What to do next

Next:

Deploy your BAR files to a broker; see "Deploying resources" on page 3234.

Related concepts:

"Message flow application deployment" on page 3213
Package all the resources in your message flow into a broker archive (BAR) file for deployment.

Related tasks:

Chapter 11, "Packaging and deploying," on page 3209
Package resources that you create in the WebSphere Message Broker Toolkit, such as message flows, and deploy them to execution groups on brokers.

Creating a broker archive

Create a separate broker archive (BAR) file for each configuration that you want to deploy to execution groups on your brokers.

About this task

You can create a BAR file in two ways:

- "Using the WebSphere Message Broker Toolkit"
- "Using the **mqsicreatebar** command"

Using the WebSphere Message Broker Toolkit

About this task

Follow these steps to create a BAR file by using the WebSphere Message Broker Toolkit:

Procedure

1. Click **File** > **New** > **Message Broker Archive**.
2. Enter a name for the BAR file that you are creating.
3. Click **Finish**.

Results

A file with a `.bar` extension is created and is displayed in the Broker Application Development perspective Navigator view, under the Broker Archives folder. The Content editor for the BAR file opens.

What to do next

Next:

Add files to the BAR file and deploy the BAR file by following the instructions in the following topics:

1. "Adding files to a broker archive" on page 3223
2. "Deploying a broker archive file" on page 3235

Using the `mqsicreatebar` command

About this task

Follow these steps to create a BAR file by using the **mqsicreatebar** command:

Procedure

1. Open a command window that is configured for your environment.
2. Enter the command, typed on a single line. For example:

```
mqsicreatebar -b barName -o filePath -p projectNames -cleanBuild
```

You must specify the **-b** (BAR file name) and **-o** (path for included files) parameters. The **-p** (project names) parameter is optional. For further details, see the "**mqsicreatebar** command" on page 3699.

If you have changed resources in the BAR file by using external tools, add the **-cleanBuild** parameter to refresh all the projects and run a clean build.

A file with a `.bar` extension is created.

What to do next

Next:

Add files to the BAR file and deploy the BAR file by following the instructions in the following topics:

1. "Adding files to a broker archive"
2. "Deploying a broker archive file" on page 3235

Related concepts:

"Message flow application deployment" on page 3213
Package all the resources in your message flow into a broker archive (BAR) file for deployment.

"Broker archive" on page 3216
The unit of deployment to the broker is the *broker archive* or *BAR* file.

Related tasks:

"Deploying resources" on page 3234
Deploy message flow applications to execution groups by sending a broker archive (BAR) file to a broker, which unpacks and stores the contents ready for when your message flows are started.

"Resolving problems when you use the WebSphere Message Broker Toolkit" on page 3480
Use the advice given here to help you to resolve common problems that might occur you use the WebSphere Message Broker Toolkit.

"Resolving problems when deploying message flows or message sets" on page 3436
Use the advice given here to help you to resolve common problems that can arise when you deploy message flows or message sets.

Related reference:

"Broker Archive editor" on page 6794
Use the Broker Archive editor to create and manage broker archive (BAR) files.

"**mqsicreatebar** command" on page 3699
Use the **mqsicreatebar** command to create deployable broker archive (BAR) files containing message flows and dictionaries.

Adding files to a broker archive

To deploy files to an execution group, include them in a broker archive (BAR) file.

Before you begin

Before you start:

Create a BAR file for each configuration that you want to deploy. For more details, see "Creating a broker archive" on page 3222.

About this task

You can add any deployable resources from your workspace to a BAR file. If you select **Include source files**, the source and project files for all message flows, message sets, or other deployable resources in the broker archive are included. For more information about the files that you can include in a broker archive, see "Broker archive" on page 3216.

Subflows are not displayed in the BAR file as separate items, and are added automatically. To include these subflows, you have to add only the parent flow.

You can add Flow, ESQL, Java JAR, PHP, MAP, XML, XSLT files and Message Sets manually by following these steps. However, JAR files or .NET assemblies (DLL files) that are required by JavaCompute nodes in message flows are added automatically from your Java project when you add the message flow. XML and XSL files are also added automatically if they are required by the flow.

You do not have to redeploy JAR files unless you have updated them. If one or more JAR files in your BAR file are present on the computer where the broker is running, you can safely remove them from your BAR file before you deploy again. JAR files available to the broker include JAR files that you have deployed and JAR files that exist in the shared-classes directories or the `classes` subdirectory of the installation directory. For example, the files `com.ibm.mq.jar`, `ConfigManagerProxy.jar`, `jplugin2.jar`, and `javacompute.jar` are always visible to the broker, and do not have to be deployed separately.

You cannot read deployed files back from broker execution groups. Therefore, keep a copy of the deployed BAR file, or of the individual files in it.

To add files to a BAR file by using the WebSphere Message Broker Toolkit, complete the following steps.

Procedure

1. Open the BAR file by double-clicking it. The contents of the BAR file are shown in the Broker Archive editor. (If the BAR file is new, this view is empty.)
2. On the Prepare page of the Broker Archive editor, select deployable workspace resources to add to the BAR file.
3. Optional: To include source files, select **Include source files**.
4. Optional: To remove existing content from the BAR file before building the new BAR file, select **Remove contents of Broker Archive before building**.
5. Optional: If you are adding a message flow to a broker archive for a second time, and have used the Manage page to change flow parameters, select **Override configurable property values** to reset configuration settings. If this control is cleared, existing settings are left in place when a flow is replaced.
6. Click **Build broker archive**.

Results

The Manage page lists the files that are now in your BAR file.

You can choose not to display your source files by selecting **Built resources** or **Configurable properties** from the list in the **Filter by** menu.

What to do next

Next:

If you use configurable properties, see “Editing configurable properties” on page 3227.

If you want to have multiple instances of a flow with different values for the configurable properties, see “Adding multiple instances of a message flow to a broker archive” on page 3229.

To make further changes to your BAR file, see “Editing a broker archive file manually.”

When your BAR file is complete, deploy it by following the instructions in “Deploying a broker archive file” on page 3235.

Related concepts:

“Message flow application deployment” on page 3213
Package all the resources in your message flow into a broker archive (BAR) file for deployment.

“Broker archive” on page 3216
The unit of deployment to the broker is the *broker archive* or *BAR* file.

Related tasks:

“Creating a broker archive” on page 3222
Create a separate broker archive (BAR) file for each configuration that you want to deploy to execution groups on your brokers.

“Resolving problems when you use the WebSphere Message Broker Toolkit” on page 3480
Use the advice given here to help you to resolve common problems that might occur you use the WebSphere Message Broker Toolkit.

“Resolving problems when deploying message flows or message sets” on page 3436
Use the advice given here to help you to resolve common problems that can arise when you deploy message flows or message sets.

Related reference:

“JavaCompute node” on page 4514
Use the JavaCompute node to work with messages by using the Java language.

“Broker Archive editor” on page 6794
Use the Broker Archive editor to create and manage broker archive (BAR) files.

Editing a broker archive file manually

Edit resources that you want to change, in an editor of your choice, by exporting a broker archive (BAR) file from the WebSphere Message Broker Toolkit.

Before you begin

Before you start:

If you have not already created a BAR file, create one by following the instructions in “Creating a broker archive” on page 3222.

About this task

To edit a BAR file manually by using the WebSphere Message Broker Toolkit, complete the following steps.

Procedure

1. Export the BAR file.
   a. From the WebSphere Message Broker Toolkit, click **File** > **Export**. The Export wizard opens.
   b. Select the export destination, such as your local file system, and click **Next**.
   c. Select the BAR file that you want to export.
   d. Specify the location to which you are exporting the resources, then click **Finish**.

      The BAR file appears at the destination that you have specified.
2. Extract files from the BAR file.
3. Edit the properties that you want to change in an editor of your choice.
4. Save the file.
5. Import the BAR file back into the WebSphere Message Broker Toolkit for deployment.
   a. From the WebSphere Message Broker Toolkit, click **File** > **Import**. The Import wizard opens.
   b. Select the location type of the BAR file, such as the local file system, and click **Next**.
   c. Specify the location of the BAR file; for example, the directory name.
   d. Select the BAR file to import.
   e. Select the project into which you want to import the BAR file.
   f. Click **Finish**.

What to do next

Next:

Deploy the BAR file by following the instructions in "Deploying a broker archive file" on page 3235.

Related concepts:

"Message flow application deployment" on page 3213
Package all the resources in your message flow into a broker archive (BAR) file for deployment.

"Broker archive" on page 3216
The unit of deployment to the broker is the *broker archive* or *BAR* file.

Related tasks:

"Deploying resources" on page 3234
Deploy message flow applications to execution groups by sending a broker archive (BAR) file to a broker, which unpacks and stores the contents ready for when your message flows are started.

"Resolving problems when you use the WebSphere Message Broker Toolkit" on page 3480
Use the advice given here to help you to resolve common problems that might occur you use the WebSphere Message Broker Toolkit.

Related reference:

“Broker Archive editor” on page 6794
Use the Broker Archive editor to create and manage broker archive (BAR) files.

Editing configurable properties

You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

Before you begin

Before you start:

This topic assumes that you have created a BAR file and added resources to it. For more information, see “Creating a broker archive” on page 3222 and “Adding files to a broker archive” on page 3223.

About this task

You can edit configurable properties in three ways:

- “Using the WebSphere Message Broker Toolkit”
- “Using the WebSphere Message Broker Explorer”
- “Using the **mqsiapplybaroverride** command” on page 3228

Using the WebSphere Message Broker Toolkit:
About this task

To edit properties by using the WebSphere Message Broker Toolkit, complete the following steps.

Procedure

1. Open the BAR file. The resources in your broker archive are listed on the **Manage** page.
2. Optional: You can view the properties that can be configured for your message flows by selecting **Build resources that are configurable** from the **Filter by** list.
3. Expand a message flow to display the nodes that you can configure, then click the node that you want to configure. The values that you can configure for the node are displayed in the **Properties** view.
4. Select the value that you want to edit in the **Properties** view, and enter the new value.

   Repeat these steps for all the properties that you want to configure in your flow.
5. Save your BAR file.

What to do next

Next:

Deploy the BAR file by following the instructions in “Deploying a broker archive file” on page 3235.

Using the WebSphere Message Broker Explorer:
Before you begin

Before you start:

To edit the configurable properties in the WebSphere Message Broker Explorer, you must import a BAR file. To import a BAR file into the WebSphere Message Broker Explorer, see "Importing a broker archive file to the WebSphere Message Broker Explorer" on page 3242.

About this task

To edit properties by using the WebSphere Message Broker Explorercomplete the following steps.

Procedure

1. Open the BAR file. The resources in your broker archive are listed on the **Manage** page.
2. Optional: You can view the properties that can be configured for your message flows by selecting **Build resources that are configurable** from the **Filter by** list.
3. Expand a message flow to display the nodes that you can configure, then click the node that you want to configure. The values that you can configure for the node are displayed in the **Properties** view.
4. Select the value that you want to edit in the **Properties** view, and enter the new value.

   Repeat these steps for all the properties that you want to configure in your flow.
5. Close the BAR file. When you close the BAR file you are prompted to save the file. Click **Yes** to save the changes you made to the BAR file.

What to do next

Next:

Deploy the BAR file by following the instructions in "Deploying a broker archive file" on page 3235.

Using the `mqsiapplybaroverride` command:
About this task

To edit properties by using the **`mqsiapplybaroverride`** command, complete the following steps.

Procedure

1. Open a command window that is configured for your environment.
2. Create a text file with a `.properties` file extension.
3. Enter the command, typed on a single line, specifying the location of your broker archive deployment descriptor (`broker.xml`) and the file that contains the properties to be changed. For examples of how to use the command, see "**`mqsiapplybaroverride`** command" on page 3684.

   A file with a `.bar` extension is created.

What to do next

Next:

Deploy the BAR file by following the instructions in "Deploying a broker archive file" on page 3235.

Related concepts:

"Message flow application deployment" on page 3213
Package all the resources in your message flow into a broker archive (BAR) file for deployment.

"Broker archive" on page 3216
The unit of deployment to the broker is the *broker archive* or *BAR* file.

"Configurable properties of a broker archive" on page 3217
System objects that are defined in message flows can have properties that you can update in the broker archive (BAR) file before deployment.

Related tasks:

"Creating a broker archive" on page 3222
Create a separate broker archive (BAR) file for each configuration that you want to deploy to execution groups on your brokers.

Chapter 11, "Packaging and deploying," on page 3209
Package resources that you create in the WebSphere Message Broker Toolkit, such as message flows, and deploy them to execution groups on brokers.

"Deploying a broker archive file" on page 3235
After you have created and populated a broker archive (BAR) file, deploy the file to an execution group on a broker, so that the file contents can be used in the broker.

"Deploying resources" on page 3234
Deploy message flow applications to execution groups by sending a broker archive (BAR) file to a broker, which unpacks and stores the contents ready for when your message flows are started.

"Resolving problems when you use the WebSphere Message Broker Toolkit" on page 3480
Use the advice given here to help you to resolve common problems that might occur you use the WebSphere Message Broker Toolkit.

"Adding files to a broker archive" on page 3223
To deploy files to an execution group, include them in a broker archive (BAR) file.

Related reference:

"Broker Archive editor" on page 6794
Use the Broker Archive editor to create and manage broker archive (BAR) files.

"**mqsiapplybaroverride** command" on page 3684
Use the **mqsiapplybaroverride** command to replace configurable values in the broker archive (BAR) deployment descriptor with new values that you specify in a properties file.

Adding multiple instances of a message flow to a broker archive

Add more than one instance of a message flow to a broker archive and use configurable properties to customize each instance.

About this task

You can deploy multiple instances of the same message flow with different values for configurable properties. By using the same flow with different configurable property values, you can reuse a flow but customize each instance to your requirements. For example, you could use one flow with a particular value for an output queue, then add a second instance of the same flow and use a configurable property to set a different value for the output queue.

If you want to create multiple instances of a message flow to improve performance, see "Optimizing message flow throughput" on page 587.

Procedure

To deploy multiple instances of a message flow with different values for the configurable properties, complete the following steps:

1. In the Broker Development view, create a copy of the message flow of which you want to deploy multiple instances; see "Copying a message flow by using copy" on page 1435.
2. Add the original message flow and the copy of the message flow to your BAR file; see "Adding files to a broker archive" on page 3223.
3. Open the BAR file. The resources in your broker archive are listed on the **Manage** page.
4. On the **Manage** page, you can now edit the configurable properties for both message flows. For more information, see "Editing configurable properties" on page 3227.

Results

Tip: The names that are assigned in the BAR file are also used on the command line; for example, if you run the **mqsilist** command on the execution group, or if you run the **mqsichangetrace** command for a message flow.

What to do next

Next:

Deploy the BAR file by following the instructions in "Deploying a broker archive file" on page 3235. Both message flows are deployed to the execution group and use the values for the configurable properties that you set in the BAR file.

Related concepts:

"Message flow application deployment" on page 3213
Package all the resources in your message flow into a broker archive (BAR) file for deployment.

"Broker archive" on page 3216
The unit of deployment to the broker is the *broker archive* or *BAR* file.

Related tasks:

"Copying a message flow by using copy" on page 1435
You might find it useful to copy a message flow as a starting point for a new message flow that has similar function. For example, you might want to replace or remove one or two nodes to process messages in a different way.

"Adding files to a broker archive" on page 3223
To deploy files to an execution group, include them in a broker archive (BAR) file.

Chapter 11, "Packaging and deploying," on page 3209
Package resources that you create in the WebSphere Message Broker Toolkit, such as message flows, and deploy them to execution groups on brokers.

"Resolving problems when you use the WebSphere Message Broker Toolkit" on page 3480
Use the advice given here to help you to resolve common problems that might occur you use the WebSphere Message Broker Toolkit.

"Resolving problems when deploying message flows or message sets" on page 3436
Use the advice given here to help you to resolve common problems that can arise when you deploy message flows or message sets.

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

"Deploying resources" on page 3234
Deploy message flow applications to execution groups by sending a broker archive (BAR) file to a broker, which unpacks and stores the contents ready for when your message flows are started.

Related reference:

"**mqsilist** command" on page 3882
Use the **mqsilist** command to list installed brokers and their associated resources.

"**mqsichangetrace** command" on page 3822
Use the **mqsichangetrace** command to set the tracing characteristics for a broker.

"Broker Archive editor" on page 6794
Use the Broker Archive editor to create and manage broker archive (BAR) files.

Configuring a message flow at deployment time with user-defined properties

Use user-defined properties (UDPs) to configure message flows at deployment and run time, without modifying program code. You can give a UDP an initial value when you declare it in your program, or when you use the Message Flow editor to create or modify a message flow.

Before you begin

Before you start:

For an overview of user-defined properties, see "User-defined properties" on page 1147.

For an example of how to code a UDP statement, see "DECLARE statement" on page 5117.

About this task

In ESQL, you can define UDPs at the module or schema level. After a UDP has been defined in the Message Flow editor, you can modify the value before you deploy it.

To configure UDPs, complete the following steps.

Procedure

1. Open the broker archive (BAR) file. The contents of the BAR file are shown in the **Manage** page of the Broker Archive editor. On this page, you can expand a flow to show the individual nodes that it contains.
2. Click the message flow in which you are interested (not the `.cmf` compiled message flow file). The UDPs that are defined in that flow are displayed with their values in the **Properties** view.
3. If the value of the UDP is unsuitable for your current environment or task, change it to an appropriate value. The value of the UDP is set at the flow level, and is the same for all eligible nodes that are contained in the flow. If a subflow includes a UDP that has the same name as a UDP in the main flow, the value of the UDP in the subflow is not changed.
4. Save your BAR file.

Results

Next:

Deploy the message flow by following the instructions in "Deploying a broker archive file" on page 3235.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"User-defined properties" on page 1147
A user-defined property (UDP) is a property that is defined when you construct a message flow by using the Message Flow editor. This property can be used by the ESQL or Java program inside message flow nodes, such as a Compute node.

Related tasks:

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

"Setting message flow user-defined properties at run time in a CMP application" on page 985
Use the CMP API to query, discover, and set message flow user-defined properties dynamically at run time. You can use the CMP API to set properties with a data type of character.

"Deploying a broker archive file" on page 3235
After you have created and populated a broker archive (BAR) file, deploy the file to an execution group on a broker, so that the file contents can be used in the broker.

Related reference:

"DECLARE statement" on page 5117
Use the DECLARE statement to define a variable, the data type of the variable and, optionally, its initial value.

"Broker Archive editor" on page 6794
Use the Broker Archive editor to create and manage broker archive (BAR) files.

"Message Flow editor" on page 6810
The Message Flow editor is the default editor provided by the Broker Application Development perspective for defining a graphical representation of a message flow in the WebSphere Message Broker Toolkit, and for setting properties for individual message flow nodes.

"Configurable message flow properties" on page 4020
When you add a message flow to a broker archive (BAR) file in preparation for deploying it to a broker, you can set additional properties that influence its run time operation. These properties are available for review and update when you select the **Manage and Configure** tab for the broker archive file.

Refreshing the contents of a broker archive

Refresh the contents of a broker archive (BAR) file by rebuilding it in the Broker Archive editor. Alternatively, remove resources from your broker archive and, having made the required changes, add them back again.

Before you begin

Before you start:

This topic assumes that you have created a BAR file and added files to it. For more information, see “Creating a broker archive” on page 3222 and “Adding files to a broker archive” on page 3223.

About this task

You are likely to change resources that you have already added to your BAR file. BAR files that need to be refreshed are shown with an "out-of-sync" icon in the Broker Development view. (When any changes are made to deployable files in the workspace that have previously been built in the broker archive, the BAR file is considered to be inconsistent. The BAR file is also inconsistent if any changes are made to the project to which the files belong.)

To refresh the contents of a BAR file before you deploy it, complete the following steps.

Procedure

1. Open the BAR file. The contents of the BAR file are shown on the **Manage** page of the Broker Archive editor.
2. To refresh all the resources in the BAR file on the **Manage** page, click **Rebuild existing broker arcive entries** .

   A dialog box opens, showing progress. When the operation is complete, click **Details** to see information about what was refreshed, what was not, and why. If the refresh process is successful, you see the same information that is placed in the user log by each of the resource compilers.

   Alternatively, you can refresh the archive contents by right-clicking a BAR file in the Broker Development view and selecting **Build Broker Archive**. The broker archive is rebuilt in the background.

   You can view the user and service logs on the User Log and Service Log pages of the Broker Archive editor. Clear the contents of logs by clicking **Remove** ( ).
3. Optional: To view details about the build of an individual deployable resource on the **Manage** page, right-click the deployable resource and click **Details**.

   The Properties view opens and the Details tab is displayed. The Details tab shows the following details about the deployable resource:

   - The workspace resource, with references to the linked workspace resources (for example, `.msgflow` `.mset`, `.xml`, and `.xslt` files).
   - The status of the last compilation, which shows the user log entry for the last compilation. You can copy text, but you cannot modify it.

What to do next

Next:

Deploy the BAR file by following the instructions in “Deploying a broker archive file” on page 3235.

Related concepts:

“Message flow application deployment” on page 3213
Package all the resources in your message flow into a broker archive (BAR) file for deployment.

“Broker archive” on page 3216
The unit of deployment to the broker is the *broker archive* or *BAR* file.

Related tasks:

“Deploying resources”
Deploy message flow applications to execution groups by sending a broker archive (BAR) file to a broker, which unpacks and stores the contents ready for when your message flows are started.

“Resolving problems when you use the WebSphere Message Broker Toolkit” on page 3480
Use the advice given here to help you to resolve common problems that might occur you use the WebSphere Message Broker Toolkit.

“Resolving problems when deploying message flows or message sets” on page 3436
Use the advice given here to help you to resolve common problems that can arise when you deploy message flows or message sets.

Related reference:

“Broker Archive editor” on page 6794
Use the Broker Archive editor to create and manage broker archive (BAR) files.

Deploying resources

Deploy message flow applications to execution groups by sending a broker archive (BAR) file to a broker, which unpacks and stores the contents ready for when your message flows are started.

Before you begin

Before you start:

Complete the following steps:

- Create and start a broker. You must also start a WebSphere MQ listener for the associated queue manager.
- In the WebSphere Message Broker Toolkit, you must connect to a broker, and create an execution group.
- Create a BAR file that contains the resources that you want to deploy; see “Packaging resources” on page 3221.

About this task

The mode in which your broker is working can affect the number of execution groups and message flows that you can deploy, and the types of node that you can use. See “Restrictions that apply in each operation mode” on page 3657.

If you are working with a single message flow, and want to deploy and test it quickly, you can deploy just that flow. Drag the message flow onto the execution group to which you want to deploy it. A BAR file is generated automatically, and

deployed to the broker. You can then run and test the message flow, change the flow, and deploy it again with a minimum of steps involved.

In most cases, you are working with one or more message flows, one or more message sets, and other objects that you want to deploy with the message flows. In these cases, you can deploy these resources together by packaging them in a BAR file yourself, and deploying just that file. For instructions about how to deploy a BAR file, see "Deploying a broker archive file."

If you are deploying a message flow that uses WebSphere Adapters, see "Deploying a message flow that uses WebSphere Adapters" on page 3240. To deploy by using the WebSphere Message Broker Explorer, see "Importing a broker archive file to the WebSphere Message Broker Explorer" on page 3242.

If your message flows include user-defined nodes, you must also distribute the compiled C or Java code for each node to every broker that uses those message flows. For more details, see "Developing user-defined extensions" on page 2970.

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

"Message flow application deployment" on page 3213
Package all the resources in your message flow into a broker archive (BAR) file for deployment.

Related tasks:

"Creating a broker" on page 611
You can create brokers on every platform that is supported by WebSphere Message Broker. The broker runs as a 64-bit application on all platforms except Linux on x86 and Windows on x86.

"Starting and stopping a broker" on page 921
Run the appropriate command to start or stop a broker.

"Connecting to a local broker using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 901
To administer a broker by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer, you must be connected to the broker.

"Creating an execution group" on page 936
Create execution groups by using WebSphere Message Broker Toolkit, the WebSphere Message Broker Explorer, or the command line.

Chapter 11, "Packaging and deploying," on page 3209
Package resources that you create in the WebSphere Message Broker Toolkit, such as message flows, and deploy them to execution groups on brokers.

Deploying a broker archive file

After you have created and populated a broker archive (BAR) file, deploy the file to an execution group on a broker, so that the file contents can be used in the broker.

Before you begin

Before you start:

This topic assumes that you have created a BAR file. For more information, see "Creating a broker archive" on page 3222.

About this task

Choose one of the following methods to deploy a BAR file:

- "Using the WebSphere Message Broker Toolkit"
- "Using the WebSphere Message Broker Explorer" on page 3237
- "Using the **mqsideploy** command" on page 3238
- "Using the CMP API" on page 3238

If you change a BAR file, and want to propagate those changes to one or more brokers, you can redeploy the updated BAR file by following the instructions in "Redeploying a BAR file" on page 3239.

The mode in which your broker is working can affect the number of execution groups and message flows that you can deploy, and the types of node that you can use. See "Restrictions that apply in each operation mode" on page 3657.

Using the WebSphere Message Broker Toolkit

About this task

To deploy a BAR file by using the WebSphere Message Broker Toolkit, complete the following steps. You can deploy to only one execution group at a time.

Procedure

1. Optional: Typically, an incremental BAR file deployment is performed. To perform a complete BAR file deployment, right-click the target execution group in the Brokers view and click **Delete** > **All Flows and Resources**. Wait for the operation to complete before continuing.

   Do not click **Delete** > **All Flows and Resources** if you want to refresh one or more of the child processes with the contents of the BAR file. For an explanation of the difference between a complete and an incremental BAR file deployment, see "Message flow application deployment" on page 3213.
2. Deploy a BAR file or message flow to an execution group by using one of the following methods:
   - Drag a message flow or BAR file onto your target execution group, shown in the Brokers view.
   - Right-click the BAR file, and click **Deploy**. The Deploy dialog box opens, listing the execution groups to which the WebSphere Message Broker Toolkit is connected.

     Select an execution group, and click **OK**.
   - Right-click the execution group, and click **Deploy**. The Resources dialog box opens. You can choose to deploy resources from the workspace, or BAR files from the file system.
     - Specify the type of resource to deploy by selecting the appropriate radio button. The available resources for that category are listed.
     - Select the resource that you want to deploy.
     - Click **OK**.
3. If you have not saved the BAR file since you last edited it, you are asked whether you want to save the file before deploying. If you click **Cancel**, the BAR file is not saved and deployment does not take place.

Results

The BAR file is transferred to the broker, which deploys the file contents (for example, message flows and message sets) to the execution group. In the Brokers view, the deployed resources are added to the appropriate execution group.

Using the WebSphere Message Broker Explorer

Before you begin

Before you start:

Import a BAR file into the WebSphere Message Broker Explorer by following the instructions in "Importing a broker archive file to the WebSphere Message Broker Explorer" on page 3242.

About this task

To deploy a BAR file by using the WebSphere Message Broker Explorer, complete the following steps.

Procedure

1. Expand the Broker Resources folder and select the folder that contains your BAR files.
2. Optional: Typically, an incremental BAR file deployment is performed. To perform a complete BAR file deployment, right-click the target execution group in the Brokers view and click **Delete** > **All Flows and Resources**. Wait for the operation to complete before continuing.

   Do not **Delete** > **All Flows and Resources** if you want to refresh one or more of the child processes with the contents of the BAR file. For an explanation of the difference between a complete and an incremental BAR file deployment, see "Message flow application deployment" on page 3213.
3. Deploy the BAR file to an execution group by using one of the following methods:
   - Drag the file onto your target execution group in the Navigator view. You can drag a BAR file from either your WebSphere Message Broker Explorer workspace, or from your file system.
   - Right-click the BAR file and click **Deploy file**. A dialog box opens, listing the execution groups to which the WebSphere Message Broker Explorer is connected.

     Select an execution group, then click **OK**.

   If you use the **Deploy file** method, you can select (and deploy to) multiple execution groups at a time.
4. If you have not saved the BAR file since you last edited it, you are asked whether you want to save the file before deploying. If you click **Cancel**, the BAR file is not saved and deployment does not take place.

Results

The BAR file is transferred to the broker, which deploys the file contents (for example, message flows and message sets) to the execution group. Expand the broker in the Navigator view to see the assigned message flows and message sets added to the appropriate execution group.

Using the mqsideploy command
About this task

To deploy a BAR file by using the **mqsideploy** command, complete the following steps.

Procedure

1. Open a command window that is configured for your environment.
2. Enter the appropriate command for your operating system and configuration, by using the following examples as a guide.

   On distributed systems:

   ```
   mqsideploy -i ipAddress -p port -q qmgr -e egroup -a barfile
   ```

   The command performs an incremental deployment. Add the **–m** parameter to perform a complete BAR file deployment.

   The **-i** (IP address), **-p** (port), and **-q** (queue manager) parameters represent the connection details for the queue manager that is associated with the broker. If you have created the broker on the computer on which you run this command, you can specify the broker name instead.

   You must also specify the **-e** (execution group name), and **-a** (BAR file name) parameters.

   On z/OS:

   ```
   /f MQ01BRK,dp e=egroup a=barfile
   ```

   The command performs an incremental deployment. Add the **m**=yes parameter to perform a complete BAR file deployment.

   In the example, *MQ01BRK* is the name of the broker. You must also specify the names of the execution group and the BAR file (the **e=** and **a=** parameters).

Results

The command reports when responses are received from the broker. If the command completes successfully, it returns 0 (zero).

Using the CMP API
About this task

To deploy by using the CMP API, use the deploy method of the ExecutionGroupProxy class.

The following code shows how an application can perform an incremental deployment:

```
import com.ibm.broker.config.proxy.*;
public class DeployBAR {

  public static void main(String[] args) {
    BrokerConnectionParameters bcp =
       new MQBrokerConnectionParameters("localhost", 2414, "MB7QMGR");
    try {
      BrokerProxy b = BrokerProxy.getInstance(bcp);
      ExecutionGroupProxy eg = b.getExecutionGroupByName("default");
      DeployResult dr = eg.deploy("MyBAR.bar", true, 30000);
      System.out.println("Result = "+dr.getCompletionCode());
```

```
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

By default, the deploy method performs an incremental deployment. To perform a complete deployment, use a variant of the method that includes a `false` value for the Boolean isIncremental parameter. For example, e.deploy("deploy.bar",false,0). Set this parameter to `true` to request an incremental deployment.

Redeploying a BAR file

About this task

You might change a BAR file and want to propagate those changes to one or more brokers. If so, you can redeploy the updated BAR file to one or more execution groups by using one of the deployment methods described previously. You do not have to stop the message flows that you deployed previously; all resources in the execution group or groups that are in the redeployed BAR file are replaced and new resources are applied.

If your updates to the BAR file include the deletion of resources, a redeployment does not result in their deletion from the broker. For example, assume that your BAR file contains message flows F1, F2, and F3. Update the file by removing F2 and adding message flow F4. If you redeploy the BAR file, all four flows are available in the execution group when the redeployment has completed. F1 and F3 are replaced by the contents of the redeployed BAR file.

To clear previously deployed resources from the execution group before you redeploy (for example, if you are deleting resources), use one of the methods described previously.

- To use the WebSphere Message Broker Toolkit, follow the instructions for a complete deployment, making sure that you select **Delete** > **All Flows and Resources** before deploying.
- To use the WebSphere Message Broker Explorer, follow the instructions for a complete deployment, making sure that you select **Delete All Flows and Resources** before deploying.
- To use the **mqsideploy** command, follow the instructions, making sure that you add the **-m** parameter to perform a complete BAR file deployment.
- To use the CMP API, follow the instructions for a complete deployment.

If your message flows are not transactional, stop the message flows before you redeploy to be sure that all the applications complete cleanly and are in a known and consistent state. You can stop individual message flows, execution groups, or brokers.

If your message flows are transactional, the processing logic that handles commitment or rollback ensures that resource integrity and consistency are maintained.

What to do next

Next:

Check the results of the redeployment by following the instructions in “Checking the results of deployment” on page 3243.

Related concepts:

"Message flow application deployment" on page 3213
Package all the resources in your message flow into a broker archive (BAR) file for deployment.

"Broker archive" on page 3216
The unit of deployment to the broker is the *broker archive* or *BAR* file.

"The Administration API for WebSphere Message Broker" on page 54
The Administration API for WebSphere Message Broker is a programming interface that your applications can use to control brokers and their resources through a remote interface.

Related tasks:

"Setting up a command environment" on page 213
After you have installed the product on one of the distributed systems, you must initialize the environment before you can use a runtime component or command.

"Checking the results of deployment" on page 3243
After you have made a deployment, check that the operation has completed successfully.

"Resolving problems when deploying message flows or message sets" on page 3436
Use the advice given here to help you to resolve common problems that can arise when you deploy message flows or message sets.

Related reference:

"Broker Archive editor" on page 6794
Use the Broker Archive editor to create and manage broker archive (BAR) files.

"**mqsideploy** command" on page 3872
Use the **mqsideploy** command to make a deployment request to the broker.

Related information:

Administration API for WebSphere Message Broker (CMP API)

Deploying a message flow that uses WebSphere Adapters

Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Before you begin

Before you start:

- Read "WebSphere Adapters nodes" on page 1914.
- Complete the steps in "Preparing the environment for WebSphere Adapters nodes" on page 717
- Complete the steps in "Connecting to an EIS by using the Adapter Connection wizard" on page 2037.

About this task

To deploy the message flow successfully, you must deploy the WebSphere Adapters component, either on its own or in the same BAR file as your message flow. If the WebSphere Adapters component is not available, deployment of the message flow fails. The following list includes the file extensions of the resources that you deploy:

- `.msgflow` (the message flow)
- `.inadapter` (the inbound WebSphere Adapters component)

- `.outadapter` (the outbound WebSphere Adapters component)
- `.xsdzip` (the message set)
- `.libzip` (the library)

The mode in which your broker is working can affect the number of execution groups and message flows that you can deploy, and the types of node that you can use. See "Restrictions that apply in each operation mode" on page 3657.

Procedure

1. For details of the steps that you must complete before you can deploy a message flow, see "Deploying resources" on page 3234.
2. Add the message flow to the BAR file. (For a description of how to add files to a BAR file, see "Adding files to a broker archive" on page 3223.) When you add a message flow that contains one or more WebSphere Adapters nodes to a BAR file, a dialog box opens so that you can identify the following resources:
   - One or more WebSphere Adapters components to be used by the WebSphere Adapters nodes
   - One or more message sets that contain an XSD for the business objects that are used by the WebSphere Adapters nodes
3. When you have added the message flow, WebSphere Adapters components, and message set, deploy the BAR file.

Related concepts:

"Packaging and deployment overview" on page 3210
Deployment is the process of transferring data to an execution group on a broker so that it can take effect in the broker. Message flows and associated resources are packaged in broker archive (BAR) files for deployment.

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"SAP adapter scalability and performance" on page 1949
You can improve performance by configuring the number of listeners on the adapter and the number of additional instances on the message flow to prevent delays when processing synchronous calls from SAP.

Related tasks:

"Deploying resources" on page 3234
Deploy message flow applications to execution groups by sending a broker archive (BAR) file to a broker, which unpacks and stores the contents ready for when your message flows are started.

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Tuning the SAP adapter for scalability and performance" on page 3278
You can configure the number of listeners on the adapter and the number of additional instances on the message flow to prevent delays when processing

synchronous calls from SAP.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

Importing a broker archive file to the WebSphere Message Broker Explorer

Before you can deploy broker resources to your brokers by using the WebSphere Message Broker Explorer, you must first import a broker archive file into a Broker Archive Folder.

Before you begin

You must have a broker archive file created and available before you can import it into the WebSphere Message Broker Explorer.

About this task

To import a broker archive file:

Procedure

1. Open the WebSphere Message Broker Explorer.
2. You can import a broker archive file into an existing Broker Archive Folder, or create a new one. To create a new Broker Archive Folder:
   a. Right-click Broker Archive Files, and click **New** > **Broker Archive Folder**. The Broker Archive Folder wizard is displayed.
   b. Enter a Broker Archive Folder name, and click **Finish**.

   A new Broker Archive Folder is created with the name you specified in the Broker Archive Folder wizard.
3. Right-click your Broker Archive Folder, and select **Import Broker Resources**. The **Import** wizard is displayed.
4. Click **Browse** next to the **From directory** field to select the directory containing your broker archive.
5. Select the broker archive files that you want to import.
6. Click **Browse** next to the **Info folder** field to select your Broker Archive Folder.
7. Click **Finish**. The selected broker archive files are imported into the selected Broker Archive Folder.

Results

You can now deploy your broker archive to your broker. A copy of the imported broker archive file is created, and placed in the WebSphere Message Broker Explorer workspace. If you change the broker archive file outside of the WebSphere Message Broker Explorer the changes are not made to the imported broker archive file. The default location on Windows for the WebSphere Message Broker Explorer workspace is `C:\Documents and Settings\<userid>\Application Data\IBM\MQ Explorer\<server project>\`. Any subsequent changes you make to the broker archive file within the WebSphere Message Broker Explorer are made to the version in the WebSphere Message Broker Explorer workspace.

Related tasks:

"Creating a broker using the WebSphere Message Broker Explorer" on page 618
On Linux on x86 or Windows, you can create brokers by using the WebSphere Message Broker Explorer.

"Starting a local broker using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 925
You can start your local brokers by using the WebSphere Message Broker Toolkit or the WebSphere Message Broker Explorer.

"Connecting to a local broker using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 901
To administer a broker by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer, you must be connected to the broker.

"Creating an execution group using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 937
Use the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer to create execution groups on your broker.

Chapter 8, "Administering brokers and broker resources," on page 899
Administering brokers and associated broker resources includes the tasks that you perform frequently to activate and manage those resources. Choose the method you prefer to administer your brokers and associated resources.

Checking the results of deployment

After you have made a deployment, check that the operation has completed successfully.

About this task

You can check the results of a deployment in the following ways:

- "Using the WebSphere Message Broker Toolkit"
- "Using the WebSphere Message Broker Explorer" on page 3244
- Using the **mqsideploy** command
- Using the CMP API

Also, check the system log on the target system where the broker was deployed to make sure that the broker has not reported any errors.

Using the WebSphere Message Broker Toolkit

About this task

Follow these steps to check a deployment using the WebSphere Message Broker Toolkit:

Procedure

1. In the Broker Application Development perspective, open the Deployment Log view:
   a. Click **Window** > **Show View** > **Other**.
   b. In the Show View window expand **Broker Runtime**, and click **Deployment Log**.
2. View the messages relating to each deployment in the Deployment Log view.

Results

The Deployment Log is only updated after a deployment has been completely processed by the broker. If a deployment fails, the reasons for the failure are shown here.

Using the WebSphere Message Broker Explorer

About this task

Follow these steps to check a deployment using the WebSphere Message Broker Explorer:

Procedure

1. Select the broker with which you want to work in the Navigator view.
2. View the messages in the Administration Log view.

Results

The Administration Log shows all recent configuration change attempts made to the broker and, where possible, the user that requested it.

Using the `mqsideploy` command

About this task

If you use the **mqsideploy** command to deploy, the command displays the results of the deployment. It also returns a numeric completion code value to indicate the outcome. If the deployment completes successfully, the command returns 0. For details of other values that you might see returned, see "**mqsideploy** command" on page 3872.

Using the CMP API

About this task

Code your application to test the results of the deploy actions that it takes. You can use code like the following snippet:

```
DeployResult dr = eg.deploy("MyBAR.bar", true, 30000);
System.out.println("Overall result = "+dr.getCompletionCode());

// Display log messages
Enumeration logEntries = dr.getLogEntries();
while (logEntries.hasMoreElements()) {
  LogEntry le = (LogEntry)logEntries.nextElement();
  System.out.println("General message: " + le.getDetail());
}
```

The deploy method blocks other processes until the broker has responded to the deployment request. When the method returns, the `DeployResult` object represents the outcome of the deployment at the time when the method returned; the object is not updated by the CMP API.

If the deployment message could not be sent to the broker, a `ConfigManagerProxyLoggedException` exception is thrown at the time of the deployment. If the broker receives the deployment message, log messages for the overall deployment are displayed, followed by completion codes specific to each

broker that is affected by the deployment. The completion code, shown in the following table, is one of the static instances from the `CompletionCodeType` class.

Completion code	Description
pending	The deployment is held in a batch and is not sent until you call BrokerProxy.sendUpdates().
submitted	The deploy message was sent to the broker, but no response was received before the timeout period expired.
success	The broker has successfully completed the deployment.
failure	The broker has generated one or more errors during deployment. You can call the getLogEntries() method of the `DeployResult` class to get more information about the deployment failure. This method returns an enumeration of available `LogEntry` objects.

Related concepts:

"Packaging and deployment overview" on page 3210
Deployment is the process of transferring data to an execution group on a broker so that it can take effect in the broker. Message flows and associated resources are packaged in broker archive (BAR) files for deployment.

"The Administration API for WebSphere Message Broker" on page 54
The Administration API for WebSphere Message Broker is a programming interface that your applications can use to control brokers and their resources through a remote interface.

Related tasks:

Chapter 11, "Packaging and deploying," on page 3209
Package resources that you create in the WebSphere Message Broker Toolkit, such as message flows, and deploy them to execution groups on brokers.

"Viewing Administration log information" on page 1007
You can view Administration log information by using either the WebSphere Message Broker Explorer, or the CMP API.

"Administration API (CMP) trace" on page 3554
Enable or disable service trace for the Administration API for WebSphere Message Broker (also known as the CMP API).

"Resolving problems when you use the WebSphere Message Broker Toolkit" on page 3480
Use the advice given here to help you to resolve common problems that might occur you use the WebSphere Message Broker Toolkit.

"Resolving problems when deploying message flows or message sets" on page 3436
Use the advice given here to help you to resolve common problems that can arise when you deploy message flows or message sets.

"Resolving problems when using publish/subscribe" on page 3501
Use the advice given here to help you to resolve common problems that can occur when you run publish/subscribe applications.

"Resolving problems when developing Administration API applications" on page 3510
Use the advice given here to help you to resolve problems that can arise when developing Administration API for WebSphere Message Broker (also known as the CMP API) applications.

Related reference:

"**mqsideploy** command" on page 3872
Use the **mqsideploy** command to make a deployment request to the broker.

Related information:

Administration API for WebSphere Message Broker (CMP API)

Renaming objects that are deployed to execution groups

You cannot rename an object while it is still deployed to an execution group. You must change it in the broker archive, then redeploy the broker archive (BAR) file.

About this task

To rename a deployed object, complete the following steps.

Procedure

1. Remove the object from the execution group. For more details, see "Removing a deployed object from an execution group."
2. Rename the object.
3. Refresh the contents of the broker archive. For more details, see "Refreshing the contents of a broker archive" on page 3233.
4. Redploy the BAR file. For more details, see "Deploying a broker archive file" on page 3235.

Related concepts:

"Packaging and deployment overview" on page 3210
Deployment is the process of transferring data to an execution group on a broker so that it can take effect in the broker. Message flows and associated resources are packaged in broker archive (BAR) files for deployment.

Related tasks:

"Deploying resources" on page 3234
Deploy message flow applications to execution groups by sending a broker archive (BAR) file to a broker, which unpacks and stores the contents ready for when your message flows are started.

"Removing a deployed object from an execution group"
You might want to remove deployed objects from an execution group if, for example, you want to rename them.

"Refreshing the contents of a broker archive" on page 3233
Refresh the contents of a broker archive (BAR) file by rebuilding it in the Broker Archive editor. Alternatively, remove resources from your broker archive and, having made the required changes, add them back again.

"Deploying a broker archive file" on page 3235
After you have created and populated a broker archive (BAR) file, deploy the file to an execution group on a broker, so that the file contents can be used in the broker.

"Resolving problems when deploying message flows or message sets" on page 3436
Use the advice given here to help you to resolve common problems that can arise when you deploy message flows or message sets.

Removing a deployed object from an execution group

You might want to remove deployed objects from an execution group if, for example, you want to rename them.

Before you begin

Before you start:

Stop all message flows in the execution group. For more details, see "Stopping an execution group using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 944.

About this task

You can remove deployed objects from an execution group in the following ways:

- "Using the WebSphere Message Broker Explorer"
- "Using the **mqsideploy** command"
- "Using the CMP API" on page 3248

Using the WebSphere Message Broker Toolkit

About this task

To remove an object from an execution group by using the WebSphere Message Broker Toolkit, complete the following steps.

Procedure

1. In the Brokers view, right-click the object that you want to remove.
2. Click **Delete**, then **OK** to confirm.

Results

The request is sent to the broker, and a synchronous response is sent back.

Using the WebSphere Message Broker Explorer

About this task

To remove an object from an execution group by using the WebSphere Message Broker Explorer, complete the following steps.

Procedure

1. In the Navigator view, expand the broker and execution group with which you want to work.
2. Right-click the object that you want to remove.
3. Click **Delete**, then **OK** to confirm.

Results

The request is sent to the broker, and a synchronous response is sent back.

Using the `mqsideploy` command

About this task

To remove an object from an execution group by using the **mqsideploy** command, complete the following steps.

Procedure

1. Open a command window that is configured for your environment.
2. Enter the appropriate command for your operating system and configuration, using the following examples as a guide.

   On distributed systems:

   ```
   mqsideploy -i ipAddress -p port -q qmgr –e egroup
              –d file1.cmf:file2.cmf:file3.dictionary:file4.xml
   ```

   The **-i** (IP address), **-p** (port), and **-q** (queue manager) parameters represent the connection details for the queue manager associated with the broker. If you have created the broker on the computer on which you run this command, you can specify the broker name instead.

   On z/OS:

   ```
   /f MQ01BRK1,dp e=egroup d=file1.cmf:file2.cmf:file3.dictionary:file4.xml
   ```

   where *MQ01BRK1* is the name of the broker.

   The **-d** parameter (**d=** on z/OS) is a colon-separated list of files that you want to remove from the named execution group. When you run the command, the deployed objects (`file1.cmf, file2.cmf, file3.dictionary, file4.xml`) are removed from the specified execution group.

   Optionally, specify the **-m** parameter (**m=** on z/OS) to clear the contents of the execution group. This option tells the execution group to completely clear all existing data before the new BAR file is deployed.

Results

The command reports when responses are received from the broker. If the command completes successfully, it returns zero (0).

Using the CMP API

About this task

To remove deployed objects from an execution group, get a handle to the relevant `ExecutionGroupProxy` object, then run the deleteDeployedObjectsByName method. Use the following code as an example.

```
import com.ibm.broker.config.proxy.*;

public class DeleteDeployedObjects {
  public static void main(String[] args) {
    BrokerConnectionParameters bcp =
            new MQBrokerConnectionParameters
                      ("localhost", 1414, "QM1");
    try {
      BrokerProxy b =
            BrokerProxy.getInstance(bcp);
      ExecutionGroupProxy e =
            b.getExecutionGroupByName("default");
      e.deleteDeployedObjectsByName(
            new String[] { "file1.cmf",
                           "file2.cmf",
                           "file3.dictionary",
                           "file4.xml" }, 0);
    }
    catch (ConfigManagerProxyException e) {
```

```
      e.printStackTrace();
    }
  }
}
```

What to do next

Next:

If you have removed one or more message flows, you can now remove the resource files that are associated with those message flows; for example, JAR files.

Related concepts:

"Packaging and deployment overview" on page 3210
Deployment is the process of transferring data to an execution group on a broker so that it can take effect in the broker. Message flows and associated resources are packaged in broker archive (BAR) files for deployment.

"Execution groups" on page 53
An execution group is a named grouping of message flows that have been assigned to a broker. The broker enforces a degree of isolation between message flows in distinct execution groups by ensuring that they run in separate address spaces, or as unique processes.

"The Administration API for WebSphere Message Broker" on page 54
The Administration API for WebSphere Message Broker is a programming interface that your applications can use to control brokers and their resources through a remote interface.

Related tasks:

"Setting up a command environment" on page 213
After you have installed the product on one of the distributed systems, you must initialize the environment before you can use a runtime component or command.

Chapter 11, "Packaging and deploying," on page 3209
Package resources that you create in the WebSphere Message Broker Toolkit, such as message flows, and deploy them to execution groups on brokers.

"Deploying resources to a broker from a CMP application" on page 982
Deploy BAR files to the brokers in your broker network from a CMP application.

"Resolving problems when you use the WebSphere Message Broker Toolkit" on page 3480
Use the advice given here to help you to resolve common problems that might occur you use the WebSphere Message Broker Toolkit.

"Resolving problems when deploying message flows or message sets" on page 3436
Use the advice given here to help you to resolve common problems that can arise when you deploy message flows or message sets.

"Resolving problems when developing Administration API applications" on page 3510
Use the advice given here to help you to resolve problems that can arise when developing Administration API for WebSphere Message Broker (also known as the CMP API) applications.

Related reference:

"**mqsideploy** command" on page 3872
Use the **mqsideploy** command to make a deployment request to the broker.

Chapter 12. Performance and monitoring

You can change various aspects of your broker configuration to tune brokers and message flows, and monitor message flows and publish/subscribe applications.

About this task

Performance

The way in which you configure your broker environment can affect the performance of your applications. For more information about these options, see "Performance."

Monitoring

You can configure your broker or message flows to generate data and statistics that you can use to assess behavior and performance. For more information about these options, see "Business-level monitoring" on page 3319.

Related tasks:

Chapter 6, "Configuring brokers for development environments," on page 563
Set up application development environments on Linux on x86 or Windows to create, test, and deploy message flows and associated resources.

Chapter 7, "Configuring brokers for test and production environments," on page 579
Create one or more brokers on one or more computers, and configure them on your test and production systems to process messages that contain your business data.

Chapter 8, "Administering brokers and broker resources," on page 899
Administering brokers and associated broker resources includes the tasks that you perform frequently to activate and manage those resources. Choose the method you prefer to administer your brokers and associated resources.

Performance

You can change various aspects of your broker configuration to tune brokers and message flows.

About this task

The way in which you configure your broker environment can affect the performance of your applications. Review the information in "Considering performance in the broker environment" on page 586, and decide what actions you can take to change performance characteristics.

The following sections provide further information about how you can collect statistics and monitor performance:

- "Tuning the broker" on page 3254
- Tuning message flow performance
- "Tuning the SAP adapter for scalability and performance" on page 3278
- "Monitoring message flow performance" on page 3279
- "Monitoring resource performance" on page 3305

- "Subscribing to statistics reports" on page 3317

Related tasks:

Chapter 6, "Configuring brokers for development environments," on page 563
Set up application development environments on Linux on x86 or Windows to create, test, and deploy message flows and associated resources.

Chapter 7, "Configuring brokers for test and production environments," on page 579
Create one or more brokers on one or more computers, and configure them on your test and production systems to process messages that contain your business data.

Chapter 8, "Administering brokers and broker resources," on page 899
Administering brokers and associated broker resources includes the tasks that you perform frequently to activate and manage those resources. Choose the method you prefer to administer your brokers and associated resources.

Related reference:

"Performance and monitoring" on page 6723
Use the reference information in this section to accomplish the performance and monitoring tasks that address your business needs.

Considering performance in the broker environment

When you design your broker environment, and the resources associated with the brokers, decisions that you make can affect the performance of your brokers and applications.

About this task

Message flows

A message flow includes an input node that receives a message from an application over a particular protocol; for example, WebSphere MQ. The message must be parsed by the input node, although some parsers support partial parsing which might reduce processing, because only the parts of the message that are referenced are parsed. Other processing in a message flow that might affect performance are the amount, efficiency, and complexity of ESQL, access to databases, and how many message tree copies are made.

You must consider how you split your business logic; how much work should the application do, and how much should the message flow do? Every interaction between an application and a message flow involves I/O and message parsing, and therefore adds to processing time. Design your message flows, and design or restructure you applications, to minimize these interactions.

For more information about these factors, see "Optimizing message flow response times" on page 3264.

Messages and message models

The type, format, and size of the messages that are processed can have a significant effect on the performance of a message flow. For example, if you process persistent messages, they have to be stored for safekeeping.

You might need to process messages with a well-defined structure; if so, you can create MRM models for your messages. If you do not plan to interrogate the structure, you can work with undefined messages, such as BLOB messages.

If you are working in XML, be aware that it can be verbose, and therefore produce large messages, but XML message content is easier to understand than other formats, such as CWF. Field size and order might be important; these factors can be included in your MRM model.

For more information about these factors, see “Optimizing message flow response times” on page 3264 and Performance considerations for regular expressions in TDS messages.

Broker configuration

You can create and configure one or more brokers, on one or more computers, and for each broker you can create multiple execution groups, and multiple message flows. Your configuration decisions can influence message flow performance, and how efficiently messages can be processed.

For more information about these factors, see “Tuning the broker” on page 3254, “Optimizing message flow throughput” on page 587.

All these factors are examined in more detail in the Designing for Performance SupportPac (IP04).

For a description of common performance scenarios, review “Resolving problems with performance” on page 3504.

For further articles about WebSphere Message Broker and performance, review these sources:

- The Business Integration Zone on developerWorks. This site has a search facility; enter "performance" and review the links that are returned.
- The developerWorks article on message flow performance.
- The developerWorks article on monitoring resource use.

Related tasks:

“Optimizing message flow throughput” on page 587
Each message flow that you design must provide a complete set of processing for messages received from a certain source. This design might result in very complex message flows that include large numbers of nodes that can cause a performance overhead, and might create potential bottlenecks. You can increase the number of message flows that process your messages to provide the opportunity for parallel processing and therefore improved throughput.

“Optimizing message flow response times” on page 3264
You can use different solutions to improve message flow response times.

“Setting configuration timeout values” on page 3258
Change timeout values that affect configuration tasks in the broker.

“Tuning the broker” on page 3254
You can complete several tasks that enable you to tune different aspects of the broker performance.

“Resolving problems with performance” on page 3504
Use the advice given here to help you to resolve common problems with performance.

Related reference:

“Performance considerations when using regular expressions” on page 6309
Take care when specifying regular expressions: some forms of regular expression can involve a large amount of work to find the best match, which might degrade performance.

Tuning the broker

You can complete several tasks that enable you to tune different aspects of the broker performance.

Before you begin

Before you start:

Ensure that the following requirements are met:

- Your user ID has the correct authorizations to perform the task. Refer to "Security requirements for administrative tasks" on page 3644.

About this task

Select the tasks that are relevant to your enterprise:

Procedure

- "Setting the JVM heap size"
- "Increasing the stack size on Windows, Linux, and UNIX systems" on page 3255
- "Increasing the stack size on z/OS" on page 3256
- "Tuning the HEAP settings on z/OS" on page 3257
- "Setting configuration timeout values" on page 3258

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related tasks:

"Configuring brokers" on page 610
Create and configure the brokers that you want on the operating system of your choice.

Related reference:

"Security requirements for administrative tasks" on page 3644
You can configure access permissions to govern which users and groups can manipulate objects in the broker network. Security requirements for administrative tasks depend on the platform that you use.

Setting the JVM heap size

When you start an execution group, it creates a Java virtual machine (JVM) for executing a Java user-defined node.

About this task

You can pass parameters to the JVM to set the minimum and maximum heap sizes; the default maximum heap size is 256 MB. To give more capacity to a message flow that is going to process large messages, reduce the minimum JVM heap size to allow the main memory heap to occupy more address space.

Increase the maximum heap size only if you use Java intensively with, for example, user-defined nodes.

Use caution when you set the maximum heap size, because the Java Runtime Environment takes the values for its initial, maximum, and current heap sizes to calculate how frequently it drives garbage collection. A large maximum heap size

drives garbage collection less frequently. If garbage collection is driven less frequently, the heap size associated with the execution group continues to grow.

Use the information on JVM parameter values on the **mqsichangeproperties** command to set the heap size that you require.

Related concepts:

"JVM heap sizing" on page 3269
The Java virtual machine (JVM) heap is an independent memory allocation that can reduce the capacity of the main memory heap.

"Stack storage" on page 3268
Depending on the design of your message flow, you might need to increase the stack size.

Related tasks:

"Optimizing message flow response times" on page 3264
You can use different solutions to improve message flow response times.

Related reference:

"Message flows" on page 4015
Use the reference information in this section to develop your message flows and related resources.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"JVM parameter values" on page 3813
Select the objects and properties associated with the Java Virtual Machine (JVM) that you want to change.

Increasing the stack size on Windows, Linux, and UNIX systems

Increase the stack size on Windows, Linux, and UNIX systems by setting the **MQSI_THREAD_STACK_SIZE** environment variable to an appropriate value.

About this task

When you restart brokers that are running on the system, they use the new value.

The value, in bytes, of **MQSI_THREAD_STACK_SIZE** that you set is used for every thread that is created in a DataFlowEngine process. If the execution group has many message flows assigned to it, and you set a large value for **MQSI_THREAD_STACK_SIZE**, the DataFlowEngine process needs a large amount of storage for the stacks.

Set this environment variable to at least 48 KB. The default values are:

Linux and UNIX
1 MB

AIX 2 MB

z/OS 50 KB

Windows
Determined by the operating system.

Related concepts:

"Stack storage" on page 3268
Depending on the design of your message flow, you might need to increase the stack size.

Related tasks:

"Increasing the stack size on z/OS"
Change the stack size on z/OS by altering or adding the Language Environment (LE) **_CEE_RUNOPTS** environment variable. This can be done for all execution groups defined to a broker, or a specific execution group.

Increasing the stack size on z/OS

Change the stack size on z/OS by altering or adding the Language Environment (LE) **_CEE_RUNOPTS** environment variable. This can be done for all execution groups defined to a broker, or a specific execution group.

Before you begin

Broker components on z/OS are compiled using the XPLINKage (extra performance linkage), which adds optimization to the runtime code. However, if the initial stack size is not large enough, stack extents are used. The initial stack size is 1 MB, and 1 MB is used in each extent. Ensure that you choose a large enough downward stack size because the performance of XPLINK can be adversely affected when stack extents are used.

To determine suitable stack sizes, you can use the Language Environment Report Storage tool.

To use this tool, use the **RPTSTG** option with the **_CEE_RUNOPTS** environment variable to test a message flow. Set this option in the component profile (`BIPBPROF` for a broker) during the development and test of message flows that are intended for production; for example:

```
export _CEE_RUNOPTS=RPTSTG\(ON\)
```

You can then override the default values for the stack sizes on z/OS by altering or adding the **_CEE_RUNOPTS** environment variable.

You can do this for all execution groups defined to a broker, or a specific execution group.

About this task

To update the component profile, take the following steps:

Procedure

1. Stop the broker.
2. Make the necessary changes to the profile.
3. Submit `BIPGEN` to re-create the `ENVFILE` and any execution group specific `ENVFILE`s.
4. Restart the broker.

Results

To update the stack sizes for a specific execution group in a broker, take the following steps:

1. Stop the broker.
2. Make the necessary changes to the execution group specific profile.
3. Submit `BIPGEN` to re-create the broker `ENVFILE` and any execution group specific `ENVFILE`s.

4. Restart the broker.

Example

The following example shows how you can change the default stack value of 1 MB to 2 MB:

```
export _CEE_RUNOPTS=THREADSTACK64\(ON,2M,1M,128M\)
```

What to do next

When you use the **RPTSTG** option, it increases the time that an application takes to run, so use it as an aid to the development of message flows only, and not in your final production environment. When you have determined the correct stack sizes needed, remove this option from the **_CEE_RUNOPTS** environment variable.

XPLINK stacks grow downward in virtual storage while the standard linkage grows upward. To avoid affecting performance by switching between downward stack space and upward stack space during run time, compile user-defined extensions using the XPLINK option where possible. If your message flow uses user-defined extensions that have been compiled with the standard linkage convention, set a suitable value for the upward stack size.

Related concepts:

"Stack storage" on page 3268
Depending on the design of your message flow, you might need to increase the stack size.

Related tasks:

"Increasing the stack size on Windows, Linux, and UNIX systems" on page 3255
Increase the stack size on Windows, Linux, and UNIX systems by setting the **MQSI_THREAD_STACK_SIZE** environment variable to an appropriate value.

Tuning the HEAP settings on z/OS

HEAP controls the allocation of the initial heap, controls allocations of additional heap increments, and specifies how that storage is managed.

About this task

WebSphere Message Broker requests an initial heap storage allocation and subsequent heap increments, the sizes of which depend on the type of process. For example, the DFE process requests an initial heap storage allocation of 40 Mb, with subsequent heap increments of 5 Mb. RPTOPTS can be used to generate a report of the runtime options in effect for each process.

- KEEP, which is the default value, specifies that storage allocated to HEAP increments is not released when the last segment of the allocated storage is freed.
- FREE specifies that storage allocated to HEAP increments is released when the last segment of the allocated storage is freed.

For performance reasons, WebSphere Message Broker takes the default, KEEP. For most message processing scenarios, when storage allocations are less than 5 Mb, it is more efficient to reuse storage that has been freed within the heap increment. With KEEP, the 5 Mb heap increment remains allocated, even if all of the storage segments have been released.

If storage requests frequently exceed 5 Mb, these requests are allocated directly on the heap. When the object is freed, the allocation remains on the heap, and is reused for subsequent storage requests whose size is less than, or equal to, the size of the heap allocation. Over time, the heap allocation is used for different-sized objects, and this can lead to fragmentation, which in turn can result in high storage usage. In these circumstances, consider setting the HEAP runtime environment variable for the Language Environment to use the FREE parameter.

To set HEAP for all execution groups in a broker, change or add the Language Environment _CEE_RUNOPTS environment variable in the component profile (BIPBPROF):

1. Stop the broker.
2. Make the necessary changes to the profile.
3. Submit BIPGEN to re-create the ENVFILE and any execution group specific ENVFILEs.
4. Restart the broker.

To set HEAP for a specific execution group, change or add the Language Environment _CEE_RUNOPTS environment variable in the execution group specific profile (renamed BIPEPROF):

1. Stop the broker.
2. Make the necessary changes to the execution group specific profile.
3. Submit BIPGEN to re-create the broker ENVFILE and any execution group specific ENVFILEs.
4. Restart the broker.

For example, you can change the default values of KEEP to FREE in the following line:

```
_CEE_RUNOPTS=HEAP64(40M,5M,FREE,9M,32K,KEEP,4096,4096,FREE)
```

Setting configuration timeout values

Change timeout values that affect configuration tasks in the broker.

Before you begin

Before you start:

Read "Packaging and deployment overview" on page 3210 to understand the conditions under which these timeout values apply.

About this task

The broker receives configuration requests from the WebSphere Message Broker Toolkit, the WebSphere Message Broker Explorer, and CMP API applications. You can also change its configuration by running several commands, for example, **mqsideploy**.

Several factors affect the time that a broker takes to apply and respond to these requests. These factors include the load on the computer on which the broker is running, network delays between components, and the work that execution groups are performing at the time the request is received. The number of message flows in an execution group, and their complexity, and large message sets, might also affect the time taken.

You can change the length of time that a broker can take to perform these actions by using two parameters that you can set on the **mqsicreatebroker** and **mqsichangebroker** commands. The combined default value for these parameters is approximately 6 minutes (360 seconds).

During development and test of message flows and broker configurations, experiment with the values that you set for these timeout parameters to determine appropriate values for your resources.

- **-g** *ConfigurationChangeTimeout*

  This value defines the maximum time (in seconds) that is allowed for a user configuration request to be processed, and defaults to 5 minutes (300 seconds). The value is affected by the system load (including processor use), and by the load of each execution group. If the request is not completed in this time, the broker generates warning message BIP2066, but continues to implement the change. The broker records further diagnosis information in the system and event logs.

- **-k** *InternalConfigurationTimeout*

  This value defines the maximum time (in seconds) that is allowed for an internal configuration change to be processed and defaults to 1 minute (60 seconds). For example, it defines the length of time that the broker can take to start an execution group before a response is required.

  The broker starts an internal process to start an execution group and to make all the message flows active. Part of this initialization is performed serially (one execution group at a time), therefore if the change affects more than one execution group, the time required increases. If an execution group exceeds this timeout value, the broker generates a warning message BIP2080. However, the initialization continues and the execution group is started. The broker records further diagnosis information in the system and event logs.

The sum of the values of *ConfigurationChangeTimeout* and the *InternalConfigurationTimeout* parameters represents the maximum length of time that a broker can take to process a deployed configuration message before it generates a negative response. Check that typical configurations complete successfully within the time that you have specified, to minimize warning messages.

Look for success messages in the Administration log in the WebSphere Message Broker Explorer, or Deployment Log in the WebSphere Message Broker Toolkit. When success messages are displayed the deployment has completed. If you start a deployment and record how long it takes for the success messages to appear, you can use this time interval as the basis for setting these timeout values.

If the broker is on a production system, increase the values for both *ConfigurationChangeTimeout* and *InternalConfigurationTimeout* parameters to allow for application messages that are currently being processed by message flows to be completed before the configuration change is applied. Also consider increasing the value if you have merged message flows into fewer execution groups that you are using for testing.

If the broker is on a development or test system, you might want to reduce timeout lengths (in particular, the value of the *ConfigurationChangeTimeout* parameter) to improve perceived response times, and to force a response from a broker that is not showing expected behavior. However, reducing the timeout values decreases the probability of deploying a configuration change successfully.

During broker startup, the *InternalConfigurationTimeout* parameter is automatically extended based on the number of execution groups which a broker contains. At this time, the timeout period reported in the BIP2080 warning message may not match the value configured for the *InternalConfigurationTimeout* parameter.

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

"Packaging and deployment overview" on page 3210
Deployment is the process of transferring data to an execution group on a broker so that it can take effect in the broker. Message flows and associated resources are packaged in broker archive (BAR) files for deployment.

Related tasks:

"Creating a broker" on page 611
You can create brokers on every platform that is supported by WebSphere Message Broker. The broker runs as a 64-bit application on all platforms except Linux on x86 and Windows on x86.

Chapter 11, "Packaging and deploying," on page 3209
Package resources that you create in the WebSphere Message Broker Toolkit, such as message flows, and deploy them to execution groups on brokers.

"Checking the results of deployment" on page 3243
After you have made a deployment, check that the operation has completed successfully.

Related reference:

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsideploy** command" on page 3872
Use the **mqsideploy** command to make a deployment request to the broker.

Message flow performance

Understand the factors that control and influence the performance of your message flows.

About this task

- "Optimizing message flow throughput" on page 587
- "Optimizing message flow response times" on page 3264
- "System resources for message flow development" on page 3267

Optimizing message flow throughput

Each message flow that you design must provide a complete set of processing for messages received from a certain source. This design might result in very complex message flows that include large numbers of nodes that can cause a performance overhead, and might create potential bottlenecks. You can increase the number of message flows that process your messages to provide the opportunity for parallel processing and therefore improved throughput.

About this task

The mode that your broker is working in can affect the number of message flows that you can use; see "Restrictions that apply in each operation mode" on page 3657.

You can also consider the way in which the actions taken by the message flow are committed, and the order in which messages are processed.

Consider the following options for optimizing message flow throughput:

Multiple threads processing messages in a single message flow

When you deploy a message flow, the broker automatically starts an instance of the message flow for each input node that it contains. This behavior is the default. However, if you have a message flow that handles a very large number of messages, the input source (for example, a WebSphere MQ queue) might become a bottleneck.

You can update the `Additional Instances` property of the deployed message flow in the BAR file: the broker starts additional copies of the message flow on separate threads, providing parallel processing. This option is the most efficient way of handling this situation, if you are not concerned about the order in which messages are processed.

If the message flow receives messages from a WebSphere MQ queue, you can influence the order in which messages are processed by setting the `Order Mode` property of the MQInput node:

- If you set `Order Mode` to By User ID, the node ensures that messages from a specific user (identified by the UserIdentifier field in the MQMD) are processed in guaranteed order. A second message from one user is not processed by an instance of the message flow if a previous message from this user is currently being processed by another instance of the message flow.
- If you set `Order Mode` to By Queue Order, the node processes one message at a time to preserve the order in which the messages are read from the queue. Therefore, this node behaves as though you have set the `Additional Instances` property of the message flow to zero.
- If you set `Order Mode` to `User Defined`, you can order messages by any message element, by setting an XPath or ESQL expression in the Order field location property. The node ensures that messages with the same value in the order field message element are processed in guaranteed order. A second message with the same value in the order field message element is not processed by an instance of the message flow if a previous message with the same value is currently being processed by another instance of the message flow.

  If the field is missing, an exception is raised, and the message is rolled back. NULL and empty values are processed separately, in parallel.

  If you set `Order Mode` to By User ID or `User Defined`, and the message flow uses transformation nodes, it is advisable to set the `Parse Timing` to Immediate.

For publish/subscribe applications that communicate with the broker over any supported protocol, messages for a particular topic are published by brokers in the same order as they are received from publishers (subject to reordering based on message priority, if applicable). Therefore each

subscriber receives messages from a particular broker, on a particular topic, from a particular publisher, in the order that they are published by that publisher.

However, it is possible for messages, occasionally, to be delivered out of order. This situation can happen, for example, if a link in the network fails and subsequent messages are routed by another link.

If you need to ensure the order in which messages are received, you can use either the **SeqNum** (sequence number) or **PubTime** (publish time stamp) parameter on the **Publish** command for each published message, to calculate the order of publishing.

For more information about the techniques recommended for all MQI and AMI users, see the *Application Programming Reference* and *Application Programming Guide* sections in the WebSphere MQ Version 7 Information Center online for programs written to the MQI, and the *WebSphere MQ Application Messaging Interface* book for programs written to the AMI.

The *WebSphere MQ Application Messaging Interface* book is available from the WebSphere MQ Library web page (listed under Version 5.3), or from SupportPac MA0F on the WebSphere MQ SupportPacs web page.

See also the *Publish/Subscribe User's Guide* section in the WebSphere MQ Version 7 Information Center online.

The broker does not provide message ordering for messages received across WebSphere MQ Web Services Transport.

Multiple copies of the message flow in a broker
You can also deploy several copies of the same message flow to different execution groups in the same broker. This option has similar effects to increasing the number of processing threads in a single message flow, although typically provides less noticeable gains.

This option also removes the ability to determine the order in which the messages are processed, because, if there is more than one copy of the message flow active in the broker, each copy can be processing a message at the same time, from the same queue. The time taken to process a message might vary, and multiple message flows accessing the same queue might therefore read messages from the input source in a random order. The order of messages produced by the message flows might not correspond to the order of the original messages.

Ensure that the applications that receive message from these message flows can tolerate out-of-order messages. Additionally, ensure that the input nodes in these message flows are suitable for deployment to different processes.

Copies of the message flow in multiple brokers
You can deploy several copies of the same message flow to different brokers. This option requires changes to your configuration, because you must ensure that applications that supply messages to the message flow can put their messages to the right input queue or port. You can often make these changes when you deploy the message flow by setting the message flow's configurable properties.

The scope of the message flow
You might find that, in some circumstances, you can split a single message flow into several different flows to reduce the scope of work that each message flow performs. If you do split your message flow, be aware that it

is not possible to run the separate message flows in the same unit of work, and if transactional aspects to your message flow exist (for example, the updating of multiple databases), this option does not provide a suitable solution.

The following two examples show when you might want to split a message flow:

1. In a message flow that uses a RouteToLabel node, the input queue has become a bottleneck. You can use another copy of the message flow in a second execution group, but this option is not appropriate if you want all of the messages to be handled in the order in which they are shown on the queue. You can consider splitting out each branch of the message flow that starts with a Label node by providing an input queue and input node for each branch. This option might be appropriate, because when the message is routed by the RouteToLabel node to the relevant Label node, it has some level of independence from all other messages.

   You might also need to provide another input queue and input node to complete any common processing that the Label node branches connect to when unique processing has been done.
2. If you have a message flow that processes very large messages that take a considerable time to process, you might be able to:
   a. Create other copies of the message flow that use a different input queue (you can set this option up in the message flow itself, or you can update this property when you deploy the message flow).
   b. Set up WebSphere MQ queue aliases to redirect messages from some applications to the alternative queue and message flow.

   You can also create a new message flow that replicates the function of the original message flow, but only processes large messages that are immediately passed on to it by the original message flow, that you modified to check the input message size and redirect the large messages.

The frequency of commits

If a message flow receives input messages on a WebSphere MQ queue, you can improve its throughput for some message flow scenarios by modifying its default properties after you have added it to a BAR file. (These options are not available if the input messages are received by other input nodes; commits in those message flows are performed for each message.)

The following properties control the frequency with which the message flow commits transactions:

- `Commit Count`. This property represents the number of messages processed from the input queue before an MQCMIT is issued.
- `Commit Interval`. This property represents the time interval that elapses before an MQCMIT is started.

Related tasks:

"Optimizing message flow response times" on page 3264
You can use different solutions to improve message flow response times.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"Configurable message flow properties" on page 4020
When you add a message flow to a broker archive (BAR) file in preparation for deploying it to a broker, you can set additional properties that influence its run time operation. These properties are available for review and update when you select the **Manage and Configure** tab for the broker archive file.

"Publish message" on page 6405

Related information:

 WebSphere MQ Version 7 Information Center online

Optimizing message flow response times

You can use different solutions to improve message flow response times.

Before you begin

Before you start:

Read the following concept topic:

- "Message flow nodes" on page 1024

About this task

When you design a message flow, the flexibility and functional capabilities of the built-in nodes often mean that there are several ways to achieve the processing and results that you require. You might find that different solutions deliver different levels of performance and, if performance is an important consideration for you, take it into account when designing your message flow

Your applications can perceive performance in either of these ways:

- The response time indicates how quickly each message is processed by the message flow. The response time is particularly influenced by how you design your message flows. Response time is discussed in this topic.
- The throughput indicates how many messages of particular sizes can be processed by a message flow in a specified time. The throughput is mainly affected by configuration and system resource factors, and is discussed in "Optimizing message flow throughput" on page 587, with other domain configuration information.

Several aspects influence message flow response times. However, as you create and modify your message flow design to arrive at the best results for your specific business requirements, also consider the eventual complexity of the message flow. The most efficient message flows are not necessarily the easiest to understand and maintain; experiment with the solutions available to arrive at the best balance for your needs.

Several factors influence message flow response times:

The number of nodes that you include in the message flow

Every node increases the amount of processing required in the broker, therefore, consider the content of the message flow carefully, including the use of subflows.

Use as few nodes as possible in a message flow; every node that you include in the message flow increases the amount of processing required in the broker. The number of nodes in a single flow has an upper limit, which is governed by system resources, particularly the stack size. For more information about stack sizes, see "System resources for message flow development" on page 3267.

How the message flow routes and processes messages

In some situations, you might find that the built-in nodes, and perhaps other nodes that are available in your system, provide more than one way of providing the same function. Choose the simplest configuration. For example, to define some specific processing based on the value of a field in each message, you might design a message flow that has a sequence of Filter nodes to handle each case. If appropriate, you can group messages through the Filter node to reduce the number of nodes through which each message type has to pass before being processed.

For example, you might have a message flow that handles eight different messages (Invoice, Despatch Note, and so on). You can include a Filter node to identify each type of message and route it according to its type. You can optimize the performance of this technique by testing for the most frequent message types in the earliest Filter nodes. However, the eighth message type must always pass through eight Filter nodes.

If you can group the message types (for example, if the message type is numeric, and you can test for all types greater than four and not greater than four), you can reduce the number of Filter nodes required. The first Filter node tests for greater than four, and passes the message on to two further Filter nodes (attached to the True and False terminals) that test for less than two and less than six. An additional four Filter nodes can then test for one or two, three or four, and so on. Although the number of Filter nodes required is the same, the number of nodes through which each message passes is reduced.

You might find that using a RouteToLabel node with a set of Label nodes provides a better alternative to a sequence of Filter nodes. Each message passes through a smaller number of nodes, improving throughput. However, you must also consider that using a RouteToLabel node necessitates the use of a Compute node: the increase in the amount of processing required in the broker that is caused by the node might outweigh the advantages. If you are dealing with a limited number of message types, a small number of Filter nodes is more efficient.

The following sample demonstrates how you can use the RouteToLabel and Label nodes instead of using multiple Filter nodes in the XML_PassengerQuery message flow.

- Airline Reservations

The following sample demonstrates how you can store routing information in a database table in an in-memory cache in the message flow.

- Message Routing

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

If your message flow includes loops
Avoid loops of repeating nodes, which can be very inefficient and can cause performance and stack problems. You might find that a Compute node with multiple PROPAGATE statements avoids the need to loop around a series of nodes.

The efficiency of the ESQL
Check all the ESQL code that you have created for your message flow nodes. As you develop and test a node, you might maintain statements that are not required when you have finalized your message processing. You might also find that something you have coded as two statements can be coded as one. Taking the time to review and check your ESQL code might provide simplification and performance improvements.

If you have imported message flows from a previous release, check your ESQL statements against the ESQL available in Version 6.1 to see if you can use new functions or statements to improve its efficiency.

The use of persistent and transactional messages
Persistent messages are saved to disk during message flow processing. You can avoid this situation by specifying that messages are non-persistent on input, output, or both. If your message flow is handling only non-persistent messages, check the configuration of the nodes and the message flow itself; if your messages are non-persistent, transactional support might be unnecessary. The default configuration of some nodes enforces transactionality; if you update these properties and redeploy the message flow, response times might improve.

Message size
A larger message takes longer to process. If you can split large messages into smaller units of information, you might be able to improve the speed at which they are handled by the message flow. The following sample demonstrates how to minimize the virtual storage requirements for the message flow to improve a message flow's performance when processing potentially large messages.

- Large Messaging

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Message format
Although WebSphere Message Broker supports multiple message formats, and provides facilities that you can use to transform from one format to another, this transformation increases the amount of processing required in the broker. Make sure that you do not perform any unnecessary conversions or transformations.

You can find more information about improving the performance of a message flow in a developerWorks article (developerWorks article on message flow performance).

Related concepts:

"System resources for message flow development"
Configure your message flows to make the best use of computer resources, especially if you will process large messages.

Related tasks:

Chapter 7, "Configuring brokers for test and production environments," on page 579
Create one or more brokers on one or more computers, and configure them on your test and production systems to process messages that contain your business data.

"Optimizing message flow throughput" on page 587
Each message flow that you design must provide a complete set of processing for messages received from a certain source. This design might result in very complex message flows that include large numbers of nodes that can cause a performance overhead, and might create potential bottlenecks. You can increase the number of message flows that process your messages to provide the opportunity for parallel processing and therefore improved throughput.

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Using more than one input node" on page 1473
You can include more than one input node in a single message flow.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

System resources for message flow development

Configure your message flows to make the best use of computer resources, especially if you will process large messages.

As well as designing your message flow to optimize throughput, you must ensure that particular areas of storage are efficiently used so that your system does not suffer from capacity issues, and that processes do not end because of lack of resources.

Consider the following storage issues when developing your message flows:

- "Stack storage" on page 3268
- "JVM heap sizing" on page 3269
- "Configuring the storage of events for aggregation nodes" on page 753
- "Configuring the storage of events for Collector nodes" on page 755
- "Configuring the storage of events for Resequence nodes" on page 758
- "Configuring the storage of events for timeout nodes" on page 760

Related tasks:

"Optimizing message flow response times" on page 3264
You can use different solutions to improve message flow response times.

Related reference:

"Message flows" on page 4015
Use the reference information in this section to develop your message flows and related resources.

Stack storage:

Depending on the design of your message flow, you might need to increase the stack size.

When a message flow thread starts, it requires storage to perform the instructions that are defined by the message flow nodes. This storage comes from the execution group's heap and stack size. The default stack size that is allocated to a message flow thread depends on the operating system that is used:

Windows On Windows, each message flow thread is allocated 1 MB of stack space.

Linux On Linux, each message flow thread is allocated 1 MB of stack space.

UNIX On UNIX, each message flow thread is allocated 1 MB of stack space.

AIX On AIX, each message flow thread is allocated 2 MB of stack space.

z/OS On z/OS, each message flow thread is allocated 512 KB of downward stack space and 50 KB of upward stack space.

In a message flow, a node typically uses 2 KB of the stack space. A typical message flow can therefore include 250 nodes on z/OS, 500 nodes on UNIX systems and 500 nodes on Windows. This amount can be higher or lower depending on the type of nodes used and the processing that they perform.

In WebSphere Message Broker, any processing that involves nested or recursive processing can cause extensive usage of the stack. For example, in the following situations you might need to increase the stack size:

- When a message flow is processing a message that contains a large number of repetitions or complex nesting.
- When a message flow is executing ESQL that calls the same procedure or function recursively, or when an operator (for example, the concatenation operator) is used repeatedly in an ESQL statement.

You can increase the stack size to improve performance. For details, see:

- "Increasing the stack size on Windows, Linux, and UNIX systems" on page 3255
- "Increasing the stack size on z/OS" on page 3256

Related tasks:

"Optimizing message flow response times" on page 3264
You can use different solutions to improve message flow response times.

Related reference:

"Message flows" on page 4015
Use the reference information in this section to develop your message flows and related resources.

JVM heap sizing:

The Java virtual machine (JVM) heap is an independent memory allocation that can reduce the capacity of the main memory heap.

Every execution group creates its own JVM. The execution group uses the JVM to execute the internal administration threads that require Java. This usage is typically minimal. The primary use of the JVM is for the message flow nodes that include the IBM primitives, user defined extensions, and Java routines that are called from ESQL. You cannot control how much of the JVM heap the IBM primitives use, but you can affect usage of the JVM heap in the Java that is implemented in the following resources:

- A Java user-defined extension node
- A JavaCompute node
- A Java routine that is called from ESQL

From WebSphere Message Broker onwards, the JVM is created with a minimum of 32 MB of space, and a maximum of 256 MB, allocated and reserved for its use. As with any JVM, you can pass parameters in to set the minimum and maximum heap sizes. Note that on 32-bit platforms, the JVM reserves heap space based on the maximum heap size.

You might need to increase the maximum heap size allocated if you plan to run large messages through the Java primitive nodes listed above.

To give more capacity to a message flow that is going to process large messages, reduce the minimum JVM heap size to allow the main memory heap to occupy more address space. For details of how to reduce the minimum JVM heap size, see "Setting the JVM heap size" on page 3254.

Related concepts:

"Stack storage" on page 3268
Depending on the design of your message flow, you might need to increase the stack size.

Related tasks:

"Setting the JVM heap size" on page 3254
When you start an execution group, it creates a Java virtual machine (JVM) for executing a Java user-defined node.

"Optimizing message flow response times" on page 3264
You can use different solutions to improve message flow response times.

Related reference:

"Message flows" on page 4015
Use the reference information in this section to develop your message flows and related resources.

Configuring the storage of events for aggregation nodes:

You can use an Aggregation configurable service to control the storage of events for AggregateControl and AggregateReply nodes.

About this task

By default, the storage queues used by all aggregation nodes are:

- SYSTEM.BROKER.AGGR.CONTROL

- SYSTEM.BROKER.AGGR.REPLY
- SYSTEM.BROKER.AGGR.REQUEST
- SYSTEM.BROKER.AGGR.UNKNOWN
- SYSTEM.BROKER.AGGR.TIMEOUT

However, you can control the queues that are used by different aggregation nodes by creating alternative queues containing a *QueuePrefix*, and using an Aggregation configurable service to specify the names of those queues for storing events.

Follow these steps to specify the queues that are used to store event states, and to set the expiry time of an aggregation:

Procedure

1. Create the storage queues to be used by the aggregation nodes. The following queues are required:
   - SYSTEM.BROKER.AGGR.*QueuePrefix*.CONTROL
   - SYSTEM.BROKER.AGGR.*QueuePrefix*.REPLY
   - SYSTEM.BROKER.AGGR.*QueuePrefix*.REQUEST
   - SYSTEM.BROKER.AGGR.*QueuePrefix*.UNKNOWN
   - SYSTEM.BROKER.AGGR.*QueuePrefix*.TIMEOUT

   The *QueuePrefix* variable can contain any characters that are valid in a WebSphere MQ queue name, but must be no longer than eight characters and must not begin or end with a period (.). For example, `SET1` and `SET.1` are valid queue prefixes, but `.SET1` and `SET1.` are invalid.

   If you do not create the storage queues, WebSphere Message Broker creates the set of queues when the node is deployed; these queues are based on the default queues. If the queues cannot be created, the message flow is not deployed.
2. Use the **mqsicreateconfigurableservice** command to create an Aggregation configurable service. You can create a configurable service to be used with either a specific aggregation or with all aggregations in an execution group.
   a. If the configurable service is to be used with a specific aggregation, ensure that the name of the configurable service is the same as the name that you specify in the `Aggregate name` property on the AggregateControl and AggregateReply nodes. If the configurable service is to be used with all aggregations in an execution group, create the configurable service with the same name as the execution group.
   b. Set the **Queue prefix** property to the required value.
   c. Optional: Set the **Timeout** property to control the expiry time of an aggregation.

   For example, create a configurable service called `myAggregation`, which specifies queues prefixed with SYSTEM.BROKER.AGGR.SET1 and a timeout of 60 seconds:

   ```
   mqsicreateconfigurableservice MYBROKER -c Aggregation -o myAggregation
   -n queuePrefix,timeoutSeconds -v SET1,60
   ```

   You can use the **mqsideleteconfigurableservice** command to delete the Aggregation configurable service. However, the storage queues are not deleted automatically when the configurable service is deleted, so you must delete them separately.For more information, see "Configurable services properties" on page 3766
3. In the AggregateControl and AggregateReply nodes:

a. Ensure that the name of the Aggregation configurable service is the same as the name specified in the `Aggregate name` property on the **Basic** tab; for example, `myAggregation`. If no Aggregation configurable service exists with the same name as the `Aggregate name` property, and if a configurable service exists with the same name as the execution group, that configurable service is used instead.

b. Optional: Use the **mqsichangeproperties** and **mqsireportproperties** commands to change or view the properties of the configurable service. Alternatively, you can use the WebSphere Message Broker Explorer to view or modify a configurable service. For more information about working with configurable services, see "Using the WebSphere Message Broker Explorer to work with configurable services" on page 644.

What to do next

The properties for the configurable service are not used by the broker until you restart or redeploy the message flow, or restart the broker.

Related concepts:

"Using the WebSphere Message Broker Explorer to work with configurable services" on page 644
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view, add, modify and delete configurable services.

Related tasks:

"Viewing configurable services" on page 647
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view the properties of configurable services defined on your broker.

"Modifying an IBM defined configurable service" on page 648
You can create a new configurable service for an external service on which the broker relies, by modifying predefined configurable services provided by IBM. Use the WebSphere Message Broker Explorer to view and modify existing configurable services.

"Modifying a configurable service" on page 649
Use the WebSphere Message Broker Explorer to view and modify existing configurable services.

"Deleting a configurable service" on page 652
Use the WebSphere Message Broker Explorer to delete custom configurable services.

Related reference:

"AggregateControl node" on page 4296
Use the AggregateControl node to mark the beginning of a fan-out of requests that are part of an aggregation.

"AggregateRequest node" on page 4303
Use the AggregateRequest node to record the fact that request messages have been sent. This node also collects information that helps the AggregateReply node to construct the compound response message.

"AggregateReply node" on page 4299
Use the AggregateReply node to mark the end of an aggregation fan-in. This node collects replies and combines them into a single compound message.

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

Configuring the storage of events for Collector nodes:

You can use a Collector configurable service to control the storage of events for Collector nodes.

About this task

By default, the storage queues used by all Collector nodes are:

- SYSTEM.BROKER.EDA.EVENTS
- SYSTEM.BROKER.EDA.COLLECTIONS

These queues are also used by the Resequence node.

However, you can control the queues that are used by different Collector nodes by creating alternative queues that contain a *QueuePrefix* variable, and by using a Collector configurable service to specify the names of those queues for storing events.

Follow these steps to specify the queues that are used to store event states, and to set the expiry for the collection:

Procedure

1. Create the storage queues to be used by the Collector node. The following queues are required:
   - SYSTEM.BROKER.EDA.*QueuePrefix*.EVENTS
   - SYSTEM.BROKER.EDA.*QueuePrefix*.COLLECTIONS

   The *QueuePrefix* variable can contain any characters that are valid in a WebSphere MQ queue name, but must be no longer than eight characters and must not begin or end with a period (.). For example, `SET1` and `SET.1` are valid queue prefixes, but `.SET1` and `SET1.` are invalid.

   If you do not create the storage queues, WebSphere Message Broker creates the set of queues when the node is deployed; these queues are based on the default queues. If the queues cannot be created, the message flow is not deployed.
2. Use the **mqsicreateconfigurableservice** command to create a Collector configurable service. You can create a configurable service to be used with either a specific collection or with all collections in an execution group.
   a. If you are creating a configurable service to be used with a specific collection, ensure that the name of the configurable service is the same as the name that you specify in the `Configurable service` property on the Collector node. If you are creating a configurable service to be used with all collections in the execution group, ensure that the configurable service has the same name as the execution group.
   b. Set the **Queue prefix** property to the required value.

c. Optional: Set the **Collection expiry** property.

For example, create a Collector configurable service called `myCollectorService`, which uses queues prefixed with SYSTEM.BROKER.EDA.SET1, and with a collection expiry of 60 seconds:

```
mqsicreateconfigurableservice MYBROKER -c Collector -o myCollectorService
-n queuePrefix,collectionExpirySeconds -v SET1,60
```

You can use the **mqsideleteconfigurableservice** command to delete the Collector configurable service. However, the storage queues are not deleted automatically when the configurable service is deleted, so you must delete them separately.

For more information, see "Configurable services properties" on page 3766

3. In the Collector node:
   a. If the configurable service is to be used for a specific collection, specify the name of the configurable service in the `Configurable service` property on the **Advanced** tab; for example, `myCollectorService`. If you do not set the `Configurable service` property, and if a configurable service exists with the same name as the execution group, that configurable service is used instead.
   b. Optional: Use the **mqsichangeproperties** and **mqsireportproperties** commands to change or view the properties of the configurable service. Alternatively, you can use the WebSphere Message Broker Explorer to view or modify a configurable service. For more information about working with configurable services, see "Using the WebSphere Message Broker Explorer to work with configurable services" on page 644.

What to do next

The properties for the configurable service are not used by the broker until you restart or redeploy the message flow, or restart the broker.

Related concepts:

"Using the WebSphere Message Broker Explorer to work with configurable services" on page 644
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view, add, modify and delete configurable services.

Related tasks:

"Viewing configurable services" on page 647
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view the properties of configurable services defined on your broker.

"Modifying an IBM defined configurable service" on page 648
You can create a new configurable service for an external service on which the broker relies, by modifying predefined configurable services provided by IBM. Use the WebSphere Message Broker Explorer to view and modify existing configurable services.

"Modifying a configurable service" on page 649
Use the WebSphere Message Broker Explorer to view and modify existing configurable services.

"Deleting a configurable service" on page 652
Use the WebSphere Message Broker Explorer to delete custom configurable services.

Related reference:

"Collector node" on page 4333
Use the Collector node to create message collections based on rules that you configure in the node.

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

Configuring the storage of events for Resequence nodes:

You can use a Resequence configurable service to control the storage of events for Resequence nodes.

About this task

By default, the storage queues used by all Resequence nodes are:

- SYSTEM.BROKER.EDA.EVENTS
- SYSTEM.BROKER.EDA.COLLECTIONS

These queues are also used by the Collector node.

However, you can control the queues that are used by different Resequence nodes by creating alternative queues that contain a *QueuePrefix* variable, and by using a Resequence configurable service to specify the names of those queues for storing events.

Follow these steps to specify the queues that are used to store event states, and to set the timeout and the start and end of the sequence:

Procedure

1. Create the storage queues to be used by the Resequence node. The following queues are required:
   - SYSTEM.BROKER.EDA.*QueuePrefix*.EVENTS
   - SYSTEM.BROKER.EDA.*QueuePrefix*.COLLECTIONS

   The *QueuePrefix* variable can contain any characters that are valid in a WebSphere MQ queue name, but must be no longer than eight characters and must not begin or end with a period (.). For example, `SET1` and `SET.1` are valid queue prefixes, but `.SET1` and `SET1.` are invalid.

   If you do not create the storage queues, WebSphere Message Broker creates the set of queues when the node is deployed; these queues are based on the default queues. If the queues cannot be created, the message flow is not deployed.
2. Use the **mqsicreateconfigurableservice** command to create a Resequence configurable service. You can create a configurable service to be used with either a specific sequence or with all sequences in an execution group.

a. If you are creating a configurable service to be used with a specific sequence, ensure that the name of the configurable service is the same as the name that you specify in the `Configurable service` property on the Resequence node. If you are creating a configurable service to be used with all sequences in the execution group, ensure that the configurable service has the same name as the execution group.

b. Set the **Queue prefix** property to the required value.

c. Optional: Set the **Missing message timeout**, **Start of sequence**, and **End of sequence** properties.

For example, create a Resequence configurable service called `myResequenceService`, which uses queues prefixed with SYSTEM.BROKER.EDA.SET1, with a missing message timeout of 60 seconds, and which waits five seconds before determining the start and end numbers in a sequence:

```
mqsicreateconfigurableservice MYBROKER -c Resequence -o myResequenceService
-n queuePrefix,missingMessageTimeoutSeconds,startSequenceSeconds,endSequenceSeconds -v SET1,60
```

You can use the **mqsideleteconfigurableservice** command to delete the Resequence configurable service. However, the storage queues are not deleted automatically when the configurable service is deleted, so you must delete them separately.For more information, see "Configurable services properties" on page 3766

3. In the Resequence node:

a. If the configurable service is to be used for a specific sequence, specify the name of the configurable service on the **Advanced** tab; for example, `myResequenceService`. If you do not set the `Configurable service` property, and if a configurable service exists with the same name as the execution group, that configurable service is used instead.

b. Optional: Use the **mqsichangeproperties** and **mqsireportproperties** commands to change or view the properties of the configurable service. Alternatively, you can use the WebSphere Message Broker Explorer to view or modify a configurable service. For more information about working with configurable services, see "Using the WebSphere Message Broker Explorer to work with configurable services" on page 644.

What to do next

The properties for the configurable service are not used by the broker until you restart or redeploy the message flow, or restart the broker.

Related concepts:

"Using the WebSphere Message Broker Explorer to work with configurable services" on page 644
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view, add, modify and delete configurable services.

Related tasks:

"Viewing configurable services" on page 647
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view the properties of configurable services defined on your broker.

"Modifying an IBM defined configurable service" on page 648
You can create a new configurable service for an external service on which the broker relies, by modifying predefined configurable services provided by IBM. Use the WebSphere Message Broker Explorer to view and modify existing configurable

services.

"Modifying a configurable service" on page 649
Use the WebSphere Message Broker Explorer to view and modify existing configurable services.

"Deleting a configurable service" on page 652
Use the WebSphere Message Broker Explorer to delete custom configurable services.

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"Resequence node" on page 4651
Use the Resequence node to control the sequence in which a group (or groups) of incoming messages are propagated in a message flow.

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

Configuring the storage of events for timeout nodes:

You can use a Timer configurable service to control the storage of events for TimeoutNotification and TimeoutControl nodes.

About this task

By default, the storage queue used by all timeout nodes is the SYSTEM.BROKER.TIMEOUT.QUEUE.

However, you can control the queues that are used by different timeout nodes by creating alternative queues that contain a *QueuePrefix* variable, and by using a Timer configurable service to specify the names of those queues for storing events.

Follow these steps to specify the queue that is used to store event states:

Procedure

1. Create the storage queue to be used by the timeout nodes. The following queue is required:
   - SYSTEM.BROKER.TIMEOUT.*QueuePrefix*.QUEUE

   The *QueuePrefix* variable can contain any characters that are valid in a WebSphere MQ queue name, but must be no longer than eight characters and must not begin or end with a period (.). For example, `SET1` and `SET.1` are valid queue prefixes, but `.SET1` and `SET1.` are invalid.

   If you do not create the storage queue, WebSphere Message Broker creates the queue when the node is deployed; this queue is based on the default queue. If the queue cannot be created, the message flow is not deployed.

2. Use the **mqsicreateconfigurableservice** command to create a Timer configurable service. You can create a configurable service to be used with either specific timeout requests or with all timeout requests in an execution group.
   a. If the configurable service is to be used with specific timeout requests, create the configurable service with the same name as the `Unique identifier` property on the TimeoutNotification and TimeoutControl nodes. If the configurable service is to be used with all timeout requests in an execution group, create the configurable service with the same name as the execution group.
   b. Set the **Queue prefix** property to the required value.

   For example, create a Timer configurable service that uses a queue prefixed with SYSTEM.BROKER.TIMEOUT.SET1:

   ```
   mqsicreateconfigurableservice MB7BROKER -c Timer -o myTimer
   -n queuePrefix -v SET1
   ```

   You can use the **mqsideleteconfigurableservice** command to delete the Timer configurable service. However, the storage queue is not deleted automatically when the configurable service is deleted, so you must delete it separately.For more information, see "Configurable services properties" on page 3766.
3. In the TimeoutNotification and TimeoutControl nodes:
   a. Ensure that the name of the Timer configurable service is the same as the name specified in the `Unique Identifier` property on the **Basic** tab; for example, `myTimer`. If there is no Timer configurable service with the same name as the `Unique Identifier`, and if there is a configurable service with the same name as the execution group, that configurable service is used instead.
   b. Optional: Use the **mqsichangeproperties** and **mqsireportproperties** commands to change or view the properties of the configurable service. Alternatively, you can use the WebSphere Message Broker Explorer to view or modify a configurable service. For more information about working with configurable services, see "Using the WebSphere Message Broker Explorer to work with configurable services" on page 644.

What to do next

The properties for the configurable service are not used by the broker until you restart or redeploy the message flow, or restart the broker.

Related concepts:

"Using the WebSphere Message Broker Explorer to work with configurable services" on page 644
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view, add, modify and delete configurable services.

"Configuring timeout flows" on page 2809
Use the TimeoutControl and TimeoutNotification nodes in message flows to process timeout requests or to generate timeout notifications at specified intervals.

"Sending timeout request messages" on page 2810
To set a controlled timeout, send a message with a set of elements with well known names to a TimeoutControl node. These elements control the properties of the timeout to be created or deleted.

Related tasks:

"Viewing configurable services" on page 647
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view the properties of configurable services defined on your broker.

"Modifying an IBM defined configurable service" on page 648
You can create a new configurable service for an external service on which the broker relies, by modifying predefined configurable services provided by IBM. Use the WebSphere Message Broker Explorer to view and modify existing configurable services.

"Modifying a configurable service" on page 649
Use the WebSphere Message Broker Explorer to view and modify existing configurable services.

"Deleting a configurable service" on page 652
Use the WebSphere Message Broker Explorer to delete custom configurable services.

Related reference:

"TimeoutNotification node" on page 4936
Use the TimeoutNotification node to manage timeout-dependent message flows.

"TimeoutControl node" on page 4932
Use the TimeoutControl node to process an input message that contains a timeout request.

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

Tuning the SAP adapter for scalability and performance

You can configure the number of listeners on the adapter and the number of additional instances on the message flow to prevent delays when processing synchronous calls from SAP.

Before you begin

Before you start:

- Ensure that you have created and saved a message flow that contains one or more SAP nodes; for more information, see "Developing message flows that use WebSphere Adapters" on page 2033.
- Read the concept topic "SAP adapter scalability and performance" on page 1949.

About this task

The following steps describe how to tune the SAP adapter for scalability and performance.

Procedure

1. Deploy the BAR file, as described in "Deploying a message flow that uses WebSphere Adapters" on page 3240.
2. Start user trace, as described in "Starting user trace" on page 3197.
3. Start to collect accounting and statistics data, as described in "Starting to collect message flow accounting and statistics data" on page 3288.
4. Run the SAP programs at a typical load.
5. Check user trace for message BIP3461. This message tells you the highest number of listeners that are waiting for message flow threads at any one time. In an ideal situation, this number should be as low as possible. To reduce the number of listeners that are waiting for message flow threads, increase the number of instances on the SAPInput node, or the message flow that contains it.
6. Inspect the accounting and statistics data for the message flow.

Related concepts:

"SAP adapter scalability and performance" on page 1949
You can improve performance by configuring the number of listeners on the adapter and the number of additional instances on the message flow to prevent delays when processing synchronous calls from SAP.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

"Starting user trace" on page 3197
Use user trace for debugging your applications; you can trace brokers, execution groups, and deployed message flows. Start user trace facilities using the **mqsichangetrace** command or the WebSphere Message Broker Explorer.

"Starting to collect message flow accounting and statistics data" on page 3288
You can start collecting message flow accounting and statistics data for an active broker at any time. You can specify what type of statistics you want to collect, and where to send the data.

Related reference:

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"Configurable message flow properties" on page 4020
When you add a message flow to a broker archive (BAR) file in preparation for deploying it to a broker, you can set additional properties that influence its run time operation. These properties are available for review and update when you select the **Manage and Configure** tab for the broker archive file.

Monitoring message flow performance

You can configure your message flow to collect statistics about the operation and behavior of your message flows. Use this information to assess the performance of your message flows.

Before you begin

Before you start:

Read the concept topic about message flow accounting and statistics data.

About this task

The following topics describe the tasks that you can complete to control collection of message flow accounting and statistics data by using the command line:

- "Starting to collect message flow accounting and statistics data" on page 3288
- "Stopping message flow accounting and statistics data collection" on page 3293
- "Viewing message flow accounting and statistics data collection parameters" on page 3294
- "Modifying message flow accounting and statistics data collection parameters" on page 3296
- "Resetting message flow accounting and statistics archive data" on page 3297

The following topics describe the tasks that you can complete to control collection of message flow accounting and statistics data by using the WebSphere Message Broker Explorer:

- "Starting accounting and statistics data collection in the WebSphere Message Broker Explorer" on page 3299
- "Viewing message flow accounting and statistics data" on page 3300
- "Filtering message flow accounting and statistics data" on page 3302
- "Stopping accounting and statistics data collection in the WebSphere Message Broker Explorer" on page 3304

The topics listed here show examples of how to issue the accounting and statistics commands. The examples for z/OS are shown for SDSF; if you are using another interface, you must modify the example shown according to the requirements of that interface. For details of other z/OS options, see "Issuing commands to the z/OS console" on page 3980.

Related concepts:

"Message flow accounting and statistics data" on page 3281
Message flow accounting and statistics data is the information that can be collected by a broker to record performance and operating details of message flow execution.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"Message flow accounting and statistics data" on page 6723
You can collect message flow accounting and statistics data in various output formats.

"**mqsichangeflowstats** command" on page 3744
Use the **mqsichangeflowstats** command to control the accumulation of statistics about message flow operation.

"**mqsireportflowstats** command" on page 3929
Use the **mqsireportflowstats** command to display the current options for accounting and statistics that have been set using the **mqsichangeflowstats** command.

"Issuing commands to the z/OS console" on page 3980
You operate the broker using the z/OS START, STOP, and MODIFY commands.

Message flow accounting and statistics data

Message flow accounting and statistics data is the information that can be collected by a broker to record performance and operating details of message flow execution.

Message flow accounting and statistics data records dynamic information about the runtime behavior of a message flow. For example, it indicates how many messages are processed and how large those messages are, as well as processor usage and elapsed processing times. The broker collects the data and records it in a specified location when one of a number of events occurs (for example, when a snapshot interval expires or when the execution group you are recording information about stops).

The broker takes information about statistics and accounting from the operating system. On some operating systems, such as Windows, Linux, and UNIX, rounding can occur because the system calls that are used to determine the processor times are not sufficiently granular. This rounding might affect the accuracy of the data.

The following restrictions apply to data collection:

- Data relating to the size of messages is not collected for WebSphere Adapters nodes (for example, the SAPInput node), the FileInput node, the JMSInput node, or any user-defined input node that does not create a message tree from a bit stream.

Collecting message flow accounting and statistics data is optional; by default it is switched off. To use this facility, request it on a message flow or execution group basis. The settings for accounting and statistics data collection are reset to the defaults when an execution group is redeployed. Previous settings for message flows in an execution group are not passed on to the new message flows deployed to that execution group. Data collection is started and stopped dynamically when you issue the **mqsichangeflowstats** command or when you click **Statistics** > **Start Statistics** on the message flow in the WebSphere Message Broker Explorer; you do not need to change the broker or the message flow, or redeploy the message flow, to request statistics collection.

You can activate data collection on both your production and test systems. If you collect the default level of statistics (message flow), the effect on broker performance is minimal. However, collecting more data than the default message flow statistics can generate high volumes of report data that might affect performance slightly.

When you plan data collection, consider the following points:

- Collection options
- Accounting origin
- Output formats

You can find more information about how to use accounting and statistics data to improve the performance of a message flow in this developerWorks article on message flow performance.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Monitoring message flow performance" on page 3279
You can configure your message flow to collect statistics about the operation and behavior of your message flows. Use this information to assess the performance of your message flows.

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"Message flow accounting and statistics data" on page 6723
You can collect message flow accounting and statistics data in various output formats.

"**mqsichangeflowstats** command" on page 3744
Use the **mqsichangeflowstats** command to control the accumulation of statistics about message flow operation.

"**mqsireportflowstats** command" on page 3929
Use the **mqsireportflowstats** command to display the current options for accounting and statistics that have been set using the **mqsichangeflowstats** command.

Message flow accounting and statistics collection options:

The options that you specify for message flow accounting and statistics collection determine what information is collected. You can request two types of data collection: Snapshot data and archive data.

- Snapshot data is collected for an interval of approximately 20 seconds. The exact length of the interval depends on system loading and the level of current broker activity. You cannot modify the length of time for which snapshot data is collected. At the end of this interval, the recorded statistics are written to the output destination and the interval is restarted.
- Archive data is collected for an interval that you have set for the broker on the **mqsicreatebroker** or **mqsichangebroker** command. You can specify an interval of between 1 and 43200 minutes, the default value is 60 minutes. At the end of this interval, the recorded statistics are written to the output destination and the interval is restarted.

  An interval is prematurely expired and restarted when any of the following events occur:
  - The message flow is redeployed.
  - The set of statistics data to be collected is modified.
  - The broker is shut down.

  This preserves the integrity of the data already collected when that event occurs.

  z/OS On z/OS, you can set the command parameter to 0, which means that the interval is controlled by an external timer mechanism. This support is provided by the Event Notification Facility (ENF), which you can use instead of the broker command parameter if you want to coordinate the expiration of this timer with other system events.

You can request snapshot data collection, archive data collection, or both. You can activate snapshot data collection while archive data collection is active. The data recorded in both reports is the same, but is collected for different intervals. If you

activate both snapshot and archive data collection, be careful not to combine information from the two different reports, because you might count information twice.

View statistics data as it is generated, using the Broker Statistics view in the WebSphere Message Broker Explorer.

You can use the statistics generated for the following purposes:

- You can record the load that applications, trading partners, or other users put on the broker. This allows you to record the relative use that different users make of the broker, and perhaps to charge them accordingly. For example, you could levy a nominal charge on every message that is processed by a broker, or by a specific message flow.

  Archive data provides the information that you need for a use assessment of this kind.
- You can assess the execution of a message flow to determine why it, or a node within it, is not performing as you expect.

  Snapshot data is appropriate for performance assessment.
- You can determine the route that messages are taking through a message flow. For example, you might find that an error path is taken more frequently than you expect and you can use the statistics to understand when the messages are routed to this error path.

  Check the information provided by snapshot data for routing information; if this is insufficient for your needs, use archive data.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Monitoring message flow performance" on page 3279
You can configure your message flow to collect statistics about the operation and behavior of your message flows. Use this information to assess the performance of your message flows.

"Starting accounting and statistics data collection in the WebSphere Message Broker Explorer" on page 3299
Use the WebSphere Message Broker Explorer to start collecting snapshot accounting and statistics data for your brokers, execution groups, and message flows. You can then view the accounting and statistics data in the Broker Statistics and Broker Statistics Graph views.

Related reference:

"Message flow accounting and statistics data" on page 6723
You can collect message flow accounting and statistics data in various output formats.

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

"**mqsichangeflowstats** command" on page 3744
Use the **mqsichangeflowstats** command to control the accumulation of statistics about message flow operation.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

Message flow accounting and statistics accounting origin:

Accounting and statistics data can be accumulated and reported with reference to an identifier associated with a message in a message flow. This identifier is the accounting origin, which provides a method of producing individual accounting and statistics data for multiple accounting origins that generate input to message flows. The accounting origin can be a fixed value, or it can be dynamically set according to your criteria.

For example, if your broker hosts a set of message flows associated with a particular client in a single execution group, you can set a specific value for the accounting origin for all these flows. You can then analyze the output provided to assess the use that the client or department makes of the broker, and charge them accordingly.

If you want to track the behavior of a particular message flow, you can set a unique accounting origin for this message flow, and analyze its activity over a specified period.

To make use of the accounting origin, you must perform the following tasks:

- Activate data collection, specifying the correct parameter to request basic support (the default is none, or no support). For details, see "**mqsichangeflowstats** command" on page 3744.
- Configure each message flow for which you want a specific origin to include ESQL statements that set the unique value that is to be associated with the data collected. Data for message flows for which a specific value has not been set are identified with the value `Anonymous`.

  The ESQL statements can be coded in a Compute, Database, or Filter node.

  You can configure the message flow either to set a fixed value, or to determine a dynamic value, and can therefore create a very flexible method of recording sets of data that are specific to particular messages or circumstances. For more information, refer to "Setting message flow accounting and statistics accounting origin" on page 3290.

You can complete these tasks in either order; if you configure the message flow before starting data collection, the broker ignores the setting. If you start data collection, specifying accounting origin support, before configuring the message flow, all data is collected with the Accounting Origin set to `Anonymous`. The broker acknowledges the origin when you redeploy the message flow. You can also modify data collection that has already started to request accounting origin support from the time that you issue the command. In both cases, data that has already been collected is written out and collection is restarted.

When data has been collected, you can review information for one or more specific origins. For example, if you select XML publication messages as your output format, you can start an application that subscribes to the origin in which you are interested.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Monitoring message flow performance" on page 3279
You can configure your message flow to collect statistics about the operation and behavior of your message flows. Use this information to assess the performance of your message flows.

"Setting message flow accounting and statistics accounting origin" on page 3290
When you request accounting origin support for collecting message flow accounting and statistics data on the **mqsichangeflowstats** command, you must also configure your message flows to provide the correct identification values that indicate what the data is associated with.

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Writing ESQL" on page 2413
How you can use ESQL to customize nodes.

Related reference:

"Message flow accounting and statistics data" on page 6723
You can collect message flow accounting and statistics data in various output formats.

"**mqsichangeflowstats** command" on page 3744
Use the **mqsichangeflowstats** command to control the accumulation of statistics about message flow operation.

Output formats for message flow accounting and statistics data:

When you collect message flow statistics, you can choose the output destination for the data.

Select one of the following destinations:
- User trace
- XML publication
- SMF

Statistics data is written to the specified output location in the following circumstances:
- When the archive data interval expires.
- When the snapshot interval expires.
- When the broker shuts down. Any data that has been collected by the broker, but has not yet been written to the specified output destination, is written during shutdown. It might therefore represent data for an incomplete interval.
- When any part of the broker configuration is redeployed. Redeployed configuration data might contain an updated configuration that is not consistent with the existing record structure (for example, a message flow might include an additional node, or an execution group might include a new message flow). Therefore the current data, which might represent an incomplete interval, is written to the output destination. Data collection continues for the redeployed configuration until you change data collection parameters or stop data collection.
- When data collection parameters are modified. If you update the parameters that you have set for data collection, all data that is collected for the message flow (or message flows) is written to the output destination to retain data integrity. Statistics collection is restarted according to the new parameters.
- When an error occurs that terminates data collection. You must restart data collection yourself in this case.

User trace

You can specify that the data that is collected is written to the user trace log. The data is written even when trace is switched off. The default output destination for accounting and statistics data is the user trace log. The data is written to one of the following locations:

Windows **Windows**

If you set the work path by using the **-w** parameter of the **mqsicreatebroker** command, the location is `workpath\log`.

If you have not specified the broker work path, the location is:

- On Windows:`%ALLUSERSPROFILE%\Application Data\IBM\MQSI\common\log`, where `%ALLUSERSPROFILE%` is the environment variable that defines the system working directory. The default directory depends on the operating system:
  - On Windows XP and Windows Server 2003: `C:\Documents and Settings\All Users\Application Data\IBM\MQSI\common\log`
  - On Windows Vista and Windows Server 2008: `C:\ProgramData\IBM\MQSI\common\log`

  The value might be different on your computer.

Linux UNIX **Linux and UNIX**

`/var/mqsi/common/log`

z/OS **z/OS**

`/component_filesystem/log`

XML publication

You can specify that the data that is collected is published. The publication message is created in XML format and is available to subscribers registered in the broker network that subscribe to the correct topic.

The topic on which the data is published has the following structure:

`$SYS/Broker/`*`brokerName`*`/StatisticsAccounting/`*`recordType`*`/`*`executionGroupLabel`*`/`*`messageFlowLabel`*

The variables correspond to the following values:

brokerName
The name of the broker for which statistics are collected.

recordType
Set to `SnapShot` or `Archive`, depending on the type of data to which you are subscribing. Alternatively, use # to register for both snapshot and archive data if it is being produced. This value is case sensitive and must be entered as `SnapShot`.

executionGroupLabel
The name of the execution group for which statistics are collected.

messageFlowLabel
The label on the message flow for which statistics are collected.

Subscribers can include filter expressions to limit the publications that they receive. For example, they can choose to see only snapshot data, or to see data that is collected for a single broker. Subscribers can specify wild cards (+ and #) to receive publications that refer to multiple resources.

The following examples show the topic with which a subscriber registers to receive different sorts of data:

- Register the following topic for the subscriber to receive data for all message flows running on *BrokerA*:

  $SYS/Broker/*BrokerA*/StatisticsAccounting/#

- Register the following topic for the subscriber to receive only archive statistics that relate to a message flow *Flow1* running on execution group *Execution* on broker *BrokerA*:

  $SYS/Broker/*BrokerA*/StatisticsAccounting/Archive/*Execution*/*Flow1*

- Register the following topic for the subscriber to receive both snapshot and archive data for message flow *Flow1* running on execution group *Execution* on broker *BrokerA*

  $SYS/Broker/*BrokerA*/StatisticsAccouting/#/*Execution*/*Flow1*

For help with registering your subscriber, see Message display, test and performance utilities SupportPac (IH03).

SMF

On z/OS, you can specify that the data collected is written to SMF. Accounting and statistics data uses SMF type 117 records. SMF supports the collection of data from multiple subsystems, and you might therefore be able to synchronize the information that is recorded from different sources.

To interpret the information that is recorded, use any utility program that processes SMF records.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Monitoring message flow performance" on page 3279
You can configure your message flow to collect statistics about the operation and behavior of your message flows. Use this information to assess the performance of your message flows.

Related reference:

"Message flow accounting and statistics data" on page 6723
You can collect message flow accounting and statistics data in various output formats.

"**mqsichangeflowstats** command" on page 3744
Use the **mqsichangeflowstats** command to control the accumulation of statistics about message flow operation.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsireportflowstats** command" on page 3929
Use the **mqsireportflowstats** command to display the current options for accounting and statistics that have been set using the **mqsichangeflowstats** command.

Starting to collect message flow accounting and statistics data

You can start collecting message flow accounting and statistics data for an active broker at any time. You can specify what type of statistics you want to collect, and where to send the data.

Before you begin

Before you start:

- Ensure that you have created a message flow; for more information, see "Creating a message flow" on page 1431.
- Ensure that you have deployed a broker archive (BAR) file; for more information, see "Deploying a broker archive file" on page 3235.
- Read the concept topic, "Message flow accounting and statistics collection options" on page 3282.

About this task

Select the granularity of the data that you want to be collected by specifying the appropriate parameters on the **mqsichangeflowstats** command. You must request statistics collection on a broker basis. If you want to collect information for more than one broker, you must issue the corresponding number of commands.

To start collecting message flow accounting and statistics data:

Procedure

1. Identify the broker for which you want to collect statistics .
2. Decide the resource for which you want to collect statistics. You can collect statistics for a specific execution group, or for all execution groups for the specified broker.
   - If you indicate a specific execution group, you can request that data is recorded for a specific message flow or all message flows in that group.
   - If you specify all execution groups, you must specify all message flows.
3. Decide if you want to collect thread-related statistics.
4. Decide if you want to collect node-related statistics. If you do, you can also collect information about terminals for the nodes.
5. Decide if you want to associate data collection with a particular accounting origin. This option is valid for snapshot and archive data, and for message flows and execution groups. However, when active, you must set its origin value in each message flow to which it refers. If you do not configure the participating message flows to set the appropriate origin identifier, the data collected for that message flow is collected with the origin set to `Anonymous`.

   For more information, see "Setting message flow accounting and statistics accounting origin" on page 3290.
6. Decide the target destination:
   - User trace log (the default setting). The output data can be processed using **mqsireadlog** and **mqsiformatlog**.
   - XML format publication message. If you chose this target destination, register the following topic for the subscriber:

`$SYS/Broker/`*`brokerName`*`/StatisticsAccounting/`*`recordType`*`/`*`executionGroupLabel`*`/`*`messageFlowLabel`*

     Where *brokerName*, *executionGroupLabel*, and *messageFlowLabel* represent the broker, execution group, and message flow on which you want to receive data. *recordType* is the type of data collection on which you want to receive publications (snapshot, archive, or the number sign (#) to receive both

snapshot and archive). The value that you specify for record type is case sensitive; therefore, if you choose to receive snapshot data, set the record type to `SnapShot`.

- z/OS SMF (on z/OS only)

7. Decide the type of data collection that you want:
   - Snapshot
   - Archive

   You can collect snapshot and archive data at the same time, but you have to configure them separately.
8. Issue the **mqsichangeflowstats** command with the appropriate parameters to reflect the decisions that you have made.

   For example, to turn on snapshot data for all message flows in the default execution group for BrokerA, and include node data with the basic message flow statistics, use the following command:

   ```
   mqsichangeflowstats BrokerA -s -e default -j -c active -n basic
   ```

   z/OS Using SDSF on z/OS, enter:

   ```
   /F BrokerA,cs s=yes,e=default,j=yes,c=active,n=basic
   ```

   For more examples, see "**mqsichangeflowstats** command" on page 3744.

Results

When the command completes successfully, data collection for the specified resources is started:

- If you have requested snapshot data, information is collected for approximately 20 seconds, and the results are written to the specified output.
- If you have requested archive data, information is collected for the interval defined for the broker (on the **mqsicreatebroker** or **mqsichangebroker** command, or by the external timer facility ENF). The results are written to the specified output, the interval is reset, and data collection starts again.

What to do next

Next:

You can now complete the following tasks:

- "Setting message flow accounting and statistics accounting origin" on page 3290
- "Stopping message flow accounting and statistics data collection" on page 3293
- "Viewing message flow accounting and statistics data collection parameters" on page 3294
- "Modifying message flow accounting and statistics data collection parameters" on page 3296
- "Resetting message flow accounting and statistics archive data" on page 3297

Related concepts:

"Message flow accounting and statistics data" on page 3281
Message flow accounting and statistics data is the information that can be collected by a broker to record performance and operating details of message flow execution.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Setting message flow accounting and statistics accounting origin"
When you request accounting origin support for collecting message flow accounting and statistics data on the **mqsichangeflowstats** command, you must also configure your message flows to provide the correct identification values that indicate what the data is associated with.

"Stopping message flow accounting and statistics data collection" on page 3293
You can stop collecting data for message flow accounting and statistics at any time. You do not have to stop the message flow, execution group, or broker to make this change, nor do you have to redeploy the message flow.

"Viewing message flow accounting and statistics data collection parameters" on page 3294
You can review and check the parameters that are currently in effect for message flow accounting and statistics data collection by using the **mqsireportflowstats** command.

"Modifying message flow accounting and statistics data collection parameters" on page 3296
You can modify the parameters that you have set for message flow accounting and statistics data collection. For example, you can start collecting data for a new message flow that you have deployed to an execution group for which you are already collecting data. You can modify parameters while data collection is active; you do not have to stop data collection and restart it.

"Resetting message flow accounting and statistics archive data" on page 3297
You can reset message flow accounting and statistics archive data to purge any accounting and statistics data not yet reported for that collecting interval. This removes unwanted data. You can request this at any time; you do not have to stop data collection and restart it to perform reset. You cannot reset snapshot data.

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"Issuing commands to the z/OS console" on page 3980
You operate the broker using the z/OS START, STOP, and MODIFY commands.

"**mqsichangeflowstats** command" on page 3744
Use the **mqsichangeflowstats** command to control the accumulation of statistics about message flow operation.

"**mqsireportflowstats** command" on page 3929
Use the **mqsireportflowstats** command to display the current options for accounting and statistics that have been set using the **mqsichangeflowstats** command.

"Message flow accounting and statistics data" on page 6723
You can collect message flow accounting and statistics data in various output formats.

Setting message flow accounting and statistics accounting origin:

When you request accounting origin support for collecting message flow accounting and statistics data on the **mqsichangeflowstats** command, you must also configure your message flows to provide the correct identification values that indicate what the data is associated with.

Before you begin

Before you start:

- Ensure that you have created a message flow; for more information, see "Creating a message flow" on page 1431.

- Read the concept topic, “Message flow accounting and statistics accounting origin” on page 3284

About this task

Accounting and statistics data is associated with an accounting origin. For more information, see “Message flow accounting and statistics data” on page 3281 and “Message flow accounting and statistics accounting origin” on page 3284.

You can set a different value for every message flow for which data collection is active, or the same value for a group of message flows (for example, those in a single execution group, or associated with a particular client, department, or application suite).

The accounting origin setting is not used until you deploy the message flow or flows to the brokers on which they are to run. You can activate data collection, or modify it to request accounting origin support, before or after you deploy the message flow. You do not have to stop collecting data when you deploy a message flow that changes accounting origin.

To configure a message flow to specify a particular accounting origin, complete the following steps.

Procedure

1. Open the message flow with which you want to work.
2. Click **Selection** above the palette of nodes.
3. Right-click a Compute, Database, or Filter node in the editor view, then click **Open ESQL**. The associated ESQL file is opened in the editor view, and your cursor is positioned at the start of the correct module. You can include the required ESQL in any of these nodes, so decide which node in each message flow is the most appropriate for this action.

   To take advantage of the accounting origin support, include one of these nodes in each message flow for which you want a specific origin set. If you have not configured one of these three nodes in the message flow, you must add one at a suitable point (for example, immediately following the input node) and connect it to other nodes in the flow.
4. Update the ESQL in the node module to set an accounting origin. The broker uses the origin identifier that is set in the Environment tree. You must set a value in the field with correlation name `Environment.Broker.Accounting.Origin`. This field is not created automatically in the Environment tree when the message is first received in the broker. The field is created only when you set it in an ESQL module that is associated with a node in the message flow.

   If you do not set a value in the message flow, the default value `Anonymous` is used for all output. If you set a value in more than one place in the message flow, the value that you set immediately before the message flow terminates is used in the output data.

   The code that you must add has the following form:

   ```
   SET Environment.Broker.Accounting.Origin = "value";
   ```

   You can set the identifier to a fixed value (as shown previously), or you can determine its value based on a dynamic value that is known only at run time. The value must be character data, and can be a maximum of 32 bytes. For example, you might set its value to the contents of a particular field in the

message that is being processed (if you are coding ESQL for a Compute node, you must use correlation name InputBody in place of Body in the following example):

```
IF Body.DepartmentName <> NULL THEN
   SET Environment.Broker.Accounting.Origin = Body.DepartmentName;
END IF;
```

5. Save the ESQL module, and check that you have not introduced any errors.
6. Save the message flow, and check again for errors.

Results

You are now ready to deploy the updated message flow; for more information, see "Deploying resources" on page 3234. Accounting and statistics data records that are collected after the message flow has been deployed includes the origin identifier that you have set.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message flow accounting and statistics data" on page 3281
Message flow accounting and statistics data is the information that can be collected by a broker to record performance and operating details of message flow execution.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Chapter 11, "Packaging and deploying," on page 3209
Package resources that you create in the WebSphere Message Broker Toolkit, such as message flows, and deploy them to execution groups on brokers.

"Modifying message flow accounting and statistics data collection parameters" on page 3296
You can modify the parameters that you have set for message flow accounting and statistics data collection. For example, you can start collecting data for a new message flow that you have deployed to an execution group for which you are already collecting data. You can modify parameters while data collection is active; you do not have to stop data collection and restart it.

"Writing ESQL" on page 2413
How you can use ESQL to customize nodes.

Related reference:

"**mqsichangeflowstats** command" on page 3744
Use the **mqsichangeflowstats** command to control the accumulation of statistics about message flow operation.

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"Filter node" on page 4452
Use the Filter node to route a message according to message content.

"Message flow accounting and statistics data" on page 6723
You can collect message flow accounting and statistics data in various output formats.

Stopping message flow accounting and statistics data collection

You can stop collecting data for message flow accounting and statistics at any time. You do not have to stop the message flow, execution group, or broker to make this change, nor do you have to redeploy the message flow.

Before you begin

Before you start:

- Start to collect message flow accounting and statistics data
- Read the concept topic about message flow accounting and statistics data

About this task

You can stop collecting data for message flow accounting and statistics at any time. You do not have to stop the message flow, execution group, or broker to make this change, nor do you have to redeploy the message flow.

You can modify the parameters that are currently in force for collecting message flow accounting and statistics data without stopping data collection. See "Modifying message flow accounting and statistics data collection parameters" on page 3296 for further details.

To stop collecting data:

Procedure

1. Check the resources for which you want to stop collecting data.

   You do not have to stop all active data collection. You can stop a subset of data collection. For example, if you started collecting statistics for all message flows in a particular execution group, you can stop doing so for a specific message flow in that execution group. Data collection for all other message flows in that execution group continues.

2. Issue the **mqsichangeflowstats** command with the appropriate parameters to stop collecting data for some or all resources.

   For example, to switch off snapshot data for all message flows in all execution groups for BrokerA, enter:

   ```
   mqsichangeflowstats BrokerA -s -g -j -c inactive
   ```

   z/OS Using SDSF on z/OS, enter:

   ```
   /F BrokerA,cs s=yes g=yes j=yes c=inactive
   ```

   Refer to the "**mqsichangeflowstats** command" on page 3744 for further examples.

Results

When the command completes successfully, data collection for the specified resources is stopped. Any outstanding data that has been collected is written to the output destination when you issue this command, to ensure the integrity of data collection.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message flow accounting and statistics data" on page 3281
Message flow accounting and statistics data is the information that can be collected by a broker to record performance and operating details of message flow execution.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Starting to collect message flow accounting and statistics data" on page 3288
You can start collecting message flow accounting and statistics data for an active broker at any time. You can specify what type of statistics you want to collect, and where to send the data.

"Modifying message flow accounting and statistics data collection parameters" on page 3296
You can modify the parameters that you have set for message flow accounting and statistics data collection. For example, you can start collecting data for a new message flow that you have deployed to an execution group for which you are already collecting data. You can modify parameters while data collection is active; you do not have to stop data collection and restart it.

Related reference:

"Message flow accounting and statistics data" on page 6723
You can collect message flow accounting and statistics data in various output formats.

"**mqsichangeflowstats** command" on page 3744
Use the **mqsichangeflowstats** command to control the accumulation of statistics about message flow operation.

"**mqsireportflowstats** command" on page 3929
Use the **mqsireportflowstats** command to display the current options for accounting and statistics that have been set using the **mqsichangeflowstats** command.

"Issuing commands to the z/OS console" on page 3980
You operate the broker using the z/OS START, STOP, and MODIFY commands.

Viewing message flow accounting and statistics data collection parameters

You can review and check the parameters that are currently in effect for message flow accounting and statistics data collection by using the **mqsireportflowstats** command.

Before you begin

Before you start:

- Start to collect message flow accounting and statistics data
- Read the concept topic about message flow accounting and statistics data

About this task

To view message flow accounting and statistics data collection parameters:

Procedure

Issue the **mqsireportflowstats** command with the appropriate parameters to view the parameters that are currently being used by the broker to control archive data collection or snapshot data collection.
You can view the parameters in force for a broker, an execution group, or an

individual message flow.
For example, to view parameters for snapshot data for all message flows in all execution groups for BrokerA, enter:

```
mqsireportflowstats BrokerA -s -g -j
```

z/OS Using SDSF on z/OS, enter:

```
/F BrokerA,rs s=yes,g=yes,j=yes
```

Refer to the "**mqsireportflowstats** command" on page 3929 for further examples.

Results

The command displays the current status, for example:

```
BIP8187I: Statistics Snapshot settings for flow MyFlow1 in execution
group default - On?: inactive,
ThreadDataLevel: basic, NodeDataLevel:  basic,
OutputFormat: usertrace, AccountingOrigin: basic
```

What to do next

Next:

You can now modify the data collection parameters.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message flow accounting and statistics data" on page 3281
Message flow accounting and statistics data is the information that can be collected by a broker to record performance and operating details of message flow execution.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Starting to collect message flow accounting and statistics data" on page 3288
You can start collecting message flow accounting and statistics data for an active broker at any time. You can specify what type of statistics you want to collect, and where to send the data.

"Modifying message flow accounting and statistics data collection parameters" on page 3296
You can modify the parameters that you have set for message flow accounting and statistics data collection. For example, you can start collecting data for a new message flow that you have deployed to an execution group for which you are already collecting data. You can modify parameters while data collection is active; you do not have to stop data collection and restart it.

Related reference:

"Message flow accounting and statistics data" on page 6723
You can collect message flow accounting and statistics data in various output formats.

"Issuing commands to the z/OS console" on page 3980
You operate the broker using the z/OS START, STOP, and MODIFY commands.

"**mqsichangeflowstats** command" on page 3744
Use the **mqsichangeflowstats** command to control the accumulation of statistics

about message flow operation.

"**mqsireportflowstats** command" on page 3929
Use the **mqsireportflowstats** command to display the current options for accounting and statistics that have been set using the **mqsichangeflowstats** command.

Modifying message flow accounting and statistics data collection parameters

You can modify the parameters that you have set for message flow accounting and statistics data collection. For example, you can start collecting data for a new message flow that you have deployed to an execution group for which you are already collecting data. You can modify parameters while data collection is active; you do not have to stop data collection and restart it.

Before you begin

Before you start:
- Start to collect message flow accounting and statistics data
- Read the concept topic about message flow accounting and statistics data

About this task

To modify message flow accounting and statistics parameters:

Procedure

1. Decide which data collection parameters you want to change. You can modify the parameters that are in force for a broker, an execution group, or an individual message flow.
2. Issue the **mqsichangeflowstats** command with the appropriate parameters to modify the parameters that are currently being used by the broker to control archive data collection or snapshot data collection.

   For example, to modify parameters to extend snapshot data collection to a new message flow MFlow2 in execution group EG2 for BrokerA, enter:

   ```
   mqsichangeflowstats BrokerA -s -e EG2 -f MFlow2 -c active
   ```

   z/OS Using SDSF on z/OS, enter:

   ```
   /F BrokerA,cs s=yes,e=EG2,f=MFlow2,c=active
   ```

   If you want to specify an accounting origin for archive data for a particular message flow in an execution group, enter:

   ```
   mqsichangeflowstats BrokerA -a -e EG4 -f MFlowX -b basic
   ```

   z/OS Using SDSF on z/OS, enter:

   ```
   /F BrokerA,cs a=yes,e=EG4,f=MFlowX,b=basic
   ```

   Refer to the "**mqsichangeflowstats** command" on page 3744 for further examples.

Results

When the command completes successfully, the new parameters that you have specified for data collection are in force. These parameters remain in force until you stop data collection or make further modifications.

Related concepts:

"Message flow accounting and statistics data" on page 3281
Message flow accounting and statistics data is the information that can be collected by a broker to record performance and operating details of message flow execution.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Starting to collect message flow accounting and statistics data" on page 3288
You can start collecting message flow accounting and statistics data for an active broker at any time. You can specify what type of statistics you want to collect, and where to send the data.

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"Message flow accounting and statistics data" on page 6723
You can collect message flow accounting and statistics data in various output formats.

"Issuing commands to the z/OS console" on page 3980
You operate the broker using the z/OS START, STOP, and MODIFY commands.

"**mqsichangeflowstats** command" on page 3744
Use the **mqsichangeflowstats** command to control the accumulation of statistics about message flow operation.

"**mqsireportflowstats** command" on page 3929
Use the **mqsireportflowstats** command to display the current options for accounting and statistics that have been set using the **mqsichangeflowstats** command.

Resetting message flow accounting and statistics archive data

You can reset message flow accounting and statistics archive data to purge any accounting and statistics data not yet reported for that collecting interval. This removes unwanted data. You can request this at any time; you do not have to stop data collection and restart it to perform reset. You cannot reset snapshot data.

Before you begin

Before you start:
- Start to collect message flow accounting and statistics data
- Read the concept topic about message flow accounting and statistics data

About this task

To reset message flow accounting and statistics archive data:

Procedure

1. Identify the broker, and optionally the execution group, for which you want to reset archive data. You cannot reset archive data on a message flow basis.
2. Issue the **mqsichangeflowstats** command with the appropriate parameters to reset archive data.

   For example, to reset archive data for BrokerA, enter:

   ```
   mqsichangeflowstats BrokerA -a -g -j -r
   ```

   z/OS Using SDSF on z/OS, enter:

```
/F BrokerA,cs a=yes,g=yes,j=yes,r=yes
```

Results

When this command completes, all accounting and statistics data accumulated so far for this interval are purged and will not be included in the reports. Data collection is restarted from this point. All archive data for all flows (indicated by `-j` or `j=yes`) in all execution groups (indicated by `-g` or `g=yes`) is reset.

This command has a minimal effect on snapshot data because the accumulation interval is much shorter than for archive data. It does not effect the settings for archive or snapshot data collection that are currently in force. When the command has completed, data collection resumes according to the current settings.

You can change any other options that are currently in effect when you reset archive data, for example accounting origin settings or output type.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message flow accounting and statistics data" on page 3281
Message flow accounting and statistics data is the information that can be collected by a broker to record performance and operating details of message flow execution.

Related tasks:

"Starting to collect message flow accounting and statistics data" on page 3288
You can start collecting message flow accounting and statistics data for an active broker at any time. You can specify what type of statistics you want to collect, and where to send the data.

"Modifying message flow accounting and statistics data collection parameters" on page 3296
You can modify the parameters that you have set for message flow accounting and statistics data collection. For example, you can start collecting data for a new message flow that you have deployed to an execution group for which you are already collecting data. You can modify parameters while data collection is active; you do not have to stop data collection and restart it.

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"Message flow accounting and statistics data" on page 6723
You can collect message flow accounting and statistics data in various output formats.

"Issuing commands to the z/OS console" on page 3980
You operate the broker using the z/OS START, STOP, and MODIFY commands.

"**mqsichangeflowstats** command" on page 3744
Use the **mqsichangeflowstats** command to control the accumulation of statistics about message flow operation.

"**mqsireportflowstats** command" on page 3929
Use the **mqsireportflowstats** command to display the current options for accounting and statistics that have been set using the **mqsichangeflowstats** command.

Starting accounting and statistics data collection in the WebSphere Message Broker Explorer

Use the WebSphere Message Broker Explorer to start collecting snapshot accounting and statistics data for your brokers, execution groups, and message flows. You can then view the accounting and statistics data in the Broker Statistics and Broker Statistics Graph views.

Before you begin

Before you start:

- Ensure that you have created a message flow; for more information, see "Creating a message flow" on page 1431.
- Ensure that you have deployed a broker archive (BAR) file; for more information, see "Deploying a broker archive file" on page 3235.
- Read the concept topic, "Message flow accounting and statistics data" on page 3281.

About this task

Use the snapshot accounting and statistics data displayed in the Broker Statistics and Broker Statistics Graph views to monitor the performance and resource usage of your broker or execution group at the message flow, node, or terminal level.

You can start collecting accounting and statistics data for an active broker at any time. You can start collecting accounting and statistics data for multiple brokers at the same time.

To start collecting snapshot message flow accounting and statistics data by using the WebSphere Message Broker Explorer:

Procedure

1. In the WebSphere MQ Explorer - Navigator view, expand the Brokers folder.
2. Right-click the execution group or message flow for which you want to collect statistics.
   - If you selected an execution group, click **Statistics All Flows** > **Start Statistics**.
   - If you selected a message flow click **Statistics** > **Start Statistics**.

   A message is sent to the broker to start collecting accounting and statistics data for the selected resource.
3. Click **Window** > **Show View** > **Broker Statistics** to open the Broker Statistics and Broker Statistics Graph view. These two views are displayed together. If you close one of the views, the other view is also closed.

Results

Accounting and statistics data for the selected broker, execution group, or message flow is displayed in the Broker Statistics and Broker Statistics Graph views. It can take up to 30 seconds for accounting and statistics data to be received from the broker and displayed in the Broker Statistics Graph view. The message `Waiting for Data` is displayed in the title bar is displayed until accounting and statistics data is received from the selected broker, execution group, or message flow. When data is received from the broker, it is displayed in numeric form in the Broker Statistics view, and a visual representation of this data is shown in the Broker Statistics Graph view. Select individual or multiple items in the Broker Statistics

view to change the information displayed in the Broker Statistics Graph view.

What to do next

Next:

To examine the current data in the two views, click **Pause** in the Broker Statistics Graph view to prevent the current data from being overwritten with new data. Click **Play** to resume displaying more accounting and statistics data. You can change the visual representation of the data by clicking **Linear**, **Logarithmic**, and **Stacked**.

You can also select the metrics that are displayed in the graph view; for more information, see "Filtering message flow accounting and statistics data" on page 3302.

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

"Message flow accounting and statistics data" on page 3281
Message flow accounting and statistics data is the information that can be collected by a broker to record performance and operating details of message flow execution.

Related tasks:

"Viewing message flow accounting and statistics data"
You can use the Broker Statistics and Broker Statistics Graph views in the WebSphere Message Broker Explorer to view snapshot accounting and statistics data as it is produced by the broker.

"Stopping accounting and statistics data collection in the WebSphere Message Broker Explorer" on page 3304
Use the WebSphere Message Broker Explorer to stop collecting accounting and statistics data for your brokers, execution groups, and message flows.

"Filtering message flow accounting and statistics data" on page 3302
You can select the metrics for the snapshot accounting and statistics data that are displayed in the Broker Statistics and Broker Statistics Graph views in the WebSphere Message Broker Explorer.

Related reference:

"Metrics for accounting and statistics data in the WebSphere Message Broker Explorer" on page 6744
You can filter the metrics displayed for accounting and statistics data in the Broker Statistics and Broker Statistics Graph views in the WebSphere Message Broker Explorer.

Viewing message flow accounting and statistics data

You can use the Broker Statistics and Broker Statistics Graph views in the WebSphere Message Broker Explorer to view snapshot accounting and statistics data as it is produced by the broker.

Before you begin

Before you start:

- "Starting accounting and statistics data collection in the WebSphere Message Broker Explorer" on page 3299
- Read the concept topic about message flow accounting and statistics data

About this task

Use the accounting and statistics data displayed in the Broker Statistics and Broker Statistics Graph views to monitor the performance and resource usage of your broker or execution group at the message flow, node or terminal level. The WebSphere Message Broker Explorer displays snapshot accounting and statistics data.

When you open the Broker Statistics and Broker Statistics Graph views, the WebSphere Message Broker Explorer connects to all the brokers that you have in your workspace and attempts to collect statistics data from the brokers. The message `Waiting for Data` is displayed in the title bar is displayed until accounting and statistics data are received from the selected broker, execution group or message flow. When data is received from the broker it is displayed in numeric form in the Broker Statistics view, and a visual representation of this data is shown in the Broker Statistics Graph view.

To view message flow accounting and statistics data using the WebSphere Message Broker Explorer:

Procedure

1. Click **Window** > **Show View** > **Broker Statistics** to open the Broker Statistics and Broker Statistics Graph view. These two views are displayed together. If you close one of the views, the other view is also be closed.
2. In the **WebSphere MQ Explorer - Navigator** view select the broker, execution group or message flow for which you want to view accounting and statistics data.

Results

Accounting and statistics data for the selected broker, execution group, or message flow is displayed in the Broker Statistics and Broker Statistics Graph views. It can take up to thirty seconds for accounting and statistics data to be received from the broker and displayed in the Broker Statistics Graph view. Select individual or multiple items in the Broker Statistics view to change the information displayed in the Broker Statistics Graph view.

What to do next

If you want to examine the current data in the two views, click the **Pause** button in the Broker Statistics Graph view to prevent the current data from being overwritten with new data. Click the **Play** button to resume displaying new accounting and statistics data. You can change the visual representation of the data by clicking on the **Linear**, **Logarithmic**, and **Stacked** buttons.

You can also select the metrics that are displayed in the graph view, see “Filtering message flow accounting and statistics data” on page 3302.

Related concepts:

“Message flow accounting and statistics data” on page 3281
Message flow accounting and statistics data is the information that can be collected by a broker to record performance and operating details of message flow execution.

Related tasks:

"Starting accounting and statistics data collection in the WebSphere Message Broker Explorer" on page 3299
Use the WebSphere Message Broker Explorer to start collecting snapshot accounting and statistics data for your brokers, execution groups, and message flows. You can then view the accounting and statistics data in the Broker Statistics and Broker Statistics Graph views.

"Filtering message flow accounting and statistics data"
You can select the metrics for the snapshot accounting and statistics data that are displayed in the Broker Statistics and Broker Statistics Graph views in the WebSphere Message Broker Explorer.

"Starting to collect message flow accounting and statistics data" on page 3288
You can start collecting message flow accounting and statistics data for an active broker at any time. You can specify what type of statistics you want to collect, and where to send the data.

"Modifying message flow accounting and statistics data collection parameters" on page 3296
You can modify the parameters that you have set for message flow accounting and statistics data collection. For example, you can start collecting data for a new message flow that you have deployed to an execution group for which you are already collecting data. You can modify parameters while data collection is active; you do not have to stop data collection and restart it.

Related reference:

"**mqsichangeflowstats** command" on page 3744
Use the **mqsichangeflowstats** command to control the accumulation of statistics about message flow operation.

"**mqsireportflowstats** command" on page 3929
Use the **mqsireportflowstats** command to display the current options for accounting and statistics that have been set using the **mqsichangeflowstats** command.

"Issuing commands to the z/OS console" on page 3980
You operate the broker using the z/OS START, STOP, and MODIFY commands.

Filtering message flow accounting and statistics data

You can select the metrics for the snapshot accounting and statistics data that are displayed in the Broker Statistics and Broker Statistics Graph views in the WebSphere Message Broker Explorer.

Before you begin

Before you start:

- Read the concept topic about message flow accounting and statistics data
- "Starting accounting and statistics data collection in the WebSphere Message Broker Explorer" on page 3299
- "Viewing message flow accounting and statistics data" on page 3300

About this task

When you open the Broker Statistics and Broker Statistics Graph views, the WebSphere Message Broker Explorer connects to all the brokers that you have in your workspace and attempts to collect snapshot statistics data from the brokers. The message `Waiting for Data` is displayed in the title bar until accounting and statistics data are received from the selected broker, execution group, or message flow.

When data is received from the broker, it is displayed in numeric form in the Broker Statistics view, and a visual representation of this data is shown in the Broker Statistics Graph view. When accounting and data statistics data has started to be collected and displayed for your broker, execution group, or message flow, you can select the metrics that are displayed in the Broker Statistics and Broker Statistics Graph views.

The metrics that you can select depend on whether you are viewing the accounting and statistics data from a broker, an execution group, or a message flow. To view a list of the metrics that you can filter on, see "Metrics for accounting and statistics data in the WebSphere Message Broker Explorer" on page 6744.

To filter the message flow accounting and statistics data in the Broker Statistics and Broker Statistics Graph views:

Procedure

1. Click **Window** > **Show View** > **Broker Statistics** to open the Broker Statistics and Broker Statistics Graph view.
2. In the **WebSphere MQ Explorer - Navigator** view, select the broker, execution group, or message flow for which you want to view accounting and statistics data.
3. Click the **Filter** button in the Broker Statistics Graph view. The Select metrics to graph window is displayed.
4. Select the metrics that you want to display from the list in the Broker Statistics and Broker Statistics Graph view. Clear all the metrics that you do not want to display in the Broker Statistics and Broker Statistics Graph view.
5. Click **OK**.

Results

The selected metrics are displayed in the Broker Statistics and Broker Statistics Graph views. Data for additional metrics that you have selected are displayed the next time the statistics data is refreshed.

Related concepts:

"Message flow accounting and statistics data" on page 3281
Message flow accounting and statistics data is the information that can be collected by a broker to record performance and operating details of message flow execution.

Related tasks:

"Starting accounting and statistics data collection in the WebSphere Message Broker Explorer" on page 3299
Use the WebSphere Message Broker Explorer to start collecting snapshot accounting and statistics data for your brokers, execution groups, and message flows. You can then view the accounting and statistics data in the Broker Statistics and Broker Statistics Graph views.

"Viewing message flow accounting and statistics data" on page 3300
You can use the Broker Statistics and Broker Statistics Graph views in the WebSphere Message Broker Explorer to view snapshot accounting and statistics data as it is produced by the broker.

"Starting to collect message flow accounting and statistics data" on page 3288
You can start collecting message flow accounting and statistics data for an active broker at any time. You can specify what type of statistics you want to collect, and where to send the data.

"Modifying message flow accounting and statistics data collection parameters" on page 3296
You can modify the parameters that you have set for message flow accounting and statistics data collection. For example, you can start collecting data for a new message flow that you have deployed to an execution group for which you are already collecting data. You can modify parameters while data collection is active; you do not have to stop data collection and restart it.

Related reference:

"**mqsichangeflowstats** command" on page 3744
Use the **mqsichangeflowstats** command to control the accumulation of statistics about message flow operation.

"**mqsireportflowstats** command" on page 3929
Use the **mqsireportflowstats** command to display the current options for accounting and statistics that have been set using the **mqsichangeflowstats** command.

"Issuing commands to the z/OS console" on page 3980
You operate the broker using the z/OS START, STOP, and MODIFY commands.

Stopping accounting and statistics data collection in the WebSphere Message Broker Explorer

Use the WebSphere Message Broker Explorer to stop collecting accounting and statistics data for your brokers, execution groups, and message flows.

About this task

You can stop collecting data for message flow accounting and statistics at any time. You do not have to stop the message flow, execution group, or broker to make this change, nor do you have to redeploy the message flow. You do not have to stop all active data collection. You can stop a subset of data collection. For example, if you started collecting statistics for all message flows in a particular execution group, you can stop doing so for a specific message flow in that execution group. Data collection for all other message flows in that execution group continues.

To stop collecting message flow accounting and statistics data:

Procedure

1. In the WebSphere MQ Explorer - Navigator view, expand the Brokers folder.
2. Right-click the execution group or message flow for which you want to stop collecting statistics.
   - If you selected an execution group, click **Statistics All Flows** > **Stop Statistics**.
   - If you selected a message flow click **Statistics** > **Stop Statistics**.

Results

A message is sent to the broker to stop collecting accounting and statistics data for the selected resource.

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

"Message flow accounting and statistics data" on page 3281
Message flow accounting and statistics data is the information that can be collected by a broker to record performance and operating details of message flow execution.

Related tasks:

"Viewing message flow accounting and statistics data" on page 3300
You can use the Broker Statistics and Broker Statistics Graph views in the WebSphere Message Broker Explorer to view snapshot accounting and statistics data as it is produced by the broker.

"Starting accounting and statistics data collection in the WebSphere Message Broker Explorer" on page 3299
Use the WebSphere Message Broker Explorer to start collecting snapshot accounting and statistics data for your brokers, execution groups, and message flows. You can then view the accounting and statistics data in the Broker Statistics and Broker Statistics Graph views.

"Filtering message flow accounting and statistics data" on page 3302
You can select the metrics for the snapshot accounting and statistics data that are displayed in the Broker Statistics and Broker Statistics Graph views in the WebSphere Message Broker Explorer.

Related reference:

"Metrics for accounting and statistics data in the WebSphere Message Broker Explorer" on page 6744
You can filter the metrics displayed for accounting and statistics data in the Broker Statistics and Broker Statistics Graph views in the WebSphere Message Broker Explorer.

Monitoring resource performance

You can collect statistics to assess the performance of certain resources used by execution groups.

Before you begin

Before you start:

Read the concept topic about resource statistics.

About this task

- "Starting resource statistics collection" on page 3307
- "Stopping resource statistics collection" on page 3308
- "Viewing status of resource statistics collection" on page 3309
- "Starting resource statistics collection in the WebSphere Message Broker Explorer" on page 3310
- "Stopping resource statistics collection in the WebSphere Message Broker Explorer" on page 3312
- "Viewing resource statistics data in the WebSphere Message Broker Explorer" on page 3313
- "Viewing the status of resource statistics collection in the WebSphere Message Broker Explorer" on page 3317

You can also work with resource statistics from CMP applications. See "Working with resource statistics in a CMP application" on page 998 for further details.

Related concepts:

"Resource statistics"
Resource statistics are collected by a broker to record performance and operating details of resources that are used by execution groups.

Related reference:

"Resource statistics data" on page 6745
Learn about the measurements for which data is returned when you activate resource statistics collection.

"**mqsichangeresourcestats** command" on page 3819
Use the **mqsichangeresourcestats** command to control statistics gathering for resources in the broker.

"**mqsireportresourcestats** command" on page 3944
Use the **mqsireportresourcestats** command to display current statistics gathering options for resources in the broker.

Resource statistics

Resource statistics are collected by a broker to record performance and operating details of resources that are used by execution groups.

As a system administrator, you can use the resource statistics to ensure that your systems are using the available resources in the most efficient manner. By monitoring systems and analyzing statistical trends, you can keep system resource usage within boundaries that you consider acceptable, and help to pre-empt situations where system resources are overstretched and might become unavailable. Analysis of the data returned potentially requires specialist skills and knowledge of each resource type.

If you detect that system resources are under pressure, you can examine the statistics collected by the broker to assess whether the cause of the concern is the use of those resources by processes in WebSphere Message Broker.

You must activate statistics collection; by default, collection is not active. If you activate statistics, you might experience a minor degradation in operating performance of the broker or brokers for which you are collecting data. You can collect data on one or more execution groups, as well as all execution groups on a broker, so that you can limit the statistics gathering activity if appropriate.

Resource statistics complement the accounting and statistics data that you can collect on message flows, which are also available in the WebSphere Message Broker Explorer; for details, see "Monitoring message flow performance" on page 3279.

To start, stop, or check the status of resource statistics collection, use one or more of the following options:

- The WebSphere Message Broker Explorer
- The **mqsichangeresourcestats** and **mqsireportresourcestats** commands
- A CMP application

To view the output that is generated by statistics collection, use one or more of the following options:

- The WebSphere Message Broker Explorer; numeric data and graphs are displayed for each execution group for which you have activated statistics collection.

- An application that subscribes to a publication, in the form of an XML message, that is published by the broker every 20 seconds. The message contains the data collected for each execution group for which you have activated statistics collection.

  The topic for each message has the following structure:

  `$SYS/Broker/`*`broker_name`*`/ResourceStatistics/`*`execution_group_name`*

  You can set up subscriptions for a specific execution group on a specific broker. For example:

  `$SYS/Broker/MB7BROKER/ResourceStatistics/default`

  You can also use wildcards in the subscriptions to broaden the scope of what is returned. For example, to subscribe to reports for all execution groups on all brokers, use the following values:

  `$SYS/Broker/+/ResourceStatistics/#`

For details of all the statistics that are reported for each resource manager, and the publication content, see "Resource statistics data" on page 6745.

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

Related tasks:

"Monitoring resource performance" on page 3305
You can collect statistics to assess the performance of certain resources used by execution groups.

"Subscribing to statistics reports" on page 3317
You can subscribe to topics that return statistics about the operation of your message flows and resource managers.

"Working with resource statistics in a CMP application" on page 998
Start, stop, and review status of resource statistics collection in your CMP applications.

Related reference:

"Resource statistics data" on page 6745
Learn about the measurements for which data is returned when you activate resource statistics collection.

"Special characters in topics" on page 6395
A topic can contain any character in the Unicode character set, but some characters have a special meaning.

"**mqsichangeresourcestats** command" on page 3819
Use the **mqsichangeresourcestats** command to control statistics gathering for resources in the broker.

"**mqsireportresourcestats** command" on page 3944
Use the **mqsireportresourcestats** command to display current statistics gathering options for resources in the broker.

Starting resource statistics collection

Use the **mqsichangeresourcestats** command to start collecting resource statistics.

Before you begin

Before you start:

- Ensure that you have created a message flow; for more information, see "Creating a message flow" on page 1431.

- Ensure that you have deployed a broker archive (BAR) file; for more information, see "Deploying a broker archive file" on page 3235.
- Read the concept topic, "Resource statistics" on page 3306.

About this task

Use resource statistics data to monitor the performance and resource usage of your execution groups. You can start collecting data for active execution groups at any time.

If you prefer, you can start collecting resource statistics by using the WebSphere Message Broker Explorer; see "Starting resource statistics collection in the WebSphere Message Broker Explorer" on page 3310.

To start resource statistics collection:

Procedure

1. If your broker is running on Linux, UNIX, or Windows systems, set up the correct command environment. For details of how to complete this task, see "Setting up a command environment" on page 213.
2. Decide whether you want to collect statistics for a specific execution group, or for all execution groups on the broker.
3. Run the **mqsichangeresourcestats** command with the appropriate parameters.

   For example, to start collecting resource statistics for the default execution group for BrokerA, enter the following command:

   ```
   mqsichangeresourcestats BrokerA -c active -e default
   ```

   For further examples, see "**mqsichangeresourcestats** command" on page 3819.

Related concepts:

"Resource statistics" on page 3306
Resource statistics are collected by a broker to record performance and operating details of resources that are used by execution groups.

Related tasks:

"Stopping resource statistics collection"
Use the **mqsichangeresourcestats** command to stop collecting resource statistics.

"Viewing status of resource statistics collection" on page 3309
Use the **mqsireportresourcestats** command to view the status of resource statistics collection.

Related reference:

"Resource statistics data" on page 6745
Learn about the measurements for which data is returned when you activate resource statistics collection.

"**mqsichangeresourcestats** command" on page 3819
Use the **mqsichangeresourcestats** command to control statistics gathering for resources in the broker.

Stopping resource statistics collection

Use the **mqsichangeresourcestats** command to stop collecting resource statistics.

Before you begin

Before you start:

- Read the concept topic about resource statistics.
- Start execution group resource statistics collection.

About this task

You can stop collecting data for an active execution group at any time.

If you prefer, you can stop collecting resource statistics by using the WebSphere Message Broker Explorer; see "Stopping resource statistics collection in the WebSphere Message Broker Explorer" on page 3312.

To stop resource statistics collection:

Procedure

1. If your broker is running on Linux, UNIX, or Windows systems, set up the correct command environment. For details of how to complete this task, see "Setting up a command environment" on page 213.
2. Decide whether you want to stop collection for a specific execution group, or for all execution groups on the broker. The way in which you started collection is not important; you can stop collection for one execution group, even if you started it for all execution groups. You can also stop collection for all execution groups, even if you started only one, or each one separately.
3. Run the **mqsichangeresourcestats** command with the appropriate parameters.

   For example, to stop collecting resource statistics for the default execution group for BrokerA, enter:

   ```
   mqsichangeresourcestats BrokerA -c inactive -e default
   ```

   See the "**mqsichangeresourcestats** command" on page 3819 for further examples.

Related concepts:

"Resource statistics" on page 3306
Resource statistics are collected by a broker to record performance and operating details of resources that are used by execution groups.

Related tasks:

"Starting resource statistics collection" on page 3307
Use the **mqsichangeresourcestats** command to start collecting resource statistics.

"Viewing status of resource statistics collection"
Use the **mqsireportresourcestats** command to view the status of resource statistics collection.

Related reference:

"Resource statistics data" on page 6745
Learn about the measurements for which data is returned when you activate resource statistics collection.

"**mqsichangeresourcestats** command" on page 3819
Use the **mqsichangeresourcestats** command to control statistics gathering for resources in the broker.

Viewing status of resource statistics collection

Use the **mqsireportresourcestats** command to view the status of resource statistics collection.

Before you begin

Before you start:

- Read the concept topic about resource statistics.

About this task

The command indicates whether resource statistics collection is active or inactive for each execution group that you have specified on the command.

If you prefer, you can view resource statistics collection status by using the WebSphere Message Broker Explorer; see "Viewing the status of resource statistics collection in the WebSphere Message Broker Explorer" on page 3317.

To view resource statistics collection status:

Procedure

1. If your broker is running on Linux, UNIX, or Windows systems, set up the correct command environment. For details of how to complete this task, see "Setting up a command environment" on page 213.
2. Decide whether you want to view status for a specific execution group, or for all execution groups on the broker.
3. Run the **mqsireportresourcestats** command with the appropriate parameters.

   For example, to view status for the default execution group on BrokerA, enter:

   ```
   mqsireportresourcestats BrokerA -e default
   ```

   See the "**mqsireportresourcestats** command" on page 3944 for further examples.

Related concepts:

"Resource statistics" on page 3306
Resource statistics are collected by a broker to record performance and operating details of resources that are used by execution groups.

Related tasks:

"Starting resource statistics collection" on page 3307
Use the **mqsichangeresourcestats** command to start collecting resource statistics.

"Stopping resource statistics collection" on page 3308
Use the **mqsichangeresourcestats** command to stop collecting resource statistics.

Related reference:

"Resource statistics data" on page 6745
Learn about the measurements for which data is returned when you activate resource statistics collection.

"**mqsichangeresourcestats** command" on page 3819
Use the **mqsichangeresourcestats** command to control statistics gathering for resources in the broker.

"**mqsireportresourcestats** command" on page 3944
Use the **mqsireportresourcestats** command to display current statistics gathering options for resources in the broker.

Starting resource statistics collection in the WebSphere Message Broker Explorer

Use the WebSphere Message Broker Explorer to start collecting resource statistics data for your execution groups. You can then view the data in the Broker Resources and Broker Resources Graph views.

Before you begin

Before you start:

- Ensure that you have created a message flow; for more information, see "Creating a message flow" on page 1431.
- Ensure that you have deployed a broker archive (BAR) file; for more information, see "Deploying a broker archive file" on page 3235.
- Read the concept topic, "Resource statistics" on page 3306.

About this task

Use the resource statistics data to monitor the performance and resource usage in your execution groups. You can start collecting data for active execution groups at any time.

If you prefer, you can start resource statistics collection by using the **mqsichangeresourcestats** command; for more information, see "Starting resource statistics collection" on page 3307.

To start resource statistics collection in the WebSphere Message Broker Explorer, complete the following steps.

Procedure

1. In the WebSphere MQ Explorer - Navigator view, expand the Brokers folder, then expand the folder of the broker with which you are working.
2. Right-click the execution group for which you want to collect statistics. If you want statistics for several execution groups on this broker, select more than one by using standard operating system interfaces. For example, on Windows systems, hold down the **Ctrl** key and select each execution group before you right-click to open the menu.
3. Click **Statistics** > **Start Resource Statistics**. A message is sent to the broker to start collecting data for the selected execution groups. The property Resource Statistics Active is updated in the properties view of each affected execution group to indicate that data collection is now active.

   A warning message is displayed in the execution group properties window to warn you that performance might be affected by your action.
4. To view statistics, click **Window** > **Show View** > **Resource Statistics** to open the Broker Resources and Broker Resources Graph views. If you are displaying statistics for this execution group for the first time, the views might be empty until the first data is received. Update messages are sent every 20 seconds, and the views refresh automatically.

   The Broker Resources and Broker Resources Graph views are displayed together. If you close one of the views, the other view is also closed.

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

"Resource statistics" on page 3306
Resource statistics are collected by a broker to record performance and operating details of resources that are used by execution groups.

Related tasks:

"Stopping resource statistics collection in the WebSphere Message Broker Explorer" on page 3312
Use the WebSphere Message Broker Explorer to stop collecting resource statistics data for your execution groups.

"Viewing resource statistics data in the WebSphere Message Broker Explorer" on page 3313
Use the WebSphere Message Broker Explorer to view resource statistics data for your execution groups in the Broker Resources and Broker Resources Graph views.

Related reference:

"Resource statistics data" on page 6745
Learn about the measurements for which data is returned when you activate resource statistics collection.

Stopping resource statistics collection in the WebSphere Message Broker Explorer

Use the WebSphere Message Broker Explorer to stop collecting resource statistics data for your execution groups.

Before you begin

Before you start:
- Read the concept topic about resource statistics.
- Start resource statistics collection

About this task

You can stop collecting data for active execution groups at any time.

If you prefer, you can stop collecting resource statistics by using the **mqsichangeresourcestats** command; see "Stopping resource statistics collection" on page 3308.

To stop collecting resource statistics in the WebSphere Message Broker Explorer:

Procedure

1. In the **WebSphere MQ Explorer - Navigator** view, expand the Brokers folder.
2. Right-click the broker or execution group for which you want to stop statistics collection. If you want to stop statistics collection for several execution groups, you can select more than one by using standard operating system interfaces; for example, on Windows systems, hold down the Ctrl key and select each execution group before you right-click to open the menu.
3. Click **Statistics** > **Stop Resource Statistics**. A message is sent to the broker to stop collecting resource data for the selected execution groups. The status of data collection is updated in the properties view for each affected execution group.

Results

If you click **Window** > **Show View** > **Resource Statistics** to open the Broker Resources and Broker Resources Graph views when statistics collection is not active, the data that is displayed represents the last update message received by the WebSphere Message Broker Explorer. If you have never started statistics collection for this execution group, the views are displayed but contain no data.

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

"Resource statistics" on page 3306
Resource statistics are collected by a broker to record performance and operating details of resources that are used by execution groups.

Related tasks:

"Starting resource statistics collection in the WebSphere Message Broker Explorer" on page 3310
Use the WebSphere Message Broker Explorer to start collecting resource statistics data for your execution groups. You can then view the data in the Broker Resources and Broker Resources Graph views.

"Viewing resource statistics data in the WebSphere Message Broker Explorer"
Use the WebSphere Message Broker Explorer to view resource statistics data for your execution groups in the Broker Resources and Broker Resources Graph views.

Related reference:

"Resource statistics data" on page 6745
Learn about the measurements for which data is returned when you activate resource statistics collection.

Viewing resource statistics data in the WebSphere Message Broker Explorer

Use the WebSphere Message Broker Explorer to view resource statistics data for your execution groups in the Broker Resources and Broker Resources Graph views.

Before you begin

Before you start:

- Read the concept topic, "Resource statistics" on page 3306.
- Start resource statistics collection: "Starting resource statistics collection in the WebSphere Message Broker Explorer" on page 3310

About this task

You can also view resource statistics collection by subscribing to the topic on which statistics are published. For further details, see "Subscribing to statistics reports" on page 3317.

To view resource statistics in the WebSphere Message Broker Explorer, complete the following steps.

Procedure

1. In the **WebSphere MQ Explorer - Navigator** view, expand the Brokers folder.
2. To open the Resource Statistics and Resource Statistics Graph views, click **Window** > **Show View** > **Resource Statistics**. These two views are displayed together. If you close one of the views, the other view is also closed.
3. Use the information in these views to review the use of resources for which statistics are available.

   The following examples demonstrate the types of question that can be answered by collecting resource statistics. This list is not exhaustive, and does include all resource types. For a full list of resource types, and the type of information that is collected for each one, see "Resource statistics data" on page 6745

   JVM statistics

   How much memory is the JVM using?

Many tools that are specific to an operating system give you the total memory that is used by the execution group, but they do not show you how that memory is divided between Java processing and other processing in the execution group. By looking at the field `CommittedMemoryInMB`, you can see how much memory is currently allocated to the JVM. Then look at the field `MaxMemoryInMB` to see the maximum amount of memory that can be allocated.

How often is garbage collection done? Is it affecting the performance of the execution group?

To see how often the JVM is doing garbage collection, check the `CumulativeNumberOfGCCollections` field to see if the rate of collections is increasing. Garbage collection is a normal process, and is therefore expected to some degree. However, excessive garbage collection can affect performance.

To see if current garbage collection is excessive, monitor the `CumulativeGCTimeInSeconds` value. If this value is increasing by more than 2 seconds in each 20-second statistics interval, try increasing the JVM maximum heap size for your execution group by using the **mqsichangeproperties** command. You might also want to inspect all the Java user-defined nodes and JavaCompute nodes that are included in your deployed message flows, to ensure that they do not create and delete many objects that could be reused; frequent deletions can contribute to excessive garbage collection.

Do I need to change the minimum or maximum heap sizes?

- If the `CumulativeGCTimeInSeconds` value is increasing by more than 2 seconds in each 20 second statistics interval, increase the maximum heap size to reduce this increase.
- If the `UsedMemoryInMB` value is never close to the `InitialMemoryInMB` value, you might have allocated more memory for the heap than is required. Therefore, reduce the JVM minimum heap size value for the execution group to a value that is closer to the `UsedMemoryInMB` value.

Change these values gradually, and check the results to find the optimum settings for your environment.

Parsers

Are message parsers using more memory than expected?

A message flow parses input messages and can create many output messages. These messages may have large bit streams or large message trees. The parsers created to perform this message processing might consume a large amount of memory. Use the Parsers statistics to determine if message flow parsers are using more memory than expected. If so, consider deploying such flows into separate execution groups or improving ESQL or Java plugin API processing to efficiently handle large messages or transformations.

Is message parsing or writing failing frequently for a particular message flow?

If a message flow receives or attempts to write an invalid message, it is likely that this will be rejected by a parser. Use the message parsers statistics to see if a message flow is rejecting a large amount of input or output messages compared with successful processing.

Outbound sockets

Are the nodes reusing outbound sockets?

Creating outbound sockets can be an expensive operation, and the number of sockets available on a computer is a finite resource. Therefore, increasing socket reuse can enhance performance. If the workload is continuous and consistent, the `TotalSockets` value indicates an initial period of activity, which then reduces when the execution group starts to reuse sockets.

A steady increase in the `TotalSockets` value over time is expected because sockets are closed after a period of inactivity, or when they have been used many times.

If the `TotalSockets` value increases significantly over time, this trend might indicate that outbound sockets are not being reused.

If your message flows include HTTPRequest nodes, check that you have set the keepalive property Enable HTTP/1.1 keepalive.

Check also whether the endpoint that is called uses keepalive sockets.

Which endpoints are most used?

The values `TotalMessages` indicates how busy each endpoint is. The value in the summary record tells you how much activity occurred across the whole execution group.

How large are sent and received messages?

The values of the `SentMessageSize_*` and `ReceivedMessageSize_*` fields give a profile of the message sizes flowing to and from each endpoint.

JDBC connection pools

Do I need to change the size of the connection pool?

If the statistics show that the count of callers waiting for connections is high, and the wait time is increasing, consider increasing the size of the pool using the `MaxConnectionPoolSize` property for the JDBCProvider configurable service.

Alternatively, try reducing the number of additional instances configured for the message flow.

TCPIPClientNodes

Are the nodes reusing outbound sockets?

Creating outbound sockets can be an expensive operation, and the number of sockets available on a computer is a finite resource. Therefore, increasing socket reuse can enhance performance. If the workload is continuous and consistent, the

`TotalSockets` value indicates an initial period of activity, which then reduces when the execution group starts to reuse sockets.

A steady increase in the `TotalSockets` value over time is expected because sockets are closed after a period of inactivity, or when they have been used many times.

If the `TotalSockets` value increases significantly over time, this trend might indicate that outbound sockets are not being reused.

If your message flows include HTTPRequest nodes, check that you have set the keepalive property Enable HTTP/1.1 keepalive.

Check also whether the endpoint that is called uses keepalive sockets.

How the does the information that I see in WebSphere Message Broker Explorer relate to my TCP/IP flows?

An entry is displayed for each configurable service, not for each flow.

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

"Resource statistics" on page 3306
Resource statistics are collected by a broker to record performance and operating details of resources that are used by execution groups.

Related tasks:

"Starting resource statistics collection in the WebSphere Message Broker Explorer" on page 3310
Use the WebSphere Message Broker Explorer to start collecting resource statistics data for your execution groups. You can then view the data in the Broker Resources and Broker Resources Graph views.

"Stopping resource statistics collection in the WebSphere Message Broker Explorer" on page 3312
Use the WebSphere Message Broker Explorer to stop collecting resource statistics data for your execution groups.

"Setting the JVM heap size" on page 3254
When you start an execution group, it creates a Java virtual machine (JVM) for executing a Java user-defined node.

"Subscribing to statistics reports" on page 3317
You can subscribe to topics that return statistics about the operation of your message flows and resource managers.

Related reference:

"Resource statistics data" on page 6745
Learn about the measurements for which data is returned when you activate resource statistics collection.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

Viewing the status of resource statistics collection in the WebSphere Message Broker Explorer

Use the WebSphere Message Broker Explorer to view the status of resource statistics collection in the Broker Resources view.

Before you begin

Before you start:
- Read the concept topic about resource statistics.

About this task

If you prefer, you can view resource statistics collection status by using the **mqsireportresourcestats** command; see "Viewing status of resource statistics collection" on page 3309.

To view the status of resource statistics collection in the WebSphere Message Broker Explorer:

Procedure

1. In the **WebSphere MQ Explorer - Navigator** view, expand the Brokers folder.
2. Click the execution group for which you want to view statistics collection status. You can view status for only one execution group at a time.

   In the **WebSphere MQ Explorer - Content** view, the **Resource Statistics Active** field indicates whether statistics collection is active. If the field contains the value `None`, statistics collection is inactive. If collection is active, the field contains the value `CICS`, `CORBA`, `FTEAgent`, `JVM`, `Parsers`, `Security`, `Sockets` or `JDBC Connection Pooling`.

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

"Resource statistics" on page 3306
Resource statistics are collected by a broker to record performance and operating details of resources that are used by execution groups.

Related tasks:

"Starting resource statistics collection in the WebSphere Message Broker Explorer" on page 3310
Use the WebSphere Message Broker Explorer to start collecting resource statistics data for your execution groups. You can then view the data in the Broker Resources and Broker Resources Graph views.

"Stopping resource statistics collection in the WebSphere Message Broker Explorer" on page 3312
Use the WebSphere Message Broker Explorer to stop collecting resource statistics data for your execution groups.

Related reference:

"Resource statistics data" on page 6745
Learn about the measurements for which data is returned when you activate resource statistics collection.

Subscribing to statistics reports

You can subscribe to topics that return statistics about the operation of your message flows and resource managers.

About this task

Message flow performance

If you enable message flow accounting and statistics collection for a broker, you can subscribe to the messages that the broker publishes on the following topic:

`$SYS/Broker/`*`brokerName`*`/StatisticsAccounting/`*`recordType`*`/`*`executionGroupLabel`*`/`*`messageFlowLabel`*

where *broker_name* is the name of the broker, *recordType* is the type of record (SnapShot or Archive), *executionGroupLabel* is the name of the execution group that you created on that broker, and *messageFlowLabel* is the name of the message flow that you deployed to the execution group.

These messages contain statistics reports and are published at a regular interval, which you control by setting the statsInterval property of the broker. Each publication is a JMS BytesMessage that contains the statistics report in XML format.

Resource performance

If you enable resource statistics collection for one or more execution groups on a broker, you can subscribe to the messages that the broker publishes at 20-second intervals on the following topic:

`$SYS/Broker/`*`brokerName`*`/ResourceStatistics/`*`executionGroupLabel`*

For more information about how to interpret the resource statistics that are included in the publication, see "Viewing resource statistics data in the WebSphere Message Broker Explorer" on page 3313.

Using wildcards in subscriptions

Procedure

You can use wild cards when you subscribe to statistics reports. For example, to receive message flow statistics reports for all brokers and all execution groups, subscribe to the following topic:

```
$SYS/Broker/+/StatisticsAccounting/#
```

To receive execution group resource statistics reports for all brokers and all execution groups, subscribe to the following topic:

```
$SYS/Broker/+/ResourceStatistics/+
```

For further details about how you can use wildcards, see "Special characters in topics" on page 6395.

Results

Subscribers receive statistics reports only from those brokers that are enabled to produce statistics.

Related tasks:

"Monitoring message flow performance" on page 3279
You can configure your message flow to collect statistics about the operation and behavior of your message flows. Use this information to assess the performance of your message flows.

"Monitoring resource performance" on page 3305
You can collect statistics to assess the performance of certain resources used by execution groups.

"Viewing resource statistics data in the WebSphere Message Broker Explorer" on page 3313
Use the WebSphere Message Broker Explorer to view resource statistics data for your execution groups in the Broker Resources and Broker Resources Graph views.

Related reference:

"Special characters in topics" on page 6395
A topic can contain any character in the Unicode character set, but some characters have a special meaning.

Business-level monitoring

You can configure your message flow to emit event messages that can be used to support transaction monitoring and auditing, and business process monitoring.

Before you begin

Before you start:

Before starting to configure monitoring, read the following information.

1. Understand the basic concepts; see "Monitoring basics" on page 3320.
2. Decide what type of monitoring you intend to do; see "Monitoring scenarios" on page 3323.
3. Decide whether to use monitoring properties, or a monitoring profile; see "Deciding how to configure monitoring events for message flows" on page 3325.

About this task

An event is a message that a message flow publishes when something interesting happens. The message contains information about the source of the event, the time of the event, and the reason for the event. The event can include the message bit stream, and can also include selected elements from the message body. These fields can be used to correlate messages that belong to the same transaction, or to convey business data to a monitoring application.

To receive monitoring events, you must take the following steps:

1. Configure event sources on the flow, either by monitoring properties or a monitoring profile.
2. Ensure that the event sources are enabled; use the **mqsireportflowmonitoring** command to report settings. Use the monitoring properties for the node, or the **mqsichangeflowmonitoring** `-i` command to enable event sources, if appropriate.
3. Activate monitoring for the flow; use the **mqsichangeflowmonitoring** `-c` command.
4. Subscribe to the topic for the flow:

   `$SYS/Broker/`*`brokerName`*`/Monitoring/`*`executionGroupName`*`/`*`flowName`*

Related concepts:

"Monitoring basics" on page 3320
Message flows can be configured to emit events. The events can be read and used by other applications for transaction monitoring, transaction auditing, and business process monitoring.

"Monitoring scenarios" on page 3323
Events can be used to support transaction monitoring, transaction auditing and business process monitoring.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Configuring monitoring event sources using monitoring properties" on page 3327
In the Message flow Editor, use the Monitoring tab on the properties of a node to add one or more monitoring events.

"Configuring monitoring event sources using a monitoring profile" on page 762
You can create a monitoring profile and use the **mqsichangeflowmonitoring** command to configure your message flows to emit monitoring events.

Related reference:

"**mqsireportflowmonitoring** command" on page 3924
Use the **mqsireportflowmonitoring** command to display the current options for monitoring that have been set using the **mqsichangeflowmonitoring** command.

"**mqsichangeflowmonitoring** command" on page 3738
Use the **mqsichangeflowmonitoring** command to enable monitoring of message flows.

Monitoring basics

Message flows can be configured to emit events. The events can be read and used by other applications for transaction monitoring, transaction auditing, and business process monitoring.

Monitoring Events

A monitoring event is an XML document that conforms to the monitoring event schema. Each event contains the following information:

- Source of the event
- Name of the event
- Sequence number and creation time
- Correlation ID for events emitted by the same transaction or unit of work

Additionally, a monitoring event can contain the following items:

- Application data extracted from the message
- Part or all of the message bit stream

See "The monitoring event" on page 6774 for more details

Event Sources

A message flow can emit two kinds of events:

Transaction events
Transaction events are emitted only from input nodes.

Terminal events
Terminal events are emitted from any terminal of any node, including input nodes.

An individual message flow can choose to emit transaction events, terminal events, or both kinds of event. You can configure, enable, and disable, both types of events in either of the following ways:

- Using the monitoring properties of the message flow.
- Using a monitoring profile configurable service.

The use of a monitoring profile configurable service overrides the monitoring properties of a message flow.

An event source address identifies an event source in a message flow.

Because terminal events can be emitted from any node in a message flow, they can be used as an alternative to dedicated event-emitting nodes or subflows such as that supplied in SupportPac IA9V.

Event sources emit events only if monitoring is activated for the message flow.

Terminal events

Any terminal in a message flow can be an event source. If the event source is active, it emits an event each time a message passes through the terminal, subject to the evaluation of the eventFilter expression; see "Event output options."

Transaction events

Each input node in a message flow contains three events sources, in addition to terminal events.

Event source	Event source address	Description
Transaction start	*Nodename*.transaction.Start	The event is emitted when the message is read from the transport.
Transaction end	*Nodename*.transaction.End	The event is emitted when WebSphere Message Broker has completed all processing of the message.
Transaction rollback	*Nodename*.transaction.Rollback	The event is emitted instead of transaction end if the message flow throws an exception which is not caught and processed within the message flow.

Events are emitted subject to the evaluation of the eventFilter expression; see "Event output options."

If a message flow handles its own exceptions, a transaction end event, rather than a transaction rollback event, is issued, because the flow has taken control of the error and terminated normally. In this case, if you need to distinguish errors, you can configure terminal events at appropriate nodes in the flow.

Event output options

When you configure an event source, you can define a filter to control whether the event is emitted. You can tailor event emission to your business requirements, by

filtering out events that do not match a set of rules. For example, you might decide to emit events only for transactions above a minimum amount.

```
$Body/StockTrade/Details/Value > 10000
```

This can reduce the number of events emitted, and reduce the workload on your monitoring application.

Events are published to a topic, where they can be read by multiple subscribers. The topic name is of the form:

```
$SYS/Broker/brokerName/Monitoring/executionGroupName/flowName
```

The hierarchical structure allows subscribers to filter the events which they receive. One subscriber can receive events from all message flows in the broker, while another receives only the events from a single execution group.

You decide whether events participate in transactions when you configure a monitoring event source. In general:

- If you want an event to be emitted only if the message flow transaction commits, configure the event source to coordinate the events with the message flow transaction.
- If you want an event to be emitted regardless of whether the message flow transaction commits or rolls back, configure the event source to emit events out of sync point. Such events are available immediately.
- If you want a group of events to be emitted together regardless of whether the message flow transaction commits or rolls back, configure the event source to emit events in a second, independent, unit of work.

Default monitoring configuration

If monitoring is activated for a message flow, and neither monitoring properties nor a monitoring profile configurable service have been configured for the flow, the default behavior is for transaction events to be emitted from each input node of the message flow. The events contain the bit stream of the input message.

Related tasks:

"Business-level monitoring" on page 3319
You can configure your message flow to emit event messages that can be used to support transaction monitoring and auditing, and business process monitoring.

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Configuring monitoring event sources using monitoring properties" on page 3327
In the Message flow Editor, use the Monitoring tab on the properties of a node to add one or more monitoring events.

"Configuring monitoring event sources using a monitoring profile" on page 762
You can create a monitoring profile and use the **mqsichangeflowmonitoring** command to configure your message flows to emit monitoring events.

Related reference:

"Monitoring profile" on page 6768
To customize events after a message flow has been deployed, but without redeploying the flow, use a monitoring profile configurable service. By using this service, you can apply a monitoring profile to one or more message flows.

"**mqsichangeflowmonitoring** command" on page 3738
Use the **mqsichangeflowmonitoring** command to enable monitoring of message flows.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"The monitoring event" on page 6774
You can configure WebSphere Message Broker to emit a monitoring event (an XML document) when something interesting happens. Events are typically emitted to support transaction monitoring, transaction auditing, and business process monitoring. The event XML conforms to the monitoring event schema `WMBEvent.xsd`.

Monitoring scenarios

Events can be used to support transaction monitoring, transaction auditing and business process monitoring.

Transaction monitoring and auditing

The events published by WebSphere Message Broker can be written to a transaction repository, creating an audit trail of the transactions that are processed by a broker. A transaction repository can be used for monitoring, auditing and replay of transactions. Bitstream data can be included so that failed transactions can be resubmitted. You can perform the following tasks to set up transaction monitoring and auditing.

Configure events for your transactions
In most cases bitstream information is not sufficient to allow querying of the logged transactions. Key fields and other correlation data can be extracted from the message payload and placed into the `wmb:applicationData/wmb:simpleContent` or `wmb:applicationData/wmb:complexContent` element of the event. The logging application or message flow can extract these fields and log them with the message bit stream.

Subscribe to the event topic and write events to a repository
You can create a message flow, or any WebSphere MQ application, that subscribes to the event topic and writes events to a relational database. The details of the database schema depend on the requirements of your organization, for example the number of key fields and transaction IDs.

Business process monitoring

The events published by a broker can be monitored by WebSphere Business Monitor. Important fields in the message payload can be added to the events emitted by your message flows, allowing them to be monitored. You can use the following items to help you use WebSphere Business Monitor to monitor your message flows:

Message Driven bean
The events must be submitted to the CEI repository in order for WebSphere Business Monitor to monitor them. A message driven bean is supplied for this purpose. The message driven bean, which runs in

WebSphere Application Server, subscribes to the event topic and writes events that match its subscription to the CEI repository as Common Base Event events.

WebSphere Business Monitor Model
WebSphere Message Broker includes an example monitor model for use with WebSphere Business Monitor. This model demonstrates how to monitor transaction events and terminal events, including events that capture data from the input message and output message. Modify the model to match your actual events and message formats.

The following sample provides a Message Driven Bean and a WebSphere Business Monitor Model to help you use WebSphere Business Monitor to monitor events in your message flows:

- WebSphere Business Monitor

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Related concepts:

"Monitoring basics" on page 3320
Message flows can be configured to emit events. The events can be read and used by other applications for transaction monitoring, transaction auditing, and business process monitoring.

Related tasks:

"Business-level monitoring" on page 3319
You can configure your message flow to emit event messages that can be used to support transaction monitoring and auditing, and business process monitoring.

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Configuring monitoring event sources using a monitoring profile" on page 762
You can create a monitoring profile and use the **mqsichangeflowmonitoring** command to configure your message flows to emit monitoring events.

"Configuring monitoring event sources using monitoring properties" on page 3327
In the Message flow Editor, use the Monitoring tab on the properties of a node to add one or more monitoring events.

Related reference:

"Monitoring profile" on page 6768
To customize events after a message flow has been deployed, but without redeploying the flow, use a monitoring profile configurable service. By using this service, you can apply a monitoring profile to one or more message flows.

"**mqsichangeflowmonitoring** command" on page 3738
Use the **mqsichangeflowmonitoring** command to enable monitoring of message flows.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your

message flows.

Deciding how to configure monitoring events for message flows

Decide whether to use monitoring properties, or a monitoring profile configurable service, to customize the events produced by a message flow.

If you want to customize events when you are designing the message flow, use monitoring properties. If you want to customize events after a message flow has been deployed, but without redeploying, use a monitoring profile. This topic gives information about both methods.

Using monitoring properties to configure events

Using the WebSphere Message Broker Toolkit Message Flow Editor, you can configure event sources on most nodes in a message flow. A node that supports the configuration of event sources has a Monitoring tab on its Properties view; use this tab to add events and to set their properties. When you deploy the message flow in a broker archive file, the monitoring properties are included as part of the message flow.

Key facts about monitoring properties:

- Use monitoring properties when you want to configure events at message flow design time.
- Monitoring properties apply only to the message flow in question. You can share monitoring properties between message flows only by creating reusable subflows.
- Monitoring properties are deployed in a BAR file as part of the message flow.
- You use the **mqsichangeflowmonitoring** command to activate the message flow to emit monitoring events.
- You can use the **mqsichangeflowmonitoring** command to enable or disable individual event sources in a message flow.
- To change any other properties of an event, you alter the monitoring properties in the flow, then redeploy the BAR file.

To configure monitoring events using monitoring properties, see this information: "Configuring monitoring event sources using monitoring properties" on page 3327.

Tip: Use the **mqsireportflowmonitoring** command to report a list of the names of the event sources for a message flow. You can then use these event source names in a **mqsichangeflowmonitoring** command to enable or disable individual event sources from the command line.

Using a monitoring profile to configure events

Using operational commands, you can create a monitoring profile configurable service directly on the broker, and associate it with one or more message flows.

Key facts about monitoring profiles:

- Use a monitoring profile when you want to configure events for a message flow that has already been deployed and for which no events have been configured.

- Use a monitoring profile to override the monitoring properties of a message flow that has already been deployed, as an alternative to redeploying the BAR file. The monitoring profile completely replaces all monitoring properties.
- A single monitoring profile can be applied to many message flows, provided that the message flows contain the same event sources.
- Monitoring profiles are created directly on the broker using the **mqsicreateconfigurableservice** command and the **mqsichangeproperties** command. They are not deployed in a BAR file.
- You use the **mqsichangeflowmonitoring** command to associate the monitoring profile with a message flow.
- You use the **mqsichangeflowmonitoring** command to activate the message flow to emit monitoring events.
- You can use the **mqsichangeflowmonitoring** command to enable or disable individual event sources in a message flow.

To configure monitoring events using monitoring properties, see this information:

"Configuring monitoring event sources using a monitoring profile" on page 762

Tip: Use the **mqsireportflowmonitoring** command to report a list of event sources for a message flow. You can then use the names of these event source in a subsequent **mqsichangeflowmonitoring** command to enable and disable individual event sources from the command line.

Tip: Use the **mqsireportflowmonitoring** command to report the monitoring profile for a message flow. You can edit the profile, to add or change event sources, then update it using the **mqsichangeproperties** command.

Tip: Use the **mqsireportflowmonitoring** command to create the equivalent monitoring profile for a message flow for which monitoring properties are configured. You can edit the profile, then use it to create a new monitoring profile configurable service using the **mqsicreateconfigurableservice** and **mqsichangeproperties** commands. You can then associate the new profile with the original flow using the **mqsichangeflowmonitoring** command. You can use this technique to override the monitoring properties for a message flow, without having to edit the flow and redeploy the BAR file.

Related concepts:

"Monitoring basics" on page 3320
Message flows can be configured to emit events. The events can be read and used by other applications for transaction monitoring, transaction auditing, and business process monitoring.

Related tasks:

"Configuring monitoring event sources using monitoring properties" on page 3327
In the Message flow Editor, use the Monitoring tab on the properties of a node to add one or more monitoring events.

"Configuring monitoring event sources using a monitoring profile" on page 762
You can create a monitoring profile and use the **mqsichangeflowmonitoring** command to configure your message flows to emit monitoring events.

Related reference:

"Monitoring profile" on page 6768
To customize events after a message flow has been deployed, but without redeploying the flow, use a monitoring profile configurable service. By using this service, you can apply a monitoring profile to one or more message flows.

"**mqsichangeflowmonitoring** command" on page 3738
Use the **mqsichangeflowmonitoring** command to enable monitoring of message flows.

"**mqsireportflowmonitoring** command" on page 3924
Use the **mqsireportflowmonitoring** command to display the current options for monitoring that have been set using the **mqsichangeflowmonitoring** command.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

Configuring monitoring event sources using monitoring properties

In the Message flow Editor, use the Monitoring tab on the properties of a node to add one or more monitoring events.

Before you begin

Before you start:

Read the following topics:

- "Business-level monitoring" on page 3319
- "Monitoring basics" on page 3320

You must have a message flow that contains a node to which you want to add a monitoring event.

You can use XPath 1.0 expressions to configure a monitoring event. Some XPath expressions, listed in "XPath expressions that are not suitable for the export monitoring information option" on page 6782, are not suitable for use with the export monitoring information option ("Creating a monitor model for WebSphere Business Monitor V7" on page 3341).

Creating events

About this task

An event source is a point in a message flow from which a monitoring event can be emitted. Each event source has a set of properties that control the contents of the monitoring events that it emits.

Procedure

1. Display the properties for the node.
2. Select the Monitoring tab.
3. Click **Add**.

   The Add entry window is displayed.
4. Complete the **Event Source** field.

   The field has a drop-down list of all events that can be defined for this node. The event source information is used to populate the attributes of the `wmb:eventPointData/wmb:messageFlowData/wmb:node` element of the event. When you have selected the event source, the corresponding value for `Event Source Address` is displayed as a read-only property.

Tip: If you later decide to enable or disable events using the **mqsichangeflowmonitoring** command, you must specify a value for `Event Source Address`, not `Event Name`.

5. Complete the **Event Name** details; select either **Literal** or **Data location**.

   Every monitoring event has a name that is placed in the `wmb:eventPointData/wmb:eventIdentity/@wmb:eventName` attribute of the event. The default names are shown in the following table:

Event source	Default event name	Example
Transaction start	*nodeLabel*.TransactionStart	MQInput.TransactionStart
Transaction end	*nodeLabel*.TransactionEnd	MQInput.TransactionEnd
Transaction rollback	*nodeLabel*.TransactionRollback	MQInput.TransactionRollback
Terminal	*nodeLabel.terminal_label*.Terminal	MQInput.OutTerminal

   You can override the default in these ways:

   - By specifying an alternative literal string.
   - By specifying an XPath query; the query extracts the event name from a field in the input message. Click **Edit** to use the XPath Expression Builder.

   You cannot use monitoring properties to configure transaction events on the following nodes:

   "Collector node" on page 4333

   "Resequence node" on page 4651

   Use a monitoring profile instead; see "Configuring monitoring event sources using a monitoring profile" on page 762.

6. Optional: Complete the Event Filter section by providing an XPath expression to control whether the event is emitted. Take one of the following steps:

   - Type in the expression (for example, `$Body/StockTrade/Details/Value > 10000`); or
   - Click **Edit** to launch XPath Expression Builder.

   The expression must evaluate to true or false, and can reference fields in the message tree, or elsewhere in the message assembly. The default value is `true()`, which means that the event is always produced.

   Using this facility, you can tailor event emissions to your business requirements, by filtering out events that do not match a set of rules. This can reduce the number of events emitted, and reduce the workload on your monitoring application.

7. Optional: Complete the **Event Payload** section if the event is to contain selected data fields extracted from the message. Click **Add** to launch the Add Data Location dialog box. Take one of the following steps:

   - Type in the location (for example, `$LocalEnvironment/File/Name`); or
   - Click **Edit** to launch XPath Expression Builder.

   You can extract one or more fields from the message data and include it with the event. The fields can be simple or complex. Simple content is contained in the `wmb:applicationData/wmb:simpleContent` field of the event; complex data is contained in the `wmb:applicationData/wmb:complexContent` field.

   This facility is commonly used for communicating significant business data in a business event. If the event contains the input bit stream, this facility can also be used to extract key fields, allowing another application to provide an audit trail or to resubmit failed messages.

8. Optional: Select the **Include bitstream data in payload** field if the event is to capture message bitstream data.

   Content
   Select from `Headers`, `Body`, `All`.

   Encoding
   Select from `base64`, `HexBinary` and `CData` (the original text, without encoding).

9. Optional: Select the Correlation tab, to complete details for event correlation.
10. Complete the **Event Correlation** details; for information about correlation, see "Correlation and monitoring events" on page 6778.

Every monitoring event must contain at least one correlation attribute, and can contain up to three. If you do not specify any correlation information, the first event source in the message flow allocates a unique identifier that all later event sources in the same transaction will use.

a. Optional: Complete the **Local transaction correlator** details.

   Automatic
   The local correlator used by the most recent event for this invocation of the message flow will be used. If no local correlator exists yet, a new unique value will be generated.

   Specify location of correlator
   Enter a value, or click Edit to launch the XPath Expression Builder. The local correlator will be read from the specified location in the message tree. Ensure that the specified location contains a correlator value unique to this invocation of the message flow.

b. Optional: Complete the **Parent transaction correlator** details to extract a correlation field from the parent transaction.

   Automatic
   The parent correlator used by the most recent event for this invocation of the message flow will be used. If no parent correlator exists yet, no parent correlator will be used.

   Specify location of correlator
   Enter a value, or click Edit to launch the XPath Expression Builder. The parent correlator will be read from the specified location in the message tree. Ensure that the specified location contains a suitable value for the parent correlator.

c. Optional: Complete the **Global transaction correlator** details to extract a correlation field from a global transaction.

   Automatic
   The global correlator used by the most recent event for this invocation of the message flow will be used. If no global correlator exists yet, no global correlator will be used.

   Specify location of correlator
   Enter a value, or click Edit to launch the XPath Expression Builder. The global correlator will be read from the specified location in the message tree. Ensure that the specified location contains a suitable value for the global correlator.

11. Optional: Choose whether the emission of monitoring events by a message flow is coordinated with the message flow transaction, or is in an independent unit of work, or is not in a unit of work.

Click the Transaction tab and select the appropriate option for **Event Unit of Work**.

Message flow
The event, and all other events with this setting, are emitted only if the message flow commits its unit of work successfully.

If the transaction start event is specified to be included in the message flow unit of work, but the message processing fails and this unit of work is not published, the transaction start event will be included in an independent unit of work. This ensures that your monitoring application receives a pair of events (start and rollback), rather than receiving a rollback event in isolation.

Independent
The event is emitted in a second unit of work, independent of the main unit of work. The event, and all other events with this setting are emitted whether or not the main unit of work commits successfully.

An independent transaction can be started only if the main transaction has been either committed or rolled back. If the **Commit count** property of the flow is greater than one, ("Configurable message flow properties" on page 4020), or the **Commit by message group** property is set ("Receiving messages in a WebSphere MQ message group" on page 1554), the events targeted for the independent transaction are instead emitted out of sync point, and a message is output stating that this has been done.

None The event is emitted out of sync point (not in any unit of work.) The event is emitted when the message passes through the event source, and is available for reading immediately.

Not all these options are available on all event types. The defaults and allowed values are shown in the following table:

Event source	Allowed values	Default
Transaction start	Message flow Independent None	Message flow
Transaction end	Message flow None	Message flow
Transaction rollback	Independent None	Independent
Terminal	Message flow Independent None	Message flow

12. Click **Finish**.

    The **Events** table in the Monitoring tab of the Properties view for the node is updated with details of the event that you just added; the event is enabled.
13. Optional: Disable the event.
14. Save the message flow.

Deploying monitoring properties

Procedure

1. When you have added all events to the flow, add the message flow to the broker archive (BAR) file and deploy the BAR file.

Monitoring is inactive for the flow; deploying the BAR file does not activate it.

2. Activate monitoring for the flow by using the **mqsichangeflowmonitoring -c** command.

Updating monitoring properties

About this task

The monitoring properties for a node show all monitoring events that are defined for that node. Edit the monitoring properties of a node to do these tasks:

- Enable or disable a monitoring event.
- Add, delete, or change the monitoring events for the node.

Procedure

1. Right-click the node and select **Properties**.
2. Select the Monitoring tab.

   The monitoring events that you have previously defined are displayed.
3. Select or clear the **Enabled** check box for each event as appropriate.
   - To disable an event, clear the check box.
   - To enable an event, select the check box.
4. To add an event, click **Add**.
5. To delete an event, select the event, then click **Delete**.
6. To edit an event, select the event, then click **Edit**.

   The Add Event is displayed. For a description of options on this window, see "Creating events" on page 3327.
7. Save the message flow.

Related concepts:

"Deciding how to configure monitoring events for message flows" on page 3325
Decide whether to use monitoring properties, or a monitoring profile configurable service, to customize the events produced by a message flow.

"Monitoring scenarios" on page 3323
Events can be used to support transaction monitoring, transaction auditing and business process monitoring.

Related tasks:

"Adding files to a broker archive" on page 3223
To deploy files to an execution group, include them in a broker archive (BAR) file.

"Activating monitoring" on page 3334
Use the **mqsichangeflowmonitoring** command to activate monitoring after you have configured monitoring event sources.

"Creating a monitor model for WebSphere Business Monitor V6.2" on page 3339
Export monitoring information from WebSphere Message Broker to create a monitoring model for WebSphere Business Monitor V6.2

"Creating a monitor model for WebSphere Business Monitor V7" on page 3341
Export monitoring information from WebSphere Message Broker to create a monitoring model for WebSphere Business Monitor V7

Related reference:

"**mqsichangeflowmonitoring** command" on page 3738
Use the **mqsichangeflowmonitoring** command to enable monitoring of message flows.

"Example XPath expressions for event filtering" on page 6781
Use numeric, string, or Boolean expressions when configuring an event source, to

determine whether the event is emitted.

Related information:

"Correlation and monitoring events" on page 6778
A monitoring application uses correlation attributes to identify events that belong to the same business transaction.

Configuring monitoring event sources using a monitoring profile

You can create a monitoring profile and use the **mqsichangeflowmonitoring** command to configure your message flows to emit monitoring events.

Before you begin

Before you start:

Read the following topics:

- "Business-level monitoring" on page 3319
- "Monitoring basics" on page 3320

You must have a message flow that contains a node to which you want to add a monitoring event.

You can use XPath 1.0 expressions to configure a monitoring event.

Creating a monitoring profile

About this task

First create a monitoring profile XML file. This is a file that lists the event sources in the message flow that will emit events, and defines the properties of each event.

Procedure

Follow the guidance at "Monitoring profile" on page 6768 to create your monitoring profile XML file.

Applying a monitoring profile

About this task

When you have created a monitoring profile XML file, follow these steps to apply it.

Procedure

1. Use the **mqsicreateconfigurableservice** command to create a configurable service for the monitoring profile. In the following command example, replace *myBroker* with the name of your broker, and *myMonitoringProfile* with the name of your monitoring profile.

   ```
   mqsicreateconfigurableservice myBroker -c MonitoringProfiles
        -o myMonitoringProfile
   ```

2. Use the **mqsichangeproperties** command to associate your monitoring profile XML file with the configurable service. In the following command example, replace *myBroker* with the name of your broker, *myMonitoringProfile* with the name of your monitoring profile, and *myMonitoringProfile.xml* with the name of the monitoring profile XML file.

```
mqsichangeproperties myBroker -c MonitoringProfiles -o myMonitoringProfile
     -n profileProperties -p myMonitoringProfile.xml
```

Set the `useParserNameInMonitoringPayload` property to TRUE to force the `wmb:applicationData/wmb:complexContent/wmb:elementName` attribute to hold the name of the input node parser, if present. See "MonitoringProfiles configurable service" on page 3781 for details.

```
mqsichangeproperties myBroker -c MonitoringProfiles -o myMonitoringProfile
     -n useParserNameInMonitoringPayload -v TRUE
```

3. Use the **mqsichangeflowmonitoring** command to apply a monitoring profile configurable service to one or more message flows.
   - Apply a monitoring profile to a single message flow *messageflow1* in execution group *EG1*:

     ```
     mqsichangeflowmonitoring myBroker -e EG1
          -f messageflow1 -m myMonitoringProfile
     ```

   - Apply a monitoring profile to all message flows in all execution groups:

     ```
     mqsichangeflowmonitoring myBroker -g -j -m myMonitoringProfile
     ```

   Monitoring for the flow is inactive; applying the monitoring profile does not activate it.
4. Alternatively, use the broker archive editor to apply a monitoring profile configurable service to one or more message flows, by setting message flow property **Monitoring Profile Name**.
   a. In the WebSphere Message Broker Toolkit, switch to the Broker Application Development perspective.
   b. In the Broker Development view, right-click the BAR file, then click **Open with** > **Broker Archive Editor**.
   c. Click the **Manage and Configure** tab.
   d. Click the message flow on which you want to set the monitoring profile configurable service. The properties that you can configure for the message flow are displayed in the **Properties** view.
   e. In the **Monitoring Profile Name** field, enter the name of a monitoring profile.
   f. Save the BAR file.
   g. Deploy the BAR file.

   Monitoring for the flow is inactive; deploying the BAR file does not activate it.
5. Activate monitoring for the flow using the `mqsichangeflowmonitoring -c` command.
   - Activate monitoring for a single message flow *messageflow1* in execution group *EG1*:

     ```
     mqsichangeflowmonitoring myBroker -e EG1
          -f messageflow1 -c active
     ```

   - Activate monitoring for all message flows in all execution groups:

     ```
     mqsichangeflowmonitoring myBroker -g -j -c active
     ```

Updating a monitoring profile

Procedure

1. Follow the guidance at "Monitoring profile" on page 6768 to update your monitoring profile XML file.
2. Use the **mqsichangeproperties** command to update the configurable service to use the new XML file. For example:

   ```
   mqsichangeproperties myBroker -c MonitoringProfiles -o myMonitoringProfile
        -n profileProperties -p myMonitoringProfile.xml
   ```

Related concepts:

"Monitoring basics" on page 3320
Message flows can be configured to emit events. The events can be read and used by other applications for transaction monitoring, transaction auditing, and business process monitoring.

"Configurable services" on page 1296
Configurable services are typically runtime properties. You can use them to define properties that are related to external services on which the broker relies; for example, an SMTP server or a JMS provider.

Related tasks:

"Business-level monitoring" on page 3319
You can configure your message flow to emit event messages that can be used to support transaction monitoring and auditing, and business process monitoring.

"Activating monitoring"
Use the **mqsichangeflowmonitoring** command to activate monitoring after you have configured monitoring event sources.

Related reference:

"Monitoring profile" on page 6768
To customize events after a message flow has been deployed, but without redeploying the flow, use a monitoring profile configurable service. By using this service, you can apply a monitoring profile to one or more message flows.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsichangeflowmonitoring** command" on page 3738
Use the **mqsichangeflowmonitoring** command to enable monitoring of message flows.

"**mqsireportflowmonitoring** command" on page 3924
Use the **mqsireportflowmonitoring** command to display the current options for monitoring that have been set using the **mqsichangeflowmonitoring** command.

"Example XPath expressions for event filtering" on page 6781
Use numeric, string, or Boolean expressions when configuring an event source, to determine whether the event is emitted.

Activating monitoring

Use the **mqsichangeflowmonitoring** command to activate monitoring after you have configured monitoring event sources.

Before you begin

Before you start: You must have configured monitoring event sources by using either monitoring properties or a monitoring profile; see these topics:

"Configuring monitoring event sources using monitoring properties" on page 3327

"Configuring monitoring event sources using a monitoring profile" on page 762

About this task

The use of a monitoring profile configurable service overrides the monitoring properties of a message flow.

If monitoring is activated for a message flow, and neither monitoring properties nor a monitoring profile configurable service have been configured for the flow, the default behavior is for transaction events to be emitted from each input node of the message flow. The events contain the bit stream of the input message.

Activating monitoring from the command line

About this task

Use the `-c active` parameter. You can activate monitoring for all message flows in all execution groups, or specify the execution groups and message flows for which monitoring is to be activated.

Procedure

- To activate monitoring for all message flows in all execution groups on Linux, UNIX, and Windows: Linux UNIX Windows

  ```
  mqsichangeflowmonitoring myBroker -c active -g -j
  ```

  To activate monitoring for all message flows in all execution groups on z/OS: z/OS

  ```
  F MI10BRK,cm c=active,g=yes,j=yes
  ```

- To activate monitoring for all message flows in the default execution group on Linux, UNIX, and Windows: Linux UNIX Windows

  ```
  mqsichangeflowmonitoring myBroker -c active -e default -j
  ```

  To activate monitoring for all message flows in the default execution group on z/OS: z/OS

  ```
  F MI10BRK,cm c=active,g=default,j=yes
  ```

- To activate monitoring for the myFlow message flow in the default execution group on Linux, UNIX, and Windows: Linux UNIX Windows

  ```
  mqsichangeflowmonitoring myBroker -c active -e default -f myFlow
  ```

  To activate monitoring for the myFlow message flow in the default execution group on z/OS: z/OS

  ```
  F MI10BRK,cm c=active,g=default,f=myFlow
  ```

Deactivating monitoring from the command line

About this task

Use the `-c inactive` parameter. You can deactivate monitoring for all message flows in all execution groups, or specify the execution groups and message flows for which monitoring is to be activated.

Procedure

- To deactivate monitoring for all message flows in all execution groups on Linux, UNIX, and Windows: Linux UNIX Windows

  ```
  mqsichangeflowmonitoring myBroker -c inactive -g -j
  ```

To deactivate monitoring for all message flows in all execution groups on z/OS: z/OS

```
F MI10BRK,cm c=inactive,g=yes,j=yes
```

- To deactivate monitoring for all message flows in the default execution group on Linux, UNIX, and Windows: Linux UNIX Windows

  ```
  mqsichangeflowmonitoring myBroker -c inactive -e default -j
  ```

  To deactivate monitoring for all message flows in the default execution group on z/OS: z/OS

  ```
  F MI10BRK,cm c=inactive,g=default,j=yes
  ```

- To deactivate monitoring for the myFlow message flow in the default execution group on Linux, UNIX, and Windows: Linux UNIX Windows

  ```
  mqsichangeflowmonitoring myBroker -c inactive -e default -f myFlow
  ```

  To deactivate monitoring for the myFlow message flow in the default execution group on z/OS: z/OS

  ```
  F MI10BRK,cm c=inactive,g=default,f=myFlow
  ```

Related tasks:

"Configuring monitoring event sources using monitoring properties" on page 3327
In the Message flow Editor, use the Monitoring tab on the properties of a node to add one or more monitoring events.

"Configuring monitoring event sources using a monitoring profile" on page 762
You can create a monitoring profile and use the **mqsichangeflowmonitoring** command to configure your message flows to emit monitoring events.

Related reference:

"**mqsichangeflowmonitoring** command" on page 3738
Use the **mqsichangeflowmonitoring** command to enable monitoring of message flows.

Enabling and disabling event sources

When events have been configured for a message flow and deployed to the broker, you can enable and disable individual events. You can do this from the command line, without having to redeploy the message flow, or you can do this from the Message Flow Editor, in which case you must redeploy.

Enabling and disabling events from the command line

Before you begin

Before you start: You must have previously configured events using either Message Flow Editor monitoring properties, or a monitoring profile configurable service.

Procedure

Use the **mqsichangeflowmonitoring** to enable and disable event sources. For example:

```
mqsichangeflowmonitoring WBRK_BROKER
     -e default
     -f myMessageFlow
     -s "SOAP Input1.terminal.out,MQOutput1.terminal.in"
     -i enable
```

```
mqsichangeflowmonitoring WBRK_BROKER
     -e default
     -f myMessageFlow
     -s "SOAP Input1.terminal.catch"
     -i disabled
```

You can enable or disable multiple events at once; the change of state takes place immediately.
If you configured events using monitoring properties, the change persists if the message flow is restarted, but is lost if the message flow is redeployed. To make the change permanent, you must also update the monitoring properties.

Tip: When specifying values for the -s parameter, use the **Event source address** property of the event, not the **Event name** property.

Tip: To find the list of configured event sources for a message flow, you can use the mqsireportflowmonitoring command. For example:

```
mqsireportflowmonitoring WBRK_BROKER
      –e default
      –f myMessageFlow
      –n
```

Tip: If you configured events using monitoring properties, you can see a list of the configured event sources in the Message Flow Editor:

1. Open the message flow using the Message Flow Editor.
2. Click the canvas.
3. Select the Monitoring tab in the Properties view.

The monitoring events that you have previously defined are displayed.

Enabling and disabling events from the Message Flow Editor

Before you begin

Before you start: You must have previously configured events using the Message Flow Editor monitoring properties.

About this task

Use the Message Flow Editor to enable and disable event sources.

Procedure

1. Open the message flow using the editor.
2. Click the canvas.
3. Select the Monitoring tab in the Properties view. The monitoring events that you have previously defined are displayed.
4. Select or clear the **Enabled** check box for each event as appropriate.
   - To disable an event, clear the check box.
   - To enable an event, select the check box.
   - To disable all events, click **Uncheck All**.
5. Save the message flow.
6. Rebuild and redeploy the BAR file.

Related concepts:

"Monitoring basics" on page 3320
Message flows can be configured to emit events. The events can be read and used by other applications for transaction monitoring, transaction auditing, and business process monitoring.

Related tasks:

"Business-level monitoring" on page 3319
You can configure your message flow to emit event messages that can be used to support transaction monitoring and auditing, and business process monitoring.

Related reference:

"**mqsichangeflowmonitoring** command" on page 3738
Use the **mqsichangeflowmonitoring** command to enable monitoring of message flows.

"**mqsireportflowmonitoring** command" on page 3924
Use the **mqsireportflowmonitoring** command to display the current options for monitoring that have been set using the **mqsichangeflowmonitoring** command.

Creating a monitoring model for use by WebSphere Business Monitor

Enable WebSphere Business Monitor to monitor WebSphere Message Broker events.

Before you begin

Before you start: You must have configured monitoring for a message flow; see "Deciding how to configure monitoring events for message flows" on page 3325.

About this task

IBM WebSphere Business Monitor is business activity monitoring software that provides a real-time view of your business processes and operations. It contains personalized business dashboards that calculate and display key performance indicators (KPIs) and metrics derived from business processes, business activity data, and business events from a wide range of information sources, including WebSphere Message Broker message flows. This gives you immediate insight into your business operations so that you can mitigate problems or take immediate advantage of opportunities, resulting in cost savings and increased revenues.

The following WebSphere Business Monitor sample shows you how to generate events from a message flow, create a corresponding monitor model, and use that model in WebSphere Business Monitor to view KPIs from the events:

- WebSphere Business Monitor

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

The steps that you take depend on the version of WebSphere Business Monitor that you are using:

"Creating a monitor model for WebSphere Business Monitor V6.2" on page 3339

"Creating a monitor model for WebSphere Business Monitor V7" on page 3341

Related tasks:

"Generating XML Schemas" on page 2963
You can generate either a single XML Schema from a message definition file, or multiple XML Schemas from a message set.

"Business-level monitoring" on page 3319
You can configure your message flow to emit event messages that can be used to support transaction monitoring and auditing, and business process monitoring.

Creating a monitor model for WebSphere Business Monitor V6.2

Export monitoring information from WebSphere Message Broker to create a monitoring model for WebSphere Business Monitor V6.2

About this task

In WebSphere Message Broker

Procedure

1. If an event contains complex data extracted from a message, supply details of the structure of the data to WebSphere Business Monitor:
   a. Ensure that the complex data is modeled as a complex type in a message definition file within a message set in the WebSphere Message Broker Toolkit.
   b. In the Application Development perspective, right-click the message set folder and click **Generate XML Schemas** to create a `.zip` file of XML schemas from the definitions in the message set.
   c. Extract the file containing the XML schemas on the computer where you will be using the Monitoring Model Editor.
2. Export the `WMBEvent.xsd` file from an existing message set (or create a message set, then export it from there):
   a. In the Application Development perspective, right-click the message set and click **New** > **Message Definition File From** > **IBM Supplied Message**.
   b. In the window that is displayed, scroll down and click **Message Broker Monitoring Event**.
   c. Click **Finish**.
   d. Right-click the message set again, and click **Generate** > **XML Schemas**.
   e. In the window that is displayed, click **Export to an external directory**, then enter or browse to a directory.
   f. Click **Finish**. A compressed file containing the `WMBEvent.xsd` file is created in the directory that you specified.

   The `WMBEvent.xsd` file gives WebSphere Business Monitor details of the structure of the WebSphere Message Broker event.

What to do next

In WebSphere Business Monitor V6.2

This section outlines the steps you need to take in WebSphere Business Monitor. See the documentation for WebSphere Business Monitor for full and up-to-date details.

1. In the WebSphere Business Monitor development toolkit, create a Monitor Model.

   Import the WMBEvent.xsd schema and any schemas describing complex data that were exported from WebSphere Message Broker, then create a monitor

model. You see errors, because the model is currently empty. You also see a key, which you can rename, for example to LocalTransactionID.

2. Create WebSphere Business Monitor inbound events.

   A *monitoring context definition* is the term used by WebSphere Business Monitor to describe all the data that should be collected about an entity (such as a transaction or business process). Each runtime instance (referred to as a monitoring context instance) collects information from inbound events and stores this information in fields that represent the business measures that a monitoring context collects: metrics, counters, and stopwatches.

   You need to define an inbound event to describe each event source defined in your message flow that contains information that you want to monitor. For example, if your message flow has Transaction start, Transaction end and Transaction rollback event sources, define an inbound event for each of these event sources.

   You typically define inbound events for these three event sources because they contain information that tells WebSphere Business Monitor when the start and end of the monitoring context instance occurs. You also define inbound events to describe any event sources downstream in the flow that contain data that you want to monitor, for example in the In terminal of the MQOutput node.Creating inbound events typically involves the following actions:

   a. Define event parts within the inbound event.

      Event parts are XML Schema definition (XSD) types that provide information about the structure of part of an event. For a WebSphere Message Broker event, define event parts to describe the different parts of the event that you want to monitor data from. For a description of the event structure, see "The monitoring event" on page 6774.

      As a minimum, define an event part for the event described by type wmb:event. To monitor data about the source of the event (information about the message flow name, broker name), also define an event part for the eventPointData section described by type wmb:eventPointData. You might also want to define event parts to describe message payload data from the applicationData section of the event.

   b. Define a correlation expression.

      You typically correlate events on fields from the eventCorrelation section in the WebSphere Message Broker event. For example, you could correlate events using the localTransactionId field from the WebSphere Message Broker event.

      You must also define whether the event should create a monitoring context; create this for a Transaction start event.

   c. Optional: Define a filter condition.

      Set a filter condition. For example you might want to filter events for a specific broker, execution group, or message flow.

   d. Optional: Define the event sequence path.

      Select a field in the inbound event that can be used to set the order in which the inbound events are processed. For example, you could use creationTime from the WebSphere Message Broker event.

   e. Complete the key.

      The key uniquely identifies a monitoring context instance. You can select any value for the key; for a WebSphere Message Broker event a typical value is the localTransactionId field from the WebSphere Message Broker event.

3. Define the metrics.

Having defined inbound events you can now define your metrics. Metrics hold data from the event in a monitoring context.

You might want to define metrics that hold event source data from the eventPointData section of the WebSphere Message Broker event, for example broker name or message flow name. You can also define metrics that hold event sequencing information, for example TimeStarted and TimeEnded metrics, which hold the creationTime for Transaction start and Transaction end or Transaction rollback events.

In addition, metrics can be defined to hold application data from the WebSphere Message Broker event.

Creating a monitor model for WebSphere Business Monitor V7

Export monitoring information from WebSphere Message Broker to create a monitoring model for WebSphere Business Monitor V7

Before you begin

Before you start:

For earlier versions, see "Creating a monitor model for WebSphere Business Monitor V6.2" on page 3339.

The monitoring information needed to create a monitor model can only be exported from WebSphere Message Broker when the following conditions are met:

- The flow is not empty.
- The flow has no error markers against it in the Problems view.

Consider these points before exporting a message flow:

- WebSphere Business Monitor must be able to identify the start and end of a monitoring context. Always define transaction events (`transaction.Start`, `transaction.End`, and `transaction.Rollback`) on message flows that are monitored by WebSphere Business Monitor.
- The WebSphere Business Monitor toolkit does not support local elements with anonymous types. The export monitoring information option therefore does not generate an event part for event payload XPath queries which resolve to an element of this type. You see a warning message in the report log *flowProjectName*`_batchgen_report.txt`.
- The WebSphere Business Monitor toolkit does not support creating metrics of type `xs:anyType`. If an XPath expression in your event payload resolves to an element of type `xs:anyType`, the export monitoring information option creates an event part of this type, but you cannot create a metric of this type in the WebSphere Business Monitor toolkit. Create an event part with a supported type; see *Defining Event Parts* in the WebSphere Business Monitor information center.
- If you include unmodeled data, the export monitoring information option cannot type the data. It assigns a type of `string` to the data.
- The option to export monitoring information does not support XPath queries that contain wildcards.

Tip: If you want to separately identify `transaction.Start` and `transaction.End` events that are issued following message flow error handling, create an event source on the Failure terminal of the Input node.

About this task

Use the subsequent WebSphere Message Broker procedure to create a .zip file that contains the following monitoring information about the message flow:

All monitoring information that is defined in the message flow files.

The WMBEvent.xsd file, which is the schema for the emitted event.

All `.xsd` files in the message sets that are referenced by the selected flows.

In WebSphere Message Broker

Procedure

1. In the WebSphere Message Broker Toolkit, right-click the message flow or message flow project, and click **Export**. The Export wizard starts.
2. Click **Business Monitoring** > **Application monitoring information**, then click **Next**.
3. Select the flows from which you want to export monitoring information, and specify a file name.
4. Optional: Specify that an existing file should be overwritten without warning.
5. Optional: If working set filtering is enabled, select the **Apply working set filtering to artifact selection(s) on this page** check box.
6. Click **Finish**. A .zip file that contains the application monitoring information is saved.

   A report of the export is written to file *flowProjectName*`_batchgen.report.txt` in the log directory of the message flow project.

What to do next

In WebSphere Business Monitor V7

In the WebSphere Business Monitor development toolkit, import the `.zip` file that you exported from WebSphere Message Broker. The Generate Monitor Model wizard starts. As you work through the wizard, you can select some or all of the following templates, depending on the event sources in your message flow:

- **Average Transaction Duration** This template is available if you have created `transaction.Start`, `transaction.End`, and `transaction.Rollback` event sources. If you select this template, a metric and stopwatch are created for you in the monitor model showing the Average Transaction Duration, and a key performance indicator (KPI) is created using the data from this stopwatch.
- **Number of Failed Transactions** This template is available if you have created `transaction.Rollback` event sources. If you select this template, metrics are created for you in the monitor model showing the number of failed transactions and the failed transaction time (this has a default value of 1 January 9999 01:00:00.) A measure and dimension are also created for use in creating multidimensional reports. See *Defining Dimensions* in the WebSphere Business Monitor information center.
- **Message Flow Correlation** This template provides information about the broker (such as broker name, execution group name, parentTransactionID, globalTransactionID). You can display this information in a Business Space Dashboard with other metrics and key performance indicators. Because these metrics are used in the correlation expression for the inbound events defined, select this template only if the events for a specified monitoring context are from the same execution group.

See the documentation for WebSphere Business Monitor for full and up-to-date details.

Related reference:

"XPath expressions that are not suitable for the export monitoring information option" on page 6782
Some XPath expressions produce a warning on the Monitoring tab.

Reporting monitoring settings

Use the **mqsireportflowmonitoring** command to report monitoring settings for a flow.

About this task

The following examples show how to report monitoring options for a broker called BROKER1, execution group default, and message flow PurchaseOrder.

Report configured events for a message flow

Procedure

Issue the following command on Linux, Unix, or Windows:

Linux UNIX Windows

```
mqsireportflowmonitoring BROKER1 -e default -f PurchaseOrder -n
```

Issue the following command on z/OS: z/OS

```
F MI10BRK,rm BROKER1,e='default',f='PurchaseOrder,n='yes''
```

Report all possible events for a message flow

Procedure

Issue the following command on Linux, Unix, or Windows:

Linux UNIX Windows

```
mqsireportflowmonitoring BROKER1 -e default -f PurchaseOrder -a
```

Issue the following command on z/OS: z/OS

```
F MI10BRK,rm e='default',f='PurchaseOrder,a='yes'
```

Report specified events for a message flow

Procedure

Issue the following command on Linux, Unix, or Windows:

Linux UNIX Windows

```
mqsireportflowmonitoring BROKER1 -e default -f PurchaseOrder
     -s "MQInput1.transaction.Start,MQOutput1.terminal.catch"
```

Issue the following command on z/OS: z/OS

```
F MI10BRK,rm e='default',f='PurchaseOrder',
     s="MQInput1.transaction.Start,MQOutput1.terminal.catch"
```

Tip: When specifying values for the -s parameter, use the **Event source address** property of the event, not the **Event name** property.

Export a message flow's monitoring profile

About this task

Use the following command to export a profile to file `myMonProf.xml`

Procedure

Issue the following command on Linux, Unix, or Windows:

Linux UNIX Windows

```
mqsireportflowmonitoring BROKER1 -e default -f PurchaseOrder -x -p myMonProf.xml
```

Issue the following command on z/OS: z/OS

```
F MI10BRK,rm e='default',f='PurchaseOrder',x='yes',p='myMonProf.xml'
```

If monitoring for a message flow is configured using monitoring properties, rather than a monitoring profile configurable service, the command creates and returns the equivalent monitoring profile XML file.

Tip: This is an alternative to using an XML editor to create a monitoring profile XML file.

Related tasks:

"Business-level monitoring" on page 3319
You can configure your message flow to emit event messages that can be used to support transaction monitoring and auditing, and business process monitoring.

"Configuring monitoring event sources using monitoring properties" on page 3327
In the Message flow Editor, use the Monitoring tab on the properties of a node to add one or more monitoring events.

"Configuring monitoring event sources using a monitoring profile" on page 762
You can create a monitoring profile and use the **mqsichangeflowmonitoring** command to configure your message flows to emit monitoring events.

Related reference:

"**mqsichangeflowmonitoring** command" on page 3738
Use the **mqsichangeflowmonitoring** command to enable monitoring of message flows.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

Chapter 13. Troubleshooting and support

If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

About this task

This section contains information about the various techniques that you can use to diagnose problems with WebSphere Message Broker.

Procedure

- "Making initial checks" on page 3347
- "Dealing with problems" on page 3363
- "Using logs" on page 3526
- "Using trace" on page 3533
- "Using dumps and abend files" on page 3558
- "Recovering after failure" on page 3574
- "Contacting your IBM Support Center" on page 3563

What to do next

You can also read the general troubleshooting guidance in the following topics:

- "Troubleshooting overview"
- "Searching knowledge bases" on page 3569
- "Getting product fixes" on page 3570
- "Contacting IBM Software Support" on page 3571
- "IBM Support Assistant Data Collector" on page 3565

If a WebSphere Message Broker component or command has returned an error, and you want further information about a message written to the screen or the log, you can browse for details of the message in Diagnostic messages; the lists of messages grouped in numeric order.

For information that is specific to debugging message flow applications, see Chapter 10, "Testing and debugging message flow applications," on page 3143.

Related reference:

"Troubleshooting" on page 6864
Use the reference information in this section to help you diagnose errors in WebSphere Message Broker.

Troubleshooting overview

Troubleshooting is the process of finding and eliminating the cause of a problem. Whenever you have a problem with your IBM software, the troubleshooting process begins as soon as you ask yourself "what happened?"

A basic troubleshooting strategy at a high level involves:

1. "Recording the symptoms of the problem" on page 3346

2. “Re-creating the problem”
3. “Eliminating possible causes”

Recording the symptoms of the problem

Depending on the type of problem that you have, whether it be with your application, your server, or your tools, you might receive a message that indicates that something is wrong. Always record the error message that you see. As simple as this sounds, error messages sometimes contain codes that might make more sense as you investigate your problem further. You might also receive multiple error messages that look similar but have subtle differences. By recording the details of each one, you can learn more about where your problem exists.

Sources of error messages:

- Problems view
- Local error log
- Eclipse log
- User trace
- Service trace
- Error dialog boxes
- **mqsiexplain** command

Re-creating the problem

Think back to what steps you were doing that led to the problem. Try those steps again to see if you can easily re-create the problem. If you have a consistently repeatable test case, it is easier to determine what solutions are necessary.

- How did you first notice the problem?
- Did you do anything different that made you notice the problem?
- Is the process that is causing the problem a new procedure, or has it worked successfully before?
- If this process worked before, what has changed? (The change can refer to any type of change that is made to the system, ranging from adding new hardware or software, to reconfiguring existing software.)
- What was the first symptom of the problem that you witnessed? Were there other symptoms occurring around the same time?
- Does the same problem occur elsewhere? Is only one machine experiencing the problem or are multiple machines experiencing the same problem?
- What messages are being generated that could indicate what the problem is?

You can find more information about these types of question in “Making initial checks” on page 3347.

Eliminating possible causes

Narrow the scope of your problem by eliminating components that are not causing the problem. By using a process of elimination, you can simplify your problem and avoid wasting time in areas that are not responsible. Consult the information in this product and other available resources to help you with your elimination process.

- Has anyone else experienced this problem? See: “Searching knowledge bases” on page 3569.
- Is there a fix you can download? See: “Getting product fixes” on page 3570.

Related tasks:

Chapter 13, "Troubleshooting and support," on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

"Making initial checks"
Before you start problem determination in detail, consider whether there is an obvious cause of the problem, or an area of investigation that is likely to give useful results. This approach to diagnosis can often save a lot of work by highlighting a simple error, or by narrowing down the range of possibilities.

"Dealing with problems" on page 3363
Learn how to resolve some of the typical problems that can occur.

"Using logs" on page 3526
There are a variety of logs that you can use to help with problem determination and troubleshooting.

"Using trace" on page 3533
You can use different types of trace to help you with problem determination and troubleshooting.

"Using dumps and abend files" on page 3558
A dump or an abend file might be produced when a problem occurs. Dumps and abend files can be used by IBM to resolve the problem.

"Searching knowledge bases" on page 3569
If you have a problem with your IBM software, you want it resolved quickly. Begin by searching the available knowledge bases to determine whether the resolution to your problem is already documented.

"Getting product fixes" on page 3570
A product fix might be available to resolve your problem. You can determine what fixes are available from the IBM support site.

"Contacting IBM Software Support" on page 3571
IBM Software Support provides assistance with product defects.

"Recovering after failure" on page 3574
Follow a set of procedures to recover after a serious problem.

Related reference:

"**mqsiexplain** command" on page 3879
Use the **mqsiexplain** command to display the contents of a WebSphere Message Broker BIP message.

"Troubleshooting" on page 6864
Use the reference information in this section to help you diagnose errors in WebSphere Message Broker.

Making initial checks

Before you start problem determination in detail, consider whether there is an obvious cause of the problem, or an area of investigation that is likely to give useful results. This approach to diagnosis can often save a lot of work by highlighting a simple error, or by narrowing down the range of possibilities.

About this task

This section contains a list of questions to consider. As you go through the list, make a note of anything that might be relevant to the problem. Even if your observations do not suggest a cause straight away, they might be useful later if you have to carry out a systematic problem determination exercise.

Procedure

- "Has WebSphere Message Broker run successfully before?"
- "Did you log off Windows while WebSphere Message Broker components were active?" on page 3349
- "Are the Linux and UNIX environment variables set correctly?" on page 3350
- "Are there any error messages or return codes that explain the problem?" on page 3350
- "Can you reproduce the problem?" on page 3352
- "Has the message flow run successfully before?" on page 3353
- "Have you made any changes since the last successful run?" on page 3355
- "Is there a problem with descriptive text for a command?" on page 3356
- "Is there a problem with a database?" on page 3356
- "Is there a problem with the network?" on page 3357
- "Does the problem affect all users?" on page 3358
- "Have you recently changed a password?" on page 3359
- "Have you applied any service updates?" on page 3359
- "Do you have a component that is running slowly?" on page 3360
- "Additional checks for z/OS users" on page 3361

Related tasks:

Chapter 13, "Troubleshooting and support," on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

Related reference:

"Troubleshooting" on page 6864
Use the reference information in this section to help you diagnose errors in WebSphere Message Broker.

Has WebSphere Message Broker run successfully before?

If you have successfully run a WebSphere Message Broker component, determine whether your problem is caused by incorrect installation and setup, or a different cause. If it has *not* run successfully before, work through the information to determine what is preventing successful operation.

About this task

Complete the following steps:

Procedure

1. **Check your setup**

   WebSphere Message Broker might not be set up correctly.

   Linux UNIX On Linux and UNIX operating systems, check that you have set up the command environment correctly; see "Setting up a command environment" on page 213 for more information.

2. **Verify the installation**

   Check "Verifying your WebSphere Message Broker installation" on page 290 for information about how you can verify a basic configuration on your system. This topic describes how to verify your installation on Linux on x86, Linux on x86-64, or Windows by using either the WebSphere Message Broker Toolkit or

the WebSphere Message Broker Explorer. On Linux on x86 and Windows, you can use the Default Configuration wizard and the Samples Preparation wizard to help you with basic configuration and verification.

For more detailed configuration information, refer to Chapter 7, “Configuring brokers for test and production environments,” on page 579.

Related tasks:

“Installing complementary products” on page 300
WebSphere Message Broker works with several other products to provide complementary services.

Chapter 7, “Configuring brokers for test and production environments,” on page 579
Create one or more brokers on one or more computers, and configure them on your test and production systems to process messages that contain your business data.

“Creating a default configuration” on page 564
Use the Default Configuration wizard to create and test a basic broker configuration.

Chapter 13, “Troubleshooting and support,” on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

“Making initial checks” on page 3347
Before you start problem determination in detail, consider whether there is an obvious cause of the problem, or an area of investigation that is likely to give useful results. This approach to diagnosis can often save a lot of work by highlighting a simple error, or by narrowing down the range of possibilities.

Did you log off Windows while WebSphere Message Broker components were active?

How to avoid problems caused by logging off while components are active.

About this task

Windows On Windows, logging off when a WebSphere Message Broker component (a broker) is active can cause a problem.

You might see messages, including BIP2070, BIP2642, BIP1102, and BIP1103, in the Windows Event log.

When you log off, any queue manager that supports the WebSphere Message Broker component is stopped unless it is defined to run as a Windows service. The component runs Windows services and remains active, but it finds that the queue manager and queue manager objects that are associated with it are no longer available.

Procedure

To avoid this problem, set the queue manager that is used by the broker to run as a Windows service, so that logging off Windows from does not cause this problem.

Related tasks:

Chapter 13, “Troubleshooting and support,” on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the

problems.

"Making initial checks" on page 3347
Before you start problem determination in detail, consider whether there is an obvious cause of the problem, or an area of investigation that is likely to give useful results. This approach to diagnosis can often save a lot of work by highlighting a simple error, or by narrowing down the range of possibilities.

"Windows: Viewing the local error log" on page 3527
The Windows Event Viewer is where WebSphere Message Broker writes records to the local system. Use Windows system facilities to view this log.

Are the Linux and UNIX environment variables set correctly?

Use the **mqsiprofile** command to set a command environment.

About this task

UNIX On Linux and UNIX systems, the basic settings are made by the **mqsiprofile** command, which is located in the following directory:

install_dir/bin

What to do next

See "Setting up a command environment" on page 213 for information about setting up the command environment; see "Creating a broker on Linux and UNIX systems" on page 615 for instructions about creating a broker on your operating system.

Related tasks:

Chapter 13, "Troubleshooting and support," on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

"Making initial checks" on page 3347
Before you start problem determination in detail, consider whether there is an obvious cause of the problem, or an area of investigation that is likely to give useful results. This approach to diagnosis can often save a lot of work by highlighting a simple error, or by narrowing down the range of possibilities.

"Installing complementary products" on page 300
WebSphere Message Broker works with several other products to provide complementary services.

"Setting up a command environment" on page 213
After you have installed the product on one of the distributed systems, you must initialize the environment before you can use a runtime component or command.

Related reference:

"Environment variables after installation" on page 3642
On distributed systems, ensure that your environment is set up correctly.

Are there any error messages or return codes that explain the problem?

You can find details of error messages and return codes in several places.

Procedure

- **BIP messages and reason codes**

WebSphere Message Broker error messages have the prefix BIP. If you receive any messages with this prefix (for example, in the UNIX, Linux, or z/OS syslog), you can search for them in the information center for an explanation.You can also view the full content of a BIP message by using the **mqsiexplain** command. For more information, see "**mqsiexplain** command" on page 3879.

Windows In the Windows Event log, references to BIP messages are identified by the source name "WebSphere Broker v6000", where v6000 can be replaced by a number representing the exact service level of your installation, for example 6001.

WebSphere Message Broker messages that have a mixture of identifiers such as BIPmsgs, BIPv700, BIPv610, BIPv500, WMQIv210, MQSIv202, and MQSIv201 indicate a mixed installation, which does not work properly.

- **Other messages**

  For messages with a different prefix, such as AMQ or CSQ for WebSphere MQ, or SQL for DB2, see the appropriate messages and codes documentation for a suggested course of action to help resolve the problem.

  Messages that are associated with the startup of WebSphere Message Broker, or were issued while the system was running before the error occurred, might indicate a system problem that prevented your application from running successfully.

  A large or complex WebSphere Message Broker broker environment might require some additional configuration of the environment beyond what is recommended in "Installing complementary products" on page 300. The need for such configuration changes is typically indicated by warning or error messages that are logged by the various components, including WebSphere MQ, the databases, and the operating system. These messages are normally accompanied by a suggested user response.

Related tasks:

Chapter 13, "Troubleshooting and support," on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

"Making initial checks" on page 3347
Before you start problem determination in detail, consider whether there is an obvious cause of the problem, or an area of investigation that is likely to give useful results. This approach to diagnosis can often save a lot of work by highlighting a simple error, or by narrowing down the range of possibilities.

"Windows: Viewing the local error log" on page 3527
The Windows Event Viewer is where WebSphere Message Broker writes records to the local system. Use Windows system facilities to view this log.

Related reference:

Diagnostic messages
Diagnostic messages are listed in this section in numeric order, grouped according to the component to which they relate.

Can you see all of your files and folders?

How to show all files in Windows Explorer:

About this task

If you are using Windows Explorer to view your files and you cannot see all of your files and folders, such as the broker workpath directory, this is because

Windows Explorer, by default, hides some files and folders.

Procedure

1. Click **Tools** > **Folder options**. The Folder Options dialog box opens.
2. Click the **View** tab and select **Show hidden files and folders**.

Related tasks:

Chapter 13, "Troubleshooting and support," on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

"Making initial checks" on page 3347
Before you start problem determination in detail, consider whether there is an obvious cause of the problem, or an area of investigation that is likely to give useful results. This approach to diagnosis can often save a lot of work by highlighting a simple error, or by narrowing down the range of possibilities.

"Changing the location of the work path" on page 1011
The work path directory is the location where a component stores internal data, such as installation logs, component details, and trace output. The shared-classes directory is also located in the work path directory and is used for deployed Java code. If the work path directory does not have enough capacity, redirect the directory to another file system that has enough capacity.

Can you reproduce the problem?

If you can reproduce the problem, consider the conditions under which you can do so.

Procedure

- **Is the problem caused by a particular message flow?** If so, use the debugging facility of the WebSphere Message Broker Toolkit and user tracing to identify the problem.
- **Is the problem caused by a command?** On distributed operating systems, you issue commands at the system command line.

  z/OS On z/OS, you can issue commands from the console, the syslog, or by submitting a batch job. You enter customization commands from an OMVS session. Console commands that you enter from the console or syslog might be converted to uppercase, depending on the system configuration. This conversion can cause some commands, such as **mqsichangetrace**, to fail, especially if these commands contain parameters that must be lowercase. An error message indicating that the execution group is not available might be caused by the execution group name being in the wrong case. The same thing can happen on message flows.
- **Does a problem command work if it is entered by another user ID?**

  If the command works when it is entered by another user ID, check the environment of each user. Paths, especially shared library paths, might be different. On Windows, UNIX systems, and Linux verify that all users have set up their command environment correctly; refer to sample profile for more information.

  Windows On Windows, the environment for the broker is determined by the system settings, not by a particular user's variables. However, the user's variables affect non-broker commands.

UNIX On LinuxUNIX systems, only the service ID that is specified when the broker was created can start a broker.

Windows On Windows, any authorized user can start a broker.

Related concepts:

"Flow debugger overview" on page 3158
Use the flow debugger in the WebSphere Message Broker Toolkit to track messages through your message flows.

"Trace" on page 6871
If you cannot get enough information about a particular problem from the entries that are available in the various logs, the next troubleshooting method to consider is using trace. Trace provides more details about what is happening while code runs. The information produced from trace is sent to a specified trace record, so that you or IBM support personnel can analyze it to discover the cause of your problem.

Related tasks:

Chapter 13, "Troubleshooting and support," on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

"Making initial checks" on page 3347
Before you start problem determination in detail, consider whether there is an obvious cause of the problem, or an area of investigation that is likely to give useful results. This approach to diagnosis can often save a lot of work by highlighting a simple error, or by narrowing down the range of possibilities.

"Are the Linux and UNIX environment variables set correctly?" on page 3350
Use the **mqsiprofile** command to set a command environment.

"Using trace" on page 3533
You can use different types of trace to help you with problem determination and troubleshooting.

"Resolving problems when developing message flows" on page 3395
Use the advice given here to help you to resolve common problems that can arise when developing message flows.

Related reference:

"Commands" on page 3672
All WebSphere Message Broker Toolkit and runtime commands that are provided on distributed systems are listed, grouped by function, with references to command details.

Has the message flow run successfully before?

Sometimes a problem appears in a message flow that has previously run successfully.

About this task

To identify the cause of the problem, answer the following questions:

Procedure

- **Have you made any changes to the message flow since it last ran successfully?**

  If so, it is likely that the error exists somewhere in the new or modified part of the flow. Examine the changes and see if you can find an obvious reason for the problem.

- **Have you used all the functions of the message flow before?**

  Did the problem occur when you used part of the message flow that had never been invoked before? If so, it is likely that the error exists in that part. Try to find out what the message flow was doing when it failed by using user tracing, trace nodes, and the WebSphere Message Broker Toolkit debugger function.

  If you have run a message flow successfully on many previous occasions, check the current queue status and the files that were being processed when the error occurred. It is possible that they contain some unusual data value that invokes a rarely-used path in the message flow.
- **Does the message flow check all return codes?**

  Has your system been changed, perhaps in a minor way, but your message flow does not check the return codes it receives as a result of the change? For example:

  – Does your message flow assume that the queues that it accesses can be shared? If a queue has been redefined as exclusive, can your message flow deal with return codes indicating that it can no longer access that queue?

  – Have any security profiles been changed? A message flow could fail because of a security violation.
- **Does the message flow expect particular message formats?**

  If a message with an unexpected message format has been put onto a queue (for example, a message from a queue manager on a different operating system) it might require data conversion or a different form of processing. Also, check whether you have changed any of the message formats that are used.
- **Does the message flow run on other WebSphere Message Broker systems?**

  Is there something different about the way that your system is set up that is causing the problem? For example, have the queues been defined with the same maximum message length or priority? Are there differences in the databases used, or their setup?
- **Are you using any user-defined extensions?**

  There might be translation or compilation problems with loadable implementation library (LIL) files. Before you look at the code, examine the output from the translator, the compiler or assembler, and the linkage editor, to see if any errors have been reported. Fix any errors to make the user-defined extension work.

  If the documentation shows that each of these steps was completed without error, consider the coding logic of the message flow, message set, or user-defined extension. Do the symptoms of the problem indicate which function is failing and, therefore, which piece of code is in error? See “User-defined extensions overview” on page 2971 for more information.
- **Can you see errors from WebSphere Message Broker or external resources, such as databases?**

  Your message flow might be losing errors because of incorrect use of the failure terminals on built-in nodes. If you use the failure terminals, make sure that you handle errors adequately. See “Handling errors in message flows” on page 2823 for more information about failure terminals.

Related concepts:

“Trace” on page 6871
If you cannot get enough information about a particular problem from the entries that are available in the various logs, the next troubleshooting method to consider is using trace. Trace provides more details about what is happening while code runs. The information produced from trace is sent to a specified trace record, so that you or IBM support personnel can analyze it to discover the cause of your

problem.

Related tasks:

Chapter 13, "Troubleshooting and support," on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

"Making initial checks" on page 3347
Before you start problem determination in detail, consider whether there is an obvious cause of the problem, or an area of investigation that is likely to give useful results. This approach to diagnosis can often save a lot of work by highlighting a simple error, or by narrowing down the range of possibilities.

"Resolving problems when developing message flows" on page 3395
Use the advice given here to help you to resolve common problems that can arise when developing message flows.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

Related reference:

"User trace" on page 6873
User trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Administration log. User trace is inactive by default; you must activate it explicitly by using a command, or by selecting options in the WebSphere Message Broker Toolkit.

Have you made any changes since the last successful run?

Have you changed WebSphere Message Broker, any related software, or any hardware?

About this task

When you are considering changes that might have been made recently, think not only about WebSphere Message Broker, but also about the other programs with which it interfaces, the hardware, device drivers, and any new applications.

Procedure

- **Have you changed your initialization procedure?**

  On z/OS, have you changed any data sets or library definitions? Has the operating system been initialized with different parameters? Check for error messages generated during initialization.

- **Has the profile of the user who is running the commands on Linux or UNIX systems been changed?**

  If so, this might mean that the user no longer has access to the required objects and commands.

- **Has any of the software on your system been upgraded to a later release?**

  Check that the upgrade completed successfully, and whether the new software is compatible with WebSphere Message Broker (check the product readme.html file).

- **Do your message flows deal with the errors and return codes that they might get as a result of any changes that you have made?**

Check that your message flow can handle all possible error situations.

Related tasks:

Chapter 13, “Troubleshooting and support,” on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

“Making initial checks” on page 3347
Before you start problem determination in detail, consider whether there is an obvious cause of the problem, or an area of investigation that is likely to give useful results. This approach to diagnosis can often save a lot of work by highlighting a simple error, or by narrowing down the range of possibilities.

“Handling errors in message flows” on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

Is there a problem with descriptive text for a command?

On Linux and UNIX systems, be careful when including special characters (backslash (\) and double quotation mark (") characters) in descriptive text for some commands.

About this task

If you use either of these characters in descriptive text, precede them with the escape character backslash (\) ; that is, enter \\ or \" if you want \ or " in your text.

Related tasks:

Chapter 13, “Troubleshooting and support,” on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

“Making initial checks” on page 3347
Before you start problem determination in detail, consider whether there is an obvious cause of the problem, or an area of investigation that is likely to give useful results. This approach to diagnosis can often save a lot of work by highlighting a simple error, or by narrowing down the range of possibilities.

Related reference:

“Commands” on page 3672
All WebSphere Message Broker Toolkit and runtime commands that are provided on distributed systems are listed, grouped by function, with references to command details.

“Characters allowed in commands” on page 3680
You must adhere to a few rules when you provide names or identifiers for the components and resources in your broker environment.

Is there a problem with a database?

If you have database problems, complete a set of initial checks to identify errors.

Procedure

- Check that the database is started.
- Check that you have correctly completed ODBC configuration:

– Linux UNIX On Linux and UNIX systems, check that you have created a copy of the sample ODBC configuration file (`odbc.ini`), and have modified them for your environment, and that you have not added any extra unsupported parameters.

– Windows On Windows systems, click **Start** > **Control Panel** > **Administrative Tools** > **Data Sources (ODBC)** to configure the connections you require.

Detailed instructions for setting up ODBC connections on distributed systems are provided in "Enabling ODBC connections to the databases" on page 668.

- Use the "**mqsicvp** command" on page 3857 to help you with ODBC database diagnostics.
- Check that you have correctly completed JDBC configuration. Detailed instructions for setting up JDBC connections are provided in "Enabling JDBC connections to the databases" on page 683.
- Check the number of database connections that are in use on DB2 for AIX. If you use local mode connections, a maximum of 10 is supported.
- If messages that indicate that `imbdfdb2v6.lil` failed to load, check that you have installed a supported database. Details of database managers and versions are given in "Supported databases" on page 3591.

Related tasks:

Chapter 13, "Troubleshooting and support," on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

"Making initial checks" on page 3347
Before you start problem determination in detail, consider whether there is an obvious cause of the problem, or an area of investigation that is likely to give useful results. This approach to diagnosis can often save a lot of work by highlighting a simple error, or by narrowing down the range of possibilities.

"ODBC trace" on page 3551
You can use various methods to trace for ODBC activity, depending on the operating system that you are using.

"Resolving problems when using databases" on page 3491
Use the advice given here to help you to resolve problems that can arise when using databases.

"Capturing database state" on page 2512
If an error occurs when the broker accesses an external database, you can either let the broker throw an exception during node processing or use ESQL statements to process the exception within the node itself.

Related reference:

"Supported databases" on page 3591
You can optionally configure databases to contain data that is accessed by your message flows. Databases from IBM and other suppliers are supported at specific versions on supported operating systems.

"**mqsicvp** command" on page 3857
Use the **mqsicvp** command to perform verification tests on a broker, or to verify ODBC connections.

Is there a problem with the network?

WebSphere Message Broker uses WebSphere MQ for inter-component communication. If components are on separate queue managers, they are connected by message channels.

About this task

There can be communication problems between any of these:

Procedure

- Brokers
- The WebSphere Message Broker Explorer
- The WebSphere Message Broker Toolkit

What to do next

If any two components are on different queue managers, make sure that the channels between them are working. Use the WebSphere MQ **display chstatus** command to see if messages are flowing.

Use the **ping** command to check that the remote computers are connected to the network, or if you suspect that the problem might be with the network itself. For example, use the command **ping** *brokername*, where *brokername* is a computer name. If you get a reply, the computer is connected. If you don't get a reply, ask your network administrator to investigate the problem. Further evidence of network problems might be messages building up on the transmission queues.

Related tasks:

Chapter 13, “Troubleshooting and support,” on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

“Making initial checks” on page 3347
Before you start problem determination in detail, consider whether there is an obvious cause of the problem, or an area of investigation that is likely to give useful results. This approach to diagnosis can often save a lot of work by highlighting a simple error, or by narrowing down the range of possibilities.

Does the problem affect all users?

On distributed systems, problems can be caused by different users having different environments.

About this task

Check whether there is a different user ID in an incoming message, or a different user ID issuing a command (or acting as the broker's service ID).

Related tasks:

Chapter 13, “Troubleshooting and support,” on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

“Making initial checks” on page 3347
Before you start problem determination in detail, consider whether there is an obvious cause of the problem, or an area of investigation that is likely to give useful results. This approach to diagnosis can often save a lot of work by highlighting a simple error, or by narrowing down the range of possibilities.

“Securing WebSphere MQ resources” on page 1559
Secure the WebSphere MQ resources that your broker configuration requires.

Have you recently changed a password?

Check that you have updated passwords correctly.

About this task

If you have changed the operating system password for one of the following user IDs, the broker or the deployed message flows might have access problems:

- The ID that you have specified for the broker's **serviceUserId** on Windows
- The ID that you have specified for access to a database

If these problems occur, complete one or more of the following steps:

Procedure

- Change the properties of the broker to reflect the password change. Use the **mqsichangebroker** command to change the appropriate parameters.
- Run the **mqsisetdbparms** command to redefine the correct values for the user ID and password.
- Ensure that your passwords do not contain reserved keywords. For example, "IBM" is a reserved keyword in DB2.

Related tasks:

Chapter 13, "Troubleshooting and support," on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

"Making initial checks" on page 3347
Before you start problem determination in detail, consider whether there is an obvious cause of the problem, or an area of investigation that is likely to give useful results. This approach to diagnosis can often save a lot of work by highlighting a simple error, or by narrowing down the range of possibilities.

"Securing WebSphere MQ resources" on page 1559
Secure the WebSphere MQ resources that your broker configuration requires.

Related reference:

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

Have you applied any service updates?

Check what service updates you have applied to your software.

About this task

If you have applied an APAR or a PTF to WebSphere MQ for z/OS, or a fix pack or interim fix has been applied to a distributed operating system, check that no error message was produced. If the installation was successful, check with the IBM Support Center for any known error with the service update.

Procedure

- If a service update has been applied to any other product, consider the effect that it might have on WebSphere Message Broker.

- Ensure that you have followed any instructions in the service update that affect your system. For example, you might need to redefine a resource, or stop and restart a component.
- If you are not sure whether a service update has been applied to your system, search for and view the release notes stored in the product installation directory. These notes include the service level and details of the maintenance that you have applied.

Related tasks:

"Finding the latest information" on page 232
Access the latest information for WebSphere Message Broker.

"Applying service to the Broker component" on page 314
Apply maintenance updates and program fixes to the Broker component.

"Applying service to the WebSphere Message Broker Toolkit" on page 325
Apply maintenance updates and program fixes to the WebSphere Message Broker Toolkit.

Chapter 13, "Troubleshooting and support," on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

"Making initial checks" on page 3347
Before you start problem determination in detail, consider whether there is an obvious cause of the problem, or an area of investigation that is likely to give useful results. This approach to diagnosis can often save a lot of work by highlighting a simple error, or by narrowing down the range of possibilities.

Do you have a component that is running slowly?

If a particular component, or the system in general, is running slowly, you can take some actions to improve the performance.

About this task

Consider the following actions:

Procedure

- Check whether tracing is on. You might have started WebSphere Message Broker user tracing or service tracing, ODBC tracing, WebSphere MQ tracing, or native database tracing. If one or more of these traces are active, turn them off.
- Clear out all old abend files from your errors directory. If you do not clear the directory of unwanted files, you might find that your system performance degrades because significant space is used up.
- On Windows, use the workpath **-w** parameter of the **mqsicreatebroker** command to create the errors directory in a hard disk partition that does not contain WebSphere Message Broker or Windows.
- Increase your system memory.

Related concepts:

"Logs" on page 6864
If an error is reported by a WebSphere Message Broker component, start your investigations into its causes by looking at the product and systems logs to which information is written during component operation.

Related tasks:

Chapter 13, “Troubleshooting and support,” on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

“Making initial checks” on page 3347
Before you start problem determination in detail, consider whether there is an obvious cause of the problem, or an area of investigation that is likely to give useful results. This approach to diagnosis can often save a lot of work by highlighting a simple error, or by narrowing down the range of possibilities.

“Using trace” on page 3533
You can use different types of trace to help you with problem determination and troubleshooting.

“Resolving problems with performance” on page 3504
Use the advice given here to help you to resolve common problems with performance.

Related reference:

“**mqsicreatebroker** command” on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

Additional checks for z/OS users

Problem determination for z/OS users.

About this task

On WebSphere Message Broker for z/OS, you might find that:

- The console has unexpected messages; look for an explanation by searching for the message number in the information center, or referring to other relevant messages and codes documentation.
- RRS is not started; issue the command **D A,RRS**.
- The queue manager is not available; issue the WebSphere MQ command **DISPLAY THREAD(*)**.
- DB2 is not working; issue the DB2 command **DISPLAY THREAD**.

The cause of your problem on z/OS could be in any of the following areas. Check each of these, starting with whichever seems the most likely, based on the nature of your problem.

Procedure

- The queue manager address space
- A queue manager in your queue-sharing group
- The channel initiator address space
- Batch or TSO address space
- The z/OS system (including ARM, RRS, or the Coupling Facility)
- The network (including APPC or TCP/IP)
- Another system, for example a queue manager on another operating system or a WebSphere MQ client
- The external security manager product, for example RACF or ACF2
- DB2

What to do next

Understanding interactions between the runtime components

There are two runtime components: the broker and the execution group. These components communicate with each other by exchanging command requests inside WebSphere MQ messages to perform actions, such as deploying a message flow.

After completion of the command request, responses are sent back to the originating component, indicating whether the request was successful.

On WebSphere Message Broker for z/OS, you see extra information messages issued by z/OS runtime components that allow you and IBM Service personnel to determine the interactions between the various runtime components, including the Configuration Manager, broker and execution group.

When the broker receives command requests the broker issues a message that identifies the command request. When this request completes, a message is issued that indicates whether the command is successful. Each command request that the broker receives results in at least one matching command request being sent to an execution group, and a corresponding message being issued. Every response from an execution group that results from these command requests result in a message being issued. These messages can be turned on and off and are to be used by the IBM Service team.

When an execution group receives command requests from the broker, the execution group issues information messages that identify the command request. For actions that are contained within a command request, an information message is issued that identifies the action to be performed and the resource upon which the action is to take effect. When the request completes, the execution group issues a message that indicates that the message has been processed.

Loading IBM and user-defined nodes and parsers

When an execution group starts, it loads all available IBM and user-defined nodes and parsers. For each library that is loaded, two messages are issued. One is issued before the library is loaded and one is issued after the library has been loaded.

Related tasks:

Chapter 13, “Troubleshooting and support,” on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

“Making initial checks” on page 3347
Before you start problem determination in detail, consider whether there is an obvious cause of the problem, or an area of investigation that is likely to give useful results. This approach to diagnosis can often save a lot of work by highlighting a simple error, or by narrowing down the range of possibilities.

“Using WebSphere MQ shared queues for input and output (z/OS)” on page 1546
On z/OS systems, you can define WebSphere MQ shared queues as input and output queues for message flows. You might need to serialize access to those messages.

Related reference:

“Other sources of diagnostic information on z/OS” on page 6891
Other files that you might find useful for problem determination.

Dealing with problems

Learn how to resolve some of the typical problems that can occur.

Before you begin

Before you start:

Make initial checks, see “Making initial checks” on page 3347, and check the following locations to see if you can rectify the problem:

- Logs, see “Using logs” on page 3526
- Trace, see “Using trace” on page 3533
- Dumps, see “Using dumps and abend files” on page 3558

About this task

This section contains the following topics:

Procedure

Related tasks:

Chapter 13, “Troubleshooting and support,” on page 3345

If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the

problems.

"Making initial checks" on page 3347
Before you start problem determination in detail, consider whether there is an obvious cause of the problem, or an area of investigation that is likely to give useful results. This approach to diagnosis can often save a lot of work by highlighting a simple error, or by narrowing down the range of possibilities.

"Contacting your IBM Support Center" on page 3563
If you cannot resolve problems that you find when you use WebSphere Message Broker, or if you are directed to do so by an error message generated by WebSphere Message Broker, you can request assistance from your IBM Support Center.

Resolving problems when running commands

Use the advice given here to help you to resolve common problems that can arise when you run commands.

Backing up or restoring the broker returns error BIP1253 or BIP1262

Procedure

- **Scenario:** An error message is generated when you back up or restore a broker:
  - The following message is returned when you run the **mqsibackupbroker** command:

    `BIP1253E Failed to backup the broker 'broker_name' (error 'error_code')`
  - The following message is returned when you run the **mqsirestorebroker** command:

    `BIP1262E Could not delete the existing configuration of broker 'broker_name'.`
- **Explanation:** Both backup and restore commands require read/write access to the files that contain broker configuration data. Access has failed and the command cannot complete its operation successfully.
- **Solution:** Typically, you might see one of these errors if you are running the backup or restore command against an active broker, and a conflict of access has occurred. If your broker is active, use the **mqsistop** command to stop the broker, and try the operation again.

  You might also see one of these errors if the directory to which the command is writing, or from which it is reading, is temporarily inaccessible because of communications or authorization problems. Check that your user ID has access to the directory that you specified in the command.

ReportEvent() error message is issued on Windows when you attempt to run a command

Procedure

- **Scenario:** A ReportEvent () message is generated whenever you attempt to run any `mqsi*` command, The ReportEvent () message is followed by the result of the command itself.
- **Explanation:** The Windows Application Log has become full.
- **Solution:** Clear the Windows Application Log from the Event Viewer.

You want to run a command that uses SSL to administer a remote broker over a secured channel.

Procedure

- **Scenario:** You want to run an `mqsi*` command using SSL to administer a remote broker over a secured channel. Note, that you can do this only for `mqsi*` commands that have the -n `.broker` option.
- **Solution:** In order to connect to a broker using SSL, you must specify the keystore and truststore password for the connection using the IBM_JAVA_OPTIONS environment variable. For example:

  ```
  export IBM_JAVA_OPTIONS=-Djavax.net.ssl.keyStorePassword=MYKEYSTOREPASSWORD
    -Djavax.net.ssl.trustStorePassword=MYTRUSTSTOREPASSWORD
  ```

  In the same environment, you must then run the command using the -n option to specify a `.broker` file that describes the connection you want to establish.

  You can export `.broker` files by using the export option in the WebSphere Message Broker Explorer. To do this, right-click on the secured broker you want to export a `.broker` file for, and select **Export *.broker**.

A time-out error is issued when you attempt to run a command on AIX when Stack Execution Disable (SED) is enabled

Procedure

- **Scenario:** You want to run an `mqsi*` command on AIX with Stack Execution Disable enabled, but the command times out even if the **-w** option is set to 360.
- **Explanation:** `mqsi*` commands attempt to create a JVM with Java Native Interface (JNI) calls. Because of Stack Execution Disable, the JVM creation fails. For more information about Stack Execution Disable, see AIX Stack Execution Disable.
- **Solution:** Use the **sedmgr** command to create exemption records from Stack Execution Disable. For example, for **mqsilist**:

  ```
  sedmgr c exempt /opt/mqsi/7.0/bin/mqsilist
  ```

  The following commands might be affected by this issue:

  - **biphttplistener**
  - **bipbroker**
  - **bipservice**
  - **mqsideploy**
  - **mqsilist**
  - **mqsimode**
  - **mqsireloadsecurity**
  - **mqsireportresourcestats**
  - **mqsiwebuseradmin**

Related tasks:

Chapter 13, “Troubleshooting and support,” on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

“Dealing with problems” on page 3363
Learn how to resolve some of the typical problems that can occur.

“Windows: Viewing the local error log” on page 3527
The Windows Event Viewer is where WebSphere Message Broker writes records to the local system. Use Windows system facilities to view this log.

Related reference:

"**mqsistop** command" on page 3972
Use the **mqsistop** command to stop the specified component.

"**mqsibackupbroker** command" on page 3720
Use the **mqsibackupbroker** command to back up the current configuration of a broker.

"**mqsirestorebroker** command" on page 3952
Use the **mqsirestorebroker** command to restore the broker configuration from a backup file.

Resolving problems when running samples

Use the advice given here to help you to resolve common problems that can arise when you run or remove samples.

About this task

Use the following instructions to diagnose the problem.

1. Use WebSphere MQ Explorer to determine which queue the input message is on:
   a. Start WebSphere MQ Explorer.
   b. Expand the folders to display the broker queue manager, MB7QMGR.
   c. Click the Queues folder in the queue manager to display its queues.
   d. Check the Current depth column to identify the queue that is holding the input message. If several messages are stored on one queue, right-click the queue, then click **Browse Messages** to determine if the message that you are interested in is on the queue.
2. Use the following table to identify the problem, and a suggested solution to overcome it. If the sample that you are running does not use a database, ignore the database-related problems listed in the table.
3. If the table did not help you solve the problem, return to the WebSphere Message Broker Toolkit and check the Problems view for error messages. Use this information to solve the problem.
4. If you created the sample yourself, you must check that all the objects in the sample have been named and configured correctly.

Problem	Reason	Suggested solution
The input message stays on the IN queue.	The broker, the queue manager, the listener, or the message flow itself has stopped.	Check that all the components are running and that the listener for the queue manager is listening on the port for the queue manager. Start any components that are not running.
	An unidentifiable message already on the IN queue cannot be processed by the message flow.	In WebSphere MQ Explorer, right-click the IN queue, then click **All tasks** > **Clear Messages**.
The input message goes to the FAIL queue.	The MQInput node cannot identify which parser it must use to parse the message.	If you are using the Enqueue facility in the workbench or the RfhUtil tool that is supplied in SupportPac IH03, you must type all the necessary message header information in the fields in the tool. If you are using the `mqsiput.exe` tool, you must add the header information to the message file itself.

Problem	Reason	Suggested solution
The input message goes to the SYSTEM.DEAD.LETTER.QUEUE	The queue on which the input message was supposed to be put does not exist.	Ensure that you have created all the queues required for the sample.
You cannot find the input message on any queue.	You have not refreshed the display in WebSphere MQ Explorer, or you have refreshed only some of the queues.	To refresh all the queues in WebSphere MQ Explorer, right-click the Queues folder, then click **Refresh**. All the queues in the folder are refreshed.
	The input message was passed to a terminal that was not connected to another node, and the message was discarded.	Ensure that all the nodes are connected to each other as required by the sample.

When using a DB2 database, the input message goes to the FAIL queue or the Event Log contains a message saying that the database was not found, or both.	DB2 is not running.	In a DB2 Command Window, enter the following command:
		`db2 start`
		
		If DB2 is already running, you receive the following message:
		`The database manager is already active.`

	The message flow is trying to access a database table that is not in the default schema. The name of the default schema is determined by, and is the same as, the user name that is used to access the database. If the table is not in the default schema, and no other schema is specified in the ESQL for the message flow, the message flow looks for the table in the default schema.	In a DB2 Command Window, enter the following commands:
		`DB2 "CONNECT TO database user` *`username`*`"`
		`DB2 "CREATE VIEW` *`tablename`*
		`  AS SELECT * FROM` *`tableschema.tablename`*`"`
		
		where:
		• *username* is the user name of the broker
		• *tableschema* is the schema that contains the table that the message flow is accessing
		• *tablename* is the table that the message flow is accessing

You receive the following error messages when you try to remove a DB2 database on Windows:
 	
`BIP9830I: Deleting the DB2 Database` *`Your_database_name`*`.`
 	
`BIP9835E: The DB2 batch command failed with the error code SQLSTATE=57019. The database could not be created/deleted. The error code SQLSTATE=57019 was returned from the DB2 batch command.`
	If you use the DB2 Control Center to perform a query, a connection is opened to the database. This connection stays open until the DB2 Control Center is closed, when the connection is ended.	Close the DB2 Control Center application. To try to remove the sample again, click **Yes**.

Problem	Reason	Suggested solution
You run a web services sample by using the prebuilt Test Client scenario and it hangs, then times out.	The problem occurs when you have a SOAPInput node that is being called by a SOAPRequest node. The default port that web services use is 7800, and the SOAPRequest nodes are set up to use this port. However, if this port is already in use, for example, by another sample, the port number is automatically incremented by one. Therefore, the default port must also be changed to match.	Issue the following **mqsireportproperties** command on one line, to check which port your provider execution group is using: `mqsireportproperties MB7BROKER -e sampleExecutionGroup -o HTTPConnector -n port` where *sampleExecutionGroup* is the appropriate execution group for the sample that is being run. To verify that the port that the SOAPRequest node is using is the correct port to call the provider flow, change the port of the SOAPRequest nodes to the port that the provider execution group is using by completing the following steps: 1. Open the message flow located in the message set project. 2. (Perform this step for all of your SOAPRequest nodes). Open the **HTTP Transport** tab in the Properties view. If the port is not correct, change the port in the `Web service URL` property to the correct port for your web services provider or TCP/IP Monitor. 3. Save the message flow. 4. Rebuild and redeploy the broker archive (BAR) file. If you have set up a TCP/IP Monitor, you have already checked which port the web services provider is using, but you must still configure the consumer to send the messages to your TCP/IP Monitor, then rebuild and redeploy the BAR file. Alternatively, you can remove one of the samples that is using the same port, so that only one sample is deployed at a time.
In some samples, the format of the XML output in the Test Client might be displayed in a different format to the format that is shown in the documentation.	In all cases the output data is identical, it is the format that is different.	You can change the format of the output by selecting either **View as Source** or **View as XML Structure** from the menu in the Test Client.

Related concepts:

"Samples" on page 98
The WebSphere Message Broker Toolkit provides samples that show the features that are available in WebSphere Message Broker, and how to use them. This topic provides links to the information about the individual samples.

Related tasks:

Chapter 13, “Troubleshooting and support,” on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

“Dealing with problems” on page 3363
Learn how to resolve some of the typical problems that can occur.

Resolving problems when creating resources

Use the advice given here to help you to resolve common problems that can occur when you create resources.

About this task

Look for the problem that you have encountered, and follow the guidance provided to recover from the error.

Problems when creating a broker:

- “Message BIP8081 is issued when creating a broker”
- “You cannot create files when creating a broker on AIX”
- “The JCL BIPGEN fails when you create a component on z/OS” on page 3370
- “Your DataFlowEngine ends with an abend when you create a broker on HP-UX using Oracle” on page 3370

Problems when creating other resources:

- “Error message BIP2624 is issued when creating an execution group” on page 3370
- “The Default Configuration wizard fails with invalid argument specified” on page 3371

Message BIP8081 is issued when creating a broker

Procedure

- **Scenario:** Message BIP8081E is displayed when you are creating a broker, the inserted message does not format correctly, and the broker is not created.
- **Explanation:** This problem occurs because you are not a member of the correct group.
- **Solution:** Read the explanation of message BIP8081, and ask your WebSphere Message Broker administrator to give your user ID access to the mqbrkrs group.

You cannot create files when creating a broker on AIX

Procedure

- **Scenario:** When you run the **mqsicreatebroker** command on WebSphere Message Broker for AIX, the following message is displayed:

  ```
  BIP8135E Unable to create files. Operating System return code 1
  ```

- **Explanation:** The user ID that you create for WebSphere Message Broker testing must have a primary group of mqbrkrs. The following example shows an AIX SMIT panel listing the **Change / Show Characteristics of a User**:

```
                Change / Show Characteristics of a User

Type or select values in entry fields.
Press Enter AFTER making all desired changes.
[TOP]                                [Entry Fields]
* User NAME                          peterc
  User ID                            [202]                    #
  ADMINISTRATIVE USER?               false                    +
  Primary GROUP                      [mqbrkrs]                +
  Group SET                          [mqbrkrs,mqm,system,sys> +
```

The JCL BIPGEN fails when you create a component on z/OS

Procedure

- **Scenario:** The BIPGEN job fails when you are copying the broker profile BIPBPROF from the PDSE to the file system.
- **Explanation:** The file system might lack the required space, the component profile might not exist, or you might not have the appropriate authority.
- **Solution:** Make the following checks:
  - The file system has sufficient space. You can check how much space is used and how much is free in a file system using the OMVS command **df -P /pathname**. 100 MB is 3 276 800 512 byte sectors.
  - The profile file exists in the PDSE.
  - Your user ID has the appropriate authority to write to the file system.

Your DataFlowEngine ends with an abend when you create a broker on HP-UX using Oracle

Procedure

- **Scenario:** Your message flow (DataFlowEngine or DFE) ends with an abend when you create a broker on HP-UX using Oracle.
- **Explanation:** This problem occurs when you have installed DB2and Oracle on the same computer.
- **Solution:** Remove the DB2 LIL files that are used by WebSphere Message Broker. For example, issue the following commands:

```
mv install dir/lib/imbdfdb2v6.lil install dir/lib/imbdfdb2v6.blank
mv install dir/lib/imbdfdb2.lil install dir/lib/imbdfdb2.blank
```

Error message BIP2624 is issued when creating an execution group

Procedure

- **Scenario:** When you create an execution group, you get several BIP2624 messages (MQRC=2012 (MQRC_ENVIRONMENT_ERROR)), and no WebSphere MQ messages are processed.
- **Explanation:** You have created the broker to run as a WebSphere MQ trusted application (that is, the broker runs in the same process as the WebSphere MQ queue manager), but the user ID that you specified does not have the required authority.
- **Solution:** If you request the trusted application option on the **mqsicreatebroker** command by specifying the **-t** parameter, perform the appropriate steps for your operating system:

  Windows **Windows**

  Using the **-i** parameter on the **mqsicreatebroker** command, specify a service user ID that is a member of WebSphere MQ group mqm.

Linux UNIX **Linux and UNIX systems**

Specify the user ID mqm on the **-i** parameter on the **mqsicreatebroker** command.

The Default Configuration wizard fails with invalid argument specified

Procedure

- **Scenario:** On a Windows system, the Default Configuration wizard fails. The Default Configuration wizard log contains the following error message:

  ```
  Invalid argument John Smith specified. Argument specified should be well formed.
  Correct and reissue the command.
  ```

- **Explanation:** You have entered a user name that contains one or more spaces. The Default Configuration does not support the use of user names that contain spaces, because the space character can cause problems in communications with other operating systems.
- **Solution:** Use an alternative user name that does not contain any spaces in the Default Configuration wizard.

Resolving problems that occur when you start resources

Use the advice given here to help you to resolve common problems that can occur when you start resources.

About this task

Procedure

- "Error message BIP8048 is issued when you start a component" on page 3382
- "You experience problems with the default configuration" on page 3382
- "A "Not Found" error is issued when you click a link to a specific sample" on page 3383
- "The Quick Tour is displayed as a blank window" on page 3383
- "Error message BIP0832 is issued on startup" on page 3383
- "Your execution groups restart repeatedly" on page 3383
- "You cannot tell whether startup is complete on z/OS" on page 3384
- "Abend code 0C1 is issued when you try to start the DataFlowEngine on z/OS" on page 3385
- "Error message BIP2604 with return code `MQRC_CONNTAG_IN_USE` is issued during the start of a message flow on z/OS" on page 3385
- "After creating or changing a configurable service, you restart your broker but your message flow does not start, and message BIP2275 is issued in the system log or Windows Event Viewer" on page 3386
- "A device allocation error is issued" on page 3387
- "Windows Vista and Windows Server 2008 fail to recognize WebSphere Message Broker digital signatures: "Unknown Publisher"" on page 3387
- "The create command fails, and error message BIP8022 is issued" on page 3388

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related tasks:

"Starting and stopping a broker" on page 921
Run the appropriate command to start or stop a broker.

"Checking APF attributes of `bipimain` on z/OS" on page 607
This task is part of the larger task of setting up your z/OS environment.

Related reference:

"**mqsistart** command" on page 3965
Use the **mqsistart** command to start the specified broker if all initial verification tests complete successfully.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

Related information:

WebSphere MQ Version 7 Information Center online

Resolving problems when starting a broker

Use the advice given here to help you to resolve common problems that can arise when you start a broker.

About this task

When you start a broker by using the **mqsistart** command, the **mqsicvp** command is run automatically to check that the broker environment is set up correctly (for example, the installed level of Java is supported). On Linux and UNIX, the **mqsicvp** command also verifies that the ODBC environment (if specified) is configured correctly. For more information, see "**mqsicvp** command" on page 3857.

For advice about specific problems that can occur when you start a broker, see the following section.

The broker fails to start because there is not enough space in the Java TMPDIR directory or access permissions for the Java TMPDIR directory are inadequate: Procedure

- **Scenario:** The broker fails to start and either an error message indicates that insufficient space is available or a BIP4512 exception indicates a java.lang.NoClassDefFoundError in the stack trace.
- **Explanation:** This error has two possible causes:
  - The broker uses Java JAR files. When the broker starts, the Java runtime environment extracts the JAR files into a temporary directory, Java TMPDIR. On Linux, UNIX, and z/OS computers, the TMPDIR directory is typically `/tmp`; on Windows computers, it is `c:\temp`. If this directory is not large enough to hold the JAR files, the broker does not start.
  - If the program is packaged as a PAR file, the user must have access to the system temporary directory and adequate space must be available in the temporary directory.
- **Solution:** Use one of the following methods to specify the location of this temporary JAR directory:
  - Use the environment variable `TMPDIR`.
  - Set the system property `java.io.tmpdir`.

  Allow at least 50 MB of space per execution group in this directory for WebSphere Message Broker components. You might need more space if you

deploy large user-defined nodes or other JARs to the broker. You should ensure that all the dependencies of the compute node class are deployed to the broker.

Diagnostic message ICH408I is issued on z/OS when your broker fails to start: About this task

Two scenarios are described here. Choose the appropriate one.

Procedure

- **Scenario 1:** The following diagnostic message is written to the SDSF SYSLOG on z/OS when your broker fails to start:

```
ICH408I USER(MA10USR ) GROUP(TSOUSER ) NAME(OTHER, A N (ANO) 484
  /argo/MA10BRK/ENVFILE
-                                        --TIMINGS (MINS.)--
 ----PAGING COUNTS---
-JOBNAME  STEPNAME PROCSTEP    RC   EXCP    CPU    SRB  CLOCK   SERV  PG
   PAGE   SWAP    VIO SWAPS
  CL(DIRSRCH ) FID(01D7D3E2E3F1F9002D08000000000003)
  INSUFFICIENT AUTHORITY TO LOOKUP
  ACCESS INTENT(--X)  ACCESS ALLOWED(OTHER ---)
```

- **Explanation:** The started task ID, under which the broker runs, must be in a RACF or z/OS UNIX System Services (USS) group that has rwx permissions on the broker directory. For example, consider a broker that is created under directory `/argo/MA00BRK`. It runs under started task ID MA00USR. Issuing the **`ls -al`** command from the root directory `/` to find the permission bit settings on `/argo` returns:

```
drwxrwx---  5 BPXROOT  ARGOUSR      8192 Jul 30  13:57  argo
```

  If you issue the **`id MA00USR`** command to find the group membership of started task ID MA00USR you see:

```
uid=14938(MA00USR) gid=5(TSOUSER) groups=229(ARGOUSR)
```

  These results show that the started task ID MA00USR potentially has rwx permissions on subdirectories to `/argo`, because these permissions are set for both the user and the group that is associated with MA00USR. If the permissions are not set correctly, you see the type of diagnostic message shown in the scenario.

- **Solution:** Make sure that the started task ID, under which the broker runs, is in a RACF or USS group that has rwx permissions on the broker directory.

- **Scenario 2:** The following diagnostic message is written to the SDSF SYSLOG on z/OS when your broker fails to start:

```
ICH408I USER(MA10USR ) GROUP(WMQIBRKS  ) NAME(MA10USR
  CSFRNG CL(CSFSERV )
  INSUFFICIENT ACCESS AUTHORITY
  FROM ** (G)
  ACCESS INTENT(READ   )  ACCESS ALLOWED(NONE   )
```

- **Explanation:** During the broker start up process, Java accesses a secure random number generator. Consequently, the broker started task ID needs access to the CSFRNG resource in the CSFSERV class.
- **Solution:** Make sure that the started task ID that the broker runs under has access to the CSFRNG resource in the CSFSERV class.

Abend code 047 is issued with a diagnostic message:

Procedure

- **Scenario:** On z/OS, your broker generates abend code 047 when you try to start it, and the following diagnostic message is written to the SDSF SYSLOG:

```
IEA995I SYMPTOM DUMP OUTPUT 463
SYSTEM COMPLETION CODE=047
 TIME=10.53.47  SEQ=00419  CPU=0000  ASID=008E
 PSW AT TIME OF ERROR  078D0000   98D09E52  ILC 2  INTC 6B
   ACTIVE LOAD MODULE           ADDRESS=18D08828  OFFSET=0000162A
   NAME=SPECIALNAME
        61819987 968995A2 A3618499 89A58599 */argoinst/driver*
        F1F46D82 96858261 A4A29961 93979761 *14_boeb/usr/lpp/*
        A6949889 61828995 61828997 89948189 *wmqi/bin/bipimai*
        95                                  *n               *
   DATA AT PSW  18D09E4C - 58109948  0A6B5820  B8E95020
   GPR  0-3  00000000  0000003C  00000000  00000000
   GPR  4-7  18D10300  18D115F0  00000013  00000004
   GPR  8-11 18D111CF  18D101D0  18D0BBBE  18D0ABBF
   GPR 12-15 98D09BC0  18D101D0  98D09E22  00000000
 END OF SYMPTOM DUMP
```

- **Explanation:** System completion code 047 means that an unauthorized program issued a restricted Supervisor Call (SVC) instruction. The diagnostic message also indicates that the program in error was bipimain.
- **Solution:** When you install WebSphere Message Broker, issue the command **extattr +a bipimain** from the `bin` directory of the installation path to give APF authorization to program `bipimain`.

Error message BIP2228 is issued when you try to start a second broker on Linux or UNIX:
Procedure

- **Scenario:** Error message BIP2228, which mentions `semctl` in the syslog, is displayed when you try to start a second broker on Linux or UNIX.
- **Explanation:** This error typically indicates a permissions problem with a semaphore used by WebSphere Message Broker. A semaphore is created when the first broker starts after a reboot (or after an initial installation), and only members of the primary group of the semaphore creator can access this semaphore. This problem is a consequence of the UNIX System V IPC primitives that are used by WebSphere Message Broker.

  The BIP2228 message is logged by any broker that is started by a user who is not a member of the primary group of the semaphore creator. The broker tries to access the semaphore, but fails with a permissions-related error. The broker then terminates with the BIP2228 message.
- **Solution:** Avoid this problem by ensuring that all user IDs used to start WebSphere Message Broker have the same primary group. If this action is impractical, ensure that all user IDs are members of primary groups of all other user IDs. Contact your IBM Support Center for further assistance.

MQIsdp client connection is refused by the broker:
Procedure

- **Scenario:** When a new MQIsdp client tries to connect to the broker, its connection is refused.
- **Explanation:** MQIsdp Client ID fields must be unique. If a client sends a CONN packet that contains the same Client ID as a currently connected client, the behavior is undefined.
- **Solution:** Ensure that Client IDs are unique.

Error messages BIP2604 and BIP2624 are issued when you start a broker or a new message flow:
Procedure

- **Scenario:** The following messages are written to the USS syslog on z/OS when your execution group, or a newly deployed or started message flow, fails to start:

```
(PMQ1BRK.default)[8]BIP2624E: Unable to connect to queue manager 'PMQ5':
        MQCC=2; MQRC=2025; message flow node 'ComIbmMQConnectionManager'
(PMQ1BRK.default)[8]BIP2604E: Node failed to open WebsphereMQ queue
       'INPUT1' owned by queue manager 'PMQ5': completion code 2; reason code 2025
```

- **Explanation:** The WebSphere MQ return code of 2025 indicates that the maximum number of concurrent connections has been exceeded. On z/OS, a typical cause of this problem is the setting for IDBACK in the WebSphere MQ CSQ6SYSP macro.
- **Solution:** See the *z/OS System Setup Guide* section of the WebSphere MQ Version 7 Information Center online for information about setting the IDBACK variable.

When you start the broker through the DataFlowEngine, it cycles continually:
Procedure

- **Scenario:** When you start the broker through the DataFlowEngine, it continually cycles, starts and stops, and errors BIP2801 and BIP2110 appear in the log:

```
Unable to load implementation file '/opt/IBM/DistHu b/v2/lib/libdhbNBIO.so',
rc=The file access permissions do not allow the specified action.
Message broker internal program error.
```

- **Explanation:** The permissions on `/opt/IBM` have a value of 700, meaning that the broker service user ID cannot read the disthub files.
- **Solution:** Set the permissions on `/opt/IBM` to 755, which is rwxr-xr-x.

You have changed your logon password and cannot start your broker on Windows:
Procedure

- **Scenario:** You have changed your logon password on Windows, and when you start the broker, you see the following error message:

```
BIP8026E: It was not possible to start the component.
The component could not be started using the service user ID that was supplied when
the component was created. Ensure that the service user ID and password are still
valid. Ensure that the service user ID has permission to access all of the products
directories, specifically the 'bin' and 'log' directories.  Check for system
messages (on Windows this would be the application event log).
```

- **Solution:** Change properties of your broker by completing the following steps:
  1. Change your broker by using the command:

```
mqsichangebroker brokername -i ServiceUserID -a ServicePassword
```

     For example, to change your logon password to `user1pwd` for user ID `user1` on the broker called `WBRK_BROKER`, use the following command:

```
mqsichangebroker WBRK_BROKER -i user1 -a user1pwd
```

  2. Restart your broker.

Broker on Windows platform fails to start with message BIP2818 and an abend:
Procedure

- **Scenario:** You have more than one broker defined on a single Windows computer, and your brokers use different service user IDs. You can start brokers that share a common service user ID, but if you try to run a broker with a different service user ID at the same time you see BIP2818 message in Event

Viewer saying “Failed to create semaphore”. An abend file is also created (message BIP2110). The abend shows function CreateMutexW and err no 17.

- **Explanation:** On Windows computers, brokers that run on the same computer must all use the same service user ID.
- **Solution:** Revise your broker configuration so that all brokers on a single Windows computer run with the same service user ID.

The Java installation is at an incorrect level:
Procedure

- **Scenario:** You issue the command to start a broker, but the broker does not start and the system log includes message BIP8892, for example:

```
Verification failed. The installed Java level 1.3.2 does not
   meet the required Java level 1.5.
```

- **Explanation:** The command performs a check on the level of the Java product installed on the computer to ensure that the Java product is at the required level. The check has failed, therefore the broker is not started.
- **Solution:**
  - On distributed systems, you must use the JVM that is supplied with the broker; no other Java product is supported. Check that you have run **mqsiprofile**, or (on Windows only) that you issued the **mqsistart** command from the correct command console.

    If you ran the profile command, check the settings of the environment variables in **mqsiprofile**; MQSI_JREPATH, PATH, and the appropriate library path environment variables for your operating system. Change these settings to point to the integrated JVM, and ensure that no other Java installation is in the path.
  - On z/OS, install the correct level of Java that is reported in this message, update the broker profile BIPBPROF and submit BIPGEN to refresh the component ENVFILE.

Authorization errors are reported on z/OS:
Procedure

- **Scenario:** You issue the command to start a broker, but the component does not start and the JOBLOG includes message BIP8903, for example:

```
Verification failed. The APF Authorization check failed
   for file '/usr/lpp/mqsi/bin/mqsireadlog'.
WebSphere Message Broker requires that only bipimain is APF Authorized
   for successful operation. File '/usr/lpp/mqsi/bin/mqsireadlog'
   fails that requirement.
If the file indicated in the message is bipimain, use the USS command
   extattr to ensure that it is APF Authorized.
If the file indicated in the message is not bipimain, use the USS
   command extattr to ensure that it is not APF Authorized.
For more information, search the information center for "APF attributes".
```

- **Explanation:** When you start a broker, a series of checks is made to ensure that the broker environment is set up correctly. One or more of the checks for the broker identified in the message has failed, therefore the broker is not started. The errors shown here indicate that the authorizations for some files are incorrect and incompatible with broker operation.

  The broker requires that a single file, `bipimain`, is APF authorized, but the checks indicate that the authorization for file `mqsireadlog` is incorrect.
- **Solution:** The file `bipimain` must be APF authorized, and all other files must not have that authorization. Make the required changes to authorization for the files that are identified in the error messages and try the operation again.

Error message BIP8875 is issued when you start a broker:
Procedure

- **Scenario:** You issue the **mqsistart** command to start a broker, but the broker does not start and the system log shows message BIP8875, for example:

```
The component verification for MB7BROKER has finished,
but one or more checks failed.
```

- **Explanation:** The command performs a series of checks to ensure that the broker environment, WebSphere MQ queues, and Java are correct and accessible. One or more of the checks for the broker identified in the message has failed, therefore the broker is not started.
- **Solution:** Look in the system log, or in the Application log in the Event Viewer on Windows. Additional messages have been written before this message to indicate which checks have failed. All the checks are performed every time you issue **mqsistart**, therefore all errors are included in the log. Some messages are also returned when you run the command from the command line.

  Investigate the one or more errors that have been reported and check return codes and additional details. Look at the complete message content to check for typical causes of the error, and follow the advice given for the messages that you see in the log. View the complete message text in the Diagnostic Messages reference topics.

  For example, you might see one or more of the following messages:

  – BIP8875W: The component verification for '*component_name*' has finished, but one or more checks failed.
  – BIP8877W: The environment verification for component '*component_name*' has finished, but one or more checks failed.
  – BIP8883W: The WebSphere MQ verification for component '*component_name*' has finished, but one or more checks failed.
  – BIP8885E: Verification failed. Failed to connect to queue manager '*queue_manager_name*'. MQRC: *return_code* MQCC: *completion_code*
  – BIP8887E: Verification failed for queue '*queue_name*' on queue manager '*queue_manager_name*' while issuing '*operation*'. MQRC: *return_code* MQCC: *completion_code*
  – BIP8888E: Verification failed. Failed to disconnect from queue manager '*queue_manager_name*'. MQRC: *return_code* MQCC: *completion_code*
  – BIP8892E: Verification failed. The installed Java level '*level_installed*' does not meet the required Java level '*level_supported*'.
  – BIP8893E: Verification failed for environment variable '*variable_name*'. Unable to access file '*file_name*' with user ID '*user_ID*'. Additional information for IBM support: *data1 data2*.
  – BIP8895E: Verification failed. Environment variable '*variable_name*' is incorrect or missing.
  – BIP8896E: Verification failed. Unable to access the registry with user ID '*user_ID*'. Additional information for IBM support: *data1 data2*
  – BIP8897E: Verification failed. Environment variable '*variable_name*' does not match the component name '*component_name*'.
  – BIP8903E: Verification failed. The APF Authorization check failed for file '*file_name*'.
  – BIP8904E: Verification failed. Failed to start file '*file_name*' with return code '*return_code*' and errno '*error_number*'.

If you cannot resolve the problems that are reported, and you receive a message such as BIP8893 that includes additional information, include these items in the information that you provide when you contact IBM Service.

Warning messages BIP8288-BIP8297 are shown in the syslog when you start a broker:
Procedure

- **Scenario:** Warning messages BIP8288-BIP8297 are shown in the syslog when you start a broker on Linux and UNIX systems.
- **Explanation:** One or more problems were detected with the ODBC environment on Linux and UNIX systems.

  When you start a broker by using the **mqsistart** command, the **mqsicvp** command is run automatically to check that the broker environment is set up correctly. On Linux and UNIX systems, this command also verifies that the ODBC environment is configured correctly. If the ODBCINI environment variable is set, the **mqsicvp** command writes warning messages to the syslog in the following situation:
  – If the file to which the ODBCINI environment variable points does not exist, or the broker does not have access to read it or write to it

  If the environment variable ODBCUOINI is set, the **mqsicvp** command writes warning messages to the syslog in the following situations:
  – If the file that is referenced by the ODBCUOINI environment variable does not exist, or the broker does not have access to read or write to the file
  – If ODBCSYSINI is not set
  – If the directory that is referenced by the ODBCSYSINI environment variable does not contain a file called `odbcinst.ini`, or the broker does not have access to read or write to this file
  – If the IE02_PATH environment variable is not set
- **Solution:** Examine the warning messages in the syslog. To view more information, run the **mqsicvp** command from the command line.

Broker startup on z/OS is very slow:
Procedure

- **Scenario:** The broker startup on z/OS takes many minutes, with an extended time taken in loading the `imbjplug2` .lil file.
- **Explanation:** When WebSphere Message Broker is run in a shared file system sysplex environment, the LPAR that the broker started in does not necessarily own the file system mount points that the broker uses. In this scenario all file accesses have to pass through the coupling facility, which adversely affects performance. During startup the broker accesses and reads many files, and loads many Java class files. All these file operations are slowed, causing longer startup times.
- **Solution:** For optimal startup performance, mount the directories that the broker accesses in the LPAR in which the broker is started. In particular, mount the WebSphere Message Broker installation directories locally. Mounting file systems lists the directories that WebSphere Message Broker on z/OS needs on the file system at run time; mount them locally for optimal performance.

Error messages AMQ7626 and BIP8048 are displayed when you try to start a broker:
Procedure

- **Explanation:** You see these messages when using a broker on a queue manager that is configured for global coordination with Oracle.

- **Solution:** Start the queue manager manually with the **-si** flag before starting the broker.

:

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related tasks:

"Resolving problems that occur when you start resources" on page 3371
Use the advice given here to help you to resolve common problems that can occur when you start resources.

"Starting and stopping a broker" on page 921
Run the appropriate command to start or stop a broker.

"Checking APF attributes of `bipimain` on z/OS" on page 607
This task is part of the larger task of setting up your z/OS environment.

Related reference:

"**mqsistart** command" on page 3965
Use the **mqsistart** command to start the specified broker if all initial verification tests complete successfully.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

Diagnostic messages
Diagnostic messages are listed in this section in numeric order, grouped according to the component to which they relate.

Related information:

WebSphere MQ Version 7 Information Center online

Resolving problems when starting other resources

Advice for dealing with some common problems that can arise when you start resources other than a broker.

About this task

- "Resources terminate during startup" on page 3381
- "Resources hang at startup on Windows" on page 3381
- "Error message BIP8048 is issued when you start a component" on page 3382
- "You experience problems with the default configuration" on page 3382
- "A "Not Found" error is issued when you click a link to a specific sample" on page 3383
- "The Quick Tour is displayed as a blank window" on page 3383
- "Error message BIP0832 is issued on startup" on page 3383
- "Your execution groups restart repeatedly" on page 3383
- "You cannot tell whether startup is complete on z/OS" on page 3384
- "Abend code 0C1 is issued when you try to start the DataFlowEngine on z/OS" on page 3385
- "Error message BIP2604 with return code `MQRC_CONNTAG_IN_USE` is issued during the start of a message flow on z/OS" on page 3385

Resources terminate during startup:
Procedure

- Windows **Scenario:** The following error message is displayed when you start a broker on Windows:

  ```
  ServiceName - DLL initialization failure Initialization of the
  dynamic link library c:\windows\system32\user32.dll failed.
  The process is terminating abnormally.
  ```

- **Explanation:** This error is issued by Windows when it fails to start a service because it has insufficient storage.
- **Solution:** This error is an operating system problem. Information about how to recover from this problem is available in the Microsoft Developer Network (MSDN). You can access MSDN on the Web at `http://msdn.microsoft.com`.

Resources hang at startup on Windows:
Procedure

- Windows **Scenario:** You try to start the broker on Windows, but nothing happens in the Event log to show that a connection has started.
- **Explanation:** This problem is typically caused by processes having only one thread. To see if this is the cause, check the Windows Task Manager. If either of the processes `bipconfigmgr.exe` or `dataflowengine.exe` has started, check the number of threads owned by the process. If the process has only one thread, you are likely to encounter this problem.
- **Solution:**
  1. Shut down the broker using the **mqsistop** command and end the process from within the Task Manager.
  2. From the Windows **Start** button, click **Settings** > **Control Panel**
  3. Double-click **Administrative Tools**
  4. Double-click **Services** to open the Services window. From the list of available services, locate and right-click the broker resource you are attempting to start (the service name begins with `IBM WebSphere Message Broker component`). Click **Properties** from the menu.
  5. Make a note of the **This Account** setting. Contact the system administrator to obtain the password associated with **This Account**, because these settings are lost when you change values.
  6. Select **System Account** as the **Log On As** option, and select **Allow Service to Interact with Desktop.** These selections allow you to see any hidden dialog messages. Click **OK** to accept the changes.
  7. Restart the resource that is failing and report any subsequent error messages and dialog box messages to your IBM Service Representative.

8. When your IBM Service Representative has resolved this problem for you, make sure that you restore the **This Account**, **Password**, and **Confirm Password** entries to the values that you used when you created the broker.

Error message BIP8048 is issued when you start a component:
Procedure

- **Scenario:** Error message BIP8048 is issued when you start a component.
- **Explanation:** This message indicates that WebSphere MQ is not responding as expected when it tries to start the queue manager. This problem might be because the **strmqm** executable file is not present on UNIX or Linux systems, or the **amqmdain** executable file is not present on Windows, or permissions are incorrect.
- **Solution:** Check that your WebSphere MQ installation is fully functional:
  - On Windows, start the "IBM MQSeries" service.
  - On UNIX or Linux, issue the **strmqm** command to start the queue manager that is associated with this component.

  If the check fails, your WebSphere MQ installation is incomplete. This error occurs typically because you have previously installed WebSphere Application Server, which installs an embedded WebSphere MQ component that does not support WebSphere Message Broker.

  Uninstall WebSphere Application Server, then install the full WebSphere MQ product that is provided with WebSphere Message Broker.

You experience problems with the default configuration:
Procedure

- **Scenario:** You have run the Default Configuration wizard but there are problems with the default configuration.
- **Solution:** Use the Default Configuration wizard to remove the default configuration.

  If the Default Configuration wizard does not remove the default configuration completely, a wizard failure window opens with instructions on where to find the log. Take the following steps:

  1. Follow any advice that is given in the Default Configuration wizard log and try each step again.
  2. If retrying fails, restart the computer and run the Default Configuration wizard again.
  3. If the wizard still does not remove the default configuration, remove each component manually by performing the following actions in the order shown:
     a. Issue the following commands to find out which components are installed:
        - **mqsilist** (lists the brokers)
        - **dspmq** (lists WebSphere MQ components: the queue manager)
     b. In the WebSphere Message Broker Toolkit, delete the connection file `LocalDomain.broker` from the project LocalProject.
     c. In the WebSphere Message Broker Toolkit, delete the project LocalProject.
     d. Stop the default broker by issuing the command:
        ```
        mqsistop MB7BROKER
        ```
     e. Delete the default broker by issuing the command:
        ```
        mqsideletebroker MB7BROKER -w
        ```

The **-w** parameter deletes all files related to this broker from the associated work path.

f. If you need to remove the queue manager manually, issue the following commands:

```
endmqlsr -w -m MB7QMGR
endmqm -i MB7QMGR
dltmqm MB7QMGR
```

4. If you still experience problems after removing the default configuration manually, contact your IBM Support Center.

A "Not Found" error is issued when you click a link to a specific sample:
Procedure

- **Scenario:** You see a "Not Found" error when you click a link to a specific sample, indicating that a URL is invalid.
- **Explanation:** You can view sample applications only when you use the information center that is integrated with the WebSphere Message Broker Toolkit. If you are viewing a stand-alone or online information center, you cannot access these resources.
- **Solution:** If you want to access samples, ensure that you are viewing the information center from within the WebSphere Message Broker Toolkit.

The Quick Tour is displayed as a blank window:
Procedure

- **Scenario:** The Quick Tour is displayed as a blank window.
- **Explanation:** The Quick Tour requires Adobe Flash Player (was Macromedia Flash Player) Version 7, and is displayed as a blank window if the Adobe Flash Player plug-in is not installed or is at the wrong level on the default browser.
- **Solution:** Download the Adobe Flash Player by using one of the following methods:
  - Follow the link at the bottom of the Quick Tour page.
  - Select the specific Adobe Flash Player for your operating system and browser from the Adobe Flash Player download page.

Error message BIP0832 is issued on startup:
Procedure

- **Scenario:** The following error message is displayed on startup:

  ```
  BIP0832E: A class java.io.FileNotFoundException exception occurred which
  reported the following message: [filepath] (The process cannot access the
  file because it is being used by another process). Resolve the reason of
  error and try again.
  ```

- **Explanation:** An invalid WebSphere MQ Java Client trace output file has been specified on the Enqueue preferences screen.
- **Solution:**
  1. Open the Enqueue preferences screen by clicking **Windows** > **Preferences**, then clicking **Enqueue** on the left.
  2. In the **To file** field, specify a valid output file (one that is not read-only or already in use).

Your execution groups restart repeatedly:
Procedure

- **Scenario:** Your execution groups restart repeatedly. The system log might show an error, such as BIP2060.

- **Explanation:** The problem might be caused by:
  - The broker environment variables incorrectly defined
  - Incorrect Loadable Implementation Library directory permissions
  - Incorrect database permissions
  - Invalid user-written LILs
- **Solution:** Check:
  - Environment variables as described in "Environment variables after installation" on page 3642
  - File and directory permissions as described in "Checking the permission of the installation directory" on page 606
  - Group memberships as described in "Execution groups" on page 53 and "Creating an execution group using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 937

You cannot tell whether startup is complete on z/OS:
Procedure

- **Scenario:** You cannot tell whether startup has completed on your z/OS system.
- **Solution:** To determine if startup is complete:
  1. Check the messages in the system log. The following example shows a system log entry for a startup of a broker with one execution group:

```
S STU3053
$HASP100 STU3053  ON STCINRDR
IEF695I START STU3053  WITH JOBNAME STU3053  IS ASSIGNED TO USER STU3
 , GROUP STCGROUP
$HASP373 STU3053  STARTED
+(broker53) 0 BIP9141W: The component was started.
+(broker53) 0 BIP2001I: The WebSphere Business Integration Message Broker
service has started
 process ID 33554919.
+(broker53.default) 0 BIP2201I: Execution Group started: process '67109
442
 196'; thread '0'; additional information 'broker53', '76eb7f2d-e800-00
 00-0080-974c271866d2', 'default', 'true', 'Q4A3', 'false', 'ARG5D651',
  'ARGO53', '********', 'false', 'f9c27f2d-e800-0000-0080-974c271866d2'
 , '/local/argo/driver/drv3', '/local/argo/tgrp53/broker53'.
+(broker53.default) 0 BIP9137I: A work manager has been registered by R
443
 RMS registration services, work manager name is BIP.STU30532.006710919
 6.IBM.UA  .
```

  2. Display the address spaces. The following example shows the display of a broker with one execution group:

```
D OMVS,U=STU3
BPXO040I 18.49.59 DISPLAY OMVS 446
OMVS     000E ACTIVE          OMVS=(68,05)
USER     JOBNAME  ASID        PID       PPID STATE   START     CT_SECS
STU3     STU30531 0069   33554696   33554919 HR    18.49.15      2.217
  LATCHWAITPID=         0 CMD=bipbroker broker53
STU3     STU30532 03FD   67109196   67109222 HR    18.49.23     19.816
  LATCHWAITPID=         0 CMD=DataFlowEngine broker53 76eb7f2d-e800-00
STU3     STU3053  0036   33554768   83886483 HRI   18.49.08       .653
  LATCHWAITPID=         0 CMD=bipservice Q4A3BRK AUTO
STU3     STU30532 03FD   67109222   33554696 1W    18.49.23     19.816
  LATCHWAITPID=         0 CMD=bipimain DataFlowEngine broker53 76eb7f2
STU3     STU3053  0036   83886483          1 1WI   18.49.08       .653
  LATCHWAITPID=         0 CMD=/local/argo/driver/drv3/bin/bipimain bip
STU3     STU30531 0069   33554919   33554768 1W    18.49.15      2.217
  LATCHWAITPID=         0 CMD=bipimain bipbroker broker53
```

Results

The infrastructure main program bipimain is the first process in every address space. It starts bipservice, bipbroker, or DataFlowEngine as the second process in the same address space. For each execution group, an additional address space is started. In this example, only one execution group is available.

Abend code 0C1 is issued when you try to start the DataFlowEngine on z/OS: Procedure

- **Scenario:** The first two WebSphere Message Broker address spaces start successfully, but the third address space (the DataFlowEngine) fails to start. The result is an 0C1 abend.
- **Explanation:** The DataFlowEngine address space is generated by the admin agent. If the region size is too small, either because an insufficient region size was specified in the procedure, or because the region size was overridden by the z/OS IEFUSI exit, the DataFlowEngine address space might fail to start, and fails with an 0C1 abend.
- **Solution:**
  1. Use the IPCS command on the dump (move the dump from the file system to a traditional MVS data set if required):

     ```
     verbx vsmdata,'noglobal,jobname(vcp0brk2)'
     ```

     where *vcp0brk2* is the name of the failing job.
  2. Find the string `'VSM LOCAL DATA AREA AT ADDRESS '`. The field ELIM gives the available region size and must be greater than `0C800000`. If the field SMFEL is not `ffffffff`, the IEFUSI exit has changed the allowable region size. This value must also be greater than 0C80000.
  3. If you have an IEFUSI exit, check that the exit does not limit the broker address spaces. For example, a commonly used field is OUCBSUBN. This field can be STC or OMVS for the broker, and indicates how the address space was started.

Error message BIP2604 with return code `MQRC_CONNTAG_IN_USE` is issued during the start of a message flow on z/OS: Procedure

- **Scenario:** Error message BIP2604 is issued with return code `MQRC_CONNTAG_IN_USE` during the start of a message flow on z/OS:

  ```
  BIP2604E: Node failed to open WebSphere MQ queue [queue name]
  owned by queue manager [queue manager name]
  ```

  This message is output every 30 minutes.
- **Explanation:** On z/OS, WebSphere MQ supports serialized access to shared resources, such as shared queues, through the use of a connection tag (serialization token) when an application connects to a queue manager that participates in a queue sharing group.

  In this case, a message flow node fails to connect to the indicated WebSphere Message Broker queue manager that is associated with the input queue, because the serialization token that it passed is already in use within the queue sharing group.

  This message is for information only. It indicates that serialization is occurring when two or more message flow input nodes try to connect to a queue manager to get messages from a shared queue.

- **Solution:** Check whether another instance of the message flow, or a flow using the same serialization token, is already running. If so, no further action is needed. Otherwise contact your IBM Support Center.

After creating or changing a configurable service, you restart your broker but your message flow does not start, and message BIP2275 is issued in the system log or Windows Event Viewer:
Procedure

- **Scenario:** After creating or changing a configurable service, you restart your broker but your message flow does not start, and message BIP2275 is issued in the system log or Windows Event Viewer, indicating that an error occurred while loading the message flow from the persistent store.
- **Explanation:** When you change or create the configurable service, the connection properties are not fully validated at that point; the broker does not attempt to use them to make a connection. For inbound adapters, the connection is made only when the broker is restarted. Therefore, the properties that you set on the configurable service might be invalid.
- **Solution:** Look at the messages following the BIP2275 message to determine if the message flow failed to start because of invalid connection properties. For example, in SAP you would see message BIP3414 with a reason such as:

```
Connect to SAP gateway failed
Connect_PM GWHOST= invalidhost.test.co, GWSERV=sapgw00, ASHOST= invalidhost.test.co,
  SYSNR=00
LOCATION CPIC (TCP/IP) on local host
ERROR partner not reached (host invalidhost.test.co, service 3300)
TIME Fri Nov 28 15:27:32 2008
RELEASE 640
COMPONENT NI (network interface)
VERSION 37
RC -10
MODULE nixxi_r.cpp
LINE 8728
DETAIL NiPConnect2
SYSTEM CALL SiPeekPendConn
ERRNO 10061
ERRNO TE'
```

  followed by a BIP3450 message with an adapter error message such as:

```
Connect to SAP gateway failed
Connect_PM GWHOST= invalidhost.test.co, GWSERV=sapgw00, ASHOST= invalidhost.test.co,
  SYSNR=00
LOCATION CPIC (TCP/IP) on local host
ERROR partner not reached (host invalidhost.test.co, service 3300)
TIME Fri Nov 28 15:27:32 2008
RELEASE 640
COMPONENT NI (network interface)
VERSION 37
RC -10
MODULE nixxi_r.cpp
LINE 8728
DETAIL NiPConnect2
SYSTEM CALL SiPeekPendConn
ERRNO 10061
ERRNO TE
```

  This error was detected by the adapter. The following message describes the diagnostic information that is provided by the adapter:

```
Connect to SAP gateway failed
Connect_PM GWHOST= invalidhost.test.co, GWSERV=sapgw00, ASHOST= invalidhost.test.co,
  SYSNR=00
```

```
LOCATION CPIC (TCP/IP) on local host
ERROR partner not reached (host invalidhost.test.co, service 3300)
TIME Fri Nov 28 15:27:32 2008
RELEASE 640
COMPONENT NI (network interface)
VERSION 37
RC -10
MODULE nixxi_r.cpp
LINE 8728
DETAIL NiPConnect2
SYSTEM CALL SiPeekPendConn
ERRNO 10061
ERRNO TE
```

This message suggests that the **applicationServerHost** and **gatewayHost** properties are incorrect. When you have determined which properties are incorrect, use the **mqsichangeproperties** command to correct the properties, or use the **mqsideleteconfigurableservice** command to revert to the properties that were deployed in the adapter. Restart the broker.

A device allocation error is issued:
Procedure

- **Scenario:** A device allocation error is issued.
- **Explanation:** A likely cause of this problem is that you do not have the correct permissions set on the component file system for the started task ID.
- **Solution:** Check the system log; if the problem is caused by having incorrect permissions set for the started task ID, you often see an RACF authorization failure message, as shown in the following example.

```
ICH408I USER(TASKID1 ) GROUP(TSOUSER ) NAME(FRED (FRED) 959
 /argo/MA11BRK/ENVFILE
 CL(DIRSRCH ) FID(01D7C7E2E3F0F8000F16000000000003)
 INSUFFICIENT AUTHORITY TO LOOKUP
 ACCESS INTENT(--X)  ACCESS ALLOWED(OTHER ---)
IEE132I START COMMAND DEVICE ALLOCATION ERROR
IEA989I SLIP TRAP ID=X33E MATCHED.  JOBNAME=*UNAVAIL, ASID=00A8.
D J,BPXAS
IEE115I 11.13.04 2001.212 ACTIVITY 601
```

In this example, the started task ID does not have access to the file system component. The ICH408I message shows:

- The file that the task is trying to access
- The user ID that is trying to access the file
- The permissions that the ID is expecting to have (`INTENT` in the message)
- The permissions that the ID actually has (`ALLOWED` in the message)

You can use this information to correct the permissions, then reissue, in this example, the start broker request. This type of message is produced if the user who is issuing the command (which might be to start the broker, or to submit JCL to start one of the utility jobs) does not have the correct file system permissions for the file system component. Use the ICH408I information to rectify the problem.

Another possible reason for authorization failures is inconsistencies in the RACF definitions for a user ID in the MVS image and the OMVS segment. Also check with your system administrator that the RACF ID that is used on MVS has a corresponding OMVS image created.

Windows Vista and Windows Server 2008 fail to recognize WebSphere Message Broker digital signatures: "Unknown Publisher":

Procedure

- **Scenario:** User A installs WebSphere Message Broker, and can run all programs (`mqsi*.exe`, `bip*.exe`, including the command console launcher). User B is created and given appropriate privileges. When User B runs an executable file such as the command console launcher, a window opens and reports that the executable file is from an unidentified publisher.
- **Explanation:** The operating system has not installed the appropriate digital certificates for User B.
- **Solution:** User B must manually install the certificates:
  1. In Windows Explorer, navigate to the `bin` directory for the WebSphere Message Broker installation; for example, on 32-bit systems, `C:\Program Files\IBM\MQSI\7.0\bin`
  2. Right-click any .exe file to open the Properties window.
  3. Click the Digital Signatures tab.
  4. Select the appropriate certificate from the list, then click **Details**. The Digital Signature Details window is displayed.
  5. Click **View Certificate**. The Certificate window is displayed.
  6. Click **Install Certificate** and complete the steps in the wizard. (Click **Next**, **Next**, **Finish**, then click **OK**.)
  7. Close the Certificate window. You return to the Digital Signature Details.
  8. Select the countersignature from the list, then click the **Details** button. A new Digital Signature Details window is displayed. You can repeat the preceding steps to install other certificates.

The create command fails, and error message BIP8022 is issued:
Procedure

- **Scenario:** Error message BIP8022 is displayed when you use the **`mqsicreatebroker`** command on Windows, even if the supplied user name and password are correct.
- **Explanation:** The Microsoft component "Shared File and Printer Services" is required.
- **Solution:** Correct this error by installing the "Share File and Printer for Microsoft network" service on the Windows system.

Your execution group restarts repeatedly with a JVM Startup failure:
Procedure

- **Scenario:** When you start the DataFlowEngine it continually starts and stops, displaying errors BIP2116E and BIP7409S in the log:

  ```
  BIP2116E: Message broker internal error: diagnostic information 'Fatal Error; exception thrown bef
  BIP7409S: The broker was unable to create a JVM. The return code indicates that an unrecognized op
  ```

- **Explanation:** When you start an execution group, it creates a Java virtual machine (JVM) for executing Java user-defined nodes, and its creation failed due to an incorrect JVM option.
- **Solution:** Correct the JVM option by completing the following steps:
  1. Stop the broker.

     ```
     mqsistop brokerName
     ```

  2. Check the jvmSystemProperty value of the failing execution group.

     ```
     mqsireportproperties brokerName -e egName -o ComIbmJVMManager -n
     jvmSystemProperty -f
     ```

  3. If the jvmSystemProperty has an invalid option, correct or reset its value.

```
mqsichangeproperties brokerName -e egName -o ComIbmJVMManager -n
jvmSystemProperty -v "" -f
```

4. Start the broker.

```
mqsistart brokerName
```

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related tasks:

"Resolving problems that occur when you start resources" on page 3371
Use the advice given here to help you to resolve common problems that can occur when you start resources.

"Starting and stopping a broker" on page 921
Run the appropriate command to start or stop a broker.

"Checking APF attributes of `bipimain` on z/OS" on page 607
This task is part of the larger task of setting up your z/OS environment.

Related reference:

"**mqsistart** command" on page 3965
Use the **mqsistart** command to start the specified broker if all initial verification tests complete successfully.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

Resolving problems that occur when migrating or importing resources

Use the advice given here to help you to resolve common problems that can occur when you import or migrate resources.

About this task

Migration information is regularly updated on the WebSphere Message Broker support web page with the latest details available. Click **Troubleshoot**, then look for a document with a title like "Problems and solutions when migrating".

Procedure

- "Resolving problems when migrating or importing message flows and message sets" on page 3390
  - "Message flows that refer to a migrated user-defined node have connection errors" on page 3390
  - "After migration, message flows cannot locate a user-defined node" on page 3390
  - "A message flow fails with exception BIP5027" on page 3390

- "Resolving problems when migrating or importing other resources" on page 3391
  - "The **mqsimigratecomponents** command fails with database error BIP2322" on page 3391
  - "The **File** > **Import** menu provides only the option to import a compressed file inside an existing project" on page 3391

Related tasks:

Chapter 3, "Migrating and upgrading," on page 137
To migrate a broker domain to WebSphere Message Broker Version 7.0, plan your migration strategy, perform pre-migration tasks, migrate your domain components, and then complete post-migration tasks.

Related reference:

"**mqsistop** command" on page 3972
Use the **mqsistop** command to stop the specified component.

Resolving problems when migrating or importing message flows and message sets

Use the advice given here to help you to resolve common problems that can occur when you import or migrate message flows and message sets.

Message flows that refer to a migrated user-defined node have connection errors:
Procedure

- **Scenario:** After migration, all message flows that refer to a migrated user-defined node have errors indicating that connections cannot be made.
- **Explanation:** One possible cause is that the original user-defined node had space characters as part of one or more terminal names. The spaces are wrongly rendered as 'X20'.
- **Solution:** Edit the user-defined node `.msgnode` file, which is available in the same project as the flows that you have migrated. Correct any terminal names that are at fault. Ensure that the names are exactly as the broker node implementation expects.

After migration, message flows cannot locate a user-defined node:
Procedure

- **Scenario:** After migration, message flows cannot locate a user-defined node.
- **Explanation:** One possible cause is that the flows do not have the correct reference internally to a user-defined node.
- **Solution:** Select the **Locate subflow** menu for the node that cannot be located. Using the Browse dialog box, locate the user-defined node (which is in the same project as migrated flows). The message flow now links to the user-defined node correctly and the task list entry is removed when you save the flow.

A message flow fails with exception BIP5027:
Procedure

- **Scenario:** You have migrated a message flow from Version 6.0 to Version 7.0. When you deploy and run the message flow on a Version 7.0 broker, it generates message BIP5027 and throws an exception.
- **Explanation:** You have set the Validate property on one or more nodes in the message flow. In Version 6.0, the XMLNSC parser ignores the Validate property setting, but in Version 7.0, it honors the setting. Because you set this property to Content or Content And Value, the parser tries to validate the message against

the message set defined by the Message Set property on the node. The validation fails, and the message flow generates an exception.

- **Solution:** Set the Validate property of the node to None and redeploy the relevant BAR file.

  For more information about changes in behavior after migration, see "Reviewing technical changes in Version 7.0" on page 205.

Resolving problems when migrating or importing other resources

Use the advice given here to help you to resolve common problems that can arise when you import or migrate resources other than message flows.

About this task

- "The **mqsimigratecomponents** command fails with database error BIP2322"
- "The **File** > **Import** menu provides only the option to import a compressed file inside an existing project"
- "COBOL compiler errors when importing a copybook" on page 3392

The mqsimigratecomponents command fails with database error BIP2322:
About this task

Procedure

- **Scenario:** The **mqsimigratecomponents** command fails with database error BIP2322: `The 'CREATE TABLE' command is not allowed within a multi-statement transaction in the 'BROKER1' database.`
- **Explanation:** If you are using the **mqsimigratecomponents** command to migrate a broker that uses a Sybase database, you must modify the database to enable the Data Definition Language (DDL) that the command uses.
- **Solution:** Take the following steps:
  1. Log on to ISQL using a system administrator account.
  2. Run the following series of commands:

```
1> use master
2> go
1> sp_dboption "BROKER1","ddl in tran",TRUE
2> go
Database option 'ddl in tran' turned ON for database 'BROKER1'.
Run the CHECKPOINT command in the database that was changed.
(return status = 0)
1> use BROKER1
2> go
1> checkpoint
2> go
```

     where *BROKER1* is the name of the Sybase broker database.

The File > Import menu provides only the option to import a compressed file inside an existing project:
Procedure

- **Scenario:** You have a compressed file that contains message set projects and message flow projects. When you click **File** > **Import**, you have only the option to import the compressed file inside an existing project, but you want to re-create the message set projects and message flow projects.

- **Solution:** When you export and import files, do not export or import the root directory, which is created for you because of the project file. When you export your message flow and message set projects:
  1. Click **Create only selected directories**.
  2. Clear the project root folder.
  3. Select the files and subdirectories as required. The project root folder is selected, but is displayed as gray.

  Then, when you import the compressed file:
  1. Clear the root (/) folder.
  2. Select the files and subfolders as required. The project root folder is selected, but is displayed as gray.

COBOL compiler errors when importing a copybook:
Procedure

- **Scenario:** The report file that is generated by the import contains COBOL compiler errors. For example, you try to import the following copybook:

  ```
  01 AIRLINE-REQUEST.
  ....05 CUSTOMER.
  ........10 NAME.............PIC X(45).
  ....05 ADDRESS.
  ........10 STREET.............PIC X(30).
  ........10 CITY.................PIC X(25).
  ........10 STATE...............PIC X(20).
  ........10 ZIP-CODE.........PIC X(5).
  ....05 FLIGHT-NO.............PIC X(6).
  ....05 TRAN-DATE............PIC X(10).
  ....05 COST.....................PIC X(7).
  ....05 CC-NO..................PIC X(20).
  ....05 RESPONSE.
  ........10 STATUS.............PIC X(100).
  ........10 DETAILS............PIC X(100).
  ```

  The report file contains errors:

  ```
  Line No : 4 IGYDS1089-S "ADDRESS" was invalid. Scanning was resumed at the next area "A" item, lev
  Line No : 14 IGYDS1089-S "STATUS" was invalid. Scanning was resumed at the next area "A" item, lev
  ```

- **Explanation:** The errors are caused by the copybook containing field names that are COBOL reserved keywords.
- **Solution:** Change the name of the fields in question, so that they are not COBOL reserved keywords, and retry the import.

Related tasks:

"Resolving problems when migrating or importing message flows and message sets" on page 3390
Use the advice given here to help you to resolve common problems that can occur when you import or migrate message flows and message sets.

Related reference:

"**mqsistop** command" on page 3972
Use the **mqsistop** command to stop the specified component.

Resolving problems when stopping resources

Use the advice given here to help you to resolve problems when you stop resources.

About this task

Procedure

- "You cannot stop the broker"
- "You cannot stop the broker queue manager"
- "The execution group ends abnormally"

You cannot stop the broker

Procedure

- **Scenario:** You run the **mqsistop** command to stop the broker, but the system freezes, and does not stop any of the execution groups.
- **Explanation:** One possible cause is that a message flow is being debugged and it is currently stopped at a breakpoint. WebSphere Message Broker regards this as a message in flight situation, and refuses to stop the broker through the normal command.
- **Solution:** Click **Stop debugging** in the Broker Application Development perspective of the WebSphere Message Broker Toolkit. After that operation has completed, the broker stops.

  If you cannot stop the debug session, end all execution group processes that are associated with that broker to allow the broker to stop. Your messages are backed out. Click **Stop debugging** after the broker restarts.

You cannot stop the broker queue manager

Procedure

- **Scenario:** You are trying to use the WebSphere MQ **endmqm** command to stop a broker queue manager on a distributed system, but it does not stop.
- **Explanation:** In certain circumstances, attempting to stop a broker queue manager does not cause the queue manager to stop. This situation can occur if you have configured any message flows with multiple threads (you have set the message flow property Additional Instances to a number greater than zero).
- **Solution:** If you want to stop the broker's queue manager, stop the broker by running the **mqsistop** command and specifying the **-q** parameter. (The **-q** parameter is not available on z/OS.) This command runs the WebSphere MQ **endmqm** command on your behalf in a controlled fashion that shuts down the broker and the queue manager cleanly.

The execution group ends abnormally

About this task

Procedure

- **Scenario:** Your execution group processes end abnormally.
- **Explanation:** When execution group processes end abnormally, they are restarted automatically by the `bipbroker` process. If an execution group process fails, it is restarted three times during each five-minute interval. The first five-minute interval begins when the execution group is first started. RetryInterval defaults to 5

  Remove the execution group from the broker configuration, deploy the broker configuration, then later add the execution group, and redeploy the broker configuration. The row is re-created and RetryInterval is set to its default value of 5.
- **Solution:** To change the default value:

  1. Stop the broker.

2. Change the value of the RetryInterval in the database table.
3. Restart the broker.

Related tasks:

"Starting and stopping a broker" on page 921
Run the appropriate command to start or stop a broker.

"Stopping a WebSphere MQ queue manager when you stop a broker" on page 929
If you are preparing to stop a broker, you can stop the broker's WebSphere MQ queue manager at the same time.

Related reference:

"**mqsistart** command" on page 3965
Use the **mqsistart** command to start the specified broker if all initial verification tests complete successfully.

"**mqsistop** command" on page 3972
Use the **mqsistop** command to stop the specified component.

Resolving problems when deleting resources

Use the advice given here to help you to resolve problems when you delete resources.

About this task

You cannot delete a project from your workspace

Procedure

- **Scenario:** You cannot delete a project from your workspace. You get error messages indicating that the containing directory cannot be deleted, or the project file is missing.
- **Explanation:** If you attempt to delete a project, and the directory that contains the project is in use, or you have any files that are contained within the project that have been opened by programs other than the WebSphere Message Broker Toolkit, some of the resources in the project are not deleted, but others, including the project file, might be deleted.
- **Solution:** Before you delete a project, make sure that other applications do not have the files open, and that you do not have an open command prompt located in the directory. To recover from this problem, manually delete any remaining files and directories from your workspace directory, then click **Delete** from the project in the WebSphere Message Broker Toolkit.

Related tasks:

"Deleting a broker" on page 930
Delete a broker using the command line on the system where the broker component is installed.

"Deleting a message flow project" on page 1427
A message flow project is the container in which you create and maintain all the resources associated with one or more message flows. These resources are created as files, and are displayed in the project in the Broker Development view. If you do not want to retain a message flow project, you can delete it.

Related reference:

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

"**mqsideletebroker** command" on page 3863
Use the **mqsideletebroker** command to delete a named broker. The command also deletes the queues on the associated queue manager (created when the broker was created). You can also specify that the queue manager is to be deleted.

Resolving problems when developing message flows

Use the advice given here to help you to resolve common problems that can arise when developing message flows.

About this task

Resolving appearance problems when developing message flows

This topic contains advice for dealing with some common appearance problems that can arise when developing message flows:

The task list does not update when you make corrections to your files: Procedure

- **Scenario:** The task list does not update any modifications that you make to an ESQL or mapping file. You have made corrections to the files, and while there are no error flags in the file or file icon, the error remains as a task list item.
- **Solution:** To work around this problem, set the following environment variable:

```
JITC_COMPILEOPT=SKIP{org/eclipse/ui/views/tasklist/TaskListContentProvider}
{resourceChanged}
```

You rename a flow that contains errors, but the task list entries remain: Procedure

- **Scenario:** When you rename a message flow in the WebSphere Message Broker Toolkit for which there are error icons (red crosses) displayed on nodes and connections, those error icons are removed when changes are made. However, the task list entries remain.
- **Solution:** Refresh the Message Flow editor by closing and reopening it.

Terminals on a subflow get out of sync as changes are made:

Procedure

- **Scenario:** You have a message flow that contains subflow nodes. The name or number of terminals on the subflow gets out of sync when changes are made on the subflow itself. The same problem can happen with promoted properties.
- **Solution:** Refresh the Message Flow editor by closing and reopening it. Close the Message Flow editor that contains subflows while the subflows are being changed.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Resolving problems when developing message flows" on page 3395
Use the advice given here to help you to resolve common problems that can arise when developing message flows.

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

Related reference:

"Message flows" on page 4015
Use the reference information in this section to develop your message flows and related resources.

Resolving problems when you use CORBA nodes

Advice for dealing with common problems that can arise when you develop message flows that contain CORBA nodes.

Before you begin

WebSphere Message Broker does not currently support all CORBA operations and types. Ensure that you are passing in a valid IDL file that contains supported operations and types. For a full list of what is supported, see "CORBA support" on page 2149. To ensure that the IDL file is valid, run it through an IDL parser.

About this task

- "Error message BIP4891 is issued when you include a CORBARequest node in a message flow"
- "A CORBA IDL file drop error is issued when you are using an IDL file that contains includes" on page 3397
- "Error message BIP4910 is issued during deployment when you are using an IDL file that contains includes" on page 3397

Error message BIP4891 is issued when you include a CORBARequest node in a message flow:

Procedure

- **Scenario:** You have created a message flow that contains a CORBARequest node, but error message BIP4891 is issued, indicating that the node did not receive a valid body.
- **Explanation:** This error message indicates that the CORBARequest node is trying to call an operation but cannot find the required input parameters in the incoming tree. WebSphere Message Broker uses the DataObject parser to read and write message from CORBA applications. If you use an input node to pass XML into the CORBARequest node, ensure that it uses the DataObject domain.
- **Solution:** Ensure that the incoming message has the correct structure. If you are using an input node to pass XML into the CORBARequest node, set the `Message domain` property on the **Input Message Parsing** tab of the input node to `DataObject`.

A CORBA IDL file drop error is issued when you are using an IDL file that contains includes:
Procedure

- **Scenario:** You have dragged a CORBA IDL file onto the canvas but a CORBA IDL file drop error is issued.
- **Explanation:** If you have imported an IDL file that contains includes, you must drag the top-level IDL file onto the canvas so that the CORBARequest node has all the relevant information. Similarly, when setting properties on the CORBARequest node, if you have imported an IDL file that contains includes, you must select the top-level IDL file in the `IDL file` property.
- **Solution:** Drag the top-level IDL file onto the canvas, or set the `IDL file` property on the CORBARequest node to the top-level IDL file.

Error message BIP4910 is issued during deployment when you are using an IDL file that contains includes:
Procedure

- **Scenario:** You are deploying a message flow that contains a CORBARequest node, but error message BIP4910 is issued.
- **Explanation:** This error is issued when you have imported an IDL file that contains includes, but not all the included IDL files have been added to the BAR file. For example, you might have dragged only the message flow onto the execution group. If you have imported an IDL file that contains includes, you must ensure that all the included IDL files are added to the BAR file so that all the relevant information is available to the message flow.
- **Solution:** When you deploy a message flow that contains a CORBARequest node and an IDL file that contains includes, ensure that all included IDL files are added to the BAR file.

  If you are dragging a message flow that uses a multifile IDL file onto an execution group in the Broker Development perspective, included IDL files are not deployed. To deploy message flows that use multifile IDL files, you must create a BAR file.

Related concepts:

"Common Object Request Broker Architecture (CORBA)" on page 2145
The Common Object Request Broker Architecture (CORBA) is a standard defined by the Object Management Group (OMG) that enables software components written in multiple computer languages and running on multiple computers to work together.

"CORBA nodes" on page 2147
Use CORBA nodes to connect WebSphere Message Broker with CORBA Internet

Inter-Orb Protocol (IIOP) applications.

"CORBA support" on page 2149
The CORBA nodes in WebSphere Message Broker support a set of types and operations in imported IDL files.

Related tasks:

"Connecting to an external CORBA application" on page 2159
Connecting to an external CORBA application involves importing an IDL file, creating a message flow, building a message, and processing the response from the CORBARequest node.

"Importing an IDL file" on page 2952
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to create a message definition from an IDL file.

Related reference:

"CORBARequest node" on page 4349
Use the CORBARequest node to call an external CORBA application over Internet Inter-Orb Protocol (IIOP).

Resolving problems when you use Email nodes

Advice for dealing with common problems that can arise when you develop message flows that contain Email nodes.

About this task

- "A negative email size displays in the local environment"
- "A parsing error displays when you reparse an email attachment as XML" on page 3399
- "Removing unwanted null characters from an email" on page 3399

A negative email size displays in the local environment:
Procedure

- **Scenario:** The EmailInput node receives an email from an email server that supports Post Office Protocol 3 (POP3) but the size of the email, including any attachments, might display a negative value in the `Root.EmailInputHeader.Size` Multipurpose Internet Mail Extensions (MIME) logical tree.
- **Explanation:** The email server provider that supports POP3 uses the **`TOP`** command to fetch the headers for the email message and the **`LIST`** command to determine the size of the entire message. The server then subtracts the two values to determine the size of the message body. If the server reports the size of the entire message incorrectly, you might see a negative number in the local environment Size field.
- **Solution:** You can use a Compute node to calculate the size of the email message and the size of any attachments. The following example ESQL can be used to calculate the size of the email content and attachments for a multipart MIME document. In this example, the result is stored in the LocalEnvironment:

```
DECLARE CURSOR REFERENCE TO InputRoot.MIME.Parts;
      DECLARE I INTEGER 0;

      FOR SOURCE AS CURSOR.Part[] DO
         SET I = I + LENGTH( SOURCE.Data.BLOB.BLOB);
      END FOR;

      SET OutputLocalEnvironment.Variables.EmailSize = I;
```

  For more information about messages that belong to the MIME domain, see "Manipulating messages in the MIME domain" on page 2612.

A parsing error displays when you reparse an email attachment as XML:
Procedure

- **Scenario:** Your message flow retrieves emails from an email server by using an EmailInput node. The email contains an XML document attachment that you want to reparse. However, when you try to reparse the attachment you receive parsing errors from WebSphere Message Broker reporting that you have an invalid XML character.
- **Explanation:** Some email servers might insert carriage return (CR) and line feed (LF) characters at the end of an email. Typically you would want to keep these characters, but in this scenario you must remove them so that you can reparse your XML data.
- **Solution:** Use the following ESQL in a Compute node to remove the CR and LF characters:

```
DECLARE NEWEMAIL BLOB TRIM( TRAILING X'0d0a' FROM InputRoot.
MIME.Data.BLOB.BLOB );
```

Removing unwanted null characters from an email:
Procedure

- **Scenario:** Your email attachment contains unwanted null characters that you would like to remove.
- **Explanation:** Your email attachment might contain null characters that you would like to remove; for example, you intend to reparse the data in the attachment.
- **Solution:** Use the following ESQL in a Compute node to remove the null characters:

```
DECLARE NEWEMAIL BLOB TRIM( TRAILING X'00' FROM InputRoot.MIME.
Data.BLOB.BLOB)
```

Related concepts:

"Local environment tree structure" on page 1056
The local environment tree is a part of the logical message tree in which you can store information while the message flow processes the message.

Related tasks:

"Processing email messages" on page 1786
You can configure the EmailOutput node to deliver an email from a message flow to an email server that supports Simple Mail Transfer Protocol (SMTP). You can also configure the EmailInput node to retrieve an email from an email server that supports Post Office Protocol 3 (POP3) or Internet Message Access Protocol (IMAP).

"Sending emails" on page 1787
You can configure WebSphere Message Broker to send an email, with or without attachments, to a static or dynamic list of recipients.

"Receiving emails" on page 1799
You can configure the EmailInput node to receive an email, with or without an attachment, from an email server that supports Post Office Protocol 3 (POP3) or Internet Message Access Protocol (IMAP).

Related reference:

"EmailInput node" on page 4394
Use the EmailInput node to retrieve an email, with or without attachments, from an email server that supports Post Office Protocol 3 (POP3) or Internet Message Access Protocol (IMAP).

Resolving ESQL problems when developing message flows

This topic contains advice for dealing with some common ESQL problems that can arise when developing message flows:

A `Routine not defined` error message is issued in ESQL when you move a routine:
Procedure

- **Scenario:** A `Routine not defined` error message is displayed in ESQL when you move a routine from one schema to another.
- **Explanation:** If a routine that was referenced by code in one schema is moved to another schema, where it is still visible, a false error is generated stating that the routine cannot be resolved.
- **Solution:** Clean the project by clicking **Project** > **Clean**.

The product fails to respond when you paste ESQL statements from Adobe Reader:
Procedure

- **Scenario:** When you copy and paste certain ESQL statements from Adobe Reader into the ESQL editor, WebSphere Message Broker stops responding.
- **Explanation:** This problem occurs when you paste text directly from Adobe Reader into either the ESQL editor or the Java editor.
- **Solution:** To work around this problem, either enter the text manually, or copy and paste it to a text editor (such as Notepad), then perform another copy and paste action from there.

You do not know how message flows handle the code page of ESQL files:
Procedure

- **Scenario:** You do not know how message flows handle the code page of ESQL files.
- **Solution:** The code page of an ESQL file is the code page of the WebSphere Message Broker Toolkit on which the file is created. You must deploy a message flow using an ESQL file on a WebSphere Message Broker Toolkit with the same code page setting as the ESQL file. When multiple ESQL files are involved in a single compiled message flow (`.cmf`) file, all these ESQL files must be in the same code page.

  See "Editor preferences and localized settings" on page 6793 for more information.

You do not know the naming restrictions for ESQL procedures and functions:
Procedure

- **Scenario:** You do not know the restrictions for choosing names for ESQL modules or schema scope ESQL and mapping procedures and functions.
- **Solution:** Module and schema scope procedures and functions cannot have names starting with `IBM_WBIMB_` because `IBM_` is reserved for IBM use, and `IBM_WBIMB_` is reserved for WebSphere Message Broker.

Error message BIP5431 is issued and the broker fails:
Procedure

- **Scenario:** Error message BIP5431 is displayed and the broker fails.
- **Explanation:** When setting output message properties, you have specified an incorrect physical format name for the message format.

- **Solution:** The name that you specify for the physical layer must match the name that you have defined for it. The default physical layer names are `Binary1`, `XML1` and `Text1`.

You are unable to call Java from ESQL:
Procedure

- **Scenario:** Your Java class files are not being found.
- **Explanation:** When creating the class files, you have not placed them in the correct location within the system CLASSPATH.
- **Solution:** See the "CREATE PROCEDURE statement" on page 5103 for further information.

Error message BIP3203 is issued: Format expression is not a valid FORMAT expression for converting expression to type:
Procedure

- **Scenario:** Your format expression contains an unrecognized character for the conversion.
- **Explanation:** Your format expression for a numeric conversion was used to convert to or from a *DATE*, *TIME*, *TIMESTAMP*, *GMTTIME* or *GMTTIMESTAMP* variable. Another possible explanation is that your format expression for a DateTime conversion was used to convert to or from an *INTEGER*, *DECIMAL* or *FLOAT* variable.
- **Solution:** Replace the format expression with one from the applicable types. For more information about valid data types and expressions, see the "ESQL reference" on page 5019 topic.

Error message BIP3204 is issued: Input expression does not match FORMAT expression. Parsing failed to match:
Procedure

- **Scenario:** You have used an input string that does not match the format expression.
- **Explanation:** Your format expression contains data that does not match the current element of the format expression.
- **Solution:** Either rewrite the format expression to match the input data, or modify the input data to match the format expression. For more information about valid data types and expressions, see the "ESQL reference" on page 5019 topic.

The CAST function does not provide the expected DST offset for non-GMT time zones:
Procedure

- **Scenario:** You are using the CAST function to convert a string to a TIME variable, in a broker that is running in a time zone other than GMT. The daylight saving time (DST) offset is not correctly calculated.
- **Explanation:** If no time zone is associated with the time string passed to CAST, it is converted to GMT time. If no date is supplied, the current system date is assumed.
- **Solution:** Specify the correct time zone and date. See "Formatting and parsing dateTimes as strings" on page 5253 for more information.

Error message BIP3205 is issued: The use of a FORMAT expression is not allowed when converting:

Procedure

- **Scenario:** You have used a format expression when it is not applicable, for example when converting from decimal to integer.
- **Explanation:** The use of format expressions is limited to casting between datetime and string values or numeric and string values. Your format expression cannot be applied in this case.
- **Solution:** Either remove the FORMAT clause, or change the parameters. For more information about valid data types and expressions, see the "ESQL reference" on page 5019 topic.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Resolving problems when developing message flows" on page 3395
Use the advice given here to help you to resolve common problems that can arise when developing message flows.

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Using trace" on page 3533
You can use different types of trace to help you with problem determination and troubleshooting.

"Accessing the Properties tree" on page 2460
The Properties tree has its own correlation name, Properties, and you must use this in all ESQL statements that refer to or set the content of this tree.

"Creating destination lists" on page 1477
Create a list of destinations to indicate where a message is sent.

Related reference:

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"Message flows" on page 4015
Use the reference information in this section to develop your message flows and related resources.

Problems when developing message flows with file nodes

Use the advice given here to help you to resolve some common problems that can arise when you develop message flows that contain file nodes.

About this task

- "A file node flow stops processing files and error message BIP3331 or BIP3332 is issued" on page 3403

- "During processing of a large file, error message BIP2106 is issued or the broker stops because of insufficient memory"
- "Missing or duplicate messages after recovery from failure in a flow attached to a FileInput node" on page 3404
- "No file is created in the output directory after FileOutput node processing" on page 3405
- "Output file name overrides have not been applied" on page 3406

A file node flow stops processing files and error message BIP3331 or BIP3332 is issued:
Procedure

- **Scenario:** Files in the specified input directory are not being processed. Error message BIP3331 or BIP3332 is issued.
- **Explanation:** The error messages explain that the FileInput node encountered an exception and could not continue file processing. This problem can be caused when the FileInput node cannot move files from its input directory to the archive or backout directory because of file system permissions or another file in the target directory preventing the file to be transferred. In this situation, the node is unable to process input files without losing data so processing stops. Two messages are issued; the first is either BIP3331 or BIP3332 which specifies a second message which describes the cause of the problem in more detail.
- **Solution:** If the first error message issued is BIP3331, stop the flow and resolve the problem. The FileInput node is unable to complete successful processing of the file.
  1. Stop the flow.
  2. Find the error message referenced in the BIP3331. This second error message identifies the problem and the files and directories causing it.
  3. Ensure the broker has the required access to these files and directories.
  4. You might need to move, delete or rename files in the archive or transit directories.
  5. Check whether the input file causing the problem has been successfully processed (except for being moved to the archive or backout directory). If it has been successfully processed, remove it from the input directory.
  6. Restart the flow.

  If the first error message issued is BIP3332, you do not need to stop the flow because the FileInput node has detected the problem before starting file processing. Find the error message referenced in the BIP3332 message. This second error message identifies the problem and the files and directories causing it.

During processing of a large file, error message BIP2106 is issued or the broker stops because of insufficient memory:
Procedure

- **Scenario:** Large input files cause the broker to issue messages, or stop, because insufficient memory is available.
- **Explanation:** The FileInput node can process very large files. Subsequent processing in the flow attached to its Out terminal might require more memory than is available to the broker.
- **Solution:** In most cases, the FileInput node imposes a limit of 100 MB on the records propagated to the attached flow. If your application needs to access large amounts of data, you might need to increase its available memory and reduce the number of available instances. See "Resolving problems with performance" on page 3504

on page 3500 for more information. If your application needs to process messages larger than 100 MB, you can override the FileInput node record size limit by taking the following actions:

– Before starting the broker, set the environment variable MQSI_FILENODES_MAXIMUM_RECORD_LENGTH to the required limit as an integer number of bytes, for example:

```
SET MQSI_FILENODES_MAXIMUM_RECORD_LENGTH=268435456
```

– When the broker first initializes a FileInput node, it will use the environment variable value instead of the default value of 100 MB. Subsequent changes to the environment variable value will not affect the broker limit until the broker is restarted.

– If the `Record detection` property is set to `Whole File`, the limit applies to the file size. If the `Record detection` property is set to `Fixed Length` or `Delimited`, the limit applies to the record size. The FileOutput node is not affected by changes to this limit.

–

Note: Increasing the FileInput node record size limit might require additional broker resources, particularly memory. You must thoroughly test and evaluate your broker's performance when processing these files. The number of factors that are involved in handling large messages make it impossible to provide specific broker memory requirements.

You can also reduce the memory required to process the file's contents in the following ways:

– If you are processing a whole file as a single BLOB, split it into smaller messages by specifying on the **Records and Elements** tab of the FileInput node's properties:

  - A value of `Fixed Length` in the `Record detection` property
  - A large value in the `Length` property, for example 1000000.

– If you are writing the file's contents to a single output file, specify `Record is Unmodified Data` in the FileOutput node's `Record definition` property; this reassembles the records in an output file of the same size as the input file. Wire the FileInput node's End of Data terminal to the FileOutput node's Finish File terminal. Configure the flow to have no additional instances to ensure that the output records arrive in sequence.

– If you are processing large records using the techniques shown in the Large Messaging sample, ensure that you do not cause the execution group to access the whole record. Avoid specifying a value of `$Body` in the `Pattern` property of a Trace node.

– If you have specified a value of `Parsed Record Sequence` in the FileInput node's `Record definition` property, the broker does not limit the size of the record. If subsequent nodes in the message flow try to access an entire large record, the broker might not have sufficient memory to allow this and stop. Use the techniques in the Large Messaging sample to limit the memory required to handle very large records.

Missing or duplicate messages after recovery from failure in a flow attached to a FileInput node:

Procedure

- **Scenario:** After the failure of a message flow containing a FileInput node processing the input file as multiple records, a subsequent restart of the flow results in duplicate messages being processed. If the flow is not restarted, some input records are not processed.

- **Explanation:** If a record produces a message which causes the flow to fail and retry processing does not solve the problem, the node stops processing the file and moves it to the backout directory. Records subsequent to the failing message are not processed. The FileInput node is not transactional; it cannot roll back the file input records. Transactional resources in the attached flow can roll back the effects of the failing input record but not preceding records. Records before the failing record will have been processed but records subsequent to the failing record will not have been processed. If you restart the flow by moving the input file from the backout directory to the input directory, messages from records preceding the point of failure are duplicated.
- **Solution:** If the input messages have unique keys, modify your flow to ignore duplicate records. If the messages do not have unique keys but each input file has a unique name, you can modify your flow to form a unique key based on the file name and record number. Define a database table and add a Database node to your flow to record the key of each record that is processed. Add a DatabaseRoute node to filter input messages so that only records without keys already in the database are processed. See the Simplified Database Routing sample to understand how to use the DatabaseRoute node to filter messages.

  If you cannot generate unique keys for each record, split your flow into two separate flows. In the first flow, wire the FileInput node to an MQOutput node so that each input record is copied as a BLOB to a WebSphere MQ queue. Ensure there are adequate WebSphere MQ resources, queue size for example, so that the first flow does not fail. In the second flow, wire an MQInput node to the flow previously wired to your FileInput node. Configure the MQInput and other nodes to achieve the desired transactional behavior.

No file is created in the output directory after FileOutput node processing: Procedure

- **Scenario:** A file created by the FileOutput node does not appear in the output directory. The node is configured so that the `Record definition` property has a value of `Record is Unmodified Data`, `Record is Fixed Length Data`, or `Record is Delimited Data` and the flow runs one or more times.
- **Explanation:** The FileOutput node accumulates messages, record by record, in an incomplete version of the output file in the transit subdirectory of the output directory. It moves the file from the transit subdirectory to the output directory only when it receives a message on its Finish File terminal; at this point, the file is complete. If the node's input processing fails before a message is sent to the Finish file terminal, the file remains in the transit directory. The file might be completed by a subsequent flow if it uses the same file name and output directory; if this does not happen, the file is never moved to the output directory
- **Solution:** If you need to ensure that incomplete files are moved to the output directory if the input flow fails, wire the input node's Failure terminal to the FileOutput node's Finish File terminal, in addition to all other flows that are wired to this terminal.

  If you need all output files to be available for a downstream process at a particular time or after a particular event, wire a separate flow to the FileOutput node's Finish File terminal to send a message at that particular time or on that particular event. If duplicate messages which identify the same file are sent to the Finish File terminal, the FileOutput node ignores them.

  If your flows use the `Request directory property location`, `Request file name property location` (default `Directory` and `Name` in the `$LocalEnvironment/Destination/File` folder), or `$LocalEnvironment/Wildcard/WildcardMatch`, ensure that messages sent to the Finish File terminal contain the correct elements and values to identify the output file and directory.

Output file name overrides have not been applied:
Procedure

- **Scenario:** The message elements set in the flow to override the output file name or directory values specified in the FileOutput node's **Basic** properties have not been applied. The output file is created using the name and directory set in the FileOutput node's **Basic** properties.
- **Explanation:** One of the following might be the cause of this problem:
  - The message sent to the FileOutput node does not contain the expected changes.
  - The FileOutput node is configured to use different elements in the message from the ones set to the new values.
  - Not all messages contain the overriding values.
- **Solution:** Use the debugger or a Trace node inserted in front of the FileOutput node's In terminal to check that the expected overriding values appear in the correct message elements. If they do not, check that the `Compute mode` property has been set correctly in Compute nodes that are upstream in the flow; for example, if $LocalEnvironment/File/Name has not changed following a Compute node, check that the Compute node has its `Compute mode` property set to `LocalEnvironment and Message`.

  If the message elements are set correctly, check that the FileOutput node's `Request directory property location` and `Request file name property location` properties identify the correct elements in the message.

  If you have specified `Record is Unmodified DataRecord is Fixed Length Data`, or `Record is Delimited Data` in the FileOutput node's `Record definition` property, ensure that messages that go to the Finish File terminal have the same override values as those that go to the in terminal. Unless you do this, the Finish file terminal message and the In terminal messages will apply to different files.

Related concepts:

"How the broker processes files" on page 1814
The broker reads files with the FileInput, FTEInput, and FileRead nodes, and writes files with the FileOutput and FTEOutput nodes.

"How multiple file nodes share access to files in the same directory" on page 1818
WebSphere Message Broker controls access to files so that only one file node at a time can read or write to a file.

"Using local environment variables with file nodes" on page 1820
You can use fields in the local environment to dynamically alter the behavior of the FileInput, FileOutput, FTEInput, and FTEOutput nodes. You can also find what values the output nodes used to process the file.

Related tasks:

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Reading files" on page 1834
Use the FileInput, CDInput, FTEInput, and FileRead nodes to read files.

"Writing a file" on page 1852
Use the FileOutput, CDOutput, and FTEOutput nodes to write files.

Related reference:

"FileInput node" on page 4415
Use the FileInput node to process messages that are read from files.

"FileOutput node" on page 4430
Use the FileOutput node to write messages to files.

Resolving problems when you use HTTP and SOAP nodes

Use the advice given here to help you to resolve common problems that can arise when you develop Web Services message flows that contain HTTP and SOAP nodes.

About this task

Use the replies to the following questions to assist you in diagnosing problems with HTTP or SOAP nodes:

- "How do I tell which listener the HTTP nodes are using?"
- "How do I collect HTTPListener trace?" on page 3408

A HandshakeException is issued when you use an HTTPRequest node to make an HTTPS call:
Procedure

- **Scenario:** You are trying to make an HTTPS call to an external web service from your WebSphere Message Broker message flow by using an HTTPRequest node. You receive the following error:

```
javax.net.ssl.SSLHandshakeException:
com.ibm.jsse2.util.h: PKIX path building failed:

java.security.cert.CertPathBuilderException:
PKIXCertPathBuilderImpl could not build a valid CertPath.;

internal cause is:
java.security.cert.CertPathValidatorException:
The certificate issued by OU=Class 3 Public Primary Certification Authority,
O="VeriSign, Inc.", C=US is not trusted;

internal cause is:
java.security.cert.CertPathValidatorException: Certificate chaining error
```

- **Explanation:** The broker cannot build the entire certificate path. The keystore of this broker must contain all the certificates in this chain of certificate authorities (CA). For a broker to verify the digital signature on a signed certificate, the keystore must contain the public key of the certificate authority (CA) that issued that certificate. If this public key is itself issued on a signed certificate, the keystore must contain the public key of the CA that issued that certificate. This chain continues until the broker reaches a root certificate authority that issues a self-signed certificate.
- **Solution:** Verify that you added all the required certificates to your keystore. If any of the components in the certificate chain are missing from your keystore, re-create your keystore by using keytool with the `genkey` option, then reimport your application certificates.

How do I tell which listener the HTTP nodes are using?:
Procedure

- **Scenario:** HTTP nodes that you include in a message flow can use either the broker-wide listener or the embedded listener that is defined to the execution group to which the containing message flow is deployed. Both listeners can handle both HTTP and HTTPS messages by handling the different message types on different ports.
- **Solution:** Use the **mqsireportproperties** command to check the properties that define what listener is in use.
  1. Check whether the broker listener has been disabled:

```
mqsireportproperties MB7BROKER -b httplistener -o HTTPListener -n startListener
```

If this property is false, all HTTP nodes in all execution groups are using an embedded listener.

2. If the broker listener is active, you must check the specific execution group.

   For example, check whether the listener in EG_A is being used to process HTTP messages for HTTP nodes:

   ```
   mqsireportproperties TEST -e EG_A -o ExecutionGroup  -n httpNodesUseEmbeddedListener
   ```

   If this property is true, all HTTP nodes that you deploy to this execution group are using the embedded listener.

   Check all execution groups for which you want to know this information.

3. If both properties are false, all HTTP nodes that you deploy to all execution groups are using the embedded listener. The value for the execution group property is ignored.

Results

If you are experiencing problems with message flows that contain HTTP nodes, and you want to collect trace information to provide to your IBM Support Center, trace the execution group. If you are using the broker-wide listener for HTTP nodes, also trace the HTTPListener component.

How do I collect HTTPListener trace?:
About this task

To gather information about HTTP nodes and listeners, you must start trace, run your message flows, then retrieve and format the trace information.

Trace the execution group and the HTTPListener component:

Procedure

- Start trace for the execution group.

  For example:

  ```
  mqsichangetrace MB7BROKER -t -e default -l debug
  ```

- If you are using the broker-wide listener for one or more execution groups, start trace for the HTTPListener component in one of the following two ways:
  - Trace all broker components:
    1. Run the **mqsichangetrace** command to start trace with the following options:

       ```
       mqsichangetrace component -t -b -l debug
       ```

       where *component* is the broker name.

    2. Retrieve the HTTPListener trace log by using the **mqsireadlog** command with the HTTPListener qualifier for the **-b** parameter.

       For example:

       ```
       mqsireadlog brokerName -t -b httplistener -f -o listenertrace.xml
       ```

    3. Format the trace log by using the **mqsiformatlog** command to view its contents.
  - Trace only the HTTPListener component:
    1. Run the **mqsichangeproperties** command to start trace with the following options:

```
mqsichangeproperties brokerName -b httplistener -o HTTPListener
 -n traceLevel -v debug
```

2. Retrieve and format the HTTPListener trace log as shown in the previous example.

What to do next

Save the trace output so that you can send it to the IBM Support Center, if requested.

Turn trace off when you finish collecting information to avoid affecting the performance of the broker.

:

Related concepts:

"Processing HTTP messages" on page 1579
Hypertext Transfer Protocol (HTTP) is an Internet protocol that is used to transfer and display hypertext and XML documents on the Web.

"HTTP listeners" on page 1589
You can choose between broker-wide listeners and execution group (embedded) listeners to manage HTTP messages in your HTTP flows. Learn about the two types of listener, how ports are assigned to them, and how you can switch from one to the other for individual execution groups.

Related reference:

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsichangetrace** command" on page 3822
Use the **mqsichangetrace** command to set the tracing characteristics for a broker.

"HTTPInput node" on page 4474
Use the HTTPInput node to receive an HTTP message from an HTTP client for processing by a message flow.

"HTTPReply node" on page 4484
Use the HTTPReply node to return a response from the message flow to an HTTP client. This node generates the response to an HTTP client from which the input message was received by the HTTPInput node, and waits for confirmation that it has been sent.

Resolving implementation problems when developing message flows

Use the advice given here to help you to resolve some common problems that can arise when running message flows.

About this task

Messages are directed to the Failure terminal of an MQInput node: Procedure

- **Scenario:** Messages that are received at a message flow are directed immediately to the Failure terminal on the MQInput node (if it is connected), or are rolled back.
- **Explanation:** When a message is received by WebSphere MQ, an error is signalled if the following conditions are all true:
  - The MQInput node requests that the message content is converted (the Convert property is set to `yes` on the node).
  - The message consists only of an MQMD followed by the body of the message.
  - The message format, as specified in the MQMD, is set to MQFMT_NONE.

  This error causes the message to be directed to the Failure terminal.
- **Solution:** In general, you do not need to request WebSphere MQ to convert the message content, because the broker processes messages in all code pages and encodings that are supported by WebSphere MQ. Set the Convert property to `no` to ensure that messages flow from the MQInput node to successive nodes in the message flow.

Error message BIP2211 is issued on z/OS by the MQInput node: Procedure

- **Scenario:** The following error message is issued by the MQInput node, indicating an invalid attribute:

```
BIP2211: (Invalid configuration message containing attribute value [attribute value]
which is not valid for target attribute [target attribute name],
object [object name]; valid values are [valid values])
```

- **Explanation:** On z/OS, WebSphere MQ supports serialized access to shared resources, such as shared queues, through the use of a connection tag (serialization token) when an application connects to the queue manager that participates in a queue sharing group. In this case, an invalid attribute has been specified for the z/OS serialization token.
- **Solution:** Check that the value that is provided for the z/OS serialization token conforms to the rules as described in the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online.

Messages enter the message flow but do not exit:

Procedure

- **Scenario:** You have sent messages into your message flow, and they have been removed from the input queue, but nothing appears at the other end of the message flow.
- **Explanation:** Several situations might cause this error to occur. Consider the following scenarios to try to identify the situation that is causing your failure:
  1. Check your message flow in the WebSphere Message Broker Toolkit.

     You might have connected the MQInput node Failure terminal to a successive node instead of the Out terminal. The Out terminal is the *middle* terminal of the three. Messages directed to an unconnected Out terminal are discarded.
  2. If the Out terminal of the MQInput node is connected correctly to a successive node, check the broker's local error log for an indication that message processing has been ended because of problems. Additional messages give more detailed information.

     If the Failure terminal of the MQInput node has been connected (for example, to an MQOutput node), these messages do not appear.

     Connecting a node to a Failure terminal of another node indicates that you have designed the message flow to deal with all error processing. If you connect a Failure terminal to an MQOutput node, your message flow ignores all errors that occur.
  3. If the Out terminal of the MQInput node is connected correctly to a successive node, and the local error log does not contain error messages, turn user tracing on for the message flow:
     a. Open the WebSphere Message Broker Explorer.
     b. In the Navigator view, expand the Brokers folder.
     c. Right-click the message flow, and click **User TraceNormal**.

     This action produces a user trace entry from only the nodes that the message visits.

     On distributed systems, you can retrieve the trace entries by using the **mqsireadlog** command, format them by using the **mqsiformatlog** command, and view the formatted records to check the path of the message through the message flow.

     z/OS For z/OS, edit and submit the BIPRELG job in COMPONENTPDS to execute the **mqsireadlog** and **mqsiformatlog** commands to process traces.
  4. If the user trace shows that the message is not taking the expected path through the message flow, increase the user trace level to Debug by selecting the message flow, right-clicking it, and clicking **User Trace** > **Debug**.

     Send your message into the message flow again. Debug-level trace produces much more detail about why the message is taking a particular route, and you can then determine the reasons for the actions taken by the message flow.

     Do not forget to turn tracing off when you have solved the problem, because performance might be adversely affected.
  5. If the MQPUT command to the output queue that is defined on the MQOutput node is not successful (for example, the queue is full or **put** is disabled), the final destination of a message depends on:
     - Whether the Failure terminal of the MQOutput node is connected.
     - Whether the message is being processed transactionally (which in turn depends on the transaction mode setting of the MQInput node, the MQOutput node, and the input and output queues).

– Whether the message is persistent or nonpersistent. When transaction mode is set to the default value of Automatic, message transactionality is derived from the way that it was specified at the input node. All messages are treated as persistent if transaction mode=yes, and as nonpersistent if transaction mode=no.

In general, if a path is not defined for a failure (that is, neither the Catch terminal nor the Failure terminal of the MQInput node is connected):

– Non-transactional messages are discarded.
– Transactional messages are rolled back to the input queue to be tried again:
  - If the backout count of the message is less than the backout threshold (BOTHRESH) of the input queue, the message is tried again and sent to the Out terminal.
  - When the backout count equals or exceeds the backout threshold, one of the following might happen:
    • The message is placed on the backout queue, if one is specified (using the BOQNAME attribute of the input queue.)
    • The message is placed on the dead-letter queue, if there is no backout queue defined or if the MQPUT to the backout queue fails.
    • If the MQPUT to the dead-letter queue fails, or if there is no dead-letter queue defined, then the message flow loops continuously trying to put the message to the dead-letter queue.
– If a path is defined for the failure, then that path defines the destination of the message. If both the Catch terminal and the Failure terminal are connected, the message is propagated through the Catch terminal.

6. If your message flow uses transaction mode=yes on the MQInput node properties, and the messages are not appearing on an output queue, check the path of the message flow.
   – If the message flow has paths that are not failures (but that do not end in an output queue), either:
     - The message flow has not failed and the message is not backed out.
     - The message flow is put to an alternative destination (for example, the Catch terminal, the dead-letter queue, or the queue's backout queue).
   – Check that all possible paths reach a final output node and do not reach a dead end. For example, check that you have:
     - Connected the Unknown terminal of a Filter node to another node in the message flow.
     - Connected both the True and False terminals of a Filter node to another node in the message flow.

Your execution group is not reading messages from the input queues: Procedure

- **Scenario:** Your execution group has started, but is not reading messages from the specified input queues.
- **Explanation:** A started execution group might not read messages from the input queues of the message flows because previous errors might have left the queue manager in an inconsistent state.
- **Solution:** Complete the following steps:
  1. Stop the broker.
  2. Stop the WebSphere MQ listener.
  3. Stop the WebSphere MQ channel initiator.

4. Stop the WebSphere MQ queue manager.
5. Restart the WebSphere MQ queue manager.
6. Restart the WebSphere MQ channel initiator.
7. Restart the WebSphere MQ listener.
8. Restart the broker.

The execution group ends while processing messages:
About this task

Procedure

- **Scenario:** While processing a series of messages, the execution group (DataFlowEngine) process size grows steadily without levelling off. This situation might cause the DataFlowEngine process to end if it cannot allocate more memory, and restart. The error message BIP2106 might be logged to indicate the out of memory condition.

  In addition, if you are using DB2 on distributed systems, you might get the message:

  ```
  SQL0954C  Not enough storage is  available in the application heap to process
  the statement.
  ```

  z/OS On z/OS, an SQLSTATE of HY014 might be returned with an SQL code of -99999, indicating that the DataFlowEngine process has reached the DB2 z/OS process limit of 254 prepared SQL statement handles.
- **Explanation:** When a database call is made from within a message flow node, the flow constructs the appropriate SQL, which is sent using ODBC to the database manager. As part of this process, the SQL statement is prepared using the SQLPrepare function, and a statement handle is acquired so that the SQL statement can be executed.

  For performance reasons, after the statement is prepared, the statement and handle are saved in a cache to reduce the number of calls to the SQLPrepare function. If the statement is already in the cache, the statement handle is returned so that it can be re-executed with newly bound parameters.

  The statement string is used to perform the cache lookup. By using hardcoded SQL strings that differ slightly for each message, the statement is not found in the cache, and an SQLPrepare function is always performed (and a new ODBC cursor is opened). When using PASSTHRU statements, use parameter markers so that the same SQL prepared statement can be used for each message processed, with the parameters being bound at run time. This approach is more efficient in terms of database resources and, for statements that are executed repeatedly, it is faster.

  However, it is not always possible to use parameter markers, or you might want to dynamically build the SQL statement strings at run time. This situation potentially leads to many unique SQL statements being cached. The cache itself does not grow that large, because these statements themselves are generally not big, but many small memory allocations can lead to memory fragmentation.
- **Solution:** If you encounter these types of situations, disable the caching of prepared statements by setting the MQSI_EMPTY_DB_CACHE environment variable to an arbitrary value. When this environment variable has been created, the prepared statements for that message flow are emptied at the end of processing for each message. This action might cause a slight performance degradation because every SQL statement is prepared.

Your execution group hangs, or ends with a core dump:

Procedure

- **Scenario:** While processing a message, an execution group either hangs with high CPU usage, or ends with a core dump. The stack trace from the core dump or abend file is large, showing many calls on the stack. Messages written to the system log might indicate "out of memory" or "bad allocation" conditions. The characteristics of the message flow in this scenario often include a hard-wired loop around some of the nodes.
- **Explanation:** When a message flow thread executes, it requires storage to perform the instructions that are defined by the logic of its connected nodes. This storage comes from the execution group's heap and stack storage. The execution of a message flow is constrained by the stack size, the default value of which differs depending on the operating system.
- **Solution:** If a message flow that is larger than the stack size is required, you can increase the stack size limit and then restart the brokers that are running on the system so that they use the new value. For information on setting the stack size for your operating system, see "System resources for message flow development" on page 3267.

Your XSLTransform node is not working after deployment and errors are issued indicating that the style sheet could not be processed:
Procedure

- **Scenario:** Your XSLTransform node is not working after deploying resources, and errors are displayed indicating that the style sheet could not be processed.
- **Solution:**
  - If the broker cannot find the style sheet or XML files that are required, migrate the style sheets or XML files with relative path references.
  - If the contents of a style sheet or XML file are damaged and therefore no longer usable (for example, if a file system failure occurs during a deployment), redeploy the damaged style sheet or XML file.

Output messages are not sent to expected destinations:
Procedure

- **Scenario:** You have developed a message flow that creates a destination list in the LocalEnvironment tree. The list might contain queues for the MQOutput node, labels for a RouteToLabel node, or URLs for an HTTPRequest node. However, the messages are not reaching these destinations, and there are no error messages.
- **Solution:**
  - Check that you have set Compute mode to a value that includes the LocalEnvironment in the output message, for example All. The default setting of Compute mode is Message, and all changes that you make to LocalEnvironment are lost.
  - Check your ESQL statements. The content and structure of LocalEnvironment are not enforced, so the ESQL editor (and content assist) does not provide guidance for field references, and you might have specified one or more of these references incorrectly.

    Some example procedures to help you set up destination lists are provided in "Populating Destination in the local environment tree" on page 2467. You can use these procedures unchanged, or modify them for your own requirements.

You experience problems when sending a message to an HTTP node's URL:

Procedure

- **Scenario:** Sending a message to an HTTP node's URL causes a timeout, or the message is not sent to the correct message flow.
- **Explanation:** The following rules are true when URL matching is performed:
  - There is one-to-one matching of HTTP requests to HTTPInput nodes. For each HTTP request, only one message flow receives the message. This statement is true even if two message flows are listening on the same URL. Similarly, you cannot predict which MQInput node that is listening on a particular queue will receive a message.
  - Messages are sent to wildcard URLs only if no other URL is matched. Therefore a URL of /* receives all messages that do not match another URL.
  - Changing a URL in an HTTPInput node does not automatically remove the entry from the HTTP listener. For example, if a URL /A is used first, then changed to a URL of /B, the URL of /A is still used to listen on, even though there is no message flow to process the message. This incorrect URL does get removed after the broker has been stopped and restarted twice.
- **Solution:** To find out which URL the broker is currently listening on, look at the file `wsplugin6.conf` in the following location:
  - Linux UNIX On Linux and UNIX: `/var/mqsi/components/broker_name/config`
  - Windows On Windows, `%ALLUSERSPROFILE%\Application Data\IBM\MQSI\components/broker_name/config` where `%ALLUSERSPROFILE%` is the environment variable that defines the system working directory. The default directory depends on the operating system:
    - On Windows XP and Windows Server 2003: `C:\Documents and Settings\All Users\Application Data\IBM\MQSI\components/broker_name/config`
    - On Windows Vista and Windows Server 2008: `C:\ProgramData\IBM\MQSI\components/broker_name/config`

    The actual value might be different on your computer.

  If problems persist, empty `wsplugin6.conf`, restart the broker, and redeploy the message flows.

When using secure HTTP connections, you change a DNS host's destination but the broker is using a cached DNS host definition:
Procedure

- **Scenario:** You are using a broker with secure HTTP connections that use the Java virtual machine (JVM). You have changed a DNS destination, but the broker is using a cached DNS host definition, therefore you have to restart the broker to use the new definition.
- **Explanation:** By default, Java caches the host lookup from DNS, which is not appropriate if you want to look up the host name each time or if you want to cache it for a limited amount of time. This situation occurs only when you use SSL connections. (When using a secure HTTPS connection, the HTTPRequest node uses the SSL protocol, which issues Java calls, whereas a non-SSL protocol uses native calls.)

  To avoid this situation, you can empty the cache on the JVM by setting the **`networkaddress.cache.ttl`** property to zero. This property dictates the caching policy for successful name lookups from the name service. The value is specified as an integer to indicate the number of seconds for which to cache the successful lookup. The default value of this property is -1, which indicates that the

successful DNS lookup value is cached indefinitely in the JVM. If you set this property to 0 (zero), the successful DNS lookup is not cached.

- **Solution:** To pick up DNS entry changes without the need to stop and restart the broker and JVM, disable DNS caching. Edit file `$JAVA_HOME/jre/lib/security/java.security`, and set the value of the **`networkaddress.cache.ttl`** property to `0` (zero).

The TimeoutControl node issues error message BIP4606 or BIP4607 when the timeout request start time that it receives is in the past:
Procedure

- **Scenario:** When a TimeoutControl node receives a timeout request message that contains a start time in the past, it issues error message BIP4606 or BIP4607: `The Timeout Control Node '&2' received a timeout request that did not contain a valid timeout start date/time value.`
- **Explanation:** The start time in the message can be calculated by adding an interval to the current time. If a delay occurs between the node that calculates the start time and the TimeoutControl node, the start time in the message will have passed by the time it reaches the TimeoutControl node. If the start time is more than approximately five minutes in the past, a warning is issued and the TimeoutControl node rejects the timeout request. If the start time is less than five minutes in the past, the node processes the request as if it were immediate.
- **Solution:** Ensure that the start time in the timeout request message is a time in the future.

You are using a TimeoutControl node with a TimeoutNotification node, with multiple clients running concurrently, and messages appear to be being dropped:
Procedure

- **Scenario:** You are using a TimeoutControl node with a TimeoutNotification node, with multiple clients running concurrently, and messages appear to be being dropped. In the timeout request message, allowOverwrite is set to TRUE.
- **Explanation:** If multiple clients are running concurrently, and allowOverwrite is set to TRUE in the timeout request message, messages can overwrite each other.
- **Solution:** Ensure that different TimeoutNotification nodes that are deployed on the same broker do not share the same unique identifier.

Error message BIP5347 is issued on AIX when you run a message flow that uses a message set:
About this task

Procedure

- **Scenario:** Error message BIP5347 (`MtilmbParser2: RM has thrown an unknown exception`) is issued on AIX in either of these circumstances:
  - When you are deploying a message set
  - When you are running a message flow that uses a message set
- **Explanation:** BIP5347 is typically caused by a database exception, and it is issued when an execution group tries to load an MRM dictionary for use by a message flow. This process involves two steps:
  1. The execution group retrieves the dictionary and wire format descriptors from the broker data store.
  2. The execution group stores the dictionary in the memory that a message flow would use to process an MRM message.

BIP5347 is typically issued during step 1. This problem can appear to be intermittent; if you restart the execution group, the message is sometimes processed correctly.

BIP5347 might also be caused by the presence of a datetime value constraint in the message set, which causes the error each time the message set is deployed.

- **Solution:** To identify the cause of the error, capture a service level debug trace to confirm that the database exception is occurring.
  - If the error is caused by the presence of a datetime value constraint, a message similar to the following message appears in the service level debug trace (the exact message depends on the datetime constraint in the message set):

    ```
    Unable to parse datetime internally, 9, 2001-12-17T09:30:47.0Z,
    yyyy-MM-dd'T'HH:mm:ss.SZZZ
    ```

    This error occurs because the MRM element in question has a datetime value that is not compatible with the datetime format string, so the dictionary is rejected. To solve this problem, ensure that the datetime value is compatible with the datetime format string.

Error message BIP2130 is issued with code page value of -1 or -2: Procedure

- **Scenario:** The following error message is issued:

```
BIP2130: Error converting a character string to or from codepage [code page value]
```

where `[code page value]` is either -1 or -2. You have not, however, specified a code page of either -1 or -2 in your message tree. You have, however, used one of the WebSphere MQ constants MQCCSI_EMBEDDED or MQCCSI_INHERIT.

- **Explanation:** The WebSphere MQ constants MQCCSI_EMBEDDED and MQCCSI_INHERIT are resolved when the whole of the message tree is serialized to produce the WebSphere MQ bit stream. This happens when the message is put on the WebSphere MQ transport. Until that time, these values exist in the message tree as either -1 (for MQCCSI_EMBEDDED) or -2 (for MQCCSI_INHERIT). If one or more parts of the message tree are serialized independently, such as with a ResetContentDescriptor node or ESQL ASBITSTREAM function, this error occurs.
- **Solution:** You do not have to set MQCCSI_EMBEDDED or MQCCSI_INHERIT in the message tree's CodedCharSetId field. You can achieve the same result by explicitly setting the required CodedCharSetId to the previous header's CodedCharSetId value. For example, you would need to replace:

```
SET OutputRoot.MQRFH2.(MQRFH2.Field)CodedCharSetId = MQCCSI_INHERIT;
```

with

```
SET OutputRoot.MQRFH2.(MQRFH2.Field)CodedCharSetId = InputRoot.MQMD.CodedCharSetId;
```

where the MQMD folder is the header preceding the MQRFH2 header.

The execution group restarts before an MQGet node has retrieved all messages: Procedure

- **Scenario:** You have created a message flow that contains an MQGet node. Not all of the messages are retrieved from the queue because the execution group restarts before the node has retrieved all the messages. No abend files are generated.
- **Explanation:** In WebSphere Message Broker, processing that involves nested or recursive processing can cause extensive use of the stack. Message flow

processing occurs in a loop until the MQGet node has retrieved all the messages from the queue. Each time that processing returns to the MQGet node, the stack size increases.

- **Solution:** Use a PROPAGATE statement. The statement propagates each message through the message flow in a loop, but each time that processing returns to the PROPAGATE statement, the stack is cleared.

  Use an ESQL variable (for example, set Environment.Complete to true) in the environment tree to terminate the ESQL loop, stop the propagations, and wait for the next trigger message. If you need to store content from the messages, store it in the environment tree because other trees are deleted when message flow processing returns to the PROPAGATE statement. For more information about how to use this statement, see "PROPAGATE statement" on page 5150.

:

Related concepts:

"WebSphere Message Broker Toolkit" on page 31
The WebSphere Message Broker Toolkit is an integrated development environment and graphical user interface based on the Eclipse platform.

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Resolving problems when developing message flows" on page 3395
Use the advice given here to help you to resolve common problems that can arise when developing message flows.

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Using trace" on page 3533
You can use different types of trace to help you with problem determination and troubleshooting.

"Accessing the Properties tree" on page 2460
The Properties tree has its own correlation name, Properties, and you must use this in all ESQL statements that refer to or set the content of this tree.

"Creating destination lists" on page 1477
Create a list of destinations to indicate where a message is sent.

Related reference:

"Message flows" on page 4015
Use the reference information in this section to develop your message flows and related resources.

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

"MQOutput node" on page 4612
Use the MQOutput node to send messages to clients that connect to the broker using the WebSphere MQ Enterprise Transport and that use the MQI and AMI application programming interfaces.

Related information:

WebSphere MQ Version 7 Information Center online

Resolving problems when you use IMS nodes

Advice for dealing with common problems that can arise when you develop message flows that contain IMS nodes.

Before you begin

Before you start:

- Read about IMS in "IBM Information Management System (IMS)" on page 2129.
- Ensure that you have set up the broker runtime environment correctly, as described in "Preparing the environment for IMS nodes" on page 731.

About this task

If you experience problems when you use IMS nodes in message flows, follow the instructions for the following scenarios to diagnose and solve the problem.

- "How can I tell if my broker is connected to IMS Connect?"
- "How can I discover the correct settings for `Hostname`, `Portnumber`, and `DataStoreName`?" on page 3420
- "What should I do when my transaction times out?" on page 3421
- "How many physical connections should I expect in IMS Connect?" on page 3421

How can I tell if my broker is connected to IMS Connect?:
Procedure

- **Scenario:** You need to check if your broker is connected to IMS Connect.
- **Explanation:** You can use SDSF on z/OS to issue a command to see which ports have clients connected to them.
- **Solution:** Using SDSF, enter the following **`QUERY MEMBER`** command, where *IM0ACONN* is the name of your IMS Connect job:

  ```
  /F IM0ACONN,QRY MEMBER TYPE(IMSCON)
  ```

  The output is in the following format:

  ```
  /F IM0ACONN,QRY MEMBER TYPE(IMSCON)
  HWSC0001I   HWS ID=IM0ACONN RACF=Y  PSWDMC=N
  HWSC0001I      MAXSOC=50  TIMEOUT=0
  HWSC0001I      RRS=N   STATUS=REGISTERED
  HWSC0001I      VERSION=V10 IP-ADDRESS=009.017.252.024
  HWSC0001I      SUPER MEMBER NAME=
  HWSC0001I    ADAPTER=N
  HWSC0001I    DATASTORE=IM0A     STATUS=ACTIVE
  HWSC0001I      GROUP=IM0AGRNM MEMBER=IM0ACONN
  HWSC0001I      TARGET MEMBER=IM0A
  HWSC0001I      DEFAULT REROUTE NAME=HWS£DEF
  HWSC0001I      RACF APPL NAME=
  HWSC0001I      OTMA ACEE AGING VALUE=2147483647
  ```

```
HWSC0001I      OTMA ACK TIMEOUT VALUE=120
HWSC0001I      OTMA MAX INPUT MESSAGE=5000
HWSC0001I    NO ACTIVE IMSPLEX
HWSC0001I    PORT=1080     STATUS=ACTIVE
HWSC0001I      CLIENTID USERID   TRANCODE STATUS       SECOND  CLNTPORT  IP-ADDRESS
HWSC0001I      HWSEHYMO JDOE   IVTNO    RECV            21      1109  009.017.137.11
HWSC0001I      TOTAL CLIENTS=1  RECV=1 CONN=0 XMIT=0 OTHER=0
```

This example shows that one client is connected on TCP port 1109, and that the client is connecting from the IP address 9.17.137.11.

You can run netstat on that system to find out in which process that client is running.

Use the following IMS command to view the TPIPEs that are created for those connections:

```
/DISPLAY TMEMBER IMSConnect_Name TPIPE ALL
```

– For transactions that have a commit mode of 1, the TPIPE name is the port number that is used for that interaction (for example, 1080 in the previous example).
– For transactions that have a commit mode of 0, the TPIPE name is the same as the client ID (for example, HWSEHYMO in the previous example). The client ID is generated automatically in the IMSRequest node.

How can I discover the correct settings for `Hostname`, `Portnumber`, and `DataStoreName`?:

Procedure

- **Scenario:** Your broker fails to connect to IMS and you want to verify your settings.
- **Explanation:** You can use SDSF on z/OS to issue a command to see members of the XCF group to which your IMS control region belongs. One of these members should be IMS Connect. You can then run a command against IMS Connect to discover these properties.
- **Solution:** Use SDSF to enter the following command:

  ```
  /xx/display OTMA
  ```

  where *xx* is the reply ID for your IMS control region job.

  For example, if you see `....... *26 DFS996I *IMS READY*  IM0A` in `S.log`, run the command `/26/DISPLAY OTMA`.

  The output from that command shows the name of the IMS Connect that is in the same XCF group as this IMS control region:

```
DFS000I     GROUP/MEMBER      XCF-STATUS   USER-STATUS    SECURITY  TIBINPT SMEM   IM0A
DFS000I                         DRUEXIT  T/O                                      IM0A
DFS000I     IM0AGRNM                                                              IM0A
DFS000I     -IM0A             ACTIVE       SERVER         CHECK                    IM0A
DFS000I     -IM0A               N/A        0                                       IM0A
DFS000I     -IM0ACONN         ACTIVE       ACCEPT TRAFFIC CHECK       05000        IM0A
DFS000I     *08350/175112*   IM0A
```

  Use SDSF to enter the following **`QUERY MEMBER`** command, where *IM0ACONN* is the name of your IMS Connect job that was reported by the previous command:

```
/F IM0ACONN,QRY MEMBER TYPE(IMSCON)
```

  The output is in the following format:

```
/F IM0ACONN,QRY MEMBER TYPE(IMSCON)
HWSC0001I   HWS ID=IM0ACONN RACF=Y  PSWDMC=N
HWSC0001I      MAXSOC=50  TIMEOUT=0
HWSC0001I      RRS=N   STATUS=REGISTERED
```

```
HWSC0001I      VERSION=V10 IP-ADDRESS=009.017.252.024
HWSC0001I      SUPER MEMBER NAME=
HWSC0001I    ADAPTER=N
HWSC0001I    DATASTORE=IM0A     STATUS=ACTIVE
HWSC0001I      GROUP=IM0AGRNM MEMBER=IM0ACONN
HWSC0001I      TARGET MEMBER=IM0A
HWSC0001I      DEFAULT REROUTE NAME=HWS£DEF
HWSC0001I      RACF APPL NAME=
HWSC0001I      OTMA ACEE AGING VALUE=2147483647
HWSC0001I      OTMA ACK TIMEOUT VALUE=120
HWSC0001I      OTMA MAX INPUT MESSAGE=5000
HWSC0001I    NO ACTIVE IMSPLEX
HWSC0001I    PORT=1080     STATUS=ACTIVE
HWSC0001I      CLIENTID USERID   TRANCODE STATUS       SECOND  CLNTPORT  IP-ADDRESS
HWSC0001I      HWSEHYMO JDOE   IVTNO    RECV             21      1109  009.017.137.11
HWSC0001I      TOTAL CLIENTS=1  RECV=1 CONN=0 XMIT=0 OTHER=0
```

What should I do when my transaction times out?:
Procedure

- **Scenario:** The transaction times out and the message is sent to the Timeout terminal, or an exception is issued.
- **Explanation:** Your transaction is taking longer than the values that are set for the execution or socket timeouts, therefore the node stops waiting for a response, and issues an exception or sends the message to the Timeout terminal.

  If the transaction subsequently completes successfully, the result depends on the commit mode that is set on the IMSRequest node:

  – If the `Commit mode` property is set to `0: COMMIT_THEN_SEND`, the unit of work is committed and the response is discarded.
  – If the `Commit mode` property is set to `1: SEND_THEN_COMMIT`, the response is not sent and the unit of work is rolled back.
- **Solution:** Increase the execution or socket timeout values to give enough time for the transaction to complete.
  – Configure the execution timeout by using the `Timeout waiting for a transaction to be executed` property on the IMSRequest node.
  – Configure the socket timeout on the configurable service.

How many physical connections should I expect in IMS Connect?:
Procedure

- **Scenario:** You need to set the values for the number of connections that are required by clients that are connecting to IMS Connect.
- **Explanation:** The number of physical connections that can be opened in IMS Connect is limited. The limit depends on the MAXSOC and MAXFEILEPROC settings.
  – The MAXSOC setting in IMS Connect determines the number of sockets that can be opened in IMS, which is the number of ports on which IMS listens for connections, plus the number of physical connections.
  – MAXFILEPROC is a UNIX System Services (USS) setting, which must be greater than or equal to MAXSOC, otherwise IMS reaches this limit before it reaches its own MAXSOC limit.

  If the IMS Connect process is granted superuser authority in USS, it sets MAXFILEPROC automatically.

  If the MAXSOC value is reached, IMS Connect issues warning message HWSS0771W, and refuses new requests for connections from clients. This behavior continues until the number of open sockets is below the limit (for example, after some clients have disconnected).

If the MAXFILEPROC value is reached, USS issues information message BPXI040I.

- **Solution:** When you set values for MAXSOC and MAXFILEPROC, consider how many clients are likely to connect concurrently to IMS Connect, and how many connections those clients will require.

  WebSphere Message Broker acts as a client to IMS Connect, and opens connections to IMS Connect. Therefore, find out how many connections are required by WebSphere Message Broker by gathering the following information:

  – The number of message flows with IMS nodes that are deployed
  – For those message flows, the values of the `Additional instances` property

  The maximum number of connections required for each broker is determined by the number of threads that can be running concurrently in the IMS nodes. In the following example, three message flows with IMS nodes exist:
  – Message flow A has 0 additional instances, therefore one thread is running.
  – Message flow B has 3 additional instances, therefore four threads can be running concurrently.
  – Message flow C has 4 additional instances, therefore five threads can be running concurrently.

  In this example, the maximum number of connections required by the broker is 10 (1+4+5). If you have four other similar brokers, all connecting to the same instance of IMS Connect, which has five ports configured, you would set the maximum number of sockets (MAXSOC) to at least 55 (the maximum number of connections for five brokers plus the number of ports: 10x5+5).

Related concepts:

"IBM Information Management System (IMS)" on page 2129
IMS is a message-based transaction manager and hierarchical-database manager for z/OS. External applications can use transactions to interact with applications that run inside IMS.

"Execution and threading models in a message flow" on page 1279
The execution model is the system used to start message flows which process messages through a series of nodes.

Related tasks:

"Changing connection information for the IMSRequest node" on page 732
You can create a configurable service that the IMSRequest node or message flow refers to at run time for connection information, instead of defining the connection properties on the node or the message flow. The advantage being that you can change the host name, performance, and security values without needing to redeploy your message flow.

Related reference:

"IMSRequest node" on page 4504
Use the IMSRequest node to send a request to run a transaction on a local or remote IMS system, and wait for a response. IMS Connect must be configured and running on the IMS system.

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

"Configurable message flow properties" on page 4020
When you add a message flow to a broker archive (BAR) file in preparation for deploying it to a broker, you can set additional properties that influence its run time operation. These properties are available for review and update when you select the **Manage and Configure** tab for the broker archive file.

"Issuing commands to the z/OS console" on page 3980
You operate the broker using the z/OS START, STOP, and MODIFY commands.

Resolving mapping and message reference problems when developing message flows

Advice for dealing with some common mapping and message reference problems that can arise when developing message flows:

Resources that are referenced by the mapping file cannot be resolved:
Procedure

- **Scenario:** You have imported some message flows into the WebSphere Message Broker Toolkit that contain mappings. An error is issued, indicating that the resources that are referenced by the mapping file cannot be resolved.
- **Explanation:** Mappings can use resources that exist in other projects. For example, a mapping reference to a message set might exist in a different project. If the reference cannot be resolved, it probably means that the reference to the other project has been lost.
- **Solution:** To reference a project:
  1. Right-click the project with the error, and click **Properties**.
  2. Under Project References, a list of the projects in your workspace is displayed. Select the required projects to reference the resources in these projects.
  3. Click **OK**.

Errors are issued when you import table schemas into the Message Mapping editor:
Procedure

- **Scenario:** When you try to import and add table schemas in the Message Mapping editor, you encounter errors like:

  ```
  /flow2/schema1/SAMPLE.conxmi cannot be loaded.
  The following error was reported: schema1/SAMPLE.conxmi
  ```

- **Explanation:** This error usually means that you have the same database files under the same broker schema name in another project. The relative paths are the same, so the Message Mapping editor cannot resolve this ambiguity and does not know which table to add.
- **Solution:** There are two courses of action:
  - If the table file already exists in the workspace, and this is what you want to use for mapping, reuse the file by clicking the **Add database table schemas from workspace** option in the Add Database Table Schemas dialog box.
  - If you want a different copy of the tables, rename the broker schema.

Warnings or errors are issued for message references:
Procedure

- **Scenario:** Warnings or errors are issued for message references, yet you are certain that your references are correct.
- **Explanation:** This is never the case with messages that are using the XML parser. For these message references, direct validation is not performed because the references could be used for generic XML.

  There is an ESQL editor preference that allows you to choose to ignore message reference mismatches, or to have them be reported as a warning or an error. By default, this type of problem is reported as a warning, so that you can still deploy the message flow.

- **Solution:** To use the validation feature, ensure that you have set up a project reference from the project that contains the ESQL to the project that contains the message set.

  If you are using reference in a subroutine, take the following steps:

  1. Create a reference to the tree and the parser in the module's main procedure.
  2. Associate the reference to the correlation name, for example InputRoot or Root. Alternatively, create the OutputRoot.*parser* node, where *parser* is the name of the parser that you want to use.
  3. Pass the reference as a parameter to an ESQL subroutine that identifies the XSD type of the reference.

Results

This practice is beneficial because the passed reference supports content assistance and validation for ESQL. The message type content properties `open`, or `open defined` are not used in validation, and the assumption is that this property is `closed`.

A `$db:select out of scope` error is generated when you map from a database source:
Procedure

- **Scenario:** You have specified a database as the data source and when you save the map file, there is an error saying `$db:select out of scope`
- **Explanation:** A $db:select expression must be within the scope of the $db:select entry in the Map Script column of the Spreadsheet pane, meaning that it must be a descendant of the select statement. If a $db:select expression is out of scope the Message Mapping editor moves the $db:select entry to a position where the $db:select expression is in scope. The $db:select expression can remain out of scope if it is positioned above the $db:select entry in the Map Script column of the Spreadsheet pane.
- **Solution:** Delete the out of scope $db:select expression or move the $db:select entry in the Map Script column. You can drag the element out of the 'for' row, and then drag the $db:select entry in the Map Script column higher in the message, above the out of scope $db:select expression. Ensure that the out of scope $db:select expression is now a descendant of the $db:select entry. See “Mapping a target element from database tables” on page 2288 topic for more information about database selects.

A `$db:proc out of scope` error is generated when you map from a database stored procedure:
Procedure

- **Scenario:** You have specified a stored procedure as the source and when you save the map file, there is an error saying `$db:proc out of scope`
- **Explanation:** A $db:proc expression must be within the scope of the $db:proc entry in the Map Script column of the Spreadsheet pane, meaning that it must be a descendant of the stored procedure statement. If a $db:proc expression is out of scope the Message Mapping editor moves the $db:proc entry to a position where the $db:proc expression is in scope. The $db:proc expression can remain out of scope if it is positioned above the $db:proc entry in the Map Script column of the Spreadsheet pane.
- **Solution:** Delete the out of scope $db:proc expression or move the $db:proc entry in the Map Script column. You can drag the $db:proc entry in the Map Script column higher in the message, above the out of scope $db:proc expression. Ensure that the out of scope $db:proc expression is now a

descendant of the $db:proc entry. See “Mapping a target element from database stored procedures” on page 2290 for more information about database stored procedures.

A `$db:func out of scope` error is generated when you map from a database user-defined function:
Procedure

- **Scenario:** You have specified a user-defined function as the source and when you save the map file, there is an error saying `$db:func out of scope`
- **Explanation:** A $db:func expression must be within the scope of the $db:func entry in the Map Script column of the Spreadsheet pane, meaning that it must be a descendant of the user defined function statement. If a $db:func expression is out of scope the Message Mapping editor moves the $db:func entry to a position where the $db:func expression is in scope. The $db:func expression can remain out of scope if it is positioned above the $db:func entry in the Map Script column of the Spreadsheet pane.
- **Solution:** Delete the out of scope $db:func expression or move the $db:func entry in the Map Script column. You can drag the $db:func entry in the Map Script column higher in the message, above the out of scope $db:func expression. Ensure that the out of scope $db:func expression is now a descendant of the $db:func entry. See “Mapping a target element from database user-defined functions” on page 2292 for more information about database user-defined functions.

Target is not referencing a valid variable warning when you set the value of a target:
Procedure

- **Scenario:** You have set the value for a target to a variable, such as a WebSphere MQ constant, and when you save the map file the warning `The target "$target" is not referencing a valid variable` is generated.
- **Explanation:** The variable that you have referenced is not recognized. For example, you might have entered an expression of the form `$mq:` followed by a WebSphere MQ constant, but the constant is not recognized. This might be because the variable has been entered incorrectly or it is not supported. Alternatively, you might be referencing a new variable or constant that can be resolved only at run time. If this is the case you can ignore the warning.
- **Solution:** Try one of the following to solve the problem:
  - Check that the variable has been entered correctly.
  - If you are using WebSphere MQ constants, use **Edit** > **Content Assist** to select from the list of available WebSphere MQ constants.

There are missing or unexpected targets in a message map:
Procedure

- **Scenario:** In your message map, warning messages are displayed that indicate a target element is missing, or a target element is at an unexpected location. As a result the output message generated by the message map might be incorrect.
- **Explanation:** If target elements are missing from the Spreadsheet pane when you edit and save the message map, a warning is displayed that a target is missing. This situation can occur if you use the Insert Children wizard and do not select all the required elements, or if you create mappings using the drag-and-drop method and do not create mappings for all the required fields. If target elements in the Spreadsheet pane are in an unexpected order, a warning that the element is unexpected at that location is displayed. This situation can occur if you drag elements to new locations in the Spreadsheet pane.

- **Solution:** To solve the problem:
  - Use **Insert Children** on the parent element to add any missing target elements to the Spreadsheet pane.
  - Drag any target elements that are in an unexpected location to the correct location. Use the message tree in the Target pane as a guide to the expected structure of the output message.

Error message BIP6118 is issued: `The remaining bitstream is too small contain the indicated structure.`:
Procedure

- **Scenario:** You have used an unsupported message domain for a target message in your message map.
- **Explanation:** The message domain that is associated with a target message is determined by the **`Message Domain`** property of your message set. Mapping nodes generate a target message that matches the message domain of the message set. Using a message domain that is not supported by the message mapper can result in an output message with a structure that is not valid for the chosen parser.
- **Solution:** To solve the problem, change the target message domain for your message set.

Error message BIP4680 is issued: `Unsupported message domain encountered in mapping node.`:
Procedure

- **Scenario:** You have used an unsupported message domain for a target message in your message map, for example BLOB.
- **Explanation:** The message domain that is associated with a target message is determined by the **`Message Domain`** property of your message set. Mapping nodes generate a target message that matches the message domain of the message set. Using a message domain that is not supported by the message mapper can result in an output message with a structure that is not valid for the chosen parser.
- **Solution:** To solve the problem, change the target message domain for your message set.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Resolving problems when developing message flows" on page 3395
Use the advice given here to help you to resolve common problems that can arise when developing message flows.

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Using trace" on page 3533
You can use different types of trace to help you with problem determination and troubleshooting.

"Accessing the Properties tree" on page 2460
The Properties tree has its own correlation name, Properties, and you must use this in all ESQL statements that refer to or set the content of this tree.

"Creating destination lists" on page 1477
Create a list of destinations to indicate where a message is sent.

Related reference:

"Message flows" on page 4015
Use the reference information in this section to develop your message flows and related resources.

"Mapping node" on page 4571
Use the Mapping node to construct one or more new messages and populate them with various types of information.

Resolving trace problems when developing message flows

Follow this advice to deal with some common trace problems that can arise when you develop message flows.

About this task

- "You cannot determine which node is being referenced in your trace file"
- "You cannot see any alerts when you change user trace"
- "Data that the Trace node sends to the syslog on UNIX is truncated" on page 3428

You cannot determine which node is being referenced in your trace file:
Procedure

- **Scenario:** You have generated a service trace file for your message flow, to trace the path of a message. However, you cannot determine which node is being referenced in the trace file.
- **Explanation:** In the trace file you might see text such as:

  ```
  Video_Test#FCMComposite_1_1 ComIbmMQInputNode , Video_Test.VIDEO_XML_IN
  ```

  The elements in the text have the following meanings:

  Video_Test
  is the name of the message flow

  FCMComposite_1_1
  is the internal name for the node

  ComIbmMQInputNode
  is the type of node

  VIDEO_XML_IN
  is the node label, and is the name you see in your flow

  The number at the end of the internal name is incremented; for example, FCMComposite_1_4 would be the fourth node you added to your flow. In the example, this section of the trace is referring to the first node in the message flow.

You cannot see any alerts when you change user trace:
Procedure

- **Scenario:** You cannot see any alerts for an execution group or message flow in the Alerts viewer when you use the **mqsichangetrace** command to change the user trace setting.

- **Explanation:** No alert is generated when an execution group runs user trace at normal or debug level. Alerts are generated only for message flow trace. For message flow trace, the WebSphere Message Broker Toolkit is not notified of trace changes that are initiated by the **mqsichangetrace** command.
- **Solution:** To see the alert, refresh the message flow, or disconnect, then reconnect to the domain.

Data that the Trace node sends to the syslog on UNIX is truncated: Procedure

- **Scenario:** You are using a Trace node on UNIX and have set the `Destination` property to `Local Error Log`. You send a message to the Trace node that consists of multiple lines, but the message that appears in the syslog is truncated at the end of the first line.
- **Explanation:** On UNIX, syslog entries are restricted in length and messages that are sent to the syslog are truncated by the new line character.
- **Solution:** To record a large amount of data in a log on UNIX, set the `Destination` property on the Trace node to `File` or `User Trace` instead of `Local Error Log`.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Resolving problems when developing message flows" on page 3395
Use the advice given here to help you to resolve common problems that can arise when developing message flows.

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Using trace" on page 3533
You can use different types of trace to help you with problem determination and troubleshooting.

Related reference:

"Message flows" on page 4015
Use the reference information in this section to develop your message flows and related resources.

Resolving problems when developing message flows with WebSphere Adapters nodes

Advice for dealing with common problems that can arise when you develop message flows that contain WebSphere Adapters nodes.

About this task

WebSphere Adapters nodes

- "Error messages BIP3414 and BIP3450 are issued when you deploy a WebSphere Adapters input node" on page 3429

- "Error messages are issued when classes cannot be found, or when problems occur with Java initialization"
- "The WebSphere Adapters are not visible when you run ITLM" on page 3430
- "A message flow with an SAPRequest, SiebelRequest, or PeopleSoftRequest node has deployed successfully, but message BIP3540 is issued indicating that connection failed" on page 3430

SAP nodes

- "You have deployed an SAP inbound adapter but do not receive expected messages" on page 3431
- "You have imported an existing project into your workspace, but messages are issued when you try to build SAP message sets" on page 3431
- "An error is issued when you use the message set that is generated by the Adapter Connection wizard" on page 3431
- "When you run the SAP samples on Linux or UNIX, WebSphere Message Broker does not connect to the SAP gateway" on page 3431
- "SAP inbound messages (ALE and BAPI) appear to be missing" on page 3432
- "When two ALE inbound modules use the same RFC program ID with SAP JCo version 3.0.2, NullPointer exceptions are logged in the JCo trace and the Adapter does not receive IDocs" on page 3432

Siebel nodes

- "You are using a SiebelInput node with the delivery type set to unordered, and error message BIP3450 is issued with a NullPointer exception" on page 3432

JD Edwards nodes

- "Error message BIP3450 is issued when you include a JDEdwardsRequest node in a message flow and try to connect to a JD Edwards EnterpriseOne server" on page 3433

Error messages BIP3414 and BIP3450 are issued when you deploy a WebSphere Adapters input node:
Procedure

- **Scenario:** When you deploy a message flow that contains a SiebelInput node, error message BIP3414 is issued.
- **Explanation:** The error messages explain that the SiebelInput node could not register with the adapter component to receive events. This problem can be caused when the broker does not know where to find the client libraries for the Siebel Enterprise Information Service (EIS). You might also encounter this problem if you are using the WebSphere Adapter for Siebel on an unsupported operating system.
- **Solution:** Use the **mqsireportproperties** and **mqsichangeproperties** commands to configure the broker with the location of the Siebel client libraries, as described in "Preparing the environment for WebSphere Adapters nodes" on page 717.

Error messages are issued when classes cannot be found, or when problems occur with Java initialization:
Procedure

- **Scenario:** You are deploying WebSphere Adapters, and error messages are issued that indicate that classes cannot be found, or problems are occurring with Java initialization. BIP3521 and BIP3522 error messages might also be issued.

- **Explanation:** The SAP, Siebel, and PeopleSoft adapters need client libraries from the manufacturer of the Enterprise Information System (EIS). If these libraries are missing, not installed correctly, or at an incorrect level, errors are issued.
- **Solution:** To solve this problem, complete the following steps.
  1. On the broker that you are using to change the configurable service of the adapter, run the **mqsichangeproperties** command to identify the location of the Java and native libraries.
  2. Ensure that the libraries are installed correctly, are valid for your operating system, and have the correct permission so that the broker can access them.
  3. Ensure that your operating system is supported by WebSphere Message Broker and the EIS provider. For details about supported operating systems, visit the WebSphere Message Broker Requirements Web site.

The WebSphere Adapters are not visible when you run ITLM:
Procedure

- **Scenario:** The adapter is not visible when you run the IBM Tivoli License Manager (ITLM).
- **Explanation:** If you want to use ITLM with the WebSphere Adapters, you must activate the ITLM file for each adapter.
- **Solution:** Follow the instructions in "Activating IBM Tivoli License Manager for WebSphere Adapters" on page 2036.

A message flow with an SAPRequest, SiebelRequest, or PeopleSoftRequest node has deployed successfully, but message BIP3540 is issued indicating that connection failed:
Procedure

- **Scenario:** From an SAPRequest, SiebelRequest, or PeopleSoftRequest node, an exception is thrown to indicate that the node is unable to make a connection even though the message flow has deployed successfully. The exception contains message BIP3540 with inserted text that indicates that connection failed. For example, for SAP, the inserted text is:

  ```
  Exception in connecting to SAP:Connect to SAP gateway failed
  Connect_PM GWHOST= invalidhost.test.co, GWSERV=sapgw00, ASHOST= invalidhost.test.co,
    SYSNR=00
  LOCATION CPIC (TCP/IP) on local host ERROR partner not reached (host
    invalidhost.test.co, service 3300) TIME Mon Dec 01 16:43:52 2008 RELEASE 640
  COMPONENT NI (network interface) VERSION 37 RC -10 MODULE nixxi_r.cpp LINE 8719
  DETAIL NiPConnect2 SYSTEM CALL SiPeekPendConn ERRNO 10061 ERRNO TE
  ```

  For PeopleSoft, the inserted text is:

  ```
  001DOWNbea.jolt.ServiceException: Invalid Session
  ```

- **Explanation:** The connection details are not verified when the message flow is deployed. For request nodes, the connection is made at first use.
- **Solution:** If you have configured a configurable service for this adapter, review the connection properties on that configurable service and review the text in the BIP3540 message to determine if the connection properties are incorrect. If the properties are incorrect, use the **mqsichangeproperties** command to correct them or use the **mqsideleteconfigurableservice** command to revert to the properties that are set on the adapter. Reload the execution group or stop and restart the broker.

  If no configurable service exists for this adapter, review the connection properties on the adapter. If the properties are incorrect, correct them and redeploy the adapter. Alternatively, use the **mqsicreateconfigurableservice** command to create a new configurable service with the correct properties to override the properties that are set on the adapter.

You have deployed an SAP inbound adapter but do not receive expected messages:
Procedure

- **Scenario:** You have deployed an SAP inbound adapter but do not receive the IDoc messages that you expected to receive.
- **Explanation:** If you have not received IDoc messages from SAP, it is possible that deployment was unsuccessful or the SAP server has not started.
- **Solution:** Check user trace for message BIP3484 occurring at the time of deployment. The adapter component writes diagnostic information to this message, in an insert that begins "CWYAP...". If this message is issued, it explains the cause of the problem.

You have imported an existing project into your workspace, but messages are issued when you try to build SAP message sets:
Procedure

- **Scenario:** You have imported an existing project into your workspace, but when you try to build an SAP message set, you see the message set compile error message BIP0182.
- **Explanation:** This error occurs when you choose the option to "Import existing projects into your workspace" from the Import dialog box. By choosing this option when you import, a link is created from the workspace to the existing projects in an external location and a required file is not available to the workspace. To copy the entire project into your workspace, use the option to import the Project Interchange (PI) file.
- **Solution:** When you import an exiting SAP project into your workspace, click **File** > **Import**, expand the **Other** folder, and click **Project Interchange**. For more information, see “Importing and exporting resources in a Project Interchange file” on page 1452.

An error is issued when you use the message set that is generated by the Adapter Connection wizard:
Procedure

- **Scenario:** You run the Adapter Connection wizard and select an inbound SAP IDoc. You run the wizard again, but this time you select an outbound SAP IDoc. When you use the message set that is generated, the following error is issued:

```
'Selector exception caught from generateEISFunctionname', 'commonj.connector.runtime.SelectorException:
commonj.connector.runtime.SelectorException: For the IDoc type SapYwmspgi01, operation key=YWMSPGIWMS not
found using the application-specific information {Create={MsgType=, MsgCode=, MsgFunction=}} verify appropriate
combination of MsgType, MsgCode, MsgFunction is set in SapYwmspgi01, application-specific information.
```

- **Explanation:** If you run the Adapter Connection wizard for an inbound SAP IDoc, then you run the wizard again for an outbound SAP IDoc, the outbound IDoc definition replaces the inbound IDoc definition. Information that is stored in the inbound definition is used to map MsgType, MsgCode, and MsgFunction to a method binding. The outbound definition does not contain these mappings, so processing of the inbound IDoc fails.
- **Solution:** To avoid this error, ensure that inbound and outbound SAP IDocs have different names if they are stored in the same message set.

When you run the SAP samples on Linux or UNIX, WebSphere Message Broker does not connect to the SAP gateway:
Procedure

- **Scenario:** When you run the SAP samples on Linux or UNIX, the message flow deploys successfully, but a connection is not established between WebSphere Message Broker and the SAP system. You might see the following error message:

```
message: Connect to SAP gateway failed
Connect parameters: TPNAME=SAMPRFC GWHOST=sapdev10 GWSERV=sapgw00
ERROR service 'sapgw00' unknown
```

- **Explanation:** For the SAP Java Connector (SAP JCo) to communicate across the network, you need to configure the TCP/IP service. If you have installed a working SAP GUI on your workstation, the TCP/IP service is configured as part of the installation. If you have not installed an SAP GUI, you must configure the TCP/IP service manually so that the SAP samples run successfully.
- **Solution:** Locate the services file in `/etc/services` and edit it so that it includes the appropriate gateway service IDs and the port numbers for the TCP/IP service in the format `sapgw`*`SID portnumber`*`/tcp`, where *SID* is the SAP system ID. For example:

  ```
  sapgw00  3300/tcp
  sapgw01  3301/tcp
  sapgw02  3302/tcp
  ```

  and so on.

  For more information about TCP/IP configuration, see the Network Integration section of the SAP Service Marketplace.

SAP inbound messages (ALE and BAPI) appear to be missing:
Procedure

- **Scenario:** SAP inbound messages (ALE and BAPI) appear to be missing. You might find that every other message does not reach the broker but no errors are issued.
- **Explanation:** This problem is typically caused when two brokers share the same program ID (SAP RFC Destination). For example, a developer has deployed a message flow, but someone else has used the same broker archive (BAR) file without having changed the program ID.
- **Solution:** Ensure that no other brokers are running with the same program ID. Use SAP transaction SMGW to determine whether other brokers are connected to the SAP system.

When two ALE inbound modules use the same RFC program ID with SAP JCo version 3.0.2, NullPointer exceptions are logged in the JCo trace and the Adapter does not receive IDocs:
Procedure

- **Scenario:** When two ALE inbound modules use the same RFC program ID with SAP JCo version 3.0.2, NullPointer exceptions are logged in the JCo trace and the Adapter does not receive IDocs.

  The following example shows a typical exception in the JCo trace.

  ```
  JCoDispatcherWorkerThread [16:44:42:140]: [JCoApi] Dispatcher.getNextListener() returns dispatch n
  JCoDispatcherWorkerThread [16:44:42:140]: [JCoApi] caught Throwable in DispatcherWorker.run() whil
   at java.util.Hashtable.get(Hashtable.java:518)
   at com.sap.conn.jco.rt.DefaultServerManager$DispatcherWorker.run(DefaultServerManager.java:268)
   at java.lang.Thread.run(Thread.java:735)

  JCoDispatcherWorkerThread [16:44:42:140]: [JCoApi] Dispatcher.getNextListener() returns no calls
  ```

- **Explanation:** SAP JCo version 3.0.2 does not support the use of two ALE inbound modules with the same RFC program ID.
- **Solution:** To solve this problem, download a hotfix from SAP; the SAP ticket reference number is 584247/2009.

You are using a SiebelInput node with the delivery type set to unordered, and error message BIP3450 is issued with a NullPointer exception:

Procedure

- **Scenario:** You are using a SiebelInput node, you have set the delivery type to unordered in the Adapter Connection wizard, and the minimum number of connections is 1 or less. The following exception is shown in user trace: `RecoverableException BIP3450E: An adapter error occurred during the processing of a message. The adapter error message is java.lang.NullPointerException`.
- **Explanation:** When using unordered events, the minimum connections (`MinimumConnections`) and maximum connections (`MaximumConnections`) properties must be greater than 1 for event delivery to be successful.
- **Solution:** Set the `MinimumConnections` and `MaximumConnections` properties on the Adapter Connection wizard to values greater than 1. For example, set the minimum number of connections to 2 and the maximum number of connections to 4.

Error message BIP3450 is issued when you include a JDEdwardsRequest node in a message flow and try to connect to a JD Edwards EnterpriseOne server: Procedure

- **Scenario:** You have created a message flow that contains a JDEdwardsRequest node, but error message BIP3450 is issued, indicating that the node is unable to connect to the JD Edwards EnterpriseOne server.
- **Explanation:** This error message indicates that the JDEdwardsRequest node is trying to connect to the JD Edwards EnterpriseOne server but is unable to do so. This error can be caused by the following situations.
  - The JD Edwards EnterpriseOne server is not running.
  - The JD Edwards adapter has been configured with incorrect connection details; for example, the name of the JD Edwards EnterpriseOne Environment to which to connect.
  - The JDBC drivers that are required to connect to the JD Edwards EnterpriseOne server are missing from the class path. The following table lists the required JDBC driver files for each database.

Database	JDBC driver files	Implementation class
Oracle	tsnames.ora classes12.zip	oracle.jdbc.driver.OracleDriver
SQLServer	sqljdbc.jar	com.ibm.microsoft.sqlserver.jdbc.SQLServerDr
AS/400	jt400.jar	com.ibm.as400.access.AS400JDBCDriver
DB2 Type-2 (JDK 1.4/1.5)	db2java.zip	com.ibm.db2.jdbc.app.DB2Driver
DB2 Type-4 (JDK 1.4/1.5)	db2jcc.jar db2jcc_license_cu.jar	com.ibm.db2.jcc.DB2Driver
DB2 Type-4 (JDK 1.6)	db2jcc4.jar	com.ibm.db2.jcc.DB2Driver

- **Solution:** Ensure that the following conditions have been met.
  - The JD Edwards EnterpriseOne server is running.
  - The JD Edwards adapter has been configured with the correct connection details.
  - The drivers that are required to connect to the JD Edwards EnterpriseOne server are in the class path.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and

PeopleSoft.

Related tasks:

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

"Changing connection details for SAP adapters" on page 719
SAP nodes can get SAP connection details from either the adapter component or a configurable service. By using configurable services, you can change the connection details for adapters without the need to redeploy the adapters. To pick up new values when a configurable service is created or modified, you must reload the broker or execution group to which the adapter was deployed, by using the **mqsistop** and **mqsistart** commands, or the **mqsireload** command.

"Changing connection details for Siebel adapters" on page 720
Siebel nodes can get Siebel connection details from either the adapter component or a configurable service. By using configurable services, you can change the connection details for adapters without the need to redeploy the adapters. To pick up new values when a configurable service is created or modified, you must reload the broker or execution group to which the adapter was deployed, by using the **mqsistop** and **mqsistart** commands, or the **mqsireload** command.

Related reference:

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

WebSphere Message Broker Requirements

Resolving other problems when developing message flows

Use the advice given here to deal with problems that can arise when developing message flows, and that are not covered in the specific categories listed in "Resolving problems when developing message flows"

About this task

- "The values of your promoted properties are lost after editing" on page 3435
- "The Message Flow editor experiences problems when opening a message flow, and opens in error mode" on page 3435
- "A message flow has subflows with the same user-defined property set to different values, but only one value is set at run time" on page 3435
- "You want to move resources to a new broker schema that you have created but it is not visible in the Broker Development view" on page 3435

- "A selector exception is raised when you use WebSphere Adapters" on page 3436

The values of your promoted properties are lost after editing:
Procedure

- **Scenario:** You edited a message flow using the Message Flow editor, and the values of your promoted properties are lost.
- **Explanation:** The values of promoted properties for nodes with more than a single subflow definition (that is, two identically named subflows in the same project reference path) are lost if the flow is edited and saved.
- **Solution:** To avoid this problem, ensure that each subflow in your project has a different name.

The Message Flow editor experiences problems when opening a message flow, and opens in error mode:
Procedure

- **Scenario:** You attempt to open an existing message flow in the Message Flow editor and it opens in read-only error mode, displaying a list of parsing or loading errors. The message flow is not open and a message is displayed indicating that the message flow file is not valid.
- **Explanation:** The message flow file is unreadable or is corrupted, and the Message Flow editor cannot render the model graphically.
- **Solution:** Contact IBM Customer Support for assistance with the corrupted file.

A message flow has subflows with the same user-defined property set to different values, but only one value is set at run time:
Procedure

- **Scenario:** You have a message flow that contains identical subflows. Each subflow has the same user-defined property (UDP), but with different values. At run time, only one of the values is set.
- **Explanation:** A UDP has global scope and is not specific to a particular subflow. If you reuse a subflow in a message flow, and those subflows have identical UDPs, you cannot set the UDPs to different values.
- **Solution:** If you need to set a different value for each subflow, use a different UDP for each subflow.

You want to move resources to a new broker schema that you have created but it is not visible in the Broker Development view:
Procedure

- **Scenario:** You have created a new broker schema and you want to move a resource to it, but the schema is not visible in the Broker Development view.
- **Explanation:** If category mode is selected, you cannot see the broker schema in the Broker Development view.
- **Solution:** To move resources to a broker schema that you have created, take one of the following steps.
  - Click **Hide Categories** () on the Broker Development view toolbar. The new broker schema appears in the Broker Development view and you can drag resources onto it.
  - Right-click a resource, click **Move**, then select the schema that you have created. When you click **OK**, the resource is moved to the selected schema.

A selector exception is raised when you use WebSphere Adapters:
Procedure

- **Scenario:** You run the Adapter Connection wizard for an inbound IDOC, then you run the wizard again for an outbound IDOC. When the message set is generated, you see the following error message:

```
'Selector exception caught from generateEISFunctionname' ,
'commonj.connector.runtime.SelectorException:
commonj.connector.runtime.SelectorException: For the IDoc type
SapYwmspgi01, operation key=YWMSPGIWMS not found using the
application-specific information {Create={MsgType=, MsgCode=, MsgFunction=}}
verify apropriate combination of MsgType,MsgCode, MsgFunction is set in
SapYwmspgi01, application-specific information.
--
```

- **Explanation:** When you run the Adapter Connection wizard for an inbound IDOC, then you run the wizard again for an outbound IDOC, the outbound IDOC definition replaces the inbound IDOC definition. Information that is stored in the inbound definition is used to map MsgType, MsgCode, and MsgFunction to a method binding. The outbound definition does not contain these mappings, so processing of the inbound IDOC fails.
- **Solution:** To avoid this error, ensure that inbound and outbound IDOCs have different names if they are stored in the same message set.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Resolving problems when developing message flows" on page 3395
Use the advice given here to help you to resolve common problems that can arise when developing message flows.

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

Related reference:

"Message flows" on page 4015
Use the reference information in this section to develop your message flows and related resources.

Resolving problems when deploying message flows or message sets

Use the advice given here to help you to resolve common problems that can arise when you deploy message flows or message sets.

Initial checks

Procedure

1. To debug problems when deploying, check the following logs:
   - The administration log
   - The local error log (the Windows Event log or the syslog)
   - The WebSphere MQ logs

These logs might be on separate computers, and must be used with the WebSphere Message Broker Explorer and WebSphere Message Broker Toolkit output to ensure that the deployment was successful.

Use the **mqsilist** command to check that the deployment was successful, or look in the Windows Event or Administration Log view.

2. Use this checklist when you have deployment problems:
   - Make sure that the mode that your broker is working in is appropriate for your requirements. See "Operation modes" on page 48.
   - Make sure that the remote queue manager is running.
   - Make sure that channels are running.
   - Display the channel status to see if the number of system messages sent increases.
   - Check the channel from the remote end.
   - Check the queue manager name.
   - Determine whether the channel is a cluster channel.

Common problems

About this task

"Resolving problems that occur when preparing to deploy message flows" on page 3438

- "An error is issued when you add a message set to a broker archive file" on page 3438
- "You cannot drag a broker archive file to a broker" on page 3438
- "You cannot deploy a message flow that uses a user-defined message flow" on page 3439
- "The compiled message flow file (`.cmf`) has not been generated" on page 3439

"Resolving problems that occur during deployment of message flows" on page 3440

- "You receive a warning message about your broker mode" on page 3440
- "The message flow deploys on the test system, but not elsewhere" on page 3441
- "Error messages about your broker mode are issued when you create an execution group" on page 3442
- "Error messages about your broker mode are issued when you deploy" on page 3442
- "Error messages about your function level are issued when you deploy" on page 3443
- "Error messages are issued when you deploy to z/OS" on page 3444
- "Expected serialization of input is not occurring for a shared queue that serves multiple instances of a message flow on z/OS" on page 3444
- "You create a configurable service, then deploy a message flow and inbound adapter, but the deployment fails" on page 3445
- "Error messages are issued when you deploy" on page 3447
- "You get an authority check failure when you deploy to z/OS" on page 3446

"Resolving problems that occur after deployment of message flows" on page 3451

- "Your XSLTransform node does not work after deployment" on page 3451

- "You receive exception ('MQCC_FAILED') reason '2042' ('MQRC_OBJECT_IN_USE')" on page 3451

Related concepts:

"Packaging and deployment overview" on page 3210
Deployment is the process of transferring data to an execution group on a broker so that it can take effect in the broker. Message flows and associated resources are packaged in broker archive (BAR) files for deployment.

"Operation modes" on page 48
The operation mode that you use for your broker is determined by the license that you purchase.

Related tasks:

Chapter 11, "Packaging and deploying," on page 3209
Package resources that you create in the WebSphere Message Broker Toolkit, such as message flows, and deploy them to execution groups on brokers.

"Are the Linux and UNIX environment variables set correctly?" on page 3350
Use the **mqsiprofile** command to set a command environment.

Related reference:

"**mqsideploy** command" on page 3872
Use the **mqsideploy** command to make a deployment request to the broker.

"**mqsilist** command" on page 3882
Use the **mqsilist** command to list installed brokers and their associated resources.

"**mqsistart** command" on page 3965
Use the **mqsistart** command to start the specified broker if all initial verification tests complete successfully.

"**mqsistop** command" on page 3972
Use the **mqsistop** command to stop the specified component.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

Resolving problems that occur when preparing to deploy message flows

Use the advice given here to help you to resolve common problems that can occur during preparations to deploy message flows or message sets.

An error is issued when you add a message set to a broker archive file: Procedure

- **Scenario:** An error is issued when you add a message set to a broker archive (BAR) file.
- **Explanation:** After you create a BAR file and add a message set project to it, two files are created in the BAR file: `messageset.user.txt` and `messageset.service.txt`. The `user.txt` file contains user log information, such as warning message BIP0177W, which states that the dictionary that you have created is not compatible with earlier versions.
- **Solution:** Use the information in the `user.txt` file to diagnose the error. The `service.txt` file contains detailed information that is used by the broker, and can be used by the IBM Support Center to diagnose problems.

You cannot drag a broker archive file to a broker:

Procedure

- **Scenario:** You cannot drag a broker archive (BAR) file to a broker.
- **Explanation:** BAR files can be deployed only on an execution group.
- **Solution:** Select an execution group in the deploy dialog.

You cannot deploy a message flow that uses a user-defined message flow:
Procedure

- **Scenario:** You have created a message flow that contains an input node in a user-defined node project. However, you cannot deploy a message flow that uses this node.
- **Explanation:** Validation, compilation, and deployment do not recognize that a user-defined message flow contains an input node.
- **Solution:** To work around the problem, add a dummy input node to the flow that you intend to deploy.

The compiled message flow file (`.cmf`) has not been generated:
Procedure

- **Scenario:** The compiled message flow file (`.cmf`) has not been generated. Therefore, it is not added to the broker archive file, and cannot be deployed.
- **Explanation:** When you create files that define message flow resources in the WebSphere Message Broker Toolkit, the overall file path length of those files must not exceed 256 characters, because of a Windows file system limitation.

  If you have a message flow that includes resource files that have a path length that exceeds 256 characters, the message flow cannot be compiled when you try to add it to a BAR file, and therefore cannot be deployed.

  You might also be affected by this restriction when you use the Adapters Connection wizard; names discovered and returned by the EIS can be very long. The Adapters Connection wizard attempts to write these files to the message set on the local file system, but their path exceeds the operating system limit, and the files appear corrupted.
- **Solution:** To ensure that the path length does not exceed 256 characters, use names that are as short as possible for all resources; for example:
  - The installation path
  - Project names and broker schema names
  - ESQL and mapping file names
  - Inbound and outbound adapter files

Related concepts:

"Packaging and deployment overview" on page 3210
Deployment is the process of transferring data to an execution group on a broker so that it can take effect in the broker. Message flows and associated resources are packaged in broker archive (BAR) files for deployment.

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Resolving problems when deploying message flows or message sets" on page 3436
Use the advice given here to help you to resolve common problems that can arise when you deploy message flows or message sets.

Chapter 11, "Packaging and deploying," on page 3209
Package resources that you create in the WebSphere Message Broker Toolkit, such

as message flows, and deploy them to execution groups on brokers.

"Are the Linux and UNIX environment variables set correctly?" on page 3350
Use the **mqsiprofile** command to set a command environment.

Related reference:

"**mqsideploy** command" on page 3872
Use the **mqsideploy** command to make a deployment request to the broker.

Resolving problems that occur during deployment of message flows

Use the advice given here to help you to resolve problems that can arise during deployment of message flows or message sets.

Procedure

You receive a warning message about your broker mode:
About this task

Message BIP1806

- **Scenario:** Warning message BIP1806 is displayed.

  ```
  The broker 'Broker_Name' did not reveal its mode in time;
   any reported information might not be up-to-date.
  ```

- **Explanation:** Your broker timed out before the command completed. You can set the timeout value on the **-w** parameter of the **mqsimode** command, or accept the default value of 60 seconds; see "**mqsimode** command" on page 3899.
- **Solution:** Check that the broker is started: if it is not started, start it by using the **mqsistart** command. If it is started, increase the timeout value and run the command again.

Message BIP1808

- **Scenario:** Warning message BIP1808 is displayed.

  ```
  Broker 'Broker_Name' is not at the required software level
   to change the mode.
  ```

- **Explanation:** Your broker is running in a previous version of the product. You must migrate brokers that you created in a version earlier than V6.1.0.2 before you can change the operation mode.
- **Solution:** Upgrade your broker to at least V6.1.0.2, and run the command again.

Message BIP1821

- **Scenario:** Warning message BIP1821 is displayed.

```
WARNING: Broker 'Broker_Name' is in 'Mode_Name' mode but has 'x' execution groups,
 which exceeds the allowed maximum for this mode.
```

- **Explanation:** Your broker is running in a mode that restricts the number of execution groups that you can use; see "Restrictions that apply in each operation mode" on page 3657.
- **Solution:** Contact your IBM representative to upgrade your license, or remove the required number of execution groups; see "Deleting an execution group by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 947.

Message BIP1822

- **Scenario:** Warning message BIP1822 is displayed.

```
WARNING: Broker 'Broker_Name' is in 'Mode_Name' mode but has 'x' message flows
 deployed, which exceeds the allowed maximum for this mode.
```

- **Explanation:** Your broker is running in a mode that restricts the number of message flows that you can use; see "Restrictions that apply in each operation mode" on page 3657.
- **Solution:** Contact your IBM representative to upgrade your license, or delete the required number of message flows; see "Deleting a message flow" on page 1441.

Message BIP1823

- **Scenario:** Warning message BIP1823 is displayed.

```
WARNING: Broker 'Broker_Name' has a message flow called 'Message_Flow' in execution
 group 'Execution_Group', which contains one or more nodes that
 are not valid in this mode: Mode_Name
```

- **Explanation:** Your broker is running in a mode that restricts the types of node that you can use in a message flow; see "Restrictions that apply in each operation mode" on page 3657.
- **Solution:** Contact your IBM representative to upgrade your license, rework your message flow to use nodes that are valid in the current mode, or remove the message flows that contain unsupported nodes; see "Deleting a message flow" on page 1441.

Message BIP1824

- **Scenario:** Warning message BIP1824 is displayed.

```
WARNING: The trial period for broker 'Broker_Name' expired on 'Day_Month_Year'.
```

- **Explanation:** Your broker is running in the Trial Edition mode and your trial period of 90 days has expired; see "Restrictions that apply in each operation mode" on page 3657.
- **Solution:** Contact your IBM representative to upgrade your license.

The message flow deploys on the test system, but not elsewhere: Procedure

- **Scenario:** The message flow that you have developed deploys on the test system, but not elsewhere.

- **Solution:** Carry out the following checks:
  - Ensure that you have verified the installation on the target system by creating and starting a broker, and deploying a single execution group. These actions confirm that the broker is correctly defined.
  - Ensure that the broker archive (BAR) file's `broker.xml` file contains references to the correct resources for the new system.
  - Ensure that any referenced message sets are deployed.
  - If database resources or user-defined nodes are not accessible or authorized from the target system, the deploy fails. On distributed systems, ensure that you have defined either ODBC or JDBC connections to your databases so that they can be accessed from the broker. Also, set the broker environment to allow access to the databases. On Linux or UNIX systems, you might have to run a database profile.
  - Any user-defined extensions that you are using in your message flow might not load if they cannot be found, or are not linked correctly. Consult the documentation for your operating system for details of tools that can help you to check the binary files of your user-defined extension.

Error messages about your broker mode are issued when you create an execution group:
About this task

Message BIP1825

- **Scenario:** Error message BIP1825 is displayed.

```
You attempted to create an execution group 'Execution_Group' on broker 'Broker_Name',
 but the broker is running in 'Mode_Name' mode which
 limits the number of execution groups that can exist at any one time.
 The execution group cannot be created.
```

- **Explanation:** The execution group cannot be created because the maximum number of execution groups for the mode of the target broker has been reached, and creating the execution group causes this limit to be exceeded; see "Restrictions that apply in each operation mode" on page 3657. The execution group was not created.
- **Solution:** Reuse an existing execution group, or delete an existing execution group and try the command again; see "Deleting an execution group by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 947. Alternatively, contact your IBM representative to upgrade your license.

Error messages about your broker mode are issued when you deploy:
About this task

Message BIP1826

- **Scenario:** Error message BIP1826 is displayed.

```
You attempted to deploy a broker archive (BAR) file to execution group
 'Execution_Group' on broker 'Broker_Name',
 but the broker is running in 'Mode_Name' mode which limits the
 number of message flows that can exist at any one time.
 The BAR file cannot be deployed.
```

- **Explanation:** The BAR file cannot be deployed because it causes the broker to run more message flows than are valid for the current mode of operation of the target broker; see "Restrictions that apply in each operation mode" on page 3657. The BAR file was not deployed.

- **Solution:** Delete message flows from the broker and try the command again; see "Deleting a message flow" on page 1441. Alternatively, contact your IBM representative to upgrade your license.

Message BIP1827

- **Scenario:** Error message BIP1827 is displayed.

```
You attempted to deploy a broker archive (BAR) file to execution group
 'Execution_Group' on broker 'Broker_Name',
 but the broker is running in 'Mode_Name' mode which has a restriction
 on the types of node that can be deployed.
 The BAR file cannot be deployed.
 The set of node types found in the BAR file that are not
 valid are: Node_Type.
```

- **Explanation:** The BAR file cannot be deployed because it contains nodes that are not valid for the current mode of the target broker; see "Restrictions that apply in each operation mode" on page 3657. The BAR file was not deployed.
- **Solution:** Rework your message flow to use nodes that are valid in the current mode, or remove the message flows that contain unsupported nodes; see "Deleting a message flow" on page 1441. Alternatively, contact your IBM representative to upgrade your license.

Message BIP1828

- **Scenario:** Error message BIP1828 is displayed.

```
You attempted to deploy a broker archive (BAR) file to execution group
 'Execution_Group' on broker 'Broker_Name',
 but the trial period for the broker has expired.
 The BAR file cannot be deployed.
```

- **Explanation:** The target broker is running in a trial mode that has expired; see "Restrictions that apply in each operation mode" on page 3657. The BAR file was not deployed.
- **Solution:** Contact your IBM representative to upgrade your license. If you have already purchased a valid license for the target broker, change the broker to the correct mode by using the **mqsimode** command; see "**mqsimode** command" on page 3899.

Message BIP1829

- **Scenario:** Error message BIP1829 is displayed.

```
You attempted to deploy a broker archive (BAR) file to execution group
 'Execution_Group' on broker 'Broker_Name',
 but the broker is running in 'Mode_Name' mode which
 limits the number of execution groups that can exist at any one time.
 The BAR file cannot be deployed.
```

- **Explanation:** The BAR file cannot be deployed because the maximum number of execution groups for the mode of the target broker has been reached; see "Restrictions that apply in each operation mode" on page 3657. The BAR file was not deployed.
- **Solution:** Delete an existing execution group and try the command again; see "Deleting an execution group by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 947. Alternatively, contact your IBM representative to upgrade your license.

Error messages about your function level are issued when you deploy:

About this task

Message BIP2276

- **Scenario:** Error message BIP2276 is displayed.

  ```
  The flow 'Message_flow', which includes a message flow of
  node type 'Node_type', cannot be deployed because the current
  fix pack function level of '<version>.<release>.<modification>.<fix>'
  does not support this node.
  Use mqsichangebroker -f all' to enable this functionality.
  ```

- **Explanation:** The broker received an instruction to create a message flow node of type *Node_type* in message flow *Message_flow*. The broker cannot create nodes of this type because the new functions have not been enabled for this broker. Use the following command to enable this function, and all other functions:

  ```
  mqsichangebroker MB7BROKER -f all
  ```

- **Solution:** Either change the flow to avoid using the unavailable node, or enable the new functions by using the **mqsichangebroker** command; see "**mqsichangebroker** command" on page 3723.

Error messages are issued when you deploy to z/OS:
Procedure

- **Scenario:** The following messages are written to the log when you deploy to z/OS:

  ```
  +(MQ05BRK) 0 BIP2070E: A problem was detected with WebSphere MQ while issuing
  MQPUT for WebSphere MQ queue SYSTEM.BROKER.ADMIN.REPLY, WebSphere MQ queue
  manager QM_01.  MQCC=2, MQRC=2030.
  +(MQ05BRK) 0 BIP2068E: The broker was unable to put an internal configuration
  message to message queue SYSTEM.BROKER.ADMIN.REPLY.
  ```

- **Explanation:** The transmission queue is not large enough for the messages that are issued by WebSphere Message Broker.
- **Solution:** See the WebSphere MQ documentation for details on how to increase the size of the transmission queue.

Expected serialization of input is not occurring for a shared queue that serves multiple instances of a message flow on z/OS:
Procedure

- **Scenario:** Expected serialization of input is not occurring for a shared queue that serves multiple instances of a message flow on z/OS.
- **Explanation:** On z/OS, WebSphere MQ supports serialized access to shared resources, such as shared queues, through the use of a connection tag (serialization token) when an application connects to the queue manager that participates in a queue sharing group.

  This problem can occur for a number of reasons:

  – You have assigned different serialization tokens to the input nodes of the flows that get messages from the shared queue.
  – The message flows are running in the same execution group. Serialization can only be effected between flows that are running in different execution groups, either on the same broker, or on different brokers whose queue managers participate in the same queue sharing group.
  – The queue managers on which the brokers are running are not participating in a queue sharing group, or not participating in the same queue sharing group.

- **Solution:** Carry out the following checks:

- Check that the same z/OS serialization token has been configured for the MQInput nodes in each flow that use the shared queue.
- Check that the message flows are running in different execution groups.
- Check that the broker queue managers are part of the same queue sharing group, and that no errors are reported from the queue managers in the z/OS system log (SDSF log).

You create a configurable service, then deploy a message flow and inbound adapter, but the deployment fails:
Procedure

- **Scenario:** You create a configurable service, then deploy a message flow and inbound adapter, but the deployment fails.
- **Explanation:** When you create a configurable service for an adapter that has not yet been deployed, the connection properties are not fully validated until you deploy the adapter and the message flow that uses that adapter. Therefore, the properties that you set on the configurable service might be invalid.
- **Solution:** Inspect the error message that is returned from the deployment and determine whether the deployment failed because of invalid connection properties on the configurable service. If so, use the **mqsichangeproperties** command to correct the properties, or use the **mqsideleteconfigurableservice** command to use the properties that are set on the adapter. Restart the broker and redeploy the message flow.

You have created a WebSphere Adapters message flow that uses secondary adapters, and a naming clash has occurred in the secondary adapters or message sets:
Procedure

- **Scenario:** You have created a WebSphere Adapters message flow that uses secondary adapters, and a naming clash has occurred. You need to know which adapters and message sets the WebSphere Adapters node is using, and if the method names in the adapters or the message type names in the message sets are clashing.
- **Explanation:** Method names must be unique across all primary and secondary adapters, and the message set that is created must not contain any types that share the same name and namespace of existing message sets. Information about secondary adapters and message sets is written to user trace at the following stages.
  - When the node is first deployed, information about the current set of secondary adapters and message sets is reported.
  - When adapters and message sets are deployed subsequently, information about them is reported at the time of deployment.
  - When the message flow, broker, or execution group is stopped then restarted, information about the entire set of secondary adapters and message sets is written to user trace, including those secondary adapters and message sets that are deployed after the message flow was first deployed.

  This information is reported by the following messages.
  - BIP3432 and BIP3434 list the adapters and message sets that are available when the node is deployed or when the broker, execution group, or message flow are restarted.
  - BIP3433 reports that a secondary adapter is being deployed and added to the set.
  - BIP3435 reports that a secondary message set is being deployed.
  - BIP3436 identifies the message set in which each type is being defined.

- BIP3437 identifies message sets that attempt to redefine a type that is already defined.
- BIP3438 identifies the adapter in which each method is being defined.
- BIP3439 identifies adapters that attempt to redefine a method that is already defined.

Messages BIP3436, BIP3437, BIP3438, and BIP3439 are issued when a message is processed to report whether definitions are used for that message.

- **Solution:** The following steps describe how to use user trace when working with secondary adapters.
  - **When you deploy the flow for the first time**
    1. Start user trace.
    2. Deploy your message flow, primary adapter, primary message set, and any secondary adapters or message sets that are ready.
    3. Read user trace, looking out for the messages described.
    4. Optional: Reset user trace, stop and restart the message flow, then read user trace again, looking out for the messages described.
  - **When you add new adapters and message sets**
    1. Start user trace.
    2. Deploy the secondary adapters and message sets.
    3. Read user trace, looking out for the messages described.
    4. Optional: Reset user trace, stop and restart the message flow, then read user trace again, looking out for the messages described.

  When you have identified where the naming clash has occurred, you can ensure that names are unique by editing them in the following ways:
  - Edit method names by clicking **Edit Operations** on the Service Generation and Deployment Configuration panel of the Adapter Connection wizard.
  - Edit the namespaces of the types in the message set by using the Business Object Namespace control on the Adapter Connection wizard.

You get an authority check failure when you deploy to z/OS: Procedure

- **Scenario:** You deploy a BAR file to z/OS, and you get an authority check failure like the following message:

```
ICH408I USER(MI09STC ) GROUP(TSOUSER ) NAME(WMB TASK ID         ) 672
  MI09.SYSTEM.BROKER.AUTH.default CL(MQQUEUE )
  PROFILE NOT FOUND - REQUIRED FOR AUTHORITY CHECKING
  ACCESS INTENT(READ   )  ACCESS ALLOWED(NONE   )
```

- **Explanation:** The z/OS queue manager accesses the MQQUEUE security profile class by default. If you have enabled broker administration security, and you have specified the names of one or more execution groups in mixed case, the security profiles that you set up are defined in MXQUEUE, because the associated authority queue names are also in mixed case. MQQUEUE does not hold the expected information.
- **Solution:** Change the security profile case parameter for the queue manager. You can use the ALTER QMGR command to update the SCYCASE parameter to M, or you can modify your start job JCL to set or change this parameter, and restart the queue manager.

Deployment fails when you have circular project dependencies:

Procedure

- **Scenario:** Circular project dependencies exist in the projects that you are deploying and deployment fails.
- **Explanation:** A circular project dependency occurs when two or more projects depend on each other.

  For example, File A in Project X depends on File B in Project Y, and File C in Project Y depends on File D in Project X. For Project X to build successfully, Project Y must be built first so that Project X can resolve the dependency on File B. However, for Project Y to build successfully, Project X must be built first so that Project Y can resolve the dependency on File D. The WebSphere Message Broker Toolkit is based on the Eclipse platform, which does not support circular project dependencies.
- **Solution:** To deploy successfully, avoid circular project dependencies.

  For example, if Project X depends on File B in Project Y, and Project Y depends on File D in Project X, move File B to Project Z. The projects must be built in the order Project Z, Project X, then Project Y so that the dependencies can be resolved.

Error messages are issued when you deploy:
About this task

Additional error messages that might be generated during a deployment are explained in this section.

Message BIP1106 with WebSphere MQ reason code 2030

- **Scenario:** Error message BIP1106 is issued with reason code 2030 when you are deploying a large message set.
- **Explanation:** The size of the message exceeds the maximum message length of the transmission queue to the broker queue manager.
- **Solution:** Increase the maximum message length for the transmission queue using the WebSphere MQ **alter qlocal** command, where the maximum message length (maxmsgl) is in bytes:

  ```
  alter ql(transmit_queue_name) maxmsgl(104857600)
  ```

  For more information about this command, see the *System Administration Guide* section of the WebSphere MQ Version 7 Information Center online.

Message BIP1835

- **Scenario:** Error message BIP1835 is displayed.
- **Explanation:** The message set that you are deploying produces a message set dictionary that is larger than the internal limit of 4 MB. This problem can occur if you have many large message definitions defined to the same message set.

  The size of an exported message set is not a good indication of the size of the message set dictionary that is generated at the time of deployment, because the exported message set is stored as XML. XML can be verbose, but the dictionary has a more compact internal format.
- **Solution:** Split the message definitions into several smaller message sets.

Message BIP2066

- **Scenario:** You have initiated a deployment request; for example, you have deployed a BAR file to an execution group. Error message BIP2066 is returned one or more times.

- **Explanation:** The deployment request was not acknowledged by the execution group before the sum of the values for the broker timeout parameters **ConfigurationChangeTimeout** and **InternalConfigurationTimeout** expired.
- **Solution:** Increase these timeout values by specifying the **-g** and **-k** parameters of the **mqsicreatebroker** or **mqsichangebroker** command. See "Setting configuration timeout values" on page 3258 for information about factors that affect timeout values, and how to set appropriate values.

Message BIP2080

- **Scenario:** The broker has started an execution group; for example, if you have issued **mqsistart** for the broker, or an error has occurred and the execution group is being recovered. Error message BIP2080 is displayed one or more times.
- **Explanation:** The internal configuration request was not acknowledged by the execution group before the value of the **InternalConfigurationTimeout** (default 60 seconds) expired.
- **Solution:** Change the configuration timeout by specifying the **-k** parameter of the **mqsicreatebroker** or **mqsichangebroker** command. See "Setting configuration timeout values" on page 3258 for information about factors that affect timeout values, and how to set appropriate values.

Message BIP2241

- **Scenario:** Error message BIP2241 is displayed.
- **Explanation:** You are attempting to deploy a message flow containing a node that is not available on the target broker.
- **Solution:** Ensure that the version of the WebSphere Message Broker Toolkit in which the message flow has been developed matches the version of the broker to which the message flow is being deployed. If the message flow is using a user-defined node, or a node supplied in a SupportPac, ensure that the runtime node implementation has been correctly installed on the computer on which the broker is running. If your message flow includes a user-defined node, see "Installing user-defined extension runtime files on a broker" on page 3125. If your message flow includes a node provided in a SupportPac, see the installation information, if supplied, for the SupportPac.

Message BIP2242

- **Scenario:** Error message BIP2242 is displayed.
- **Explanation:** The deployment request (configuration change) was not accepted before the timeout value set by the broker parameter **ConfigurationChangeTimeout** expired. This configuration timeout value must be long enough for the message flow to complete processing its current message, then accept the deploy request; the default is 300 seconds.
- **Solution:** Set the configuration timeout values by specifying the **-g** and **-k** parameters of the **mqsicreatebroker** or **mqsichangebroker** command.

Message BIP3226

- **Scenario:** Error message BIP3226 is displayed; for example:

  ```
  (Semipersistent_Compute1.Main, 27.89) : Array index evaluated to '0' but must
  evaluate to a positive, nonzero integer value.
  ```

  The first insert in BIP3226 (in this example, `Semipersistent_Compute1.Main`) identifies the node and routine in which the statement occurs. The second insert (in this example, `27.89`) identifies the approximate line and column of the index value shown in the third insert (in this example, '0').

- **Explanation:** The validity of using a field reference index of zero was corrected in WebSphere Message Broker Version 7.0. If you have statements in your ESQL modules that include an index of zero, error BIP3226E is generated.

  For example, your ESQL module might contain the following statement:

  ```
  SET OutputRoot.XMLNSC.Top.A[0].B = 42;
  ```

- **Solution:**

  You must correct all ESQL statements that use an index of zero to use an index of 1. Statements might use a variable as well as a literal value for the index; check for both possible situations. For example, your changed code might read:

  ```
  SET OutputRoot.XMLNSC.Top.A[1].B = 42;
  ```

Message BIP7053S

- **Scenario:** When you deploy to a broker, error message BIP7053S is displayed.
- **Explanation:** This error occurs in a multi TCP/IP stack environment, and indicates that the UNIX System Services (USS) TCP/IP environment has not been set up correctly.

  WebSphere Message Broker uses USS functions to obtain the host name for a particular system. The following error message is displayed if the default host name is not set up correctly in the USS environment:

  ```
  BIP7053S: Broker $SYS_mqsi 0 unexpected Java exception java.lang.Error:
  -2103399272!java.net.UnknownHostException :
  Hostname: Hostname
  ```

  The host name that is reported in the error message is the one that has been returned to the broker as a result of the `gethostname` call.

- **Solution:** Ensure that the TCP/IP environment is configured correctly in USS.

Tagged/Delimited String (TDS) Validator error

- **Scenario:** You try to deploy a message set with a TDS wire format that has an error.
- **Explanation:** The following extract from an error log illustrates what you might see for a TDS Validator error. In this case, the cause of the problem is that the element `Town` does not have a tag defined.

  ```
  TDS Extractor Trace File
  ========================

  Beginning Extract..

  Extracting Identification Info
  Extracting Project Info
  Extracting Messages
  Extracting Elements
  Extracting Compound Types
  Extracting Type Members
  Extracting Type Members
  Extracting Type Members
  Extracting Type Members
  Extracting Type Members
  Beginning Indexing..

  Creating Member IDs to Tags Index Table.

  Beginning Validation..

  Validating Project
  Validating Types
  ERROR: TDSValidator::ValidateTypeMemberSimpleElement:
  ```

```
    Simple elements in a type with Data Element Separation attribute = Tagged
    Delimited must have the following attribute set:
    Element Level - Tag
  (Element ID: Town)
  (Type ID: AddressType)
  Return Code: -80

  Validating Messages

  Trace Info
  ==========
  EXCEPTION: TDSValidator::Validate:
    TDS Validation failed.
      1 errors
      0 warnings
  Return Code: -1
```

- **Solution:** Use the information in the error log to correct the problem.

:

Related concepts:

"Packaging and deployment overview" on page 3210
Deployment is the process of transferring data to an execution group on a broker so that it can take effect in the broker. Message flows and associated resources are packaged in broker archive (BAR) files for deployment.

"WebSphere Adapters deployment" on page 3219
When you have run the Adapter Connection wizard and created a message flow, you must deploy the resources that are generated by adding them to a broker archive (BAR) file.

Related tasks:

"Resolving problems when deploying message flows or message sets" on page 3436
Use the advice given here to help you to resolve common problems that can arise when you deploy message flows or message sets.

Chapter 11, "Packaging and deploying," on page 3209
Package resources that you create in the WebSphere Message Broker Toolkit, such as message flows, and deploy them to execution groups on brokers.

"Are the Linux and UNIX environment variables set correctly?" on page 3350
Use the **mqsiprofile** command to set a command environment.

Related reference:

"**mqsideploy** command" on page 3872
Use the **mqsideploy** command to make a deployment request to the broker.

"**mqsilist** command" on page 3882
Use the **mqsilist** command to list installed brokers and their associated resources.

"**mqsistart** command" on page 3965
Use the **mqsistart** command to start the specified broker if all initial verification tests complete successfully.

"**mqsistop** command" on page 3972
Use the **mqsistop** command to stop the specified component.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

Related information:

WebSphere MQ Version 7 Information Center online

Resolving problems that occur after deployment of message flows

Use the advice given here to help you to resolve common problems that can arise after deploying message flows or message sets.

You receive exception ('MQCC_FAILED') reason '2042' ('MQRC_OBJECT_IN_USE'): Procedure

- **Scenario:** This exception message occurs when the following conditions are met:
  1. You have a message flow containing a SOAPRequest node that is using the JMS Transport.
  2. The message flow has additional instances defined, and is running under load.
  3. You are using the WebSphere MQ JMS provider.
  4. You have not specified a reply-to destination, and so are using JMS temporary dynamic queues.
- **Explanation:** If you do not specify a reply-to destination, and you are using the WebSphere MQ JMS provider in a message flow with additional instances, you must configure JMS temporary dynamic queues before deploying your message flow.
- **Solution:** Complete the steps in the topic "Configuring JMS temporary dynamic queues for the WebSphere MQ JMS provider" on page 4848.

Your XSLTransform node does not work after deployment: About this task

Two scenarios explain why your XSLTransform node might not work after deployment:

Error messages are displayed indicating that the style sheets were not found

Procedure

- **Scenario:** Error messages are displayed indicating that the style sheets were not found.
- **Explanation:** This error message is displayed if the broker cannot find the style sheets or XML files required, or if the content of a style sheet or XML file is damaged and therefore no longer usable. This error message can happen if a file system failure occurs during a deployment.
- **Solution:** If the style sheets or XML files are damaged, redeploy the damaged style sheets or XML files.

What to do next

You get unexpected transformation results

- **Scenario:** You get unexpected transformation results.
- **Explanation:** For complex message flows, incompatibility might arise among style sheets and XML files after a deployment. The two typical reasons for this error are:
  - Only part of the cooperating style sheets or XML files are deployed and updated (a file system failure could cause this failure).
  - Multiple XSLTransform nodes that are running inside the same execution group are supposed to use compatible style sheets, but are using different versions to process the same incoming message.
- **Solution:** If only part of the cooperating style sheets or XML files are deployed and updated, resolve all incompatibility by redeploying the compatible versions. To avoid multiple XSLTransform nodes using different versions of the style sheet, pause relevant message flows in the target execution group before performing the deployment, then restart the flows.

Related concepts:

"Packaging and deployment overview" on page 3210
Deployment is the process of transferring data to an execution group on a broker so that it can take effect in the broker. Message flows and associated resources are packaged in broker archive (BAR) files for deployment.

Related tasks:

"Resolving problems when deploying message flows or message sets" on page 3436
Use the advice given here to help you to resolve common problems that can arise when you deploy message flows or message sets.

Chapter 11, "Packaging and deploying," on page 3209
Package resources that you create in the WebSphere Message Broker Toolkit, such as message flows, and deploy them to execution groups on brokers.

"Configuring JMS temporary dynamic queues for the WebSphere MQ JMS provider" on page 4848
Configure JMS temporary dynamic queues for the WebSphere MQ JMS provider, so that you can use them with SOAPRequest nodes, by using WebSphere MQ Explorer.

"Are the Linux and UNIX environment variables set correctly?" on page 3350
Use the **mqsiprofile** command to set a command environment.

Related reference:

"**mqsideploy** command" on page 3872
Use the **mqsideploy** command to make a deployment request to the broker.

"**mqsistart** command" on page 3965
Use the **mqsistart** command to start the specified broker if all initial verification tests complete successfully.

"**mqsistop** command" on page 3972
Use the **mqsistop** command to stop the specified component.

"XSLTransform node" on page 4968
Use the XSLTransform node to transform an XML message to another form of message, according to the rules provided by an XSL (Extensible Stylesheet Language) style sheet, and to set the `Message domain`, `Message set`, `Message type`, and `Message format` for the generated message.

Resolving problems that occur when debugging message flows

Advice on dealing with some of the common problems that you see when debugging message flows.

About this task

"Resolving problems that occur when starting and stopping the debugger" on page 3454

- "An endless "waiting for communication" progress bar is displayed when you start the debugger" on page 3454
- "The debugger seems to stop" on page 3454
- "The session ends abnormally while debugging" on page 3454
- "An error message is displayed indicating that the debug session cannot be launched" on page 3455
- "Errors are generated when you copy a message map into a message flow project" on page 3455

"Resolving problems when debugging message flows" on page 3456

- "The debugger does not pause at the next breakpoint" on page 3456
- "The message does not stop executing at any breakpoint" on page 3456
- "Editing problems occur in the Message Flow editor" on page 3456
- "Editing the MQ message descriptor (MQMD) causes unexpected behavior in the debugger" on page 3456
- "You cannot see the message content when debugging your message flow" on page 3456
- "An exclamation mark appears above a node during debugging" on page 3457
- "The message does not stop processing at breakpoints" on page 3457

"Resolving problems that occur after debugging:" on page 3458

- "You cannot change a message flow after debugging" on page 3458
- "You redeployed a debugged message flow but deployment hangs" on page 3458

Related concepts:

"Flow debugger overview" on page 3158
Use the flow debugger in the WebSphere Message Broker Toolkit to track messages through your message flows.

Related tasks:

"Starting the flow debugger" on page 3160
To start the flow debugger, you must attach it to an execution group. When the flow debugger is started, you can introduce test messages to your message flow.

"Attaching the flow debugger to an execution group for debugging" on page 3160
Before you can debug your message flow, you must attach the flow debugger to the execution group where your flow is deployed, then start a debugging session.

"Debugging a message flow" on page 3157
Use the tasks described in this section of the documentation to manage and work with the flow debugger.

"Debug: redeploying a message flow" on page 3189
If you want to change your message flow while you are debugging it, you must redeploy it to the execution group, then reattach the flow debugger.

Resolving problems that occur when starting and stopping the debugger

Use the advice given here to help you to resolve common problems that can arise when you debug message flows.

An endless "waiting for communication" progress bar is displayed when you start the debugger:
Procedure

- **Scenario:** After you click **Start Debugging**, you get an endlessly cycling progress bar entitled "waiting for communication". The "debug session started" message is not displayed in the information pane.
- **Explanation:** If the message flow has nodes with ESQL statements, the flow might not deploy even if the statements are correct syntactically. This situation can occur, for example, because of multiple declarations or uninitialized variables (that is, semantic problems that the syntax parser does not pick up). Always check the WebSphere Message Broker Toolkit Administration log to confirm that the debugged version of your message flow has deployed successfully; it has the same name as the original message flow, with the suffix `_debug_`.

  If the message flow does not deploy properly, the debugger cannot establish communication with the flow, and you see the endless progress bar.
- **Solution:** Click **Cancel** to clean up and return to a good state, then fix your errors and try again. As a check, see if your flow can deploy without the debugger.

The debugger seems to stop:
Procedure

- **Scenario:** You are debugging a message flow and continue after encountering a breakpoint. However, nothing seems to happen and after about a minute, a progress bar appears, indicating that the debugger is waiting for communication.
- **Explanation:** This situation can occur for the following reasons:
  - The message flow might have encountered a time-intensive operation, such as a huge database query, and you must wait for the flow to complete the action.
  - The broker ended, or some other extraordinary condition occurred, and communication was lost. In this case, click **Cancel** to stop the debug session.

The session ends abnormally while debugging:
Procedure

- **Scenario:** After debugging a message flow, the session ends abnormally and you still have the debug instance of the message flow (mf_debug_) deployed to the broker's execution group. You are concerned that this is going to affect the operation of the flow, and want to put the execution group back to its original state.
- **Explanation:** The orphaned message flow should behave as the flow would have done normally, and the Debug nodes have no effect on message processing. If you have a small number of nodes in the message flow, corrective action makes no noticeable difference to the flow, apart from its name. However, if you have a large message flow (that is, more than 15 nodes or several subflows), take the corrective action described later in this section, because the performance of message processing might be affected.
- **Solution:** Redeploy the broker.

A full redeploy of the broker should replace the orphaned flow with the original message flow. If this action has no effect, remove the orphaned flow from the execution group, deploy, then add the flow and deploy to restore the original state of the broker as it was before the debugging session.

An error message is displayed indicating that the debug session cannot be launched:
Procedure

- **Scenario:** You try to relaunch or invoke a new debug session but when you click the green **Debug** icon  , an error message is displayed stating: `Cannot launch this debug session`.
- **Explanation:** When you click **Debug** , it relaunches the last debug session. It fails if you have not created a debug session previously. It also fails if the broker and execution group that were attached previously in a debug session are no longer running, or have been restarted; the session cannot be reattached without re-selection of the broker and execution group process instance.
- **Solution:**
  1. Close the error message and click the arrow immediately to the right of the **Debug** icon.
  2. Re-select or modify the broker and execution group information from the previous debug launch configuration by clicking **Debug** in the menu and selecting the previous debug launch configuration. See "Attaching the flow debugger to an execution group for debugging" on page 3160 for more information.

Errors are generated when you copy a message map into a message flow project:
Procedure

- **Scenario:** You are copying a message map into a message flow project and errors have appeared in the task list.
- **Explanation:** The message flow project did not have the correct references set before you copied the message mapping.
- **Solution:** These errors remain in the task list, even if you reset the project references immediately after copying. Therefore, you must perform a clean build of the message flow project.

Related concepts:

"Flow debugger overview" on page 3158
Use the flow debugger in the WebSphere Message Broker Toolkit to track messages through your message flows.

Related tasks:

"Resolving problems that occur when debugging message flows" on page 3453
Advice on dealing with some of the common problems that you see when debugging message flows.

"Starting the flow debugger" on page 3160
To start the flow debugger, you must attach it to an execution group. When the flow debugger is started, you can introduce test messages to your message flow.

"Attaching the flow debugger to an execution group for debugging" on page 3160
Before you can debug your message flow, you must attach the flow debugger to the execution group where your flow is deployed, then start a debugging session.

"Debugging a message flow" on page 3157
Use the tasks described in this section of the documentation to manage and work with the flow debugger.

"Debug: redeploying a message flow" on page 3189
If you want to change your message flow while you are debugging it, you must redeploy it to the execution group, then reattach the flow debugger.

Resolving problems when debugging message flows

This topic contains advice for dealing with some of the common problems that can arise when debugging message flows.

The debugger does not pause at the next breakpoint:
Procedure

- **Scenario:** The message flow debugger does not pause at the next breakpoint in your message flow.
- **Solution:** Perform the following checks:
  1. Check whether your DataFlowEngine is running; if it is not, restart it.
  2. Check the input queue. If your input queue has the leftover messages from the previous time that you used the debugger, clear them before you send a new message.

The message does not stop executing at any breakpoint:
Procedure

- **Scenario:** The message does not stop executing at any breakpoint after you attach to the debugger.
- **Explanation:** This error could be caused by a timing problem, or if you have set the wrong parameters for the debug session.
- **Solution:** Perform the following steps.
  1. Check your launch configuration settings, ensuring that you have specified the correct Flow Project, HostName and Flow Engine for the debug session.
  2. Restart the debug session.

Editing problems occur in the Message Flow editor:
Procedure

- **Scenario:** While debugging a message flow, editing problems occur when you are using the Message Flow editor.
- **Solution:** Do not attempt to edit the message while the flow debugger is attached. To edit a message flow, detach the debugger, edit the message flow, then redeploy the message flow.

Editing the MQ message descriptor (MQMD) causes unexpected behavior in the debugger:
Procedure

- **Scenario:** You edit properties of the message MQMD descriptor in the Message Set editor, but this causes unexpected behavior in the debugger.
- **Explanation:** If you edit the content of the MQMD descriptor, these fields take a certain range of values. You need to know these ranges before editing the properties. Unless you explicitly specify the value of these fields, they take default values, and certain fields might not have been specified in the message. The values in the fields that are not set explicitly in the message are default values; do not change these unless you are aware of their importance or the possible range of values.

You cannot see the message content when debugging your message flow:

Procedure

- **Scenario:** You are using the message flow debugger, and you can see the message passing through the message flow, but you cannot see the content of the message.
- **Solution:** Open the Flow Debug Message view by clicking **Window** > **Show View** > **Other** > **Message Flow** > **Flow Debug Message**, then **OK**.

An exclamation mark appears above a node during debugging:
Procedure

- **Scenario:** In the Message Flow editor, an exclamation mark (!) is displayed above a node during debugging.
- **Explanation:** An exception has occurred in the node during debugging.
- **Solution:** Look under the ExceptionList in the Variables view of the Debug perspective to find out what error has occurred.

The message does not stop processing at breakpoints:
Procedure

- **Scenario:** Message processing continues when a breakpoint is encountered.
- **Explanation:** This error could be caused by a timing problem, or if you have set the wrong parameters for the debug session.
- **Solution:** Check your launch configuration setting. Ensure you have specified the correct Flow Project, HostName and Flow Engine for the debug session. Restart the debug session.

You cannot see where the debugger is in the message mapping editor:
Procedure

- **Scenario:** The message mapping editor has opened in the toolkit, but it is unclear where the debugger is in the map.
- **Explanation:** The source lookup path for the message map file is not configured correctly.
- **Solution:** Check your debug launch configuration settings and ensure you have configured the source lookup path for the message map file correctly.

When debugging a message map, the debugger does not move to the next field:
Procedure

- **Scenario:** You are debugging a message map, and the debugger does not move to the next field. You have to click the **Step over** button multiple times.
- **Explanation:** The source lookup path for the message map file is not configured correctly.
- **Solution:** Check your debug launch configuration settings and ensure you have configured the source lookup path for the message map file correctly.

When debugging a message map, the debugger does not move out of the mapping node:
Procedure

- **Scenario:** You are debugging a message map, and the debugger does not move out of the message map.
- **Explanation:** The source lookup path for the message map file is not configured correctly.
- **Solution:** Check your debug launch configuration settings and ensure you have configured the source lookup path for the message map file correctly.

The message flow stops at a collector node:
Procedure

- **Scenario:** Message processing stops after selecting the Step into Source Code icon on a Collector node.
- **Explanation:** The collector node is a multithreaded node and the thread is ended by selecting Step into Source Code.
- **Solution:** Set a breakpoint manually after the collector node.

Related concepts:

"Flow debugger overview" on page 3158
Use the flow debugger in the WebSphere Message Broker Toolkit to track messages through your message flows.

Related tasks:

"Resolving problems that occur when debugging message flows" on page 3453
Advice on dealing with some of the common problems that you see when debugging message flows.

"Starting the flow debugger" on page 3160
To start the flow debugger, you must attach it to an execution group. When the flow debugger is started, you can introduce test messages to your message flow.

"Attaching the flow debugger to an execution group for debugging" on page 3160
Before you can debug your message flow, you must attach the flow debugger to the execution group where your flow is deployed, then start a debugging session.

"Debugging a message flow" on page 3157
Use the tasks described in this section of the documentation to manage and work with the flow debugger.

"Debug: redeploying a message flow" on page 3189
If you want to change your message flow while you are debugging it, you must redeploy it to the execution group, then reattach the flow debugger.

Resolving problems that occur after debugging:

This topic contains advice for dealing with some of the common problems that can arise after debugging message flows.

You cannot change a message flow after debugging:
Procedure

- **Scenario:** You were debugging, but now your message flow seems to be stuck. When you put a new message in, nothing happens.
- **Explanation:** This might be because a message was backed out, but you have not set the **`Backout requeue name`** property of your input queue.
- **Solution:** Set the **`Backout requeue name`** property to a valid queue name (such as the name of the input queue itself) and your flow will become unstuck.

You redeployed a debugged message flow but deployment hangs:
Procedure

- **Scenario:** You found problems in your message flow by using the debugger. You changed your message flow and then redeployed it, but now the deployment hangs.
- **Solution:** Make sure that when you redeploy the flow to an execution group, the execution group is not still attached to the debugger.

Related concepts:

"Flow debugger overview" on page 3158
Use the flow debugger in the WebSphere Message Broker Toolkit to track messages through your message flows.

Related tasks:

"Resolving problems that occur when debugging message flows" on page 3453
Advice on dealing with some of the common problems that you see when debugging message flows.

"Starting the flow debugger" on page 3160
To start the flow debugger, you must attach it to an execution group. When the flow debugger is started, you can introduce test messages to your message flow.

"Attaching the flow debugger to an execution group for debugging" on page 3160
Before you can debug your message flow, you must attach the flow debugger to the execution group where your flow is deployed, then start a debugging session.

"Debugging a message flow" on page 3157
Use the tasks described in this section of the documentation to manage and work with the flow debugger.

"Debug: redeploying a message flow" on page 3189
If you want to change your message flow while you are debugging it, you must redeploy it to the execution group, then reattach the flow debugger.

Resolving problems when developing message models

This topic contains advice for dealing with some common problems that can arise when working with message sets.

About this task

Message definition files:

- "Error message BIP5505 is issued because input data is not valid for the data type" on page 3464

Other:

- "A deprecation error is issued on an imported `.mrp` file" on page 3465
- "User trace detects an element length error" on page 3465
- "MRM dateTime value has changed after it has been parsed" on page 3466

Your message definition file does not open

Procedure

- **Scenario:** You have created a complex type with a base type that causes the types to be recursive, and now your message definition file does not open.
- **Solution:** Recover the last saved version of your message definition file from the local history or from your repository.

A message definition file error is written to the task list

Procedure

- **Scenario:** An error is written to the task list indicating that the message definition file has been corrupted.
- **Explanation:** This error message appears when the base type of the complex type is itself, or a circular definition of two or more simple types. This type of recursion is not supported.
- **Solution:** Examine your code and ensure that the type of recursion described above does not occur.

Your physical format property values have reverted to defaults

Procedure

- **Scenario:** Someone has sent you a message definition file. You have added it to a message set project, but all the physical format property values have reverted to defaults.
- **Explanation:** You cannot transfer message definition files on their own in this way because these files are not entirely independent. It is the `messageSet.mset` file that lists all the physical formats that are applicable to a message set. For example, if you have a message set *A* with CWF format *Binary1*, and a message set *B* without any physical formats, a message definition file copied from *A* to *B* does not show *Binary1* as a known physical format (although the properties are still in the message definition file).
- **Solution:** If possible, request the entire message set project.

  Alternatively, add physical formats to the receiving message set so that they match the originating message set. Then any dormant properties become visible. However, make sure that the message set level physical format properties match, or those object properties that inherit defaults from the message set level will be incorrect.

You are unable to enter text in the Message Definition editors

Procedure

- **Scenario:** You are unable to enter text in the Message Definition Overview editor, or change the text of an enumeration or pattern facet in the Message Definition Properties editor.
- **Solution:** The Message Definition editors are cell editors, and to enable them you have to double-click the cell. The first click selects the cell; the second click puts the cell into active state, allowing you to edit it.

Objects in your message definition file are not listed in alphabetic order

Procedure

- **Scenario:** The objects in your message definition file (`.mxsd`) are listed in the order that you created them, but you want them to be in alphabetic order.
- **Solution:**
  1. In the Broker Application Development perspective, double-click the message definition file to open it in the Outline view.
  2. Click **az** (at the top of the Outline view).

  This action changes the order of objects within each of the message definition file's folders (the Messages, Types, Groups, Elements, and Attributes folders) to be alphabetic.

Error messages are written to the task list after you have imported related XML Schema files

Procedure

- **Scenario:** When you have finished importing a collection of related XML Schema files, you find that the WebSphere Message Broker Toolkit task list contains error messages for the message set project, indicating that referenced types or other objects cannot be found.
- **Explanation:** These errors are typically an indication that one message definition file includes or imports another message definition file, but that the Message Definition editor is unable to resolve the specified link.
- **Solution:**
  - Check that a message definition file exists in the WebSphere Message Broker Toolkit for each of the related XML Schema files that you are importing. If the new message definition files have been created, they appear in the Broker Development view.
  - Using the Properties Hierarchy in the Message Definition editor, check that any message definition files that include or import other message definition files are correctly locating the target files.

  One common scenario is where you have two XML Schema files *X* and *Y*, which both exist in the same directory in the file system but where *X*, which includes *Y*, is defined with a real target namespace, while *Y* is defined in the notarget namespace. After you import the two files, *X* is placed into the location that is determined by its namespace, but *Y* is placed into the default namespace location used for files defined in the notarget namespace. This situation causes the link between the two files to break, and you must modify *X* so that it correctly includes *Y* in its new relative location.

A group contains two different elements with the same XML name in the same namespace

Procedure

- **Scenario:** A warning is written to the task list because a group contains two different elements that have the same XML element name, in the same namespace.
- **Explanation:** When a message that has an XML physical format is validated, the validation includes a test to identify any part of a message definition where the parser might not be able to uniquely determine which element is represented by the XML name. This test generates a warning when a group contains two different XML elements with the same element name in the same namespace, even in cases where the duplication is legitimate.

- **Solution:** Determine whether the duplication that is identified in the warning message is in fact a problem that needs to be corrected, or whether it has arisen because of two elements on opposite sides of a choice sharing the same XML name, in which case the duplication is legitimate because no ambiguity can occur.

You are unable to set up a message definition file to include a message definition file within another message definition file

Procedure

- **Scenario:** You are unable to set up the include property of a message definition file to include a second message definition file that is included within another message definition file.
- **Explanation:** A message definition file can reference objects in another message definition file only if the first file references the second file directly. For example, if three message definition files exist, *A*, *B* and *C*, where *A* includes *B* and *B* includes *C*, *A* can reference objects in *B*, and *B* can reference objects in *C*, but *A* cannot reference objects in *C*.

  You might also encounter this problem after importing XML Schema files that have nested includes.
- **Solution:** You can work around this problem by including the message definition file directly, which, in the above example, would mean including *C* directly in *A*. Alternatively, you can define all the message set definitions that are in *C* directly in *B*, then delete *C* so that *A* needs to include only *B*.

You want to make the Properties tab the default tab in the Message Definition editor

Procedure

- **Scenario**: When using the Message Definition editor to edit your message definition files, you prefer to use the **Properties** tab rather than the **Overview** tab, but the **Overview** tab is always selected as the default.
- **Solution:** Use the WebSphere Message Broker Toolkit preferences to choose the **Properties** tab as the default tab:
  1. From the menu, click **Windows** > **Preferences**.
  2. Expand **Broker Development**, then **Message Set**.
  3. Expand **Editors** and select **Tab Extensions**.
  4. In the **Message Definition Editor** section, select the **Properties** and the **Overview** check boxes.
  5. Click **OK**.

Error message BIP5410 is issued because a union type cannot be resolved

Procedure

- **Scenario:** Error message BIP5410 is raised indicating that an element or attribute is based on a union type and that the element or attribute could not be cast to any member of the union.
- **Explanation:** When parsing an element or attribute that is based on a union type, the MRM XML parser uses an xsi:type attribute, where present, to resolve the union.

  If an xsi:type attribute is not present, or an attribute is being parsed, the parser tries each union member type in turn, attempting to cast to the associated simple type, until the cast succeeds. The order of precedence for the attempted cast is the order in which the member types are listed in the message model, under the union type, in the Outline view.

If the data cannot be cast to any of the simple types within the union, the union cannot be resolved and a parser error is reported.

- **Solution:** Perform the following checks:
  - Check that the message contains a valid value for the element or attribute that is identified in the error message.
  - Check that the member types of the union that are identified in the error message contain the correct list of simple types.
  - If possible, use an xsi:type attribute to resolve the union explicitly.

Error message BIP5395 is issued because an xsi:type attribute value does not correspond to a valid member type of the union

Procedure

- **Scenario:** Error message BIP5395 is issued, indicating that an element is based on a union type that has an xsi:type attribute with a value that should resolve the union explicitly to one of its modeled member types and that the xsi:type attribute value does not correspond to a valid member type of the union.
- **Explanation:** When parsing or writing an element or attribute that is based on a union type, the MRM XML parser or writer uses an xsi:type attribute, where present, to resolve the union. The parser resolves the value of the xsi:type attribute to a simple type in the dictionary and checks that the simple type is a valid member type of the union.

  If the xsi:type attribute identifies a type that is not a member type of the union, it reports an error.
- **Solution:** Perform one of the following steps:
  - Modify the message so that the xsi:type attribute identifies a valid member type of the union.
  - Check that the member types of the union identified in the error message contain the correct list of simple types.

Error message BIP5396 is issued because a data type does not correspond to any of the valid data types of the union

Procedure

- **Scenario:** Error message BIP5396 is issued, indicating that an element or attribute is based on a union type but the data type does not correspond to any of the valid data types in the union.
- **Explanation:** When writing an element or attribute that is based on a union type, the MRM XML writer uses an xsi:type attribute, where present, to resolve the union. If an xsi:type attribute is not present, the writer tries to match the data type of the literal value in the tree to a simple type in the union. If the literal value cannot be matched to any of the simple types in the union, it reports a writing error.
- **Solution:** Perform one of the following steps:
  - Check that the message or ESQL contains a valid value for the element or attribute.
  - Check that the union type on which the element is based contains the correct list of simple types.
  - Consider using an xsi:type attribute to resolve the union explicitly.
  - Consider changing the type of the element in the tree to correspond with one of the union member types.

A union type contains two or more simple types that are derived from the same fundamental type

Procedure

- **Scenario:** A warning is issued during logical validation indicating that a union type contains two or more simple types that are derived from the same fundamental type.
- **Explanation:** The broker does not apply value constraints until the data is in the logical tree. Therefore, it is not possible to choose between two simple types that are derived from the same fundamental type but with different constraints. For example, the broker cannot differentiate between a member type of integer with a range of 1-10 and another member type of integer with a range of 11-20, therefore it resolves the union to the first member type of integer.
- **Solution:** Ensure that the union type that is identified in the warning does not contain more than one simple type that is derived from the same fundamental type or ensure that the ordering of such member types is as desired.

A list type is based on a union that also contains a list type

Procedure

- **Scenario:** An error message is issued during logical validation, indicating that a list type is based on a union type that includes a list type as one of its member types.
- **Explanation:** Lists of lists are not legal. An item type of a list type cannot be a list type itself nor can it be derived at any level from another list type. Therefore, a list type cannot have an item type of a union type that includes a list type as one of its member types.
- **Solution:** Ensure that any union type specified as an item type for a list type does not include a list type as one of its member types.

A union type contains an enumeration or pattern facet

Procedure

- **Scenario:** An error message is raised during logical validation, indicating that a union type contains an enumeration or pattern facet that is not supported because enumeration and pattern facets must be specified directly on the member type.
- **Explanation:** The broker cannot support the union type facets of pattern and enumeration applied directly to a restriction of a union type. It can support facets directly on the chosen member type only. The Message Definition editor provides support for these facets to enable the import of schemas using direct facets on a restriction of a union type but it issues a warning notifying that they will be ignored by the broker.
- **Solution:** If you want to use the enumeration or pattern facets, specify them directly on the member type and not on the union type itself.

Error message BIP5505 is issued because input data is not valid for the data type

Procedure

- **Scenario:** A data conversion fails while a message is being read or written. Error message BIP5505 is issued, indicating that the input data was not valid for the data type.
- **Solution:** Perform the following checks:

- Ensure that the bit stream is correctly aligned to the model. If the incoming message is not consistent with the model, the wrong bytes are allocated to fields, and the data presented to a field is unlikely to contain valid data for the logical type.
- Ensure that the correct encoding and CodedCharSetId are applied to the body of the input message. If the wrong encoding is used, the input message bit stream is subject to byte swapping and changes to the byte order. This change can affect the nature of data that applies to a field and make it unusable.
- Ensure that an appropriate WebSphere MQ format is applied to the body of the input message. For example, if MQSTR is applied to a bit stream that contains numeric data, character conversions, rather than encodings, are used to translate the bytes. This use can change the meaning of bytes and make them not valid for the logical data type. If character conversions are used with negative external decimals, consider using the EBCDIC custom option.
- If the error is on a numeric field, ensure that the input data is compatible with the physical format settings. For example, if the field is signed, ensure that a valid sign byte has been passed in, that it corresponds to the orientation of the sign, and that it is included or excluded as appropriate.
- If the input message is found not to be compatible with the MRM message definitions, check the message at the source. If it is sent over channels, check that the conversion settings on the channels are correct.

A deprecation error is issued on an imported `.mrp` file

Procedure

- **Scenario:** You have imported an `.mrp` file, and get an error message indicating that complex elements or embedded simple types are deprecated.
- **Explanation:** Complex elements and embedded simple types have no exact equivalent in XML Schema. The closest equivalent to a complex type in XML Schema is to derive a complex type from a simple type. However, such a type is not allowed to contain elements; only attributes are allowed.
- **Solution:** If your complex type contains elements that cannot be modeled as attributes, set the **`Mixed content`** flag on the complex type. If you need to parse messages that contain mixed content, set the **`Mixed content`** flag on the parent complex type. The anonymous data that used to be modeled by the complex type or the embedded simple type, then appears as a self-defining node in the parsed message tree.

User trace detects an element length error

Procedure

- **Scenario:** You have defined an MRM message and when you execute the message flow, an error message is written to the user trace indicating that the length of an element is invalid.
- **Explanation:** You have added an element of type string to the complex type making up the message, but you have not defined the length of the string in the instance of the element in its type.
- **Solution:**
  1. Open the message definition file in the Message Definition editor.
  2. In the Outline view, expand the appropriate complex type, then click the element to display its properties in the editor area.
  3. In the Properties Hierarchy, make sure that the element is selected within the CWF physical format under **Physical Properties**.

4. Specify the properties that this element will have when it is part of the complex type; for example, in the above case, make sure that the **Length Count** value is set.
5. Save any changes in the Message Definition editor and redeploy the message.

MRM dateTime value has changed after it has been parsed

Procedure

- **Scenario:** You have defined an MRM dateTime element but the value created in the message tree is different from the value that you defined.
- **Explanation:** The parser uses lenient dateTime checking, converting out-of-band data values to the appropriate in-band value.
- **Solution:** See the information about lenient dateTime checking in "DateTime as string data" on page 6311 with particular reference to how years and fractional seconds are represented.

Related concepts:

"Message definition files" on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

"Message model objects: simple types" on page 1180
A *simple type* is an abstract definition of an item of data such as a number, a string, or a date.

Related tasks:

"Creating a message set" on page 2842
Use the New Message Set wizard to create a message set.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

"Working with a message definition file" on page 2863
Create, open, and delete a message definition file.

"Linking from one message definition file to another" on page 2921
Add an 'include', or an 'import' to the file that you want to reference.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message set projects and files" on page 6823

"Message Definition editor" on page 6804
The Message Definition editor is the default editor provided by the Broker Application Development perspective for editing message definition (`.mxsd`) files.

"DateTime as string data" on page 6311
You can use a string of pattern letters to specify the dateTime format.

XML Schema Part 0: Primer on the World Wide Web Consortium (W3C) Web site

Resolving problems when using messages

Use the advice given here to help you to resolve common problems that can arise when you use messages.

About this task

Procedure

A communication error is issued when you use the enqueue facility

Procedure

- **Scenario:** You use the enqueue or dequeue tools to put a message on a queue, but an error message is issued indicating that a communication error has occurred with the queue manager name.

- **Explanation:** The WebSphere MQ queue manager has not started.
- **Solution:** Restart the WebSphere MQ queue manager.

The enqueue facility is not picking up changes made to a message

Procedure

- **Scenario:** You are using the WebSphere Message Broker Toolkit message enqueue facility to put messages to WebSphere MQ queues. You have updated a message and want to put the message to the queue, but your changes do not seem to have been picked up.
- **Solution:**
  1. Close, then reopen your enqueue file.
  2. Select the message that you want to put to the queue.
  3. Save and close the enqueue file.
  4. Select the menu next to the **Put a message to a queue** icon.
  5. Click **Put message**.
  6. Click the enqueue file in the menu.
  7. Click **Finish**.

  This action puts your updated message to the queue.

You do not know which header elements affect enqueue

Procedure

- **Scenario:** When using the Enqueue editor, the accounting token, correlation ID, group ID, and message ID in the message header do not seem to affect behavior.
- **Explanation:** These fields do not affect behavior because they are not serialized properly.

Enqueue message files are still listed after they have been deleted

Procedure

- **Scenario:** Enqueue message files are still listed in the menu after they have been deleted.
- **Explanation:** Deleted enqueue files are not removed from the menu. Nothing happens if you selecting these files.

The ESQL transform of an XML message gives unexpected results

Procedure

- **Scenario:** You have created an XML message that looks like the following content:

  ```
  <?xml version ="1.0" standalone="no"?><!DOCTYPE doc
  [<!ELEMENT doc (#PCDATA)*>]><doc><I1>100</I1></doc>
  ```

  You apply the ESQL transform:

  ```
  SET OutputRoot.XMLNS.doc.I1 = 112233;
  ```

  This transform generates the XML message (after serialization):

  ```
  <?xml version ="1.0" standalone="no"?><!DOCTYPE doc
  [<!ELEMENT doc (#PCDATA)*>]<I1>112233<I1>><doc><I1>100</I1></doc>
  ```

  The new value for *I1* has been put inside the `DOCTYPE`, and has not replaced the value of *100*, as you expected.

- **Explanation:** The XML in your message contains two `doc` elements:
  - The `doctype` element
  - The `xmlElement` that represents the body of the message

  The parser has found the first instance of an element called `doc` and has created a child *I1* with the value *112233*.
- **Solution:** To assign a new value to the element *I1* within the message body element `doc`, explicitly identify the second `doc` element, in the following way:

  ```
  SET OutputRoot.XMLNS.(XML.tag)doc.I1 = 112233;
  ```

An XML message loses carriage return characters

Procedure

- **Scenario:** You are parsing an input XML message that contains carriage return or line feed characters, or you are writing an output XML message that contains carriage return line feed characters in an XML element. However, some or all the carriage return characters are not present in the output message.
- **Explanation:** This behavior is correct, as described by the XML specification, and occurs in the XML, XMLNS, or XMLNSC domains.

  In XML, the main line separator characters is a line feed character. Carriage return characters are accepted in an XML document, but an XML parser normalizes any carriage return line feed characters into a single-line feed character. For more information, see the latest XML specification at Extensible Markup Language (XML), in particular, Section 2.11 *End of Line Handling*.

  You cannot work around this problem by embedding your data into a CDATA section; the XML specification states that a CDATA section is intended only to escape blocks of text that contain characters that could be interpreted as markup. In addition, CDATA sections are not protected from the parser.

  You cannot use the `xml:space` attribute to preserve carriage return line feed characters; the XML specification states that normalization of carriage return and line feed characters is done before any other processing is performed.
- **Solution:** XML-compliant data must not rely on the presence of carriage return line feed characters because in XML, the line feed character is only a line feed character and XML-compliant applications or data must be aware that the parser normalizes any carriage return line feed characters.

The broker is unable to parse an XML message

Procedure

- **Scenario:** You receive a large XML file and wrap it in a SOAP envelope to be passed to a .NET web service. The broker is unable to parse the XML message
- **Explanation:** The message that you receive is defined as UTF-8 but it contains UTF-16 characters, such as £. The presence of these characters causes a problem with the parser: the broker is unable to parse the XML message because it contains an invalid character.
- **Solution:** Force the coded character set ID (CCSID) to 1208 instead of the default 437.

Unexpected characters are displayed when using the XSLTransform node on z/OS

Procedure

- **Scenario:** When using the XSLTransform node on z/OS, all the uppercase Os that are in an XML file on the incoming message are changed to underscore characters.

- **Explanation:** The XSLTransform node input message must come in as ASCII for the transform to work correctly. The XSLTransform node does not work with XML or XSL data in EBCDIC code. Java assumes a conversion from EBCDIC 1047. WebSphere Message Broker then converts to EBCDIC 500, because the CCSID is set to 500. EBCDIC 1047 and EBCDIC 500 are similar. Only uppercase 0, J, and Z are different. (J and Z are also converted incorrectly.) The conversions leave a string that is unreadable because it is really in ASCII. However, it does convert the 0 from an EBCDIC 1047 character to an EBCDIC 500 character.
- **Solution:** Change your program either to do a string assignment without any conversions, or specify that the string is in ASCII ISO-8859-1 (CCSID 819).

Error message BIP5004 is issued by the XMLNS parser

Procedure

- **Scenario:** Error message BIP5004 is issued, indicating that the XMLNS parser has encountered a problem with an input XML message.
- **Explanation:** This message is issued for a number of reasons. Some of the more commonly occurring scenarios when this message is issued are:
  - You have specified an invalid XML character in the input XML message.
  - You have included binary data in your XML message that, when treated as character data, is invalid.
  - The message has arrived as part of a WebSphere MQ message and the MQMD.CodedCharSetId does not correctly represent the XML text bit stream.
  - You have used characters that are recognized as markup.
- **Solution:**
  - Check that your sending application is sending only valid data. If, however, it is not possible to prevent invalid characters from being included in the XML message, represent it in BLOB domain and use the ESQL REPLACE function to replace or remove the invalid characters. You can then assign the modified bit stream to the required XML parser.

    In accordance with XML specification, a CDATA section can be used only to protect characters that would be interpreted as markup. It cannot be used to protect invalid characters or binary data from the XML parser.
  - If the input XML message contains binary data, ensure that the sending application is changed to represent this data as base encoded binary encoded data. If the application cannot be changed, represent the message in the BLOB domain, and extract and replace the binary data before the bit stream is assigned to the required XML parser.
  - Check that the incoming XML message is being represented by the correct MQMD.CodedCharSetId.

Error message BIP5378 is issued by the MRM parser

Procedure

- **Scenario:** Error message BIP5378 is issued, which reports a problem with a missing mandatory repeating element in an MRM message.
- **Explanation:** This message indicates that a mandatory repeating element is not present in the message. In previous releases, this condition was reported by error message BIP5374 which now reports only when a mandatory repeating element exists in the message but has the incorrect number of instances.

  If you have programmed automated error checking routines, review and change these routines if appropriate.
- **Solution:** Check your message definition to ensure that the element must be mandatory, and repeating. The error message tells you the elements that occur

before and after the expected location of the missing element. If the definition is correct, the message has not been created correctly by the sending application, which must be amended.

Error message BIP5005 is issued by the Compute node

Procedure

- **Scenario:** You send a simple XML message into a simple message flow. The message is:

  ```
  <doc><I1>100</I1></doc>
  ```

  The Compute node in the message flow contains the following ESQL:

  ```
  SET OutputRoot.XMLNS.abc = InputBody;
  ```

  You expect the following output message to be created:

  ```
  <abc><doc><I1>100</I1></doc></abc>
  ```

  The Compute node generates error message BIP5005 and does not implement the ESQL.
- **Explanation:** You are assigning an element of one type (`root`) to an element of another type (`xmlElement`). The parser does not do this implicit cast for you.
- **Solution:** You can do the cast yourself in the ESQL to achieve the result that you want, using either of the following two casts:

  ```
  SET OutputRoot.XMLNS.(XML.Element)abc = InputBody;
  ```

  or:

  ```
  SET OutputRoot.XMLNS.(XML.tag)abc = InputBody;
  ```

A message is propagated to the Failure terminal of a TimeoutControl node

Procedure

- **Scenario:** The TimeoutControl node receives a message with invalid, corrupt, or missing timeout information. The message is propagated to the Failure terminal of the TimeoutControl node and an exception list is generated.
- **Explanation:** The following error conditions can cause propagation to the Failure terminal:
  - A timeout request has no Action or no Identifier.
  - A SET request has an Identifier that matches an existing stored SET request for this TimeoutControl node (identified by the Unique Identifier property) and AllowOverwrite of the original request is set to FALSE.
  - A CANCEL request has an Identifier that does not match an existing stored SET request for this TimeoutControl node (identified by the Unique Identifier property).
  - A SET request has a Count of 0 or is less than -1.
  - The StartDate or StartTime are not in the correct format (or one of the understood keywords).
  - The calculated timeout is in the past.
  - The Interval is less than 0, or 0 with a Count of -1.
- **Solution:** Check for one or more of the error conditions listed here, and correct them.

Message processing fails within a TimeoutNotification node

Procedure

- **Scenario:** A message is propagated to the Failure or Catch terminal of a TimeoutNotification node.
- **Explanation:** If the processing of a timeout generates an error within the TimeoutNotification node, an exception list is generated and a message is propagated to the Failure terminal. This action is done under the same transaction, if one is being used. If the Failure terminal is not connected, propagation does not occur.

  If an error happens downstream of the TimeoutNotification node after a successful propagation (either to the Out or Failure terminal), the message is propagated to the Catch terminal (all under the same transaction). If the Catch terminal is not connected, or the propagation along the Catch flow fails, the processing of that timeout is rolled back.
- **Solution:** Ensure that the Failure and Catch terminals of your TimeoutNotification node have been connected correctly.

An MRM CWF message is propagated to the Failure terminal

Procedure

- **Scenario:** Your MRM CWF message is propagated to a Failure terminal, and generates error messages BIP5285, BIP5125, and BIP5181 or messages BIP5285, BIP5125, and BIP5288.
- **Explanation:** These errors report an inconsistency between the length of the message being processed, and the length of the message that is defined in the message model.
- **Solution:** Ensure that the length of the message as defined in the CWF layer is accurate. Check and correct the definition.

Problems with XML attributes

About this task

XML tags are written where XML attributes are expected, and vice versa.

Procedure

- **Explanation:** The XML domains and the XML Wire Format in the MRM domain have their own representation of XML attributes.
  - XML domains rely on setting a field type of XML.Attribute in the message tree, because they have no model to parse against.
  - For the XML Wire Format in the MRM domain, the message model indicates whether an element is an attribute or a tag, therefore the message tree does not need to reflect whether a field is an attribute or a tag.

  Therefore, if fields are copied out of the XMLNS or MRM domains, the fact that fields are attributes is lost. This loss happens if they are copied out to each other, or to another message tree, such as the environment tree.

  This problem typically appears in the following situations:
- **Scenario 1:** You are writing an XML message in the MRM domain, and XML tags are being written instead of XML attributes.
- **Solution 1:** Check that your message tree has the same structure and sequence as the message model. If the message tree does not match the message model, the field is written as self-defining, and consequently the XML Render property is not used.

- Switch on message validation. Validation shows where the message tree and message definition do not match.
- Alternatively, take a user debug trace of the message flow; BIP5493W messages indicate which fields are being written as self-defining. Use this information to ensure that the message tree matches the model. When you have corrected any discrepancies, attributes are correctly written.

- **Scenario 2:** An MRM message has been copied to an XMLNS domain, and the XML attributes are now written as tags.
- **Solution 2:** Take one of these actions:
  - Serialize the XML message in the MRM domain, for example using the ESQL ASBITSTREAM function, then use the ESQL CREATE PARSE clause to reparse the message using the required XML domain.
  - When copying fields between the MRM domain and XMLNS, copy attribute fields individually, and specifically specify XML.Attribute on the target XML field.
- **Scenario 3:** An XML message has been copied to another message tree, such as Environment. When the message is copied back to the XML message tree, XML attributes are now seen as XML tags.
- **Solution 3:** Serialize the XML message, for example using the ESQL ASBITSTREAM function, then use the ESQL CREATE PARSE clause to reparse the XML message into the required target message tree. See "CREATE statement" on page 5082 for an example.
- **Scenario 4:** A portion of a non-XML message tree has been detached and attached to an XML tree, and XML tags are now written as XML attributes.
- **Solution 4:** Do not detach and attach portions of message trees that are owned by different parsers. Instead, use a tree copy.
- **Scenario 5:** An XML tag is copied to an XML attribute and the XML attribute is not written in the output message.
- **Solution 5:** When referencing the source XML tag, use the ESQL FIELDVALUE function to copy the specific field value to the target XML attribute field.

An MRM XML message exhibits unexpected behavior

Procedure

- **Scenario:** Your message flow is processing a message that you have modeled in the MRM. The message tree has not been created as you expected, the output XML message does not have the expected contents, or the message contents are not being validated. No error message has been issued.
- **Explanation:** Two reasons might cause this problem:
  - **Explanation 1:** The XML physical format settings for a message set contain a property called Root Tag Name. This property defaults to MRM, in order to remain compatible with previous releases of the product. If you have not deleted the contents of this field, the MRM XMLNS parser expects the root tag for all XML messages to be MRM.

    Solution 1: Clear this field, or set it to the root tag used by all your XML messages. If you provide a value in this field, the root tag does not need to be modeled in all your message definitions.
  - **Explanation 2:** In order to remain backwards compatible, the broker recognizes the format XML and invokes the XMLNS parser with specific default values. If you have created an XML physical layer for this message with the name XML, the broker uses your definition. However, if you have not created an XML physical layer with this name, but have specified XMLNS as the format, either in the input node or the MQRFH2 header (when the

input bit stream is parsed to a message tree), the broker accepts the value specified and passes default values to the parser to create the message tree.

Similarly, if you set XML in the Properties folder for the output message in the Compute node, this value is passed to the parser when it creates the message bit stream from the message tree, typically in the output node.

The use of these default values by the parser might result in different content, structure, or both, for either message tree or output message. You can find further information about the action taken by the broker in the user trace log where the following information might be written:

```
XMLWorker::initializeParse file:C:\s000\src\cpi\pwf\xml\xmlworker.cpp
line:126 message:5409.BIPmsgs
No dictionary present have you specified Wire Format 'XML' in error? ,
UserTrace   BIP5409E: XML Worker: Wire Format 'XML' specified.
                      Default MRM XML settings are being used because wire format
                      identifier 'XML' was specified and not found.
                      This can be due to an incorrect setting of the wire format
                      identifier in a message.
```

Solution 2: If you have incorrectly entered the identifier of the format that you have defined, correct your code and try again. If you do not want the default action to be taken, define a physical layer that produces the required results.

The MRM parser has failed to parse a message because two attributes have the same name

Procedure

- **Scenario:** Two attributes in different name spaces have identical names. Error message BIP5117 is issued.
- **Explanation:** The MRM parser has failed to parse the message.
- **Solution:** Modify the attribute names so that they are not identical. This problem is a known limitation with the parser.

You encounter problems when messages contain EBCDIC newline characters

Procedure

- **Scenario:** If your bit stream input message contains EBCDIC newline (NL) characters, problems might arise if your message flow changes the target CCSID to an ASCII CCSID. For example, during conversion from CCSID 1047 (EBCDIC used for z/OS Open Edition) to CCSID 437 (US PC ASCII), an NL character is translated from hex '15' to hex '7F', which is an undefined character. The error occurs because no corresponding code point for the newline character exists in the ASCII code page.
- **Solution:** You can overcome the problem in these cases:
  - On a system where the queue manager uses an ASCII code set, make sure that incoming messages do not contain any EBCDIC NL characters by:
    - Specifying that WebSphere MQ performs the conversion at the input node
    - Setting the queue manager attribute to convert NL to Line Feed (LF)
  - Where the input bit stream is character data, you can use MRM Tagged/Delimited message sets in a Compute node to replace the NL character with the required output.

The MIME parser produces a runtime error while parsing a message

Procedure

- **Scenario:** A MIME message is received by a message flow and produces a runtime error when the message is parsed.
- **Explanation:** The following errors can cause the MIME parser to reject a message:
  - The MIME header is not correctly formatted.
  - Either the top-level MIME header block, or a MIME header block for a nested multipart part, does not have a valid Content-Type header.
  - The top-level Content-Type has a media type of `message`.
  - The top-level Content-Type has a media type of `multipart` and no boundary definition.
  - A MIME header block is not correctly terminated by a blank line.
  - The constituent MIME parts are not correctly separated by boundary lines.
  - A boundary parameter value occurs in the content of a MIME part.
- **Solution:** Check the MIME message for one or more of the error conditions listed here, and correct them.

Runtime errors are issued when you write a MIME message from the logical message tree

Procedure

- **Scenario:** You are writing a MIME logical message tree as a bit stream and the parser generates a runtime error.
- **Explanation:** The following errors can cause the MIME parser to reject a logical message tree:
  - The root of the tree is not called MIME.
  - The last child of MIME is not called Parts or Data.
  - The Parts element has a value-only element, and this element is not the first or last child of Parts.
  - The Parts element has children that are not value-only elements or Part children.
  - The Parts element does not have any children called Part.
  - The last child of a Data element is not a BLOB.
- **Solution:** Check the MIME logical message tree for one or more of the error conditions listed here, and correct them.

Output message has an empty message body

Procedure

- **Scenario:** You have unexpectedly encountered an empty message body, or the ASBITSTREAM function has produced a zero length BLOB.
- **Explanation:** This error can happen for the following reasons:
  - You have created a message tree folder in a user-defined node but have not associated it with an owning parser. An owning parser is not associated with the message tree if you have created standard elements by using code similar or equivalent to:

```
MbElement createElementAfter(int)
MbElement createElementAfter(int, String, Object)
MbElement createElementAsFirstChild(int)
MbElement createElementAsFirstChild(int, String, Object)
MbElement createElementAsLastChild(int)
```

```
MbElement createElementAsLastChild(int, String, Object)
MbElement createElementBefore(int)
MbElement createElementBefore(int, String, Object)
```

– You have used ESQL to create a message tree folder by using ESQL CREATE but without setting an owning parser for it. You might have used code similar or equivalent to:

```
CALL CopyMessageHeaders();
   DECLARE outRef REFERENCE TO OutputRoot;
   CREATE LASTCHILD OF outRef AS outRef NAME 'BLOB';
   CREATE LASTCHILD OF outRef NAME 'BLOB' VALUE X'01';
```

or the `outRef` reference variable was passed to an ESQL function or procedure that contained similar CREATE statements. You have not specified an owning parser by using the DOMAIN clause on the CREATE statement.

– An MRM message tree has been constructed, then only part of it has been passed, by specifying a subfolder or field, to the ASBITSTREAM function with the parser mode option set to RootBitStream. This combination is not valid, and results in a zero length BLOB.

– You have copied a message tree, or part of a message tree, to a folder and the owning parser association is not maintained.

• **Solution:** Depending on the reason for the empty message body or zero length BLOB, ensure that:

– When you create a message tree folder in a user-defined node, associate an owning parser with it. Use code similar or equivalent to:

```
createElementAfter(String parserName)
   createElementAsFirstChild(String parserName)
   createElementAsLastChild(String parserName)
   createElementBefore(String parserName)
```

– When you use ESQL CREATE to create a message tree folder, use the DOMAIN clause to associate an owning parser with the message tree, for example:

```
CALL CopyMessageHeaders();
   DECLARE outRef REFERENCE TO OutputRoot;
   CREATE LASTCHILD OF outRef AS outRef DOMAIN 'BLOB' NAME 'BLOB';
   CREATE LASTCHILD OF outRef NAME 'BLOB' VALUE X'01';
```

This code creates a BLOB folder that has the BLOB parser associated with it.

– When copying a message tree, or part of a message tree, ensure that the owning parser association is maintained, by first serializing, then reparsing the message into the appropriate message tree. An example scenario is where you have created a field:

```
SET Environment.Variables.myMsg = InputRoot.XMLNS;
```

Now you must pass it to the ASBITSTREAM function. Unless you use ESQL similar or equivalent to:

```
DECLARE xmlMsgBlob BLOB
          ASBITSTREAM(InputRoot.XMLNS, InputRoot.MQMD.Encoding, InputRoot.MQMD.CodedCharSetId);
  CREATE LASTCHILD OF Environment.Variables.myMsg DOMAIN('XMLNS')
         PARSE(xmlMsgBlob,
               InputRoot.MQMD.Encoding,
               InputRoot.MQMD.CodedCharSetId);
```

the result is a zero length bit stream.

Output message has an invalid message body indicated by error message BIP5005, BIP5016, or BIP5017

Procedure

- **Scenario:** You have unexpectedly encountered a multi-root message body or a message without any root.
- **Explanation:** This error can occur when you have created an XML message tree folder with multiple roots or no root at all by:
  - Using a user-defined node
  - Using an MQGet node
  - Using ESQL
- **Solution:** Ensure that the final output message tree has one, and only one, XML root node.

Error message BIP5651 is issued when receiving a SOAP with Attachments message from a WebSphere Application Server client

Procedure

- **Scenario:** When a WebSphere Application Server client sends a SOAP with Attachments message to the broker over JMS, error message BIP5651 is issued stating that no valid Content-Type header has been found.
- **Explanation:** When a WebSphere Application Server client sends a SOAP with Attachments message to the broker over JMS, the MIME Content-Type value appears in the MQRFH2 header and not in the MIME message body.
- **Solution:** To solve this problem, the Content-Type value needs to be copied from the MQRFH2 header to the beginning of the message as a MIME header before the message is parsed. The following ESQL adds the Content-Type value to the beginning of the WebSphere Application Server message, then invokes the MIME parser on the result.

```
create procedure parseWAS_JMS(IN InputMessage reference,IN OutputMessage reference)
/**********************************************************************
* convert a WAS/JMS message to the correct format for the MIME parser
**********************************************************************/
begin
  -- get the data as a BLOB
  declare body BLOB InputMessage.BLOB.BLOB;

  -- get the Content-Type value from the RFH2 header. Content-Type is the only
  -- header which is critical for the MIME parser, but the same approach can be
  -- used for any MIME headers which have been stored under the RFH2 header.
  declare contentType char InputMessage.MQRFH2.usr.contentType;

  -- add the contentType to the bit stream as part of an RFC822 header block
  set body = cast(('Content-Type: '||contentType) as blob ccsid 819)||x'0d0a0d0a'||body;

  -- invoke MIME parser on the modified bit stream
  CREATE LASTCHILD OF OutputMessage DOMAIN('MIME') PARSE(body);
end;
```

A message flow can take in the JMS message in the BLOB domain, and call the procedure shown here from a Compute node. The procedure can be called by using the following ESQL from a Compute node:

```
CALL CopyMessageHeaders();                -- standard procedure to copy headers
CALL parseWAS_JMS(InputRoot, OutputRoot); -- parse the 'body' as MIME
```

WebSphere Application Server produces an error when receiving a SOAP with Attachments message

Procedure

- **Scenario:** When sending a SOAP with Attachments message to a WebSphere Application Server client using JMS, an error is generated stating that the bit stream contains unexpected characters.
- **Solution:** When the broker sends a SOAP with Attachments message to WebSphere Application Server over JMS, the MIME Content-Type value must appear in the MQRFH2 header and not in the body of the message. The following procedure removes any MIME style headers from the front of the message bit stream and adds the Content-Type value to the MQRFH2 header. The supplied boundary value allows you to locate the start of the multipart MIME content.

```
create procedure writeWAS_JMS(IN OutputTree reference,IN boundary char)
/*****************************************************************************
* Serialise a MIME tree as normal, but then strip off any initial headers
* and save the Content-Type value in the RFH2 header as expected by WAS/JMS.
* Note: boundary - must be supplied with the leading hyphen pair
*****************************************************************************/
begin
  -- convert MIME subtree to BLOB
  declare body BLOB asbitstream(OutputTree.MIME);

  -- locate first occurrence of boundary and discard any data before this
  declare firstBoundary integer;
  set firstBoundary = position (cast(boundary as blob ccsid 819) in body);
  set body = substring(body from firstBoundary);

  -- save the MIME Content-Type value in the RFH2 header. Any other MIME
  -- headers which need to be preserved in the RFH2 header can be handled
  -- in the same way
  set OutputTree.MQRFH2.usr."contentType" = OutputTree.MIME."Content-Type";

  -- clear the MIME tree and create a new BLOB child for the modified body
  set OutputTree.MIME = null;
  CREATE LASTCHILD OF OutputTree DOMAIN('BLOB')PARSE(body);
end
```

Before calling this procedure, the message flow needs to be able to obtain the value of the boundary. This value might be available only within a Content-type header. The following procedure allows you to extract the Boundary value:

```
create procedure getBoundary(IN ct reference,OUT boundary char)
/*****************************************************************
* return value of the boundary parameter from a Content-Type value
*****************************************************************/
begin
  declare boundaryStart  integer;
  declare boundaryEnd    integer;

  set boundaryStart  = position('boundary=' in ct) + 9;
  set boundaryEnd    = position(';' in ct from boundaryStart);
  if (boundaryStart <> 0) then
    if (boundaryEnd <> 0) then
     set boundary  = substring(ct from boundaryStart for boundaryEnd-boundaryStart);
    else
      set boundary  = substring(ct from boundaryStart);
    end if;
  end if;
end;
```

A Compute node can invoke these procedures for sending a MIME message using the following ESQL:

```
SET OutputRoot = InputRoot;

declare boundary char;
CALL getBoundary(OutputRoot.Properties.ContentType, boundary);

CALL writeWAS_JMS(OutputRoot,boundary);
```

java_lang_StackOverflowError on AIX when processing a message flow that contains Java nodes and uses Java 5

Procedure

- **Scenario:** You get an abend on AIX when processing a message flow that contains Java nodes and uses Java 5. The abend file shows that there was an abend which indicates a segmentation fault, but a check of the `stderr` file shows a stack overflow in the JVM:

```
Exception in thread "Thread-15" java/lang/StackOverflowError: operating system stack overflow
        at com/ibm/broker/plugin/MbOutputTerminal._propagate    (Native Method)
        at com/ibm/broker/plugin/MbOutputTerminal.propagate    (MbOutputTerminal.java:103)
        at com/ibm/xsl/mqsi/XMLTransformNode.evaluate    (XMLTransformNode.java:1002)
        at com/ibm/broker/plugin/MbNode.evaluate (MbNode.java:1434)
        at com/ibm/broker/plugin/MbOutputTerminal._propagate   (Native Method)
        at com/ibm/broker/plugin/MbOutputTerminal.propagate   (MbOutputTerminal.java:103)
        at com/ibm/xsl/mqsi/XMLTransformNode.evaluate   (XMLTransformNode.java:1002)
        at com/ibm/broker/plugin/MbNode.evaluate   (MbNode.java:1434)
        at com/ibm/broker/plugin/MbOutputTerminal._propagate   (Native Method)
        at com/ibm/broker/plugin/MbOutputTerminal.propagate   (MbOutputTerminal.java:103)
        at com/ibm/xsl/mqsi/XMLTransformNode.evaluate   (XMLTransformNode.java:1002)
        at com/ibm/broker/plugin/MbNode.evaluate (MbNode.java:1434)
```

- **Explanation:**Java 5 has a parameter to adjust the stack size for Java threads. The default operating system stack size for Java 5 is only 256 KB. In certain message flows (for example, flows that contain Java user-defined nodes or XMLT nodes) this size might not be sufficient, and so you see a stack overflow error in the `stderr` file. Use the JVM option `-Xmso` to adjust the operating system stack for Java. You can display information about the stack by using the following command:

```
export MQSIJVERBOSE=-verbose:stack,sizes
```

  This command creates in the `stderr` file on startup an entry that has the folllowing content, or similar:

```
  -Xmca32K        RAM class segment increment
  -Xmco128K       ROM class segment increment
  -Xmns0K         initial new space size
  -Xmnx0K         maximum new space size
  -Xms125000K     initial memory size
  -Xmos125000K    initial old space size
  -Xmox250000K    maximum old space size
  -Xmx250000K     memory maximum
  -Xmr16K         remembered set size
  -Xlp0K          large page size
                  available large page sizes: 4K 16M
  -Xmso256K       OS thread stack size
  -Xiss2K         java thread stack initial size
  -Xssi16K        java thread stack increment
  -Xss256K        java thread stack maximum size
  -Xscmx16M       shared class cache size
```

  Note: The stack size defaults to 256K.

- **Solution:**

  1. Issue the following command to set the operating system stack size for Java to 2 MB:

     ```
     export IBM_JAVA_OPTIONS=-Xmso2m
     ```

2. Restart the broker.

Problems when using code page translation on HP-UX Procedure

- **Scenario:** You experience code page translation problems on HP-UX.
- **Solution:** Check the WebSphere MQ queue manager attribute *CodedCharSetID*. The default value for this attribute is 1051. Change this value to 819 for queue managers that host WebSphere Message Broker components.

Related concepts:

"XML parsers and domains" on page 1084
You can use XML domains to parse and write messages that conform to the W3C XML standard.

"MIME messages" on page 1120
A MIME message consists of both data and metadata. MIME metadata consists of HTTP-style headers and MIME boundary delimiters.

"MIME tree details" on page 1123
A MIME message is represented in the broker as a logical tree. When it writes a message, the MIME parser creates a message bit stream by using the logical message tree.

Related tasks:

"Configuring Custom Wire Format (CWF) properties: Message sets" on page 2849
Configure the Custom Wire Format (CWF) properties of a message set using the Message Set editor.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"MQGet node" on page 4578
Use the MQGet node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and the MQI and AMI application programming interfaces.

"TimeoutControl node" on page 4932
Use the TimeoutControl node to process an input message that contains a timeout request.

"TimeoutNotification node" on page 4936
Use the TimeoutNotification node to manage timeout-dependent message flows.

"User-defined nodes" on page 6415
You can define your own nodes to use in WebSphere Message Broker message flows.

Resolving problems when you use the WebSphere Message Broker Toolkit

Use the advice given here to help you to resolve common problems that might occur you use the WebSphere Message Broker Toolkit.

About this task

Resolving problems that occur when connecting:

- "Your broker is not recognized by the WebSphere Message Broker Toolkit" on page 3482
- "You cannot connect to the broker from the WebSphere Message Broker Toolkit or command line" on page 3482

- "You cannot connect to a migrated broker" on page 3483
- "Connection to the broker is slow" on page 3483

Resolving error messages that occur when using the WebSphere Message Broker Toolkit:

- "Errors occur while you are using the WebSphere Message Broker Toolkit" on page 3484
- "An error is issued when you access the help system" on page 3484
- "The WebSphere Message Broker Toolkit displays an error message after the error has been fixed" on page 3485
- "The message `Unable to create part: filename.extension` is issued" on page 3485

Resolving problems relating to the appearance of the workbench:

- "A view is missing from the workbench" on page 3486
- "You cannot close a message dialog box" on page 3486
- "You want to filter entries in the Problems view" on page 3486
- "Your message flow and message set projects have changed their appearance" on page 3486

Resolving non-specific problems when using the workbench:

- "Main menu entries missing on Linux on x86" on page 3487
- "Editors do not update automatically when the same file is open in multiple windows" on page 3487
- "Deleting or closing a project takes a long time" on page 3487
- "You are experiencing poor performance when working with large or complex projects" on page 3488
- "You do not know how to return to the welcome page" on page 3488

Related concepts:

"WebSphere Message Broker Toolkit perspectives" on page 34
A perspective is a group of views and editors that shows various aspects of the resources in the WebSphere Message Broker Toolkit.

"Editors" on page 35
An editor is a component of the WebSphere Message Broker Toolkit. Editors are typically used to edit or browse resources, which are the files, folders, and projects that exist in the workbench.

Related tasks:

"Changing WebSphere Message Broker Toolkit preferences" on page 571
The WebSphere Message Broker Toolkit has a large number of preferences that you can change to suit your requirements. Some of these are specific to the product plug-ins that you have installed within the workbench, including those for WebSphere Message Broker. Others control more general options, such as the colors and fonts in which information is displayed.

"Viewing the Eclipse error log" on page 3532
The Eclipse error log captures internal errors that are caused by the operating system or your code.

"Using trace" on page 3533
You can use different types of trace to help you with problem determination and troubleshooting.

Related reference:

"Resource types in the WebSphere Message Broker Toolkit" on page 6821
Links to information on projects and resource files.

"Broker Application Development perspective" on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

"WebSphere Message Broker logs" on page 6867
WebSphere Message Broker writes information to a number of product-specific logs to report the results of actions that you take.

Resolving problems that occur when connecting the WebSphere Message Broker Toolkit and a broker

Use the advice given here to help you to resolve common problems that can arise when you connect to a broker.

Your broker is not recognized by the WebSphere Message Broker Toolkit: Procedure

- **Scenario:** Your broker is not recognized by the WebSphere Message Broker Toolkit.
- **Explanation:** Broker names in the workbench are case sensitive.
- **Solution:** When you identify a broker in the WebSphere Message Broker Toolkit, make sure that you use the same case that was used when it was created. On some operating systems, the name of the broker might be folded to uppercase when it was created, even though you entered its name in lowercase.

You cannot connect to the broker from the WebSphere Message Broker Toolkit or command line:
Procedure

- **Scenario:** Error messages are issued when you try to connect to the broker from the WebSphere Message Broker Toolkit or the command line.
- **Solution:** The following table shows the checks to carry out when an error message is issued in the WebSphere Message Broker Toolkit, or returned to the command line, when you try to connect to the broker:

Error message		Actions
WebSphere Message Broker Toolkit	**Command line**	
BIP0914E	BIP1046E with WebSphere MQ reason code 2059	Check that the broker queue manager and listener are running, and that the correct port is specified for the queue manager. (The WebSphere MQ documentation describes the WebSphere MQ return codes.)
BIP0889E	BIP1047E	Check that the broker is running. Check the system event log to ensure that the broker is available for use, and correct all errors that are shown.
BIP0915E	BIP1046E with WebSphere MQ reason code 2035	Check that the WebSphere Message Broker Toolkit local user ID is created on the broker system. If it is not, create it. Check if the WebSphere Message Broker Toolkit local user ID that is created on the broker system is authorized to connect to the broker queue manager. • Use the WebSphere MQ **dspmqaut** command to determine what WebSphere MQ authorities the user has. • If necessary, add the WebSphere Message Broker Toolkit local user ID to the mqm group on the broker system.

You cannot connect to a migrated broker:
Procedure

- **Scenario:** You cannot connect from the WebSphere Message Broker Toolkit, the WebSphere Message Broker Toolkit, or a WebSphere Message Broker Toolkit application to a broker that you have migrated to Version 7.0.
- **Explanation:** The connection is failing because the WebSphere MQ channel does not exist. When you migrate a broker to Version 7.0, the default server connection channel SYSTEM.BKR.CONFIG is created if it does not exist. However, if the broker shared a queue manager with a Configuration Manager, this channel is deleted when you delete the Configuration Manager.
- **Solution:** Re-create the server connection channel SYSTEM.BKR.CONFIG on the queue manager.

You cannot connect to a remote broker:
Procedure

- **Scenario:** You have correctly configured your WebSphere Message Broker Toolkit to connect to a remote broker, but attempting to connect to the remote broker by using the WebSphere Message Broker Toolkit results in error MQRC 2059. You can connect to your remote broker from the same computer without error by using the WebSphere Message Broker Explorer, or by completing the following steps:
  1. Set the following environment variable:
     - On Windows: `set MQSERVER=ChannelName/TCP/server-address(port)`
     - On Linux: `export MQSERVER=ChannelName/TCP/'server-address(port)'`
  2. Open a WebSphere Message Broker command prompt, and enter the following test command:

     `amqsputc` *`queueName qmgrName`*
- **Explanation:** The network configuration of your WebSphere Message Broker Toolkit installation is incorrect.
- **Solution:**
  1. In the WebSphere Message Broker Toolkit, click **Window** > **Preferences** > **General** > **Network Connection**. The Preferences window opens, and the Network Connections preferences are displayed.
  2. From the Active Provider list, select **Direct connection to the internet**. This is the default setting. You can now connect to your remote broker by using the WebSphere Message Broker Toolkit.

Connection to the broker is slow:
Procedure

- **Scenario:** Connection to the broker is slow and you cannot complete other actions in the WebSphere Message Broker Toolkit when you request the following operations:
  - Creating a broker
  - Creating an execution group
  - Deploying a broker archive file, using the Deploy a BAR File wizard
  - Opening the Administration log editor
- **Solution:** Click **Cancel** on the progress monitor dialog box to cancel the in-progress connection. Connection to the broker is canceled, so that you can perform other actions in the WebSphere Message Broker Toolkit. You can then resubmit the canceled operation.

Related concepts:

"WebSphere Message Broker Toolkit perspectives" on page 34
A perspective is a group of views and editors that shows various aspects of the resources in the WebSphere Message Broker Toolkit.

"Editors" on page 35
An editor is a component of the WebSphere Message Broker Toolkit. Editors are typically used to edit or browse resources, which are the files, folders, and projects that exist in the workbench.

Related tasks:

"Resolving problems when you use the WebSphere Message Broker Toolkit" on page 3480
Use the advice given here to help you to resolve common problems that might occur you use the WebSphere Message Broker Toolkit.

"Changing WebSphere Message Broker Toolkit preferences" on page 571
The WebSphere Message Broker Toolkit has a large number of preferences that you can change to suit your requirements. Some of these are specific to the product plug-ins that you have installed within the workbench, including those for WebSphere Message Broker. Others control more general options, such as the colors and fonts in which information is displayed.

"Viewing the Eclipse error log" on page 3532
The Eclipse error log captures internal errors that are caused by the operating system or your code.

"Using trace" on page 3533
You can use different types of trace to help you with problem determination and troubleshooting.

Related reference:

"Resource types in the WebSphere Message Broker Toolkit" on page 6821
Links to information on projects and resource files.

"Broker Application Development perspective" on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

"WebSphere Message Broker logs" on page 6867
WebSphere Message Broker writes information to a number of product-specific logs to report the results of actions that you take.

Resolving error messages that occur when using the WebSphere Message Broker Toolkit

Use the advice given here to help you to resolve common problems that can occur when you use the WebSphere Message Broker Toolkit.

Errors occur while you are using the WebSphere Message Broker Toolkit: Procedure

- **Scenario:** An error has occurred while you are using the WebSphere Message Broker Toolkit, and you want information to help you to diagnose the problem.
- **Solution:** Some errors that occur in the WebSphere Message Broker Toolkit are logged in the `.log` file in your `workspace\.metadata` directory. You can view this log by switching to the Plug-in Development perspective, and clicking the Error Log tab in the lower-right pane. The log shows in which plug-in the error occurred, and gives further information.

  You can also use trace to try and determine the cause of your problem. See "Using trace" on page 3533 for more information about tracing.

An error is issued when you access the help system:

Procedure

- **Scenario:** You are accessing the help system through the WebSphere Message Broker Toolkit, and an error message is issued stating that the Web page that you requested is not available offline.
- **Explanation:** This error might occur if you previously had a connection to a Web-based version of the help system and lost it, or if you have **Work Offline** selected in your Internet Explorer options.
- **Solution:** Click **Connect** to load the help system.

The WebSphere Message Broker Toolkit displays an error message after the error has been fixed:
Procedure

- **Scenario:** The WebSphere Message Broker Toolkit is in an inconsistent state, or displays an error message after the error has been fixed
- **Solution:** Clean the workspace:
  1. From the WebSphere Message Broker Toolkit, click **Project** > **Clean**.
  2. Click **Clean all projects** and **Finish**.

     This action cleans the whole workspace of any internal files, which are then re-created so that none of your data is lost.

The message Unable to create part: *filename.extension* is issued:
Procedure

- **Scenario:** You open an editor in the WebSphere Message Broker Toolkit, and an error message is raised stating that a file cannot be created. This problem is caused by one of following situations:
- **Explanation 1:** The given file is not associated with a recognized editor.
  - **Solution 1:** Right-click the file and choose the default editor to open it, or choose another editor if the default editor cannot open it.
- **Explanation 2:** The given file contains syntax errors and cannot be loaded into the chosen editor. However, if you then try to open a valid file, you get the same error message repeatedly.
  - **Solution 2:** Restart the WebSphere Message Broker Toolkit; do not load the file into an editor again until you have fixed the syntax errors.

:

Related concepts:

"WebSphere Message Broker Toolkit perspectives" on page 34
A perspective is a group of views and editors that shows various aspects of the resources in the WebSphere Message Broker Toolkit.

"Editors" on page 35
An editor is a component of the WebSphere Message Broker Toolkit. Editors are typically used to edit or browse resources, which are the files, folders, and projects that exist in the workbench.

Related tasks:

"Resolving problems when you use the WebSphere Message Broker Toolkit" on page 3480
Use the advice given here to help you to resolve common problems that might occur you use the WebSphere Message Broker Toolkit.

"Changing WebSphere Message Broker Toolkit preferences" on page 571
The WebSphere Message Broker Toolkit has a large number of preferences that you can change to suit your requirements. Some of these are specific to the product plug-ins that you have installed within the workbench, including those for WebSphere Message Broker. Others control more general options, such as the colors and fonts in which information is displayed.

"Viewing the Eclipse error log" on page 3532
The Eclipse error log captures internal errors that are caused by the operating system or your code.

"Using trace" on page 3533
You can use different types of trace to help you with problem determination and troubleshooting.

Related reference:

"Resource types in the WebSphere Message Broker Toolkit" on page 6821
Links to information on projects and resource files.

"Broker Application Development perspective" on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

"WebSphere Message Broker logs" on page 6867
WebSphere Message Broker writes information to a number of product-specific logs to report the results of actions that you take.

Resolving problems relating to the appearance of the workbench

Use the advice given here to help you to resolve common problems that relate to the appearance of the WebSphere Message Broker Toolkit.

A view is missing from the workbench:
Procedure

- **Scenario:** A view you are expecting to be displayed in the workbench is not visible, or you have closed a view.
- **Explanation:** The perspective has changed since it was first created, and the default views are no longer visible.
- **Solution:** To restore the default views in a perspective you can reset the perspective. Click **Window** > **Reset perspective**.

You cannot close a message dialog box:
Procedure

- **Scenario:** You have opened a menu, and a message dialog box tells you that previous changes made have been processed successfully by the broker. You cannot close the dialog.
- **Solution:** If you cannot close the dialog box by clicking **OK**, press Esc.

You want to filter entries in the Problems view:
Procedure

- **Scenario:** The Problems view has many entries, and it is difficult to find the entry that you want.
- **Solution:** At the top of the Problems View, click the icon with the arrows pointing to the right. This action opens a dialog box, in which you can tailor your selections to display only a subset of the entries in the view. For example, you can select only those entries for the project that you have selected.

Your message flow and message set projects have changed their appearance:

Procedure

- **Scenario:** You have created a new simple project in the WebSphere Message Broker Toolkit and now all your message flow and message set projects look different.
- **Explanation:** When you create a new simple project, the WebSphere Message Broker Toolkit switches automatically to the Broker Application Development perspective.
- **Solution:** To return to the previous view, switch to the perspective with which you were working..

Resolving non-specific problems when using the WebSphere Message Broker Toolkit

Use the advice given here to help you to resolve some common problems that can occur when you use the WebSphere Message Broker Toolkit that are not dealt with in previous categories.

Main menu entries missing on Linux on x86:
Procedure

- **Scenario:** The main menu on Linux on x86 no longer contains items that relate to the WebSphere Message Broker Toolkit.
- **Explanation:** If you have installed the WebSphere Message Broker Toolkit in more than one package group, and have now removed one of those installations, a known restriction causes the main menu items to be removed when the component is uninstalled.
- **Solution:** Invoke the WebSphere Message Broker Toolkit from the command line. Navigate to the installation directory for the package group in which you have installed the WebSphere Message Broker Toolkit, and enter the following command:

  ```
  ./eclipse -product com.ibm.etools.msgbroker.tooling.ide
  ```

Editors do not update automatically when the same file is open in multiple windows:
Procedure

- **Scenario:** You are working in the Broker Application Development perspective, and are using the associated editor to work with one or more resources; for example, you are editing a message flow in the message flow editor or an ESQL module in the ESQL editor. You have clicked **Window** > **New Window** to create a second Eclipse view, and have opened the same resource in the second window. Changes that you make to the resource in the first editor window are not reflected in the second editor window.
- **Explanation:** The WebSphere Message Broker Toolkit editors do not automatically update multiple windows in which you have opened the same resource.
- **Solution:** Save the contents of the resource file in the first editor window, then close and reopen additional windows. The reopened windows reflect the updated content.

Deleting or closing a project takes a long time:
Procedure

- **Scenario:** Deleting or closing a project to save memory takes a long time.

- **Explanation:** If a project is referenced by other projects, removing that project requires all the other projects, and the projects that refer to them recursively, to be built fully. This process occurs to keep the content-assist and validation models current.
- **Solution:** To keep a project open in the workspace requires very little memory, therefore you do not need to close or delete projects.

You are experiencing poor performance when working with large or complex projects:
Procedure

- **Scenario:** You are experiencing poor performance in the WebSphere Message Broker Toolkit when working with large or complex projects.
- **Explanation:** Frequent project changes, such as adding and removing projects, or using **Project** > **Clean**, use large amounts of memory because of the size and number of files and the connections between them.
- **Solution:** Increase your system memory.

You do not know how to return to the welcome page:
Procedure

- **Scenario:** You do not know how to return to the welcome page that was displayed in the WebSphere Message Broker Toolkit when you first started using it.
- **Solution:** To open the welcome page:
  1. From the **Help** menu, select **Welcome**. If only one welcome page is available, it is displayed. If more than one is available, a list is displayed.
  2. Select the welcome page that you want, for example, WebSphere Message Broker WebSphere Message Broker Toolkit.
  3. Click **OK**.

:

Related concepts:

"WebSphere Message Broker Toolkit perspectives" on page 34
A perspective is a group of views and editors that shows various aspects of the resources in the WebSphere Message Broker Toolkit.

"Editors" on page 35
An editor is a component of the WebSphere Message Broker Toolkit. Editors are typically used to edit or browse resources, which are the files, folders, and projects that exist in the workbench.

Related tasks:

"Resolving problems when you use the WebSphere Message Broker Toolkit" on page 3480
Use the advice given here to help you to resolve common problems that might occur you use the WebSphere Message Broker Toolkit.

"Changing WebSphere Message Broker Toolkit preferences" on page 571
The WebSphere Message Broker Toolkit has a large number of preferences that you can change to suit your requirements. Some of these are specific to the product plug-ins that you have installed within the workbench, including those for WebSphere Message Broker. Others control more general options, such as the colors and fonts in which information is displayed.

"Viewing the Eclipse error log" on page 3532
The Eclipse error log captures internal errors that are caused by the operating system or your code.

"Using trace" on page 3533
You can use different types of trace to help you with problem determination and troubleshooting.

Related reference:

"Resource types in the WebSphere Message Broker Toolkit" on page 6821
Links to information on projects and resource files.

"Broker Application Development perspective" on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

"WebSphere Message Broker logs" on page 6867
WebSphere Message Broker writes information to a number of product-specific logs to report the results of actions that you take.

Resolving problems when using the WebSphere Message Broker Explorer

Work through the advice provided to help you to deal with problems that can arise when you are using the WebSphere Message Broker Explorer.

About this task

- "Tracing problems with accounting and statistics"
- "The Brokers folder is missing"
- "You cannot open the information center on Red Hat Enterprise Linux" on page 3490
- "Accounting and statistics do not work for a remote broker" on page 3490
- "SYSTEM.DEF.SVRCONN error" on page 3490

Tracing problems with accounting and statistics

Procedure

- **Scenario:** When you view accounting and statistics data using the WebSphere Message Broker Explorer you encounter a problem, and you want to collect trace.
- **Explanation:** The WebSphere Message Broker Explorer uses the Java Message Service (JMS) to retrieve the accounting and statistics data from the broker. To trace problems with this connection, you must turn on JMS tracing in WebSphere MQ.
- **Solution:** See the WebSphere MQ documentation for information about how to turn on JMS tracing. You can find this information in the following part of the information center: **Using Java** > **Using WebSphere MQ classes for JMS** > **Solving problems** > **Tracing programs**.

The Brokers folder is missing

Procedure

- **Scenario:** You have opened the WebSphere MQ Explorer to work with brokers and broker resources, but you cannot see the Brokers folder.
- **Explanation:** The WebSphere Message Broker Explorer is an extension to the WebSphere MQ Explorer, and provides the Brokers and Broker Archive Files folders, and related actions. You must install the WebSphere Message Broker Explorer component on all computers on which you want to use the WebSphere MQ Explorer to manage WebSphere Message Broker resources, including brokers, execution groups and BAR files.
- **Solution:** Close your MQ Explorer session, follow the instructions to install the WebSphere Message Broker Explorer, then start the MQ Explorer again.

You cannot open the information center on Red Hat Enterprise Linux

Procedure

- **Scenario:** You are using the WebSphere Message Broker Explorer on Red Hat Enterprise Linux Server 5.2 or Red Hat Enterprise Linux Server 5.3 but you cannot open the information center from inside WebSphere Message Broker Explorer.
- **Solution:** Set the environment variable MOZILLA_FIVE_PATH then run the **strmqcfg** command. For more information, see the question about running the SWT browser inside Eclipse at: Eclipse SWT FAQ

Accounting and statistics do not work for a remote broker

Procedure

- **Scenario:** You try to view broker statistics, but instead see a message box:

  ```
  Could not connect to broker brokerName.
  Is MQ configured for JMS?
  ```

- **Explanation:** Java Message Service (JMS) has not been correctly configured. Either JMS has not been enabled for the queue manager, or you do not have access to the channel that the broker uses for JMS activity. Check the log for more details. Note: In WebSphere MQ Version 7.0, 7.0.1, 7.1 and 7.5 onwards JMS is automatically configured, so the problem could be due to the security settings on the publish/subscribe queues. Verify that WebSphere Message Broker Explorer can communicate with the queue manager's publish/subscribe queues. For more information on WebSphere MQ publish/subscribe security, see IBM Education Assistant module: WebSphere MQ V7 Publish/subscribe security.
- **Solution:** If the log shows that JMS has not been correctly set up, take the following steps:

  1. In a WebSphere Message Broker command console, change to the *MBExplorerInstallLocation*\eclipse\plugins\com.ibm.etools.wmadmin.broker.stats_*releaseNumber* directory.
  2. Run the following command:

     ```
     runmqsc queuemanager < setup_jms.mqsc
     ```

  If the log shows that you do not have permission to access the channel:

  1. Check which channel is being used by viewing the Statistics panel for the broker in WebSphere Message Broker Explorer. The channel is normally SYSTEM.DEF.SVRCONN.
  2. In a WebSphere Message Broker command console, issue the following command to allow any user on your broker machine, who is a member of the mqm and mqbrkrs groups, to run with the permissions of the broker user:

  ```
  alter chl (SYSTEM.DEF.SVRCONN) CHLTYPE(SVRCONN) MCAUSER(brokerUser)
  ```

  Note: Setting SYSTEM.DEF.SVRCONN MCAUSER(*brokerUser*) when the broker user is a member of the mqm group requires security protocols, such as Transport Layer Security (TLS), Secure Sockets Layer (SSL), channel authentication security, or a security exit, to be set up. For more information on security, see "Security overview" on page 351.

SYSTEM.DEF.SVRCONN error:

Procedure

- **Scenario:** When you try to connect to a broker to view accounting and statistical information, an error message reports that WebSphere Message Broker Explorer cannot connect to SYSTEM.DEF.SVRCONN.

- **Explanation:** This error might indicate that the SHARECNV parameter on the SVRCONN channel of the queue manager is not compatible with the current WebSphere MQ level of JMS.
- **Solution:**
  1. For more information about the error, turn on service trace; see "Changing trace settings from the WebSphere Message Broker Explorer" on page 3549.
  2. In WebSphere MQ Explorer, set **Sharing Conversations** (SHARECNV) to at least 1.

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

Related tasks:

"Installing WebSphere Message Broker Explorer" on page 280
To use WebSphere Message Broker Explorer only, without installing the complete WebSphere Message Broker Toolkit, use the WebSphere Message Broker Explorer installation wizard to install the WebSphere Message Broker Explorer.

"Configuring brokers in the WebSphere Message Broker Explorer" on page 635
Configure your local and remote brokers by using the WebSphere Message Broker Explorer.

"Viewing message flow accounting and statistics data" on page 3300
You can use the Broker Statistics and Broker Statistics Graph views in the WebSphere Message Broker Explorer to view snapshot accounting and statistics data as it is produced by the broker.

Related information:

WebSphere MQ Version 7 Information Center online

Resolving problems when using databases

Use the advice given here to help you to resolve problems that can arise when using databases.

Before you begin

Before you start:

- Read "Is there a problem with a database?" on page 3356

Procedure

DB2 error message SQL0443N is issued

Procedure

- **Scenario:** After you upgrade your DB2 server to a new fix pack level, a DB2 error message SQL0443N is issued if you invoke a DB2 Call Level Interface (CLI) catalog function, such as SQLTables(), SQLColumns(), or SQLStatistics(). An example of the error message is:

  ```
  SQL0443N Routine "SYSIBM.SQLTABLES" (specific name "TABLES") has returned
  an error SQLSTATE with diagnostic text SYSIBM:CLI:-805". SQLSTATE=38553.
  ```

- **Solution:** Bind the `db2schema.bnd` file against each database by entering the following commands at a command prompt:

  ```
  db2 terminate
  db2 connect to database-name
  db2 bind path\db2schema.bnd blocking all grant public sqlerror continue
  db2 terminate
  ```

  where *database-name* is the name of the database to which the utilities must be bound, and *path* is the full path name of the directory where the bind files are located. For example, the default location on Windows is `C:\Program Files\IBM\SQLLIB\bnd\`(on Windows 32-bit editions) or `C:\Program Files(x86)\IBM\SQLLIB\bnd\` (on Windows 64-bit editions).

  To list all the names of databases for a particular DB2 instance, run the DB2 CLI command **`db2 list database directory`**. For further information, see the DB2 documentation.

DB2 error message SQL0805N is issued

Procedure

- **Scenario:** When a message flow that includes a Database node runs, SQL error `SQL0805N NULLID.SQLLF000` is issued.
- **Solution:** Open a DB2 Command Line Processor window and issue a bind command to the database.

  Linux UNIX On Linux and UNIX systems, enter the commands:

```
connect to db
bind ~/sqllib/bnd/@db2cli.lst grant public CLIPKG 5
connect reset
```

where *db* is the database name.

Windows On Windows systems, enter the commands:

```
connect to db
bind x:\sqllib\bnd\@db2cli.lst blocking all grant public
connect reset
```

where *x:* identifies the drive onto which you installed DB2, and *db* is the database name.

DB2 error message SQL0998N is issued on Linux

Procedure

- **Scenario:** You are trying to use a globally coordinated message flow with DB2 on Linux and error message SQL0998N is issued with Reason Code 09 and Subcode " ".
- **Solution:** Check that the `LD_ASSUME_KERNEL` environment variable is not set. If it is set, use the **`unset`** command to remove it from your environment and ensure that you modify your profile scripts so that it remains unset.

DB2 error message SQL0998N or SQL1248N is issued

Procedure

- **Scenario:** When you try to use a globally coordinated message flow with a DB2 database, you get one of the following error messages:
  - SQL0998N with Reason Code 09 and Subcode ""
  - SQL1248N with a message indicating that the database is not defined with the transaction manager
- **Solution:** Use the instructions in “Configuring global coordination with DB2” on page 700 to configure the database and XAResourceManager stanza.

DB2 error message SQL1040N is issued

Procedure

- **Scenario:** You are using a DB2 database, and error message BIP2322 is issued with error SQL1040N.
- **Explanation:** The following DB2 message indicates that the value of the DB2 database configuration parameter **`maxappls`** has been reached:

```
"SQL1040N The maximum number of applications is already connected to the database.
SQLSTATE=57030"
```

  DB2 has rejected the attempt to connect.
- **Solution:**
  1. Stop all brokers that connect to the affected database.
  2. Increase the value of the **`maxappls`** configuration parameter. Also, check the value of the associated parameter **`maxagents`**, and increase it in line with **`maxappls`**.
  3. Restart the DB2 database.
  4. Restart the brokers.

DB2 error message SQL1224N is issued when you connect to DB2

Procedure

- **Scenario:** DB2 error message SQL1224N is issued when you connect to a DB2 database. This error indicates that a database agent could not be started, or was ended because of a database shutdown or force command.
- **Solution:** On AIX, use TCP/IP to connect to DB2 databases, to avoid the shared memory limit of 10 connections. To set up AIX and DB2 loop-back to use a TCP/IP connection:
  1. Configure DB2 to use TCP/IP, and to start the TCP/IP listener. On the database server machine, log in as the DB2 instance owner, typically `db2inst1`, and issue the following commands:
     ```
     db2set DB2COMM=tcpip
     db2stop
     db2start
     ```
  2. If the DB2 connection port is not defined in `/etc/services`, edit the services file to add the DB2 connection and interrupt ports. You must use unique names, and port numbers that are not already defined in the services file; for example:
     ```
     db2svc1     3700/tcp          # DB2 Connection Service
     db2isvc1    3701/tcp          # DB2 Interrupt Service
     ```
  3. Update the DB2 configuration; for example:
     ```
     db2 update dbm cfg using svcename db2svc1
     ```

     where *db2svc1* is the name of the DB2 Connection port service in `/etc/services`.

     Alternatively, you can specify a port number directly.
  4. Stop and restart the database by using the following commands:
     ```
     db2stop
     db2start
     ```
  5. Catalog a new TCP/IP connection node:
     ```
     db2 catalog tcpip node NODENAME remote HOSTNAME server db2svc1
     ```

     where:

     NODENAME
     : is the name of the new TCP/IP connection node. You can use `local` as your node name, if it is a unique identifier.

     HOSTNAME
     : is the name of your computer.

     db2svc1
     : is the name of the DB2 connection port service in `/etc/services`.

     Message DB20000I is displayed when the command completes successfully.
  6. Catalog the database with a new alias name; for example:
     ```
     db2 catalog database DATABASE as DBALIAS at node NODENAME
     ```

     where:

     DATABASE
     : is the physical name of the database.

     DBALIAS
     : is the database alias name that you want to use.

Specify the new alias name in all subsequent references to the local database.

7. Stop and start DB2:

```
db2 terminate
db2stop
db2start
```

8. Log on with the user ID under which the broker is running.
9. Update the ODBC configuration file for each broker to add definitions for the database:
   a. At the top of the file, add a definition for the database alias name:

```
DBALIAS=IBM DB2 ODBC Driver
```

   b. Add a new stanza for the database alias:

```
 [DBALIAS]
Driver=INSTHOME/sqllib/lib/libdb2.a
Description=Database Alias
Database=DBALIAS
```

   where *INSTHOME* is the path to your DB2 Instance directory.
10. Update your message flows to specify the alias database name, redeploy the BAR file to the broker, and test the flows.

DB2 or ODBC error messages are issued on z/OS

Procedure

- **Scenario:** DB2 or ODBC messages are issued on z/OS indicating one or more of the following errors:
  - An exception was caught while issuing the database SQL connect command.
  - A database error occurred with an ODBC return code of -1, an SQL state of 58004 and a native error code of -99999.
- **Solution:** If an ODBC message is displayed:
  1. Turn ODBC application tracing on to produce the `traceodbc` file.
  2. Locate the `traceodbc` file, which is written to the `/output` subdirectory. For example, the full path might be `/u/argo/VCP0BRK/output/traceodbc`.
  3. Go to the bottom of this file and search for previous instances of SQLerror.

  Common DB2 problems include:
  - ODBC return code -1, SQL state 58004, Native error code -99999

    These codes might be returned for the following reasons:
    - No SQL code. The DB2 subsystem is not started
    - RRS is not started.
  - SQLCODE 922.

    The user ID of the started task is not authorized to use plan DSNACLI.
  - ODBC return code -1, SLQ state 42503, Native error code -553

    These codes might be returned if the user ID of the started task is not authorized to use the current SQL ID. Reconfigure the broker and specify `DB2_TABLE_NAME` as a valid name, or create a RACF group, and connect the started task user ID to this group.

You do not know how many database connections a broker requires

Procedure

- **Scenario:** You do not know how many database connections to set up for your broker.

- **Solution:** Determine the number of database connections required by a broker for capacity and resource planning. On DB2, the default action taken is to limit the number of concurrent connections to a database to the value of the **maxappls** configuration parameter; the default value for **maxappls** is 40. The associated parameter **maxagents** also affects the current connections.

  The connection requirements for a single broker are:

  – Five are required by internal broker threads.
  – One is required for each message flow thread that parses MRM messages.
  – One is required for each database access node to separate ODBC data source names for each message flow thread (that is, if the same DSN is used by a different node, the same connection is used).

You want to use XA with DB2 databases

Procedure

- **Scenario:** You want to use XA with one or more DB2 databases.
- **Solution:** Ensure that your queue manager is configured to use **ThreadOfControl=THREAD**.

  – Linux UNIX On Linux and UNIX systems, configure this parameter in the XAResourceManager stanza in the file `qm.ini` for the broker queue manager.

  – Windows On Windows systems, configure this parameter in WebSphere MQ Explorer.

XA coordination fails if the database restarts while the broker is running

Procedure

- **Scenario:** XA global coordination fails, and you get an error like the following example, which is from a DB2 user database:

  ```
  Database error: SQL State '40003'; Native Error Code '-900';  Error Text '[IBM]
  [CLI Driver] SQL0900N  The application state  is in error. A database connection
  does not exist.SQLSTATE=08003'.
  ```

- **Explanation:** A globally coordinated message flow cannot automatically reconnect to a user database if the user database is restarted while the broker is still running.
- **Solution:** Stop and restart the broker if the user database goes down, or is brought down for a scheduled maintenance.

Error message BIP2322 is issued when you access DB2 on z/OS

Procedure

- **Scenario:** You are running a message flow in which a node attempts to access a table on a DB2 data-sharing group. If ODBC tracing is turned on, an error message is written to the `traceodbc` file:

```
SQLError( hEnv=0, hDbc=0, hStmt=1, pszSqlState=&302f8ecc, pfNativeError=&302f8ec8,
pszErrorMsg=&28f6a6d0, cbErrorMsgMax=1024, pcbErrorMsg=&302f8eb4 )
SQLError( pszSqlState="51002", pfNativeError=-805, pszErrorMsg="{DB2 for OS/390}
{ODBC Driver}{DSN06011}
 DSNT408I SQLCODE = -805, ERROR:  DBRM OR PACKAGE NAME DSN610GH..DSNCLICS.16877-
          BE5086005F4 NOT FOUND IN PLAN DSNACLI. REASON 02
 DSNT418I SQLSTATE   = 51002 SQLSTATE RETURN CODE
 DSNT415I SQLERRP    = DSNXEPM SQL PROCEDURE DETECTING ERROR
 DSNT416I SQLERRD    = -350  0  0  -1  0  0 SQL DIAGNOSTIC INFORMATION
 DSNT416I SQLERRD    = X'FFFFFEA2'  X'00000000'  X'00000000'  X'FFFFFFFF'
          X'00000000'  X'00000000' SQL DIAGNOSTIC INFORMATION
```

This error is accompanied by the following BIP2322 error message in the syslog:

```
BIP2322E: DATABASE ERROR: SQL STATE '51002'; NATIVE ERROR CODE '-805'
```

- **Explanation:** This error occurs when the DSNACLI plan has not been bound in the correct way.
- **Solution:** Ensure that the DSNACLI plan is bound correctly. See "Binding a DB2 plan to use data-sharing groups on z/OS" on page 3990 for information about how to complete this task.

Error message BIP2322 IM004 is issued when you connect to an Informix database

Procedure

- **Scenario:** You are using the **mqsicvp** command to connect to an Informix database, and you see the following error message.

  ```
  BIP2322E: Database error: SQL State 'IM004'; Native Error Code '0'; Error Text '[DataDirect][ODB
  failed'.
  ```

- **Explanation:** This error can be caused when the environment for the database has not been initialized.
- **Solution:** Check the documentation for the client on your broker system for details of the actions that you must take. For example, you might have to specify the following environment variables:

  ```
  export INFORMIXDIR=/installation_directory_of_informix_client_software
  export PATH=${INFORMIXDIR}/bin:${PATH}
  export INFORMIXSERVER=server_name
  export INFORMIXSQLHOSTS=${INFORMIXDIR}/etc/sqlhosts
  export TERMCAP=${INFORMIXDIR}/etc/termcap
  export TERM=vt100
  export LIBPATH=${INFORMIXDIR}/lib:${INFORMIXDIR}/lib/esql:
                 ${INFORMIXDIR}/lib/cli:$LIBPATH
  ```

  where *server_name* is defined in the file `sqlhosts` (the required value is typically the machine name), and the location of the file `sqlhosts` is set up as part of the installation process.

  To configure your system to run this setup at the start of every session, add these statements to the login profile of the user that is going to run the broker.

  For more information, see "Command environment: Linux and UNIX systems" on page 310.

On Oracle, a database operation fails to return any rows, even though the rows exist

Procedure

- **Scenario:** You are using Oracle databases in your message flows, and ESQL binds against columns that are declared as data type CHAR, and those parameter markers are referenced in a WHERE clause. The database operation fails to return any rows, even though the rows exist.
- **Explanation:** Fixed-length character strings must be padded with blank characters on Oracle for this type of comparison to succeed.
- **Solution:** Define the CHAR columns as VARCHAR2 columns, or pad the ESQL variable with blank characters to the required column length, so that the comparison locates the required rows from the table.

Broker commands fail when the Oracle 10g Release 2 client runs on Linux on POWER with Red Hat Enterprise Linux Advanced Server V4.0

Procedure

- **Scenario:** Broker commands fail in an environment where an Oracle 10g Release 2 client runs on Linux on POWER with Red Hat Enterprise Linux Advanced Server V4.0. Abend files might be created, with contents like the following data:

  ```
  /opt/mqsi/lib/libCommonServices.so(.ImbAbendSignalHandler+0x10)[0x800017d3c0]
  [0x80022b33f0]
  /lib64/tls/libpthread.so.0[0x80cc2339d8]
  /lib64/tls/libpthread.so.0[0x80cc230e4c]
  /lib64/tls/libpthread.so.0[0x80cc22af54]
  bipservice[0x10005e38]
  /lib64/tls/libpthread.so.0[0x80cc22a1d8]
  /lib64/tls/libc.so.6[0x80cc0dd3e4]
  ```

  or

  ```
  /opt/mqsi/lib/libImbCmdLib.so(._ZN15ImbCreateTables15createAllTablesEv+0x38)
  [0x800014c9bc]mqsicreatebroker[0x10023324]mqsicreatebroker[0x1002c0c4]
  /opt/mqsi/lib/libImbCmdLib.so(._ZN10ImbCmdBase20processCommandStringEv+0x6c)
  [0x80001dcad4]mqsicreatebroker[0x1000d728]/lib64/tls/libc.so.6[0x80cc02423c]
  /lib64/tls/libc.so.6[0x80cc0243c4]
  ```

- **Explanation:** The Oracle installation library contains copies of `libgcc` library files. The Oracle user profile adds the directory containing these files to the environment variable LD_LIBRARY_PATH. This action causes the `libgcc` library files to be found before the system libraries, and leads to the failure of broker commands and the production of abend files.
- **Solution:** Ensure that you add `/lib64` to the environment variable LD_LIBRARY_PATH before the Oracle library path. Before you run **mqsiprofile**, include a string like:

  ```
  /lib64::/usr/lib:/oracle/app/oracle/product/10.2.0/db_1/lib
  ```

  which shows `/lib64` preceding the Oracle library directory.

Error message BIP2322 `Driver not capable` is issued when you use an Informix database

Procedure

- **Scenario:** When you try to access an Informix database from a node in a message flow, the following error message is issued:

  ```
  BIP2322E: Database error: SQL State 'HYC00'; Native Error Code '-11092';
  Error Text '[Informix][Informix ODBC Driver]Driver not capable.'.
  ```

- **Explanation:** The broker uses transaction statements, therefore the database must be created, and configured to enable logging.
- **Solution:** Consult your database administrator to ensure that transaction logging has been enabled on the Informix database that the broker is trying to access. For example, create the database with a buffered log:

  ```
  create database with [buffered] log;
  ```

Database updates are not committed as expected

Procedure

- **Scenario:** You have included a Database node, Compute node, or Filter node in a message flow, and you have set the **Transaction** property to Commit. The message flow has raised an exception and has rolled back, but the database updates have not been committed.

- **Explanation:** When you set **Transaction** to Commit, the database updates performed by the node are committed when the node completes successfully. If an exception is raised before the node has completed, and causes the message flow to be rolled back, the commit is not issued and the database updates are also rolled back. The conditions under which this situation can occur are:
  - The node itself causes an exception to be raised. The commit is never performed.
  - The ESQL contains a PROPAGATE statement. This statement does not complete until all processing along the path taken by the propagated message has completed, and control returns to the node. Only then can the commit be performed. If an exception is raised along this path, control is not returned and the database updates are rolled back as part of the message flow.
- **Solution:** Review the operation of the node that performs the database updates. For example, you might be able to split the work between two nodes, with the first updating the database, and the second propagating the output message. Consider changing the ESQL code to process the message in a different way.

You want to list the database connections that the broker holds

Procedure

- **Scenario:** You want to list the database connections that the broker holds.
- **Solution:** The broker does not have any functionality to list the connections that it has to a database. Use the facilities that your database supplies to list connections. Refer to the documentation for your database to find out how to perform this task.

The queue manager finds the XA resource manager is unavailable when configured for XA with DB2 on Windows

Procedure

- **Scenario:** You have configured a queue manager for XA coordination with DB2 on your Windows computer. When you restart the queue manager, it reports error AMQ7604 in the queue manager error log. All subsequent attempts at XA coordination between WebSphere MQ and DB2 fail.

  The error message has the following content, or similar:

  ```
  23/09/2008 15:43:54 - Process(5508.1) User(MUSR_MQADMIN) Program(amqzxma0.exe)
  AMQ7604: The XA resource manager 'DB2 MQBankDB database' was not available
  when called for xa_open.
  The queue manager is continuing without this resource manager.
  ```
- **Explanation:** The user ID that runs the WebSphere MQ Services process `amqmsrvn.exe`, which has a default value of MUSR_MQADMIN, is running with an access token that does not contain group membership information for the group DB2USERS.
- **Solution:** Check that the WebSphere MQ Services user ID is a member of the group DB2USERS, stop the WebSphere MQ service (for example, by issuing the command **net stop "IBM MQSeries"**), and all other processes that are running under the same user ID, and then restart these processes. You can restart your computer to stop and restart these processes after you have checked the user ID status, but this action is typically not required.

Error messages are received when you are trying to remove a DB2 database on Windows when you are using a sample

Procedure

- **Scenario:** You are running a sample, and you are trying to remove a DB2 database on your Windows computer, and you receive a BIP9835E error message with the error code SQLSTATE=57019.

The error message has content like the following data:

```
BIP9830I: Deleting the DB2 Database <Your database name>.
BIP9835E: The DB2 batch command failed with the error code SQLSTATE=57019.
The database <Your database name> could not be created/deleted.
The error code SQLSTATE=57019 was returned from the DB2 batch command.
```

- **Explanation:** If you use the DB2 Control Center to perform a query, a connection is opened against the database. This connection stays open until the DB2 Control Center is closed, at which point the connection is ended.
- **Solution:** Close the DB2 Control Center application, and try to run the sample again.

DB2 error message SQL7008N is issued

Procedure

- **Scenario:** You are using DB2 and encounter error SQL7008N when updating or inserting into tables on iSeries.
- **Explanation:** SQL7008N is a common error when the tables being accessed on an iSeries server are not being journaled.
- **Solution:** To correct the error, take one of the following steps:
  - Journal your tables.
  - Change the isolation level to `No Commit`. You can change the isolation level of the database alias referenced by your ODBC data source to `No Commit` by using the following DB2 command:

    ```
    db2 update cli cfg for section <db_alias> using TxnIsolation 32
    ```

SQLCODE -981 is issued when you access DB2 on z/OS

Procedure

- **Scenario:** You are running a message flow that uses ODBC database interaction. When a commit or rollback is attempted, DB2 reports an error with SQLCODE=-981 and SQLSTATE=57015. An error message is seen similar to: {DB2 FOR OS/390}{ODBC DRIVER}{DSN09015} DSNT408I SQLCODE = -981, ERROR: THE SQL STATEMENT FAILED BECAUSE THE RRSAF CONNECTION IS NOT IN A STATE THAT ALLOWS SQL OPERATIONS, REASON 00C12219
- **Explanation:** You can choose for ODBC database operations to be committed or rolled back irrespective of the success or failure of the message flow transaction as a whole. This error can be seen if you attempt to use more than one uncoordinated ODBC database connection on a single message flow thread.
- **Solution:** Only one uncoordinated ODBC database connection per thread is supported. Update your message flow to perform ODBC database operations outside of the message flow transaction on only one database. Any number of different databases are supported as part of a coordinated message flow transaction.

Related concepts:

"ESQL procedures" on page 2386
An ESQL procedure is a subroutine that has no return value. It can accept input parameters from, and return output parameters to, the caller.

Related tasks:

"Database security" on page 495
You can access databases from the message flows that you deploy to your brokers, and must therefore consider the steps you might want to take to secure that access.

"Is there a problem with a database?" on page 3356
If you have database problems, complete a set of initial checks to identify errors.

"ODBC trace" on page 3551
You can use various methods to trace for ODBC activity, depending on the operating system that you are using.

"Invoking stored procedures" on page 2503
To invoke a procedure that is stored in a database, use the ESQL CALL statement. The stored procedure must be defined by a CREATE PROCEDURE statement that has a Language clause of DATABASE and an EXTERNAL NAME clause that identifies the name of the procedure in the database and, optionally, the database schema to which it belongs.

Related reference:

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsideletebroker** command" on page 3863
Use the **mqsideletebroker** command to delete a named broker. The command also deletes the queues on the associated queue manager (created when the broker was created). You can also specify that the queue manager is to be deleted.

"**mqsichangetrace** command" on page 3822
Use the **mqsichangetrace** command to set the tracing characteristics for a broker.

"Database facilities" on page 6891
The database products used by WebSphere Message Broker also record information that might be useful if you have any problems with their access.

Resolving problems when using publish/subscribe

Use the advice given here to help you to resolve common problems that can occur when you run publish/subscribe applications.

About this task

Procedure

- "Application responses are not received"
- "Your application is not receiving publications" on page 3502
- "Publishing a message causes a filter error" on page 3502
- "Symbols in subscription filters cause problems" on page 3503
- "The Publication node fails with MQRC 2035" on page 3503
- "There are performance problems on AIX when the JIT compiler is not loaded" on page 3503

Application responses are not received

Procedure

- **Scenario:** Application responses are not received.
- **Explanation:** Depending on the application code, a publisher or subscriber might request confirmation that its message was processed successfully. Responses can make debugging client problems much easier, because a response code is given if a problem occurs.

  Typically, a response is always returned to a subscriber. However, for the publishing side, a message might be published without the knowledge of the originating application (for example, by using the default topic property on the input node, or by using a Compute node to modify this property in a message flow). The results of processing that message are still logged in user trace, which might give clues as to what is happening.

  If a response is not received, it is typically due to one of the following causes:

- The system is busy. Messages might build up on the input queue, and the client might not be waiting long enough for its response.
- WebSphere MQ expiry is being used. There are cases where this option is what is required, but the expiry of the input message is copied to the response. As a result, the message might expire on the input queue, or it might expire on the way back to the client. This situation is not an error, even with a persistent message.
- The input message or response might have been put to the dead-letter queue, if one is configured. Look on this queue to see if any new messages are there. This situation is typically accompanied by error messages in the log written by the broker that describe the problem. For example, the reply-to queue might have been specified incorrectly in the input message, therefore the reply message has been put to the dead-letter queue.

- **Solution:** If your application is not asking for responses (that is, not using messages of type MQMT_REQUEST) consider doing so, particularly when developing applications.

Your application is not receiving publications

Procedure

- **Scenario:** Your application is not receiving publications.
- **Explanation:** If an application has subscribed successfully (that is, has received an OK response to a RegSub message), it receives publications that match its subscription.

  Subscribers are sent messages only if they match the topic, the subscription point, *and* the filter. Because the subscription point is specified in the message flow, not in the publication message, an incorrect message flow setting can cause unexpected failures.

  A user trace of the flow that contains the Publication node shows you whether publications are matching anything.
- **Solution:** If a filter is being used, a user trace shows you whether this message is being evaluated as expected.

  The case with multiple execution groups, or multiple brokers, is more complex. A response is sent to a subscriber after the message has been processed by the target execution group. Other execution groups (and brokers) are updated asynchronously. As a result, there might be a delay before publications made elsewhere are received. If the broker is busy, there can be a delay before messages are processed fully. In a multi-broker setup, if communications have been suspended, subscription changes are propagated through the network of brokers. Check the channels.

  With multiple execution groups or brokers, it might be possible to fill intermediate WebSphere MQ queues if the load is high. This situation might be reported in the syslog (if a broker cannot put to a queue because it is full) or in the WebSphere MQ log (if a message coming across a channel cannot be put to the target queue because it is full). If you see messages of this type, display the queue depths on all your queue managers to see if any are almost full.

Publishing a message causes a filter error

Procedure

- **Scenario:** When you publish a message, you receive an error response message with reason text MQRCCF_FILTER_ERROR.
- **Explanation:** A broker returns this message to a publication when subscriptions have been registered that specify filter expressions (for Content Based Routing) and an error has been encountered when the broker attempts to filter the

published message. This situation can occur, for example, if a message is published that includes unsupported data types, or if the message body is corrupted.

Symbols in subscription filters cause problems

Procedure

- **Scenario:** If you specify certain symbols when you use filters in a subscription, the filter does not work. Sometimes your subscription messages are put to the dead-letter queue, and a number of error messages are written to the local error log indicating MQRFH2 parsing errors.
- **Explanation:** The MQRFH2 header employs standard XML encoding, so that its parser interprets some symbols in a special way.
- **Solution:** If you want to include these symbols in your filters, use the appropriate escape character to ensure that they are parsed correctly:

Symbol	Escape character
<	&lt;
>	&gt;
"	&quot;
'	&apos;
&	&amp;

For example, if you want to use:

```
<Filter>Body.e_ALERT_BODY.eqnum<6</Filter>
```

specify:

```
<Filter>Body.e_ALERT_BODY.eqnum&amp;lt;6</Filter>
```

The Publication node fails with MQRC 2035

Procedure

- **Scenario:** The Publication node fails with MQRC 2035.
- **Explanation:** WebSphere Message Broker publishes messages with the user ID in the original message, not the broker service ID.
- **Solution:** You can force WebSphere Message Broker to use the broker service ID in all circumstances by setting the environment variable *MQSI_PUBSUB_USE_BROKER_USERID* to any value. If there is no MQMD header, or if there is an MQMD header but its `UserIdentifier` field is blank, WebSphere Message Broker continues to use the broker's user ID.

There are performance problems on AIX when the JIT compiler is not loaded

Procedure

- **Scenario:** There are performance problems on AIX when the JIT compiler is not loaded.
- **Explanation:** The environment variable LIBPATH can affect the loading of the Java JIT (just-in-time) compiler on AIX. WebSphere Message Broker for AIX runs correctly without the JIT compiler, but publish/subscribe performance is adversely affected.
- **Solution:** Ensure that the JIT compiler runs.

  The JIT compiler loads and runs correctly if LIBPATH is not set, therefore do not set LIBPATH. You can make libraries available by linking them into `/var/wmqi/lib` (for all WebSphere Message Broker for AIX processes) or

`/usr/lib` (for all processes on the system). AIX WebSphere Message Broker for AIX configuration does this action for DB2 libraries.

If it is necessary to set LIBPATH, update it to include the directory `/usr/java130/bin`. For example, you can use the following command to start the broker:

```
LIBPATH=/usr/local/lib:/usr/java130/bin mqsistart mybroker
```

Related concepts:

"Publish/Subscribe" on page 2215
Publish/subscribe is a style of messaging application in which the providers of information (publishers) are decoupled from the consumers of that information (subscribers).

"Trace" on page 6871
If you cannot get enough information about a particular problem from the entries that are available in the various logs, the next troubleshooting method to consider is using trace. Trace provides more details about what is happening while code runs. The information produced from trace is sent to a specified trace record, so that you or IBM support personnel can analyze it to discover the cause of your problem.

"Routing using publish/subscribe applications" on page 2215
You can route your messages to applications using the publish/subscribe method of messaging.

Related tasks:

"Viewing Administration log information" on page 1007
You can view Administration log information by using either the WebSphere Message Broker Explorer, or the CMP API.

"Interpreting trace" on page 3546
Use the information in a formatted trace file to identify unexpected behavior.

Related reference:

"Publish/subscribe" on page 6395
Use the reference information in this section to help you develop publish/subscribe applications.

"User trace" on page 6873
User trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Administration log. User trace is inactive by default; you must activate it explicitly by using a command, or by selecting options in the WebSphere Message Broker Toolkit.

"WebSphere MQ logs" on page 6869
WebSphere MQ messages are written to the local error log in the same way as WebSphere Message Broker messages.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

Resolving problems with performance

Use the advice given here to help you to resolve common problems with performance.

About this task

- **Scenario:** You are experiencing problems with performance, such as:
  - Poor response times in the WebSphere Message Broker Toolkit when developing message flows
  - Poor response time at deployment
  - Individual messages taking a long time to process
  - Poor overall performance, or performance that does not scale well
- **Solution:** Possible solutions are:
  - Tune the broker
  - Speed up WebSphere MQ persistent messaging by optimizing the I/O (input/output)
  - Speed up database access by optimizing I/O
  - Increase system memory
  - Use additional instances or multiple execution groups
  - Optimize ESQL statements for best performance

A good source of information about performance is the set of reports in WebSphere MQ Family Category 2 (freeware) SupportPacs, available for download from the WebSphere MQ SupportPacs web page.

For more information, read "Do you have a component that is running slowly?" on page 3360.

This topic contains advice for dealing with some common performance problems that can arise:

The system is continuously running slower

Procedure

- **Scenario:** The system is continuously running slower.
- **Explanation:** When a process ends abnormally, an entry is made in the local error log, and a dump data file might be written to the errors or z/OS log directory. This directory is unbounded, and if you do not clear it of unwanted files, you might find that your system performance degrades due to significant space being used by old abend files.
- **Solution:** Periodically clear the errors or z/OS log directory of unwanted files.

On Windows, use the **-w** parameter of the **mqsicreatebroker** command to create the errors directory in a hard disk drive partition that does not contain WebSphere Message Broker or Windows itself.

You experience configuration problems with many components

Procedure

- **Scenario:** You are experiencing configuration problems with many components.
- **Explanation:** If you are running WebSphere Message Broker with many configured components, the memory footprint of the broker processes (particularly the execution groups or DataFlowEngines) might exceed their memory limits. In particular, the user process limit might be exceeded, or the address space limit might be reached. You might encounter problems, such as the BIP2106E or BIP2137E error messages, when running a broker with:
  - Many message flows
  - Multiple databases
  - Large input or output messages
- **Solution:** Use tools that are specific to your operating system to check the maximum size of the failing process, then check for any user limits (if applicable) or computer limits on process size.
  1. On Windows, the maximum process size is 2 GB. Increasing user limits beyond this value does not make any difference. If your broker processes regularly reach this size, consider spreading your message flows across more execution groups to reduce the size of each one below these limits.
  2. On HP-UX, the number of threads per process is limited by the kernel parameter **max_thread_proc**. The more flows you add to an execution group, the more threads you use for each process. If you need many flows in an execution group, increase the value of the **max_thread_proc** kernel parameter. Try setting the kernel parameter **max_thread_proc** to 256, or to 512 if you are using WebSphere Transformation Extender (WTX).

A WHILE loop in a large XML array takes a long time to process

Procedure

- **Scenario:** A WHILE loop in a large XML array takes a long time to process.
- **Explanation:** This problem arises when you use the CARDINALITY() function to determine the size of the array in the WHILE statement. With large arrays, the cost of determining the number of elements is proportional to the size of the array. The CARDINALITY function is invoked each time the WHILE statement is executed. The message has many elements, therefore takes a long time to process when running the loop in this way.
- **Solution:** Unless you have a message in which the number of elements of the array grows dynamically, determine the size of the array before entering the WHILE loop. Use code like the following example:

```
DECLARE i,c INTEGER;
SET i = 1;
SET c=CARDINALITY(OutputRoot.XMLNS.MyMessage.Array[ ]);
WHILE i <= c DO
      . . .
      . . .  loop processing
      . . .
END WHILE;
```

Remote waitForMessages calls with WebSphere MQ Everyplace are slow

Procedure

- **Scenario:** Remote waitForMessages calls with WebSphere MQ Everyplace are slow.
- **Explanation:** A remote waitForMessages() call necessarily results in a polling action: the client queue manager attempts to get a message repeatedly until one is available on the remote queue or a timeout is reached.
- **Solution:** Define a store-and-forward queue on the broker queue manager and a home-server queue on the client. These queues provide a level of de-coupling that allows the arrangement to work in the event of the client queue manager becoming unavailable. Alternatively, specify a remote queue in the WebSphere MQ Everyplace output node, so that a local GET call is sufficient on the client side.

Performance is reduced for stored procedures

Procedure

- **Scenario:** Stored procedures that do not return results sets do not perform as well as they did when using the DataDirect version 4.1 drivers.
- **Explanation:** The new DataDirect version 5 Oracle drivers now use the configuration parameter **`ProcedureRetResults`** to allow the driver to return results sets from stored procedures. This parameter applies only to Oracle. By default, the parameter is set to 1 and has to be 1 if you use stored procedures that return results sets. When this parameter is set to 0, the driver does not return results sets from stored procedures, which can result in better performance.
- **Solution:** If your stored procedures do not return any results sets, set the **`ProcedureRetResults`** parameter to 0 (zero).

Message flow performance is reduced when you access message trees with many repeating records

Procedure

- **Scenario:** Message flow performance is reduced when the following conditions are true:
  - You are using ESQL processing to manipulate a large message tree.
  - The message tree consists of repeating records or many fields.
  - You have used explicit SET statements with field reference paths to access or create the fields.
  - You have observed a gradual slowing of message flow processing as the ESQL processes more fields or repetitions.
- **Explanation:** This problem occurs when you use field references, rather than reference variables, to access or create consecutive fields or records.

  Consider the following example, in which independent SET statements use field reference paths to manipulate the message tree. The SET statement takes a source and target parameter, where either or both parameters are field references:

  ```
  SET OutputRoot.XMLNS.TestCase.StructureA.ParentA.field = '1';
  ```

  The problem arises when the SET statement is used to create many more fields, as shown in the following example:

```
SET OutputRoot.XMLNS.TestCase.StructureA.ParentA.field1 = '1';
SET OutputRoot.XMLNS.TestCase.StructureA.ParentA.field2 = '2';
SET OutputRoot.XMLNS.TestCase.StructureA.ParentA.field3 = '3';
SET OutputRoot.XMLNS.TestCase.StructureA.ParentA.field4 = '4';
SET OutputRoot.XMLNS.TestCase.StructureA.ParentA.field5 = '5';
```

In this example, the five fields that are created are all children of ParentA. Before the specified field can be created or modified, the broker must navigate the named message tree to locate the point in the message tree that is to be altered. For example:

- To access field 1, the SET statement navigates to ParentA, then to the first field, therefore involving two navigations.
- To access field 5, the SET statement navigates to ParentA, then traverses each of the previous fields until it reaches field 5, therefore involving six navigations.

Navigating over all the fields that precede the specified field causes the loss in performance.

Now consider a scenario that accesses repeating fields in an input message tree; for example:

```
DECLARE myChar CHAR;
DECLARE thisRecord INT 0;
WHILE thisRecord < 10000 DO
 SET thisRecord = thisRecord + 1;
 SET myChar = InputRoot.MRM.myParent.myRepeatingRecord[thisRecord];
END WHILE;
```

When index notation is used, as the count increases, the processing needs to navigate over all the preceding fields to get the one it wants; that is, it has to count over the previous records to get to the one that is represented by the current indexed reference.

- When accessing InputRoot.MRM.myParent.myRepeatingRecord[1], one navigation takes place to get to the first record.
- When accessing InputRoot.MRM.myParent.myRepeatingRecord[2], two navigations take place to get to the second record.
- When accessing InputRoot.MRM.myParent.myRepeatingRecord[N], *N* navigations take place to get to the N-th record.

Therefore, the total number of navigations for this WHILE loop is: 1 + 2 + 3 + ... + *N*, which is not linear.

- **Solution:** If you are accessing or creating consecutive fields or records, use reference variables. When you use reference variables, the statement navigates to the main parent, which maintains a pointer to the field in the message tree. The following example shows the ESQL that can be used to reduce the number of navigations when creating new output message tree fields:

```
SET OutputRoot.XMLNS.TestCase.StructureA.ParentA.field1 = '1';
DECLARE outRef REFERENCE TO OutputRoot.XMLNS.TestCase.StructureA.ParentA;
SET outRef.field2 = '2';
SET outRef.field3 = '3';
SET outRef.field4 = '4';
SET outRef.field5 = '5';
```

When referencing repeating input message tree fields, you could use the following ESQL:

```
DECLARE myChar CHAR;
DECLARE inputRef REFERENCE TO InputRoot.MRM.myParent.myRepeatingRecord[1];
WHILE LASTMOVE(inputRef) DO
 SET myChar = inputRef;
 MOVE inputRef NEXTSIBLING NAME 'myRepeatingRecord';
END WHILE;
```

For further information, see "Creating dynamic field references" on page 2431.

You are experiencing poor performance in the WebSphere Message Broker Toolkit when working with large projects

Procedure

- **Scenario:** You are experiencing poor performance in the WebSphere Message Broker Toolkit when working with large or complex projects.
- **Explanation:** Performance is reduced because of frequent project changes, such as adding and removing projects, or using **Project** > **Clean**. Complete project updates use large amounts of memory due to the size, number, and connections between files.
- **Solution:** Increase your system memory.

Performance is reduced when you run Web Services with small message sizes

Procedure

- **Scenario:** You see poor response times and throughput rates when you run Web Services using HTTP, and send smaller messages sizes (typically less than 32 KB). Throughput rates can fluctuate with message size. WebSphere Message Broker running on the AIX platform might be affected.
- **Explanation:** The default configuration of HTTP enables the Nagle algorithm, which seeks to improve the efficiency of Internet Protocol networks by reducing the number of packets sent. It works by buffering small packets together, creating a smaller number of large packets. The HTTPRequest node uses the platform default for the **tcpnodelay** setting of its sockets. You can disable the Nagle algorithm at either the operating system level (system wide) or through WebSphere Message Broker (affecting only the WebSphere Message Broker HTTP sockets).
- **Solution:** Use the following commands to disable the Nagle algorithm:

  HTTP Request node

  ```
  mqsichangeproperties <BrokerName> -e <ExecutionGroupName>
        -o ComIbmSocketConnectionManager -n tcpNoDelay -v true|false
  mqsichangeproperties <BrokerName> -e <ExecutionGroupName>
        -o ComIbmSocketConnectionManager -n tcpNoDelaySSL -v true|false
  ```

  HTTP Listener / Tomcat servlet engine

  ```
  mqsichangeproperties <BrokerName> -b httplistener
        -o HTTPConnector -n tcpNoDelay -v true|false
  mqsichangeproperties <BrokerName> -b httplistener
        -o HTTPSConnector -n tcpNoDelay -v true|false
  ```

  To determine the value set, take the following steps:

  HTTP Request node

  Use the following command:

  ```
  mqsireportproperties <BrokerName> -e <ExecutionGroupName>
        -o ComIbmSocketConnectionManager -r
  ```

  HTTP Listener / Tomcat servlet engine

  Check the WebSphere Message Broker registry.

You are experiencing reduced Java performance, or Java performance degrades after debugging a message flow

Procedure

- **Scenario:** The Java code in the JavaCompute, the Java user-defined node, or the XSLTransform node does not run as quickly as expected, or performance in these nodes degrades after debugging a message flow. This performance degradation might particularly affect the deployment of XML schemas. Other Java based nodes might also experience degraded performance: for example, adapter nodes (SAP, PeopleSoft, JD Edwards, Siebel), SOAP nodes, CORBA nodes, IMS nodes, CICS nodes and JMS nodes.
- **Explanation:** The broker disables the Java JIT (just-in-time) compiler on the Java virtual machine (JVM) that is created when the execution group is started if you are debugging a message flow. A disabled Java JIT compiler provides a greater choice of debugging options, however any optimizations typically performed during JIT compilation are prevented, which might result in degraded performance.
- **Solution:** If you are not debugging a message flow, set the JVM debug port to zero using this command:

  ```
  mqsichangeproperties broker_name -e execution_group_name
  -o ComIbmJVMManager -n jvmDebugPort -v 0
  ```

  For example:

  ```
  mqsichangeproperties TEST -e default
  -o ComIbmJVMManager -n jvmDebugPort -v 0
  ```

Related tasks:

Chapter 12, “Performance and monitoring,” on page 3251
You can change various aspects of your broker configuration to tune brokers and message flows, and monitor message flows and publish/subscribe applications.

“Considering performance in the broker environment” on page 586
When you design your broker environment, and the resources associated with the brokers, decisions that you make can affect the performance of your brokers and applications.

“Tuning the broker” on page 3254
You can complete several tasks that enable you to tune different aspects of the broker performance.

Resolving problems when developing Administration API applications

Use the advice given here to help you to resolve problems that can arise when developing Administration API for WebSphere Message Broker (also known as the CMP API) applications.

About this task

- “Your CMP application hangs if the broker is not available”
- “You set a property of an object and query its value, but the value has not changed” on page 3511
- “You cannot connect to a broker when using a `.broker` file” on page 3511

Your CMP application hangs if the broker is not available

Procedure

- **Scenario:** When the broker is unavailable, the CMP application hangs.

- **Explanation:** Communication between the CMP and the broker is asynchronous, therefore the CMP hangs because it is waiting for a message from the broker.
- **Solution:** Configure the maximum amount of time that the CMP waits by using the following method:

  ```
  // Wait for a maximum of 10 seconds
  BrokerProxy.setRetryCharacteristics(10000);
  ```

  The value specified represents the time in milliseconds that the CMP will wait for information before throwing the `BrokerProxyPropertyNotInitializedException` exception.

  If you set this timeout value too low, an exception is thrown, even if the broker is available.

You set a property of an object and query its value, but the value has not changed

Procedure

- **Scenario:** You have set a property of an object, then queried its value; the value has not changed.
- **Explanation:** Methods that change properties of broker objects are not processed immediately. If you call a property change method on a CMP object, the CMP API sends a message that requests the specified change to the broker. The broker processes the request asynchronously, and notifies all AdministeredObjectListeners of the affected object when the change has been attempted.
- **Solution:** Methods that change state typically return to the calling program as soon as the request has been put to the queue manager of the broker, or, following a call to BrokerProxy.beginUpdates(), as soon as the request has been added to the current batch. If the property has still not been updated after the action's response to the request has been returned to the application, consult the response message for more details.

You cannot connect to a broker when using a `.broker` file

Procedure

- **Scenario:** You cannot connect to a broker when you use a `.broker` file.
- **Explanation:** If your CMP applications use the MQPropertyFileBrokerConnectionParameters class, they can connect to a broker by using a connection file that has a `.broker` extension. However, this file can be parsed only if an XML parser is available.
- **Solution:** Ensure that a supported parser is available on the CLASSPATH. A supported parser is shipped with WebSphere Message Broker.

  Alternatively, your application can use the MQBrokerConnectionParameters class instead of the MQPropertyFileBrokerConnectionParameters class. This class connects to a broker by specifying the host name, queue manager name, and queue manager port of the target broker directly. This method does not require an XML parser.

Resolving problems with user-defined extensions

Advice for dealing with some common problems that can arise when you work with user-defined extensions

About this task

Procedure

- "You cannot deploy one of your user-defined nodes, despite having a plug-in LIL in the correct directory."
- "You cannot deploy a flow with one of your user-defined nodes in it."
- "You get problems when nodes try to use the ESQL path interface in the plug-in API"
- "After migration your custom property editor does not work" on page 3513
- "Interpreting problems in user-defined extensions" on page 3513
- "You want to debug classloading" on page 3515
- "An error is issued when you deploy a user-defined extension on z/OS" on page 3516
- "You cannot determine which user-defined extensions have been loaded by the broker on startup" on page 3516
- "You are migrating a C user-defined node and cniDefineNodeClass returns CCI_INV_IMPL_FUNCTION." on page 3517

You cannot deploy one of your user-defined nodes, despite having a plug-in LIL in the correct directory.

Procedure

- **Scenario:** You cannot deploy one of your user-defined nodes, despite having a plug-in LIL in the correct directory.
- **Explanation:** You have `memset()` the data area to zero and have not initialized the CNI_VFT structure with the initialization constant `{CNI_VFT_DEFAULT}`.
- **Solution:** Initialize by copying a predefined initialization structure over the function table area, as follows:

  ```
  static CNI_VFT virtualFunctionTable = {CNI_VFT_DEFAULT};
  ```

  In addition, implement logging from your user-defined node so that you can see if the plug-in API is producing error codes; the broker does not log these to its own log, unless you take a service trace.

You cannot deploy a flow with one of your user-defined nodes in it.

Procedure

- **Scenario:** You cannot deploy a flow with one of your user-defined nodes in it.
- **Explanation:** Your LIL file has failed to load.
- **Solution:** Check system log (syslog or Eventviewer) of broker startup; did you see a BIP2308 message saying a LIL file failed to load? If there are any problems loading a LIL file, a BIP2308 message appears in the system log.

You get problems when nodes try to use the ESQL path interface in the plug-in API

Procedure

- **Scenario:** When you attempt to use the ESQL path interface in the plug-in API, the return value is CCI_PATH_NOT_NAVIGABLE.
- **Explanation:** The plug-in API allows you to specify a path in the form of an ESQL path expression and navigate to that element, returning a handle to it if it exists. It also allows you to create elements along the path to the requested element.

The navigate path utility function (`cniSqlNavigatePath`) executes the SQLPathExpression created with the `cniSqlCreateReadOnlyPathExpression` or `cniSqlCreateModifiablePathExpression` utility functions, as defined by the `sqlPathExpression` argument.

If the path is not navigable, the return code is set to CCI_PATH_NOT_NAVIGABLE. This might be returned when embedding a path expression in another path expression. The input `cciMessage*` functions must not be NULL; however, any of the output `cciMessage*` functions can be NULL. If you embed a path expression that can be NULL inside a path expression that cannot be NULL, CCI_PATH_NOT_NAVIGABLE is returned.

- **Solution:** If the return code is set to CCI_PATH_NOT_NAVIGABLE, ensure that if a correlation name is specified in a path, the respective parameter is not NULL.

After migration your custom property editor does not work

Procedure

- **Scenario:** You have migrated to a new version of WebSphere Message Broker and your custom property editor no longer works.
- **Explanation:** Custom property editors can use Eclipse or RAD APIs. If any of those APIs are changed in a new version of WebSphere Message Broker, your property editor might not work.
- **Solution:** Update your property editor code to comply with the changed API.

Interpreting problems in user-defined extensions

Procedure

- **Scenario:** You want to debug problems in user-defined nodes and parsers.
- **Solution:** Start user trace at debug level. In order to see BIP4142, BIP4144, BIP4145, and BIP4146 messages, this must be done at the execution group level. For example, use the **`mqsichangetrace`** command without the **`-f`** parameter.

  The following debug messages are available to help you to understand the execution of your user-defined nodes and parsers:

  – BIP2233 and BIP2234: a pair of messages that are traced before and after a user-defined extension implementation function is invoked. These messages report the input parameters and returned value. For example:

    BIP2233 `Invoking user-defined extension function [function name] ([function call parameters])`

    BIP2234 `Returned from user-defined extension function [function name] with result: [result of call]`

    Note: In these messages, an *implementation function* can be interpreted as either a C implementation function or a Java implementation method.

  – BIP2308: a message that is logged when the broker fails to load a LIL file.

    BIP2308 `File [name of LIL file] could not be loaded; operating system return code [error code return from operating system]`

  – BIP3904: a message that is traced before invoking the Java `evaluate()` method of a user-defined node. For example:

    BIP3904 `(for Java): Invoking the evaluate() method of node (class=[node class name], name=[label of node in flow])` where *node class name* is the name of the Java user-defined extension class.

  – BIP3905: a message that is traced before invoking the C `cniEvaluate` implementation function (iFpEvaluate member of CNI_VFT) of a user-defined node. For example:

BIP3905 (for C): Invoking the cniEvaluate() implementation function of node (class=[node class name], name=[label of node in flow]) where *node class name* is the name of the user-defined extension class that is provided by the user-defined extension while calling C cniDefineNodeClass.

– BIP4142: a debug message that is traced when invoking a user-defined node utility function, where the utility function alters the state of a syntax element. This includes all utility functions that start with cniSetElement*, where * represents all nodes with that stem. For example:

  BIP4142 Evaluating cniSetElement [element identifier type]. Changing value from [value before user's change] to [value after user's change]"

– BIP4144 and BIP4145: a pair of messages that are traced by certain implementation functions that, when invoked by a user-defined extension, can modify the internal state of a message broker's object. Possible message broker objects include syntax element, node, and parser. These messages report the input parameter provided to the invoked method and the returned value. For example:

  BIP4144 Entered function [function name] ([function call parameters])

  BIP4145 Exiting function [function name] with result: [result to be returned]

  In these messages, an *implementation function* can be interpreted as either a C implementation function or a Java implementation method.

  The C implementation functions that invoke messages BIP4144 and BIP4145 include:

For user-defined parsers	For user-defined nodes
cpiCreateParserFactory	cniCreateElement*
cpiDefineParserClass	cniDeleteMessage
cpiAppendToBuffer	cniAdd*
cpiCreateElement	cniDetach
cpiCreateAndInitializeElement	cniCopyElementTree
cpiAddBefore	cniFinalize
cpiAddAfter	cniWriteBuffer
cpiAddAsFirstChild	cniSql*
cpiAddAsLastChild	cniSetInputBuffer
cpiSetNameFromBuffer	cniDispatchThread

  (* represents all nodes with that stem; for example, cniAdd* includes cniAddAfter, cniAddasFirstChild, cniAddasLastChild, and cniAddBefore.)

  The Java methods that invoke messages BIP4144 and BIP4145 are:

For user-defined nodes
com.ibm.broker.plugin.MbElement.CreateElement*
com.ibm.broker.plugin.MbElement.add*
com.ibm.broker.plugin.MbElement.detach
com.ibm.broker.plugin.MbElement.copyElementTree

– BIP4146: a debug message that is traced when invoking a user-defined parser utility function, where the utility function alters the state of a syntax element.

This includes all utility functions that start with `cpiSetElement*`, where * represents all nodes with that stem. For example:

BIP4146 `Evaluating cpiSetElement [element identifier type]. Changing value from [value before user's change] to [value after user's change]`

For information on the C user-defined API, see the "C language user-defined parser API" on page 6538 and the "C language user-defined node API" on page 6416.

- BIP4147: an error message that is traced when a user-defined extension passes an invalid input object to a user-defined extension utility API function. For Example:

  BIP4147 `User-defined extension input parameter failed debug validation check. Input parameter [parameter name] passed into function [function name] is not a valid object.`

- BIP4148: an error message that is traced when a user-defined extension damages a broker's object. For Example:

  BIP4148 `User-defined extension damaged broker's object. Function [function name] has damaged broker's object passed as parameter [parameter name].`

- BIP4149: an error message that is traced when a user-defined extension passes an invalid input data pointer to a user-defined extension utility API function. For Example:

  BIP4149 `User-defined extension input parameter failed debug validation check. Input parameter [parameter name] passed into function [function name] is a NULL pointer.`

- BIP4150: an error message that is traced when a user-defined extension passes invalid input data to a user-defined extension utility API function. For example:

  BIP4150 `User-defined extension input parameter failed debug validation check. Input parameter [parameter name] passed into function [function name] does not have a valid value.`

- BIP4151: a debug message that is traced when `cniGetAttribute2` or `cniGetAttributeName2` sets the return code to an unexpected value. Expected values are CCI_SUCCESS, CCI_ATTRIBUTE_UNKNOWN, and CCI_BUFFER_TOO_SMALL. Any other value is an unexpected value. For example:

  BIP4151 `An unexpected value was returned from User-defined extension implementation function [function name].`

- BIP4152: a debug message that is traced when `cniGetAttribute2` or `cniGetAttributeName2` sets the return code to CCI_BUFFER_TOO_SMALL, and then `cniGetAttribute2` or `cniGetAttributeName2` is called again, this time with the correct size buffer, however the return code is still set to CCI_BUFFER_TOO_SMALL. For example:

  BIP4152 `User-defined extension Implementation function [function name] returned CCI_BUFFER_TOO_SMALL on 2nd attempt.`

You want to debug classloading

Procedure

- **Scenario:** You want to debug classloading.
- **Solution:** Classes and the location from which they are loaded are written to user trace. Use this information to check that the correct classes are being loaded.

An error is issued when you deploy a user-defined extension on z/OS

Procedure

- **Scenario:** When you deploy a user-defined extension on z/OS, Linux, or UNIX, an error is displayed in the log of each execution group, stating that there is insufficient authority to open the LIL file.
- **Explanation:** On Linux and UNIX, the user-defined extension must have group read permission. On z/OS, the user-defined extension must have group execute permission.
- **Solution:**
  - On Linux and UNIX, set the file permissions of the user-defined extension to group read by issuing the command **chmod a+r**.
  - On z/OS, set the file permissions of the user-defined extension to group read and execute by issuing the command **chmod a+rx**.

You cannot determine which user-defined extensions have been loaded by the broker on startup

Procedure

- **Scenario:** You cannot determine which user-defined extensions have been loaded by the broker on startup.
- **Solution:** Use the **mqsireportproperties** command for each type of user-defined extension.
  - For a Java user-defined extension, issue the command:

```
mqsireportproperties WBRK_BROKER -e default -o ComIbmJavaPluginNodeFactory -r
```

You see a report similar to this example:

```
ComIbmJavaPluginNodeFactory
 uuid='ComIbmJavaPluginNodeFactory'
 userTraceLevel='none'
 traceLevel='none'
 userTraceFilter='none'
 traceFilter='none'
 NodeClassName='ComIbmJMSClientInputNode'
 NodeClassName='ComIbmJMSClientOutputNode'
 NodeClassName='ComIbmJavaComputeNode'
 NodeClassName='ComIbmXslMqsiNode'
 NodeClassName='SearchFilterNode'

BIP8071I: Successful command completion.
```

The user-defined extension called SearchFilter has a NodeClassName of SearchFilterNode.

  - For a C user-defined extension (assuming that CONST_PLUGIN_NODE_FACTORY was set to ComIbmSamplePluginNodeFactory in the `NodeFactory.h` file, as in the sample NumComputeNode), issue the command:

```
mqsireportproperties WBRK_BROKER -e default -o ComIbmSamplePluginNodeFactory -r
```

You see a report similar to this example:

```
ComIbmSamplePluginNodeFactory
 uuid='ComIbmSamplePluginNodeFactory'
 userTraceLevel='none'
 traceLevel='none'
 userTraceFilter='none'
 traceFilter='none'
 NodeClassName='NumComputeNode'

BIP8071I: Successful command completion.
```

The user-defined extension called NumCompute has a NodeClassName of NumComputeNode.

You are migrating a C user-defined node and cniDefineNodeClass returns CCI_INV_IMPL_FUNCTION.

Procedure

- **Scenario:** When you attempt to migrate a C user-defined node, `cniDefineNodeClass` returns CCI_INV_IMPL_FUNCTION.
- **Explanation:** New fields have been added to the CNI_VFT struct. CNI_VFT_DEFAULT has been updated to initialize these new fields in the header file BipCci.h. If you initialize your CNI_VFT with CNI_VFT_DEFAULT, you should not need to make any code changes. However, if you do not initialize CNI_VFT with CNI_VFT_DEFAULT, these new fields are initialized with random values.
- **Solution:** Initialize your CNI_VFT with CNI_VFT_DEFAULT.

Related concepts:

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

Related tasks:

"Installing user-defined extension runtime files on a broker" on page 3125
Install the compiled runtime files for your user-defined extension on the broker on which you want to test its function.

"Starting user trace" on page 3197
Use user trace for debugging your applications; you can trace brokers, execution groups, and deployed message flows. Start user trace facilities using the **mqsichangetrace** command or the WebSphere Message Broker Explorer.

"Is there a problem with a database?" on page 3356
If you have database problems, complete a set of initial checks to identify errors.

"Has the message flow run successfully before?" on page 3353
Sometimes a problem appears in a message flow that has previously run successfully.

Related reference:

"cciGetLastExceptionData" on page 6647
Gets diagnostic information about the last exception generated. Information about the last exception generated on the current thread is returned in a CCI_EXCEPTION_ST output structure. The user-defined extension can use this function to determine whether any recovery is required when a utility function returns an error code.

"Utility function return codes and values" on page 6686
By convention, the return code output parameter of all utility functions is set to indicate successful completion, or an error. The table lists all return codes with their meanings.

Resolving problems when installing

Work through the advice provided to help you to deal with problems that can arise when the product is installed.

About this task

The following list describes typical problems when installing, with a corresponding solution or workaround:

- Broker component:
  - "Dealing with problems during installation"
  - "Installation process is interrupted" on page 3519
  - "java.lang.UnsatisfiedLinkError" on page 3519
  - "RPM query fails" on page 3519
  - "Display problems" on page 3520
  - "Insufficient temporary space" on page 3520
  - "Fix Pack installation fails with license agreement error" on page 3521
  - "Installation fails with version error" on page 3521
- WebSphere Message Broker Toolkit:
  - "Dealing with problems during installation" on page 3521
- WebSphere Message Broker Explorer:
  - "Dealing with problems during installation" on page 3522
- Installation Manager:
  - "Installation Manager hangs" on page 3522
  - "Installation Manager does not show the WebSphere Message Broker Toolkit components" on page 3523
  - "Error restoring Installation Manager state" on page 3523
- WebSphere Message Broker Launchpad:
  - "WebSphere Message Broker Launchpad return code indicates that an installation has failed" on page 3524

Dealing with problems during installation

Before you begin

Broker component on all operating systems:

Procedure

- **Scenario:** You have problems during the installation of the Broker component.
- **Solution:** Complete the following steps:
  1. If you experience problems when installing on z/OS, see the *Program Directory for WebSphere Message Broker for z/OS* on the WebSphere Message Broker Library web page.
  2. Refer to the readme file `readme.html` for any late changes to the installation instructions.
  3. If you transferred the installation media to your local or networked filesystem, you might see the following error:

     ```
     The Install executable launcher was unable to locate its companion shared library.
     ```

     Ensure that you have read and execute privileges on all installation directories and files. Also ensure that the length of the path to the installation media is within the limits imposed by your operating system.
  4. If installation is successful, the installation wizard returns a return code of zero. If a non-zero return code is returned, check the installation log file for

errors and explanations. Problems associated with the installation of the Broker component are recorded in the log file `mqsi7_install.log`, which is stored in your installation directory.

If you accepted the default location during installation, this directory is as follows. The default directory includes the version and release of the product that you are installing in the format v.r (version.release):

Linux `/opt/ibm/mqsi/v.r`

UNIX `/opt/IBM/mqsi/v.r`

Windows 32-bit
`C:\Program Files\IBM\MQSI\v.r`

Windows 64-bit
`C:\Program Files\IBM\MQSI\v.r` for the 64-bit version of WebSphere Message Broker

`C:\Program Files (x86)\IBM\MQSI\v.r` for the 32-bit version of WebSphere Message Broker

These locations define the default value of *install_dir* on each platform.

5. If you are still unable to resolve the problem, contact your IBM Support Center.

Installation process is interrupted

Before you begin

Broker component on all operating systems:

Procedure

- **Scenario:** You are installing the Broker component but the process is interrupted before completion.
- **Explanation:** The process might be interrupted because of a problem, such as a power failure.
- **Solution:** Delete the *install_dir* and all its contents before you restart the program.

java.lang.UnsatisfiedLinkError

Before you begin

Broker component on Linux:

Procedure

- **Scenario:** You are using the Broker component graphical interface to install on Linux but the interface does not work correctly.
- **Explanation:** The additional packages that are required to complete the installation are not installed.
- **Solution:** You must install additional packages for the interface to work correctly. For more information about the required additional packages, see "Operating system requirements" on page 3590.

RPM query fails

Before you begin

Broker component on Linux:

Procedure

- **Scenario:** You start a Red Hat package manager (RPM) query after you have installed the product, but nothing is returned. An information message like the following example might have been reported in the installation log file:

```
(01-Jun-2005 09:02:27), mqsi.Setup,
   com.ibm.wizard.platform.linux.LinuxProductServiceImpl, wrn,
   The installer could not successfully add the product information
   into the RPM database. Installation will continue as this is not
   critical to the installation of the product.
```

- **Explanation:** It is likely that your system does not have the required RPM support.
- **Solution:** Install the additional RPM build package that is described in "Operating system requirements" on page 3590.

Display problems

Before you begin

Broker component on Linux and UNIX:

Procedure

- **Scenario:** You try to install the Broker component by using the graphical interface and see one of two common errors reported.
- **Explanation:** Both errors typically occur if you log in remotely, or you switch user ID.
- **Solution:** Address the appropriate error with the following corresponding solution:

  `Can't open display localhost:1.0`
  Check that the DISPLAY variable is set to the correct value. If you are logged in locally, the typical value is `:0.0` or `localhost:0.0`.

  `Connection to ":0.0" refused by server`
  Run the following command, where *user* is the user ID you are logged in as:

  ```
  xauth merge ~user/.Xauthority
  ```

  If you are unable to correct these errors, contact your systems administrator for further help.

Insufficient temporary space

Before you begin

Broker component on Linux and UNIX:

Procedure

- **Scenario:** You install the Broker component and the installation program tries to unpack product files into the temporary file space of the local system. On Linux and UNIX systems, the temporary space is typically located in `/tmp`.
- **Explanation:** If sufficient file space is not available in this directory, the command might fail without reason and returns no comment, or reports a lack of file space.
- **Solution:** Give the installation wizard a temporary file system to use, for example; `setupaix`. The command-line option is `-is:tempdir` *name of temp directory*. For example, on AIX, enter the following command:

  ```
  ./setupaix -is:tempdir /largetemp
  ```

Do not specify a temporary directory that is NFS-mounted from another server; if you do so, the installation might fail because user permission checks made by the installer sometimes report an error that security principals mqm and mqbrkrs do not exist on the local machine.

For more information about checking how much temporary space is required, see "Memory and disk space requirements" on page 3584.

Fix Pack installation fails with license agreement error

Before you begin

Broker component on AIX with WPAR configuration:

Procedure

- **Scenario:** You are installing a Fix Pack for the Broker component on an AIX WPAR system but the installation fails with the message: "The License Agreement could not be properly loaded."
- **Explanation:** Fix Pack installations check that the license agreement was accepted by checking that the Broker component exists on the target system. On AIX, the installer performs this check by searching the `/usr/lib/objrepos/vpd.properties` file for an entry that matches this version and release of WebSphere Message Broker. If the `/usr` file system is read-only, the base product installation succeeds, but cannot update the `vpd.properties` file. Subsequent Fix Pack installations cannot determine that the Broker component is installed.
- **Solution:** Ensure that the user ID root can write to file `/usr/lib/objrepos/vpd.properties`. Reinstall the base product and check that `/usr/lib/objrepos/vpd.properties` was successfully updated. Then apply a Fix Pack.

Installation fails with version error

Before you begin

Broker component on all operating systems:

Procedure

- **Scenario:** You are installing the WebSphere Message Broker runtime component, but the installation fails with the message: "Installed product has newer version."
- **Explanation:** Files or directories left behind from previous uninstallation.
- **Solution:** Manually delete the residual files and directories from the installation location.

Dealing with problems during installation

Before you begin

WebSphere Message Broker Toolkit on all operating systems:

Procedure

- **Scenario:** You have problems during the installation of the WebSphere Message Broker Toolkit.
- **Solution:** Complete the following steps:
  1. Refer to the readme file `readme.html` for any late changes to the installation instructions.
  2. If installation is successful, the installation wizard returns a return code of zero. If a non-zero return code is returned, check the Installation Manager log file for errors and explanations. Problems associated with the installation

of the WebSphere Message Broker Toolkit are recorded in the Installation Manager log file *YYYYMMDD_TIME*`.xml`. Where *YYYYMMDD_TIME* is the date and time of installation. The Installation Manager log file is stored in the following location:

Linux `/var/ibm/InstallationManager/logs`

Windows
On Windows systems, the directory is created in the following default location, however the actual location might differ on your computer:

`%ALLUSERSPROFILE%\Application Data\IBM\Installation Manager\logs`Where `%ALLUSERSPROFILE%` is the environment variable that defines the system working directory. The default directory depends on the operating system:

- On Windows XP and Windows Server 2003:
  `C:\Documents and Settings\All Users\Application Data\IBM\Installation Manager\logs`
- On Windows Vista and later operating systems:
  `C:\ProgramData\IBM\Installation Manager\logs`

3. If you are still unable to resolve the problem, export the Installation Manager related information from **Installation Manager** > **Help menu** > **Export Data for Problem Analysis**, and contact your IBM Support Center with the exported information.

Dealing with problems during installation

Before you begin

WebSphere Message Broker Explorer on all operating systems:

Procedure

- **Scenario:** You have problems during the installation of the WebSphere Message Broker Explorer.
- **Solution:** Complete the following steps:
  1. See the *Program Directory for WebSphere Message Broker for z/OS* if you experience problems when installing on z/OS.
  2. Refer to the readme file `readme.html` for any late changes to the installation instructions.
  3. If installation is successful, the installation wizard returns a return code of zero. If a non-zero return code is returned, check the log file for errors and explanations. Problems associated with the installation of the WebSphere Message Broker Explorer are recorded in the log file `MBExplorer_install.log`, which is stored in the installation directory. The default installation directories are as follows:

     Linux `opt/IBM/MBExplorer`

     Windows
     `C:\Program Files\IBM\MBExplorer`

Installation Manager hangs

Before you begin

Installation Manager on Linux on x86 and Windows:

Procedure

- **Scenario:** You click **Next** in the Installation Manager when it first opens and the Installation Manager hangs.
- **Solution:** Close the window and reopen it.

Installation Manager does not show the WebSphere Message Broker Toolkit components

Before you begin

Installation Manager on Linux on x86 and Windows:

Procedure

- **Scenario:** You are installing the WebSphere Message Broker Toolkit but the initial Install Packages page that is displayed by the Installation Manager does not show the WebSphere Message Broker Toolkit components.
- **Explanation:** The location of the update repository has not been set correctly.
- **Solution:** Select **File** > **Preferences** and click **Add Preferences**. Enter the web address or directory where the installation packages are stored or click **Browse** to search for the correct location. Click **OK**. The packages are listed in the Install Packages page.

Error restoring Installation Manager state

Before you begin

Installation Manager on Linux and Windows:

Procedure

- **Scenario:** You already have the IBM Installation Manager installed on your workstation and the version is earlier than Installation Manager Version 1.3.4.1. While installing a newer version of the IBM Installation Manager, as supplied with WebSphere Message Broker Toolkit, you cancel the installation before completion. When you try to restart the installed Installation Manager from your workstation, you receive the following error:

  ```
  Error restoring Installation Manager state
  Installation data has incompatible version 0.0.4; expected [0.0.2,0.0.3].
  Newer version of the Installation Manager was used on the system
  ```

- **Solution:** To work around this error, complete the following steps:

  1. Open the Installation Manager `installRegistry.xml` file, which is in the Installation Manager agent data location. The location of this file depends on your platform. See the Installation Manager Information Center to find out the location on your workstation: http://publib.boulder.ibm.com/infocenter/install/v1r2/index.jsp?topic=/com.ibm.silentinstall12.doc/topics/r_app_data_loc.html.
  2. When you have located the `installRegistry.xml` file, change the following line in the file and the Installation Manager should start correctly:

     ```
     <?installRegistry version='0.0.4'?>
     ```

     to match the following line:

     ```
     <?installRegistry version='0.0.3'?>
     ```

WebSphere Message Broker Launchpad return code indicates that an installation has failed

Before you begin

WebSphere Message Broker Launchpad on Windows:

Procedure

- **Scenario:** You install products by using the WebSphere Message Broker Launchpad. The Launchpad waits for a return code from each installation wizard that it initiates, but the return code indicates that an installation has failed. The Launchpad reports the error and refers you to the documentation for the product that has failed.
- **Explanation:** Most installation wizards roll back from the point of the error and return your system to the state it was in before the failed attempt, and you can therefore try again after you have corrected the error. However, in this scenario, the Launchpad has already installed one or more products successfully before the error occurred, so does not roll back these installations. When you restart the Launchpad, the status of installed products reflects successful installations from the previous invocation.
- **Solution:** To resolve the failed installation, you must either correct the error and restart the Launchpad, or click **Refresh** and clear the selection of the product that failed.

  If you are unable to install the failed product, complete the following steps:

  – Refer to the readme file `readme.html` for late changes to the installation instructions.

  – If the Broker component fails to install, check the contents of the installation log file `mqsi7_install.log`, which is stored in your installation directory.

    If you accepted the default location during installation, this directory is as follows. The default directory includes the version and release of the product that you are installing in the format v.r (version.release):

    Linux `/opt/ibm/mqsi/v.r`

    UNIX `/opt/IBM/mqsi/v.r`

    Windows 32-bit
    `C:\Program Files\IBM\MQSI\v.r`

    Windows 64-bit
    `C:\Program Files\IBM\MQSI\v.r` for the 64-bit version of WebSphere Message Broker

    `C:\Program Files (x86)\IBM\MQSI\v.r` for the 32-bit version of WebSphere Message Broker

    These locations define the default value of *install_dir* on each platform.

  – If the WebSphere Message Broker Toolkit fails to install, check the contents of the installation log file *YYYYMMDD_TIME*`.xml`, where *YYYYMMDD_TIME* is the date and time of installation.

  – If the WebSphere Message Broker Explorer fails to install, check the contents of the installation log file `MBExplorer_install.log`. The log file is stored in the installation directory.

  – If WebSphere MQ fails to install, check the contents of `MQV7_install.`*date_time*`.log` stored in the `temp` directory of your home directory.

  If you are still unable to resolve the problem, contact your IBM Support Center.

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

"Installing the Broker component" on page 267
Use the installation wizard to install the Broker component.

"Installing the WebSphere Message Broker Toolkit" on page 276
Use the installation wizard graphical interface to install the WebSphere Message Broker Toolkit on Windows and Linux on x86.

"Installing by using the Windows Launchpad" on page 262
Use the Windows Launchpad to install the WebSphere Message Broker components and the prerequisite products.

"Uninstalling" on page 331
Remove the Broker component, the WebSphere Message Broker Toolkit, or the WebSphere Message Broker Explorer from your computer.

"Resolving problems when uninstalling"
Work through the advice provided to help you to deal with problems that can arise when the product is uninstalled.

"Finding the latest information" on page 232
Access the latest information for WebSphere Message Broker.

Related reference:

"WebSphere Message Broker logs" on page 6867
WebSphere Message Broker writes information to a number of product-specific logs to report the results of actions that you take.

Related information:

WebSphere Message Broker Requirements

Resolving problems when uninstalling

Work through the advice provided to help you to deal with problems that can arise when the product is uninstalled.

About this task

The following list describes typical problems when uninstalling, with a corresponding solution or workaround:

- Broker component:
  - "Space problems uninstalling on Solaris"
  - "Files left behind after uninstallation completes" on page 3526
  - "The uninstallation process is interrupted" on page 3526

Space problems uninstalling on Solaris

Procedure

- **Scenario:** When you uninstall the Broker component on Solaris on x86-64, the process fails and reports an error that there is no space on the device.
- **Explanation:** The process is trying to use temporary files in the `/var` directory, but insufficient space is available.
- **Solution:** Delete unwanted files from the `/var` directory, then rerun the process.

Files left behind after uninstallation completes

Procedure

- **Scenario:** When you uninstall the WebSphere Message Broker runtime components, the wizard completes successfully, but some files and directories are left behind in the installation directory.
- **Explanation:** Some processes might still be running during the uninstallation, which might be locking the directories.
- **Solution:** When the uninstallation wizard for the WebSphere Message Broker runtime components has completed, manually delete all remaining files and directories in the installation directory.

The uninstallation process is interrupted

Procedure

- **Scenario:** When you uninstall the Broker component on any distributed system, the process is interrupted, for example by a power failure.
- **Solution:** Delete the *install_dir* and all its contents. You can now reinstall if required.

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

"Resolving problems when installing" on page 3517
Work through the advice provided to help you to deal with problems that can arise when the product is installed.

"Uninstalling" on page 331
Remove the Broker component, the WebSphere Message Broker Toolkit, or the WebSphere Message Broker Explorer from your computer.

Related reference:

"WebSphere Message Broker logs" on page 6867
WebSphere Message Broker writes information to a number of product-specific logs to report the results of actions that you take.

Using logs

There are a variety of logs that you can use to help with problem determination and troubleshooting.

About this task

This section describes how to view the various logs available to you with WebSphere Message Broker, and how to interpret the information in those logs. It contains the following topic areas:

Local error log

- "Windows: Viewing the local error log" on page 3527
- "Linux and UNIX systems: Configuring the syslog daemon" on page 3529
- "z/OS: Viewing broker job logs" on page 3530

Eclipse log

- "Viewing the Eclipse error log" on page 3532

There is also a section of reference topics about the various types of log.

Related concepts:

"Logs" on page 6864
If an error is reported by a WebSphere Message Broker component, start your investigations into its causes by looking at the product and systems logs to which information is written during component operation.

Related tasks:

Chapter 13, "Troubleshooting and support," on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

Related reference:

"Local error logs" on page 6867
WebSphere Message Broker components use the local error log (also known as the system log) to record information about major activities within the system. When an error occurs, check the local error log first.

"WebSphere Message Broker logs" on page 6867
WebSphere Message Broker writes information to a number of product-specific logs to report the results of actions that you take.

Windows: Viewing the local error log

The Windows Event Viewer is where WebSphere Message Broker writes records to the local system. Use Windows system facilities to view this log.

Viewing the system log

About this task

The system log contains events logged by the Windows system components. For example, the failure of a driver or other system component to load during startup is recorded in the system log. To view the system log:

Procedure

1. Open a command prompt.
2. At the prompt, type **eventvwr**. This opens the Windows Event Viewer.
3. On Windows XP and Windows Server 2003, click **System Log** in the left pane of this window. This option might be selected by default.

   On Windows Vista and Windows Server 2008, expand the Windows Logs tree in the left pane of this window, and click **System**. This option might be selected by default.

   All the events that have been written to the local system are displayed.

Viewing the application log

About this task

The application log contains events that are logged by applications or programs. For example, a database program might record a file error in the application log. To view the application log:

Procedure

1. Open a command prompt.
2. At the prompt, type **eventvwr**. This opens the Windows Event Viewer.
3. On Windows XP and Windows Server 2003, click **Application Log** in the left pane of this window. This option might be selected by default.

   On Windows Vista and Windows Server 2008, expand the Windows Logs tree in the left pane of this window, and click **Application**. This option might be selected by default.

   All the events that have been written by applications or programs to the local system are displayed.

Interpreting log information

About this task

In both logs, each event is displayed on a separate row, in date and time order (most recent first), with the following information:

- **Type:** The event type, which can be information, a warning, or an error.
- **Date and time:** The date and time when the event was written to the log.
- **Source:** What action has caused the event.
- **Category:** The category of the event. The default category is `none`.
- **Event:** The event number.
- **User:** The name of the user at the time of the event.
- **Computer:** The name of the local machine.

To view an individual log entry:

Procedure

1. Within the system or application log, find the log entry.
2. Right-click the entry.
   - On Windows XP and Windows Server 2003, click **Properties** to open the Information Properties window.

     The window shows a description of the event and a Data section that details bytes or words that were parsed when the record was written to the log.
   - On Windows Vista and Windows Server 2008, click **Events** to open the Event Properties window.

     The window shows a description of the event. Select the Details tab to view bytes or words that were parsed when the record was written to the log.
3. In the Information Properties or Event Properties window, use the up and down arrows to move through the events of the log.
4. To close the Information Properties or Event Properties window, click **OK** to return to the system or application log.

Related concepts:

"Logs" on page 6864
If an error is reported by a WebSphere Message Broker component, start your investigations into its causes by looking at the product and systems logs to which information is written during component operation.

Related tasks:

Chapter 13, "Troubleshooting and support," on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the

problems.

"Using logs" on page 3526
There are a variety of logs that you can use to help with problem determination and troubleshooting.

"Linux and UNIX systems: Configuring the syslog daemon"
On Linux and UNIX systems, all WebSphere Message Broker messages (other than messages that are generated by the command-line utilities) are sent to the syslog subsystem.

"z/OS: Viewing broker job logs" on page 3530
On z/OS, the broker writes messages to the appropriate z/OS system log and job logs. These messages might include information, warning, error, and severe messages to indicate various situations and events.

Related reference:

"Local error logs" on page 6867
WebSphere Message Broker components use the local error log (also known as the system log) to record information about major activities within the system. When an error occurs, check the local error log first.

Linux and UNIX systems: Configuring the syslog daemon

On Linux and UNIX systems, all WebSphere Message Broker messages (other than messages that are generated by the command-line utilities) are sent to the syslog subsystem.

About this task

UNIX You must configure this subsystem so that all diagnostic messages that enable you to monitor the performance and behavior of your broker environment are displayed.

The configuration steps you make to ensure that all relevant messages are displayed depend on the version of Linux and UNIX that you are using. Refer to your operating system documentation relating to syslog (or syslog-ng for some versions of Linux) for information about how to configure the syslog subsystem.

WebSphere Message Broker processes call the syslog commands on the operating system but only those messages that correspond to the filter defined for the output destination are displayed. WebSphere Message Broker messages have:

- A facility of `user`.
- A level of `err`, `warn`, or `info`, depending on the severity of the situation causing the message to be issued.

To record all WebSphere Message Broker messages, create a filter on the `user` facility for messages of level `info` or greater. It is good practice to write these messages to a separate file; there might be a high number of them and they are more likely to be of interest to broker administrators rather than to system administrators.

The following line in a `syslog.conf` file causes all WebSphere Message Broker events to be written to a file `/var/log/user.log`

```
user.info /var/log/user.log
```

Many UNIX systems provide a command-line utility, known as logger, to help you test and refine your configuration of the syslog subsystem.

On UNIX, syslog entries are restricted in length and messages that are sent to the syslog are truncated by the new line character. To record a large amount of data in a log on UNIX, set the `Destination` property on the Trace node to `File` or `User Trace` instead of `Local Error Log`.

What to do next

See the documentation for your operating system.

Related concepts:

"Logs" on page 6864
If an error is reported by a WebSphere Message Broker component, start your investigations into its causes by looking at the product and systems logs to which information is written during component operation.

Related tasks:

Chapter 13, "Troubleshooting and support," on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

"Using logs" on page 3526
There are a variety of logs that you can use to help with problem determination and troubleshooting.

"Windows: Viewing the local error log" on page 3527
The Windows Event Viewer is where WebSphere Message Broker writes records to the local system. Use Windows system facilities to view this log.

"z/OS: Viewing broker job logs"
On z/OS, the broker writes messages to the appropriate z/OS system log and job logs. These messages might include information, warning, error, and severe messages to indicate various situations and events.

Related reference:

"Local error logs" on page 6867
WebSphere Message Broker components use the local error log (also known as the system log) to record information about major activities within the system. When an error occurs, check the local error log first.

z/OS: Viewing broker job logs

On z/OS, the broker writes messages to the appropriate z/OS system log and job logs. These messages might include information, warning, error, and severe messages to indicate various situations and events.

Understanding the broker address spaces

About this task

The broker runs in multiple address spaces. A single control address space is always running when the broker is active, and is responsible for communicating with the WebSphere Message Broker Toolkit.

Each execution group within the broker is mapped to its own address space, and as these execution groups start and stop, the corresponding address spaces are started and stopped. The control address space is assigned a JOBNAME and STEPNAME, which is the same as the broker name. The execution groups have a JOBNAME that is also the same as the broker name, and a STEPNAME that matches the last seven characters of the execution group name.

Viewing the z/OS system console log
About this task

The broker writes all its messages to the z/OS system console log. You can see messages from all address spaces running on the z/OS system in this log. It is easy to identify jobs that are associated with the broker in the console log, because of the naming of broker address spaces. By using the console log, you can see the order of event reporting for different products. This information can be helpful for cross-product problem determination.

Viewing the broker job logs
About this task

The broker control address space, and each of the execution group address spaces, has its own job log. When you select the job log for the appropriate address space, you can see all messages relating to that address space. Use this option in a busy system where the system console log might have many messages from different products obscuring the information in which you are interested.

Interpreting log information
About this task

In both logs, each message is displayed on its own, in date and time order; it might span multiple lines, if necessary. For each message, the following information is available:

- **Date and time**: The exact date and time when the event was written to the log.
- **JOBID**: The started task job identifier of the address space.
- **Message Number**: The message number that identifies whether the event is information, a warning, or an error, with diagnostic text.
- **JOBNAME**: The JOBNAME of the address space issuing the message. This name is always the same as the broker name.
- **STEPNAME**: The STEPNAME of the address space that is issuing the message. For the control address space, this name is the same as the broker name; for execution groups, it is the same as the last eight characters of the execution group that is issuing the message.
- **PROCSTEP**: The procedure STEPNAME, which equals BROKER for the main broker control address space. For execution group address spaces, this is either **EGNOENV**, if there is not an execution group specific profile, or **EGENV** if there is an execution group specific profile.

Related concepts:

"Logs" on page 6864
If an error is reported by a WebSphere Message Broker component, start your investigations into its causes by looking at the product and systems logs to which information is written during component operation.

Related tasks:

"Using logs" on page 3526
There are a variety of logs that you can use to help with problem determination and troubleshooting.

"Linux and UNIX systems: Configuring the syslog daemon" on page 3529
On Linux and UNIX systems, all WebSphere Message Broker messages (other than messages that are generated by the command-line utilities) are sent to the syslog subsystem.

"Windows: Viewing the local error log" on page 3527
The Windows Event Viewer is where WebSphere Message Broker writes records to the local system. Use Windows system facilities to view this log.

Related reference:

"Local error logs" on page 6867
WebSphere Message Broker components use the local error log (also known as the system log) to record information about major activities within the system. When an error occurs, check the local error log first.

"WebSphere Message Broker logs" on page 6867
WebSphere Message Broker writes information to a number of product-specific logs to report the results of actions that you take.

Viewing the Eclipse error log

The Eclipse error log captures internal errors that are caused by the operating system or your code.

About this task

To view the Eclipse error log:

Procedure

1. Switch to the Plug-in Development perspective.
2. From the main menu, select **Window** > **Show view** > **Other**. Then select **General** > **Error Log**.

   The error log is displayed, showing the following information for each error:
   - The status of the error (for example, error or warning)
   - A brief description of the error
   - From which plug-in the error derived
   - The date and time that the error was produced
3. If an error has a plus sign (+) at the start of it, it is a complex problem, and there are a number of errors contributing to it. Click the plus sign to view the individual errors.
4. To see the details of a particular problem, double-click the entry in the Problems view. A separate window is displayed, showing more details of the error.

Related concepts:

"Logs" on page 6864
If an error is reported by a WebSphere Message Broker component, start your investigations into its causes by looking at the product and systems logs to which information is written during component operation.

Related tasks:

"Using logs" on page 3526
There are a variety of logs that you can use to help with problem determination and troubleshooting.

Related reference:

"WebSphere Message Broker logs" on page 6867
WebSphere Message Broker writes information to a number of product-specific logs to report the results of actions that you take.

Using trace

You can use different types of trace to help you with problem determination and troubleshooting.

About this task

How to use the optional trace.

For user trace:

- "Starting user trace" on page 3197
- "Checking user trace options" on page 3199
- "Changing user trace options" on page 3201
- "Retrieving user trace" on page 3204
- "Stopping user trace" on page 3202

For service trace:

- "Starting service trace" on page 3534
- "Checking service trace options" on page 3537
- "Changing service trace options" on page 3538
- "Retrieving service trace" on page 3542
- "Stopping service trace" on page 3540

For both types of trace:

- "Formatting trace" on page 3543
- "Interpreting trace" on page 3546
- "Clearing old information from trace files" on page 3548
- "Changing trace settings from the WebSphere Message Broker Explorer" on page 3549

Other types of trace:

- "ODBC trace" on page 3551
- "Administration API (CMP) trace" on page 3554
- "Switching Trace nodes on and off" on page 3555

You can also use the "IBM Support Assistant Data Collector" on page 3565 to help with data collection.

Related concepts:

"Trace" on page 6871
If you cannot get enough information about a particular problem from the entries that are available in the various logs, the next troubleshooting method to consider is using trace. Trace provides more details about what is happening while code runs. The information produced from trace is sent to a specified trace record, so that you or IBM support personnel can analyze it to discover the cause of your problem.

Related tasks:

Chapter 13, "Troubleshooting and support," on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

"Making initial checks" on page 3347
Before you start problem determination in detail, consider whether there is an obvious cause of the problem, or an area of investigation that is likely to give useful results. This approach to diagnosis can often save a lot of work by highlighting a simple error, or by narrowing down the range of possibilities.

"Contacting your IBM Support Center" on page 3563
If you cannot resolve problems that you find when you use WebSphere Message Broker, or if you are directed to do so by an error message generated by WebSphere Message Broker, you can request assistance from your IBM Support Center.

"IBM Support Assistant Data Collector" on page 3565
You can collect diagnostic documents by using IBM Support Assistant Data Collector, and submit a problem report to IBM. IBM Support Assistant Data Collector is included with your WebSphere Message Broker installation.

Related reference:

"User trace" on page 6873
User trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Administration log. User trace is inactive by default; you must activate it explicitly by using a command, or by selecting options in the WebSphere Message Broker Toolkit.

"Service trace" on page 6875
Service trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Event Log or to user trace. Service trace is inactive by default; you must activate it explicitly by using a command.

"**mqsichangetrace** command" on page 3822
Use the **mqsichangetrace** command to set the tracing characteristics for a broker.

"**mqsiformatlog** command" on page 3880
Use the **mqsiformatlog** command to process the XML log created by **mqsireadlog**. The command retrieves and formats any messages that the XML log contains into a form suitable for the locale of the user who runs the command.

"**mqsireadlog** command" on page 3905
Use the **mqsireadlog** command to retrieve trace records for the specified component.

"**mqsireporttrace** command" on page 3947
Use the **mqsireporttrace** command to display the trace options currently in effect. Trace can be run only against a broker, or any of its resources.

"Database facilities" on page 6891
The database products used by WebSphere Message Broker also record information that might be useful if you have any problems with their access.

Starting service trace

Service trace is used to get detailed information about your environment for use by your IBM Support Center.

About this task

Activate service traces only when you receive an error message that instructs you to start service trace, or when directed to do so by your IBM Support Center

Use the **mqsichangetrace** command to start WebSphere Message Broker service trace facilities.

You can select only one broker on each invocation of the command, but you can activate concurrent traces for more than one broker by invoking the command more than once.

To limit the scope of a trace, you must specify the individual broker that you want to trace.

If the trace cannot be associated with a specific component, the component name part of the file name is set to `utility`; for example, when tracing a command such as **mqsilist**, when no arguments are used.

To trace the **mqsichangeresourcestats**, **mqsicreateexecutiongroup**, **mqsideleteexecutiongroup**, **mqsideploy**, **mqsilist**, **mqsimode**, **mqsireloadsecurity**, **mqsireportresourcestats**, **mqsistartmsgflow**, and **mqsistopmsgflow** commands, use the **-v** parameter. This parameter takes an argument that is the name of the file to which trace records are written. For example, the following command outputs trace of the **mqsistartmsgflow** command to the specified file:

```
mqsistartmsgflow MB7BROKER -e eg1 -m simpleflow -v .\trace.txt
```

If you want to trace the command executable files themselves, set the environment variables MQSI_UTILITY_TRACE and MQSI_UTILITY_TRACESIZE before you run the command.

MQSI_UTILITY_TRACE
: Set this variable to `normal` for a basic level of trace, or `debug` for a fuller trace.

MQSI_UTILITY_TRACESIZE
: The size, in kilobytes, of the binary trace file. See the **-c** parameter of the **mqsichangetrace** command.

Ensure that you reset these variables when the command that you are tracing has completed. If you do not do so, all subsequent commands are also traced, and their performance is therefore degraded.

To enable service trace of your CMP applications, take one of the following steps:

- Call the method BrokerProxy.enableAdministrationAPITracing(String filename).
- Before running your CMP application, set the environment variable MQSI_CMP_TRACE to the name of the file to which trace is sent.

You can also use the WebSphere Message Broker Explorer to activate service trace on execution groups and message flows.

To enable service trace for execution groups or messages flows in the WebSphere Message Broker Explorer:

1. In the Navigator view, expand the Brokers folder and right-click the execution group or message flow with which you want to work.
2. Click **Service Trace** > **Normal** or **Service Trace** > **Debug** to select the level of service trace that you require.

Example: starting service trace for the broker

About this task

To start debug level service tracing for the broker `A` on distributed systems, enter the following command:

```
mqsichangetrace BrokerA -t -b -l debug
```

where:

-t specifies service trace
-b specifies that trace for the `agent` subcomponent of the specified component is to be started
-l specifies the level of trace (in this case, debug)

z/OS On z/OS, enter the following command:

```
F MQPIBRK,ct t=yes, b=yes, l=debug
```

Example: starting service trace for an execution group

About this task

To start debug level service tracing for an execution group `EG1` on broker `A` on distributed systems, enter the following command:

```
mqsichangetrace BrokerA -t -e EG1 -l debug -m fast -c 200000 -r
```

where:

-t specifies service trace
-l specifies the level of trace (in this case, debug)
-m specifies the way trace information is to be buffered (in this case, fast)
-c specifies the size of the trace file in KB (in this case, 200000)
-r specifies that the trace file is reset

z/OS On z/OS, enter the following command:

```
F MQPIBRK,ct t=yes, e=EG1, l=debug, m=fast, c=20000, r=yes
```

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

"Trace" on page 6871
If you cannot get enough information about a particular problem from the entries that are available in the various logs, the next troubleshooting method to consider is using trace. Trace provides more details about what is happening while code runs. The information produced from trace is sent to a specified trace record, so that you or IBM support personnel can analyze it to discover the cause of your problem.

Related tasks:

Chapter 13, "Troubleshooting and support," on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

"Using trace" on page 3533
You can use different types of trace to help you with problem determination and troubleshooting.

"Starting user trace" on page 3197
Use user trace for debugging your applications; you can trace brokers, execution groups, and deployed message flows. Start user trace facilities using the **mqsichangetrace** command or the WebSphere Message Broker Explorer.

"Checking service trace options" on page 3537
Use the **mqsireporttrace** command or the WebSphere Message Broker Explorer to check what tracing options are currently active for your brokers.

"Stopping service trace" on page 3540
Use the **mqsichangetrace** command or the WebSphere Message Broker Explorer to stop an active service trace.

"Contacting your IBM Support Center" on page 3563
If you cannot resolve problems that you find when you use WebSphere Message Broker, or if you are directed to do so by an error message generated by WebSphere Message Broker, you can request assistance from your IBM Support Center.

Related reference:

"Service trace" on page 6875
Service trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Event Log or to user trace. Service trace is inactive by default; you must activate it explicitly by using a command.

"**mqsichangetrace** command" on page 3822
Use the **mqsichangetrace** command to set the tracing characteristics for a broker.

Checking service trace options

Use the **mqsireporttrace** command or the WebSphere Message Broker Explorer to check what tracing options are currently active for your brokers.

About this task

Specify the component for which the check is required. The command responds with the current trace status for the component that you have specified.

Example: checking service trace options for an execution group using the WebSphere Message Broker Explorer

About this task

To check what options are currently set for the execution group by using the WebSphere Message Broker Explorer:

Procedure

1. In the Navigator view, expand the Brokers folder and right-click the execution group with which you want to work.
2. Click **Service** to view user trace settings for your execution group.

Example: checking service trace options for a broker

About this task

To check what options are currently set for the broker, on distributed systems, enter the following command:

```
mqsireporttrace brokerA -t
```

where **-t** specifies service trace.

z/OS On z/OS, enter the following command:

```
F MQP1BRK,reporttrace t=yes
```

Results

If you have started tracing by following the example in "Starting service trace" on page 3534, the response to the **mqsireporttrace** command is:

```
BIP8098I: Trace level: debug, mode: safe, size: 1024 KB
BIP8071I: Successful command completion
```

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

"Trace" on page 6871
If you cannot get enough information about a particular problem from the entries that are available in the various logs, the next troubleshooting method to consider is using trace. Trace provides more details about what is happening while code runs. The information produced from trace is sent to a specified trace record, so that you or IBM support personnel can analyze it to discover the cause of your problem.

Related tasks:

Chapter 13, "Troubleshooting and support," on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

"Using trace" on page 3533
You can use different types of trace to help you with problem determination and troubleshooting.

"Checking user trace options" on page 3199
Use the **mqsireporttrace** command or the WebSphere Message Broker Explorer to check what tracing options are currently active for your brokers, execution groups and message flows.

"Starting service trace" on page 3534
Service trace is used to get detailed information about your environment for use by your IBM Support Center.

"Changing service trace options"
Use the **mqsichangetrace** command or the WebSphere Message Broker Explorer to change the service trace options that you have set.

Related reference:

"Service trace" on page 6875
Service trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Event Log or to user trace. Service trace is inactive by default; you must activate it explicitly by using a command.

"**mqsireporttrace** command" on page 3947
Use the **mqsireporttrace** command to display the trace options currently in effect. Trace can be run only against a broker, or any of its resources.

Changing service trace options

Use the **mqsichangetrace** command or the WebSphere Message Broker Explorer to change the service trace options that you have set.

Example: changing service trace from debug to normal on an execution group using the WebSphere Message Broker Explorer

About this task

To change from a debug level of trace to a normal level on the execution group using the WebSphere Message Broker Explorer:

Procedure

1. In the Navigator view, expand the Brokers folder and right-click the execution group with which you want to work.
2. Click **Service** > **Normal**.

Example: changing service trace from debug to normal

About this task

To change from a debug level of trace to a normal level on the broker, on distributed systems, enter the following command:

```
mqsichangetrace BrokerA -t -b -l normal
```

where:

-t specifies service trace
-b specifies that tracing for the `agent` subcomponent of the specified component is to be changed
-l specifies the level of trace (in this case, changing it to normal)

z/OS On z/OS, enter the following command:

```
F MQP1BRK,ct t=yes, b=yes, l=normal
```

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

"Trace" on page 6871
If you cannot get enough information about a particular problem from the entries that are available in the various logs, the next troubleshooting method to consider is using trace. Trace provides more details about what is happening while code runs. The information produced from trace is sent to a specified trace record, so that you or IBM support personnel can analyze it to discover the cause of your problem.

Related tasks:

Chapter 13, "Troubleshooting and support," on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

"Using trace" on page 3533
You can use different types of trace to help you with problem determination and troubleshooting.

"Changing user trace options" on page 3201
Use the **mqsichangetrace** command to change the trace options that you have set. You can also use the WebSphere Message Broker Explorer to change the trace options for execution groups and assigned message flows.

"Starting service trace" on page 3534
Service trace is used to get detailed information about your environment for use by your IBM Support Center.

"Checking service trace options" on page 3537
Use the **mqsireporttrace** command or the WebSphere Message Broker Explorer to check what tracing options are currently active for your brokers.

Related reference:

"Service trace" on page 6875
Service trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Event Log or to user trace. Service trace is inactive by default; you must activate it explicitly by using a command.

"**mqsichangetrace** command" on page 3822
Use the **mqsichangetrace** command to set the tracing characteristics for a broker.

Stopping service trace

Use the **mqsichangetrace** command or the WebSphere Message Broker Explorer to stop an active service trace.

Procedure

- Use the **mqsichangetrace** command with a trace level of `none` to stop an active trace. This action stops the trace activity for the component that you specify on the command. It does not affect active traces on other components. For example, if you stop tracing on the execution group `test`, an active trace on another execution group continues.
- Click **Service Trace** > **None** on an execution group or message flow, in the WebSphere Message Broker Explorer, to stop an active service trace on the selected object.

Results

If you redeploy a component from the WebSphere Message Broker Toolkit, trace for that component is returned to its default setting of `none`.

Example: stopping service trace on an execution group, using the WebSphere Message Broker Explorer

About this task

To stop the service trace using the WebSphere Message Broker Explorer:

Procedure

1. In the Navigator view, expand the Brokers folder and right-click the execution group with which you want to work.
2. Click **Service TraceNone**.

Example: stopping service trace on an execution group, using the `mqsichangetrace` command

About this task

To stop debug level service tracing for an execution group`EG1` on broker `A` on distributed systems, enter the command

```
mqsichangetrace BrokerA -t -e EG1 -l none
```

where:

-t specifies service trace
-l specifies the level of trace (in this case, none)

z/OS On z/OS, enter the command

```
F MQPIBRK,ct t=yes, b=yes, l=none
```

Example: stopping service trace on the broker

About this task

To stop the trace started by the command shown in "Starting service trace" on page 3534, on distributed systems, enter the following command:

```
mqsichangetrace BrokerA -t -b -l none
```

where:

-t specifies service trace
-b specifies that trace for the `agent` subcomponent of the specified component is to be stopped
-l specifies the level of trace (in this case, none)

z/OS On z/OS, enter the following command:

```
F MQP1BRK,ct t=yes, b=yes, l=none
```

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

"Trace" on page 6871
If you cannot get enough information about a particular problem from the entries that are available in the various logs, the next troubleshooting method to consider is using trace. Trace provides more details about what is happening while code runs. The information produced from trace is sent to a specified trace record, so that you or IBM support personnel can analyze it to discover the cause of your problem.

Related tasks:

Chapter 13, "Troubleshooting and support," on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

"Using trace" on page 3533
You can use different types of trace to help you with problem determination and troubleshooting.

"Stopping user trace" on page 3202
Use user trace for debugging your applications; you can trace brokers, execution groups, and deployed message flows. Stop user trace facilities by using the **mqsichangetrace** command or the WebSphere Message Broker Explorer.

"Starting service trace" on page 3534
Service trace is used to get detailed information about your environment for use by your IBM Support Center.

"Clearing old information from trace files" on page 3548
If the component that you are tracing has stopped, you can delete its trace files from the log subdirectory of the WebSphere Message Broker home directory.

Related reference:

"Service trace" on page 6875
Service trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Event Log or to user trace. Service trace is inactive by default; you must activate it explicitly by using a command.

"**mqsichangetrace** command" on page 3822
Use the **mqsichangetrace** command to set the tracing characteristics for a broker.

Retrieving service trace

Use the **mqsireadlog** command to access the trace information recorded by the service trace facilities.

Procedure

This command retrieves the trace details according to parameters that you specify on the command, and writes the requested records to a file, or to the command-line window, in XML format.

Example: retrieving service trace information in XML format

Procedure

To retrieve information for the service trace activated with the **mqsichangetrace** command, and write it to an output file, on distributed systems, enter the following command:

```
mqsireadlog brokerName -t -b agent -f -o /path/to/strace.xml
```

where:

-t specifies service trace.

-b *agent* specifies that trace for the `agent` subcomponent of the specified component is to be retrieved.

-f specifies that the log file is to be read directly from the file system (this flag is mandatory for service trace).

-o specifies the output file (in this case, `/path/to/strace.xml`.) If you do not specify a path to the file, the file is created in the current directory. Ensure that there is enough space for the output file.

Results

This command sends a log request to the broker to retrieve the service trace log, and stores the responses in the specified file. You can view this file using a plain text editor.

Example: retrieving service trace information in XML format for an execution group

Procedure

To retrieve information for the service trace activated with the **mqsichangetrace** command and associated with execution group EG1 on broker A, and write it to an output file, on distributed systems, enter the following command:

```
mqsireadlog BrokerA -t -e EG1 -f -o /path/to/strace.xml
```

where:

-t specifies service trace.

-f specifies that the log file is to be read directly from the file system (this flag is mandatory for service trace).

-o specifies the output file (in this case, `/path/to/strace.xml`.) If you do not specify a path to the file, the file is created in the current directory. Ensure that there is enough space for the output file.

Results

This command sends a log request to the broker to retrieve the service trace log, and stores the responses in the specified file. You can view this file using a plain text editor.

Related concepts:

"Trace" on page 6871
If you cannot get enough information about a particular problem from the entries that are available in the various logs, the next troubleshooting method to consider is using trace. Trace provides more details about what is happening while code runs. The information produced from trace is sent to a specified trace record, so that you or IBM support personnel can analyze it to discover the cause of your problem.

Related tasks:

Chapter 13, "Troubleshooting and support," on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

"Using trace" on page 3533
You can use different types of trace to help you with problem determination and troubleshooting.

"Retrieving user trace" on page 3204
Use the **mqsireadlog** command to access the trace information that is recorded by the user trace facilities.

"Formatting trace"
Use the **mqsiformatlog** command to format trace information.

"Interpreting trace" on page 3546
Use the information in a formatted trace file to identify unexpected behavior.

Related reference:

"Service trace" on page 6875
Service trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Event Log or to user trace. Service trace is inactive by default; you must activate it explicitly by using a command.

"**mqsireadlog** command" on page 3905
Use the **mqsireadlog** command to retrieve trace records for the specified component.

Formatting trace

Use the **mqsiformatlog** command to format trace information.

About this task

The trace information that is generated by the **mqsireadlog** command is not easy to read unless you use an XML viewer (such as an Internet browser) or an XML editor that understands the document type descriptor (DTD) in the file.

Procedure

WebSphere Message Broker provides the command **mqsiformatlog** to format the trace information to a flat file, so that you can view it using a text editor.

Results

The **mqsiformatlog** command takes a file generated by the **mqsireadlog** command as input, and flattens the XML log into structured records. It also retrieves the inserts for the XML message in your current locale. You can direct the formatted output to a file, or view it in the command line window.

Each trace entry contains a time stamp and a WebSphere Message Broker message that contains a number (for example, BIP2622) and a text string containing variable inserts.

Example: formatting service trace information for an execution group

About this task

To format the service trace file for an execution group, enter the command

```
mqsiformatlog -i </path/to/output.xml> -o </path/to/output.txt>
```

where:

-i specifies the path to the input file that contains the unformatted service trace information for the execution group

-o specifies the path to the output file that is to contain the formatted service trace information

Example: formatting user trace information on Windows

About this task

Windows On Windows, to format the trace file that is created in "Starting user trace" on page 3197, enter the command

```
mqsiformatlog -i trace.xml -o formattrace.log
```

where:

-i specifies the input file (in this case, trace.xml)

-o specifies the output file (in this case, formattrace.log)

This command reads the trace information in the file trace.xml, formats it, and writes it to the file formattrace.log. The following example shows a portion of the output of the **mqsiformatlog** command for a normal level trace file.

```
Timestamps are formatted in local time, 330 minutes past GMT.

2003-06-19 11:30:29.795999     2852   UserTrace   BIP2632I: Message received and
 propagated  to 'out' terminal of MQ Input node 'Video_Test.VIDEO_XML_IN'.
2003-06-19 11:30:29.795999     2852   UserTrace   BIP6060I: Parser type 'Properties'
 created on behalf of node 'Video_Test.VIDEO_XML_IN' to handle portion of incoming
 message of length
 0 bytes beginning at offset '0'.
2003-06-19 11:30:29.795999     2852   UserTrace   BIP6061I: Parser type 'MQMD'
 created on behalf of node 'Video_Test.VIDEO_XML_IN' to handle portion of incoming
 message of length '364' bytes beginning at offset '0'. Parser type selected based
 on value 'MQHMD' from previous parser.
2003-06-19 11:30:29.795999     2852   UserTrace   BIP6061I: Parser type 'MRM'
 created on behalf of node 'Video_Test.VIDEO_XML_IN' to handle portion of incoming
 message of length '650' bytes beginning at offset '364'. Parser type selected based
 on value 'MRM' from previous parser.
2003-06-19 11:30:29.795999     2852   UserTrace   BIP2537I: Node 'Video_Test.Extract
 Fields':
```

```
 Executing statement 'BEGIN ... END;' at (.Video_Test_Compute.Main, 2.2).
2003-06-19 11:30:29.795999     2852   UserTrace   BIP2537I: Node 'Video_Test.Extract
 Fields':
 Executing statement 'SET OutputRoot = InputRoot;' at (.Video_Test_Compute.Main, 7.3).
2003-06-19 11:30:29.795999     2852   UserTrace   BIP2538I: Node 'Video_Test.Extract
 Fields':
 Evaluating expression 'InputRoot' at (.Video_Test_Compute.Main, 7.20).
2003-06-19 11:30:29.795999     2852   UserTrace   BIP2568I: Node 'Video_Test.Extract
 Fields':
 Performing tree copy of 'InputRoot' to 'OutputRoot'.

...

2003-06-19 11:30:29.827999     2852   UserTrace   BIP4124I: Message propagated to
 'out' terminal of Compute node 'Video_Test.Extract Fields'.
2003-06-19 11:30:29.827999     2852   UserTrace   BIP2638I: The MQ Output node
 'Video_Test.VIDEO_OUT' attempted to write a message to queue 'VIDEO_OUT' connected
 to queue manager ''. The MQCC was '0' and the MQRC was '0'.
2003-06-19 11:30:29.827999     2852   UserTrace   BIP2622I: Message successfully
 output by output node 'Video_Test.VIDEO_OUT' to queue 'VIDEO_OUT' on queue manager ''.

Threads encountered in this trace:
  2852
```

Related concepts:

"Trace" on page 6871
If you cannot get enough information about a particular problem from the entries that are available in the various logs, the next troubleshooting method to consider is using trace. Trace provides more details about what is happening while code runs. The information produced from trace is sent to a specified trace record, so that you or IBM support personnel can analyze it to discover the cause of your problem.

Related tasks:

Chapter 13, "Troubleshooting and support," on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

"Using trace" on page 3533
You can use different types of trace to help you with problem determination and troubleshooting.

"Retrieving user trace" on page 3204
Use the **mqsireadlog** command to access the trace information that is recorded by the user trace facilities.

"Retrieving service trace" on page 3542
Use the **mqsireadlog** command to access the trace information recorded by the service trace facilities.

"Interpreting trace" on page 3546
Use the information in a formatted trace file to identify unexpected behavior.

"Clearing old information from trace files" on page 3548
If the component that you are tracing has stopped, you can delete its trace files from the log subdirectory of the WebSphere Message Broker home directory.

Related reference:

"User trace" on page 6873
User trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Administration log. User trace is inactive by default; you must activate it explicitly by using a command, or by selecting options in the WebSphere Message Broker Toolkit.

"Service trace" on page 6875
Service trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Event Log or to user trace. Service trace is inactive by default; you must activate it explicitly by using a command.

"**mqsiformatlog** command" on page 3880
Use the **mqsiformatlog** command to process the XML log created by **mqsireadlog**. The command retrieves and formats any messages that the XML log contains into a form suitable for the locale of the user who runs the command.

Interpreting trace

Use the information in a formatted trace file to identify unexpected behavior.

About this task

A formatted log file, like the one in "Formatting trace" on page 3543, contains a sequence of WebSphere Message Broker messages. These messages record the activity in a specific part of the system (the part that you identify when you start the trace). You can use this sequence to understand what is happening, and to check that the behavior that is recorded is what you are expecting.

For example, message flow trace records the path that a message takes through the message flow. You can see why decisions result in this path (where a choice is available).

Procedure

1. Verify that the trace file is complete.

   If the size of the trace log is too small to contain all events, trace output continues at the start of the trace log, overwriting existing entries. This is known as *wrapping*.

   An indication that a trace has wrapped is the relationship between the first and last timestamp in it, and the time that trace was enabled. For example, assume that you start tracing at 10:15, and collect the trace at 10:30. If the trace timestamps run from 10:20 to 10:30, it is probable that trace wrapped. Of course it could mean that nothing happened between 10:15 and 10:20.

   Examine the trace and decide whether the beginning of it makes sense, and whether it looks complete. For example, if you want to trace the passage of three messages through a flow, and trace starts half way through the second message, it could have wrapped, or trace might not have been enabled early enough.
2. If trace has wrapped, increase the size of the trace file, and rerun trace. For information on trace settings, see "**mqsichangetrace** command" on page 3822.
3. If you see unexpected behavior in a message flow or execution group, use this trace information to check the actions that have been taken and identify the source of an error or other discrepancy.

Results

The messages contain identifiers for the resources that are being traced, for example the execution groups and message flows. The identifier that is given is typically the label (the name) that you gave to the resource when you defined it.

Here is an extract from a user trace file. In the example, each column has been labeled:

```
Timestamp                 Thread ID Trace type  Message
2005-07-12 16:17:18.242605   5344   UserTrace   BIP2537I: Node 'Reply.MapToRequestor':
                                                Executing statement   ''SET I = I + 1;''
                                                at ('.MapToRequestor.CopyMessageHeaders',
                                                '6.4').
2005-07-12 16:17:18.242605   5344   UserTrace   BIP2539I: Node 'Reply.MapToRequestor':
                                                Evaluating expression ''I'' at
                                                ('.MapToRequestor.CopyMessageHeaders',
                                                '6.12'). This resolved to ''I''. The
                                                result was ''1''.
2005-07-12 16:17:18.242605   5344   UserTrace   BIP2539I: Node 'Reply.MapToRequestor':
                                                Evaluating expression ''I + 1'' at
                                                ('.MapToRequestor.CopyMessageHeaders',
                                                '6.14'). This resolved to ''1 + 1''.
                                                The result was ''2''.
2005-07-12 16:17:18.242605   5344   UserTrace   BIP2566I: Node 'Reply.MapToRequestor':
                                                Assigning value ''2'' to field / variable
                                                ''I''.
```

References such as '6.12' apply to the row and column number within a function that specify the location of the command that is being executed; in this case, row 6, column 12.

Related concepts:

"Trace" on page 6871
If you cannot get enough information about a particular problem from the entries that are available in the various logs, the next troubleshooting method to consider is using trace. Trace provides more details about what is happening while code runs. The information produced from trace is sent to a specified trace record, so that you or IBM support personnel can analyze it to discover the cause of your problem.

Related tasks:

Chapter 13, "Troubleshooting and support," on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

"Using trace" on page 3533
You can use different types of trace to help you with problem determination and troubleshooting.

"Retrieving user trace" on page 3204
Use the **mqsireadlog** command to access the trace information that is recorded by the user trace facilities.

"Retrieving service trace" on page 3542
Use the **mqsireadlog** command to access the trace information recorded by the service trace facilities.

"Formatting trace" on page 3543
Use the **mqsiformatlog** command to format trace information.

"Resolving problems with user-defined extensions" on page 3511
Advice for dealing with some common problems that can arise when you work with user-defined extensions

Related reference:

"User trace" on page 6873
User trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Administration log. User trace is inactive by default; you must activate it explicitly by using a command, or by selecting options in the WebSphere Message Broker Toolkit.

"Service trace" on page 6875
Service trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Event Log or to user trace. Service trace is inactive by default; you must activate it explicitly by using a command.

Clearing old information from trace files

If the component that you are tracing has stopped, you can delete its trace files from the log subdirectory of the WebSphere Message Broker home directory.

About this task

If you are tracing an execution group, you can use the **-r** parameter of the **mqsichangetrace** command to reset (clear) the trace log (the **-r** parameter can be specified only if you specify the **-e** parameter). You might clear the log when you start a new trace, to ensure that all the records on the log are unique to the new trace.

Example: clearing the user trace log for the default execution group

About this task

To clear the user trace log for the default execution group, on distributed systems, enter the command: **mqsichangetrace WBRK_BROKER -u -e default -r** where:

WBRK_BROKER specifies the name of the broker
-u specifies user trace
-e specifies the execution group (in this case the default execution group)
-r clears the trace log

z/OS On z/OS, enter the command `F MQP1BRK,ct u=yes, e='default', r=yes`

Related concepts:

"Trace" on page 6871
If you cannot get enough information about a particular problem from the entries that are available in the various logs, the next troubleshooting method to consider is using trace. Trace provides more details about what is happening while code runs. The information produced from trace is sent to a specified trace record, so that you or IBM support personnel can analyze it to discover the cause of your problem.

Related tasks:

Chapter 13, "Troubleshooting and support," on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

"Using trace" on page 3533
You can use different types of trace to help you with problem determination and troubleshooting.

"Starting user trace" on page 3197
Use user trace for debugging your applications; you can trace brokers, execution groups, and deployed message flows. Start user trace facilities using the **mqsichangetrace** command or the WebSphere Message Broker Explorer.

"Starting service trace" on page 3534
Service trace is used to get detailed information about your environment for use by your IBM Support Center.

Related reference:

"Local error logs" on page 6867
WebSphere Message Broker components use the local error log (also known as the system log) to record information about major activities within the system. When an error occurs, check the local error log first.

"User trace" on page 6873
User trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Administration log. User trace is inactive by default; you must activate it explicitly by using a command, or by selecting options in the WebSphere Message Broker Toolkit.

"Service trace" on page 6875
Service trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Event Log or to user trace. Service trace is inactive by default; you must activate it explicitly by using a command.

"**mqsichangetrace** command" on page 3822
Use the **mqsichangetrace** command to set the tracing characteristics for a broker.

Changing trace settings from the WebSphere Message Broker Explorer

Collect trace information, in addition to user and service trace, by selecting options in the WebSphere Message Broker Explorer.

About this task

- Trace for the broker and WebSphere Message Broker Explorer components.
- Trace for the Administration API (also known as the CMP API)

The following sections tell you how to change the settings for these types of trace.

Changing the WebSphere MQ Java Client trace settings

About this task

You control tracing of the WebSphere MQ Java Client from the WebSphere Message Broker Explorer:

Procedure

1. Click **Window** > **Preferences**.
2. In the left menu, expand **Broker Explorer**.
3. Click **Service Trace**.
4. In the **WebSphere MQ Java Client** section, select the check box to enable tracing on the WebSphere MQ Java Client. The default file to which trace is written is `C:/Documents and Settings/`*`userid`*`/Application Data/IBM/MQ Explorer/.metadata/MQClientTraceEnabled.log` where *userid* is your user ID.
5. Click **OK** to apply your changes and to close the Preferences window.

Changing the CMP trace settings

About this task

You control tracing of the CMP API from the WebSphere Message Broker Explorer:

Procedure

1. Click **Window** > **Preferences**.
2. In the left menu, expand **Broker Explorer**.
3. Click **Service Trace**.
4. In the **CMP Administration API** section, select the check box to enable tracing on the CMP. The default file to which trace is written is `C:/Documents and Settings/`*`userid`*`/Application Data/IBM/MQ Explorer/.metadata/AdminAPITrace.log` where *userid* is your user ID.
5. Click **OK** to apply your changes and to close the Preferences window.

Changing the Broker Explorer trace settings

About this task

You can control tracing of the WebSphere Message Broker Explorer using the Broker Explorer Preferences page:

Procedure

1. Click **Window** > **Preferences**.
2. In the left menu, expand **Broker Explorer**.
3. Click **Service Trace**.
4. Select the check box for the component that you want to trace, and select a location for the log file for the trace. The default location for the trace files in is the WebSphere MQ Explorer workspace directory.
5. Set a value for the log file size in the Maximum Broker Explorer Trace File Size field. When the size of the log file you set is exceeded, the original log file is copied to `filename.ext.bak`. A new log file with the selected name is started.
6. Click **OK** to apply your changes and to close the Preferences window.

Related concepts:

"Trace" on page 6871
If you cannot get enough information about a particular problem from the entries that are available in the various logs, the next troubleshooting method to consider is using trace. Trace provides more details about what is happening while code runs. The information produced from trace is sent to a specified trace record, so that you or IBM support personnel can analyze it to discover the cause of your problem.

"The Administration API for WebSphere Message Broker" on page 54
The Administration API for WebSphere Message Broker is a programming interface that your applications can use to control brokers and their resources through a remote interface.

Related tasks:

"Using trace" on page 3533
You can use different types of trace to help you with problem determination and troubleshooting.

"Starting user trace" on page 3197
Use user trace for debugging your applications; you can trace brokers, execution groups, and deployed message flows. Start user trace facilities using the **mqsichangetrace** command or the WebSphere Message Broker Explorer.

"Administration API (CMP) trace" on page 3554
Enable or disable service trace for the Administration API for WebSphere Message Broker (also known as the CMP API).

Related reference:

"User trace" on page 6873
User trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Administration log. User trace is inactive by default; you must activate it explicitly by using a command, or by selecting options in the WebSphere Message Broker Toolkit.

"Service trace" on page 6875
Service trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Event Log or to user trace. Service trace is inactive by default; you must activate it explicitly by using a command.

"WebSphere Message Broker Toolkit" on page 6783
The WebSphere Message Broker Toolkit provides your application development environment on Windows and Linux on x86.

ODBC trace

You can use various methods to trace for ODBC activity, depending on the operating system that you are using.

About this task

Windows For Windows, use the **Tracing** tab of the ODBC function:

1. Click **Start** > **Settings** > **Control Panel** > **Administrative Tools**.
2. Double-click **Data Sources**.
3. Click the **Tracing** tab.
4. Click **Start Tracing Now**.
5. Click **OK**.

To stop ODBC tracing, on the **Tracing** tab, click **Stop Tracing Now**, then **OK**.

Linux UNIX For Linux and UNIX operating systems using DataDirect drivers:

- To initiate trace for ODBC activity, edit the [ODBC] stanza in the file pointed to by your ODBCINI environment variable as follows:
  1. Change `Trace=0` to `Trace=1`.
  2. Specify a path and file name for `TraceFile`
  3. Ensure that the `TraceFile` entry points to a file system that has enough space to receive the trace output

Trace information is written to the location that is specified in the `odbc.ini` file.

Linux UNIX For Linux and UNIX operating systems using WebSphere Message Broker ODBC Database Extender (IE02) drivers:

- To initiate trace for ODBC activity, edit the [ODBC] stanza in the file pointed to by your ODBCSYSINI environment variable as follows:
  1. Change `Trace=no` to `Trace=yes`.
  2. Specify a path and file name for `TraceFile`
  3. Ensure that the `TraceFile` entry points to a file system that has enough space to receive the trace output

z/OS For z/OS, to initiate application trace for ODBC activity:

1. Edit the BIPDSNAO file in the component dataset and under the stanza entry [COMMON], change APPLTRACE=0 to APPLTRACE=1
2. Remove the comment from the COMPDIR variable declaration and the APPLTRC DD from the steps EGNOENV and EGENV, in the WebSphere Message Broker started task JCL.
3. Stop and restart the broker after you have made all the changes to the BIPDSNAO file and the started task JCL.

   By default, the trace output file is written to <component_HFS>/output/, into a file called db2appltrace.. Each address space has a unique number, and the eight character execution group label appended to the end of db2appltrace.

   This unique number, appended to the ODBC file, is the SE number in the execution group address space JOBLOG.

   If the eight character execution group label is not unique across multiple execution groups, look for the value of SE in the JOBLOG for which you want to view the ODBC trace, and find the file that specifies this value.

Results

Example

DB2 on WebSphere Message Broker for z/OS

The following sample ODBC trace files show the layout of a trace file, together with some examples of successful and error returns. The general layout of each group in an ODBC file is that:

- Each line is preceded by a time stamp.
- The first line displays what the call does.
- The second line displays the return.
- The third line displays the result.

The first trace file shows a trace where a call fails because an object does not have the correct authority to perform an action:

```
[2008-09-24 15:49:20.544123] SQLAllocStmt( hDbc=2, phStmt=&1c7f9554 )
[2008-09-24 15:49:20.544156] SQLAllocStmt( phStmt=1 )
[2008-09-24 15:49:20.544163]     ---> SQL_SUCCESS

[2008-09-24 15:49:20.544179] SQLFreeStmt( hStmt=1, fOption=SQL_CLOSE )
[2008-09-24 15:49:20.544189] SQLFreeStmt( )
[2008-09-24 15:49:20.544194]     ---> SQL_SUCCESS

[2008-09-24 15:49:20.544205] SQLPrepare( hStmt=1 )
[2008-09-24 15:49:20.544212] ( pszSqlStr="SELECT TESTTABLE.ID FROM
WMQI77.TESTTABLE TESTTABLE", cbSqlStr=-3 )
[2008-09-24 15:49:20.587083] SQLPrepare( )
[2008-09-24 15:49:20.587101]     ---> SQL_ERROR

[2008-09-24 15:49:20.587157] SQLError( hEnv=0, hDbc=0, hStmt=1,
pszSqlState=&3902af28, pfNativeError=&3902af24, pszErrorMsg=&1b88b0b0,
cbErrorMsgMax=1024, pcbErrorMsg=&3902aefc )
[2008-09-24 15:49:20.587190] SQLError( pszSqlState="42501", pfNativeError=-551,
pszErrorMsg="{DB2 FOR OS/390}{ODBC DRIVER}{DSN09015}
 DSNT408I SQLCODE = -551, ERROR:  WMQI83 DOES NOT HAVE THE PRIVILEGE TO PERFORM
          OPERATION SELECT ON OBJECT WMQI77.TESTTABLE
 DSNT418I SQLSTATE   = 42501 SQLSTATE RETURN CODE
 DSNT415I SQLERRP    = DSNXOSC SQL PROCEDURE DETECTING ERROR
 DSNT416I SQLERRD    = -100  0  0  -1  0  0 SQL DIAGNOSTIC INFORMATION
 DSNT416I SQLERRD    = X'FFFFFF9C'  X'00000000'  X'00000000'  X'FFFFFFFF'
          X'00000000'  X'00000000' SQL DIAGNOSTIC INFORMATION
```

```
  ERRLOC=1:13:2", pcbErrorMsg=623 )
[2008-09-24 15:49:20.587666]     ---> SQL_SUCCESS

[2008-09-24 15:49:20.587725] SQLError( hEnv=0, hDbc=0, hStmt=1,
pszSqlState=&3902af28, pfNativeError=&3902af24, pszErrorMsg=&1b88b0b0,
cbErrorMsgMax=1024, pcbErrorMsg=&3902aefc )
[2008-09-24 15:49:20.587752] SQLError( )
[2008-09-24 15:49:20.587757]     ---> SQL_NO_DATA_FOUND

[2008-09-24 15:49:20.588049] SQLFreeStmt( hStmt=1, fOption=SQL_DROP )
[2008-09-24 15:49:20.588075] SQLFreeStmt( )
[2008-09-24 15:49:20.588080]     ---> SQL_SUCCESS

[2008-09-24 15:49:20.593800] SQLTransact( hEnv=1, hDbc=0, fType=SQL_COMMIT )
[2008-09-24 15:49:20.593887] SQLTransact( )
[2008-09-24 15:49:20.593893]     ---> SQL_SUCCESS
```

The second trace file shows the same trace file with the operation working:

```
[2008-09-24 16:00:25.287052] SQLAllocStmt( hDbc=1, phStmt=&1c7f8e54 )
[2008-09-24 16:00:25.287068] SQLAllocStmt( phStmt=1 )
[2008-09-24 16:00:25.287075]     ---> SQL_SUCCESS

[2008-09-24 16:00:25.287088] SQLFreeStmt( hStmt=1, fOption=SQL_CLOSE )
[2008-09-24 16:00:25.287098] SQLFreeStmt( )
[2008-09-24 16:00:25.287104]     ---> SQL_SUCCESS

[2008-09-24 16:00:25.287114] SQLPrepare( hStmt=1 )
[2008-09-24 16:00:25.287121] ( pszSqlStr="SELECT TESTTABLE.ID FROM
WMQI77.TESTTABLE TESTTABLE", cbSqlStr=-3 )
[2008-09-24 16:00:25.302484] SQLPrepare( )
[2008-09-24 16:00:25.302510]     ---> SQL_SUCCESS

[2008-09-24 16:00:25.302539] SQLFreeStmt( hStmt=1,
fOption=SQL_CLOSE )
[2008-09-24 16:00:25.302555] SQLFreeStmt( )
[2008-09-24 16:00:25.302560]     ---> SQL_SUCCESS

[2008-09-24 16:00:25.302573] SQLExecute( hStmt=1 )
[2008-09-24 16:00:25.302622] SQLExecute( )
[2008-09-24 16:00:25.302628]     ---> SQL_SUCCESS

[2008-09-24 16:00:25.302660] SQLNumResultCols( hStmt=1,
pcCol=&3902c7fa )
[2008-09-24 16:00:25.302672] SQLNumResultCols( pcCol=1 )
[2008-09-24 16:00:25.302679]     ---> SQL_SUCCESS

[2008-09-24 16:00:25.302697] SQLDescribeCol( hStmt=1, iCol=1,
pszColName=&3902cb10, cbColNameMax=200, pcbColName=&3902c804,
pfSQLType=&3902c802, pcbColDef=&3902c858, pibScale=&3902c800,
pfNullable=&3902c7fe )
[2008-09-24 16:00:25.302733] SQLDescribeCol( pszColName="ID",
pcbColName=2, pfSQLType=SQL_CHAR, pcbColDef=10, pibScale=0,
pfNullable=SQL_NULLABLE )
[2008-09-24 16:00:25.302819]     ---> SQL_SUCCESS

[2008-09-24 16:00:25.302826] SQLColAttribute( hStmt=1, iCol=1,
fDescType=SQL_DESC_OCTET_LENGTH, rgbDesc=NULL, cbDescMax=0,
pcbDesc=NULL, pfDesc=&3902c864 )
[2008-09-24 16:00:25.302850] SQLColAttribute( pfDesc=10 )
[2008-09-24 16:00:25.302857]     ---> SQL_SUCCESS

[2008-09-24 16:00:25.302866] SQLBindCol( hStmt=1, iCol=1,
fCType=SQL_C_CHAR, rgbValue=&1b48829c, cbValueMax=12,
pcbValue=&1b488298 )
[2008-09-24 16:00:25.302888] SQLBindCol( )
[2008-09-24 16:00:25.302894]     ---> SQL_SUCCESS
```

```
[2008-09-24 16:00:25.302901] SQLSetStmtAttr( hStmt=1,
fAttribute=SQL_ATTR_ROW_BIND_TYPE, pvParam=&10, iStrLen=0 )
[2008-09-24 16:00:25.302917] SQLSetStmtAttr( )
[2008-09-24 16:00:25.302922]     ---> SQL_SUCCESS

[2008-09-24 16:00:25.302928] SQLSetStmtAttr( hStmt=1,
fAttribute=Unknown value 9, pvParam=&20, iStrLen=0 )
[2008-09-24 16:00:25.302943] SQLSetStmtAttr( )
[2008-09-24 16:00:25.302949]     ---> SQL_SUCCESS

[2008-09-24 16:00:25.302956] SQLExtendedFetch( hStmt=1,
fFetchType=SQL_FETCH_NEXT,iRow=0, pcRow=&1c7f6894,
rgfRowStatus=&1bca17d0 )
[2008-09-24 16:00:25.317947] ( Row=1, iCol=1, fCType=SQL_C_CHAR,
rgbValue="TABLG     ", pcbValue=10 )
[2008-09-24 16:00:25.317980] ( Row=2, iCol=1, fCType=SQL_C_CHAR,
rgbValue="TABLF     ", pcbValue=10 )
[2008-09-24 16:00:25.318001] ( Row=3, iCol=1, fCType=SQL_C_CHAR, r
gbValue="TABLE     ", pcbValue=10 )
[2008-09-24 16:00:25.318022] ( Row=4, iCol=1, fCType=SQL_C_CHAR,
rgbValue="TABLD     ", pcbValue=10 )
[2008-09-24 16:00:25.318044] ( Row=5, iCol=1, fCType=SQL_C_CHAR,
rgbValue="TABLC     ", pcbValue=10 )
[2008-09-24 16:00:25.318065] ( Row=6, iCol=1, fCType=SQL_C_CHAR,
rgbValue="TABLB     ", pcbValue=10 )
[2008-09-24 16:00:25.318087] ( Row=7, iCol=1, fCType=SQL_C_CHAR,
rgbValue="TABLA     ", pcbValue=10 )
[2008-09-24 16:00:25.318109] SQLExtendedFetch( pcRow=7 )
```

Related tasks:

Chapter 13, "Troubleshooting and support," on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

"Using trace" on page 3533
You can use different types of trace to help you with problem determination and troubleshooting.

Related reference:

"Supported databases" on page 3591
You can optionally configure databases to contain data that is accessed by your message flows. Databases from IBM and other suppliers are supported at specific versions on supported operating systems.

"Database facilities" on page 6891
The database products used by WebSphere Message Broker also record information that might be useful if you have any problems with their access.

Administration API (CMP) trace

Enable or disable service trace for the Administration API for WebSphere Message Broker (also known as the CMP API).

Enabling CMP API trace

About this task

To enable tracing for the CMP API for your application, set the environment variable MQSI_CMP_TRACE, where <*filename*> is the name of the file to which the trace is sent:

```
export MQSI_CMP_TRACE=<filename>
```

Or you can enable tracing by using the following API call in your code:

```
// Enable Message Broker Administration service trace
   BrokerProxy.enableAdministrationAPITracing("outputfile.txt");
```

This request logs all calls to the CMP API to the `outputfile.txt` file in the current directory. All CMP API activity in the entire Java virtual machine is logged.

You can also enable CMP API service trace from the **File** menu of the CMP API Exerciser.

In addition, because the CMP API uses the WebSphere MQ Java client, you can enable WebSphere MQ Java client tracing.

Disabling CMP API trace

About this task

To disable tracing for the CMP API for your application, use the following API call in your code:

```
// Disable Message Broker Administration service trace
   BrokerProxy.disableAdministrationAPITracing();
```

You can also disable CMP API service trace from the **File** menu of the CMP API Exerciser.

Related concepts:

"Trace" on page 6871
If you cannot get enough information about a particular problem from the entries that are available in the various logs, the next troubleshooting method to consider is using trace. Trace provides more details about what is happening while code runs. The information produced from trace is sent to a specified trace record, so that you or IBM support personnel can analyze it to discover the cause of your problem.

"The Administration API for WebSphere Message Broker" on page 54
The Administration API for WebSphere Message Broker is a programming interface that your applications can use to control brokers and their resources through a remote interface.

Related tasks:

"Using trace" on page 3533
You can use different types of trace to help you with problem determination and troubleshooting.

"Resolving problems when developing Administration API applications" on page 3510
Use the advice given here to help you to resolve problems that can arise when developing Administration API for WebSphere Message Broker (also known as the CMP API) applications.

Related reference:

"Service trace" on page 6875
Service trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Event Log or to user trace. Service trace is inactive by default; you must activate it explicitly by using a command.

Switching Trace nodes on and off

Use the **mqsichangetrace** command or the WebSphere Message Broker Explorer to switch Trace nodes on and off.

About this task

When an execution group is created, or a message flow is deployed, its Trace node switch is set to `on` by default. Execution groups and message flows that you have migrated from a previous version are also handled in this way. The message flow level Trace node switch setting is not changed on redeployment. You can significantly improve the performance of a flow that includes Trace nodes by switching Trace nodes off.

Restriction: If you switch off Trace nodes in a flow that was migrated from a version earlier than 6.1, you can no longer revert that flow to its earlier state.

If the Trace node setting for an execution group is off, all Trace nodes in its flows are disabled. You can change the settings for Trace nodes in individual message flows; the settings are applied when you turn on Trace nodes for the execution group. If the Trace node setting for an execution group is on, the Trace node switch setting of each message flow determines the effective settings. Use the "**mqsireporttrace** command" on page 3947 to check the settings for message flows and execution groups.

Example: switching off Trace nodes for an execution group on distributed systems, using the command line

Procedure

To disable the Trace node switch settings of all message flows in an execution group of a broker called MB7BROKER, (that is, stop all the Trace nodes in any of the message flows deployed to the execution group from executing), enter the following command:

```
mqsichangetrace MB7BROKER –n off -e default
```

In this example, `-n off` switches off Trace nodes in the default execution group (`-e default`).
All Trace nodes are switched off, even if the message flow that contains them has its Trace node switch set to *on*.

Example: switching off Trace nodes for an execution group on z/OS systems, using the command line

Procedure

To disable the Trace node switch settings of all message flows in an execution group of a broker called MQP1BRK, (that is, stop all the Trace nodes in any of the message flows deployed to the execution group from executing), enter the following command:

```
F MQP1BRK,ct n='off', e='default'
```

In this example, `n='off'` switches off Trace nodes in the default execution group (`e='default'`).
All Trace nodes are switched off, even if the message flow that contains them has its Trace node switch set to *on*.

Example: switching off Trace nodes for a message flow on distributed systems, using the command line

About this task

Trace nodes for the default execution group in broker MB7BROKER are switched on. You want to turn off Trace nodes for the myFlow message flow.

Procedure

Enter the following command:

```
mqsichangetrace MB7BROKER -n off -e default -f myFlow
```

Example: switching on Trace nodes for a message flow and an execution group on distributed systems, using the command line

About this task

Trace nodes for the default execution group in broker MB7BROKER are switched off. You want to turn on Trace nodes for the myFlow message flow, then turn on Trace nodes for the execution group.

Procedure

1. To turn on Trace nodes for the message flow, enter the following command:

   ```
   mqsichangetrace MB7BROKER -n on -e default -f myFlow
   ```

   Trace nodes for the myFlow message flow are turned on, but this setting is not applied until you turn on trace nodes for the execution group.
2. To turn on Trace nodes for the execution group, enter the following command:

   ```
   mqsichangetrace MB7BROKER –n on -e default
   ```

   Trace nodes are turned on for the default execution group. Within that execution group, Trace nodes are enabled in all message flows that have Trace nodes turned on, including the myFlow message flow.

Example: switching off Trace nodes for a message flow on z/OS systems, using the command line

About this task

Trace nodes for the default execution group in broker MQP1BRK are switched on. You want to turn off Trace nodes for one of the message flows (myFlow).

Procedure

Enter the following command:

```
F MQP1BRK,ct n='off', e='default', f='myFlow'
```

Example: switching on Trace nodes for an execution group, using the WebSphere Message Broker Explorer

About this task

Trace nodes are enabled by default. If you have switched them off, follow these steps to enable Trace nodes for an execution group from the WebSphere Message Broker Explorer:

Procedure

1. In the Navigator view, expand the Brokers folder and right-click the execution group with which you want to work.
2. Click **Trace Nodes All Flows** > **Enable**. An alert saying Trace nodes are switched on is displayed in the Alert Viewer. The Trace node switch setting of each message flow determines the effective settings.

Example: switching on Trace nodes for a message flow, using the WebSphere Message Broker Explorer

Before you begin

If the Trace node setting for an execution group is off, all Trace nodes in its flows are disabled. You can change the settings for Trace nodes in individual message flows; the settings are applied when you turn on Trace nodes for the execution group. If the Trace node setting for an execution group is on, the Trace node switch setting of each message flow determines the effective settings.

About this task

Follow these steps to enable Trace nodes for one of your message flows from the WebSphere Message Broker Explorer:

Procedure

1. In the Navigator view, expand the Brokers folder and right-click the message flow with which you want to work.
2. Click **Trace Nodes** > **Enable** An alert is displayed in the Alert Viewer.

Related concepts:

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

Related reference:

"**mqsichangetrace** command" on page 3822
Use the **mqsichangetrace** command to set the tracing characteristics for a broker.

"**mqsireporttrace** command" on page 3947
Use the **mqsireporttrace** command to display the trace options currently in effect. Trace can be run only against a broker, or any of its resources.

"Trace node" on page 4942
Use the Trace node to generate trace records that you can use to monitor the behavior of a message flow.

Using dumps and abend files

A dump or an abend file might be produced when a problem occurs. Dumps and abend files can be used by IBM to resolve the problem.

About this task

This section contains the following topics:

Procedure

- "Checking for dumps" on page 3559
- "Using the DUMP command on z/OS" on page 3560
- "Checking for abend files" on page 3562

What to do next

You can also use the "IBM Support Assistant Data Collector" on page 3565 to help with data collection.

Related tasks:

Chapter 13, “Troubleshooting and support,” on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

“Contacting your IBM Support Center” on page 3563
If you cannot resolve problems that you find when you use WebSphere Message Broker, or if you are directed to do so by an error message generated by WebSphere Message Broker, you can request assistance from your IBM Support Center.

“IBM Support Assistant Data Collector” on page 3565
You can collect diagnostic documents by using IBM Support Assistant Data Collector, and submit a problem report to IBM. IBM Support Assistant Data Collector is included with your WebSphere Message Broker installation.

Related reference:

“Abend files” on page 6880
When a process does not end normally an abend file is generated.

“Dumps” on page 6877
Under exceptional circumstances, Windows MiniDumps, UNIX core dumps, or z/OS SVC or core dumps might be produced.

Checking for dumps

If a dump occurs on your system, an error message is produced.

Procedure

- **On Windows**

  BIP2111 error message (message broker internal error). The error message contains the path to the MiniDump file in your errors directory.

- **On UNIX**

  BIP2060 error message (execution group terminated unexpectedly). Look in the directory where the broker was started, or in the service user ID's home directory, to find the core dump file.

- **On z/OS**
  - BIP2060 error message (execution group ended unexpectedly) from the main Broker Address Space. This message should be accompanied by one of the following messages and dump.
  - IEF450I message in the syslog, or component's joblog, showing an abend code followed by a reason code, for example:

    ```
    IEF450I MQ83BRK DEFAULT - ABEND=S2C1 U0000 REASON=000000C4
    ```

    Look in the system's dump dataset `hlq` for the dump dataset, or search the syslog for the appropriate IEA611I message to find out the dump dataset name.
  - IEA993I message in the syslog for a SYSMDUMP. Look in the started task user's directory for the `coredump.pid` file, as specified in the syslog:

    ```
    IEA993I SYSMDUMP TAKEN TO coredump.00500319
    ```
  - An error message for an SVC dump; see “Dumps on WebSphere Message Broker for z/OS” on page 6878 for further information on SVC dumps.

Related tasks:

Chapter 13, “Troubleshooting and support,” on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

"Using dumps and abend files" on page 3558
A dump or an abend file might be produced when a problem occurs. Dumps and abend files can be used by IBM to resolve the problem.

"Checking for abend files" on page 3562
Abend files are produced when a process ends abnormally. The information contained in an abend file helps the IBM Support Center to diagnose and fix the problem.

"Using the DUMP command on z/OS"
Follow the steps in this task to use the DUMP command on z/OS.

Related reference:

"Dumps" on page 6877
Under exceptional circumstances, Windows MiniDumps, UNIX core dumps, or z/OS SVC or core dumps might be produced.

"Abend files" on page 6880
When a process does not end normally an abend file is generated.

Using the DUMP command on z/OS

Follow the steps in this task to use the DUMP command on z/OS.

About this task

z/OS You might be asked to dump any or several of the following address spaces for IBM to resolve the problem:

- Control address space
- DataFlowEngine address space
- OMVS kernel address space

The following procedure demonstrates how to dump the DataFlowEngine address space. This procedure is the same for any of the address spaces.

Procedure

1. Find the address space ID of the address space that you want to dump using the display command on the z/OS syslog:

   ```
   D OMVS,U=your started task user ID
   ```

   This command displays the address spaces of all the processes that are running from your started task user ID, for example:

   ```
   D OMVS,U=MQ01BRK
   BPXO040I 16.14.30 DISPLAY OMVS 237
   OMVS     000D ACTIVE          OMVS=(14)
   USER     JOBNAME  ASID        PID       PPID STATE   START     CT_SECS

   MQ01BRK  MQ01BRK  009D   67306064   84083282 HRI--- 15.41.55     48.37
     LATCHWAITPID=         0 CMD=bipservice MQ01BRK AUTO
   MQ01BRK  MQ01BRK  009D   84083282          1 1WI--- 15.41.55     48.37
     LATCHWAITPID=         0 CMD=/argoinst/S000_L30307_P/usr/lpp/mqsi/bin
   MQ01BRK  MQ01BRK  009D   16974444   67306064 HRI--- 15.42.01     48.37
     LATCHWAITPID=         0 CMD=bipbroker MQ01BRK
   MQ01BRK  MQ01BRK  009F   16974445          1 1W---- 15.42.05   2914.22
     LATCHWAITPID=         0 CMD=/argoinst/S000_L30307_P/usr/lpp/mqsi/bin
   MQ01BRK  MQ01BRK  009F   33751662   16974445 HR---- 15.42.05   2914.22
     LATCHWAITPID=         0 CMD=DataFlowEngine MQ01BRK ca614eec-f300-000
   ```

   The infrastructure main program bipimain is the first process in every address space. For a control address space, bipimain starts the bipservice process, which

starts the bipbroker process, which might also start the biphttplistener process, depending on the configuration. For a DataFlowEngine address space, bipimain starts the DataFlowEngine process. For each execution group, an additional DataFlowEngine address space is started. In this example, only one execution group is deployed.

2. Use the z/OS DUMP command to dump the DataFlowEngine address space, which is shown in the above example as 9F.
   a. Enter the following command:

      ```
      DUMP TITLE=(DFE)
      ```

      The console returns:

      ```
      *`15 IEE094D SPECIFY OPERAND(S) FOR DUMP COMMAND
      ```

   b. Enter:

      ```
      R 15,ASID=9F,CONT
      ```

      The console returns:

      ```
      *16 IEE094D SPECIFY OPERAND(S) FOR DUMP COMMAND
      ```

   c. Enter:

      ```
      R 16,SDATA=(CSA,RGN,PSA,SQA,LSQA,LPA,TRT,GRSQ,SUM),END
      ```

      The console returns:

      ```
      IEE600I REPLY TO 16 IS;SDATA=(CSA,RGN,PSA,SQA,LSQA,LPA,TRT,GRSQ,SUM),END
      IEA794I SVC DUMP HAS CAPTURED: 356
      DUMPID=014 REQUESTED BY JOB (*MASTER*)
      DUMP TITLE=DFE
      IEF196I IGD101I SMS ALLOCATED TO DDNAME (SYS00018)
      IEF196I         DSN (SYS3.DUMP.ARG5.#MASTER#.T142958.S00014      )
      IEF196I         STORCLAS (SMS) MGMTCLAS (DUMP) DATACLAS (        )
      IEF196I         VOL SER NOS= ARGSMR
      ```

      The dump is stored in either a pre-allocated dump data set called SYS1.DUMPxx, or an automatically allocated dump data set named according to an installation-specified pattern.

3. In some scenarios, all address spaces for a given broker, that is, all those listed in the example above, plus the OMVS address space and dataspaces, are needed by IBM to resolve the problem. Use the z/OS DUMP command to dump all MQ01BRK address spaces.
   a. Enter the command:

      ```
      DUMP TITLE=(ALL)
      ```

      The console returns:

      ```
      `15 IEE094D SPECIFY OPERAND(S) FOR DUMP COMMAND
      ```

   b. Enter:

      ```
      R 15,JOBNAME=(OMVS,MQ01BRK),DSPNAME=('OMVS'.*),SDATA=(PSA,SQA,LSQA,RGN,TRT,
        LPA,CSA,GRSQ,SUM,NUC)
      ```

      The console returns:

      ```
      IEE600I REPLY TO 15 IS;JOBNAME=(OMVS,MQ01BRK),DSPNAME=('OMVS'.*),S
      IEA794I SVC DUMP HAS CAPTURED: 303
      DUMPID=040 REQUESTED BY JOB (*MASTER*)
      DUMP TITLE=ALL
      IEE853I 13.40.40 SYS1.DUMP TITLES 306
      SYS1.DUMP DATA SETS AVAILABLE=000 AND FULL=000
      CAPTURED DUMPS=0001, SPACE USED=00000447M, SPACE FREE=00001053M
          DUMP.MVK4.#MASTER#.D030415.T134007.S00039 DATA UNAVAILABLE WHILE
      ```

```
        BEING DUMPED TO
IEA611I COMPLETE DUMP ON DUMP.MVK4.#MASTER#.D030415.T134007.S00039 309
DUMPID=040 REQUESTED BY JOB (*MASTER*)
FOR ASIDS(000D,009D,009F)
```

Results

You can also find information on the individual thread by issuing the **DISPLAY** z/OS console command, as in the example:

```
D OMVS,PID=83886535
```

Related tasks:

Chapter 13, "Troubleshooting and support," on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

"Using dumps and abend files" on page 3558
A dump or an abend file might be produced when a problem occurs. Dumps and abend files can be used by IBM to resolve the problem.

"Checking for dumps" on page 3559
If a dump occurs on your system, an error message is produced.

"Contacting your IBM Support Center" on page 3563
If you cannot resolve problems that you find when you use WebSphere Message Broker, or if you are directed to do so by an error message generated by WebSphere Message Broker, you can request assistance from your IBM Support Center.

Related reference:

"Dumps" on page 6877
Under exceptional circumstances, Windows MiniDumps, UNIX core dumps, or z/OS SVC or core dumps might be produced.

"Abend files" on page 6880
When a process does not end normally an abend file is generated.

Checking for abend files

Abend files are produced when a process ends abnormally. The information contained in an abend file helps the IBM Support Center to diagnose and fix the problem.

About this task

The following list contains examples of what might cause the broker to produce an abend file:

Procedure

- The broker runs out of memory.
- A user-defined extension causes an instruction in the broker process that is not valid.
- An unrecoverable error occurs in the broker.

Results

Abend files are never produced during normal operation. If an abend file is produced, contact the IBM Support Center for assistance.

Related tasks:

"Contacting your IBM Support Center"
If you cannot resolve problems that you find when you use WebSphere Message Broker, or if you are directed to do so by an error message generated by WebSphere Message Broker, you can request assistance from your IBM Support Center.

Chapter 13, "Troubleshooting and support," on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

"Checking for dumps" on page 3559
If a dump occurs on your system, an error message is produced.

"Using dumps and abend files" on page 3558
A dump or an abend file might be produced when a problem occurs. Dumps and abend files can be used by IBM to resolve the problem.

"Using the DUMP command on z/OS" on page 3560
Follow the steps in this task to use the DUMP command on z/OS.

Related reference:

"Dumps" on page 6877
Under exceptional circumstances, Windows MiniDumps, UNIX core dumps, or z/OS SVC or core dumps might be produced.

"Abend files" on page 6880
When a process does not end normally an abend file is generated.

Contacting your IBM Support Center

If you cannot resolve problems that you find when you use WebSphere Message Broker, or if you are directed to do so by an error message generated by WebSphere Message Broker, you can request assistance from your IBM Support Center.

About this task

Before you contact your Support Center, use the checklist shown here to gather important information. Some items might not be relevant in every situation, but provide as much information as you can to enable the IBM Support Center to re-create your problem. You can also use the "IBM Support Assistant Data Collector" on page 3565 to help with data collection.

For WebSphere Message Broker:

- The product version.
- Any fix packs applied.
- Any interim fixes applied.
- All current trace and error logs, including relevant Windows Event log or LinuxUNIX operating system syslog entries, and any abend or dump files from the *install_dir*\errors directory on Windows, or the /var/mqsi/errors directory on LinuxUNIX. Obtain user trace log files at debug level for all relevant message flows and preferably format them. Also include any requested service trace files.

  To send files from distributed systems, create a compressed file using any compression utility.

  To send a file from the file system to IBM, use **tar** to compress the file. For example `tar -cx -f coredump.0002009E coredump.toibm`. To send

MVS data sets to IBM, terse them using `TRSMAIN`, which you can download from z/OS tools download.

- A list of the components installed. Include details of the number of computers and their operating systems, the number of brokers and the computers on which they are running, and the existence and details of each User Name Server.
- The compressed file obtained by exporting your workspace and appropriate message flows and message sets. This action is performed from the WebSphere Message Broker Toolkit.
- Details of the operation that you were performing, the results that occurred, and the results that you were expecting.
- A sample of the messages that were being used when the problem arose
- If relevant, the report file from the C or COBOL importer. This file is located in the directory from which the file import was attempted.
- If you are using tagged delimited wire format on message sets, the TDS log files.

For WebSphere MQ:

- The product version.
- Any fix packs applied.
- Any interim fixes applied.
- All current trace and error logs, including relevant Windows Event log or Linux and UNIX operating system syslog entries and First Failure Support Technology™ (FFST™) output files. You can find these files, which have the extension `.fdc`, in the errors subdirectory in the WebSphere MQ home directory.
- Details of WebSphere MQ client software, if appropriate.

For each database that you are using:

- The product and release level (for example, DB2 9.1).
- Any fix packs applied.
- Any interim fixes applied.
- All current trace and error logs, including relevant Windows Event log or Linux and UNIX operating system syslog entries, for example the `db2dialog.log` file on DB2. Check the database product documentation for details of where to find these files.
- Definitions of any database tables.
- Any ODBC traces.

Windows **For Windows:**

- The version.
- The Service Pack level.
- The environment settings.

UNIX **For Linux and UNIX operating systems:**

- The product version. You can find the version installed by using the **`uname -a`** command.
- Any service level and patches that have been applied.
- The environment settings.

z/OS **For z/OS:**

- The product version

- The list of PTFs that have been applied
- The environment settings
- The job logs from all address spaces

Related tasks:

Chapter 13, “Troubleshooting and support,” on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

“Windows: Viewing the local error log” on page 3527
The Windows Event Viewer is where WebSphere Message Broker writes records to the local system. Use Windows system facilities to view this log.

“Linux and UNIX systems: Configuring the syslog daemon” on page 3529
On Linux and UNIX systems, all WebSphere Message Broker messages (other than messages that are generated by the command-line utilities) are sent to the syslog subsystem.

“IBM Support Assistant Data Collector”
You can collect diagnostic documents by using IBM Support Assistant Data Collector, and submit a problem report to IBM. IBM Support Assistant Data Collector is included with your WebSphere Message Broker installation.

Related reference:

“User trace” on page 6873
User trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Administration log. User trace is inactive by default; you must activate it explicitly by using a command, or by selecting options in the WebSphere Message Broker Toolkit.

“Service trace” on page 6875
Service trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Event Log or to user trace. Service trace is inactive by default; you must activate it explicitly by using a command.

“**mqsichangetrace** command” on page 3822
Use the **mqsichangetrace** command to set the tracing characteristics for a broker.

“**mqsiformatlog** command” on page 3880
Use the **mqsiformatlog** command to process the XML log created by **mqsireadlog**. The command retrieves and formats any messages that the XML log contains into a form suitable for the locale of the user who runs the command.

“**mqsireadlog** command” on page 3905
Use the **mqsireadlog** command to retrieve trace records for the specified component.

“**mqsireporttrace** command” on page 3947
Use the **mqsireporttrace** command to display the trace options currently in effect. Trace can be run only against a broker, or any of its resources.

IBM Support Assistant Data Collector

You can collect diagnostic documents by using IBM Support Assistant Data Collector, and submit a problem report to IBM. IBM Support Assistant Data Collector is included with your WebSphere Message Broker installation.

About this task

You might have other versions of IBM Support Assistant, but if you run the commands provided in the following topics, you will collect the appropriate information for WebSphere Message Broker.

Procedure

- "Collecting data in console mode with IBM Support Assistant Data Collector"
- "Selecting a problem collector for IBM Support Assistant Data Collector" on page 3568

What to do next

Note: The IBM Support Assistant Data Collector is not supported for z/OS.

Related concepts:

"Selecting a problem collector for IBM Support Assistant Data Collector" on page 3568
You can use the problem collectors installed with IBM Support Assistant Data Collector to gather diagnostic information.

Related tasks:

"Collecting data in console mode with IBM Support Assistant Data Collector"
You can use the IBM Support Assistant Data Collector in console mode to collect diagnostic documents for submission to IBM.

"Contacting your IBM Support Center" on page 3563
If you cannot resolve problems that you find when you use WebSphere Message Broker, or if you are directed to do so by an error message generated by WebSphere Message Broker, you can request assistance from your IBM Support Center.

"Making initial checks" on page 3347
Before you start problem determination in detail, consider whether there is an obvious cause of the problem, or an area of investigation that is likely to give useful results. This approach to diagnosis can often save a lot of work by highlighting a simple error, or by narrowing down the range of possibilities.

Chapter 13, "Troubleshooting and support," on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

Collecting data in console mode with IBM Support Assistant Data Collector

You can use the IBM Support Assistant Data Collector in console mode to collect diagnostic documents for submission to IBM.

Before you begin

Before you start:

Before contacting IBM Software Support, ensure that your company has an active IBM software subscription and support contract, and that you are authorized to submit problems to IBM. See "Contacting IBM Software Support" on page 3571 for more details.

About this task

Procedure

To run the IBM Support Assistant Data Collector and collect diagnostic documents, complete the following steps:

1. Ensure that the WebSphere Message Broker run time variables are set correctly for your environment by using the **mqsiprofile** command. See "Environment variables after installation" on page 3642 for more information.
2. At a command prompt, enter the **mqsidc** command.

   Note: You can ensure that the script is executable by entering the following command to change the file permissions: **chmod 755 mqsidc**

   The IBM Support Assistant Data Collector starts in console mode.
3. Start the data collection. Available options are presented as numbered lists.
   a. Type a file name for saving the collected data, or press Enter to generate a unique file name.
   b. At the input field prompts, type the number of the required option and press Enter.
4. Choose a data transfer method.
   a. Type the number of the required option for transferring the diagnostic documents to IBM and press Enter. The available options are:
      1) Send the documents to IBM Software Support using secure transfer (HTTPS). You require a problem management record (PMR) number obtained through IBM Software Support.
      2) FTP the documents to IBM Software Support (unencrypted). You require a PMR number obtained through IBM Software Support. This option is less secure than option i.
      3) FTP the documents to another location (unencrypted). You are required to provide a target FTP location and appropriate authentication to access the documents.
      4) End the collection without sending.

      If you choose any of the first three options above, you are required to enter additional information to complete the upload.
   b. Type the number of the required option to confirm the data collection.

   When the IBM Support Assistant Data Collector completes, a .zip file is created at the location specified during the collection. You can extract the compressed files and examine the collected data by using a suitable tool.

What to do next

Note: The IBM Support Assistant Data Collector is not supported for z/OS.

Related concepts:

"Selecting a problem collector for IBM Support Assistant Data Collector" on page 3568
You can use the problem collectors installed with IBM Support Assistant Data Collector to gather diagnostic information.

Related tasks:

"IBM Support Assistant Data Collector" on page 3565
You can collect diagnostic documents by using IBM Support Assistant Data Collector, and submit a problem report to IBM. IBM Support Assistant Data

Collector is included with your WebSphere Message Broker installation.

"Contacting your IBM Support Center" on page 3563
If you cannot resolve problems that you find when you use WebSphere Message Broker, or if you are directed to do so by an error message generated by WebSphere Message Broker, you can request assistance from your IBM Support Center.

"Contacting IBM Software Support" on page 3571
IBM Software Support provides assistance with product defects.

"Making initial checks" on page 3347
Before you start problem determination in detail, consider whether there is an obvious cause of the problem, or an area of investigation that is likely to give useful results. This approach to diagnosis can often save a lot of work by highlighting a simple error, or by narrowing down the range of possibilities.

Chapter 13, "Troubleshooting and support," on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

Selecting a problem collector for IBM Support Assistant Data Collector

You can use the problem collectors installed with IBM Support Assistant Data Collector to gather diagnostic information.

Note: Before you can use the problem collectors, WebSphere Message Broker must have installed successfully, and an **mqsiprofile** must exist which you can start successfully (see "Environment variables after installation" on page 3642 for more information).

Generic problem collector
The generic problem collector gathers configuration information about WebSphere Message Broker, operating system details, WebSphere MQ levels, environment details, the application event log, and the syslog.

Broker problem collector
The broker problem collector gathers information about the broker's deployed configuration, the standard system logs (STDOUT and STDERR) for its components (admin agent, ExecutionGroup, and httplistener), and abend files. This problem collector also gathers the diagnostic data collected by the generic problem collector.

Related tasks:

"IBM Support Assistant Data Collector" on page 3565
You can collect diagnostic documents by using IBM Support Assistant Data Collector, and submit a problem report to IBM. IBM Support Assistant Data Collector is included with your WebSphere Message Broker installation.

"Collecting data in console mode with IBM Support Assistant Data Collector" on page 3566
You can use the IBM Support Assistant Data Collector in console mode to collect diagnostic documents for submission to IBM.

"Contacting your IBM Support Center" on page 3563
If you cannot resolve problems that you find when you use WebSphere Message Broker, or if you are directed to do so by an error message generated by WebSphere Message Broker, you can request assistance from your IBM Support Center.

“Making initial checks” on page 3347
Before you start problem determination in detail, consider whether there is an obvious cause of the problem, or an area of investigation that is likely to give useful results. This approach to diagnosis can often save a lot of work by highlighting a simple error, or by narrowing down the range of possibilities.

Chapter 13, “Troubleshooting and support,” on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

Searching knowledge bases

If you have a problem with your IBM software, you want it resolved quickly. Begin by searching the available knowledge bases to determine whether the resolution to your problem is already documented.

Procedure

1. **Search the information center**

   IBM provides extensive documentation in the form of online information centers. An information center can be installed on your local machine or on a local intranet. An information center can also be viewed on the IBM Web site. You can use the powerful search function of the information center to query conceptual and reference information as well as detailed instructions for completing tasks.

2. **Search the Internet**

   If you cannot find an answer to your question in the information center, search the Internet for the latest, most complete information that might help you resolve your problem, including:

   - IBM technotes
   - IBM downloads
   - IBM Redbooks publications
   - IBM developerWorks
   - Forums and newsgroups
   - Internet search engines

Related concepts:

“Troubleshooting overview” on page 3345
Troubleshooting is the process of finding and eliminating the cause of a problem. Whenever you have a problem with your IBM software, the troubleshooting process begins as soon as you ask yourself "what happened?"

Related tasks:

Chapter 13, “Troubleshooting and support,” on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

“Making initial checks” on page 3347
Before you start problem determination in detail, consider whether there is an obvious cause of the problem, or an area of investigation that is likely to give useful results. This approach to diagnosis can often save a lot of work by highlighting a simple error, or by narrowing down the range of possibilities.

“Dealing with problems” on page 3363
Learn how to resolve some of the typical problems that can occur.

"Using logs" on page 3526
There are a variety of logs that you can use to help with problem determination and troubleshooting.

"Using trace" on page 3533
You can use different types of trace to help you with problem determination and troubleshooting.

"Using dumps and abend files" on page 3558
A dump or an abend file might be produced when a problem occurs. Dumps and abend files can be used by IBM to resolve the problem.

"Getting product fixes"
A product fix might be available to resolve your problem. You can determine what fixes are available from the IBM support site.

"Contacting IBM Software Support" on page 3571
IBM Software Support provides assistance with product defects.

"Recovering after failure" on page 3574
Follow a set of procedures to recover after a serious problem.

Related reference:

"Troubleshooting" on page 6864
Use the reference information in this section to help you diagnose errors in WebSphere Message Broker.

Getting product fixes

A product fix might be available to resolve your problem. You can determine what fixes are available from the IBM support site.

About this task

To determine what fixes are available from the IBM support site:

1. Open the WebSphere Message Broker support web page.
2. Click **Downloads**, then **Recommended fixes**. This web page provides links to the latest available maintenance for the in service WebSphere Message Broker family of products.

To receive weekly email notifications about fixes and other news about IBM products, follow these steps.

Procedure

1. From the support site (WebSphere Message Broker support web page), locate the **Notifications** box in the center of the page.
2. If you are not signed in, click **Sign in to create, manage or view your subscriptions**. If you have not registered, click **register now** on the sign-in page and follow the on-screen instructions.
3. Click **Manage my subscriptions**.
4. Click the **Subscribe** tab. A list of products families is shown.
5. In the Software column, click **WebSphere**. A list of products is shown.
6. Select the product for which you want to receive notifications (for example, **WebSphere Message Broker**), then click **Continue**.
7. Set options to determine what notifications you receive, how often you receive them, and to which folder they are saved, then click **Submit**.

Related concepts:

"Troubleshooting overview" on page 3345
Troubleshooting is the process of finding and eliminating the cause of a problem. Whenever you have a problem with your IBM software, the troubleshooting process begins as soon as you ask yourself "what happened?"

Related tasks:

Chapter 13, "Troubleshooting and support," on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

"Making initial checks" on page 3347
Before you start problem determination in detail, consider whether there is an obvious cause of the problem, or an area of investigation that is likely to give useful results. This approach to diagnosis can often save a lot of work by highlighting a simple error, or by narrowing down the range of possibilities.

"Dealing with problems" on page 3363
Learn how to resolve some of the typical problems that can occur.

"Using logs" on page 3526
There are a variety of logs that you can use to help with problem determination and troubleshooting.

"Using trace" on page 3533
You can use different types of trace to help you with problem determination and troubleshooting.

"Using dumps and abend files" on page 3558
A dump or an abend file might be produced when a problem occurs. Dumps and abend files can be used by IBM to resolve the problem.

"Getting product fixes" on page 3570
A product fix might be available to resolve your problem. You can determine what fixes are available from the IBM support site.

"Contacting IBM Software Support"
IBM Software Support provides assistance with product defects.

"Recovering after failure" on page 3574
Follow a set of procedures to recover after a serious problem.

Related reference:

"Troubleshooting" on page 6864
Use the reference information in this section to help you diagnose errors in WebSphere Message Broker.

Contacting IBM Software Support

IBM Software Support provides assistance with product defects.

About this task

Before you contact IBM Software Support, you must ensure that your company has an active IBM software subscription and support contract, and that you are authorized to submit problems to IBM. The type of software subscription and support contract that you need depends on the type of product that you have:

- For IBM distributed software products (including, but not limited to, Tivoli, Lotus®, and Rational products, as well as DB2 and WebSphere products that run on Windows or UNIX operating systems), enroll in Passport Advantage in one of the following ways:
  - Online: Go to the Passport Advantage web page and click **How to Enroll**.

  - By telephone: For the telephone number to call in your country, go to the Software Support Handbook, click **Contacts**, then **Worldwide contacts**.
- For customers with Subscription and Support (S & S) contracts, go to the Open service request web page.
- For customers with IBMLink, CATIA, Linux, S/390®, iSeries, pSeries, zSeries, and other support agreements, go to the IBM Support Line web page.
- For IBM eServer™ software products (including, but not limited to, DB2 and WebSphere products that run in zSeries, pSeries, and iSeries environments), you can purchase a software subscription and support agreement by working directly with an IBM marketing representative or an IBM Business Partner. For more information about support for eServer software products, go to the Support for IBM Systems web page.

If you are not sure what type of software subscription and support contract you need, call 1-800-IBMSERV (1-800-426-7378) in the United States or, from other countries, go to the contacts page of the Software Support Handbook and click the name of your geographic region for telephone numbers of people who provide support for your location.

Follow the steps in this topic to contact IBM Software Support:

Procedure

1. "Determine the effect of the problem on your business"
2. "Describe your problem and gather background information" on page 3573
3. "Submit your problem to IBM Software Support" on page 3573

Determine the effect of the problem on your business

About this task

When you report a problem to IBM, you will be asked to supply a severity level. Therefore, you need to understand and assess the effect on your business of the problem that you are reporting. Use the following criteria:

Severity 1
: **Critical** effect on business: You are unable to use the program, resulting in a critical effect on operations. This condition requires an immediate solution.

Severity 2
: **Significant** effect on business: The program is usable but is severely limited.

Severity 3
: **Some** effect on business: The program is usable with less significant features (not critical to operations) unavailable.

Severity 4
: **Minimal** effect on business: The problem has little effect on operations, or a reasonable workaround to the problem has been implemented.

Describe your problem and gather background information

About this task

When you are explaining a problem to IBM, be as specific as possible. Include all relevant background information so that IBM Software Support specialists can help you to solve the problem efficiently. To save time, know the answers to these questions:

- What software versions were you running when the problem occurred?
- Do you have logs, traces, and messages that are related to the problem symptoms? IBM Software Support is likely to ask for this information.
- Can the problem be re-created? If so, what steps led to the failure?
- Have any changes been made to the system? (For example, hardware, operating system, networking software, and so on.)
- Are you currently using a workaround for this problem? If so, be prepared to explain it when you report the problem.

Submit your problem to IBM Software Support

About this task

You can submit your problem in one of two ways:

- Online: Go to the Software Support Handbook and enter your information into the appropriate problem submission tool.
- By telephone: For the telephone number to call in your country, go to the contacts page of the Software Support Handbook and click the name of your geographic region for telephone numbers of people who provide support for your location.

The IBM Support Assistant Data Collector can be used to submit diagnostic documents if you have an existing problem management record (PMR). For more information, see “IBM Support Assistant Data Collector” on page 3565.

If the problem that you submit is for a software defect or for missing or inaccurate documentation, IBM Software Support might create an Authorized Program Analysis Report (APAR). The APAR describes the problem in detail. Whenever possible, IBM Software Support provides a workaround for you to implement until the APAR is resolved and a fix is delivered.

IBM publishes resolved APARs on the IBM product support Web pages daily, so that other users who experience the same problem can benefit from the same resolutions.

Related concepts:

“Troubleshooting overview” on page 3345
Troubleshooting is the process of finding and eliminating the cause of a problem. Whenever you have a problem with your IBM software, the troubleshooting process begins as soon as you ask yourself "what happened?"

Related tasks:

Chapter 13, “Troubleshooting and support,” on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

"Making initial checks" on page 3347
Before you start problem determination in detail, consider whether there is an obvious cause of the problem, or an area of investigation that is likely to give useful results. This approach to diagnosis can often save a lot of work by highlighting a simple error, or by narrowing down the range of possibilities.

"Dealing with problems" on page 3363
Learn how to resolve some of the typical problems that can occur.

"Using logs" on page 3526
There are a variety of logs that you can use to help with problem determination and troubleshooting.

"Using trace" on page 3533
You can use different types of trace to help you with problem determination and troubleshooting.

"Using dumps and abend files" on page 3558
A dump or an abend file might be produced when a problem occurs. Dumps and abend files can be used by IBM to resolve the problem.

"Getting product fixes" on page 3570
A product fix might be available to resolve your problem. You can determine what fixes are available from the IBM support site.

"Contacting IBM Software Support" on page 3571
IBM Software Support provides assistance with product defects.

"Recovering after failure"
Follow a set of procedures to recover after a serious problem.

Related reference:

"Troubleshooting" on page 6864
Use the reference information in this section to help you diagnose errors in WebSphere Message Broker.

Recovering after failure

Follow a set of procedures to recover after a serious problem.

About this task

Use the recovery methods described here only if you cannot resolve the underlying problem by using the diagnostic techniques described throughout the Chapter 13, "Troubleshooting and support," on page 3345 section of the information center. If your problem cannot be resolved by using these techniques, contact your IBM Support Center.

This section contains the following topics:

Procedure

- "Recovering after the broker fails" on page 3575
- "Recovering after an execution group fails" on page 3576
- "Recovering after the broker's queue manager fails" on page 3576

Related tasks:

"Making initial checks" on page 3347
Before you start problem determination in detail, consider whether there is an obvious cause of the problem, or an area of investigation that is likely to give useful results. This approach to diagnosis can often save a lot of work by highlighting a simple error, or by narrowing down the range of possibilities.

"Using logs" on page 3526
There are a variety of logs that you can use to help with problem determination and troubleshooting.

"Using trace" on page 3533
You can use different types of trace to help you with problem determination and troubleshooting.

"Dealing with problems" on page 3363
Learn how to resolve some of the typical problems that can occur.

"Contacting your IBM Support Center" on page 3563
If you cannot resolve problems that you find when you use WebSphere Message Broker, or if you are directed to do so by an error message generated by WebSphere Message Broker, you can request assistance from your IBM Support Center.

Recovering after the broker fails

Check what recovery procedures are available, according to what has failed.

Before you begin

Try to get to the root of the problem first, using the diagnosis techniques described throughout the Chapter 13, "Troubleshooting and support," on page 3345 section of the information center. If your problem cannot be resolved using these techniques, contact your IBM Support Center. Use the procedure in this section only as a last resort.

If you are able to recover, and can start the broker again, the broker attempts to recover and re-establish all sessions with CMP applications that were active at the time of failure. CMP applications include the WebSphere Message Broker Explorer, the WebSphere Message Broker Toolkit, and applications that you have written to this API.

About this task

If you cannot correct the current problem by using problem determination, complete the following sequence of operations to re-create the broker:

Procedure

1. Stop the broker by using the **mqsistop** command.
2. Stop the broker queue manager by using the **endmqm** command.
3. Delete the broker by using the **mqsideletebroker** command.
4. Re-create the broker by using the **mqsicreatebroker** command.
5. Start the broker by using the **mqsistart** command.
6. Redeploy all resources to the broker.

Related tasks:

"Creating a broker" on page 611
You can create brokers on every platform that is supported by WebSphere Message Broker. The broker runs as a 64-bit application on all platforms except Linux on x86 and Windows on x86.

"Deleting a broker" on page 930
Delete a broker using the command line on the system where the broker component is installed.

"Recovering after failure" on page 3574
Follow a set of procedures to recover after a serious problem.

Related reference:

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsideletebroker** command" on page 3863
Use the **mqsideletebroker** command to delete a named broker. The command also deletes the queues on the associated queue manager (created when the broker was created). You can also specify that the queue manager is to be deleted.

"**mqsistart** command" on page 3965
Use the **mqsistart** command to start the specified broker if all initial verification tests complete successfully.

"**mqsistop** command" on page 3972
Use the **mqsistop** command to stop the specified component.

Recovering after an execution group fails

Follow the steps in this task to recover if an execution group fails.

Before you begin

Try to get to the root of the problem first, using the diagnosis techniques described throughout the Chapter 13, "Troubleshooting and support," on page 3345 section of the information center. If your problem cannot be resolved using these techniques, contact your IBM Support Center. Use the procedure in this section only as a last resort.

About this task

If a single execution group fails, and the problem cannot be corrected using problem determination, or by the IBM Support Center, perform the following sequence of operations to re-create the execution group:

Procedure

1. Delete the execution group.
2. Create an execution group of the same name.
3. Redeploy the configuration.

What to do next

If more than one execution group fails, you might need to re-create the broker. See "Recovering after the broker fails" on page 3575 for information on how to do this.

Related tasks:

"Recovering after failure" on page 3574
Follow a set of procedures to recover after a serious problem.

"Recovering after the broker fails" on page 3575
Check what recovery procedures are available, according to what has failed.

Recovering after the broker's queue manager fails

Check what recovery procedures are available, according to what has failed.

Before you begin

Try to get to the root of the problem first, by using the diagnosis techniques described throughout the Chapter 13, "Troubleshooting and support," on page 3345 section of the information center. If your problem cannot be resolved by using these techniques, contact your IBM Support Center. Use the procedure in this section only as a last resort.

About this task

If the broker's queue manager fails and cannot be corrected by using problem determination, or by the IBM Support Center, perform the following sequence of operations to re-create the queue manager:

Procedure

1. Ensure that no WebSphere Message Broker Toolkit users are deploying to the broker. You must wait until any such actions have completed.
2. Stop the broker by using the **mqsistop** command.
3. Delete the broker by using the **mqsideletebroker** command, with the **-q** parameter to remove the queue manager.
4. Re-create the broker by using the **mqsicreatebroker** command. The **mqsicreatebroker** command creates the queue manager and default queues automatically.
5. Re-create any specific queues that are needed for your message flows.
6. Start your brokers by using the **mqsistart** command.
7. Redeploy all resources to the broker to ensure that its configuration is consistent.

Related tasks:

"Creating a broker" on page 611
You can create brokers on every platform that is supported by WebSphere Message Broker. The broker runs as a 64-bit application on all platforms except Linux on x86 and Windows on x86.

"Deleting a broker" on page 930
Delete a broker using the command line on the system where the broker component is installed.

"Recovering after failure" on page 3574
Follow a set of procedures to recover after a serious problem.

Related reference:

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsideletebroker** command" on page 3863
Use the **mqsideletebroker** command to delete a named broker. The command also deletes the queues on the associated queue manager (created when the broker was created). You can also specify that the queue manager is to be deleted.

"**mqsistart** command" on page 3965
Use the **mqsistart** command to start the specified broker if all initial verification tests complete successfully.

"**mqsistop** command" on page 3972
Use the **mqsistop** command to stop the specified component.

Chapter 14. Reference

Use the reference information in this section to accomplish the tasks that address your business needs.

Migration and upgrade

Consider the factors involved in the migration of components and resources from Version 6.0 or Version 6.1 to Version 7.0.

This section contains the following topics:

Related concepts:

Chapter 4, "Installing and uninstalling," on page 231
Install and uninstall WebSphere Message Broker components and service.

Related tasks:

Chapter 3, "Migrating and upgrading," on page 137
To migrate a broker domain to WebSphere Message Broker Version 7.0, plan your migration strategy, perform pre-migration tasks, migrate your domain components, and then complete post-migration tasks.

Supported migration paths

You can migrate to WebSphere Message Broker Version 7.0 from previous versions of the product.

You cannot migrate from the Windows on x86 product (any version) to the Windows on x86-64 product. Similarly, you cannot migrate from the Linux on x86 product (any version) to the Linux on x86-64 product. You must re-create the brokers from scratch.

For the latest details of all supported levels of hardware and software, visit the WebSphere Message Broker Requirements website.

From ...	You can migrate to ...
WebSphere Message Broker [1]	WebSphere Message Broker Version 7.0
WebSphere Message Broker with Rules and Formatter Extension	WebSphere Message Broker Version 7.0[2]
WebSphere Message Broker Version 6.0[3]	WebSphere Message Broker Version 7.0

From ...	You can migrate to ...
WebSphere Message Broker with Rules and Formatter Extension Version 6.0[3]	WebSphere Message Broker Version 7.0[2]
WebSphere Event Broker Version 6.0[3]	• WebSphere MQ Version 7.0.1[4] • WebSphere Message Broker Version 7.0
WebSphere Event Broker Version 6.0[3]	• WebSphere MQ Version 7.0.0.1[5] • WebSphere Message Broker Version 7.0
WebSphere Message Broker with Rules and Formatter Extension Version 6.0[3]	• WebSphere Message Broker Version 7.0.0.1 or later[2] • WebSphere Message Broker with Rules and Formatter Extension Version 7.0[2]
WebSphere Message Broker with Rules and Formatter Extension [3]	• WebSphere Message Broker Version 7.0.0.1 or later[2] • WebSphere Message Broker with Rules and Formatter Extension Version 7.0[2]

Notes:

1. You can migrate WebSphere Message Broker at Fix Pack 3 (Version 6.1.0.3) or later. You must upgrade your Version 6.1 installation to use WebSphere MQ Version 7.0.1 before migration.

   If you are using WebSphere Application Server with WebSphere Message Broker, or you have publish/subscribe applications that use the SubIdentity option, you must upgrade WebSphere Message Broker to Fix Pack 4 before you can migrate to WebSphere Message Broker Version 7.0.
2. You can migrate to WebSphere Message Broker Version 7.0 only if you do not use the additional features provided by Rules and Formatter, or you choose not to use them after migration. If you want to use the additional features provided by Rules and Formatter, you need to migrate to WebSphere Message Broker Version 7.0.0.1 or later, with WebSphere Message Broker with Rules and Formatter Extension Version 7.0.
3. You can migrate Version 6.0 products at Fix Pack 9 (Version 6.0.0.9) or later for runtime components, and Version 6.0.2 or later for the WebSphere Message Broker Toolkit. You must upgrade your Version 6.0 installation to use WebSphere MQ Version 7.0.1 before migration.

   If you are using WebSphere Application Server with WebSphere Message Broker, or you have publish/subscribe applications that use the SubIdentity option, you must upgrade WebSphere Message Broker Version 6.0 or WebSphere Event Broker Version 6.0 to Fix Pack 10 before you can migrate to WebSphere Message Broker Version 7.0.
4. If you use only publish/subscribe functions with WebSphere Event Broker Version 6.0, your typical migration path is to WebSphere MQ Version 7.0.1. This migration path is handled completely by WebSphere MQ; see the *Migration* section of the WebSphere MQ Version 7 Information Center online.
5. If you use only publish/subscribe functions with WebSphere Event Broker Version 6.0, your typical migration path is to WebSphere MQ Version 7.0.0.1. This migration path is handled completely by WebSphere MQ; see the *Migration* section of the WebSphere MQ Version 7 Information Center online.

Related concepts:

Chapter 4, “Installing and uninstalling,” on page 231
Install and uninstall WebSphere Message Broker components and service.

Related tasks:

Chapter 3, “Migrating and upgrading,” on page 137
To migrate a broker domain to WebSphere Message Broker Version 7.0, plan your migration strategy, perform pre-migration tasks, migrate your domain components, and then complete post-migration tasks.

Related information:

WebSphere MQ Version 7 Information Center online

Installation

Use the reference information in this section to understand installation requirements, installation options, and how they affect your computer.

- “System requirements”
- “Installation and uninstallation interfaces” on page 3617
- “Installation and uninstallation authorization” on page 3628
- “Multicultural support” on page 3628
- System changes

Read the product `readme.html` file for late changes to the installation instructions; this file is provided on CD or DVD and installed with the product. You can find the most up-to-date version on the web, as described in “Finding the latest information” on page 232.

Product requirements are also available on the web, and are occasionally updated. Check for the latest information about WebSphere Message Broker Requirements.

Related tasks:

“Installing” on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

“Uninstalling” on page 331
Remove the Broker component, the WebSphere Message Broker Toolkit, or the WebSphere Message Broker Explorer from your computer.

Related information:

WebSphere Message Broker Requirements

System requirements

Use the reference information in this section to understand the hardware, software, and license requirements.

- “Hardware requirements” on page 3582
- “Software requirements” on page 3588
- “License requirements” on page 3606

The product readme file `readme.html` might contain updates to the information in this chapter. The readme file includes information pertinent to all components and platforms, and is maintained in US English on the product readmes web site:

`www.ibm.com/support/docview.wss?uid=swg27006913`

You must check this file to ensure that you have the latest information. Translated readme files are available on the documentation FTP site:

`ftp://public.dhe.ibm.com/software/integration/wbibrokers/docs/`

A readme file is included with the product; it contains a minimum level of information and directs you to the online version. It is available in these locations:

- Before installation, on the product media.

  The readme file is included in location `\readmes\locale\` (where *locale* identifies country, region, or language, for example `en_US`) on all of the following disks:
  - DVD (on Linux on x86, Linux on x86-64, and Windows only)
  - WebSphere Message Broker component Disk 1 (all operating systems)
  - WebSphere Message Broker component Disk 2 (Windows only)
  - WebSphere Message Broker Toolkit Disk 1
- After installation, in the installation directory:
  - For runtime components, in *install_dir*`\readmes\locale\` (where *locale* identifies country, region, or language, for example `en_US`).
  - For the WebSphere Message Broker Toolkit, in *package_group_directory*`\wmbt\documentation\locale\` (where *locale* identifies country, region, or language, for example `en_US`).

For further support information, including latest fixes and troubleshooting techniques, visit the WebSphere Message Broker support web page:

`www.ibm.com/software/integration/wbimessagebroker/support/`

The supported hardware and software environments are updated occasionally; view the latest requirements information about the WebSphere Message Broker Requirements web site:

`www.ibm.com/software/integration/wbimessagebroker/requirements/`

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

"Uninstalling" on page 331
Remove the Broker component, the WebSphere Message Broker Toolkit, or the WebSphere Message Broker Explorer from your computer.

Related information:

WebSphere Message Broker Requirements

Hardware requirements

View the processor and related hardware requirements on all platforms.

- "Supported processors" on page 3583
- "Memory and disk space requirements" on page 3584
- "Communications" on page 3588

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

"Finding the latest information" on page 232
Access the latest information for WebSphere Message Broker.

Related reference:

"Software requirements" on page 3588
View the operating system, database, and other software requirements.

"License requirements" on page 3606
Use the reference information in this section to understand license requirements.

Related information:

WebSphere Message Broker Requirements

Supported processors:

WebSphere Message Broker is supported on multiple processors.

The hardware requirements for each supported operating system are given in the following table. All support statements assume that the stated systems can run the required level of a compatible operating system and have enough storage for the WebSphere Message Broker components that you install, and all prerequisite products.

Table 31. Hardware requirements

Operating system	Requirements[1]
AIX	64-bit IBM System p systems Any hardware from IBM or other vendors that can run trademarked AIX systems[2]
HP-Itanium	Itanium systems
Linux on POWER	64-bit System i® and System p IBM POWER processor-based systems only
Linux on x86	IBM eserver System x or equivalent Intel-based servers[3]
Linux on x86-64	AMD64, EM64T, or compatible processor servers[3]
Linux on IBM z Systems	Any server capable of running one of the supported Linux on IBM z Systems releases
Solaris on SPARC	Sun Microsystems SPARC processor servers
Solaris on x86-64	AMD64, EM64T, and compatible processor servers
Windows 32-bit	Windows x86 technology-compatible PC hardware[3]
Windows 64-bit	AMD64, EM64T, or compatible processor servers[3]
z/OS[4]	Any server capable of running one of the supported z/OS releases

Notes:

1. Always check the WebSphere Message Broker Requirements web site and the `readme.html` file for the latest information about supported processors.

   The `readme.html` file that is supplied on the product DVD or CD (for all components) provides a minimum level of information, and directs you to the online file on the product readmes web page, which is updated regularly. Always use the online file to check that you have the latest level of information.
2. You can use AIX systems only if they have passed a set of verification tests for compliance with the AIX application binary and programming interfaces.
3. The WebSphere Message Broker Toolkit is supported on 32-bit and 64-bit systems. It requires a computer with an Intel Pentium III processor (or higher) that has a speed of at least 700 MHz. This specification is the minimum supported level; for improved performance use a 2 GHz processor.

A minimum display resolution of at least 1024 x 768 is required for some dialogs (for example, the Preferences dialog).

4. For further information, see the *Program Directory for WebSphere Message Broker for z/OS* on the WebSphere Message Broker Library web page.

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

"Finding the latest information" on page 232
Access the latest information for WebSphere Message Broker.

Related reference:

"Operating system requirements" on page 3590
WebSphere Message Broker is supported on multiple operating systems.

"Additional software requirements" on page 3598
WebSphere Message Broker requires additional software products to run successfully.

Related information:

WebSphere Message Broker Requirements

Memory and disk space requirements:

Check the memory and disk space that is required for your installation.

Requirements for memory and disk space depend on the installation operating system, and on the WebSphere Message Broker components and other products that you are installing.

Distributed systems

- 512 MB of RAM is required to support runtime operations (1 MB equals approximately 1 000 000 bytes).
- 512 MB of RAM is required to support WebSphere Message Broker Toolkit operations on Linux on x86, Linux on x86-64, or Windows. This specification is the minimum supported level; for improved performance, provide 1 GB (1 GB equals approximately 1 000 000 000 bytes).
- 512 MB of RAM is required to support the WebSphere Message Broker Explorer.
- Disk space requirements are dependent on the components that you install and the working space that is required by those components (for example, for WebSphere MQ queues and persistent messages).

  Check that your computer has at least the space shown in the table, which provides guidance for both permanent product requirements and temporary space. Check that these requirements have not been updated in the latest product readme file `readme.html`.

  If the installation directory and the temporary space are on the same partition or drive, add the two figures to check that you have enough space available. If you do not, increase the available storage or change the location of either the temporary space or the installation directory. The temporary files are deleted when installation is complete.

– On AIX, HP-UX, and Linux systems, the default temporary space directory is `/tmp`.
– On Solaris systems, the default temporary space directory is `/var/tmp`.
– On Windows, the default temporary space directory is pointed to by the *TEMP* system variable. On some systems, the variable *TMP* exists and is used before *TEMP*, therefore you must check or change the setting of both these variables.

The installation wizard displays requirements for permanent space, but not for temporary space. If the figure that the installation wizard displays is greater than the figure shown in the following tables, check that your computer has sufficient space before you continue with the installation.

Table 32. Disk space requirements (Linux and Windows systems)

Component / product	Linux on POWER	Linux on x86[1]	Linux on x86-64	Linux on IBM z Systems	Windows 32-bit[1]	Windows 64-bit[1]
Broker component	520 MB plus 330 MB temporary space	456 MB plus 330 MB temporary space	477 MB plus 330 MB temporary space	469 MB plus 330 MB temporary space	495 MB plus 330 MB temporary space	538 MB plus 330 MB temporary space
WebSphere Message Broker Toolkit	Unavailable	1.6 GB plus 1.6 GB temporary space	1.9 GB plus 220 MB temporary space	Unavailable	1.6 GB plus 1.6 GB temporary space	1.9 GB plus 220 MB temporary space
WebSphere Message Broker Explorer	Unavailable	250 MB plus 300 MB temporary space	250 MB plus 300 MB temporary space	Unavailable	200 MB plus 250 MB temporary space	200 MB plus 250 MB temporary space
WebSphere Message Broker ODBC Database Extender (IE02)	90 MB plus 90 MB temporary space	80 MB plus 80 MB temporary space	75 MB plus 75 MB temporary space	80 MB plus 80 MB temporary space	Unavailable	Unavailable

Note:

1. The space required for the WebSphere Message Broker Toolkit includes space for the shared resources directory and the package group directory.

Table 33. Disk space requirements (UNIX)

Component / product	AIX	HP-Itanium	Solaris on SPARC	Solaris on x86-64
Broker component	710 MB plus 330 MB temporary space	960 MB plus 330 MB temporary space	620 MB plus 330 MB temporary space	620 MB plus 330 MB temporary space
WebSphere Message Broker Toolkit	Unavailable	Unavailable	Unavailable	Unavailable
WebSphere Message Broker Explorer	Unavailable	Unavailable	Unavailable	Unavailable
WebSphere Message Broker ODBC Database Extender (IE02)	80 MB plus 80 MB temporary space	200 MB plus 200 MB temporary space	130 MB plus 130 MB temporary space	130 MB plus 130 MB temporary space

- On computers on which you create a broker, up to 60 MB is required for the broker configuration data within your file system.

- If you create user databases that are accessed by message flows, additional space is required on those computers.
- If you intend to create more than one broker on any one computer, you require additional memory and swap space. For example, you might create more than one broker at different versions to complete migration. Plan for 1 GB RAM and 1 GB of swap space for each broker, in addition to the space and memory requirements of other applications.

  You must increase these figures if you deploy complex message flows to the brokers, or if you process large messages (of many megabytes), or complex messages that contain many different tags.

z/OS Details are given in "Disk space requirements on z/OS." You must also check for later updates to this information in the section about DASD storage requirements in the *Program Directory for WebSphere Message Broker for z/OS* on the WebSphere Message Broker Library web page.

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

"Uninstalling" on page 331
Remove the Broker component, the WebSphere Message Broker Toolkit, or the WebSphere Message Broker Explorer from your computer.

Related reference:

"System requirements" on page 3581
Use the reference information in this section to understand the hardware, software, and license requirements.

"Disk space requirements on z/OS"
The installation of WebSphere Message Broker for z/OS uses approximately 400 MB of disk space; plan on using 500 MB to allow for the component directories, and for new service fixes to be applied.

Disk space requirements on z/OS:

The installation of WebSphere Message Broker for z/OS uses approximately 400 MB of disk space; plan on using 500 MB to allow for the component directories, and for new service fixes to be applied.

When you apply service, if you do not replace your existing installation (for example, you apply the new fix pack level alongside your existing installation), you must plan the same amount of disk space for the higher service level libraries.

If you are transferring the files by using *tar* to package them, you need approximately 200 MB of space for the `.tar` file.

You can check how much space is used and how much is free in a file system by using the OMVS command:

```
df -P /pathname
```

100 MB is 3 276 800 512 byte sectors.

The following table gives guidance on the space required for a minimum installation (base installation and verification test) of WebSphere Message Broker for each component implemented on z/OS.

Component	Space required	Purpose
Component directory	20 MB	Holds the runtime-deployed configuration information and output directories for the component. This information includes all deployment information, such as ESQL, JAR files, message sets, XSLT files, and so on. This information also includes WebSphere Message Broker trace files and other user problem determination data, which might become large. Consideration must be given to the potential size of deployments to the WebSphere Message Broker runtime environment and, therefore, the size of this directory (including sub directories).
Component PDSE	1 MB	Holds the customization and administration jobs, procedures, and data for the component. The data set must be allocated with a fixed record length of 80 (`LRECL=80`) and a format of FB 80. Reserve directory space for 50 members, or use a PDSE if possible.
Started task user ID home directory	8 GB	Collects diagnostic materials: for example, dumps. Dumps are usually more than 500 MB in size. 8 GB of space must be available in the file system, but many user IDs can have their home directory in the file system.

The Component directory and the Started task user ID home directory must be separate to ensure that, when dumps are taken in the Started task user ID home directory, they do not cause problems with the runtime broker that still has to write to the Component directory.

Related tasks:

"Customizing the z/OS environment" on page 591
If you are planning to use a z/OS environment, consider whether to create your brokers on z/OS. You must also complete a number of tasks to configure your environment.

"Using the file system on z/OS" on page 596
If you have more than one MVS image, consider how you will use the file system. You can share files in a file system with different members of a sysplex. The file system is mounted on one MVS image and requests to the file are routed to the owning system using XCF from systems which do not have it mounted.

Communications:

Your system must have communications hardware that supports the protocols that brokers can use.

Choose one or more of the following protocols:
- NetBIOS
- SNA LU 6.2
- SPX
- TCP/IP

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

Related reference:

"Supported processors" on page 3583
WebSphere Message Broker is supported on multiple processors.

"Memory and disk space requirements" on page 3584
Check the memory and disk space that is required for your installation.

Software requirements

View the operating system, database, and other software requirements.

This section provides information about requirements of WebSphere Message Broker:
- "Support for 32-bit and 64-bit platforms" on page 3589
- "Operating system requirements" on page 3590
- "Supported databases" on page 3591
- "Additional software requirements" on page 3598
- "Optional software support" on page 3603

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

"Finding the latest information" on page 232
Access the latest information for WebSphere Message Broker.

Related reference:

"Hardware requirements" on page 3582
View the processor and related hardware requirements on all platforms.

"License requirements" on page 3606
Use the reference information in this section to understand license requirements.

Related information:

WebSphere Message Broker Requirements

Support for 32-bit and 64-bit platforms:

WebSphere Message Broker operates in 32-bit mode or 64-bit mode, on supported operating systems.

The following table shows support for 32-bit and 64-bit mode. Changes since Version 6.1 are summarized beneath the table. If support on your platform has changed, you might need to make additional changes to your configuration. See "Migrating from Version 6.1 products" on page 163 and Chapter 7, "Configuring brokers for test and production environments," on page 579 for further details.

Table 34. Support for 32-bit and 64-bit operation

Platform	WebSphere Message Broker internal components and commands	32-bit execution groups	64-bit execution groups
AIX[1]	64-bit	No	Yes
HP-Itanium	64-bit	No	Yes
Linux on POWER	64-bit	No	Yes
Linux on x86	32-bit	Yes	No
Linux on x86-64[2]	32-bit	Yes	No
Linux on x86-64[2]	64-bit	No	Yes
Linux on IBM z Systems	64-bit	No	Yes
Solaris on SPARC[1]	64-bit	No	Yes
Solaris on x86-64	64-bit	No	Yes
Windows 32-bit	32-bit	Yes	No
Windows 64-bit[2]	32-bit	Yes	No
Windows 64-bit[2]	64-bit	No	Yes
z/OS[1]	64-bit	No	Yes

Changes since Version 6.1:

1. On this platform, you can no longer create or run with 32-bit execution groups. Execution groups can be only 64-bit mode.
2. On the Windows 64-bit and Linux on x86-64 platforms, you can install either the 32-bit or the 64-bit version of WebSphere Message Broker.

Execution groups

You can create an execution group by using one of the following:

- WebSphere Message Broker Toolkit
- WebSphere Message Broker Explorer
- **`mqsicreateexecutiongroup`** command
- CMP

If you do not specify the size of execution group that you require, the size that is created depends on how you create it, and the version of the target broker to which you deploy it. The following default values apply:

- If the target broker is Version 7.0 or later, the execution group is 64-bit except on those platforms that support 32-bit only (Linux on x86, and Windows).

Table 35. Execution group default sizes

Workbench	Command	CMP API	32-bit only platforms	32-bit and 64-bit platforms	64-bit only platforms
Default	no options	`createExecutionGroup(name)`	32-bit	64-bit	64-bit

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

"Finding the latest information" on page 232
Access the latest information for WebSphere Message Broker.

"Migrating from Version 6.1 products" on page 163
Migrate your components and resources to WebSphere Message Broker Version 7.0.

Chapter 7, "Configuring brokers for test and production environments," on page 579
Create one or more brokers on one or more computers, and configure them on your test and production systems to process messages that contain your business data.

"Creating an execution group using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer" on page 937
Use the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer to create execution groups on your broker.

"Creating an execution group using the **mqsicreateexecutiongroup** command" on page 939
Use the **mqsicreateexecutiongroup** command to create execution groups on your broker.

Related reference:

"Hardware requirements" on page 3582
View the processor and related hardware requirements on all platforms.

Operating system requirements:

WebSphere Message Broker is supported on multiple operating systems.

The following operating systems support WebSphere Message Broker V7.0:

- AIX
- HP-Itanium
- Linux on POWER
- Linux on x86
- Linux on x86-64
- Linux on IBM z Systems
- Solaris on SPARC
- Solaris on x86-64
- Windows 32-bit

- Windows 64-bit
- z/OS

The software requirements for these operating systems are defined in the List of supported software for WebSphere Message Broker V7.0 web page.

In all operating environments except z/OS, defect support is available for virtualization environments where they relate to releases that are already supported by WebSphere Message Broker. Unless stated elsewhere in the system requirements, WebSphere Message Broker has not been tested in virtualization environments. WebSphere Message Broker support is therefore unable to assist in issues related to configuration and setup, or issues that are directly related to the virtualization environment itself.

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

"Finding the latest information" on page 232
Access the latest information for WebSphere Message Broker.

Related reference:

"Supported processors" on page 3583
WebSphere Message Broker is supported on multiple processors.

"Supported databases"
You can optionally configure databases to contain data that is accessed by your message flows. Databases from IBM and other suppliers are supported at specific versions on supported operating systems.

"Additional software requirements" on page 3598
WebSphere Message Broker requires additional software products to run successfully.

Related information:

WebSphere Message Broker Requirements

Supported databases:

You can optionally configure databases to contain data that is accessed by your message flows. Databases from IBM and other suppliers are supported at specific versions on supported operating systems.

The information shown here indicates the support for databases on each operating system, valid when the information center was published. However, database support might be enhanced after the release is made available. For the latest information about database support, visit the WebSphere Message Broker Requirements website.

If you run message flows that access data that is held in databases, you must install and configure a supported database. Some data types supported by these databases are not supported by WebSphere Message Broker; for details, see "Data types of values from external databases" on page 5288.

In most environments, the broker and the database server must not be running on the same operating system. For details about local and remote database use, and the restrictions that apply, see "Database locations" on page 3595.

WebSphere Message Broker supports both transactional (XA) and non-transactional connections to databases. You can define an ODBC connection, or a JDBC type 4 connection, or both, to your database instances. XA support is referred to as a globally coordinated message flow.

On a single broker, you can use ODBC XA, or JDBC XA, but not both. This restriction applies to all supported platforms, and to all database servers for which XA is supported.

The following sections give details of this support, and describe restrictions where applicable:

- "ODBC support"
- "JDBC type 4 support" on page 3594

ODBC support

The following table lists the supported database servers for each broker platform for non-XA support. XA connections are supported by most of these servers: See the notes following the table for details.

For the latest details of the versions of the database servers that are supported, visit the WebSphere Message Broker Requirements website.

Unless otherwise stated, the ODBC client code for the database is at the same version and release as the server.

Operating system	DB2[1, 2]	Informix[5]	Microsoft SQL Server	Oracle[1, 3, 4]	solidDB	Sybase[1]
AIX	Supported	Supported	Supported[6]	Supported	Supported	Supported
HP-Itanium	Supported	Supported	Supported[6]	Supported	Supported	Supported
Linux on POWER	Supported	Supported	Not supported	Supported	Not supported	Supported
Linux on x86	Supported	Supported	Supported[6]	Supported	Supported	Supported
Linux on x86-64	Supported	Supported	Supported[6]	Supported	Supported	Supported
Linux on IBM z Systems	Supported	Supported	Supported[6]	Supported	Not supported	Not supported
i5/OS and OS/400®[7]	Supported	Not supported	Not supported	Not supported	Not supported	Not supported
Solaris on SPARC	Supported	Supported	Supported[6]	Supported	Supported	Supported
Solaris on x86-64	Supported	Supported	Not supported	Supported	Supported	Supported
Windows 7	Supported	Supported[8]	Supported	Supported[8]	Supported[8]	Supported[8]
Windows 7 (64-bit)	Supported	Not supported	Supported	Supported[9]	Supported[8]	Not supported
Windows XP and Server 2003	Supported	Supported[8]	Supported	Supported[8]	Supported[8]	Supported[8]
Windows Vista	Supported	Supported[8]	Supported	Supported[8]	Supported[8]	Supported[8]

Operating system	DB2[1, 2]	Informix[5]	Microsoft SQL Server	Oracle[1, 3, 4]	solidDB	Sybase[1]
Windows Server 2008	Supported	Supported[8]	Supported	Supported[8]	Supported[8]	Supported[8]
Windows Server 2008 R2	Supported	Not supported	Supported	Supported[9]	Supported[8]	Not supported
z/OS	Supported	Not supported	Not supported	Not supported	Not supported	Not supported

Notes:

1. XA support:

   Supported releases of DB2, Oracle, and Sybase can participate as a resource manager in a distributed XA transaction, and can be coordinated by WebSphere MQ as the XA Transaction Manager, unless otherwise stated in the following notes. On z/OS, all transactions are coordinated by Recoverable Resource Services (RRS).

   - Additional conditions forWebSphere MQ:
     - If you use WebSphere MQ Version 7 for XA coordination on Windows, you must configure the queue manager to run under the broker service user ID by specifying the **-si** option on the **strmqm** command; for example, `strmqm -si QM_name`
   - Additional conditions for databases:
     - DB2 on i5/OS and OS/400, is not supported for XA connections.
     - If you deploy message flows that access databases, you can define the message flows to be coordinated so that updates to those databases are synchronized with updates to other resources.
     - ODBC drivers for Oracle and Sybase on all relevant systems, and for SQL Server on Linux and UNIX systems, are supplied with WebSphere Message Broker. Alternative drivers are not supported on these systems. For other systems, and for other databases on all systems, obtain these files from your database vendor.
   - Additional conditions for z/OS
     - On z/OS all transactions are globally coordinated by using RRS. As with other platforms you can choose for ODBC database operations to be committed or rolled back irrespective of the success or failure of the message flow transaction as a whole. However only one uncoordinated ODBC database connection per thread is supported.
2. Automatic Client Reroute for DB2 is supported on all platforms.
3. If you install the Oracle Database Server on 32-bit systems, you must also install the Oracle Runtime Client.
4. On all Linux, UNIX, and Windows systems, you can use Oracle RAC (Real Application Clusters) databases. For Oracle RAC XA and non-XA, failover support is limited to the Connect-Time Failover feature only.
5. Each broker system requires only the Client SDK; install the Dynamic Server on the system on which you create databases.

   Large Objects (LOBs) are not supported.
6. On Linux and UNIX systems, you can remotely access an SQL Server database on Windows by using a supplied wire protocol driver.
7. You can configure message flows to access DB2 databases on i5/OS and OS/400 for user data. The message flows can run on all supported broker

platforms. For further details of these restrictions, and for information about the PTFs that are required with DB2 on these operating systems, see “Database locations” on page 3595.

8. Windows 32-bit edition of WebSphere Message Broker only.
9. WebSphere Message Broker for Windows 64-bit is supported only in non-XA environments.

JDBC type 4 support

For non-XA transactions, you can create a JDBC type 4 connection from a broker to all the database servers that are listed in the ODBC support table. Connections are supported from all broker platforms, including z/OS.

For XA connections, the following restrictions apply:

- On distributed platforms, only DB2 and Oracle are supported.
- On z/OS, XA connections are not supported.

JDBC type 4 drivers are not supplied with WebSphere Message Broker; obtain these files from your database vendor. For the latest details of the drivers that are supported, visit the WebSphere Message Broker Requirements website.

Related tasks:

“Finding the latest information” on page 232
Access the latest information for WebSphere Message Broker.

“Working with databases” on page 2109
Create and configure databases to use with your message flow applications.

Related reference:

“Database locations” on page 3595
The broker can access databases set up on the local computer or on a remote server, subject to restrictions.

“Data types of values from external databases” on page 5288
How database data types are implicitly cast to ESQL data types.

“Supported processors” on page 3583
WebSphere Message Broker is supported on multiple processors.

“Operating system requirements” on page 3590
WebSphere Message Broker is supported on multiple operating systems.

“Additional software requirements” on page 3598
WebSphere Message Broker requires additional software products to run successfully.

“Support for 32-bit and 64-bit platforms” on page 3589
WebSphere Message Broker operates in 32-bit mode or 64-bit mode, on supported operating systems.

Related information:

- WebSphere Message Broker Requirements
- DB2 V9.5 Information Center (distributed systems)
- DB2 V9.1 Information Center (distributed systems)
- DB2 Information Center (z/OS)

Database locations:

The broker can access databases set up on the local computer or on a remote server, subject to restrictions.

You can deploy message flows that access user databases to one or more of your brokers.

- You can use a local or remote database for user data, subject to the following restrictions:

  Databases on i5/OS and OS/400

  - You can use only DB2 (UDB) on iSeries (System i) under OS/400 or i5/OS.
  - You must install OS/400 V5R2, or i5/OS V5R3 or V5R4.
  - On Linux, UNIX, and Windows, you can connect using DB2 Connect. On Windows only, you can also use iSeries Access for Windows.
  - You cannot define globally coordinated (XA) transactions.
  - You can call stored procedures only if access to the remote database is provided by DB2 Connect.

  Databases on z/OS

  - You can use only DB2 (UDB) on z/OS.
  - You must use DB2 Connect on Linux, UNIX, and Windows.

    For further details, see the *Program Directory for WebSphere Message Broker for z/OS* on the WebSphere Message Broker Library web page.

  WebSphere Information Integrator databases

  - You can configure message flows on all distributed systems to access user data in WebSphere Information Integrator for Linux, UNIX, and Windows Version 9.1.

    You can also configure message flows to access user data, by using JDBC only, in WebSphere Information Integrator Classic Federation (IICF) for z/OS Version 9.1, on operating systems for which the WebSphere IICF client code is provided.
  - You can configure message flows on z/OS brokers to access user data, by using JDBC only, in WebSphere Information Integrator Classic Federation for z/OS Version 8.2. This support provides connectivity with the following z/OS data sources:
    - IMS
    - VSAM
    - ISAM
    - Sequential files
    - CA-IDMS
    - CA-Datacom
    - Software AG ADABAS
    - DB2

    Brokers can connect directly to WebSphere Information Integrator Classic Federation for z/OS Version 9.1 by JDBC only.

    Brokers can connect indirectly to WebSphere Information Integrator Classic Federation for z/OS Version 9.1 through an intermediate DB2 system on z/OS. If you use this configuration, only an ODBC CAF (Call Attachment Facility) connection is supported between the broker and DB2; this connection therefore does not support two-phase commit.

You must configure all broker ODBC data resources as either CAF or RRSAF (Recoverable Resource Services Attachment Facility). If you use CAF, database operations are not coordinated by RRS.

- If you use a remote database, you must configure the ODBC connection to the database correctly. Refer to the documentation for the database product that you are using for further information.

See the documentation for the database product that you are using with WebSphere Message Broker to determine the best options for your specific environment and requirements, and information about how to configure remote database access.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"User database connections" on page 2110
User databases contain business data that is written and accessed by deployed message flows. You must create connections from the broker to the user database by using ODBC or JDBC.

Related tasks:

"Working with databases" on page 2109
Create and configure databases to use with your message flow applications.

"Accessing databases from message flows" on page 2112
Create and configure message flows to access user databases.

"Configuring transactionality for message flows" on page 1290
A message flow runs in a single transaction, which is started when data is received by an input node, and can be committed or rolled back when all processing has completed.

Related reference:

"Supported databases" on page 3591
You can optionally configure databases to contain data that is accessed by your message flows. Databases from IBM and other suppliers are supported at specific versions on supported operating systems.

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Sample WebSphere Message Broker ODBC Database Extender (IE02) configuration files:

How you use WebSphere Message Broker ODBC Database Extender (IE02) to load the correct data source driver.

To use extended database support, the following configuration files need to be set up, and be accessible from the environment in which you are running the broker.

solidDB data sources

Client side

odbc.ini file

The `odbc.ini` configuration file is referenced by the environment variable `ODBCUOINI` on Linux and UNIX systems, or in the ODBC configuration on Windows.

The odbc.ini file must present, at least, the following two items for each data source:

- ODBC Data Source Name
- Logical name of the actual ODBC driver being used

For example, on AIX:

```
[ODBC DATA SOURCES]
SOLIDDB_DB=IBM solidDB Datasource

[SOLIDDB_DB]
Description=solidDB Datasource
Driver=solid_db
```

Note that all additional information is ignored.

odbcinst.ini file

The directory containing the odbcinst.ini file is referenced by the environment variable ODBCSYSINI on Linux and UNIX systems, or in the ODBC configuration on Windows.

The odbcinst.ini file must present, at least, the following two items for each data source:

- The logical name of the driver
- The full path to, and name of, the ODBC driver

For example, on AIX:

```
[solid_db]
Driver=<Your solidDB install Directory>/bin/saca5x6465.so
```

solid.ini file

This configuration file is located in the directory referenced by the environment variable SOLIDDIR.

The solid.ini mapping is from the data source name (as defined in ODBCUOINI) to the solidDB connection string.

The connection string takes the form <logical name of the driver> = <physical solidDB connect string>.

The Physical connection string specifies the:

- Protocol
- Machine name or IP address
- Port number to use

For example, on AIX:

```
[Data Sources]
SOLIDDB_DB=tcp my_aix_system 1964
```

Server side

solid.ini file

For example, on AIX:

```
[Data Sources]
SOLIDDB_DB=tcp my_aix_system 1964

[COM]
Listen=tcpip 1964

[SQL]
CharPadding=yes
NumericPadding=yes
```

Related tasks:

"Connecting to a database from Linux and UNIX systems using the WebSphere Message Broker ODBC Database Extender (IE02)" on page 682
WebSphere Message Broker ODBC Database Extender encapsulates the unixODBC driver manager and this topic describes how you set up and configure the broker to use it.

Additional software requirements:

WebSphere Message Broker requires additional software products to run successfully.

- "WebSphere MQ"
- "Microsoft Visual C++ " on page 3599
- Java runtime environment
- "IBM Installation Manager" on page 3600
- "Browsers" on page 3600

WebSphere MQ

All WebSphere Message Broker components require WebSphere MQ at the minimum supported level shown in the table.

WebSphere MQ Version 7.0.1 (with tailored terms and conditions for use with WebSphere Message Broker) is supplied on DVD on Linux on x86, Linux on x86-64, and Windows only, and on CD on all other platforms. If you have a previous version of WebSphere MQ, you can use the supplied CD or DVD to upgrade your current installation.

A broker requires a WebSphere MQ queue manager. More than one broker cannot share a single queue manager.

Table 36. WebSphere MQ requirements

Operating system	Requirements
All distributed systems[1]	WebSphere MQ Version 7.0.1 (or later)[2, 3]
z/OS[4]	WebSphere MQ Version 7.0.1 (or later) with MQ Java Classes feature

Notes:

1. WebSphere MQ is not required on Linux on x86, Linux on x86-64, or Windows systems on which you install only the WebSphere Message Broker Toolkit.

   You can configure SSL connections between the WebSphere Message Broker Toolkit or the WebSphere Message Broker Explorer and the broker. To configure an SSL connection you must have an SSL certificate manager installed on the computer on which you have installed the WebSphere Message Broker Toolkit or the WebSphere Message Broker Explorer. If you choose to use WebSphere MQ to provide SSL management, install either the Client or the Server; both components include IBM Key Management tools.
2. The minimum set of components that you must install are the server and Java Messaging components.

   You can install WebSphere MQ before or after you have installed WebSphere Message Broker.

If you have already installed WebSphere MQ, check that your installation includes the Java Messaging component; add it from the WebSphere MQ media if it is not installed.

If you start the WebSphere MQ installation program directly on any platform, including Windows, select a custom installation and include the server and Java Messaging components.

If you want to use the WebSphere MQ Explorer, the graphical interface that is available on Linux on x86, Linux on x86-64, and Windows only, install the WebSphere Eclipse Platform Version 3.3 and the WebSphere MQ Explorer components. The WebSphere MQ Explorer is a required prerequisite if you want to use the WebSphere Message Broker Explorer.

Other WebSphere MQ components are optional.

3. On Windows, before WebSphere MQ Version 7.1 all configuration information, and most queue manager configuration information, was stored in the Windows registry. From WebSphere MQ Version 7.1 onwards all configuration information is stored in files. See the WebSphere MQ documentation for further information.
4. On z/OS, WebSphere MQ is a mandatory requirement and must be installed before you install WebSphere Message Broker.

   See the *Program Directory for WebSphere Message Broker for z/OS* for further details about required levels of WebSphere MQ.

For details of WebSphere MQ products and supported versions, see the WebSphere MQ product information website.

Microsoft Visual C++

When you install the Windows 64-bit version of WebSphere Message Broker on Windows 7 or Windows Server 2008 R2 only, Microsoft Visual C++ Runtime V10 for 32-bit and for 64-bit are required. The Visual C++ Runtime V10 installers supplied with the product media are in US English only. The installer for the Broker component installs Visual C++ Runtime V10 automatically for you. If you want to install Visual C++ Runtime V10 separately, the installers are on the root of Runtime Disk 1.

If you want to install a multicultural version of this product, which displays a translated installation interface and product license agreement, but is otherwise identical to the US English version, you must download both the required versions that you want from the Microsoft website and install them before you install the WebSphere Message Broker runtime.

JRE

A Java runtime environment (JRE) is required on all platforms:

- On distributed platforms, Java Runtime Environment (JRE) Version 6 is embedded in product components.
- On z/OS, you must acquire and install a JRE. See WebSphere Message Broker Requirements for the latest information on JRE requirements for z/OS.

  For more information, see the *Program Directory for WebSphere Message Broker for z/OS* available on the IBM Publications Center website. In particular, review the information in "Preventative Service Planning".

WebSphere Message Broker supports all JMS providers that conform to the Java Message Service Specification, version 1.1, and requires the minimum JRE levels

stated. You must consider both these factors when you select a JMS provider whose client is embedded in the broker.

Browsers

For best results when viewing the information center from the WebSphere Message Broker Toolkit, use Mozilla 1.4.2 (or later) on Linux on x86 and Linux on x86-64.

On Windows, an embedded browser is launched to view the information center.

Some Linux on x86 and Linux on x86-64 offerings do not install Mozilla by default. If you plan to install the WebSphere Message Broker Toolkit on your Linux on x86 or Linux on x86-64 system, check that a supported version of Mozilla is already installed. If not, install Mozilla from your Linux on x86 or Linux on x86-64 operating system media.

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

"Finding the latest information" on page 232
Access the latest information for WebSphere Message Broker.

"Setting up a command environment" on page 213
After you have installed the product on one of the distributed systems, you must initialize the environment before you can use a runtime component or command.

"Customizing the version of Java on z/OS" on page 607
Check the version of Java in your enterprise, and change it, if necessary.

"Considering security for the WebSphere Message Broker Toolkit and WebSphere Message Broker Explorer" on page 500
Set up appropriate levels of security for the WebSphere Message Broker Toolkit and WebSphere Message Broker Explorer.

Related reference:

"IBM Installation Manager"
WebSphere Message Broker Toolkit is installed by the IBM Installation Manager.

"Hardware requirements" on page 3582
View the processor and related hardware requirements on all platforms.

"Operating system requirements" on page 3590
WebSphere Message Broker is supported on multiple operating systems.

Related information:

WebSphere Message Broker Requirements

IBM Installation Manager:

WebSphere Message Broker Toolkit is installed by the IBM Installation Manager.

All Rational products at Version 7.0 or later are installed by IBM Installation Manager, which also controls management, updates, licensing, and uninstallation. The WebSphere Message Broker Toolkit includes some Rational product components, and therefore includes and uses Installation Manager.

The following Rational products are controlled by Installation Manager:

- WebSphere Message Broker Toolkit Version 7.0 or later
- Rational Application Developer (RAD) Version 7.5 or later
- Rational Software Architect (RSA) Version 7.0 or later
- WebSphere Integration Developer (WID) Version 7.0.0.1 or later

Installation Manager is included with the WebSphere Message Broker Toolkit and with the other products in this list. When you first install the WebSphere Message Broker Toolkit (or another listed product), Installation Manager installs itself into a directory that you specify, and then drives the installation of the WebSphere Message Broker Toolkit (or other listed product). If you install another product, Installation Manager detects that it is already installed, and drives only the installation of that product.

The WebSphere Message Broker Toolkit shares certain resources with these other products, if installed; for example, Eclipse features and plug-ins. All common resources that are used by the installed products must be installed into a single directory, which is known as the *shared resources directory*. You are asked to specify the location of this directory when you first install the WebSphere Message Broker Toolkit or another Rational product.

If you install another product, Installation Manager finds the shared resources directory and uses its content during the installation process; you cannot change the location of this directory.

The shared resources directory must be on a drive that is local to this computer; you cannot specify a mapped or remote drive. The drive that you specify for the shared resources directory must be of sufficient size to handle all your expected installations; you cannot change or expand this directory after installation. When you specify the directory during a first installation, specify a new directory to ensure that it does not contain any files that might cause conflicts.

Memory requirements for the WebSphere Message Broker Toolkit are listed in "Memory and disk space requirements" on page 3584. If you plan to install additional Rational products, allow 2 GB for each additional product.

You must also allocate space in another directory in which you can manage the workspace resources that you create for the installed Rational products.

Package groups

When you install the WebSphere Message Broker Toolkit, you are also asked to specify a package group. Products that you install into a single package group share Eclipse features and plug-ins, and these resources are loaded and viewable in a single Eclipse instance. You can choose whether to install a product in a package group with other products, or to install WebSphere Message Broker Toolkit in a new package group.

Each package group is isolated from products in other package groups, although all package groups access a single shared resources directory. You are asked to specify the location for the package group directory; you must specify a new directory for each new package group. All product-specific files are installed into this directory.

You might choose to use separate package groups to install different combinations of the WebSphere Message Broker Toolkit and other Rational products, so that

users can gain access to tailored Eclipse instances. When you install the first product, the first package group is created with the name **IBM WebSphere Message Broker Toolkit**. This name is fixed; you cannot change it.

If you choose to install another product into a new package group, another group is created with the name **IBM Software Development Platform_1**. Each new package group name follows this same naming pattern with the numeric suffix incremented by one.

For example, you might have defined the following package groups:

- **IBM WebSphere Message Broker Toolkit** into which you have installed the WebSphere Message Broker Toolkit and RAD.
- **IBM WebSphere Message Broker Toolkit_1** into which you have installed the WebSphere Message Broker Toolkit and RSA.
- **IBM WebSphere Message Broker Toolkit_2** into which you have installed WID and RSA.

When you start an Eclipse session in one of these package groups, you can access only those resources that are associated with the products installed in that group.

If you install later versions of any of the products in a different package group, the updates are available only in that group. The shared resources directory is also updated with later versions of shared files, which are maintained separately from the original versions and are used only for the upgraded products.

Each Rational product specifies which versions of plug-ins and features it requires, if appropriate. Installation Manager ensures the integrity of these requirements in each package group. If the product that you are currently installing breaks this integrity, Installation Manager prevents the installation into that package group.

Installation Manager also controls uninstallation of the WebSphere Message Broker Toolkit and the other products previously listed; you cannot uninstall Installation Manager until all listed products in all package groups have been removed.

Installation Manager is not required for any of the Broker component prerequisite products, such as WebSphere MQ.

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

"Finding the latest information" on page 232
Access the latest information for WebSphere Message Broker.

Related reference:

"Additional software requirements" on page 3598
WebSphere Message Broker requires additional software products to run successfully.

Related information:

WebSphere Message Broker Requirements

Optional software support:

The products listed here are not required, but might be useful. Except where stated, these products are not supplied with WebSphere Message Broker.

- "EIS client libraries"
- "WebSphere Adapters"
- "Tivoli License Manager"
- "Security providers"
- "JMS providers"
- "Compilers" on page 3604
- "Adobe Flash Player" on page 3604
- Citrix XenApp
- "WebSphere Process Server and WebSphere Integration Developer" on page 3604
- "WebSphere Service Registry and Repository" on page 3604

EIS client libraries

If you plan to use WebSphere Adapters on any supported operating system, you must obtain the appropriate Enterprise Information System (EIS) client libraries from the relevant EIS vendor (for example, PeopleSoft, Siebel, or SAP). The client libraries are required to complete connections between Adapters nodes and the EIS; install them on each computer on which you run a broker that hosts message flows that include Adapters nodes.

WebSphere Adapters

For details of the versions of WebSphere Adapters supported by WebSphere Message Broker Version 7.0, visit the WebSphere Message Broker Requirements website. For more information about the hardware and software requirements of WebSphere Adapters, see Supported software for WebSphere Adapters.

Tivoli License Manager

To monitor the use of software products including WebSphere Message Broker, install IBM Tivoli License Manager (ITLM). For the details of the version of ITLM that is required, visit the WebSphere Message Broker Requirements website.

Security providers

The message flow security manager interacts with external security providers:

- Lightweight Directory Access Protocol (LDAP) provides authentication and authorization services. For details of the LDAP version that your LDAP server must support, visit the WebSphere Message Broker Requirements website.
- WS-Trust V1.3 STS providers (including TFIM V6.2) provide authentication, mapping, and authorization services.
- Tivoli Federated Identity Manager (TFIM) V6.1 provides authentication, mapping, and authorization services.

JMS providers

For details of the version of the Java Message Service Specification that the JMSInput and JMSOutput nodes are compatible with, and work with, visit the WebSphere Message Broker Requirements website.

Compilers

C/C++ user-defined extensions that were built with older levels of compilers can be used when migrating an existing broker or setting up a new broker.

However, C/C++ user-defined extensions on Linux might cause problems if linked against libstdc++.so.5 as WebSphere Message Broker is now linked against libstdc++.so.6. Recompile your user-defined extensions using a supported compiler.

For details of the versions of the compilers that are supported, visit the WebSphere Message Broker Requirements website.

Adobe Flash Player

If you want to start the Quick Tour from the information center in the WebSphere Message Broker Toolkit, you must install the Adobe Flash Player. For details of the versions of the Adobe Flash Player that are supported, visit the WebSphere Message Broker Requirements website. You can freely download from the Adobe website an Adobe Flash Player plug-in for the web browsers that are supported by the WebSphere Message Broker Toolkit.

WebSphere Process Server and WebSphere Integration Developer

The SCA nodes allow interoperability with WebSphere Process Server, a business integration server that supports solutions that are based on service-oriented architecture (SOA). Service Component Architecture (SCA) is a specification that describes a model for building applications and systems using SOA and the SCA nodes support scenarios in which service components that run on WebSphere Process Server either are invoked from WebSphere Message Broker or invoke a WebSphere Message Broker message flow. For details of the versions of WebSphere Process Server that are supported by WebSphere Message Broker, visit the WebSphere Message Broker Requirements website.

WebSphere Integration Developer is the development environment for WebSphere Process Server. It is the tool for building and deploying SOA-based integration solutions on WebSphere Process Server. To enable WebSphere Message Broker to interoperate with WebSphere Process Server, SCA import and export components need to be imported into WebSphere Message Broker from WebSphere Integration Developer, and Broker SCA definitions need to be exported from WebSphere Message Broker to WebSphere Integration Developer. For details of the versions of WebSphere Integration Developer that are supported by WebSphere Message Broker, visit the WebSphere Message Broker Requirements website.

WebSphere Service Registry and Repository

WebSphere Service Registry and Repository provides a central repository of documents that describe services, service interfaces, and associated policies. For details of the versions of WebSphere Service Registry and Repository that are supported by WebSphere Message Broker, see the WebSphere Message Broker Requirements website.

Related concepts:

"WebSphere Service Registry and Repository" on page 1875
The WebSphere Service Registry and Repository (WSRR) is a central repository of entities. A wide range of entities can be stored and retrieved, including user-defined concepts and definitions related specifically to Web services, such as

WSDL services, service interfaces, and associated policies.

"Working with Service Component Architecture (SCA)" on page 2095
Start here to find out how you can use SCA to allow interoperability with WebSphere Process Server Version 6.2.

Related tasks:

"Finding the latest information" on page 232
Access the latest information for WebSphere Message Broker.

Chapter 10, "Testing and debugging message flow applications," on page 3143
Use the flow debugger to track messages through your message flows and check for errors, or use the Test Client to test your message flows in a safe environment before they are used on a production system.

"Installing Tivoli License Manager" on page 301
IBM Tivoli License Manager (ITLM) enables you to monitor the use of IBM (and other) software products. WebSphere Message Broker includes support for ITLM Version 2.1.

"Compiling a C user-defined extension" on page 3047
Compile user-defined extensions in C for all supported operating systems.

"Compiling a Java user-defined node" on page 3074
When you have created the code for your Java user-defined node, you must compile it for your operating system.

Quick Tour

System requirements for Citrix XenApp:

This topic gives information about licensing issues, and the software and hardware that you need to use WebSphere Message Broker in a Citrix XenApp environment.

Operating system

The required operating system is Windows 2003 Server. Client versions of Windows, such as Windows XP, cannot be used because Citrix XenApp requires Terminal Server, which is not available on client versions of Windows.

The recommended version of Windows Server 2003 is Enterprise Edition. You can also use Windows Server 2003 Standard Edition, but you might need to reduce the maximum heap size for the Java virtual machine that is used by the WebSphere Message Broker Toolkit.

Software prerequisites

Citrix XenApp (previously known as Citrix Presentation Server 4.0 or 4.5) and its prerequisites must be installed on the Windows 2003 server.

Hardware prerequisites

The server must have enough CPU, RAM, disk space, and network bandwidth for each concurrent user. Performance depends on the tasks that are being performed, and on the hardware, software and networking setup. As a guide, the server should have a minimum of 1 GB of RAM per concurrent user and at least 4 GB of RAM in total. If the tasks require frequent disk access, it should help performance to have a separate physical disk for each user so that they do not slow each other down.

Licensing

Any number of users can use the WebSphere Message Broker Toolkit through Citrix, provided they all connect only to a Message Broker that has a valid license.

Related concepts:

"Users and Citrix" on page 304
Review the categories of user that might want to use with Citrix, and how to configure for concurrent multi-users.

Related tasks:

"Publishing in a Citrix XenApp environment" on page 302
Supply application location and user details to Citrix to publish a WebSphere Message Broker command console or WebSphere Message Broker Toolkit.

License requirements

Use the reference information in this section to understand license requirements.

You can install WebSphere Message Broker to support a full range of transformation and routing operations. If appropriate, you can install an edition that supports a restricted set of functions, if that subset fulfills your business requirements. You must ensure that your use and configuration of the product conforms to the license agreement that you have purchased:

- WebSphere Message Broker Trial Edition. You can download this edition from the Web, at no charge. This edition has its own license and terms and conditions which is valid for 90 days. You can use all available function, and are not limited in the number of resources that you create and maintain.
- WebSphere Message Broker Starter Edition. If you expect to use all or most of the features that are available, but you configure a limited environment because of low capacity requirements, purchase this edition. You can use all available function, but are limited in the number of resources that you create and maintain.
- WebSphere Message Broker Entry Edition. If you expect to use only a basic set of features and have only low capacity requirements, purchase this edition. You can use a limited set of nodes, and you are limited in the number of resources that you create and maintain.
- WebSphere Message Broker Remote Adapter Deployment. If you expect your typical use of WebSphere Message Broker to be integration with Enterprise Information Systems (EIS), purchase this edition. This edition supports the subset of development resources that provide EIS interaction.
- WebSphere Message Broker. If you want to set up a full broker environment that uses most or all of the features available, you require a full (unrestricted) license.

WebSphere Message Broker for z/OS is unavailable in Remote Adapter Deployment, Starter Edition, Entry Edition, and Trial Edition.

If you choose to change your license agreement from a full license to either of the two specialized licenses, you might find that your current configuration is no longer supported. For further details about what features are available for each license, and how to configure your environment, see "WebSphere Message Broker technical overview" on page 27.

You can upgrade to the full license from another edition, if appropriate, by purchasing another license.

Your license also covers use of the product for development and unit test purposes. All developers in your organization, who are working on resources and applications for WebSphere Message Broker, can install one copy of all components on their computer. They can create and configure a broker environment without any functional or resource restrictions. Installation of the WebSphere Message

Broker Toolkit limits this use to Windows, Linux on x86, and Linux on x86-64 computers. The unit test environment is limited to these three platforms even if you have purchased a license for WebSphere Message Broker for z/OS.

You can also install the supplied WebSphere MQ product on the computers on which your developers create their development and test configurations, regardless of the license agreement that you have purchased.

You can view licenses after installation in your chosen language in directory *install_dir*/license/. Terms and conditions are also supplied for third-party products used by WebSphere Message Broker. The file containing these details is stored in the same license subdirectory when you install one or more runtime components.

Contact your IBM representative if you want further details about license agreements, or if you want to purchase additional licenses or change the type of license that you have purchased.

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

"Finding the latest information" on page 232
Access the latest information for WebSphere Message Broker.

Related reference:

"Hardware requirements" on page 3582
View the processor and related hardware requirements on all platforms.

"Software requirements" on page 3588
View the operating system, database, and other software requirements.

Related information:

WebSphere Message Broker Requirements

General industry standards supported by WebSphere Message Broker

WebSphere Message Broker supports general industry standards that are associated with message processing.

Supported standards

The following table details the general standards that are supported by WebSphere Message Broker, with additional comments and supported versions where appropriate.

Standard	Comments or supported releases in Version 7.0
Java	1.6 (Version 6)
SSL	
SOAP	1.1, 1.2
TLS	
WSDL	1.1

Standard	Comments or supported releases in Version 7.0
WS-Addressing	1.0 (Final 2005/08), Submission 2004/08
WS-Security	1.0
XPath	1.0
XSD	1.0
XSLT	1.0

Related concepts:

"WebSphere Message Broker technical overview" on page 27
WebSphere Message Broker enables information packaged as messages to flow between different business applications, ranging from large traditional systems through to unmanned devices such as sensors on pipelines.

Installation packages

View the installation packages that are available, and the contents of those packages.

This section contains the following topics:

- "Packaging options"
- "Package contents" on page 3610
- "The Broker component and WebSphere Message Broker Explorer package" on page 3611
- "Toolkit package" on page 3614
- "DVD package" on page 3616

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

"Finding the latest information" on page 232
Access the latest information for WebSphere Message Broker.

Related information:

WebSphere Message Broker Requirements

Packaging options

Both physical media and electronic images are available for the installation of WebSphere Message Broker.

Physical media

You can order the physical media for WebSphere Message Broker Starter Edition, Entry Edition, Remote Adapter Deployment, and the full (unrestricted) licenses. Contents are described in "Package contents" on page 3610.

When you install the product from the media, configure your brokers to operate in the mode that conforms to the license you have purchased. See the information center for further details.

Electronic images

If you are registered with IBM Passport Advantage, you can download electronic images from IBM Passport Advantage for WebSphere Message Broker Starter Edition, Entry Edition, Remote Adapter Deployment, and the full (unrestricted) licenses. You can also request one set of the physical media.

The electronic images exactly mirror the physical media that are described in "Package contents" on page 3610, although are not formatted as CD or DVD images. For further information, and to register, access the IBM Passport Advantage website.

Electronic images are available on request for WebSphere Message Broker for z/OS. Contact your IBM representative for further information and assistance.

When you install the product from these images, you must configure your brokers to operate in the mode that conforms to the license that you have purchased. See the information center for further details.

Trial Edition electronic images

For distributed systems only, you can download electronic images for WebSphere Message Broker Trial Edition from the developerWorks WebSphere Message Broker Trial package website.

Images are provided for the Broker component and the WebSphere Message Broker Toolkit. Use this edition to assess how the product can address your business requirements, and explore how you might use it with existing software in your enterprise.

For the latest information about supported trial versions, always check the WebSphere Message Broker `readme.html` file on the product readmes website.

The following restrictions apply:

- All product features are available, but the Broker component operates for only 90 days after installation.
- Electronic images of WebSphere MQ are not included with the WebSphere Message Broker Trial Edition. If you have not already ordered and installed WebSphere MQ, you can download a trial package from the Web. The WebSphere MQ trial versions include the required Java and Eclipse components. Download from the WebSphere MQ Trial package website.
- Physical images are not available for the Trial Edition.
- The Trial Edition is not available for z/OS.

If you choose to buy WebSphere Message Broker during or after the trial period, and want to continue to use the product components that you have installed, you do not have to reinstall them, but you must reconfigure existing brokers, and create new brokers, in the mode that conforms to the license that you have purchased. You can retain all the associated resources that you have developed or imported during the trial period. For further information, see "Changing the operation mode of your broker" on page 655.

Unless otherwise stated in this book, you can use electronic images in the same way as the physical CDs or DVDs, and all installation and set up procedures described are identical for the trial and full packages.

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

"Finding the latest information" on page 232
Access the latest information for WebSphere Message Broker.

Related reference:

"Installation packages" on page 3608
View the installation packages that are available, and the contents of those packages.

Related information:

WebSphere Message Broker Requirements

Package contents

Contents of the physical media packages for WebSphere Message Broker Starter Edition, Remote Adapter Deployment, Entry Edition, and the full (unrestricted) licenses. The electronic images that you can download have equivalent content.

Content for the Trial Edition packages is restricted; differences are described where they apply.

The contents of the package depend on the product that you have ordered:

WebSphere Message Broker

The package includes product code for all supported distributed operating systems, plus other optional software and documentation:

- The Quick Start CD, which contains documentation in PDF format. This CD is always at the top of the package. The CD includes the following documentation:
  - PDF files for the Quick Start Guide (US English and translations).
  - PDF files for the Installation Guide (US English and translations).
- A plastic wallet that contains four DVDs, one for Linux on x86, one for Linux on x86-64, one for Windows 32-bit, and one for Windows 64-bit. The DVDs contains all required and optional product code. The structure of the DVD content is described in "DVD package" on page 3616.

  The WebSphere Message Broker Toolkit package is contained on the DVDs. For more information about the contents of the WebSphere Message Broker Toolkit package, see "Toolkit package" on page 3614.
- A set of plastic wallets that contain CDs for installation of the broker, grouped by operating system, for systems other than Linux on x86, Linux on x86-64, and Windows. The CDs are listed in "The Broker component and WebSphere Message Broker Explorer package" on page 3611.
- The Quick Start Guide, printed in US English, French, and Japanese.

WebSphere Message Broker for z/OS

The package includes product code for the z/OS operating system on tape, plus other optional software and documentation. In addition, you receive WebSphere Message Broker for Linux on x86, Linux on x86-64, and Windows, because the WebSphere Message Broker Toolkit is available only on those operating systems.

The electronic images that you can download have equivalent content.

- A plastic wallet that contains four DVDs, one for Linux on x86, one for Linux on x86-64, one for Windows 32-bit, and one for Windows 64-bit. The DVDs contain all required and optional product code. The structure of the DVD content is described in "DVD package" on page 3616.

  The WebSphere Message Broker Toolkit package is contained on the DVDs. For more information about the contents of the WebSphere Message Broker Toolkit package, see "Toolkit package" on page 3614.
- z/OS tapes

  For information about tapes supplied with WebSphere Message Broker for z/OS, see the *Program Directory for WebSphere Message Broker for z/OS* on the WebSphere Message Broker Library web page.

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

"Finding the latest information" on page 232
Access the latest information for WebSphere Message Broker.

Related reference:

"Installation packages" on page 3608
View the installation packages that are available, and the contents of those packages.

Related information:

WebSphere Message Broker Requirements

The Broker component and WebSphere Message Broker Explorer package:

The Broker component and WebSphere Message Broker Explorer packages contain CDs and images supplied for WebSphere Message Broker and associated products.

The contents listed in the following table are supplied for installation of the Broker component. WebSphere MQ images are not included in the Trial Edition.

On Linux on x86, Linux on x86-64, and Windows, the product code is delivered on DVD. The structure of the DVD content is described in "DVD package" on page 3616. On these operating systems, the following table shows the contents of the downloaded images.

The physical package for the DVD is marked with the symbol 

.

On operating systems other than Linux on x86, Linux on x86-64, Windows, and z/OS, the product code is delivered on CD.

The physical packages for the CDs are marked with the symbol 
.

Table 37. CDs and images supplied for WebSphere Message Broker and associated products

Operating System[1]	CD label	Description
AIX	WebSphere Message Broker Version 7.0 AIX (Runtime Disk 1)	Product code[2]
	WebSphere MQ Version 7.0.1 AIX (Runtime Disk 2)	Product code
	WebSphere Message Broker ODBC Database Extender (IE02) (Runtime Disk 1)	Product code[2]
HP-Itanium	WebSphere Message Broker Version 7.0 HP-Itanium (Runtime Disk 1)	Product code [2]
	WebSphere MQ Version 7.0.1 HP-Itanium platform (Runtime Disk 2)	Product code
	WebSphere Message Broker ODBC Database Extender (IE02) (Runtime Disk 1)	Product code[2]
Linux on POWER	WebSphere Message Broker Version 7.0 Linux on POWER (Runtime Disk 1)	Product code[2]
	WebSphere MQ Version 7.0.1 Linux on POWER platform (Runtime Disk 2)	Product code
	WebSphere Message Broker ODBC Database Extender (IE02) (Runtime Disk 1)	Product code[2]
Linux on x86	WebSphere Message Broker Version 7.0 Linux on x86 (Runtime Disk 1)	Product code[2]
	WebSphere MQ Version 7.0.1 Linux on x86 (Runtime Disk 2)	Product code
	WebSphere Message Broker ODBC Database Extender (IE02) (Runtime Disk 1)	Product code[2]
Linux on x86-64	WebSphere Message Broker Version 7.0 Linux on x86-64 (Runtime Disk 1)	Product code[2]
	WebSphere MQ Version 7.0.1 Linux on x86-64 platform (Runtime Disk 2)	Product code
	WebSphere Message Broker ODBC Database Extender (IE02) (Runtime Disk 1)	Product code[2]
Linux on IBM z Systems	WebSphere Message Broker Version 7.0 Linux on IBM z Systems (Runtime Disk 1)	Product code[2]
	WebSphere MQ Version 7.0.1 Linux on IBM z Systems platform (Runtime Disk 2)	Product code
	WebSphere Message Broker ODBC Database Extender (IE02) (Runtime Disk 1)	Product code[2]
Solaris on SPARC	WebSphere Message Broker Version 7.0 Solaris (Runtime Disk 1)	Product code[2]
	WebSphere MQ Version 7.0.1 Solaris (Runtime Disk 2)	Product code
	WebSphere Message Broker ODBC Database Extender (IE02) (Runtime Disk 1)	Product code[2]
Solaris on x86-64	WebSphere Message Broker Version 7.0 Solaris on x86-64 (Runtime Disk 1)	Product code[2]
	WebSphere MQ Version 7.0.1 Solaris on x86-64 (Runtime Disk 2)	Product code
	WebSphere Message Broker ODBC Database Extender (IE02) (Runtime Disk 1)	Product code[2]

Table 37. CDs and images supplied for WebSphere Message Broker and associated products (continued)

Operating System[1]	CD label	Description
Windows 32-bit	WebSphere Message Broker Version 7.0 Windows 32-bit (Runtime Disk 1)	Product code[3]
	WebSphere MQ Version 7.0.1 Windows 32-bit (Runtime Disk 2)	Product code[4]
Windows 64-bit	WebSphere Message Broker Version 7.0 Windows 64-bit (Runtime Disk 1)	Product code[3]
	WebSphere MQ Version 7.0.1 Windows 64-bit (Runtime Disk 2)	Product code[4]

Notes:

1. On all operating systems, the CDs and images for WebSphere MQ (Runtime Disk 2) are not included in the Trial Edition. You must obtain prerequisite software from other sources to support the Trial Edition of WebSphere Message Broker. See "Additional software requirements" on page 3598 for more information.
2. Disk 1 for all Linux and UNIX systems includes the following resources:
   - WebSphere Message Broker installation files.
   - WebSphere Message Broker Explorer installation file (Linux on x86 and Linux on x86-64 only).
   - WebSphere Message Broker ODBC Database Extender (IE02) (Linux and UNIX only).
   - License files. These files are used by the installation wizard and are supplied in all supported languages.
   - The readme file in English and supported languages. The `readme.html` file contains late updates about the product and its documentation. The latest version of the readme file is available in English only on the product readmes web page. The readme file that is included on the product media contains a link to the latest version on the product readmes web page.
   - The *Installation Guide* PDF file in English and supported languages. The *Installation Guide* PDF file is available after a major release of the information center, such as Version 7.0. If significant changes are accumulated between major releases, the *Installation Guide* PDF file is refreshed and provided in English, on the customer FTP site and the Library web page. The PDF file content is the same as the content that is provided in the information center at the time of publish. However the PDF file documentation is updated less frequently than the information center. For the latest information, see the "Installation" on page 3581 and "Installing" on page 231 sections in the online information center.
   - Sample scripts. Use the sample response files to run the silent interface to install and uninstall components.
3. Disk 1 for Windows includes the following resources:
   - WebSphere Message Broker installation files.
   - WebSphere Message Broker Explorer installation file.
   - The Launchpad.
   - The Quick Tour stand-alone executable program.
   - License files. These files are used by the installation wizard and are supplied in all supported languages.
   - The readme file in English and supported languages. The `readme.html` file contains late updates about the product and its documentation. The latest version of the readme file is available in English only on the product

readmes web page. The readme file that is included on the product media contains a link to the latest version on the product readmes web page.
- The *Installation Guide* PDF file in English and supported languages. The *Installation Guide* PDF file is available within a quarter of a year after a major release of the information center, such as Version 7.0, or when significant changes are accumulated. The PDF file content is the same as the content that is provided in the information center at the time of publish. However the PDF file documentation is updated less frequently than the information center. For the latest information, see the "Installation" on page 3581 and "Installing" on page 231 sections in the online information center.
- Sample scripts. Use the sample response files to run the silent interface to install and uninstall components.

4. The Launchpad and stand-alone Quick Tour are also included on Disk 2 on Windows.

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

"Finding the latest information" on page 232
Access the latest information for WebSphere Message Broker.

Related reference:

"Installation packages" on page 3608
View the installation packages that are available, and the contents of those packages.

Related information:

WebSphere Message Broker Requirements

Toolkit package:

Contents of the Toolkit package.

The Toolkit package is supplied on the product DVDs for installation of the WebSphere Message Broker Toolkit. The structure of the DVD content is described in "DVD package" on page 3616.

The Toolkit package contains three directories, called disk1, disk2, and disk3. The contents of the disks in shown in the following table.

Table 38. DVD and images supplied for WebSphere Message Broker Toolkit

Operating System	Directory	Description
Linux on x86	disk1	• Product code • Installation Manager[1] • Additional resources[2]
	disk2	Product code
	disk3	Product code

Table 38. DVD and images supplied for WebSphere Message Broker Toolkit (continued)

Operating System	Directory	Description
Linux on x86-64	disk1	• Product code • Installation Manager[1] • Additional resources[2]
	disk2	Product code
	disk3	Product code
Windows 32-bit	disk1	• Product code • Installation Manager[1] • Additional resources[2]
	disk2	Product code
	disk3	Product code
Windows 64-bit	disk1	• Product code • Installation Manager[1] • Additional resources[2]
	disk2	Product code
	disk3	Product code

Notes:

1. Installation Manager is installed only if it does not exist on the target computer; this product is required to manage the WebSphere Message Broker Toolkit installation.
2. Disk 1 includes the following additional resources:
   - Readme files. The `readme.html` files contain late updates about the product and its documentation and are supplied in all supported languages.
   - Installation Guides. PDF files of the Installation Guide are supplied in all supported languages to which it has been translated.
   - Sample scripts. Use these sample response files to run the silent interface to install and uninstall components.
   - On Windows only, the Launchpad.
   - On Windows only, the Quick Tour stand-alone executable program.

   These files are identical to the equivalent files described in "The Broker component and WebSphere Message Broker Explorer package" on page 3611.

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

"Finding the latest information" on page 232
Access the latest information for WebSphere Message Broker.

Related reference:

"Installation packages" on page 3608
View the installation packages that are available, and the contents of those packages.

Related information:

WebSphere Message Broker Requirements

DVD package:

Use the DVD package to install the Broker component, the WebSphere Message Broker Toolkit, the WebSphere Message Broker Explorer, WebSphere MQ Version 7.0.1, and WebSphere Message Broker ODBC Database Extender (IE02) (Linux and UNIX only).

DVDs (type DVD-R, size DVD-5) are supplied only for Linux on x86, Linux on x86-64, and Windows. The DVD labels are WebSphere Message Broker Version 7.0 Linux on x86, WebSphere Message Broker Version 7.0 Linux on x86-64, WebSphere Message Broker Version 7.0 Windows 32-bit, and WebSphere Message Broker Version 7.0 Windows 64-bit.

DVD images are not included in the Trial Edition.

The physical package is marked with the symbol 

.

The DVDs contain code for the following products:
- Broker component
- WebSphere Message Broker Toolkit
- WebSphere Message Broker Explorer
- WebSphere MQ Version 7.0.1
- WebSphere Message Broker ODBC Database Extender (IE02) (Linux and UNIX only)

If you use the Launchpad to install on Windows, as described in "Installing by using the Windows Launchpad" on page 262, the Launchpad navigates the DVD to find the products and components that you have chosen to install; therefore, you do not need to be familiar with the structure of the DVD contents.

If you install on Linux on x86, Linux on x86-64, or Windows, and choose not to use the Launchpad, use the DVD structure shown in the following table to find what you need.

Table 39. DVD contents

Directory	Description
\ (root)	• WebSphere Message Broker installation files • (Windows only) Launchpad and Quick Tour[1]
\ie02	WebSphere Message Broker ODBC Database Extender (IE02) (Linux and UNIX only)
\installation_guide	Installation Guides[3, 4]
\license	License files[3]
\Message_Broker_Toolkit_V7.0	WebSphere Message Broker Toolkit installation repository [2]
\MBExplorer	WebSphere Message Broker Explorer installation repository
\readmes	Readme files[3]
\sample-scripts	Sample response files[3]
\WebSphere_MQ_V7.0.1	WebSphere MQ installation images

Notes:

1. This item is a stand-alone executable version of the Quick Tour, which is available only on Windows. On Linux on x86, Linux on x86-64, and Windows, you can access the Quick Tour from the WebSphere Message Broker Toolkit.
2. This directory contains WebSphere Message Broker Toolkit installation repository and Installation Manager. The version of the Installation Manager in this directory is specific to the target operating system.
3. The files in this directory are identical to the equivalent files described in "The Broker component and WebSphere Message Broker Explorer package" on page 3611.
4. On Linux on x86 and Linux on x86-64, this directory has two subdirectories: one for US English and one for translated PDF files.

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

"Finding the latest information" on page 232
Access the latest information for WebSphere Message Broker.

Related reference:

"Installation packages" on page 3608
View the installation packages that are available, and the contents of those packages.

Related information:

WebSphere Message Broker Requirements

Installation and uninstallation interfaces

You can use different interfaces for installation and uninstallation of the Broker component and the WebSphere Message Broker Toolkit.

When you install or uninstall the Broker component, you can choose a graphical, console, or silent interface. These options are described in "How to install and uninstall the Broker component" on page 3618.

When you install or uninstall the WebSphere Message Broker Toolkit, you can choose a graphical or silent interface. These options are described in "How to install and uninstall the WebSphere Message Broker Toolkit" on page 3623.

Examples of commands in the topics in this section use *installer* and *uninstaller* for the names of the installation or uninstallation wizard. Substitute the correct names for the platform on which you are using the wizard. Unless otherwise specified, you can use the examples for uninstalling components, or applying service; the same format is used for all three operations.

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

"Applying service to the Broker component" on page 314
Apply maintenance updates and program fixes to the Broker component.

"Uninstalling the Broker component" on page 332
You can uninstall the Broker component on distributed systems in a number of ways.

"Applying service to the WebSphere Message Broker Toolkit" on page 325
Apply maintenance updates and program fixes to the WebSphere Message Broker Toolkit.

"Uninstalling the WebSphere Message Broker Toolkit" on page 340
Choose the method that you want to use to uninstall the WebSphere Message Broker Toolkit and follow the instructions to remove the component.

Related reference:

"Installation and uninstallation authorization" on page 3628
Check the authorization requirements for the user ID that you use to install or uninstall the Broker component and the WebSphere Message Broker Toolkit.

How to install and uninstall the Broker component

You can install and uninstall the Broker component by using one of three interfaces.

- "Graphical interface"
- "Console interface" on page 3619
- "Silent interface" on page 3619

Each interface has different advantages, which are discussed in the appropriate sections. When you have chosen the interface that you want to use, complete the following steps:

- Check that your user ID has the correct authority to complete this task; see "Installation and uninstallation authorization" on page 3628 for details.
- If you have multiple broker installations on your system, see "How to uninstall multiple installations of the Broker component" on page 3620.
- Follow the installation instructions in "Installing" on page 231 (for a new installation or to install service to existing components), or in "Uninstalling" on page 331 (for product components and service).

Graphical interface

The installation and uninstallation wizards open a graphical interface if you start them with no options (the default interface). The wizards guide you through the installation or uninstallation process with a series of pages that present options and defaults. You can accept the default values, or change them to suit your environment and requirements.

The graphical interface provides the highest level of information and guidance. Use this interface when you are unfamiliar with the product, or to monitor progress.

If you click **Cancel** before the Install Progress or Uninstall Progress panel appears, you can exit the setup. If you choose to exit, your system returns to the state that it was in before the wizard was started. However, if you cancel the installation wizard after installation or uninstallation has completed, and the final summary panel is displayed, your system is not restored to its previous state; the wizard stops immediately. If you want to remove any program that has been installed, invoke the uninstallation wizard.

When you use the wizards, you might have to wait a few seconds to move to the next panel after clicking **Next**. Progress is not always displayed on all panels. If you click **Next** twice, you might skip an entire panel. To ensure that the installer or uninstaller is progressing, monitor your processor usage, which increases greatly during both installation and uninstallation.

Console interface

The console interface is a character-based interface with which you interact in a command window. It presents the same options as the graphical interface.

Use the console interface if you want a command-line or text interface rather than a graphical interface. This interface is suitable for users who use only the keyboard to choose values and navigate through installation, and those users with screen reader software, such as JAWS.

Invoke the installer by using the following command. Use the same format for the uninstaller.

```
installer -console
```

Use these prompts to navigate through the wizard:

- 1 Move to the next panel
- 2 Return to the previous panel
- 3 Cancel and terminate the wizard
- 4 Redisplay the current screen

The default option is always displayed within brackets, for example [1]. If this default option is the correct choice, press Enter to continue.

Silent interface

Use the silent interface for automated installation or uninstallation over many identical systems. If you start a silent installation or uninstallation, the wizard runs without any interaction. Using this interface, the process is completed with default options, or according to a predefined set of options. The silent interface does not provide any feedback to the caller, therefore you must view the log to check whether the action was successful.

If you use the silent interface to uninstall the Broker component, the wizard always uninstalls components from the last known Version 7.0 installation location (that is, the most recent installation), regardless of the location of the uninstallation wizard that you invoke. To remove components from an earlier Version 7.0 installation, use the console or graphical interface.

You can perform a silent installation in the following ways:

- With default options:

  The installation wizard performs the following actions:
  - Checks that prerequisite software is installed
  - Installs to the default directory
  - Installs all selectable features

  Because the installation wizard checks for prerequisite software by using the silent interface, the program fails if the prerequisite software is not already installed. You can override this check if you use a response file (see “Using

response files with the Broker component" on page 3621), or include the appropriate parameter with a non-default value on the command invocation.

The uninstallation wizard performs the following actions:
– Removes all selectable features

To run the wizard with default options, specify the `-silent -G licenseAccepted=true` options on the command:

```
installer -silent -G licenseAccepted=true
```

- With one or more non-default options:

  If you want the wizard to use non-default values for one or more options, specify non-default options either on the command invocation, or in a response file, as described in "Using response files with the Broker component" on page 3621.

  A sample response file is provided in the `sample-scripts` directory of the root CD or DVD directory. This file includes detailed information about the options that you can change, and the values that you must enter to change them. Tailor this file to match your requirements, or generate a new response file.

  To run a tailored silent installation using a response file called `response1.txt`, specify the `-silent` option and the file name on the command:

  ```
  installer -silent -G licenseAccepted=true -options response1.txt
  ```

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

"Applying service to the Broker component" on page 314
Apply maintenance updates and program fixes to the Broker component.

"Uninstalling the Broker component" on page 332
You can uninstall the Broker component on distributed systems in a number of ways.

Related reference:

"Using response files with the Broker component" on page 3621
You can use a response file to define the behavior of an installation or uninstallation wizard that is running the silent interface.

"Installation and uninstallation interfaces" on page 3617
You can use different interfaces for installation and uninstallation of the Broker component and the WebSphere Message Broker Toolkit.

"Installation and uninstallation authorization" on page 3628
Check the authorization requirements for the user ID that you use to install or uninstall the Broker component and the WebSphere Message Broker Toolkit.

How to uninstall multiple installations of the Broker component:

Uninstalling the Broker component when you have multiple installations.

If you install the same version and release of the Broker component on a single computer (for example, Version 7.0) more than once, the installer support provided by the operating system cannot manage these installations in the normal way.

If you later want to uninstall one of the multiple installations, use the uninstallation program *uninstaller* in the `_uninst_runtime` directory of the specific installation that you want to remove, not the facilities provided by the operating system.

You can view the `install.properties` file to see current installations and their locations, and check the operating system representation:

AIX The first installation is recorded as mqsi*vr*, for example `mqsi70`. Subsequent installations at the same *vr* level are displayed under the first one when you list installed products with **`lslpp`**. If you use **`smitty`** and **`geninstall`** to manage those subsequent installations, results are unpredictable.

Linux, HP-UX, and Solaris
The first installation is recorded as mqsi/*vr*, for example `mqsi/70`. Subsequent installations at the same *vr* level are recorded as mqsi/*vr*-2, and so on.

Windows
The most recent installation that you completed for any given version and release is displayed in **Add/Remove Programs**. No other installations are shown here. Similarly, the **Command Console** option in the **Start** menu is that associated with the most recent installation for any given version and release.

If you uninstall the product at a specific version and release listed by **Add/Remove Programs**, earlier installations that you completed on the computer are not reinstated in that view.

To uninstall other instances, navigate to the directory that contains the uninstallation program. For details of uninstallation tasks, see the information center.

Using response files with the Broker component:

You can use a response file to define the behavior of an installation or uninstallation wizard that is running the silent interface.

A response file is a text file that contains options that define the choices that the wizard makes. You can use response files to install or uninstall the Broker component, or to apply service updates by using non-default values.

- "Editing the sample response files"
- "Recording a response file" on page 3622
- "Generating response files" on page 3622
- "Calling response files in commands" on page 3622

Editing the sample response files:
A sample response file is supplied. On Linux and UNIX systems, the file is `/sample-scripts/install.opt`. On Windows, the file is `\sample-scripts\install.opt`. The sample response files include detailed information about the options that you can change, and the values that you can enter to change them. You can tailor the file to match your requirements.

A number sign (#) at the start of a line denotes a comment. Remove the comment character to enable the line.

You must remove the comment character at the start of the following line. If you do not remove this character, your other options are ignored.

```
#    -W setupTypes.selectedSetypTypeId=
```

Some examples of how you can modify the installation response file follow:

- To accept the product license:

  ```
  -G licenseAccepted=true
  ```

- Choose a custom installation (typical is the default option):

  ```
  -W setupTypes.selectedSetupTypeId=custom
  ```

- Install to a non-default directory.

  Find the following line, remove the number signs, and insert your chosen installation directory:

  ```
  ### -P installLocation=new_location
  ```

- Specify whether the program is to check for prerequisite software.

  Add the following line to the file to instruct the installation wizard to ignore the check for WebSphere MQ:

  ```
  # don't check for WebSphere MQ
  -P mqPrerequisite.active=false
  ```

Recording a response file:
Use the following command to record a response file:

```
installer -options-record responsefile
```

where *responsefile* is the full path and name of your chosen response file. On Windows, surround the path and name with quotation marks if it contains spaces ("response file"). Create this file in a directory different from the one in which the product is installed.

The installation wizard starts its graphical interface and records your responses as it progresses. When installation is complete, the response file contains all the choices that you have made during installation.

If you want to record a response file during a console installation, use the following command:

```
installer -options-record responsefile -console
```

Generating response files:
Use the following command to generate a template installation response file. The wizard does not perform installation or uninstallation when you start it with these options:

```
installer -options-template responsefile
```

where *responsefile* is the full path and name of your chosen response file. On Windows, surround the path and name with quotation marks if it contains spaces. If you are uninstalling, create the response file in another directory to ensure that it is not deleted as part of the uninstallation.

The generated template response file contains full instructions on how to edit it to specify your required options.

To generate a response file during a console installation, use the following command:

```
installer -options-template responsefile -console
```

Calling response files in commands:
Use the following command to run the silent interface with a response file:

```
installer -options responsefile -silent
```

where *responsefile* is the full path and name of your chosen response file. On Windows, surround the path and name with quotation marks if it contains spaces.

The wizard runs, taking its input from the response file.

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

"Applying service to the Broker component" on page 314
Apply maintenance updates and program fixes to the Broker component.

"Installing the Broker component in silent mode" on page 272
Install the Broker component by using the installation wizard in silent mode.

"Uninstalling the Broker component" on page 332
You can uninstall the Broker component on distributed systems in a number of ways.

How to install and uninstall the WebSphere Message Broker Toolkit

Install and uninstall the WebSphere Message Broker Toolkit by using one of two interfaces.

- "Graphical interface"
- "Silent interface" on page 3624

Each interface has different advantages, which are explained in the appropriate sections. When you have chosen the interface that you want to use:

- Check that your user ID has the correct authority to complete this task; see "Installation and uninstallation authorization" on page 3628 for details.
- Before installing WebSphere Message Broker Toolkit on Linux, check that the default permissions are set correctly by running the following command under a user ID with root authority:

  ```
  umask
  ```

  A value of *0022* should be returned, indicating the permissions are correctly set. If any other value is returned, set the correct permissions by running the following command:

  ```
  umask 0022
  ```

- Follow the installation instructions in "Installing" on page 231 (for a new installation or to install service to existing components), or in "Uninstalling" on page 331 (for product components and service).

Graphical interface

The installation and uninstallation wizards open a graphical interface if you start them with no options (the default interface). The wizards guide you through the installation or uninstallation process with a series of pages that present options and defaults. You can accept the default values, or change them to suit your environment and requirements.

The graphical interface provides the highest level of information and guidance. Use this interface when you are unfamiliar with the product, or to monitor progress.

If you click **Cancel** before the Install Progress or Uninstall Progress panel appears, you can exit the setup. If you choose to exit, your system returns to its state that it was in before the wizard was started. However, if you cancel the installation wizard after installation or uninstallation has completed, and the final summary panel is displayed, your system is not restored to its previous state: the wizard stops immediately. If you want to remove any program that has been installed, start the uninstallation wizard.

When you use the wizards, you must wait a few seconds to move to the next panel after clicking **Next**. Progress is not always displayed on all panels. If you click **Next** twice, you might skip an entire panel. To ensure that the installer or uninstaller is progressing, you can monitor your processor usage, which increases greatly during both installation and uninstallation.

Silent interface

Use the silent interface for automated installations over many identical systems. If you start a silent installation or uninstallation, the wizard runs without any interaction; the process is completed with default options, or according to a predefined set of options. The silent interface does not provide any feedback to the caller; therefore, you must check the log to determine whether the action was successful.

You can run a silent installation with default settings, or with one or more non-default values:

- With default settings, the installation wizard performs the following actions:
  - Installs to the default directories
  - Installs all supported locales

  To run a default silent installation, change the path to the `/Message_Broker_Toolkit_V7.0/disk1` directory of the local or remote DVD, or the network drive, and run the following command:

  Linux **Linux on x86**

```
installToolkit-silent.sh
```

  Windows **Windows**

```
installToolkit-silent.bat
```

- With one or more non-default settings, the wizard applies the options that you specify in a response file to determine what actions to take.

  If you want the wizard to use non-default values for one or more options, specify a recorded response file, as described in "Using response files with the WebSphere Message Broker Toolkit" on page 3625.

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

"Applying service to the WebSphere Message Broker Toolkit" on page 325
Apply maintenance updates and program fixes to the WebSphere Message Broker Toolkit.

"Uninstalling the WebSphere Message Broker Toolkit" on page 340
Choose the method that you want to use to uninstall the WebSphere Message

Broker Toolkit and follow the instructions to remove the component.

Related reference:

"Using response files with the WebSphere Message Broker Toolkit"
Specify a response file to define the behavior of the installation or uninstallation wizard.

"Installation and uninstallation interfaces" on page 3617
You can use different interfaces for installation and uninstallation of the Broker component and the WebSphere Message Broker Toolkit.

"Installation and uninstallation authorization" on page 3628
Check the authorization requirements for the user ID that you use to install or uninstall the Broker component and the WebSphere Message Broker Toolkit.

Using response files with the WebSphere Message Broker Toolkit:

Specify a response file to define the behavior of the installation or uninstallation wizard.

You can use response files to install or uninstall the WebSphere Message Broker Toolkit, or to apply service updates.

- "Editing the sample response files"
- "Recording a response file"
- "Calling response files in commands"

Editing the sample response files:
A sample response file `mbtoolkit-silent.xml` is supplied on Linux on x86 and Windows. The files assume that IBM Installation Manager has not yet been installed, and sets options to install both Installation Manager and the WebSphere Message Broker Toolkit in the default locations.

Although you can tailor these files to match your requirements, for example by changing the installation locations, the record option on a graphical installation or uninstallation is preferable. If you use the record option, you do not have to modify the file content, which is complex because the files handle multiple installations, directories, and options.

Recording a response file:
Use the following command to record a response file:

- On Linux on x86:

```
./install -record response.xml
```

- On Windows:

```
install.exe -record response.xml
```

where *response.xml* is the full path and name of your chosen response file. On Windows, surround the path and name with quotation marks if it contains spaces. Create this file in a directory different from the one in which the product is installed.

The installation wizard opens its graphical interface, and requires your input as it progresses. Your responses are recorded during installation. When installation is complete, the response file contains all the choices that you have made during installation.

Calling response files in commands:

Use the following command to run the installation wizard with the silent interface and a recorded response file:

- On Linux on x86:

```
./install -nosplash --launcher.suppressErrors -silent -input response.xml
```

- On Windows:

```
install.exe -nosplash --launcher.suppressErrors -silent -input response.xml
```

where *response.xml* is the full path and name of the response file you recorded. On Windows, surround the path and name with quotation marks if it contains spaces.

The wizard runs, taking its input from the response file.

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

"Applying service to the WebSphere Message Broker Toolkit" on page 325
Apply maintenance updates and program fixes to the WebSphere Message Broker Toolkit.

"Uninstalling the WebSphere Message Broker Toolkit" on page 340
Choose the method that you want to use to uninstall the WebSphere Message Broker Toolkit and follow the instructions to remove the component.

Installation wizard names

View the installation wizards that are used to install the Broker component, the WebSphere Message Broker Toolkit, and the WebSphere Message Broker Explorer.

The installation wizard has a different name on each operating system. To help you find these programs quickly, the names are shown in the following tables.

Installation wizard names for the Broker component

The following table shows the installation wizard names that are used to start the installers for the Broker component.

Table 40. Installation wizard names for the Broker component

Operating system	Installation wizard name
AIX	`setupaix`
HP-Itanium	`setuphpia64`
Linux on POWER	`setuplinuxppc`
Linux on x86	`setuplinuxia32`
Linux on x86-64	`setuplinuxx64`
Linux on IBM z Systems	`setuplinux390`
Solaris on SPARC	`setupsolaris`
Solaris on x86-64	`setupsolarisx64`
Windows 32-bit	`setup.exe`
Windows 64-bit	`setup.exe`

Installation wizard names for WebSphere Message Broker Toolkit

The following table shows the installation wizard names that are used to start Installation Manager that controls the installation of the WebSphere Message Broker Toolkit.

Table 41. Installation wizard names for WebSphere Message Broker Toolkit

Operating system	Installation wizard name
Linux on x86	`install`
Linux on x86-64	`install`
Windows 32-bit	`install.exe`[1]
Windows 64-bit	`install.exe`[1]

Note:

1. You can also use `installc.exe` to start Installation Manager. This program operates synchronously and does not return control to the command line until the installation has completed.

Installation wizard names for WebSphere Message Broker Explorer

The following table shows the installation wizard names that are used to start the installers for the WebSphere Message Broker Explorer.

Table 42. Installation wizard names for WebSphere Message Broker Explorer

Operating system	Installation wizard name
Linux on x86	`install`
Linux on x86-64	`install`
Windows 32-bit	`install.exe`
Windows 64-bit	`install.exe`

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

"Installing the Broker component" on page 267
Use the installation wizard to install the Broker component.

"Installing the WebSphere Message Broker Toolkit" on page 276
Use the installation wizard graphical interface to install the WebSphere Message Broker Toolkit on Windows and Linux on x86.

"Installing WebSphere Message Broker Explorer" on page 280
To use WebSphere Message Broker Explorer only, without installing the complete WebSphere Message Broker Toolkit, use the WebSphere Message Broker Explorer installation wizard to install the WebSphere Message Broker Explorer.

Related reference:

"Installation and uninstallation interfaces" on page 3617
You can use different interfaces for installation and uninstallation of the Broker component and the WebSphere Message Broker Toolkit.

Installation and uninstallation authorization

Check the authorization requirements for the user ID that you use to install or uninstall the Broker component and the WebSphere Message Broker Toolkit.

Check that your user ID has the following authority to complete installation or uninstallation tasks:

- On AIX systems, you must log in as root.
- On Linux and other UNIX systems, your user ID must have root authority to complete installation. Follow your local security guidelines to acquire root authority; either log in as root, or log in as another user and become root.
- On all Windows operating systems and editions, your user ID must be a member of the Administrators group.
- On z/OS systems, your user ID must be no more than eight characters in length. It must also have suitable privileges to install in your environment with SMP/E. Use a supported external security manager, for example RACF or ACF2, to grant the required privileges. Your user ID must have a valid OMVS segment, because the product installs into the file system paths specified during the SMP/E APPLY processing.

Related tasks:

"Installing" on page 231
Installation information for WebSphere Message Broker is provided in the WebSphere Message Broker Installation Guide, which is available in the WebSphere Message Broker Version 7.0 information center, on the WebSphere Message Broker Version 7.0 Library web page, and in the WebSphere Message Broker Version 7.0 Quick Start Guide.

"Applying service to the Broker component" on page 314
Apply maintenance updates and program fixes to the Broker component.

"Uninstalling the Broker component" on page 332
You can uninstall the Broker component on distributed systems in a number of ways.

"Applying service to the WebSphere Message Broker Toolkit" on page 325
Apply maintenance updates and program fixes to the WebSphere Message Broker Toolkit.

"Uninstalling the WebSphere Message Broker Toolkit" on page 340
Choose the method that you want to use to uninstall the WebSphere Message Broker Toolkit and follow the instructions to remove the component.

"Setting up z/OS security" on page 556
On z/OS, you must complete several security configuration tasks before WebSphere Message Broker can work correctly.

Related reference:

"Installation and uninstallation interfaces" on page 3617
You can use different interfaces for installation and uninstallation of the Broker component and the WebSphere Message Broker Toolkit.

Multicultural support

Multicultural support is available for a selection of languages on both distributed systems and z/OS.

The user interface and message catalogs are provided in the following languages on distributed systems:

- Brazilian Portuguese
- French

- German
- Italian
- Japanese
- Korean
- Simplified Chinese
- Spanish
- Traditional Chinese
- US English

The message catalogs are provided in the following languages on z/OS:
- Japanese
- Simplified Chinese
- US English

The messages written to the z/OS operator console (which are a subset of the messages written to the syslog) are in US English only, and are written in mixed case or in uppercase depending on your chosen system configuration.

WebSphere Message Broker provides a selection of message catalogs that are used by the product components to report any problems that occur. Products that are used in conjunction with WebSphere Message Broker might cause WebSphere Message Broker to report errors using its message catalogs, or might report problems using their own techniques.

You must refer to the documentation supplied with any other products that you use to determine the process they employ. In particular, you must check the documentation supplied by the databases that you use and documentation provided with any user-defined node or parser that you integrate into the WebSphere Message Broker environment.

You can install WebSphere Message Broker and WebSphere MQ in any supported language; all language versions for each product are compatible with all language versions for the other product. All languages for the WebSphere MQ messaging products are included on the WebSphere MQ server CD or DVD supplied with WebSphere Message Broker.

All messages generated for internal inter-component message exchange (for example, deployed configuration messages and log files for **mqsireadlog**) are generated in code page 1208 (utf-8).

Related tasks:

"Converting code page and message encoding" on page 2476
You can use ESQL within a Compute node to convert data for code page and message encoding.

"Changing locales" on page 819
You can change the locale for the system on which a runtime component is installed.

Locales

Message support is provided in a number of locales.

WebSphere Message Broker supports at least the following locales:

Windows	AIX	Solaris	HP-UX[1]	Linux[2]	z/OS
English (United States)	en_US	en_US	en_US.iso88591, en_US.roman8	en_US	En_US.IBM-1047, En_US.IBM-037

Windows	AIX	Solaris	HP-UX[1]	Linux[2]	z/OS
German (Standard)	de_DE, De_DE	de	de_DE.ISO88591, de_DE.roman8	de_DE	not supported
Spanish (Modern Sort)	es_ES, Es_ES	es	es_ES.ISO88591, es_ES.roman8	es_ES	not supported
French (Standard)	fr_FR, Fr_FR	fr	fr_FR.ISO88591, fr_FR.roman8	fr_FR	not supported
Italian (Standard)	it_IT, It_IT	it	it_IT.ISO88591, it_IT.roman8	it_IT	not supported
Portuguese (Brazilian)	pt_BR, Pt_BR	pt_BR	pt_BR.ISO88591, pt_BR.utf8	pt_BR	not supported
Japanese	Ja_JP, ja_JP	ja_JP.PCK, ja	ja_JP.SJIS, ja_JP.eucJP	ja_JP	Ja_JP.IBM-939, Ja_JP.IBM-930
Simplified Chinese (China)	Zh_CN, zh_CN	zh, zh.GBK	zh_CN.hp15CN	zh_CN	Zh_CN.IBM-1388, Zh_CN.IBM-935
Traditional Chinese (Taiwan)	Zh_TW, zh_TW	zh_TW, zh_TW.BIG5	zh_TW.big5, zh_TW.eucTW	zh_TW	not supported
Korean	ko_KR	ko	ko_KR.eucKR	ko_KR	not supported

Notes:

1. Because of limited syslog support on HP-Itanium operating systems, messages are written to the log in US English only.
2. These values are the same for all Linux systems.

Other locales might be supported; check your operating system for further details.

Related tasks:

"Changing locales" on page 819
You can change the locale for the system on which a runtime component is installed.

Related reference:

"Supported processors" on page 3583
WebSphere Message Broker is supported on multiple processors.

"Operating system requirements" on page 3590
WebSphere Message Broker is supported on multiple operating systems.

"Supported code pages" on page 4176
Application messages must conform to supported code pages.

System changes caused by installation and configuration

When you install WebSphere Message Broker, the installation program makes certain changes to your computer.

Resources are also affected by how you configure your system (for example, the brokers that you create).

- Start and main menu updates
- Directory structures
- Registry contents
- Environment variables
- Default WebSphere MQ resources

Related tasks:

"Finding the latest information" on page 232
Access the latest information for WebSphere Message Broker.

Related reference:

"Software requirements" on page 3588
View the operating system, database, and other software requirements.

"Hardware requirements" on page 3582
View the processor and related hardware requirements on all platforms.

"Supported processors" on page 3583
WebSphere Message Broker is supported on multiple processors.

"Operating system requirements" on page 3590
WebSphere Message Broker is supported on multiple operating systems.

"Supported databases" on page 3591
You can optionally configure databases to contain data that is accessed by your message flows. Databases from IBM and other suppliers are supported at specific versions on supported operating systems.

Start and main menu updates after installation

When you install WebSphere Message Broker components on Linux on x86 or Windows, the installation program updates the main or Start menus.

WebSphere Message Broker Toolkit on Linux on x86 running Red Hat

The main menu is populated with the following entries:

- **Programming** > **IBM WebSphere Message Broker Toolkit**
- **Programming** > **IBM Installation Manager**

If you install multiple products with Installation Manager on this operating system, a single entry exists in the main menu and refers to the last product that you installed. The package groups of installed products are not reflected in the main menu. To invoke applications that you installed previously, including other installations of the WebSphere Message Broker Toolkit, you must use the command-line interface.

If you want to start Installation Manager, for example to update or uninstall one or more packages, you cannot use the main menu items, because these actions require root authority. Become root and start the Installation Manager program `/eclipse/IBMIM` in the Installation Manager installation directory on the command line.

WebSphere Message Broker Toolkit on Linux on x86 running SUSE

The main menu is populated with the following entries:

- **All Applications** > **Development** > **IBM WebSphere Message Broker Toolkit**
- **All Applications** > **Development** > **IBM Installation Manager**

If you install multiple products with Installation Manager on this operating system, a single entry exists in the main menu and refers to the last product that you installed. The package groups of installed products are not reflected in the main menu. To invoke applications that you installed previously, including other installations of the WebSphere Message Broker Toolkit, you must use the command-line interface.

If you want to start Installation Manager, for example to update or uninstall one or more packages, you cannot use the main menu items, because these actions require root authority. Become root and start the Installation Manager program `/eclipse/IBMIM` in the Installation Manager installation directory on the command line.

WebSphere Message Broker Toolkit on Windows

The **Start** menu is populated with the following entries:

- **IBM Software Development Platform** > **IBM WebSphere Message Broker Toolkit** > **WebSphere Message Broker Toolkit**
- **IBM Software Development Platform** > **IBM WebSphere Message Broker Toolkit** > **Release Notes**
- **IBM Software Development Platform** > **IBM WebSphere Message Broker Toolkit** > **Start Toolkit Help**
- **IBM Software Development Platform** > **IBM WebSphere Message Broker Toolkit** > **Stop Toolkit Help**

These entries assume that you have installed into the first or default package group. Additional products that you install in this package group might also appear in this start list.

Entries are also added to the **Start** menu when Installation Manager is first installed:

- **Programs** > **IBM Installation Manager** > **IBM Installation Manager**
- **Programs** > **IBM Installation Manager** > **Release Notes**
- **Programs** > **IBM Installation Manager** > **Uninstall IBM Installation Manager**
- **Programs** > **IBM Installation Manager** > **View Installed Packages**

WebSphere Message Broker Explorer on Linux on x86

No additions are made to the main menu on either SUSE or Red Hat Linux systems.

WebSphere Message Broker Explorer on Windows

The **Start** menu is populated with the following entry:

- **Programs** > **IBM WebSphere Message Broker Explorer**

Broker component on Windows

The **Start** menu is populated with the following entries:

- **Start** > **Programs** > **IBM WebSphere Message Broker 7.0** > **Command Console**
- **Start** > **Programs** > **IBM WebSphere Message Broker 7.0** > **Java Programming APIs** > **CMP API Documentation**
- **Start** > **Programs** > **IBM WebSphere Message Broker 7.0** > **Java Programming APIs** > **CMP API Exerciser**
- **Start** > **Programs** > **IBM WebSphere Message Broker 7.0** > **Java Programming APIs** > **Java Plugin API Documentation**

Related tasks:

"Finding the latest information" on page 232
Access the latest information for WebSphere Message Broker.

Related reference:

"Software requirements" on page 3588
View the operating system, database, and other software requirements.

"Hardware requirements" on page 3582
View the processor and related hardware requirements on all platforms.

"Supported processors" on page 3583
WebSphere Message Broker is supported on multiple processors.

"Operating system requirements" on page 3590
WebSphere Message Broker is supported on multiple operating systems.

"Supported databases" on page 3591
You can optionally configure databases to contain data that is accessed by your message flows. Databases from IBM and other suppliers are supported at specific versions on supported operating systems.

Directory structures after installation

When you install WebSphere Message Broker components, the installation program creates a structure of subdirectories under the directory that you specified as the installation directory. The exact structure depends on the platform on which you have installed WebSphere Message Broker, and the components that you have installed. If you install the WebSphere Message Broker Toolkit, other directories might be created for IBM Installation Manager, the Shared Resources Directory, and the package group in which you install the toolkit.

Platforms covered are:

- "AIX, Solaris on SPARC, and Linux on x86-64 for the 64-bit version of WebSphere Message Broker"
- "HP-Itanium, Linux on POWER, Linux on IBM z Systems, and Solaris on x86-64" on page 3634
- "Linux on x86, and Linux on x86-64 for the 32-bit version of WebSphere Message Broker" on page 3636
- "Windows" on page 3638
- "z/OS" on page 3641

AIX, Solaris on SPARC, and Linux on x86-64 for the 64-bit version of WebSphere Message Broker

On AIX, and Solaris on SPARC, you can install only the broker component.

On Linux on x86-64 for the 64-bit version of WebSphere Message Broker, in addition to the 64-bit broker component, you have the option of installing the 32-bit version of the WebSphere Message Broker Toolkit, and the WebSphere Message Broker Explorer. For more information, see the relevant Toolkit and Explorer sections under "Linux on x86, and Linux on x86-64 for the 32-bit version of WebSphere Message Broker" on page 3636.

The default home directory *install_dir* is `/opt/IBM/mqsi/v.r` or `/opt/ibm/mqsi/v.r`. The default directory includes the version and release of the product, in the format v.r (version.release).

The directories created in *install_dir* are shown in the following table.

Directories under *install_dir*	**Contents**
`_uninst_runtime`	Uninstall programs
`_uninst_runtime_jvm`	Uninstall programs (JVM)
`adapters`	WebSphere Adapters files
`bin`	Executable binary files
`catalina`	Web Services support files
`classes`	Java class files
`DD64`	Symbolic link to the ODBC driver files
`docs`	Java API files

Directories under *install_dir*	Contents
exmltConfig	XML transformation files
include	Header and other files for samples
itlm	Tivoli License Manager files
jplugin	Java plug-in files
jre16	IBM Runtime Environment for the Java Platform Version 6 (also known as Version 1.6)
lib	Shared library files
license	License files
lil	Loadable library files
messages	Description files for messages and exceptions
migration	Migration files
ODBC	ODBC driver and related files
ODBC64	Symbolic link to the ODBC driver files
readmes	Product readme files
sample	C, C++, and Java sample files
SecurityProviders	Security files
webservices	Web Services files
xlxpc	Product library files
xml4c	XML processing files
isadc	IBM Support Assistant Data Collector files

On AIX, Linux on x86-64, and Solaris on SPARC, the working directory is /var/mqsi/. The directories created are shown in the following table.

Directories under /var/mqsi	Contents
common/errors	Error files
common/locks	Resource locks
common/log	Trace files
common/profiles	User profiles
components	Component details
config	User configuration of profiles and shared classes
ODBC64	Symbolic link to the ODBC driver files
registry	Registry information
shared-classes	User installed user-defined node classes
XML	XML files
XSL	XSL style sheets

HP-Itanium, Linux on POWER, Linux on IBM z Systems, and Solaris on x86-64

On HP-Itanium, Linux on POWER, Linux on IBM z Systems, and Solaris on x86-64, you can install only the broker component.

The default home directory *install_dir* is `/opt/IBM/mqsi/v.r` or `/opt/ibm/mqsi/v.r`. The default directory includes the version and release of the product, in the format v.r (version.release).

The directories created in *install_dir* are shown in the following table.

Directories under *install_dir*	**Contents**
`_uninst_runtime`	Uninstall programs
`_uninst_runtime_jvm`	Uninstall programs (JVM)
`adapters`	WebSphere Adapters files
`bin`	Executable binary files
`catalina`	Web Services support files
`classes`	Java class files
`DD64`	Symbolic link to the ODBC driver files
`docs`	Java API files
`exmltConfig`	XML transformation files
`include`	Header and other files for samples
`itlm`	Tivoli License Manager files
`jplugin`	Java plug-in files
`jre16`	IBM Runtime Environment for the Java Platform Version 6 (also known as Version 1.6)
`lib`	Shared library files
`license`	License files
`lil`	Loadable library files
`messages`	Description files for messages and exceptions
`migration`	Migration files
`ODBC`	ODBC driver and related files
`ODBC64`	Symbolic link to the ODBC driver files
`readmes`	Product readme files
`sample`	C, C++, and Java sample files
`SecurityProviders`	Security files
`webservices`	Web Services files
`xlxpc`	Product library files
`xml4c`	XML processing files
`isadc`	IBM Support Assistant Data Collector files

On HP-Itanium, Linux on POWER, Linux on IBM z Systems, and Solaris on x86-64, the default working directory is `/var/mqsi/`. The directories created are shown in the following table.

Directories under /var/mqsi	**Contents**
`common/errors`	Error files
`common/locks`	Resource locks
`common/log`	Trace files
`common/profiles`	User profiles

Directories under /var/mqsi	Contents
components	Component details
config	User configuration of profiles and shared classes
ODBC64	Symbolic link to the ODBC driver files
registry	Registry information
shared-classes	User installed user-defined node classes
XML	XML files
XSL	XSL style sheets

Linux on x86, and Linux on x86-64 for the 32-bit version of WebSphere Message Broker

On this platform, you can install the broker, the WebSphere Message Broker Toolkit, and the WebSphere Message Broker Explorer.

Broker

The default home directory *install_dir* is /opt/ibm/mqsi/*v.r*. The default directory includes the version and release of the product, in the format v.r (version.release).

The directories created in *install_dir* are shown in the following table.

Directories under *install_dir*	Contents
_uninst_runtime	Uninstall programs
_uninst_runtime_jvm	Uninstall programs (JVM)
adapters	WebSphere Adapters files
bin	Executable binary files
catalina	Web Services support files
classes	Java class files
docs	Java API files
exmltConfig	XML transformation files
include	Header and other files for samples
itlm	Tivoli License Manager files
jplugin	Java plug-in files
jre16	IBM Runtime Environment for the Java Platform Version 6 (also known as Version 1.6)
lib	Shared library files
license	License files
lil	Loadable library files
merant	Symbolic link to the ODBC driver files
messages	Description files for messages and exceptions
migration	Migration files
ODBC	ODBC driver and related files
ODBC32	Symbolic link to the ODBC driver files
readmes	Product readme files

Directories under *install_dir*	Contents
sample	C, C++, and Java sample files
SecurityProviders	Security files
webservices	Web Services files
xlxpc	Product library files
xml4c	XML processing files
isadc	IBM Support Assistant Data Collector files

The default working directory is /var/mqsi/. The directories created are shown in the following table.

Directories under /var/mqsi	Contents
common/errors	Error files
common/locks	Resource locks
common/log	Trace files
common/profiles	User profiles
components	Component details
config	User configuration of profiles and shared classes
ODBC64	Symbolic link to the ODBC driver files
registry	Registry information
shared-classes	User installed user-defined node classes
XML	XML files
XSL	XSL style sheets

WebSphere Message Broker Toolkit

The default IBM Installation Manager installation directory is /opt/ibm/InstallationManager. The default Shared Resources Directory is /opt/IBM/SDPShared/. The default package group directory is /opt/IBM/WMBT700. These directories are defined and described in the "Installing the WebSphere Message Broker Toolkit" on page 276.

The directories created are shown in the following tables.

Directories under /opt/ibm/InstallationManager	Contents
eclipse	Eclipse plug-in directories and files
license	License files

Directories under /opt/IBM/SDPShared/	Contents
atoc	Eclipse plug-in directories and files
extra	License files
features	Product files for features installed on this computer
native	

Directories under /opt/IBM/SDPShared/	Contents
plugins	Eclipse plug-in directories and files common to features installed on this computer

Directories under /opt/IBM/WMBT700	Contents
bin	Directories and files for products within the package group
configuration	Configuration files
eclipse	Eclipse files
features	Directories and files for products within the package group
jdk	Java directories and files
lum	License files
plugins	Eclipse plug-ins and related files
ResourceAdapters	Rational directories and files
runtimes	Rational directories and files
samples	Sample response files
uninstall	Product uninstallation files
wmbt	Toolkit files

Other files are also stored by the Installation Manager in the directory /var/ibm/InstallationManager.

WebSphere Message Broker Explorer

Windows

On this platform, you can install the broker, the WebSphere Message Broker Toolkit, and the WebSphere Message Broker Explorer.

Broker

The default home directory *install_dir* is:

- On Windows 32-bit editions: C:\Program Files\IBM\MQSI*v.r*.
- On Windows 64-bit editions:
  - C:\Program Files\IBM\MQSI*v.r* for the 64-bit version of WebSphere Message Broker.
  - C:\Program Files (x86)\IBM\MQSI*v.r* for the 32-bit version of WebSphere Message Broker.

The default directory includes the version and release of the product, in the format v.r (version.release).

The directories created in *install_dir* are shown in the following table.

Directories under *install_dir*	Contents
_uninst_runtime	uninstallation programs
_uninst_runtime_jvm	uninstallation programs (JVM)

Directories under *install_dir*	**Contents**
`adapters`	WebSphere Adapters files
`bin`	Executable binary files
`catalina`	Web Services support files
`classes`	Java class files
`DataDirect`	ODBC driver and related files
`docs`	Java API files
`exmltConfig`	XML transformation files
`include`	Header and other files for samples
`itlm`	Tivoli License Manager files
`jplugin`	Java plug-in files
`jre16`	IBM Runtime Environment for the Java Platform Version 6 (also known as Version 1.6)
`lib`	Shared library files
`license`	License files
`messages`	Description files for messages and exceptions
`migration`	Migration files
`readmes`	Product readme files
`sample`	C, C++, and Java sample files
`webservices`	Web Services files
`WMQFTE`	Files for WebSphere MQ File Transfer Edition.
`isadc`	IBM Support Assistant Data Collector files

The default working directory is `%ALLUSERSPROFILE%\Application Data\IBM\MQSI` where `%ALLUSERSPROFILE%` is the environment variable that defines the system working directory. The default directory depends on the operating system:

- On Windows XP and Windows Server 2003: `C:\Documents and Settings\All Users\Application Data\IBM\MQSI`
- On Windows Vista and later operating systems: `C:\ProgramData\IBM\MQSI`

The actual value might be different on your computer.

The directories created are shown in the following table.

Directories under *work_dir*	**Contents**
`Common/errors`	Error files
`Common/locks`	Resource locks
`Common/log`	Trace files
`Common/profiles`	User profiles
`components`	Component details
`config`	User configuration of profiles and shared classes
`odbc`	ODBC files
`registry`	Registry information
`shared-classes`	User installed user-defined node classes
`XML`	XML files

Directories under *work_dir*	Contents
XSL	XSL style sheets

WebSphere Message Broker Toolkit

On Windows 32-bit, the default IBM Installation Manager installation directory is C:\Program Files\IBM\InstallationManager, the default Shared Resources Directory is C:\Program Files\IBM\SDPShared\, and the default package group directory is C:\Program Files\IBM\WMBT700.

On Windows 64-bit, the default IBM Installation Manager installation directory is C:\Program Files(x86)\IBM\InstallationManager, the default Shared Resources Directory is C:\Program Files (x86)\IBM\SDPShared*v.r*, and the default package group directory is C:\Program Files (x86)\IBM\WMBT700.

These directories are defined and described in the "Installing the WebSphere Message Broker Toolkit" on page 276.

The directories created are shown in the following tables.

Directories under \IBM\InstallationManager	Contents
eclipse	Eclipse plug-in directories and files
license	License files

Directories under \SDPShared	Contents
atoc	Eclipse plug-in directories and files
extra	License files
features	Product files for features installed on this computer
native	
plugins	Eclipse plug-in directories and files common to features installed on this computer

Directories under \WMBT700	Contents
bin	Eclipse plug-in directories and files
configuration	License files
features	Product files for features installed on this computer
jdk	Java directories and files
lum	License files
plugins	Eclipse plug-in directories and files common to features installed on this computer
ResourceAdapters	Rational directories and files
runtimes	Rational directories and files
samples	Sample response files
uninstall	Product uninstall files
wmbt	Toolkit files

Other files are also stored by the Installation Manager in the following directories:

- On Windows XP and Windows Server 2003: `C:\Documents and Settings\All Users\Application Data\IBM\Installation Manager` and `C:\Documents and Settings\Administrator\IBM\InstallationManagerInstaller`. Your computer might have a different value for `C:\Documents and Settings`, but the remainder of these paths are fixed.
- On Windows Vista and later operating systems: `C:\ProgramData\IBM\Installation Manager` and `C:\ProgramData\IBM\InstallationManagerInstaller`.

WebSphere Message Broker Explorer

z/OS

The default installation directory is `/usr/lpp/mqsi/`*`VxRxMx`* where *VxRxMx* represents Version X, Release X, Modification X, for example, `V7R0M0`.

For more details about locations, libraries, and file system paths, see the *Program Directory for WebSphere Message Broker for z/OS* on the WebSphere Message Broker Library web page.

Related reference:

"Hardware requirements" on page 3582
View the processor and related hardware requirements on all platforms.

"Software requirements" on page 3588
View the operating system, database, and other software requirements.

Registry changes created by installation and configuration

When you install WebSphere Message Broker, the installation program creates a number of entries in a registry. Further changes are made by some configuration updates (for example, when you create a broker).

Runtime components

On Windows, the ALLUSERPROFILE environment variable is used; on Linux and UNIX systems, equivalent values are stored within the installation directory structure. On all platforms, the environment variable MQSI_REGISTRY is set to point to the exact location. Do not alter or remove these entries unless instructed to do so by your IBM Service representative.

On Windows, some entries are also added to the system registry under `HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBCINST.INI` (for 32-bit brokers on 32-bit operating system editions and 64-bit brokers on 64-bit operating system editions) or under `HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\ODBC\ODBCINST.INI` (for 32-bit brokers on 64–bit operating system editions); these entries record the installed 32–bit database drivers for Oracle and Sybase, which contain driver locations and parameters.

An entry is also added under the `HKEY_LOCAL_MACHINE\SOFTWARE\IBM\WebSphereMQIntegrator` (for 32-bit brokers on 32-bit operating system editions and 64-bit brokers on 64-bit operating system editions) or under `HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\IBM\WebSphereMQIntegrator` (for 32-bit brokers on 64–bit operating system editions).

A final entry is added under `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Eventlog\Application\WebSphere Broker v***` for Event Log information, where `v***` is the current three-digit product version number.

WebSphere Message Broker Toolkit

All registry entries are controlled by IBM Installation Manager. On Windows, the system registry is used; on Linux on x86, equivalent entries are stored in `/var/ibm/InstallationManager/`. Do not alter or remove these entries unless instructed to do so by your IBM Service representative.

The installation directory for Installation Manager is stored in `HKEY_LOCAL_MACHINE\SOFTWARE\IBM\Installation Manager` (for 32–bit operating system editions) or `HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\IBM\Installation Manager` (for 64–bit operating system editions), under the location value.

Uninstall keys are created for all products that are installed by Installation Manager under `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\Current\Version\Uninstall\IBM Installation Manager`, under the name `IM-`*`packagegroupname`* where *packagegroupname* is the name of the package group in which the product is installed; for example, IBM Software Development Platform.

WebSphere Message Broker Explorer

Related reference:

"Supported processors" on page 3583
WebSphere Message Broker is supported on multiple processors.

"Operating system requirements" on page 3590
WebSphere Message Broker is supported on multiple operating systems.

"Directory structures after installation" on page 3633
When you install WebSphere Message Broker components, the installation program creates a structure of subdirectories under the directory that you specified as the installation directory. The exact structure depends on the platform on which you have installed WebSphere Message Broker, and the components that you have installed. If you install the WebSphere Message Broker Toolkit, other directories might be created for IBM Installation Manager, the Shared Resources Directory, and the package group in which you install the toolkit.

Environment variables after installation

On distributed systems, ensure that your environment is set up correctly.

If you have installed on z/OS, see "Customizing the z/OS environment" on page 591.

Sample profile files are provided with WebSphere Message Broker; check their content to ensure the variables are set correctly for your environment before you use the product or configure any resources.

- On Linux and UNIX systems, the profile is *`install_dir`*`/bin/mqsiprofile`.
- On Windows systems, a command console is set up when you install components. Access this through the **Start** menu. When you select this option, a command window containing the correct environment is opened.

  If you prefer, run *`install_dir`*`\bin\mqsiprofile.cmd` in a command window before working with WebSphere Message Broker in the same command window.

If you have more than one installation on a single system, ensure that you run the correct profile, or access the correct command console, for the installation that you want to work with. If an installation is at Version 6.0, the profile is incompatible with the Version 7.0 profile. You must log off and log on again before you run a second profile.

Check the readme file (`readme.html`) to ensure that you have the latest version of the profile.

Related tasks:

Chapter 7, "Configuring brokers for test and production environments," on page 579
Create one or more brokers on one or more computers, and configure them on your test and production systems to process messages that contain your business data.

"Setting up a command environment" on page 213
After you have installed the product on one of the distributed systems, you must initialize the environment before you can use a runtime component or command.

"Are the Linux and UNIX environment variables set correctly?" on page 3350
Use the **mqsiprofile** command to set a command environment.

Default WebSphere MQ resources created during installation and configuration

When you install WebSphere Message Broker and create components, WebSphere MQ resources are created for use by those components.

The names of these resources begin with the reserved characters SYSTEM. The resources are additional to the default WebSphere MQ objects that are created when you install that product. The following table lists the resources created.

Resource name	Type	Description
SYSTEM.BROKER.ADAPTER.FAILED	Queue	Failure messages generated by adapters.
SYSTEM.BROKER.ADAPTER.INPROGRESS	Queue	In progress messages generated by adapters.
SYSTEM.BROKER.ADAPTER.NEW	Queue	New request messages generated by adapters.
SYSTEM.BROKER.ADAPTER.PROCESSED	Queue	Processed messages generated by adapters.
SYSTEM.BROKER.ADAPTER.UNKNOWN	Queue	Unknown messages generated by adapters.
SYSTEM.BROKER.ADMIN.QUEUE	Queue	Target for request messages sent by CMP API applications (including commands) to modify the configuration and operation of the broker.
SYSTEM.BROKER.ADMIN.REPLYTODM	Queue	Target for internal messages sent by the broker.
SYSTEM.BROKER.AGGR.CONTROL	Queue	Used internally to store a control message for each aggregate group.
SYSTEM.BROKER.AGGR.REPLY	Queue	Used internally to store each known response message received by an AggregateReply node.
SYSTEM.BROKER.AGGR.REQUEST	Queue	Used internally to store each request message that forms part of an aggregate group.
SYSTEM.BROKER.AGGR.TIMEOUT	Queue	Used internally in the timeout and unknown timeout processing within an AggregateReply node.
SYSTEM.BROKER.AGGR.UNKNOWN	Queue	Stores each unknown response received by an AggregateReply node.
SYSTEM.BROKER.AUTH	Queue	Stores authorization records for administration requests against the broker.
SYSTEM.BROKER.DEPLOY.QUEUE	Queue	Target for publish/subscribe control requests that applications send to the broker.
SYSTEM.BROKER.DEPLOY.REPLY	Queue	Target for response messages sent by publish/subscribe control requests that applications send to the broker.

Resource name	Type	Description
SYSTEM.BROKER.EDA.COLLECTIONS	Queue	Used internally to store event handler information for message collections generated by the Collector node.
SYSTEM.BROKER.EDA.EVENTS	Queue	Used internally to store messages received by the Collector node.
SYSTEM.BROKER.EXECUTIONGROUP.QUEUE	Queue	Target for messages sent to execution groups by internal broker processes.
SYSTEM.BROKER.EXECUTIONGROUP.REPLY	Queue	Target for response messages sent by execution groups to internal broker processes.
SYSTEM.BROKER.INTER.BROKER.COMMUNICATIONS	Queue	Target for publish/subscribe messages sent by neighbor brokers.
SYSTEM.BROKER.MODEL.QUEUE	Queue	Model for dynamic response queues.
SYSTEM.BROKER.TIMEOUT.QUEUE	Queue	Used internally to support the TimeoutControl and TimeoutNotification nodes.
SYSTEM.BROKER.WS.ACK	Queue	Used internally for Web Services client support.
SYSTEM.BROKER.WS.INPUT	Queue	Used internally for Web Services client support.
SYSTEM.BROKER.WS.REPLY	Queue	Used internally for Web Services client support.
SYSTEM.BKR.CONFIG	SVRCONN	A connection channel for WebSphere Message Broker Toolkit and WebSphere Message Broker Explorer clients.

Related reference:

"Designing the WebSphere MQ infrastructure" on page 584
You must create and manage the WebSphere MQ resources that are required to support your brokers, and the applications that connect to them to supply or receive messages.

"Software requirements" on page 3588
View the operating system, database, and other software requirements.

Security requirements for administrative tasks

You can configure access permissions to govern which users and groups can manipulate objects in the broker network. Security requirements for administrative tasks depend on the platform that you use.

This section contains the following topics:

Related concepts:

Chapter 5, "Security," on page 351
Security is an important consideration for both developers of WebSphere Message Broker applications, and for system administrators configuring WebSphere Message Broker authorities.

"Security overview" on page 351
When you are designing a WebSphere Message Broker application it is important to consider the security measures that are needed to protect the information in the

system.

Related tasks:

"Planning for security when you install WebSphere Message Broker" on page 353
The Installation Guide describes the security tasks that you must complete before, during, and after installation.

Tasks and authorizations for broker administration security

If you have enabled broker administration security, users require specific authority so that they can complete administration tasks.

The following table shows the list of actions that a user can perform, and the authorizations that you must set to allow them to complete these tasks when broker administrative security is enabled. The authority is required regardless of the way in which the user requests the action; from a CMP API application, the WebSphere Message Broker Explorer, or the WebSphere Message Broker Toolkit.

In addition to the permissions for the specific tasks that are shown in the following table, you must also be able to connect to the broker. For more information, see "Authorizing users for broker administration" on page 371.:

Tasks	Authorization	Queue
Set broker properties	Read and write	SYSTEM.BROKER.AUTH
View broker properties	Read	SYSTEM.BROKER.AUTH
Create or delete configurable services	Read and write	SYSTEM.BROKER.AUTH
Set configurable services properties	Read and write	SYSTEM.BROKER.AUTH
View configurable services properties	Read	SYSTEM.BROKER.AUTH
Create or delete execution groups	Read and write	SYSTEM.BROKER.AUTH
Rename execution groups	Read and write	SYSTEM.BROKER.AUTH
List execution groups	Read	SYSTEM.BROKER.AUTH
Start or stop execution groups	Read	SYSTEM.BROKER.AUTH
	Execute	SYSTEM.BROKER.AUTH or SYSTEM.BROKER.AUTH.*E*
Set execution group properties	Read	SYSTEM.BROKER.AUTH
	Write	SYSTEM.BROKER.AUTH.*EG*
View execution group properties	Read	SYSTEM.BROKER.AUTH
	Read	SYSTEM.BROKER.AUTH.*EG*
Start or stop resource statistics collection	Read	SYSTEM.BROKER.AUTH
	Execute	SYSTEM.BROKER.AUTH.*EG*[1]
Report resource statistics	Read	SYSTEM.BROKER.AUTH
	Read	SYSTEM.BROKER.AUTH.*EG*[2]
Deploy	Read	SYSTEM.BROKER.AUTH
	Write	SYSTEM.BROKER.AUTH.*EG*
List message flows and other deployed objects	Read	SYSTEM.BROKER.AUTH
	Read	SYSTEM.BROKER.AUTH.*EG*

Tasks	Authorization	Queue
Start or stop message flows	Read	SYSTEM.BROKER.AUTH
	Execute	SYSTEM.BROKER.AUTH.*EG*
Delete resources from an execution group	Read	SYSTEM.BROKER.AUTH
	Write	SYSTEM.BROKER.AUTH.*EG*

Notes:

1. If you are changing resource statistics collection for all execution groups on the broker, you must grant execute authority for all execution groups.
2. If you are reporting resource statistics collection for all execution groups on the broker, you must grant read authority for all execution groups.
3. In the queue name SYSTEM.BROKER.AUTH.EG, the *EG* refers to the name of your execution group.
4. In the queue name SYSTEM.BROKER.AUTH.*EG*, the *EG* refers to the value of the *egForView* property that you specify in your DataCaptureStore configurable service.
5. In the queue name SYSTEM.BROKER.AUTH.*EG*, the *EG* refers to the value of the *egForReplay* property that you specify in your DataDestination configurable service.

If you grant a user ID authority at the broker level (on queue SYSTEM.BROKER.AUTH), it does not inherit authority for execution groups. You must explicitly grant authority to all, or to individual, execution groups.

Related tasks:

"Setting up broker administration security" on page 368
Control the actions that users can request against a broker and its resources.

"Enabling broker administration security" on page 368
Enable broker administration security on a broker to control which users can complete specific tasks against that broker and its resources.

"Authorizing users for broker administration" on page 371
Grant authority to one or more groups or users to authorize them to complete specific tasks against a broker and its resources.

Commands and authorizations for broker administration security

If you have enabled broker administration security, users require specific authority to be able to run the administration commands.

The following table shows the list of commands, and the authorizations you must set up before users can to run them.

Command	Authorization	Queue
mqsichangeresourcestats	Read	SYSTEM.BROKER.AUTH
	Execute	SYSTEM.BROKER.AUTH.*EG*[1]
mqsicreateexecutiongroup	Read and write	SYSTEM.BROKER.AUTH
mqsideleteexecutiongroup	Read and write	SYSTEM.BROKER.AUTH
mqsideploy	Read	SYSTEM.BROKER.AUTH
	Write	SYSTEM.BROKER.AUTH.*EG*

Command	Authorization	Queue
mqsilist	Read	SYSTEM.BROKER.AUTH
	Read	SYSTEM.BROKER.AUTH.**[2]
mqsimode	Read (to display) or read and write (to change)	SYSTEM.BROKER.AUTH
mqsireloadsecurity	Read	SYSTEM.BROKER.AUTH
	Write	SYSTEM.BROKER.AUTH.**[3]
mqsireportresourcestats	Read	SYSTEM.BROKER.AUTH
	Read	SYSTEM.BROKER.AUTH.*EG*[4]
mqsistartmsgflow[5]	Read	SYSTEM.BROKER.AUTH
	Execute	SYSTEM.BROKER.AUTH.*EG*
mqsistopmsgflow[5]	Read	SYSTEM.BROKER.AUTH
	Execute	SYSTEM.BROKER.AUTH.*EG*

Notes:

1. If you are changing resource statistics collection for all execution groups on the broker, you must have execute authority for all execution groups.
2. You must have read authority for every broker and every execution group for which you are requesting information. If you request details about a resource for which you do not have authority, one or more of the following messages are returned to identify each resource with inappropriate authority:

   ```
   BIP1185S: You cannot view execution group '<egname>' on broker '<brokername>'.

   BIP1014S: You cannot view broker '<brokername>'.
   ```

   The command completes the request and returns results for all the resources for which authority is correct.
3. Where SYSTEM.BROKER.AUTH.** is specified, the user ID running the command must have authority for all execution groups. You can set up this level of authority by either creating a generic profile for all execution groups, or a specific profile for every execution group.
4. If you are reporting resource statistics collection for all execution groups on the broker, you must have read authority for all execution groups.
5. Exact requirements for this command depend on the combination of parameters that you specify on the command; for details, see the authorization section in "**mqsistartmsgflow** command" on page 3969 and "**mqsistopmsgflow** command" on page 3975.
6. In the queue name SYSTEM.BROKER.AUTH.EG, the *EG* refers to the name of your execution group.

Only the commands that are listed in this table are subject to broker administration security.

Note: The authorizations that are listed in this table are in addition to the authorizations required to run the command on specific platforms. Refer to the following topics for information about platform-specific authorizations:

- "Security requirements for Linux and UNIX platforms" on page 3648
- "Security requirements for Windows systems" on page 3651
- "Security requirements for z/OS" on page 3655

Related tasks:

"Setting up broker administration security" on page 368
Control the actions that users can request against a broker and its resources.

"Enabling broker administration security" on page 368
Enable broker administration security on a broker to control which users can complete specific tasks against that broker and its resources.

"Authorizing users for broker administration" on page 371
Grant authority to one or more groups or users to authorize them to complete specific tasks against a broker and its resources.

Related reference:

"Commands" on page 3672
All WebSphere Message Broker Toolkit and runtime commands that are provided on distributed systems are listed, grouped by function, with references to command details.

Security requirements for Linux and UNIX platforms

View a summary of the authorizations in a Linux or UNIX environment.

You must add the required user IDs to the appropriate group to enable them to complete the relevant tasks.

Note: If you have enabled broker administration security, you must also set up the authority detailed in "Tasks and authorizations for broker administration security" on page 3645.

Task	Command	Authorization
Create, delete or migrate a broker	**mqsicreatebroker** **mqsideletebroker** **mqsimigratecomponents**	• Member of mqbrkrs and mqm. • Using LDAP: Ensure that the registry is appropriately secured to prevent unauthorized access. The setting of **LdapPrincipal** and **LdapCredentials** parameters on **mqsichangebroker** is not required for correct operation of the broker. The password is not stored in clear text in the file system.
Change a broker	**mqsichangebroker**	• Member of mqbrkrs. • If you specify the **-s** parameter to activate broker administration security, the user ID used to run this command must be a member of the mqm group, because several queues are created for use by the broker. • Using LDAP: Ensure that the registry is appropriately secured to prevent unauthorized access. The setting of **LdapPrincipal** and **LdapCredentials** parameters on **mqsichangebroker** is not required for correct operation of the broker. The password is not stored in clear text in the file system.

Task	Command	Authorization
Add or remove a broker instance	**mqsiaddbrokerinstance** **mqsiremovebrokerinstance**	• Member of mqbrkrs and mqm. Additionally, you need to make the `uid` and `gid` for this user ID the same on all the systems, and the user ID needs to be the same one that created the first instance of the multi-instance broker, using the **mqsicreatebroker** command. • Change the `uid` and `gid` with caution, as it affects the permission levels of files on the system. Changing a `uid` or `gid` causes the ownership of all the files previously owned by that user or group to change to the integer of the previous owner of the file. Therefore, you must ensure that your system administrator manually restores the ownerships of the affected files and directories.
Backup or restore a broker	**mqsibackupbroker** **mqsirestorebroker**	• Member of mqbrkrs.
Start a broker, or verify a broker	**mqsistart** **mqsicvp**	• Member of mqbrkrs. • Member of mqm if the queue manager is not already running.
Stop a broker	**mqsistop**	• Member of mqbrkrs. However, the root user ID can stop a broker without membership of mqbrkrs. • The user ID must be the same as the user ID that started the broker. • Member of mqm if `-q` is specified.
Create or delete an execution group	**mqsicreateexecutiongroup** **mqsideleteexecutiongroup**	Member of mqbrkrs. • If broker administration security is active, the user ID that runs this command must be a member of the group mqm. If you do not want your broker to run with mqm authority, you must work with your WebSphere MQ administrator to create or delete the appropriate authority queue when you create or delete an execution group.
Start or stop a message flow	**mqsistartmsgflow** **mqsistopmsgflow**	• Member of mqbrkrs.
Create or delete a configurable service	**mqsicreateconfigurableservice** **mqsideleteconfigurableservice**	• Member of mqbrkrs.
List brokers	**mqsilist**	• Member of mqbrkrs.

Task	Command	Authorization
Show broker properties	**mqsireportbroker** **mqsireportproperties** **mqsireportflowmonitoring** **mqsireportflowstats** **mqsireportflowuserexits** **mqsireportresourcestats**	• Member of mqbrkrs.
Change properties	**mqsichangeproperties** **mqsichangeflowmonitoring** **mqsichangeflowstats** **mqsichangeflowuserexits** **mqsichangeresourcestats**	• Member of mqbrkrs.
Set and update passwords	**mqsisetdbparms**	• Member of mqbrkrs.
Report or update a broker mode	**mqsimode**	• Member of mqbrkrs.
Deploy an object to a broker	**mqsideploy**	• Member of mqbrkrs.
Reload a broker, execution groups or security	**mqsireload** **mqsireloadsecurity**	• Member of mqbrkrs.
Trace a broker	**mqsichangetrace** **mqsireporttrace** **mqsireadlog** **mqsiformatlog**	• Member of mqbrkrs.
Set up symbolic links needed for coordinated transactions	**mqsimanagexalinks**	• Root user.
Add the mqbrkrs group	**mqsisetsecurity**	• Root user.

User is...[1]	Command Used	Local domain (WORKSTATION)
Running a broker (WebSphere MQ non-trusted application) (login ID).	• Not applicable	• Member of mqbrkrs. • The broker runs under the login ID that started it.
Running a broker (WebSphere MQ trusted application) (login ID).	• Not applicable	• Login ID must be mqm. • mqm must be a member of mqbrkrs.

Ensure that mqbrkrs has access to all user-defined queues that you have defined for use by your message flows. You can use the **setmqaut** command to set permissions.

- Set the following permissions on all input queues:

```
setmqaut -m MB7BROKER -n TEST_INPUT -t queue -g mqbrkrs  +get +inq
```

- Set the following permissions on all output queues:
  `setmqaut -m MB7BROKER -n TEST_OUTPUT -t queue -g mqbrkrs +put +inq +setall`
- You might also need to add **+passid +passall +setid +setall**, depending on your requirements.

Related tasks:

"Broker component security" on page 497
You must consider several security aspects when you are setting up brokers running on Windows, Linux, or UNIX platforms.

Security requirements for Windows systems

Security requirements depend on the administrative task that you want to perform.

The following tables summarize the requirements for administrative tasks. They show what group membership is required if you are using a local security domain defined on your local system.

Note: If you have enabled broker administration security, you must also set up the authority detailed in "Tasks and authorizations for broker administration security" on page 3645.

Domain users in a multi-workstation domain, or from domains that are in a Windows transitive trust relationship with the local domain, can also perform these administrative tasks. They need to fulfill the group membership requirements specified in the tables. One way to set up this group membership is by adding the domain user to a domain group which in turn is a member of the local group. For an example of how to set up security by using domain groups, see "Security in a Windows domain environment" on page 249.

Task	Command	Authorization
Create, delete or migrate a broker	**mqsicreatebroker** **mqsideletebroker** **mqsimigratecomponents**	• Must be a user ID defined in WORKSTATION. • Member of Administrators. • On Windows 7, Windows Vista, and Windows Server 2008 systems, the user ID used to run this command must be run from a command prompt with elevated privileges. For more information, see "**mqsicommandconsole** command" on page 3830. • Using LDAP: Ensure that the registry is appropriately secured to prevent unauthorized access. The setting of **LdapPrincipal** and **LdapCredentials** parameters on **mqsichangebroker** is not required for correct operation of the broker. The password is not stored in clear text in the file system.

Task	Command	Authorization
Change a broker	**mqsichangebroker**	• Must be a user ID defined in WORKSTATION. • Member of mqbrkrs. • If you specify the **-s** parameter to activate broker administration security, the user ID used to run this command must be a member of the mqm group, because several queues are created for use by the broker. • Using LDAP: Ensure that the registry is appropriately secured to prevent unauthorized access. The setting of **LdapPrincipal** and **LdapCredentials** parameters on **mqsichangebroker** is not required for correct operation of the broker. The password is not stored in clear text in the file system.
Add or remove a broker instance	**mqsiaddbrokerinstance** **mqsiremovebrokerinstance**	• Must be a user ID defined in WORKSTATION. Member of Administrators. • On Windows 7, Windows Vista, and Windows Server 2008 systems, the user ID used to run this command must be run from a command prompt with elevated privileges. For more information, see "**mqsicommandconsole** command" on page 3830.
Backup or restore a broker	**mqsibackupbroker** **mqsirestorebroker**	• Member of mqbrkrs.
Start a broker, or verify a broker	**mqsistart** **mqsicvp**	• Member of mqbrkrs. • Member of mqm if the queue manager is not already running.
Stop a broker	**mqsistop**	• Member of mqbrkrs. • Member of mqm if **-q** is specified.
Create or delete an execution group	**mqsicreateexecutiongroup** **mqsideleteexecutiongroup**	Member of mqbrkrs. • If broker administration security is active, the user ID that runs this command must be a member of the group mqm. If you do not want your broker to run with mqm authority, you must work with your WebSphere MQ administrator to create or delete the appropriate authority queue when you create or delete an execution group.
Start or stop a message flow	**mqsistartmsgflow** **mqsistopmsgflow**	• Member of mqbrkrs.
Create or delete a configurable service	**mqsicreateconfigurableservice** **mqsideleteconfigurableservice**	Member of mqbrkrs.

Task	Command	Authorization
List brokers	**mqsilist**	• Must be a user ID defined in WORKSTATION. • Member of mqbrkrs or mqm to run the command with broker and execution group specified: **mqsilist** *broker_name execution_group_name*
Show broker properties	**mqsireportbroker** **mqsireportproperties** **mqsireportflowmonitoring** **mqsireportflowstats** **mqsireportflowuserexits** **mqsireportresourcestats**	• Member of mqbrkrs.
Change properties	**mqsichangeproperties** **mqsichangeflowmonitoring** **mqsichangeflowstats** **mqsichangeflowuserexits** **mqsichangeresourcestats**	• Member of mqbrkrs.
Set and update passwords	**mqsisetdbparms**	• Member of mqbrkrs.
Report or update a broker mode	**mqsimode**	• Member of mqbrkrs.
Deploy an object to a broker	**mqsideploy**	• Member of mqbrkrs.
Reload a broker, execution groups or security	**mqsireload** **mqsireloadsecurity**	• Member of mqbrkrs.
Trace a broker	**mqsichangetrace** **mqsireporttrace** **mqsireadlog** **mqsiformatlog**	• Must be a user ID defined in WORKSTATION. • Member of mqbrkrs.
Add the mqbrkrs group	**mqsisetsecurity**	• Must be a user ID defined in WORKSTATION. • Member of Administrators. • On Windows 7, Windows Vista, and Windows Server 2008 systems, the user ID used to run this command must be run from a command prompt with elevated privileges. For more information, see “**mqsicommandconsole** command” on page 3830.

Task	Command	Authorization
Run commands that require elevated privileges	**mqsicommandconsole**	• Member of Administrators. • On Windows 7 and Windows Server 2008 systems, the user ID used to run this command must be run from a command prompt with elevated privileges. For more information, see "**mqsicommandconsole** command" on page 3830.

User is...[1]	Command Used	Local domain (WORKSTATION)
Running a broker (WebSphere MQ fast path off) (service user ID)[2]	• Not applicable	• Must be a user ID defined in WORKSTATION • Member of mqbrkrs • Must have the *Logon as a service* privilege in the Windows Local Security Policy. This privilege is added when **mqsicreatebroker** is run, if necessary.
Running a broker (WebSphere MQ fast path on) (service user ID)[2]	• Not applicable	• Must be a user ID defined in WORKSTATION • Member of mqbrkrs • Member of mqm • Must have the *Logon as a service* privilege in the Windows Local Security Policy. This privilege is added when **mqsicreatebroker** is run, if necessary.
Running a WebSphere Message Broker Toolkit[3]	• Start WebSphere Message Broker Toolkit from the **Start** menu	• Must be a user ID defined in WORKSTATION. For example, WORKSTATION\User1 is valid, PRIMARY\User2 and TRUSTED\User3 are not.

Notes:

1. By default when a broker is created, the service user ID is given the required permissions to access to relevant directories of the product directory tree; for example, write access to the logs directory.

   This happens even if you set a location that is not the default, with the *–w* flag on the **mqsicreatebroker** command, or use the *–e* flag on the **mqsicreatebroker** command to create a multi-instance broker. If these permissions are changed manually, you must always ensure that the mqbrkrs group has appropriate access to these locations.
2. Ensure that mqbrkrs has access to all user-defined queues that you have defined for use by your message flows. You can use the **setmqaut** command to set permissions.
   - Set the following permissions on all input queues:

     ```
     setmqaut -m MB7BROKER -n TEST_INPUT -t queue -g mqbrkrs  +get +inq
     ```
   - Set the following permissions on all output queues:

     ```
     setmqaut -m MB7BROKER -n TEST_OUTPUT -t queue -g mqbrkrs +put +inq +setall
     ```
   - You might also need to add **+passid +passall +setid +setall**, depending on your requirements.

3. All WebSphere Message Broker Toolkit users need read access to the WebSphere MQ Java `\lib` subdirectory of the WebSphere MQ home directory (the default location is `X:\Program Files\WebSphere MQ`, where X: is the operating system disk). This access is restricted to users in the local group mqm by WebSphere MQ. WebSphere Message Broker installation overrides this restriction and gives read access for this subdirectory to all users.

Broker security requirements on Windows

On all Windows platforms, there is no longer any requirement for the service user ID to be a member of the Administrators group.

The only requirement is that the service user ID is a member of the mqbrkrs group. In addition, the `LocalSystem` account can be used as the service user ID by specifying `LocalSystem` for the *–i* parameter on the **mqsicreatebroker** command.

In this case you must enter the *–a* (password) parameter on the command line, but the value entered is ignored.

Related tasks:

"Broker component security" on page 497
You must consider several security aspects when you are setting up brokers running on Windows, Linux, or UNIX platforms.

Security requirements for z/OS

View a summary of the authorizations in a z/OS environment.

The following table summarizes the UNIX System Services file access authorizations in a z/OS environment.

Note: If you have enabled broker administration security, you must also set up the authority detailed in "Tasks and authorizations for broker administration security" on page 3645.

Task	Command	Authorization
Create, delete or migrate a broker	**mqsicreatebroker** **mqsideletebroker** **mqsimigratecomponents**	• *READ* and *WRITE* access to the component directory by the z/OS user ID running the command. • The broker runs under its z/OS assigned started task user ID.
Change a broker	**mqsichangebroker**	• *READ* and *WRITE* access to the component directory by the z/OS user ID running the command. • The broker runs under its z/OS assigned started task user ID.
Backup or restore a broker	**mqsibackupbroker** **mqsirestorebroker**	• *READ* and *WRITE* access to the component directory by the z/OS assigned started task user ID.
Start or stop a broker	Console commands	• *READ* and *WRITE* access to the component directory by the z/OS assigned started task user ID. • **UPDATE** authority in class **OPERCMDS** to the `MVS.START.STC.message_broker_component_started_ta` resource.

Task	Command	Authorization
Create or delete an execution group	**mqsicreateexecutiongroup** **mqsideleteexecutiongroup**	*READ* and *WRITE* access to the component directory by the z/OS assigned started task user ID.
Start or stop a message flow	**mqsistartmsgflow** **mqsistopmsgflow**	• *READ* and *WRITE* access to the component directory by the z/OS assigned started task user ID.
Create or delete a configurable service	**mqsicreateconfigurableservice** **mqsideleteconfigurableservice**	• *READ* and *WRITE* access to the component directory by the z/OS assigned started task user ID.
List brokers	**mqsilist**	• *READ* and *WRITE* access to the component directory by the z/OS assigned started task user ID.
Show broker properties	**mqsireportbroker** **mqsireportproperties** **mqsireportflowmonitoring** **mqsireportflowstats** **mqsireportflowuserexits** **mqsireportresourcestats**	• *READ* and *WRITE* access to the component directory by the z/OS assigned started task user ID.
Change properties	**mqsichangeproperties** **mqsichangeflowmonitoring** **mqsichangeflowstats** **mqsichangeflowuserexits** **mqsichangeresourcestats**	• *READ* and *WRITE* access to the component directory by the z/OS assigned started task user ID.
Set and update passwords	**mqsisetdbparms**	• *READ* and *WRITE* access to the component directory by the z/OS assigned started task user ID.
Deploy an object to a broker	**mqsideploy**	• *READ* and *WRITE* access to the component directory by the z/OS assigned started task user ID.
Reload a broker, execution groups or security	**mqsireload** **mqsireloadsecurity**	• *READ* and *WRITE* access to the component directory by the z/OS assigned started task user ID.
Trace a broker	**mqsichangetrace** **mqsireporttrace** **mqsireadlog** **mqsiformatlog**	• *READ* and *WRITE* access to the component directory by the z/OS assigned started task user ID.
Report or update a broker mode	**mqsimode**	• *READ* and *WRITE* access to the component directory by the z/OS assigned started task user ID.

Related tasks:

"Setting up z/OS security" on page 556
On z/OS, you must complete several security configuration tasks before WebSphere Message Broker can work correctly.

"Broker component security" on page 497
You must consider several security aspects when you are setting up brokers running on Windows, Linux, or UNIX platforms.

Related reference:

"Summary of required access (z/OS)" on page 3985
The professionals in your organization require access to components and resources on z/OS.

"Customization tasks and roles on z/OS" on page 3984
System programmers do most of the customization of WebSphere Message Broker.

Configuration and administration

Use the reference information in this section to accomplish the configuration and administration tasks that address your business needs.

- "Restrictions that apply in each operation mode"
- "Database configuration" on page 3659
- "Administration API" on page 3672
- "Commands" on page 3672

Related tasks:

Chapter 6, "Configuring brokers for development environments," on page 563
Set up application development environments on Linux on x86 or Windows to create, test, and deploy message flows and associated resources.

Chapter 7, "Configuring brokers for test and production environments," on page 579
Create one or more brokers on one or more computers, and configure them on your test and production systems to process messages that contain your business data.

Chapter 8, "Administering brokers and broker resources," on page 899
Administering brokers and associated broker resources includes the tasks that you perform frequently to activate and manage those resources. Choose the method you prefer to administer your brokers and associated resources.

"Developing applications that use the Administration API" on page 956
Develop Java applications that use the Administration API (also known as the CMP API) to communicate with, deploy to, and manage brokers and their associated resources.

Restrictions that apply in each operation mode

The operation mode in which your broker is working defines how many execution groups you can use, and which nodes are available.

Trial Edition mode

All features are enabled, but you can use the product for only 90 days after installation.

Starter Edition mode

All features are enabled. You can create one execution group, but the number of message flows that you can deploy is unlimited.

Not all samples work in Starter Edition mode because the sample import and deploy wizards create one or more execution groups, and the number of execution groups is limited in this mode.

You can run a sample in Starter Edition mode if the default execution group is deleted after you have run the Default Configuration wizard, but before you install the sample. For example, to run the Pager sample:

1. Run the Default Configuration wizard.
2. Delete the default execution group on the MB7BROKER that is created by the wizard.
3. To import and deploy the Pager sample, click Set up the Pager samples.
4. Follow the documented steps to run the Pager samples.

If you want to run samples to explore and understand the features of the product, you can install them on your development and unit test computers; see "Development and unit test" on page 50.

Entry Edition mode

In `entry` mode, the broker operates with a limited set of nodes available for deployed flows. You are also limited to one execution group. If you attempt to exceed the limits of this mode, the deployment is rejected. Use this edition for simple use cases only.

The functions that are enabled and the number of execution groups that you can create are limited. Therefore, not all samples work in Entry Edition mode.

The following message flow nodes can be used in `entry` mode. Plug-in nodes are also supported in `entry` mode.

• Check node • Database node • EmailOutput node • FileInput node • FileOutput node • FlowOrder node • HTTPHeader node • HTTPInput node • HTTPReply node • HTTPRequest node • Input node • JavaCompute node • JMSHeader node • JMSInput node • JMSMQTransform node • JMSOutput node • JMSReply node	• Label node • Mapping node • MQGet node • MQHeader node • MQInput node • MQJMSTransform node • MQOutput node • MQReply node • Output node • Passthrough node • PHPCompute node • Publication node • ResetContentDescriptor node • RouteToLabel node • SOAPAsyncRequest node • SOAPAsyncResponse node • SOAPEnvelope node	• SOAPExtract node • SOAPInput node • SOAPReply node • SOAPRequest node • TCPIPClientInput node • TCPIPClientOutput node • TCPIPClientReceive node • TCPIPServerInput node • TCPIPServerOutput node • TCPIPServerReceive node • Throw node • Trace node • TryCatch node • Validate node • XSLTransform node

Remote Adapter Deployment mode

Only adapter-related features are enabled, and the types of node that you can use, and the number of execution groups that you can create, are limited. You can create up to two execution groups, with no limit on the number of deployed message flows in each of these execution groups.

Not all samples work in this mode. If you want to run samples to explore and understand the features of the product, you can install them on your development and unit test computers; see "Development and unit test" on page 50.

The following message flow nodes can be used in `adapter` mode. Plug-in nodes are also supported in `adapter` mode.

• CDInput node • CDOutput node • CICSRequest node • CORBARequest node • DatabaseInput node • EmailInput node • EndpointLookup node • FileInput node • FileOutput node • FTEInput node • FTEOutput node • HTTPInput node • HTTPReply node • HTTPRequest node • IMSRequest node • Input node • JavaCompute node • JDEdwardsInput node • JDEdwardsRequest node	• JMSInput node • JMSOutput node • MQGet node • MQInput node • MQOutput node • Output node • PeopleSoftInput node • PeopleSoftRequest node • PHPCompute node • Real-timeInput node • Real-timeOptimizedFlow node • RegistryLookup node • SAPInput node • SAPReply node • SAPRequest node • SCAAsyncRequest node • SCAAsyncResponse node	• SCAInput node • SCAReply node • SCARequest node • SecurityPEP node • SiebelInput node • SiebelRequest node • SOAPInput node • SOAPReply node • SOAPRequest node • TCPIPClientInput node • TCPIPClientOutput node • TCPIPClientReceive node • TCPIPServerInput node • TCPIPServerOutput node • TCPIPServerReceive node • TimeoutControl node • TimeoutNotification node

Enterprise mode

All features are enabled and no restrictions or limits are imposed.

Related concepts:

"Operation modes" on page 48
The operation mode that you use for your broker is determined by the license that you purchase.

"Samples" on page 98
The WebSphere Message Broker Toolkit provides samples that show the features that are available in WebSphere Message Broker, and how to use them. This topic provides links to the information about the individual samples.

Related tasks:

"Checking the operation mode of your broker" on page 657
Use the **mqsimode** command to find out the operation mode of your broker.

"Changing the operation mode of your broker" on page 655
Change the operation mode in which your broker is working by using the **mqsimode** command.

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

Related reference:

"What the Default Configuration wizard creates" on page 107
A table of the components that are created by the wizard, details of how to resolve problems, and how to view errors.

Database configuration

Sample ODBC definition files are supplied to help you to define an ODBC connection to a database from your brokers or applications.

Using the DataDirect drivers

For a copy of the sample ODBC definition file that is supplied with WebSphere Message Broker, for use with the DataDirect drivers, see the following topic:

- "Sample DataDirect odbc.ini file"

Using the WebSphere Message Broker Database Extender (IE02) drivers

For a copy of the sample ODBC definition file that is supplied with WebSphere Message Broker for use with the WebSphere Message Broker Database Extender (IE02) drivers, see the following topic:

- "Sample WebSphere Message Broker ODBC Database Extender (IE02) configuration files" on page 3596

Related concepts:

"Databases overview" on page 2109
Brokers can access *user databases* to manipulate your business data. If you have user databases, you must configure them before you can access them from your message flow.

Related tasks:

"Working with databases" on page 2109
Create and configure databases to use with your message flow applications.

"Enabling ODBC connections to the databases" on page 668
Set up the resources and environment that the broker requires for Open Database Connectivity (ODBC) connections to user databases on distributed systems.

Sample DataDirect odbc.ini file

A copy of the sample DataDirect ODBC definition file that is supplied with WebSphere Message Broker.

Purpose

Configure the `odbc.ini` file when you define an ODBC connection to a DB2, Informix, Oracle, Sybase, or SQLServer database. Follow the instructions in "Connecting to a database from Linux and UNIX systems using the DataDirect drivers" on page 674.

Some lines in this topic have been split to improve readability; you must not split these lines within your copy of the file.

AIX, Linux on x86-64, Solaris on SPARC, and HP-Itanium

```
;#########################################################################
;# ODBC database driver manager initialisation file.                     #
;#########################################################################
;# It is recommended that you take a copy of this file and then edit the  #
;# copy.                                                                 #
;#                                                                       #
;# 1. Complete the 'Mandatory information stanza' section                #
;# at the end of the file.                                               #
;#                                                                       #
;# 2. For each data source, add the name of the data source into         #
;# the 'List of data sources stanza' section.                            #
;#                                                                       #
;# 3. For each data source, create a stanza in the                       #
;# 'Individual data source stanzas' section.                             #
;#########################################################################
```

```
;########################################
;###### List of data sources stanza #######
;########################################
[ODBC Data Sources]
DB2DB=IBM DB2 ODBC Driver
ORACLEDB=DataDirect 6.0 Oracle Wire Protocol
ORACLERACDB=DataDirect 6.0 Oracle Wire Protocol (Real Application Clusters)
SYBASEDB=DataDirect 6.0 Sybase Wire Protocol
SYBASEDBUTF8=DataDirect 6.0 Sybase UTF8 Wire Protocol
SQLSERVERDB=DataDirect 6.0 SQL Server Wire Protocol
INFORMIXDB=IBM Informix ODBC Driver

;########################################################
;########## Individual data source stanzas ###############
;########################################################

;# DB2 stanza
[DB2DB]
DRIVER=libdb2Wrapper.so
Description=DB2DB DB2 ODBC Database
Database=DB2DB

;# Oracle stanza
[ORACLEDB]
Driver=<Your Broker install directory>/ODBC/V6.0/lib/UKora24.so
Description=DataDirect 6.0 Oracle Wire Protocol
HostName=<Your Oracle Server Machine Name>
PortNumber=<Port on which Oracle is listening on HostName>
SID=<Your Oracle SID>
CatalogOptions=0
EnableStaticCursorsForLongData=0
ApplicationUsingThreads=1
EnableDescribeParam=1
OptimizePrepare=1
WorkArounds=536870912
ProcedureRetResults=1
ColumnSizeAsCharacter=1
LoginTimeout=0

;# Oracle Real Application Clusters stanza
[ORACLERACDB]
Driver=<Your Broker install directory>/ODBC/V6.0/lib/UKora24.so
Description=DataDirect 6.0 Oracle Wire Protocol
HostName=<Your Oracle Server Machine Name>
PortNumber=<Port on which Oracle is listening on HostName>
ServiceName=<Your Oracle Real Application Cluster Service Name>
AlternateServers=(HostName=<Your alternative host name>:PortNumber=<Port on
which Oracle is listening on the alternative host>:ServiceName=<Your
Oracle Real Application Cluster Service Name>)
CatalogOptions=0
EnableStaticCursorsForLongData=0
ApplicationUsingThreads=1
EnableDescribeParam=1
OptimizePrepare=1
WorkArounds=536870912
ProcedureRetResults=1
ColumnSizeAsCharacter=1
LoginTimeout=0

;# Sybase Stanza
[SYBASEDB]
Driver=<Your Broker install directory>/ODBC/V6.0/lib/UKase24.so
Description=DataDirect 6.0 Sybase Wire Protocol
Database=<Your Database Name>
ApplicationUsingThreads=1
EnableDescribeParam=1
OptimizePrepare=1
SelectMethod=0
NetworkAddress=<Your Sybase Server Name>,<Your Sybase Port Number>
SelectUserName=1
```

```
ColumnSizeAsCharacter=1
EnableSPColumnTypes=2
LoginTimeout=0
TimestampTruncationBehavior=1
XAConnOptBehaviour=3

;# Sybase Stanza for a UTF8 datasource
[SYBASEDBUTF8]
Driver=<Your Broker install directory>/ODBC/V6.0/lib/UKase24.so
Description=DataDirect 6.0 Sybase Wire Protocol
Database=<Your Database Name>
ApplicationUsingThreads=1
EnableDescribeParam=1
OptimizePrepare=1
SelectMethod=0
NetworkAddress=<Your Sybase Server Name>,<Your Sybase Port Number>
SelectUserName=1
ColumnSizeAsCharacter=1
EnableSPColumnTypes=2
Charset=UTF8
LoginTimeout=0
TimestampTruncationBehavior=1
XAConnOptBehaviour=3

;# UNIX to SQLServer stanza
[SQLSERVERDB]
Driver=<Your Broker install directory>/ODBC/V6.0/lib/UKmsss24.so
Description=DataDirect 6.0 SQL Server Wire Protocol
Address=<Your SQLServer Machine Name>,<Your SQLServer Port Number>
;# Alternative way to locate server using a named instance
;# Address=<Your SQLServer Machine Name>\<Your SQLServer Instance Name>
AnsiNPW=Yes
Database=db
QuotedId=No
ColumnSizeAsCharacter=1

;# Informix Stanza
[INFORMIXDB]
Driver=libinfWrapper.so
Description=IBM Informix ODBC Driver
ServerName=<Your Informix Server Name>
Database=<Your Database Name>

;########################################
;###### Mandatory information stanza ######
;########################################

[ODBC]
;# To turn on ODBC trace set Trace=1
Trace=0
TraceOptions=3
TraceFile=<A Directory with plenty of free space to hold trace
output>/odbctrace.out
TraceDll=<Your Broker install directory>/ODBC/V6.0/lib/odbctrac.so
InstallDir=<Your Broker install directory>/ODBC/V6.0
UseCursorLib=0
IANAAppCodePage=4
UNICODE=UTF-8
```

Linux on x86

```
;#########################################################################
;# ODBC database driver manager initialisation file.                     #
;#########################################################################
;# It is recommended that you take a copy of this file and then edit the  #
;# copy.                                                                 #
;#                                                                       #
;# 1. Complete the 'Mandatory information stanza' section                #
;# at the end of the file.                                               #
;#                                                                       #
```

```
;# 2. For each data source, add the name of the data source into          #
;# the 'List of data sources stanza' section.                             #
;#                                                                        #
;# 3. For each data source, create a stanza in the                        #
;# 'Individual data source stanzas' section.                              #
;##########################################################################

;########################################
;###### List of data sources stanza #######
;########################################
[ODBC Data Sources]
DB2DB=IBM DB2 ODBC Driver
ORACLEDB=DataDirect 6.0 Oracle Wire Protocol
ORACLERACDB=DataDirect 6.0 Oracle Wire Protocol (Real Application Clusters)
SYBASEDB=DataDirect 6.0 Sybase Wire Protocol
SYBASEDBUTF8=DataDirect 6.0 Sybase UTF8 Wire Protocol
SQLSERVERDB=DataDirect 6.0 SQL Server Wire Protocol
INFORMIXDB=IBM Informix ODBC Driver

;########################################################
;######### Individual data source stanzas ###############
;########################################################

;# DB2 stanza
[DB2DB]
Driver=<Your DB2 install directory>/lib32/libdb2.so
Description=DB2 ODBC Database
Database=DB2DB

;# Oracle stanza
[ORACLEDB]
Driver=<Your Broker install directory>/ODBC/V6.0/lib/UKora24.so
Description=DataDirect 6.0 Oracle Wire Protocol
HostName=<Your Oracle Server Machine Name>
PortNumber=<Port on which Oracle is listening on HostName>
SID=<Your Oracle SID>
CatalogOptions=0
EnableStaticCursorsForLongData=0
ApplicationUsingThreads=1
EnableDescribeParam=1
OptimizePrepare=1
WorkArounds=536870912
ProcedureRetResults=1
ColumnSizeAsCharacter=1
LoginTimeout=0

;# Oracle Real Application Cluster stanza
[ORACLERACDB]
Driver=<Your Broker install directory>/ODBC/V6.0/lib/UKor824.so
Description=DataDirect 6.0 Wire Protocol
HostName=<Your Oracle Server Machine Name>
PortNumber=<Port on which Oracle is listening on HostName>
ServiceName=<Your Oracle Real Application Cluster Service Name>
;#This shows one alternate server definition. Add extra ones using a ',' to seperate each definiti
AlternateServers=(HostName=<Your alternative hostname>:PortNumber=<Port number on which Oracle is
CatalogOptions=0
EnableStaticCursorsForLongData=0
ApplicationUsingThreads=1
EnableDescribeParam=1
OptimizePrepare=1
WorkArounds=536870912
ProcedureRetResults=1
ColumnSizeAsCharacter=1
LoginTimeout=0

;# Sybase Stanza
[SYBASEDB]
Driver=<Your Broker install directory>/ODBC/V6.0/lib/UKase24.so
Description=DataDirect 6.0 Sybase Wire Protocol
Database=<Your Database Name>
ServerName=<Your Sybase Server Name>
```

```
EnableDescribeParam=1
OptimizePrepare=1
SelectMethod=0
NetworkAddress=<Your Sybase Server Name>,<Your Sybase Port Number>
SelectUserName=1
ColumnSizeAsCharacter=1
EnableSPColumnTypes=2
LoginTimeout=0
TimestampTruncationBehavior=1
XAConnOptBehaviour=3

;# Sybase Stanza for a UTF8 datasource
[SYBASEDBUTF8]
Driver=<Your Broker install directory>/ODBC/V6.0/lib/UKase24.so
Description=DataDirect 6.0 Sybase Wire Protocol
Database=<Your Database Name>
ServerName=<Your Sybase Server Name>
EnableDescribeParam=1
OptimizePrepare=1
SelectMethod=0
NetworkAddress=<Your Sybase Server Name>,<Your Sybase Port Number>
SelectUserName=1
ColumnSizeAsCharacter=1
Charset=UTF8
EnableSPColumnTypes=2
LoginTimeout=0
TimestampTruncationBehavior=1
XAConnOptBehaviour=3

;# UNIX to SQLServer stanza
[SQLSERVERDB]
Driver=<Your Broker install directory>/ODBC/V6.0/lib/UKmsss24.so
Description=DataDirect 6.0 SQL Server Wire Protocol
Address=<Your SQLServer Machine Name>,<Your SQLServer Port Number>
;# Alternative way to locate server using a named instance
;# Address=<Your SQLServer Machine Name>\<Your SQLServer Instance Name>
AnsiNPW=Yes
QuotedId=No
ColumnSizeAsCharacter=1

;# Informix Stanza
[INFORMIXDB]
Driver=<Your Informix Client Directory>/lib/cli/iclit09b.so
Description=IBM Informix ODBC Driver
ServerName=<Your Informix Server Name>
Database=<Your Database Name>

;########################################
;###### Mandatory information stanza ######
;########################################

[ODBC]
;# To turn on ODBC trace set Trace=1
Trace=0
TraceOptions=3
TraceFile=<A Directory with plenty of free space to hold trace
output>/odbctrace.out
TraceDll=<Your Broker install directory>/ODBC/V6.0/lib/odbctrac.so
InstallDir=<Your Broker install directory>/ODBC/V6.0
UseCursorLib=0
IANAAppCodePage=4
UNICODE=UTF-8
```

Linux on POWER, and Solaris on x86-64

```
;#########################################################################
;# ODBC database driver manager initialisation file.                     #
;#########################################################################
;# It is recommended that you take a copy of this file and then edit the  #
;# copy.                                                                 #
```

```
;#                                                                            #
;# 1. Complete the 'Mandatory information stanza' section                     #
;# at the end of the file.                                                    #
;#                                                                            #
;# 2. For each data source, add the name of the data source into              #
;# the 'List of data sources stanza' section.                                 #
;#                                                                            #
;# 3. For each data source, create a stanza in the                            #
;# 'Individual data source stanzas' section.                                  #
;##############################################################################

;########################################
;###### List of data sources stanza #######
;########################################
[ODBC Data Sources]
DB2DB=IBM DB2 ODBC Driver
ORACLEDB=DataDirect 6.0 Oracle Wire Protocol
ORACLERACDB=DataDirect 6.0 Oracle Wire Protocol (Real Application Clusters)
SYBASEDB=DataDirect 6.0 Sybase Wire Protocol
SYBASEDBUTF8=DataDirect 6.0 Sybase UTF8 Wire Protocol
INFORMIXDB=IBM Informix ODBC Driver

;#########################################################
;########## Individual data source stanzas ################
;#########################################################

;# DB2 stanza
[DB2DB]
DRIVER=libdb2Wrapper.so
Description=DB2DB DB2 ODBC Database
Database=DB2DB

;# Oracle stanza
[ORACLEDB]
Driver=<Your Broker install directory>/ODBC/V6.0/lib/UKora24.so
Description=DataDirect 6.0 Oracle Wire Protocol
HostName=<Your Oracle Server Machine Name>
PortNumber=<Port on which Oracle is listening on HostName>
SID=<Your Oracle SID>
CatalogOptions=0
EnableStaticCursorsForLongData=0
ApplicationUsingThreads=1
EnableDescribeParam=1
OptimizePrepare=1
WorkArounds=536870912
ProcedureRetResults=1
ColumnSizeAsCharacter=1
LoginTimeout=0

;# Oracle Real Application Clusters stanza
[ORACLERACDB]
Driver=<Your Broker install directory>/ODBC/V6.0/lib/UKora24.so
Description=DataDirect 6.0 Oracle Wire Protocol
HostName=<Your Oracle Server Machine Name>
PortNumber=<Port on which Oracle is listening on HostName>
ServiceName=<Your Oracle Real Application Cluster Service Name>
AlternateServers=(HostName=<Your alternative host name>:PortNumber=<Port on
which Oracle is listening on the alternative host>:ServiceName=<Your
Oracle Real Application Cluster Service Name>)
CatalogOptions=0
EnableStaticCursorsForLongData=0
ApplicationUsingThreads=1
EnableDescribeParam=1
OptimizePrepare=1
WorkArounds=536870912
ProcedureRetResults=1
ColumnSizeAsCharacter=1
LoginTimeout=0
```

```
;# Sybase Stanza
[SYBASEDB]
Driver=<Your Broker install directory>/ODBC/V6.0/lib/UKase24.s0
Description=DataDirect 6.0 Sybase Wire Protocol
Database=<Your Database Name>
ApplicationUsingThreads=1
EnableDescribeParam=1
OptimizePrepare=1
SelectMethod=0
NetworkAddress=<Your Sybase Server Name>,<Your Sybase Port Number>
SelectUserName=1
ColumnSizeAsCharacter=1
EnableSPColumnTypes=2
LoginTimeout=0
TimestampTruncationBehavior=1
XAConnOptBehaviour=3

;# Sybase Stanza for a UTF8 datasource
[SYBASEDBUTF8]
Driver=<Your Broker install directory>/ODBC/V6.0/lib/UKase24.so
Description=DataDirect 6.0 Sybase Wire Protocol
Database=<Your Database Name>
ApplicationUsingThreads=1
EnableDescribeParam=1
OptimizePrepare=1
SelectMethod=0
NetworkAddress=<Your Sybase Server Name>,<Your Sybase Port Number>
SelectUserName=1
ColumnSizeAsCharacter=1
Charset=UTF8
EnableSPColumnTypes=2
LoginTimeout=0
TimestampTruncationBehavior=1
XAConnOptBehaviour=3

;# Informix Stanza
[INFORMIXDB]
Driver=libinfWrapper.so
Description=IBM Informix ODBC Driver
ServerName=<Your Informix Server Name>
Database=<Your Database Name>

;########################################
;###### Mandatory information stanza ######
;########################################

[ODBC]
;# To turn on ODBC trace set Trace=1
Trace=0
TraceOptions=3
TraceFile=<A Directory with plenty of free space to hold trace
output>/odbctrace.out
TraceDll=<Your Broker install directory>/ODBC/V6.0/lib/odbctrac.so
InstallDir=<Your Broker install directory>/ODBC/V6.0
UseCursorLib=0
IANAAppCodePage=4
UNICODE=UTF-8
```

Linux on IBM z Systems

```
;#############################################################################
;# ODBC database driver manager initialisation file.                         #
;#############################################################################
;# It is recommended that you take a copy of this file and then edit the  #
;# copy.                                                                     #
;#                                                                           #
;# 1. Complete the 'Mandatory information stanza' section                    #
;# at the end of the file.                                                   #
;#                                                                           #
```

```
;# 2. For each data source, add the name of the data source into          #
;# the 'List of data sources stanza' section.                             #
;#                                                                        #
;# 3. For each data source, create a stanza in the                        #
;# 'Individual data source stanzas' section.                              #
;##########################################################################

;########################################
;###### List of data sources stanza #######
;########################################
[ODBC Data Sources]
DB2DB=IBM DB2 ODBC Driver
ORACLEDB=DataDirect 6.0 Oracle Wire Protocol
ORACLERACDB=DataDirect 6.0 Oracle Wire Protocol (Real Application Clusters)
SQLSERVERDB=DataDirect 6.0 SQL Server Wire Protocol
INFORMIXDB=IBM Informix ODBC Driver

;#########################################################
;########## Individual data source stanzas ################
;#########################################################

;# DB2 stanza
[DB2DB]
DRIVER=libdb2Wrapper.so
Description=DB2DB DB2 ODBC Database
Database=DB2DB

;# Oracle stanza
[ORACLEDB]
Driver=<Your Broker install directory>/ODBC/V6.0/lib/UKora24.so
Description=DataDirect 6.0 Oracle Wire Protocol
HostName=<Your Oracle Server Machine Name>
PortNumber=<Port on which Oracle is listening on HostName>
SID=<Your Oracle SID>
CatalogOptions=0
EnableStaticCursorsForLongData=0
ApplicationUsingThreads=1
EnableDescribeParam=1
OptimizePrepare=1
WorkArounds=536870912
ProcedureRetResults=1
ColumnSizeAsCharacter=1
LoginTimeout=0

;# Oracle Real Application Clusters stanza
[ORACLERACDB]
Driver=<Your Broker install directory>/ODBC/V6.0/lib/UKora24.so
Description=DataDirect 6.0 Oracle Wire Protocol
HostName=<Your Oracle Server Machine Name>
PortNumber=<Port on which Oracle is listening on HostName>
ServiceName=<Your Oracle Real Application Cluster Service Name>
AlternateServers=(HostName=<Your alternative host name>:PortNumber=<Port on
which Oracle is listening on the alternative host>:ServiceName=<Your
Oracle Real Application Cluster Service Name>)
CatalogOptions=0
EnableStaticCursorsForLongData=0
ApplicationUsingThreads=1
EnableDescribeParam=1
OptimizePrepare=1
WorkArounds=536870912
ProcedureRetResults=1
ColumnSizeAsCharacter=1
LoginTimeout=0

;# SQLServer stanza
[SQLSERVERDB]
Driver=<Your Broker install directory>/ODBC/V6.0/lib/UKmsss24.so
Description=DataDirect 6.0 SQL Server Wire Protocol
Address=<Your SQLServer Machine Name>,<Your SQLServer Port Number>
;#Alternative way to locate server using a named instance
;#Address=<Your SQLServer Machine Name>\<Your SQLServer Instance Name>
```

```
AnsiNPW=Yes
Database=db
QuotedId=No
ColumnSizeAsCharacter=1

;# Informix Stanza
[INFORMIXDB]
Driver=libinfWrapper.so
Description=IBM Informix ODBC Driver
ServerName=<Your Informix Server Name>
Database=<Your Database Name>

;########################################
;###### Mandatory information stanza ######
;########################################

[ODBC]
;# To turn on ODBC trace set Trace=1
Trace=0
TraceOptions=3
TraceFile=<A Directory with plenty of free space to hold trace
output>/odbctrace.out
TraceDll=<Your Broker install directory>/ODBC/V6.0/lib/odbctrac.so
InstallDir=<Your Broker install directory>/ODBC/V6.0
UseCursorLib=0
IANAAppCodePage=4
UNICODE=UTF-8
```

On UNIX and Linux systems, you can check that the ODBC environment is configured correctly by running the **mqsicvp** command. This command also validates the connection to all data sources that are listed in the `odbc.ini` file that have been associated with a broker by using the **mqsisetdbparms** command. For more information, see "**mqsicvp** command" on page 3857.

Related tasks:

"Connecting to a database from Linux and UNIX systems using the DataDirect drivers" on page 674
To enable a broker to connect to a database, define the ODBC data source name (DSN) for the database.

Related reference:

"Supported databases" on page 3591
You can optionally configure databases to contain data that is accessed by your message flows. Databases from IBM and other suppliers are supported at specific versions on supported operating systems.

"Sample WebSphere Message Broker ODBC Database Extender (IE02) configuration files" on page 3596
How you use WebSphere Message Broker ODBC Database Extender (IE02) to load the correct data source driver.

Support for Unicode and DBCS data in databases

You can manipulate Unicode Standard version 3.0 data, in suitably configured databases, using ESQL, in nodes that access databases by ODBC. The broker does not support DBCS-only columns in tables that are defined in databases.

The broker does not, therefore, support certain data types, including the following types:

- NCHAR, NVARCHAR, NVARCHAR2, NCLOB (on Oracle)
- NCHAR, NVARCHAR, NTEXT, UNICHAR, UNIVARCHAR (on Sybase)
- NCHAR, NVARCHAR (on Informix)

For WebSphere Message Broker 7.0.0.4 and later, GRAPHIC, VARGRAPHIC, LONGVARGRAPHIC, and DBCLOB data type support on DB2 is provided for the broker with the following limitations:

- Due to issues related to the z/OS Unicode support when calling stored procedures by using PASSTHRU, you must not use this method of calling stored procedures. For more information, see "PASSTHRU statement" on page 5147.
- Using the DB2 string functions with Unicode data can return unexpected results. For more information, see "Unicode string functions in DB2" on page 3670.

Support for Unicode is available only for the generally-supported versions of the following database managers:

- IBM DB2 for Windows, Linux, UNIX, and z/OS operating systems.
- Oracle
- Microsoft SQL server
- Sybase Adaptive Server Enterprise (ASE)

For information about the versions of databases supported, see WebSphere Message Broker Requirements.

Support for the manipulation of Unicode data is not available for nodes that access databases that use JDBC; for example, DatabaseRetrieve and DatabaseRoute.

The following instructions apply to both 32-bit and 64-bit applications.

If you are using DB2:

- On Windows, Linux, and UNIX operating systems, your database must be created with code set `utf-8`.
- On z/OS, set the variable MQSI_DB2_CONVERSION in the broker environment to the value `UNICODE`. In the ODBC definition add the statement `CURRENTAPPENSCH=UNICODE` to the `[COMMON]` stanza.
- On all platforms, DB2 returns the lengths of strings in bytes, rather than characters; this response has implications for the behavior of string length-related ESQL functions.

  Some functions might fail, or function differently, when processed by the database. See "Unicode string functions in DB2" on page 3670 for further information.

If you are using Oracle:

- Your database must be created with NLS_CHARACTERSET of AL32UTF8.
- Your ODBC data source definition must include the setting `ColumnSizeAsCharacter=1`.

  On UNIX and Linux platforms, this setting must be included in the appropriate stanza in the ODBC ini files.

  On Windows platforms, this string value must be added to the ODBC data source key in the registry.

  See "Enabling ODBC connections to the databases" on page 668 for further information.
- For 32-bit connections, you must set the variable NLS_LANG in the broker environment to the value `<yourlanguage>_<yourterritory>.AL32UTF8`.

if you are using Microsoft SQL server:

- You must use NCHAR, NVARCHAR, and NTEXT data types for your column definitions.
- For brokers on UNIX and Linux platforms, your ODBC data source definition must include the setting `ColumnSizeAsCharacter=1`; this setting must be included in the appropriate stanza in the ODBC `.ini` files.

If you are using Sybase ASE:

- The default character set of your ASE server must be `UTF-8`.
- Your ODBC data source definition must include the settings `ColumnSizeAsCharacter=1` and `CharSet=UTF8`.

  On UNIX and Linux platforms, this setting must be included in the appropriate stanza in the ODBC `.ini` files.

  On Windows platforms, this string value must be added to the ODBC data source key in the registry.

  See "Enabling ODBC connections to the databases" on page 668 for further information.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"User database connections" on page 2110
User databases contain business data that is written and accessed by deployed message flows. You must create connections from the broker to the user database by using ODBC or JDBC.

Related tasks:

"Accessing databases from message flows" on page 2112
Create and configure message flows to access user databases.

"Configuring transactionality for message flows" on page 1290
A message flow runs in a single transaction, which is started when data is received by an input node, and can be committed or rolled back when all processing has completed.

Related reference:

"Supported databases" on page 3591
You can optionally configure databases to contain data that is accessed by your message flows. Databases from IBM and other suppliers are supported at specific versions on supported operating systems.

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Unicode string functions in DB2:

When you use string functions in ESQL expressions, certain parameters refer to character positions or counts.

For example, with the SUBSTRING function, coding:

```
SUBSTRING('Hello World!' FROM 7 FOR 4)
```

results in the string `Worl` because `7` refers to the seventh character position, and `4` refers to four characters.

If the string `Hello World!` is represented in an ASCII code page, the seventh *byte* of that representation is the seventh character. However, in other code pages or encoding schemes (for example, some Unicode representations) the seventh character could start at byte 13, and 4 characters can occupy 8 bytes.

WebSphere Message Broker correctly handles this situation; that is, the numeric parameters and results of these string functions always refer to *characters* and not the bytes used to represent them.

In some situations, WebSphere Message Broker delegates expressions to a database engine for processing. For example, if there is a WHERE clause, in a SELECT function, applied to a database data source, the database is passed the WHERE clause if it can interpret all the functions in the expression.

If there are functions that are not supported by the database, WebSphere Message Broker passes only those parts of the expression that can be interpreted, retrieves an unfiltered record set, and performs the remaining filtering itself.

DB2 string functions use *byte* indexing and not *character* indexing. Therefore, for Unicode data, the meaning of certain functions differs from the WebSphere Message Broker functions, even though they can be ‘interpreted'.

Characters in Unicode UTF8 representation, for example, can occupy from 1-4 bytes, so that the seventh character can start anywhere from byte 7 to byte 25.

The following string functions are affected:

- INSERT function
- LEFT function
- LENGTH function
- OVERLAY function
- POSITION function
- RIGHT function
- SPACE function
- SUBSTRING function

These functions either take numeric parameters, or return numeric results that refer to indexes, or counts, of characters in a string.

When expressions involving these functions are passed to the DB2 database, and Unicode string data is manipulated in the database, the results can be unexpected, or an error might occur.

This error might also occur if, for example, an expression of this type is passed directly to the database by using the PASSTHRU function. In this situation, you could modify each expression yourself, as necessary, for the target database.

It is not possible to systematically modify expressions to avoid this problem and WebSphere Message Broker does not attempt to do so.

If the Unicode strings do not use any characters that, in UTF8 representation, occupy more than 1 byte each, the functions perform correctly.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Accessing databases from message flows" on page 2112
Create and configure message flows to access user databases.

Related reference:

"Support for Unicode and DBCS data in databases" on page 3668
You can manipulate Unicode Standard version 3.0 data, in suitably configured databases, using ESQL, in nodes that access databases by ODBC. The broker does not support DBCS-only columns in tables that are defined in databases.

Administration API

Use the Administration API for WebSphere Message Broker (CMP API) Java classes and methods to develop CMP applications.

Full details of the Java classes and methods are provided in the CMP API Javadoc information.

Related concepts:

"The Administration API for WebSphere Message Broker" on page 54
The Administration API for WebSphere Message Broker is a programming interface that your applications can use to control brokers and their resources through a remote interface.

Related tasks:

"Developing applications that use the Administration API" on page 956
Develop Java applications that use the Administration API (also known as the CMP API) to communicate with, deploy to, and manage brokers and their associated resources.

Commands

All WebSphere Message Broker Toolkit and runtime commands that are provided on distributed systems are listed, grouped by function, with references to command details.

For information about the equivalent commands on z/OS, see "Summary of commands on Linux, UNIX, Windows, and z/OS systems" on page 3674.

WebSphere Message Broker Toolkit commands

Command name	Topic reference
mqsiapplybaroverride	"**mqsiapplybaroverride** command" on page 3684
mqsicreatebar	"**mqsicreatebar** command" on page 3699
mqsicreatemsgdefs	"**mqsicreatemsgdefs** command" on page 3702
mqsicreatemsgdefsfromwsdl	"**mqsicreatemsgdefsfromwsdl** command" on page 3712
mqsireadbar	"**mqsireadbar** command" on page 3697

WebSphere Message Broker commands

Table 43. Broker commands

Command name	Topic reference
mqsiaddbrokerinstance	"**mqsiaddbrokerinstance** command" on page 3715
mqsiapplybaroverride	"**mqsiapplybaroverride** command" on page 3684
mqsibackupbroker	"**mqsibackupbroker** command" on page 3720
mqsichangebroker	"**mqsichangebroker** command" on page 3723
mqsicreatebroker	"**mqsicreatebroker** command" on page 3831
mqsicreateexecutiongroup	"**mqsicreateexecutiongroup** command" on page 3854
mqsideletebroker	"**mqsideletebroker** command" on page 3863
mqsideleteexecutiongroup	"**mqsideleteexecutiongroup** command" on page 3869
mqsimode	"**mqsimode** command" on page 3899
mqsireadbar	"**mqsireadbar** command" on page 3697
mqsireload	"**mqsireload** command" on page 3909
mqsiremovebrokerinstance	"**mqsiremovebrokerinstance** command" on page 3918
mqsireportbroker	"**mqsireportbroker** command" on page 3919
mqsirestorebroker	"**mqsirestorebroker** command" on page 3952

Table 44. Database commands

Command name	Topic reference
mqsimanagexalinks	"**mqsimanagexalinks** command" on page 3891
mqsisetdbparms	"**mqsisetdbparms** command" on page 3954

Table 45. Security commands

Command name	Topic reference
mqsireloadsecurity	"**mqsireloadsecurity** command" on page 3911
mqsisetdbparms	"**mqsisetdbparms** command" on page 3954
mqsisetsecurity	"**mqsisetsecurity** command" on page 3964

Table 46. Start and stop commands

Command name	Topic reference
mqsistart	"**mqsistart** command" on page 3965
mqsistartmsgflow	"**mqsistartmsgflow** command" on page 3969
mqsistop	"**mqsistop** command" on page 3972
mqsistopmsgflow	"**mqsistopmsgflow** command" on page 3975

Table 47. List and trace commands

Command name	Topic reference
mqsichangetrace	"**mqsichangetrace** command" on page 3822
mqsiformatlog	"**mqsiformatlog** command" on page 3880
mqsilist	"**mqsilist** command" on page 3882
mqsireadlog	"**mqsireadlog** command" on page 3905

Table 47. List and trace commands (continued)

Command name	Topic reference
mqsireporttrace	"**mqsireporttrace** command" on page 3947

Table 48. Migration commands

Command name	Topic reference
mqsimigratecomponents	"**mqsimigratecomponents** command" on page 3894

Table 49. Properties commands

Command name	Topic reference
mqsichangeproperties	"**mqsichangeproperties** command" on page 3756
mqsireportproperties	"**mqsireportproperties** command" on page 3937

Table 50. Monitoring commands

Command name	Topic reference
mqsichangeflowmonitoring	"**mqsichangeflowmonitoring** command" on page 3738
mqsireportflowmonitoring	"**mqsireportflowmonitoring** command" on page 3924

Table 51. Statistics commands

Command name	Topic reference
mqsichangeflowstats	"**mqsichangeflowstats** command" on page 3744
mqsichangeresourcestats	"**mqsichangeresourcestats** command" on page 3819
mqsireportflowstats	"**mqsireportflowstats** command" on page 3929
mqsireportresourcestats	"**mqsireportresourcestats** command" on page 3944

Table 52. Miscellaneous commands

Command name	Topic reference
mqsichangeflowuserexits	"**mqsichangeflowuserexits** command" on page 3751
mqsicommandconsole	"**mqsicommandconsole** command" on page 3830
mqsicreateconfigurableservice	"**mqsicreateconfigurableservice** command" on page 3849
mqsicvp	"**mqsicvp** command" on page 3857
mqsideleteconfigurableservice	"**mqsideleteconfigurableservice** command" on page 3866
mqsideploy	"**mqsideploy** command" on page 3872
mqsiexplain	"**mqsiexplain** command" on page 3879
mqsireportflowuserexits	"**mqsireportflowuserexits** command" on page 3933

Summary of commands on Linux, UNIX, Windows, and z/OS systems

The following table summarizes the runtime commands that are available on Linux, UNIX, and Windows systems, and provides the z/OS equivalent, where it is available.

Command on Windows, Linux, and UNIX systems	z/OS equivalent: type	z/OS equivalent	z/OS References
mqsiapplybaroverride	Utility JCL	BIPOBAR	"Contents of the broker PDSE" on page 3991
mqsibackupbroker	Utility JCL	BIPBUBK	"Contents of the broker PDSE" on page 3991
mqsichangebroker	1. Console command: modify 2. Utility JCL	1. changebroker 2. BIPCHBK	1. "**mqsichangebroker** command" on page 3723 2. "Contents of the broker PDSE" on page 3991
mqsichangeflowmonitoring	1. Console command: modify 2. Utility JCL	1. changeflowmonitoring 2. BIPCHME	1. "**mqsichangeflowmonitoring** command" on page 3738 2. "Contents of the broker PDSE" on page 3991
mqsichangeflowstats	1. Console command: modify 2. Utility JCL	1. changeflowstats 2. BIPCHMS	1. "**mqsichangeflowstats** command" on page 3744 2. "Contents of the broker PDSE" on page 3991
mqsichangeflowuserexits	1. Console command: modify 2. Utility JCL	1. changeflowuserexits 2. BIPCHUE	1. "**mqsichangeflowuserexits** command" on page 3751 2. "Contents of the broker PDSE" on page 3991
mqsichangeproperties	Utility JCL	BIPCHPR	"Contents of the broker PDSE" on page 3991
mqsichangeresourcestats	Utility JCL	BIPCHRS	"**mqsichangeresourcestats** command" on page 3819
mqsichangetrace	Console command: modify	changetrace	"**mqsichangetrace** command" on page 3822
mqsicreatebroker	Utility JCL	BIPCRBK	"Contents of the broker PDSE" on page 3991
mqsicreateconfigurableservice	Utility JCL	BIPJADPR	"Contents of the broker PDSE" on page 3991
mqsicreateexecutiongroup	Utility JCL	BIPCREG	"Contents of the broker PDSE" on page 3991
mqsicvp	Not applicable		
mqsideletebroker	Utility JCL	BIPDLBK	"Contents of the broker PDSE" on page 3991
mqsideleteconfigurableservice	Utility JCL	BIPJADPR	"Contents of the broker PDSE" on page 3991
mqsideleteexecutiongroup	Utility JCL	BIPDLEG	"Contents of the broker PDSE" on page 3991
mqsideploy	1. Console command: modify 2. Utility JCL	1. deploy 2. BIPDPLY	1. "**mqsideploy** command" on page 3872 2. "Contents of the broker PDSE" on page 3991
mqsiexplain	Utility JCL	BIPEXPL	"Contents of the broker PDSE" on page 3991

Command on Windows, Linux, and UNIX systems	z/OS equivalent: type	z/OS equivalent	z/OS References
mqsiformatlog	Utility JCL	BIPFMLG	"Contents of the broker PDSE" on page 3991
mqsilist	1. Console command: modify 2. Utility JCL	1. list 2. BIPLIST	1. "**mqsilist** command" on page 3882 2. "Contents of the broker PDSE" on page 3991
mqsimanagexalinks	Not applicable		
mqsimigratecomponents	Utility JCL	BIPMGCMP	"**mqsimigratecomponents** command" on page 3894
mqsimode	Utility JCL	BIPMODE	"Contents of the broker PDSE" on page 3991
mqsireadbar	Utility JCL	BIPRBAR	"Contents of the broker PDSE" on page 3991
mqsireadlog	Utility JCL	BIPRELG	"Contents of the broker PDSE" on page 3991
mqsireload	Console command: modify	reload	"**mqsireload** command" on page 3909
mqsireloadsecurity	1. Console command: modify 2. Utility JCL	1. reloadsecurity 2. BIPRLSEC	1. "**mqsireloadsecurity** command" on page 3911 2. "Contents of the broker PDSE" on page 3991
mqsireportbroker	Utility JCL	BIPRPBK	"Contents of the broker PDSE" on page 3991
mqsireportflowmonitoring	1. Console command: modify 2. Utility JCL	1. reportflowmonitoring 2. BIPRPME	1. "**mqsireportflowmonitoring** command" on page 3924 2. "Contents of the broker PDSE" on page 3991
mqsireportflowstats	1. Console command: modify 2. Utility JCL	1. reportflowstats 2. BIPRPMS	1. "**mqsireportflowstats** command" on page 3929 2. "Contents of the broker PDSE" on page 3991
mqsireportflowuserexits	1. Console command: modify 2. Utility JCL	1. reportflowuserexits 2. BIPRPUE	1. "**mqsireportflowuserexits** command" on page 3933 2. "Contents of the broker PDSE" on page 3991
mqsireportproperties	Utility JCL	BIPRPPR	"Contents of the broker PDSE" on page 3991
mqsireportresourcestats	Utility JCL	BIPRPRS	"**mqsireportresourcestats** command" on page 3944
mqsireporttrace	Console command: modify	reporttrace	"**mqsireporttrace** command" on page 3947
mqsirestorebroker	Utility JCL	BIPRSBK	"Contents of the broker PDSE" on page 3991
mqsisetdbparms	Utility JCL	BIPSDBP	"**mqsisetdbparms** command" on page 3954
mqsisetsecurity	Not applicable		

Command on Windows, Linux, and UNIX systems	z/OS equivalent: type	z/OS equivalent	z/OS References
mqsi_setupdatabase	Not applicable		
mqsistart	1. Console command: start 2. Console command: modify	1. Standard MVS start command 2. `startcomponent`	1. - 2. "**mqsistart** command" on page 3965
mqsistartmsgflow	Utility JCL	`BIPSTMF`	"Contents of the broker PDSE" on page 3991
mqsistop	1. Console command: stop 2. Console command: modify	1. Standard MVS stop command 2. `'p' stopcomponent`	1. - 2. "**mqsistop** command" on page 3972
mqsistopmsgflow	Utility JCL	`BIPSPMF`	"Contents of the broker PDSE" on page 3991

Related reference:

"WebSphere Message Broker Toolkit commands" on page 3699
Several commands are provided as part of the WebSphere Message Broker Toolkit.

"Runtime commands" on page 3715
The topics in this section describe the WebSphere Message Broker runtime commands.

"Runtime and WebSphere Message Broker Toolkit commands (common)" on page 3683
Some commands are common to both runtime and WebSphere Message Broker Toolkit environments.

"Characters allowed in commands" on page 3680
You must adhere to a few rules when you provide names or identifiers for the components and resources in your broker environment.

"Rules for using commands" on page 3681
Observe the following rules when using the WebSphere Message Broker commands on distributed systems.

Syntax diagrams

The syntax for commands and ESQL statements and functions is presented in the form of a railroad diagram. The diagram tells you what you can do with the command, statement, or function and indicates relationships between different options and, sometimes, different values of an option.

For details about how to read a railroad diagram, see "How to read railroad diagrams."

Related reference:

"Commands" on page 3672
All WebSphere Message Broker Toolkit and runtime commands that are provided on distributed systems are listed, grouped by function, with references to command details.

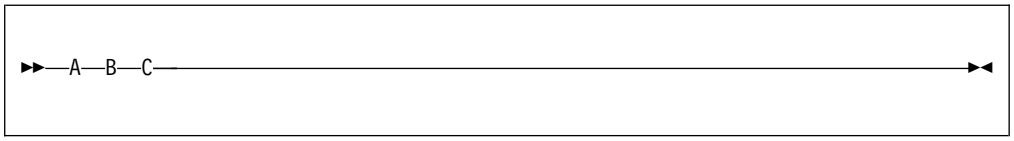
How to read railroad diagrams:

Understand the syntax used in railroad diagrams that show syntax.

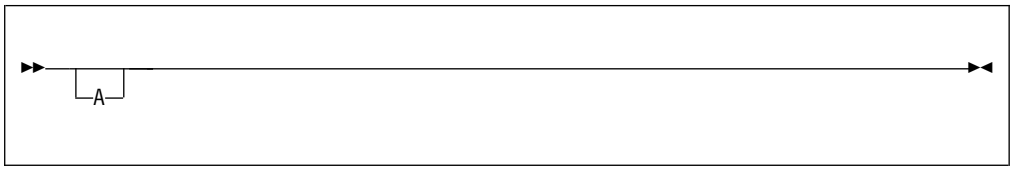
Each railroad diagram begins with a double right arrow and ends with a right and left arrow pair. Lines that begin with a single right arrow are continuation lines. You read a railroad diagram from left to right and from top to bottom, following the direction of the arrows.

The following examples show other conventions used in railroad diagrams.

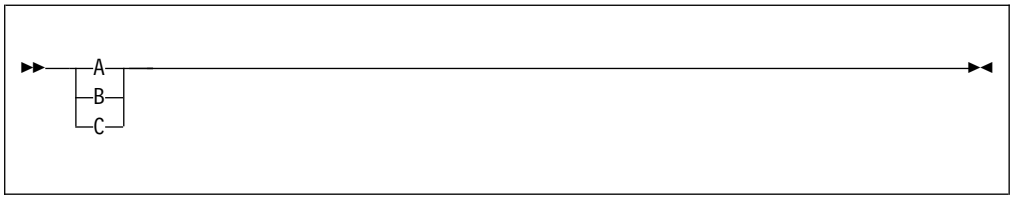
This example shows that you must specify values `A`, `B`, and `C`. Required values are shown on the main line of a railroad diagram:



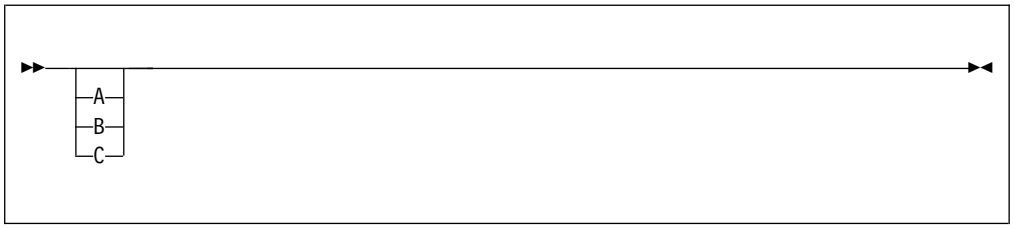
This shows that you can specify value `A`. Optional values are shown below the main line of a railroad diagram:



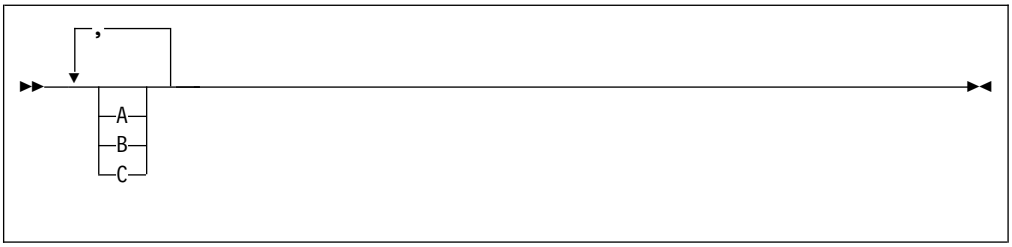
The next example specifies that values `A`, `B`, and `C` are options, one of which you must specify:



Values `A`, `B`, and `C` are options in this example, one of which you can specify:

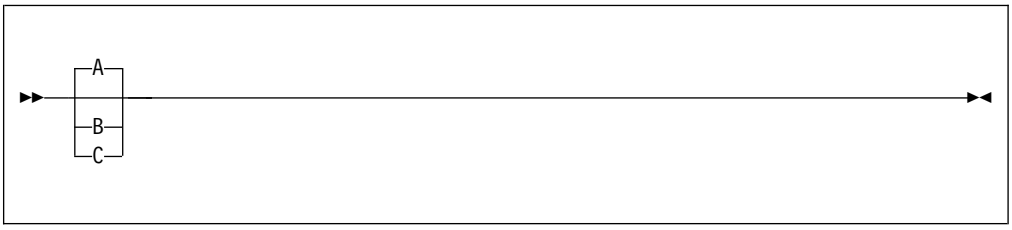


The next example shows that you can specify one or more of the values `A`, `B`, and `C`. Any required separator for multiple or repeated values (in this example, the comma (,)) is shown on the arrow:



In this example, you can specify value A multiple times. The separator in this example is optional:

Values A, B, and C are alternatives in the next example, one of which you can specify. If you specify none of the values shown, the default A (the value shown above the main line) is used:



The last example shows the use of a syntax fragment Name, which is shown separately from the main railroad diagram. This technique is used to simplify the diagram, or help fit it into the page of text. The fragment can be used multiple times in the railroad diagram:

Punctuation and uppercase values must be specified exactly as shown.

Lowercase values (for example, *name*) indicate where to type your own text in place of the *name* variable.

Related reference:

"Commands" on page 3672
All WebSphere Message Broker Toolkit and runtime commands that are provided on distributed systems are listed, grouped by function, with references to command details.

Characters allowed in commands

You must adhere to a few rules when you provide names or identifiers for the components and resources in your broker environment.

The character set that you can use to name brokers, execution groups, and message identifiers is as follows:

- Uppercase alphabetic characters A-Z
- Lowercase alphabetic characters a-z
- Numeric characters 0-9
- All special characters supported by the underlying file system:
  - The following special characters are accepted on Windows platforms:

$	%	' (apostrophe)	" (quotation mark)
- (dash)	_ (underscore)	@	~ (tilde)
!	(	)	{
}	[	]	&
#	&	+	, (comma)
;	=	(space)	

  - The following special characters, except for a space, are accepted on Linux and UNIX platforms:

. (dot)	%	- (dash)	_ (underscore)
@	~ (tilde)	!	{
}	[	]	&
#	, (comma)	=	(space)
+			

In general, you can use characters **A** through **Z**, **a** through **z**, and **0** through **9**.

If you expect to trace the operation of an execution group, restrict the name of the execution group to include only the valid alphabetic and numeric characters listed. The trace commands do not support the use of special characters for an execution group name.

If you intend to use WebSphere MQ File Transfer Edition for managed file transfers, see "Preparing the environment for WebSphere MQ File Transfer Edition nodes" on page 740 for information about naming execution groups.

The acceptable character set for configurable service names is identical to the character set for directory names on the file system of the target broker. The names of deployable resources (such as message flows, dictionaries, and JAR files) must consist only of characters that are acceptable in file names on the file systems of both the WebSphere Message Broker Toolkit and the target broker.

On Windows platforms, broker names are not case-sensitive. For example, broker names `Broker1` and `BROKER1` refer to the same broker.

On Linux and UNIX systems, broker names are case sensitive, and the previous examples refer to different brokers.

On z/OS systems, you must enclose mixed-case names in quotation marks.

Additional rules are enforced for naming message service folders in the MQRFH2 header.

Related reference:

"Commands" on page 3672
All WebSphere Message Broker Toolkit and runtime commands that are provided on distributed systems are listed, grouped by function, with references to command details.

"Syntax diagrams" on page 3677

"Message service folders" on page 6401
A number of folders are defined for use by WebSphere MQ products.

Rules for using commands

Observe the following rules when using the WebSphere Message Broker commands on distributed systems.

If you are using commands on z/OS, refer to the section on z/OS commands in "Commands" on page 3672.

- Each command must be issued on the system on which the resource to which it relates is defined (or is to be created).
- Each command starts with a primary keyword (the executable command name) followed by one or more blanks.
- Following the primary keyword, flags (parameters) can occur in any order.
- Flags are shown in this book in the form **-t**, for example. In all cases, the character **/** can be substituted for the **-** character.
- If a flag has a corresponding value, its value must follow the flag to which it relates. A flag can be followed by its value directly or can be separated by any number of blanks.
- Flags can be concatenated if they do not have corresponding values, although the last flag in a concatenated group *can* have a value associated with it. For example, the command:

  ```
  mqsireadlog MB7BROKER -u -e default -o trace.xml -f
  ```

  can be entered as:

  ```
  mqsireadlog MB7BROKER -ufedefault -o trace.xml
  ```

  where the name of the execution group, `default`, relates to the **-e** flag. For clarity, all examples given in this documentation are shown with separate flags and with a space before any associated value.
- Repeated flags are not allowed.
- Strings that contain blanks or special characters must be enclosed in double quotation marks. For example:

  ```
  mqsireadlog "My Broker" -u -e default -o trace.xml -f
  ```

  Additionally, you can specify a null, or empty, string with a pair of quotation marks with nothing between: "".
- z/OS If you are running a command by submitting one of the JCL utilities (see "Contents of the broker PDSE" on page 3991), and an argument to one of the parameters contains a blank character, you must replace the blank with the mnemonic ` ` before you submit the JCL. For example, if you are using **BIPDPLY** to run the **mqsideploy** command to delete a deployed `.jar` file that is named `my jar.jar` from the execution group *default*, then modify the JCL to contain the following sample:

```
mqsideploy MB7BROKER -e default -d my jar.jar
```

- The case sensitivity of primary keywords and parameters depends on the underlying operating system. On Windows platforms, keywords are not case sensitive; **mqsistart**, mqsiSTART, and MQSISTART are all acceptable. On Linux and UNIX platforms, you must use lowercase; only **mqsistart** is acceptable.

All WebSphere Message Broker commands have dependencies on WebSphere MQ function. You must ensure that WebSphere MQ is available before issuing these commands.

Related reference:

"Characters allowed in commands" on page 3680
You must adhere to a few rules when you provide names or identifiers for the components and resources in your broker environment.

"Commands" on page 3672
All WebSphere Message Broker Toolkit and runtime commands that are provided on distributed systems are listed, grouped by function, with references to command details.

"Responses to commands"
Responses are issued to the commands as messages. Messages that are returned to several commands are listed here; messages that are specific to a command, or have a specific meaning, are listed with the command.

Responses to commands

Responses are issued to the commands as messages. Messages that are returned to several commands are listed here; messages that are specific to a command, or have a specific meaning, are listed with the command.

If a command is successful, it returns a return code of zero, and a message with the number BIP8071I (command successful).

Warning and error responses are listed in the command descriptions. If the command is unsuccessful and returns, for example, the message BIP8083, it has an exit code, in this case, of 83.

The following responses are returned by many commands, and are not listed with each individual command:

- BIP8001 Unknown flag selected
- BIP8002 Selected flags are not valid
- BIP8003 Duplicate flag
- BIP8004 Invalid flags or arguments specified
- BIP8005 Flag or argument missing
- BIP8006 Mandatory flag missing
- BIP8007 Mandatory argument missing
- BIP8009 Program name not valid
- BIP8010 Component name not supplied
- BIP8013 Component does not exist
- BIP8071 Successful command completion
- BIP8073 Not a valid component
- BIP8081 Error generated by command processing
- BIP8083 Invalid component name
- BIP8087 Component exists and cannot be created

- BIP8092 Cannot locate return code in message catalog
- BIP8093 and BIP8094 Unable to access the WebSphere MQ queue manager
- BIP8099 The supplied parameters are not valid

Related reference:

"Characters allowed in commands" on page 3680
You must adhere to a few rules when you provide names or identifiers for the components and resources in your broker environment.

"Commands" on page 3672
All WebSphere Message Broker Toolkit and runtime commands that are provided on distributed systems are listed, grouped by function, with references to command details.

"Rules for using commands" on page 3681
Observe the following rules when using the WebSphere Message Broker commands on distributed systems.

Runtime and WebSphere Message Broker Toolkit commands (common)

Some commands are common to both runtime and WebSphere Message Broker Toolkit environments.

The following commands are available on workstations that have either the runtime, the WebSphere Message Broker Toolkit, or both components installed:

- "**mqsiapplybaroverride** command" on page 3684
- "**mqsireadbar** command" on page 3697

If either of these commands is run on a workstation that has both the runtime and WebSphere Message Broker Toolkit components installed, the version of the command that is used is determined by the relative locations of the WebSphere Message Broker Toolkit and runtime directories in the PATH variable of the workstation. The directory that appears earlier in the PATH takes precedence.

In most cases the WebSphere Message Broker Toolkit and runtime versions of the command are identical. However, if different levels of service pack have been applied to the WebSphere Message Broker Toolkit and runtime components, the updated commands might differ. For this reason, when either command is run, the first line of the output describes which version of the command is being used. For example:

```
BIP1138I: Overriding BAR File using runtime mqsiapplybaroverride.
```

Related concepts:

"WebSphere Message Broker Toolkit" on page 31
The WebSphere Message Broker Toolkit is an integrated development environment and graphical user interface based on the Eclipse platform.

Related reference:

"Commands" on page 3672
All WebSphere Message Broker Toolkit and runtime commands that are provided on distributed systems are listed, grouped by function, with references to command details.

"Characters allowed in commands" on page 3680
You must adhere to a few rules when you provide names or identifiers for the components and resources in your broker environment.

"Rules for using commands" on page 3681
Observe the following rules when using the WebSphere Message Broker commands

on distributed systems.

"Syntax diagrams" on page 3677

"Runtime commands" on page 3715
The topics in this section describe the WebSphere Message Broker runtime commands.

"WebSphere Message Broker Toolkit commands" on page 3699
Several commands are provided as part of the WebSphere Message Broker Toolkit.

mqsiapplybaroverride command:

Use the **mqsiapplybaroverride** command to replace configurable values in the broker archive (BAR) deployment descriptor with new values that you specify in a properties file.

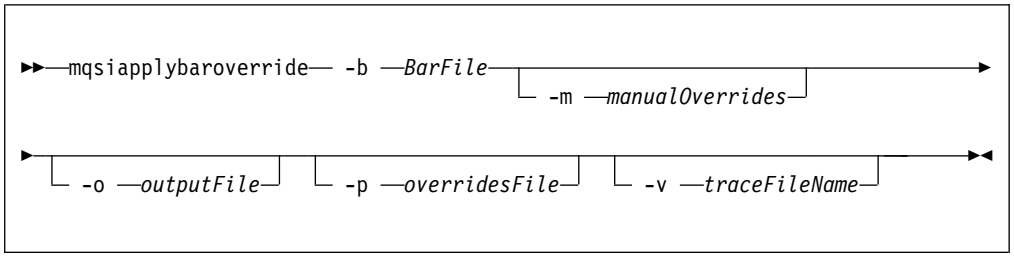
Supported operating systems:

- Windows
- Linux and UNIX systems
- z/OS. Run this command by customizing and submitting BIPOBAR.

Purpose:
Use the **mqsireadbar** command to see which properties you can configure by using the **mqsiapplybaroverride** command.

Write scripts to create BAR files and apply different override values in the broker deployment descriptor archive file by using the **mqsiapplybaroverride** command, together with the **mqsicreatebar** command. For a list of message flow node properties and the corresponding properties of the **mqsiapplybaroverride** command, see "Configurable properties" on page 3687. The **mqsiapplybaroverride** command properties are also listed in the tables of properties in the reference topic for each affected node.

Syntax:



Parameters:

-b *BarFile*
(Required) The path to the BAR file (in compressed format) to which the override values apply. The path can be absolute or relative to the executable command.

-m *manualOverrides*
(Optional) A list of the `property-name=override` pairs, `current-property-value=override` pairs, or a combination of them, to be applied to the BAR file. The pairs in the list are separated by commas (,). On Windows, you must enclose the list in quotation marks (" "). If used with the overridesFile (**-p**)

parameter, overrides specified by the manualOverrides (**–m**) parameter are performed after any overrides specified the **–p** parameter have been made.

-o *outputFile*
(Optional) The name of the output BAR file to which the BAR file changes are to be made. If an output file is not specified, the input file is overwritten.

-p *overridesFile*
(Optional) The path to one of the following resources:

- A BAR file that contains the deployment descriptor that is used to apply overrides to the BAR file.

  You can use XML for the deployment descriptor file. If you use an XML property file written in EBCDIC onz/OS, ensure that you remove any `encoding="UTF-8"` from the XML header; for example, change:

  ```
  <?xml version="1.0" encoding="UTF-8"?>
  ```

  to

  ```
  <?xml version="1.0"?>
  ```

- A properties file in which each line contains a `property-name=override` or `current-property-value=new-property-value` pair.
- A deployment descriptor that is used to apply overrides to the BAR file.

-v *traceFileName*
(Optional) Specifies that the internal trace is to be sent to the named file.

In all cases, any existing deployment descriptor in the BAR file is renamed to `META-INF\broker.xml.old`, replacing any existing file of that name.

Each override that is specified in a **–p** overrides file or a **–m** overrides list must conform to one of the following syntaxes:

- `FlowName#NodeName.PropertyName=NewPropertyValue` (or `FlowName#PropertyName=NewPropertyValue` for message flow properties) where:
  - *FlowName* is the name of the message flow without the `.cmf` extension (for example, Flow1).
  - *NodeName* is the optional name of the node whose property is overridden (for example, InputNode).
  - *PropertyName* is the name of the property to be overridden (for example, queueName).
  - *NewPropertyValue* is the value to assign to that property (for example, PRODUCTION_QUEUE_NAME).
- `OldPropertyValue=NewPropertyValue`. This syntax does a global search and replace on the property value `OldPropertyValue`. It overrides the value fields of `OldPropertyValue` in the deployment descriptor with `NewPropertyValue`.
- `FlowName#NodeName.PropertyName` (or `FlowName#PropertyName` for message flow properties). This syntax removes any override that is applied to the property of the supplied name.

Variables and comments are not allowed in property files.

When the **mqsiapplybaroverride** command runs, it displays the version of the command that is being used (either runtime environment or WebSphere Message Broker Toolkit) before it does anything else. For example:

```
BIP1138I: Overriding BAR File using runtime mqsiapplybaroverride
```

Authorization:
On Windows 7, Windows Vista, and Windows Server 2008 systems, the user ID used to run this command must be running with elevated privileges on the local system:

- The user ID must be a member of the group Administrators.
- The command must be started from an environment that has **Run as Administrator** authority.

If you do not run the command from a privileged environment, you are asked to confirm that you want to continue. When you click **OK**, a new privileged command console is created and the command completed, but all responses are written to the privileged environment and are lost when that console closes when the command completes.

On all operating systems, the user ID used to invoke this command must have write authority to the BAR file on the local system.

Responses:
This command returns the following responses:

0 The command completed successfully.

99 One or more of the parameters that you specified is invalid.

Windows This command is supplied as a batch file. If you run the command in an automation system or from a script, use the Windows CALL command, to ensure that the correct ERRORLEVEL is returned:

```
...
CALL mqsiapplybaroverride
...
```

Examples:
Open the BAR file `myflow.bar`, and replace configurable values in its deployment descriptor (typically `broker.xml`) with the values that are specified in the properties file `mychanges.properties`:

```
mqsiapplybaroverride -b myflow.bar -p mychanges.properties
```

Override the deployment descriptor in `c:\test.bar` by using the key=value pairs specified in `c:\my.properties`:

```
mqsiapplybaroverride -b c:\test.bar -p c:\my.properties
```

Override the deployment descriptor in `c:\test.bar` by using the deployment descriptor contained in `c:\previous.bar`:

```
mqsiapplybaroverride –b c:\test.bar –p c:\previous.bar
```

Override the deployment descriptor in `c:\test.bar` by using the deployment descriptor contained in `c:\broker.xml`:

```
mqsiapplybaroverride –b c:\test.bar –p c:\broker.xml
```

Override any properties with values set to OLDA and OLDB in `c:\test.bar` with the values NEWA and NEWB:

```
mqsiapplybaroverride –b c:\test.bar –m OLDA=NEWA,OLDB=NEWB
```

Override the value of the property name **sampleFlow#MQInput.queueName** to NEWC:

```
mqsiapplybaroverride –b c:\test.bar –m sampleFlow#MQInput.queueName=NEWC
```

For an example of the details that are contained in a properties file, see "Editing configurable properties" on page 3227.

Related concepts:

"Broker archive" on page 3216
The unit of deployment to the broker is the *broker archive* or *BAR* file.

"Configurable properties of a broker archive" on page 3217
System objects that are defined in message flows can have properties that you can update in the broker archive (BAR) file before deployment.

Related tasks:

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

Chapter 11, "Packaging and deploying," on page 3209
Package resources that you create in the WebSphere Message Broker Toolkit, such as message flows, and deploy them to execution groups on brokers.

Related reference:

"**mqsicreatebar** command" on page 3699
Use the **mqsicreatebar** command to create deployable broker archive (BAR) files containing message flows and dictionaries.

"**mqsireadbar** command" on page 3697
Use the **mqsireadbar** command to read a deployable BAR file and identify its defined keywords.

"Syntax diagrams" on page 3677

Configurable properties:

Some properties of message flow nodes are configurable and can be changed by using the **mqsiapplybaroverride** command. The following table maps the message flow node properties to the corresponding properties of the **mqsiapplybaroverride** command.

The **mqsiapplybaroverride** command properties are also listed in the tables of properties in the reference topic for each affected node.

Node	Configurable node property	mqsiapplybaroverride command property
AggregateControl	Aggregate Name	aggregateName
AggregateReply	Aggregate Name	aggregateName

Node	Configurable node property	**mqsiapplybaroverride command property**
CDInput	Message Coded Character Set ID	messageCodedCharSetIDProperty
	Message Encoding	messageEncodingProperty
	Retry threshold	retryThreshold
	Short retry interval	shortRetryInterval
	Long retry interval	longRetryInterval
	Validate	validateMaster
	Instances	instances
	Instances pool	instancesPool
	Configurable service	configurableService
	Directory filter	inputDirectory
	File name filter	filenamePattern
	Action on successful processing	processedFileAction
	Action on failing file	failedFileAction
CDOutput	Process name	process Name
	Configurable service	configurableService
	SNODE	snode
	Destination directory	destinationDirectory
	Destination file name	destinationFileName
	Disposition	disposition
	Transfer mode	transferMode
	Validate	validateMaster
CICSRequest	CICS server	cicsServer
	Mirror transaction ID	mirrorTran
	Request timeout (sec)	requestTimeoutSecs
	Security identity	securityIdentity
	Security profile	securityProfileName
	Set EIBTRNID only	eibtrnidOnly
	Validate	validateMaster
Collector	Collector Expiry	collectionExpiry
Compute	Data Source	dataSource
	Validate	validateMaster
CORBARequest	Naming service	namingService
	Object reference name	referenceName
Database	Data Source	dataSource

Node	Configurable node property	**mqsiapplybaroverride command property**
DatabaseInput	Additional instances	additionalInstances
	Additional instances pool	componentLevel
	Data Source	dataSource
	Long retry interval	longRetryInterval
	Polling interval (sec)	waitInterval
	Retry threshold	retryThreshold
	Short retry interval	shortRetryInterval
DataDelete	Data source	dataSource
DataInsert	Data source	dataSource
DataUpdate	Data source	dataSource
DatabaseRetrieve	Data source	dataSource
DatabaseRoute	Data source	dataSource
EmailInput	Action on failing email	emailFailureAction
	Additional instances	additionalInstances
	Additional instances pool	componentLevel
	Email server	emailServer
	Polling interval (sec)	waitInterval
	Long retry interval (sec)	longRetryInterval
	Retry threshold	retryThreshold
	Security identity	securityIdentity
	Short retry interval (sec)	shortRetryInterval
EmailOutput	SMTP Server and Port	smtpServer
	Security Identity	securityIdentity
	Validate	validateMaster
EndpointLookup	Port Type Name	name
	Port Type Namespace	namespace
	Port Type Version	portVersion
Extract (deprecated)	Data Source	dataSource

Node	Configurable node property	**mqsiapplybaroverride** command property
FileInput	Additional Instances	additionalInstances
	Additional Instances Pool	componentLevel
	File Name or Pattern	filenamePattern
	Input Directory	inputDirectory
	Long Retry Interval	longRetryInterval
	Message CCSID	messageCodedCharSetIdProperty
	Message Encoding	messageEncodingProperty
	Polling Interval (sec)	waitInterval
	Remote Transfer	fileFtp
	Retry Threshold	retryThreshold
	Security Identity	fileFtpUser
	Server and Port	fileFtpServer
	Server Directory	fileFtpDirectory
	Short Retry Interval	shortRetryInterval
	Transfer Protocol	remoteTransferType
	Validate	validateMaster
FileOutput	Directory	outputDirectory
	File Name or Pattern	outputFilename
	Remote Transfer	fileFtp
	Req Dir Property Location	requestDirectoryLocation
	Req File Name Prop Location	requestNameLocation
	Security Identity	fileFtpUser
	Server and Port	fileFtpServer
	Server Directory	fileFtpDirectory
	Transfer Protocol	remoteTransferType
	Validate	validateMaster
FileRead	Custom Delimiter	customDelimiter
	File Name or Pattern	filenamePattern
	Input Directory	inputDirectory
	Message Coded Character Set ID	messageCodedCharSetIdProperty
	Message Encoding	messageEncodingProperty
	Validate	validateMaster
Filter	Data Source	dataSource

Node	Configurable node property	**mqsiapplybaroverride command property**
FTEInput	Additional instances	additionalInstances
	Additional instances pool	componentLevel
	Directory filter	inputDirectory
	File name filter	filenamePattern
	Long retry interval (sec)	longRetryInterval
	Message coded character set ID	messageCodedCharSetIdProperty
	Message encoding	messageEncodingProperty
	Retry threshold	retryThreshold
	Short retry interval (sec)	shortRetryInterval
	Validate	validateMaster
FTEOutput	Destination agent	destinationAgent
	Destination file directory	destinationDirectory
	Destination file name	destinationFileName
	Destination queue manager	destinationQMgr
	Disable computation of MD5 check sum	checkSumDisabled
	Job name	jobName
	Mode	transferMode
	Overwrite files on destination system	overwriteDestination
	Validate	validateMaster
HTTPInput	Fault Format	faultFormat
	Path Suffix for URL	URLSpecifier
	Security Profile	securityProfileName
	Use HTTPS	useHTTPS
	Validate	validateMaster
HTTPReply	Validate	validateMaster
HTTPRequest	Allow SSL Ciphers	allowedCiphers
	Enable HTTP/1.1 keep-alive	enableKeepAlive
	HTTP Version	httpVersion
	HTTP(S) Proxy Location	httpProxyLocation
	Protocol	protocol
	Request Timeout (sec)	timeoutForServer
	Security Profile	securityProfileName
	Validate	validateMaster
	Web Service URL	URLSpecifier

Node	Configurable node property	**mqsiapplybaroverride command property**
IMSRequest	Configurable service	configurableService
	Data store name	dataStoreName
	Hostname	hostname
	Message CCSID	messageCodedCharSetIdProperty
	Message encoding	messageEncodingProperty
	Port number	portNumber
	Security identity	securityIdentity
	Security profile	securityProfileName
	Validate	validateMaster
JavaCompute	Validate	validateMaster
JDEdwardsRequest	Default method	defaultMethod
	Secondary adapter mode	secondaryAdapterMode
JMSHeader	JMS Reply-to	jmsReplyTo
JMSInput	Backout Destination	backoutDestination
	Connection Factory Name	connectionFactoryName
	Durable Subscription ID	durableSubscriptionID
	Initial Context Factory	initialContextFactory
	Location JNDI bindings	locationJndiBindings
	Source Queue	sourceQueueName
	Subscription Topic	topic
	Validate	validateMaster
JMSOutput	Connection Factory Name	connectionFactoryName
	Destination Queue	destinationQueueName
	Initial Context Factory	initialContextFactory
	Location JNDI Bindings	locationJndiBindings
	Message Type	messageType
	Publication Topic	topic
	Reply-to Destination	replyToDestination
	Send to Destination List	useDistList
	Validate	validateMaster
JMSReply	Connection Factory Name	connectionFactoryName
	Initial Context Factory	initialContextFactory
	Location JNDI Bindings	lcationJndiBindings
	Send to Destination List	useDistList
	Validate	validateMaster
Mapping	Data Source	dataSource
MQGet	Queue Name	queueName
	Validate	validateMaster

Node	Configurable node property	**mqsiapplybaroverride command property**
MQHeader	Destination QMgr Name	mqdlhDestQMgrName
	Destination Queue Name	mqdlhDestQName
	Reply-to Queue	mqmdReplyToQ
	Reply-to Queue Manager	mqmdReplyToQMgr
MQInput	Additional Instances	additionalInstances
	Additional Instances Pool	componentLevel
	Queue Name	queueName
	Reset Browse Timeout	resetBrowseTimeout
	Security Profile	securityProfileName
	Topic	topicProperty
	Validate	validateMaster
	z/OS serialization token	serializationToken
MQOptimizedFlow	Queue Name	queueName
MQOutput	Queue Manager Name	queueManagerName
	Queue Name	queueName
	Reply-to Queue	replyToQ
	Reply-to Queue Manager	replyToQMgr
	Security Profile	securityProfileName
	Validate	validateMaster
MQReply	Validate	validateMaster
PeopleSoftInput	Adapter Component	adapterComponent
	Secondary adapter mode	secondaryAdapterMode
	Additional Instances Pool	componentLevel
	Additional Instances	additionalInstances
	Retry Threshold	retryThreshold
	Short retry interval	shortRetryInterval
	Long retry interval	longRetryInterval
PeopleSoftRequest	Default Method	defaultMethod
	Secondary adapter mode	secondaryAdapterMode
PHPCompute	PHP Script	ScriptName
	Validate	validateMaster
RegistryLookup	Name	name
	Namespace	namespace
	Template	template
	Version	serviceVersion
Resequence	Queue prefix	queuePrefix
	Missing message timeout	missingMessageTimeoutSeconds
	Start of sequence definition	startSequenceSeconds
	End of sequence definition	endSequenceSeconds
ResetContentDescriptor	Validate	validateMaster

Node	Configurable node property	**mqsiapplybaroverride** command property
Route	Distribution Mode	distributionMode
SAPInput	Adapter Component	adapterComponent
	Secondary adapter mode	secondaryAdapterMode
	Additional Instances Pool	componentLevel
	Additional Instances	additionalInstances
	Retry Threshold	retryThreshold
	Short retry interval	shortRetryInterval
	Long retry interval	longRetryInterval
SAPRequest	Default method	defaultMethod
	Secondary adapter mode	secondaryAdapterMode
	Security profile	securityProfileName
SecurityPEP	Security Profile	securityProfileName
SCAAsyncResponse	Security Profile	securityProfileName
SCAInput	Security Profile	securityProfileName
SiebelInput	Adapter Component	adapterComponent
	Secondary adapter mode	secondaryAdapterMode
	Additional Instances Pool	componentLevel
	Additional Instances	additionalInstances
	Retry Threshold	retryThreshold
	Short retry interval	shortRetryInterval
	Long retry interval	longRetryInterval
SiebelRequest	Default Method	defaultMethod
	Secondary adapter mode	secondaryAdapterMode
SOAPInput	Additional Instances	additionalInstances
	Failure Action	validateFailureAction
	Max Client Wait Time (sec)	maxClientWaitTime
	Path Suffix for URL	urlSelector
	Policy Set	policySet
	Policy Set Bindings	policySetBindings
	Rte inb processing fails	sendProcessingFaultsToFailure
	Security Profile	securityProfileName
	Use HTTPS	useHTTPS
	Validate	validateMaster
SOAPReply	Allow MTOM	allowMTOM
	Validate	validateMaster

Node	Configurable node property	**mqsiapplybaroverride command property**
SOAPRequest	Allow MTOM	allowMTOM
	Allow SSL Ciphers	allowedSSLCiphers
	Failure Action	validateFailureAction
	HTTP(S) Proxy Location	httpProxyLocation
	Policy Set	policySet
	Policy Set Bindings	policySetBindings
	Protocol (if using SSL)	sslProtocol
	Security Profile	securityProfileName
	Validate	validateMaster
	Web Service URL	webServiceURL
SOAPAsyncRequest	Allow MTOM	allowMTOM
	Allow SSL Ciphers	allowedSSLCiphers
	HTTP(S) Proxy Location	httpProxyLocation
	Policy Set	policySet
	Policy Set Bindings	policySet Bindings
	Request Timeout	requestTimeout
	Protocol (if using SSL)	sslProtocol
	Security Profile	securityProfileName
	Unique Identifier	asyncResponseCorrelator
	Web Service URL	webServiceURL
SOAPAsyncResponse	Additional Instances	additionalInstances
	Additional Instances Pool	componentLevel
	Failure Action	validateFailureAction
	Unique Identifier	asyncRequestCorrelator
	Validate	validateMaster
TCPIPClientInput	Additional Instances	additionalInstances
	Additional Instances Pool	componentLevel
	Connection Details	connectionDetails
	Long Retry Interval	longRetryInterval
	Message CCSID	messageCodedCharSetIdProperty
	Message Encoding	messageEncodingProperty
	Retry Threshold	retryThreshold
	Short Retry Interval	shortRetryInterval
	Timeout Waiting for Data	timeoutWaitingForData
	Validate	validateMaster
TCPIPClientOutput	Connection Details	connectionDetails
	Timeout Sending Data Recs	timeoutSendingData
	Validate	validateMaster

Node	Configurable node property	**mqsiapplybaroverride** command property
TCPIPClientReceive	Connection Details	connectionDetails
	Message CCSID	messageCodedCharSetIdProperty
	Message Encoding	messageEncodingProperty
	Timeout Waiting for Data	timeoutWaitingForData
	Validate	validateMaster
TCPIPServerInput	Additional Instances	additionalInstances
	Additional Instances Pool	componentLevel
	Connection Details	connectionDetails
	Long Retry Interval	longRetryInterval
	Message CCSID	messageCodedCharSetIdProperty
	Message Encoding	messageEncodingProperty
	Retry Threshold	retryThreshold
	Short Retry Interval	shortRetryInterval
	Timeout Waiting for Data	timeoutWaitingForData
	Validate	validateMaster
TCPIPServerOutput	Connection Details	connectionDetails
	Timeout Sending Data Recs	timeoutSendingData
	Validate	validateMaster
TCPIPServerReceive	Connection Details	connectionDetails
	Message CCSID	messageCodedCharSetIdProperty
	Message Encoding	messageEncodingProperty
	Timeout Waiting for Data	timeoutWaitingForData
	Validate	validateMaster
TimeoutControl	Unique Identifier	uniqueIdentifier
TimeoutNotification	Unique Identifier	uniqueIdentifier
	Validation	validateMaster
	Timeout Interval	timeoutInterval
Trace	File Path	filePath
TwineballInput	Adapter Component	adapterComponent
	Secondary adapter mode	secondaryAdapterMode
	Additional Instances Pool	componentLevel
	Additional Instances	additionalInstances
	Retry Threshold	retryThreshold
	Short retry interval	shortRetryInterval
	Long retry interval	longRetryInterval
TwineballRequest	Default Method	defaultMethod
	Secondary adapter mode	secondaryAdapterMode
	Validate	validateMaster
Validate	Validate	validateMaster
Warehouse	Data Source	dataSource

Node	Configurable node property	**mqsiapplybaroverride** command property
XSLTransform	Stylesheet Directory	stylesheetPath
	Stylesheet Name	stylesheetName

Related concepts:

"Configurable properties of a broker archive" on page 3217
System objects that are defined in message flows can have properties that you can update in the broker archive (BAR) file before deployment.

Related tasks:

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

Related reference:

"**mqsiapplybaroverride** command" on page 3684
Use the **mqsiapplybaroverride** command to replace configurable values in the broker archive (BAR) deployment descriptor with new values that you specify in a properties file.

"Configurable message flow properties" on page 4020
When you add a message flow to a broker archive (BAR) file in preparation for deploying it to a broker, you can set additional properties that influence its run time operation. These properties are available for review and update when you select the **Manage and Configure** tab for the broker archive file.

mqsireadbar command:

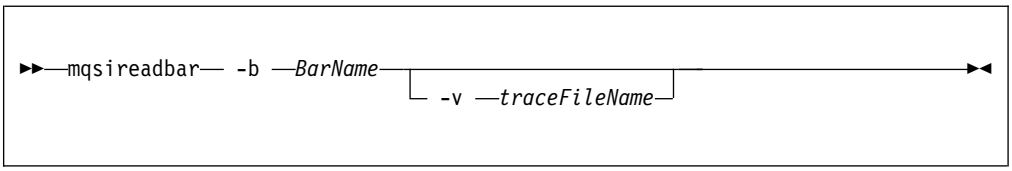
Use the **mqsireadbar** command to read a deployable BAR file and identify its defined keywords.

Supported platforms:

- Windows
- Linux and UNIX systems
- z/OS. Run this command by customizing and submitting BIPRBAR.

Purpose:
The **mqsireadbar** command returns the keywords defined for each deployable file in a deployable broker archive file.

Syntax:



Parameters:

-b *BarName*
(Required) The name of the BAR archive file to be read. The BAR file is in compressed format.

-v *traceFileName*
(Optional) The name of the file to which the command trace is sent. This option activates an internal debug trace; specify this option only at the request of an IBM service representative.

Authorization:
On Windows 7, Windows Vista, and Windows Server 2008 systems, the user ID used to run this command must be running with elevated privileges on the local system:

- The user ID must be a member of the group Administrators.
- The command must be started from an environment that has **Run as Administrator** authority.

If you do not run the command from a privileged environment, you are asked to confirm that you want to continue. When you click **OK**, a new privileged command console is created and the command completed, but all responses are written to the privileged environment and are lost when that console closes when the command completes.

On all platforms, the user ID that is used to start this command must have the authority to read the BAR file on the local system.

Responses:
This command returns the following responses:

0 The command completed successfully.

99 One or more of the parameters that you specified is invalid.

The command displays the version of the command that is being run (either WebSphere Message Broker Toolkit or runtime environment), before all other response data:

```
BIP1052I: Reading BAR File using runtime mqsireadbar
```

The command then displays a list of the deployable files, together with their keywords. For example:

```
C:\test.bar
     BAR Entry: simpleflow.cmf (07/10/07 10:43:44)
          Author = Matt
          VERSION = v1.1
          Information = This flow simply removes messages from IN.Q
```

It also displays deployment descriptors and the list of any properties in the BAR file that can be overridden, together with the new values of any overrides that are currently applied. For example:

```
Deployment descriptor:
    simpleflow#additionalInstances
    simpleflow#commitCount
    simpleflow#commitInterval
    simpleflow#coordinatedTransaction
    simpleflow#MQInput.topicProperty
    simpleflow#MQInput.validateMaster
    simpleflow#MQInput.queueName = OVERRIDDEN.Q
    simpleflow#MQInput.serializationToken
```

Windows This command is supplied as a batch file. If you run the command in an automation system or from a script, use the Windows CALL command to ensure that the correct ERRORLEVEL is returned:

```
...
CALL mqsireadbar
...
```

Examples:
The following example reads the file `my_bar_file.bar` and returns defined keywords within the given file:

```
mqsireadbar -b my_bar_file.bar
```

Related concepts:

"Broker archive" on page 3216
The unit of deployment to the broker is the *broker archive* or *BAR* file.

Related reference:

"**mqsicreatebar** command"
Use the **mqsicreatebar** command to create deployable broker archive (BAR) files containing message flows and dictionaries.

"Syntax diagrams" on page 3677

WebSphere Message Broker Toolkit commands

Several commands are provided as part of the WebSphere Message Broker Toolkit.

These commands are available only on a computer that has the WebSphere Message Broker Toolkit installed.

See "Commands" on page 3672 for a list of all the WebSphere Message Broker commands.

Related concepts:

"WebSphere Message Broker Toolkit" on page 31
The WebSphere Message Broker Toolkit is an integrated development environment and graphical user interface based on the Eclipse platform.

Related reference:

"Commands" on page 3672
All WebSphere Message Broker Toolkit and runtime commands that are provided on distributed systems are listed, grouped by function, with references to command details.

"Syntax diagrams" on page 3677

mqsicreatebar command:

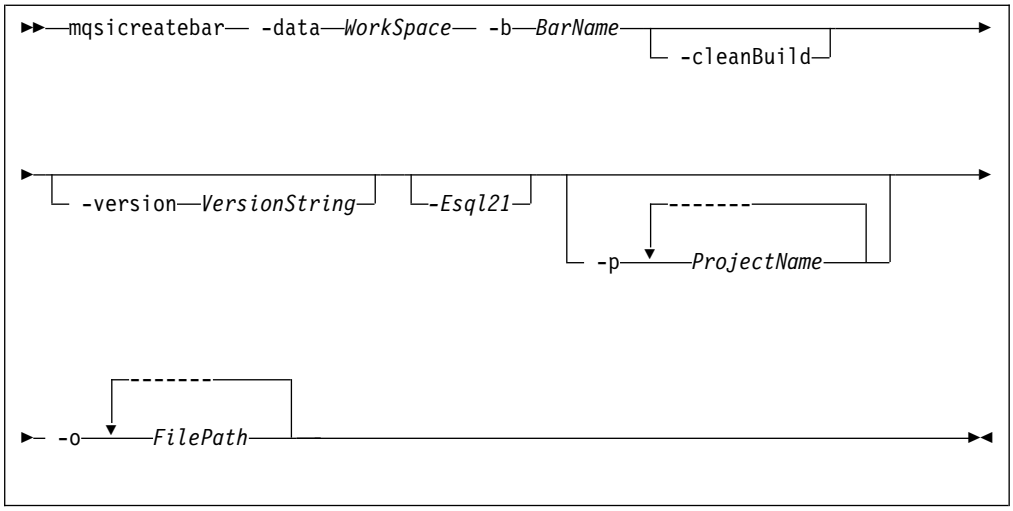
Use the **mqsicreatebar** command to create deployable broker archive (BAR) files containing message flows and dictionaries.

Supported platforms:

- Windows
- Linux on x86

Purpose:
If you use a repository to store your message flows and dictionaries, you can write scripts that use the **mqsicreatebar** command and the repository's command-line tools to deploy your message flow applications.

Syntax:



Parameters:

-data *WorkSpace*
(Required) The path of the workspace in which your projects are created.

The workspace location is also the default location for projects. Relative paths are interpreted as being relative to the directory from which the command was started.

-b *BarName*
(Required) The name of the BAR (compressed file format) archive file where the result is stored. The BAR file is replaced if it already exists and the `META-INF/broker.xml` file is created.

-cleanBuild
(Optional) Refreshes the projects in the workspace and then invokes a clean build before new items are added to the BAR file.

Use the **-cleanBuild** parameter to refresh all the projects in the BAR file and run a clean build if amendments have been made to BAR file resources by using external tools.

-version *VersionString*
(Optional) Appends the _ (underscore) character and the value of *VersionString* to the names of the compiled versions of the message flows (`.cmf`) files added to the BAR file, before the file extension.

-Esql21
(Optional) Compile ESQL for brokers at Version 2.1 of the product.

-p *ProjectName*
(Optional) Projects containing files to include in the BAR file in a new workspace. A new workspace is a system folder without the `.metadata` folder.

The projects defined must already exist in the folder defined in the -data parameter, and must include all projects and their reference projects that a deployable resource, defined in the -o parameter, needs.

The -p parameter is optional with an existing workspace, but you should use -p, together with a new workspace, in a build environment.

If a project that you specify is part of your workspace but is currently closed, the command opens and builds the project so that the files in the project can be included in the BAR file.

-o *FilePath*
(Required) The workspace relative path (including the project) of a deployable file to add to the BAR file; for example, a `msgflow` or `messageSet.mset` file.

You can add more than one deployable file to this command by using the following format: `-o FilePath1 FilePath2 .... FilePath'n'`

Authorization:
On Windows XP and Windows Server 2003 systems, the user ID used to run this command must be a member of the group Administrators on the local system.

On Windows 7, Windows Vista, and Windows Server 2008 systems, the user ID used to run this command must be running with elevated privileges on the local system:

- The user ID must be a member of the group Administrators.
- The command must be started from an environment that has **Run as Administrator** authority.

If you do not run the command from a privileged environment, you are asked to confirm that you want to continue. When you click **OK**, a new privileged command console is created and the command completed, but all responses are written to the privileged environment and are lost when that console closes when the command completes.

On Linux on x86, the user ID must have write access to the **-data** (workspace) and **-b** (BAR file location) directories.

Responses:
This command returns the following responses:

- BIP0956 Unable to start **mqsicreatebar**
- BIP0957 Incorrect arguments supplied to **mqsicreatebar**
- BIP0958 Nothing to do in **mqsicreatebar**
- BIP0959 Incorrect arguments supplied to **mqsicreatebar** (Project name)
- BIP0960 Incorrect arguments supplied to **mqsicreatebar** (Project directory)
- BIP0961 Error opening workspace in **mqsicreatebar** (Project could not be created)
- BIP0962 Error opening workspace in **mqsicreatebar** (Project could not be opened)
- BIP0963 Error saving file in **mqsicreatebar**
- BIP0964 Incorrect "-o" argument supplied to **mqsicreatebar**
- BIP0965 Error compiling files in **mqsicreatebar**

Examples:
You can run the **mqsicreatebar** command from a window opened anywhere in the file structure as long as the PATH environment variable specifies the correct path to the `WMB toolkit` directory.

- On Windows 32-bit editions, the default location of the `WMB toolkit` directory is `C:\Program Files\IBM\WMBT700`
- On Windows 64-bit editions, the default is `C:\Program Files (x86)\IBM\WMBT700`, and
- On Linux on x86, the default is `/opt/IBM/WMBT700`.

Your installation might differ from the default.

The -b parameter specifies the name of the BAR file and an optional alternative path for BAR file generation. When a path is not specified as part of the -b parameter, the location in the file structure where the **mqsicreatebar** command runs specifies where the BAR file is created. For example, if you are currently in C:\> and run the **mqsicreatebar** command, the BAR file is created on C:\. However, if you run the **mqsicreatebar** with -b c:\myfiles\myflow.bar specified, the BAR file is created in the myfiles directory.

The following example creates a BAR file called myflow.bar in the directory where the command is issued. The Test.msgflow message flow from the TestFlowProject is added to the BAR file as a compiled message flow (.cmf) file.

```
mqsicreatebar -data C:\Workspace -b myflow.bar -p TestFlowProject -o TestFlowProject\TestFlow\Test.msgflow
```

The following example creates a BAR file called mySet.bar. The messageSet.mset message set from the TestSetProject is added to the BAR file.

```
mqsicreatebar -data C:\Workspace -b mySet.bar -o TestSetProject\TestSet\messageSet.mset
```

The following example creates a BAR file called mySet.bar. The messageSet.mset message set from the TestSetProject and Test.msgflow message flow from the TestFlowProject are added to the BAR file. The message flow is added as a compiled message flow (.cmf) file.

```
mqsicreatebar -data C:\Workspace -b mySet.bar -o TestFlowProject\TestFlow\Test.msgflow
TestSetProject\TestSet\messageSet.mset
```

Related tasks:

"Creating a broker" on page 611
You can create brokers on every platform that is supported by WebSphere Message Broker. The broker runs as a 64-bit application on all platforms except Linux on x86 and Windows on x86.

Related reference:

"Syntax diagrams" on page 3677

"**mqsiapplybaroverride** command" on page 3684
Use the **mqsiapplybaroverride** command to replace configurable values in the broker archive (BAR) deployment descriptor with new values that you specify in a properties file.

mqsicreatemsgdefs command:

Use the **mqsicreatemsgdefs** command to create message definition files.

Supported platforms:

- Windows
- Linux on x86

Purpose:
The **mqsicreatemsgdefs** command generates message definition files (*.mxsd), according to a set of import options that are specified in an option file. The generated files are placed in the specified message set folder.

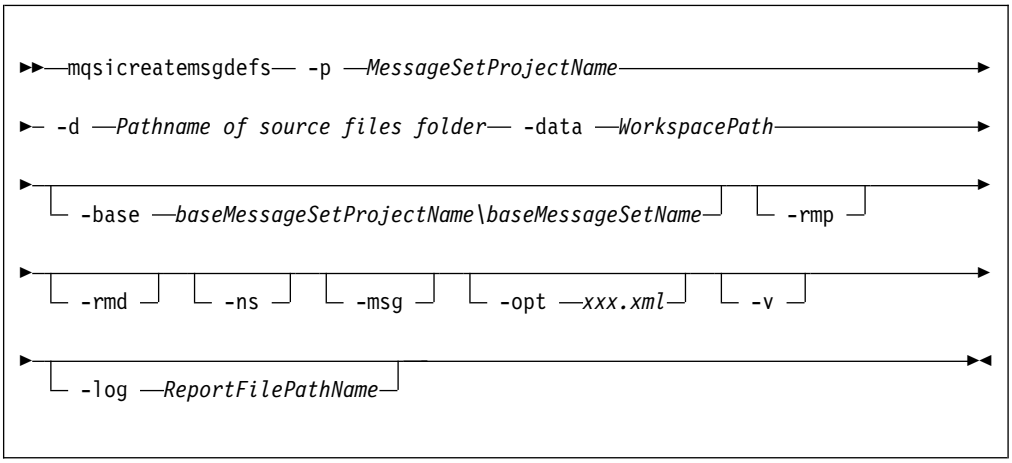
The command takes as a parameter a directory where source files of various types, for example, C and COBOL source files, are located (in addition to various other parameters), and starts the appropriate operation based on the extensions to the files.

1. Ensure that only the files that are required for the command to run exist in the directory and subdirectory structure that you specify. One of the actions that

the **mqsicreatemsgdefs** command performs is to copy all the files in the directory and subdirectories into the workspace, before creating the message definition. The **mqsicreatemsgdefs** command copies the source files into the import files folder of the messageSet project. Therefore the source directory should not be, or contain the message set project folder. Files that are not related to the message definitions that you are trying to create might also be copied.

2. You must specify the **-data** *WorkspacePath* parameter to specify the target workspace.

Syntax:



Parameters:

-p *MessageSetProjectName*
(Required) The name of the message set project. If the project does not exist, a new message set project is created.

-d *Pathname of source files folder*
(Required) The absolute or relative path name of the directory of definition files (source files).

All relevant files that are located in any sub-folders under the source files folder are scanned and imported.

-data *WorkspacePath*
(Required) The path of the workspace in which your projects are created.

The workspace location is also the default location for projects. Relative paths are interpreted as being relative to the directory from which the command was started.

-base *baseMessageSetProjectName\baseMessageSetName*
(Optional) If a new message set is to be created, specifies the existing message set project and message set, on which it is based.

-rmp
(Optional) Replaces the existing project of the same name.

-rmd
(Optional) Replaces an existing message definition file of the same name.

1. If you omit this flag, and a message definition file of the same name exists, a warning is returned.

2. The location of the generated message definition file in the message set is determined by the target namespace.

-ns
(Optional) If a new message set is to be created, the message set is enabled for namespace support.

-msg
(Optional) Creates messages from complex imported structures.

-opt *xxx.xml*
(Optional) The absolute or relative path name of the options file. The options file can be one of the following types:

- **C language** - "C options file for the **mqsicreatemsgdefs** command" on page 3705
- **COBOL language** - "COBOL options file for the **mqsicreatemsgdefs** command" on page 3707
- **XSD_NO_NS** - "XSD options file for the **mqsicreatemsgdefs** command" on page 3709

If you do not specify an option, the default options file (`mqsicreatemsgdefs.xml`) is used; see "Default options file for the **mqsicreatemsgdefs** command" on page 3710.

You can copy the default options file, and customize it, to create an options file for your environment.

-v
(Optional) Verbose report.

-log *ReportFilePathname*
(Optional) Absolute or relative path name of the report file. If you omit this option, the report is written to the default log file (`mqsicreatemsgdefs.report.txt`) in the Eclipse current directory.

If you specify *-log* without the report file path name, or with a path name that is not valid, the command issues an error message and stops.

Authorization:
On Windows 7, Windows Vista, and Windows Server 2008 systems, the user ID used to run this command must be running with elevated privileges on the local system:

- The user ID must be a member of the group Administrators.
- The command must be started from an environment that has **Run as Administrator** authority.

If you do not run the command from a privileged environment, you are asked to confirm that you want to continue. When you click **OK**, a new privileged command console is created and the command completed, but all responses are written to the privileged environment and are lost when that console closes when the command completes.

On other platforms, no specific authority is required to run this command.

Examples:
The following example creates or uses the message set project `newproject` in the source file `c:\myproject\source` and replaces the existing message project and message definition files of the same name.

```
mqsicreatemsgdefs -p newproject -d c:\myproject\source -rmp -rmd
```

Related concepts:

"Message definition files" on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

Related tasks:

"Creating a message definition file" on page 2865
Creating an empty message definition file to contain your message model objects.

Related reference:

"Syntax diagrams" on page 3677

C options file for the **mqsicreatemsgdefs** *command:*

Specify the options for the **mqsicreatemsgdefs** command when you import a C header file.

The following table lists the elements in the C language section of the options file. The following restrictions apply:

- You must specify, in an XML file, a valid value for the options listed, unless otherwise specified. See "Default options file for the **mqsicreatemsgdefs** command" on page 3710 for details of the syntax.
- Values of options are case-sensitive.
- If you do not specify the *-opt* parameter on the **mqsicreatemsgdefs** command, the default options file called `mqsicreatemsgdefs.xml` is used; see "Default options file for the **mqsicreatemsgdefs** command" on page 3710.

For further information on using these options, see "Importing from C" on page 2934.

<C> element	**Possible values**
COMPILER_NAME[1]	• Msvc (Default) • Icc • AIXgcc • AIXxlc • OS390
CODEPAGE	• SO8859-1 • Cp037 • Cp1252 (Default)
FLOATING_POINT_FORMAT	• IEEE Extended INTEL (Default) • IEEE Extended AIX • IEEE Extended OS/390® • IEEE Non-Extended • IBM 390 Hexadecimal
INCLUDE_PATH[2]	Absolute path names of other header files, or an empty string (this is the default value).
BYTE_ORDER	• Little Endian (Default) • Big Endian
ADDRESS_SIZE	• 32 (Default) • 64

<C> element	Possible values
SIZE_OF_LONG_DOUBLE	• 64 (Default) • 128 (Not supported)
PACK_LEVEL[1]	• 1 • 2 • 4 • 8 (Default) • 16
SIZE_OF_ENUM	• 1 • 2 • 4 • 5 (Default)
PRESERVE_CASE_IN_VARIABLE_NAMES	• True (Default) • False
STRING_ENCODING	• SPACE - Fixed-length strings (Default) • NULL - Null-terminated strings
STRING_PADDING_CHARACTER	• SPACE (Default) • NUL • 'c' • "c" • 0xYY • YY • U+xxxx
SCHEMA_TARGET_NAMESPACE_URI	A valid namespace URI, or empty (default)
MESSAGE_PREFIX[3]	A string with which to prefix created messages, or empty. Default is `msg_`.
PRE_PROCESSING_OPTION	• none (default) • ale_idoc • file_idoc

Notes:

1. If COMPILER_NAME is set to AIXxlc, the value of PACK_LEVEL is not used.
2. In INCLUDE_PATH, separate paths by the system-dependent path separator character.
3. MESSAGE_PREFIX is ignored if PRE_PROCESSING_OPTION is ale_idoc or file_idoc.

Related reference:

"**mqsicreatemsgdefs** command" on page 3702
Use the **mqsicreatemsgdefs** command to create message definition files.

"COBOL options file for the **mqsicreatemsgdefs** command" on page 3707
Specify the options for the **mqsicreatemsgdefs** command when you import a COBOL copybook.

"XSD options file for the **mqsicreatemsgdefs** command" on page 3709
Specify the options for the **mqsicreatemsgdefs** command when you import an XML Schema file.

"Default options file for the **mqsicreatemsgdefs** command" on page 3710
Options for the **mqsicreatemsgdefs** command take default values if you do not specify an options file.

COBOL options file for the **mqsicreatemsgdefs** *command:*

Specify the options for the **mqsicreatemsgdefs** command when you import a COBOL copybook.

The following table lists the elements in the COBOL language section of the options file. The following restrictions apply:

- You must specify in an XML file a valid value for the options listed, unless otherwise specified. See "Default options file for the **mqsicreatemsgdefs** command" on page 3710 for details of the syntax.
- Options values are case-sensitive.
- If you do not specify the *-opt* parameter on the **mqsicreatemsgdefs** command, the default options file (`mqsicreatemsgdefs.xml`) is used; see "Default options file for the **mqsicreatemsgdefs** command" on page 3710.

For further information about using these options, see "Importing from COBOL copybooks" on page 2937. For more information about COBOL parameters, see Compiler Options

<COBOL> element	**Possible values**
PLATFORM_SELECTION	Operating system that your enterprise uses: • 0 (Win32) (Default) • 1 (AIX) • 2 (z/OS)
CODEPAGE	Indicates the numeric identifier of the code page associated with a language. • ISO8859_1 (Default) • 037
FLOATING_POINT_FORMAT	Indicates how floating point numbers are encoded. • IEEE Non-Extended (Default) • IBM 390 Hexadecimal
ENDIAN	Indicates whether the most significant byte is first (big endian) or last (little endian). • Big • Little (Default)
EXT_DECIMAL_SIGN	Indicates the physical type of the sign numeric element. For more information, see "CWF properties for compound element decimal types" on page 6107. • ASCII (Default) • EBCDIC • EBCDIC Custom
TRUNC	Specifies how arithmetic fields are truncated. • STD (Default) • OPT • BIN

<COBOL> element	Possible values
NSYMBOL	Indicates whether to use the National or double-byte character sets. • DBCS • NATIONAL (Default)
QUOTE	Indicates whether single or double quotes are used. • SINGLE • DOUBLE (Default)
CREATE_DEFAULT_VALUES FROM_INITIAL_VALUES	Indicates whether initial values are used as defaults. • True • False (Default)
CREATE_FACETS_FROM LEVEL_88_VALUE_CLAUSES	Indicates whether to create XSD facets from level-88 values. • True • False (Default)
PRESERVE_CASE_IN VARIABLE_NAMES	Indicates whether upper- and lowercase are kept when using names of variables. • True (Default) • False
CREATE_NULL_VALUES_FOR_FIELDS	Indicates whether fields should be set up with null values. • True • False (Default)
NULL_CHARACTER	Indicates which character is to be used as the null character. • SPACE (Default) • NUL • 'c' • "c" • 0xYY • YY • U+xxxx
STRING_PADDING_CHARACTER	Indicates which character is to be used as the padding character for strings. • SPACE (Default) • NUL • 'c' • "c" • 0xYY • YY • U+xxxx
SCHEMA_TARGET_NAMESPACE_URI	A valid namespace URI, or empty (default)
MESSAGE_PREFIX	A string with which to prefix created messages, or empty. Default is `msg_`.

Related reference:

"**mqsicreatemsgdefs** command" on page 3702
Use the **mqsicreatemsgdefs** command to create message definition files.

"C options file for the **mqsicreatemsgdefs** command" on page 3705
Specify the options for the **mqsicreatemsgdefs** command when you import a C header file.

"XSD options file for the **mqsicreatemsgdefs** command"
Specify the options for the **mqsicreatemsgdefs** command when you import an XML Schema file.

"Default options file for the **mqsicreatemsgdefs** command" on page 3710
Options for the **mqsicreatemsgdefs** command take default values if you do not specify an options file.

XSD options file for the **mqsicreatemsgdefs** *command:*

Specify the options for the **mqsicreatemsgdefs** command when you import an XML Schema file.

The following tables list the elements in the XML Schema sections of the options file.

The first table applies to all message sets, but the second table applies only to message sets that do not support namespaces. The following restrictions apply:

- You must specify in an XML file a valid value for each of the options listed, unless otherwise specified.
- Options values are case-sensitive.
- If you do not specify the *-opt* parameter on the **mqsicreatemsgdefs** command, the default options file (`mqsicreatemsgdefs.xml`) is used; see "Default options file for the **mqsicreatemsgdefs** command" on page 3710.

For further information about using these options, see "Importing from XML Schema" on page 2957.

<XSD> element	**Possible values**
MSG	• elements (default) • types • both

<XSD_NO_NS> element	**Possible values**
IMPORT	• modify (default) • reject
REDEFINE	• modify (default) • reject • accept
LIST	• modify (default) • reject • accept
UNION	• modify (default) • reject • accept
ABSTRACT_CT	• modify (default) • reject • accept

<XSD_NO_NS> element	Possible values
ABSTRACT_ELEMENT	• modify (default) • reject • accept
XSD_PREFIX	• xsi (default) • <any other prefix>
URI_PREFIX_PAIRS **Note:** You can specify zero, or more URI_PREFIX_PAIRS elements.	Attribute pair • uri=<uri_value> • prefix=<prefix_value>

Related reference:

"**mqsicreatemsgdefs** command" on page 3702
Use the **mqsicreatemsgdefs** command to create message definition files.

"C options file for the **mqsicreatemsgdefs** command" on page 3705
Specify the options for the **mqsicreatemsgdefs** command when you import a C header file.

"COBOL options file for the **mqsicreatemsgdefs** command" on page 3707
Specify the options for the **mqsicreatemsgdefs** command when you import a COBOL copybook.

"Default options file for the **mqsicreatemsgdefs** command"
Options for the **mqsicreatemsgdefs** command take default values if you do not specify an options file.

Default options file for the **mqsicreatemsgdefs** *command:*

Options for the **mqsicreatemsgdefs** command take default values if you do not specify an options file.

The following text lists the supplied default options file used with the **mqsicreatemsgdefs** command.

If you want to make any changes to the default file contents, the file is stored in the package group directory structure at `%PACKAGE_GROUP_LOCATION%\plugins\com.ibm.etools.msg.importer.cmdline_%VERSION%\mqsicreatemsgdefs.xml`. For example, if you have installed into the default location on Windows XP, the file is stored at `C:\Program Files\IBM\SDP70Shared\plugins\com.ibm.etools.msg.importer.cmdline_`*`build_version`*`\mqsicreatemsgdefs.xml`. *build_version* is the exact build version of the installed component; for example, `6.1.100.200803031447`.

```
<?xml version="1.0" encoding="UTF-8"?>
<OPTIONS>
<!-- Message Definition File Import Options -->
<!-- Import Options for C -->
  <C>
    <!-- COMPILER_NAME = (Msvc|icc|AIXgcc|AIXxlc|OS390) -->
    <COMPILER_NAME>Msvc</COMPILER_NAME>
    <!-- CODEPAGE = (ISO8859-1|Cp037|Cp1252) -->
    <CODEPAGE>Cp1252</CODEPAGE>
    <!-- FLOATING_POINT_FORMAT = (IEEE Extended INTEL|
         IEEE Extended AIX|IEEE Extended OS/390|
         IEEE Non-Extended|IBM 390 Hexadecimal) -->
    <FLOATING_POINT_FORMAT>IEEE Extended INTEL</FLOATING_POINT_FORMAT>
    <!-- BYTE_ORDER = (Little Endian|Big Endian) -->
    <BYTE_ORDER>Little Endian</BYTE_ORDER>
    <!-- ADDRESS_SIZE = (32|64) -->
```

```
    <ADDRESS_SIZE>32</ADDRESS_SIZE>
    <!-- SIZE_OF_LONG_DOUBLE = (64|128) -->
    <!-- NOTE: 128 is not supported; therefore 64 is always the value -->
    <SIZE_OF_LONG_DOUBLE>64</SIZE_OF_LONG_DOUBLE>
    <!-- PACK_LEVEL = (1|2|4|8|16) -->
    <PACK_LEVEL>8</PACK_LEVEL>
    <!-- SIZE_OF_ENUM = (1|2|4|5) -->
    <SIZE_OF_ENUM>5</SIZE_OF_ENUM>
    <!-- STRING ENCODING = SPACE | NULL) -->
    <!-- NOTE: SPACE = Fixed length strings, NULL = Null terminated strings -->
    <STRING_ENCODING>SPACE</STRING_ENCODING>
    <!-- STRING PADDING CHARACTER  = (SPACE|NUL|'c'|"c"|0xYY|YY|U+xxxx)-->
    <!-- Note: Only used for Fixed Length strings -->
    <STRING_PADDING_CHARACTER>SPACE</STRING_PADDING_CHARACTER>
    <!-- PRESERVE_CASE_IN_VARIABLE_NAMES = (true|false) -->
    <PRESERVE_CASE_IN_VARIABLE_NAMES>true</PRESERVE_CASE_IN_VARIABLE_NAMES>
    <!-- INCLUDE_PATH = absolute paths to other include files -->
    <!--  Paths should be separated by the system-dependent path-separator character.
          On UNIX systems, this character is ':'; on Win32 systems it is ';' -->
    <INCLUDE_PATH />
    <!-- SCHEMA TARGET NAMESPACE URI = (... any valid namespace URI or empty) -->
    <SCHEMA_TARGET_NAMESPACE_URI/>
    <!-- MESSAGE_PREFIX = (msg_ ... any string including empty string) -->
    <MESSAGE_PREFIX>msg_</MESSAGE_PREFIX>
    <!-- PRE_PROCESSING_OPTION = (none|ale_idoc|file_idoc) -->
    <PRE_PROCESSING_OPTION>none</PRE_PROCESSING_OPTION>
  </C>
<!-- Import Options for COBOL -->
  <COBOL>
    <!-- PLATFORM_SELECTION = (0:"Win32"|1:"AIX"|2:"z/OS") -->
    <PLATFORM_SELECTION>Win32</PLATFORM_SELECTION>
    <!-- CODEPAGE = (ISO8859_1|037) -->
    <CODEPAGE>ISO8859_1</CODEPAGE>
    <!-- FLOATING_POINT_FORMAT = (IEEE Non-Extended|IBM 390 Hexadecimal) -->
    <FLOATING_POINT_FORMAT>IEEE Non-Extended</FLOATING_POINT_FORMAT>
    <!-- ENDIAN = (Big|Little) -->
    <ENDIAN>Little</ENDIAN>
    <!-- EXT_DECIMAL_SIGN = (ASCII|EBCDIC|EBCDIC Custom) -->
    <EXT_DECIMAL_SIGN>ASCII</EXT_DECIMAL_SIGN>
    <!-- TRUNC = (STD|OPT|BIN) -->
    <TRUNC>STD</TRUNC>
    <!-- NSYMBOL = (DBCS|NATIONAL) -->
    <NSYMBOL>NATIONAL</NSYMBOL>
    <!-- QUOTE = (SINGLE|DOUBLE) -->
    <QUOTE>DOUBLE</QUOTE>
    <!-- CREATE_DEFAULT_VALUES_FROM_INITIAL_VALUES = (true|false) -->
    <CREATE_DEFAULT_VALUES_FROM_INITIAL_VALUES>false</CREATE_DEFAULT_
           VALUES_FROM_INITIAL_VALUES>
    <!-- CREATE FACETS FROM LEVEL 88 VALUE CLAUSES = (true|false) -->
    <CREATE_FACETS_FROM_LEVEL_88_VALUE_CLAUSES>false</CREATE_FACETS_
           FROM_LEVEL_88_VALUE_CLAUSES>
    <!-- PRESERVE_CASE_IN_VARIABLE_NAMES = (true|false) -->
    <PRESERVE_CASE_IN_VARIABLE_NAMES>true</PRESERVE_CASE_IN_VARIABLE_NAMES>
    <!--  CREATE NULL VALUES FOR FIELDS = (true|false) -->
    <CREATE_NULL_VALUES_FOR_FIELDS>false</CREATE_NULL_VALUES_FOR_FIELDS>
    <!-- NULL CHARACTER  = (SPACE|NUL|'c'|"c"|0xYY|YY|U+xxxx)-->
    <NULL_CHARACTER>SPACE</NULL_CHARACTER>
    <!-- STRING PADDING CHARACTER  = (SPACE|NUL|'c'|"c"|0xYY|YY|U+xxxx)-->
    <!-- Note: Only used for Fixed Length strings -->
    <STRING_PADDING_CHARACTER>SPACE</STRING_PADDING_CHARACTER>
    <!-- SCHEMA TARGET NAMESPACE URI = (... any valid namespace URI or empty) -->
    <SCHEMA_TARGET_NAMESPACE_URI/>
    <!-- MESSAGE_PREFIX = (msg_ ... any string including empty string) -->
    <MESSAGE_PREFIX>msg_</MESSAGE_PREFIX>
  </COBOL>
<!-- Import Options for XML Schema in general -->
  <XSD>
```

```
    <!-- MSG = (elements|types|both) -->
    <!-- Create messages from imported complex global elements, -->
    <!-- or from imported global complex types, or both -->
    <MSG>elements</MSG>
  </XSD>
<!-- Import Options for XML Schema when importing into a message set
       that does NOT support namespaces -->
  <XSD_NO_NS>
    <!-- IMPORT = (modify|reject|accept) -->
    <IMPORT>modify</IMPORT>
    <!-- REDEFINE = (modify|reject|accept) -->
    <REDEFINE>modify</REDEFINE>
    <!-- LIST = (modify|reject|accept) -->
    <LIST>modify</LIST>
    <!-- UNION = (modify|reject|accept) -->
    <UNION>modify</UNION>
    <!-- ABSTRACT_CT = (modify|reject|accept) -->
    <ABSTRACT_CT>modify</ABSTRACT_CT>
    <!-- ABSTRACT_ELEMENT = (modify|reject|accept) -->
    <ABSTRACT_ELEMENT>modify</ABSTRACT_ELEMENT>
    <!-- XSD_PREFIX = (xsi|... any other prefix) -->
    <XSD_PREFIX>xsi</XSD_PREFIX>
    <!-- This is where you list the additional uri/prefix pairs. -->
    <!-- URI prefix pairs can be listed as follows:  -->
    <!-- <URI_PREFIX_PAIRS uri="http://www.ibm.com" prefix="ibm" /> -->
    <!-- <URI_PREFIX_PAIRS uri="http://www.eclipse.org" prefix="eclipse"/> -->
  </XSD_NO_NS>
</OPTIONS>
```

Related reference:

"**mqsicreatemsgdefs** command" on page 3702
Use the **mqsicreatemsgdefs** command to create message definition files.

"C options file for the **mqsicreatemsgdefs** command" on page 3705
Specify the options for the **mqsicreatemsgdefs** command when you import a C header file.

"COBOL options file for the **mqsicreatemsgdefs** command" on page 3707
Specify the options for the **mqsicreatemsgdefs** command when you import a COBOL copybook.

"XSD options file for the **mqsicreatemsgdefs** command" on page 3709
Specify the options for the **mqsicreatemsgdefs** command when you import an XML Schema file.

mqsicreatemsgdefsfromwsdl command:

Use the **mqsicreatemsgdefsfromwsdl** command to import a single WSDL definition.

Supported platforms:

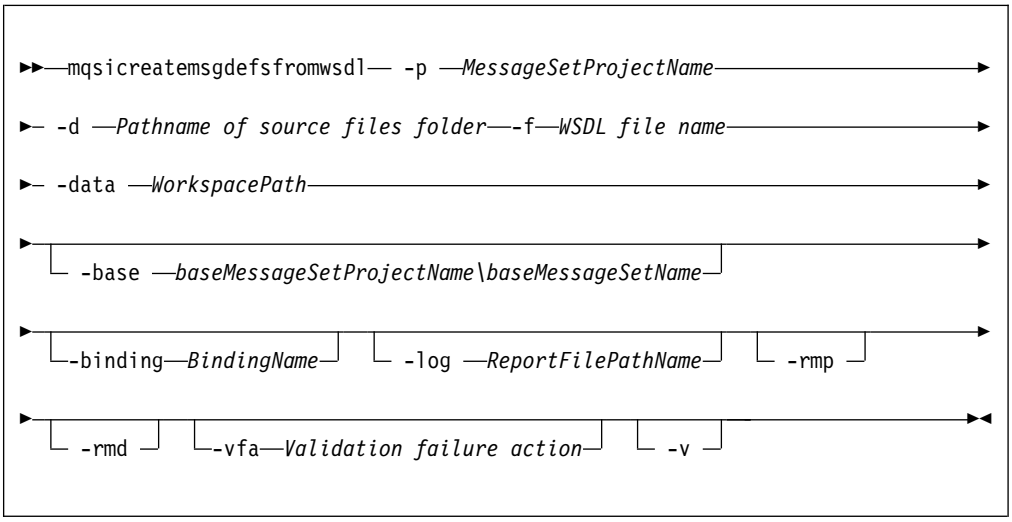
- Windows
- Linux on x86

Purpose:
If the WSDL is split into multiple files then the file specified must contain the WSDL service definition or binding definition. The WS-I validator can be run automatically on the imported WSDL under the control of the -vfa flag.

1. Ensure that only the files that are required for the WSDL definition you are importing exist in the directory and subdirectory structure. One of the actions the **mqsicreatemsgdefsfromwsdl** command performs is to copy all the files in the directory and subdirectories into the workspace prior to creating the message definition. Files that are not associated to that WSDL definition but exist in the directory are also copied.

2. If the WSDL definition uses a relative path that includes files outside of the directory or subdirectory structure specified, you must import these files into the workspace before you run the command. Make sure that the relative paths are still valid after importing these files into the workspace.
3. Message sets that are created are namespace enabled.
4. Existing message sets must be namespace enabled and have an XML physical format.
5. If you are creating a new message set for runtime parsing, you should base it on an existing message set which has an XML physical format.

Syntax:



Parameters:

-p *MessageSetProjectName*
(Required) The name of the message set project. If the project exists, it must be namespace-enabled. If the project does not exist, a new namespace-enabled project is created.

-base *baseMessageSetProjectName\baseMessageSetName*
(Optional) If a new message set is to be created, specify the existing message set project and message set on which it is based

-binding *BindingName*
(Optional) The name of a binding to be imported. This parameter is mandatory if the WSDL definition includes more than one binding, but optional if the WSDL definition includes a single binding

-d *Pathname of source files folder*
(Required) The absolute or relative path name of the directory where the top-level WSDL file is located. The top-level WSDL file can contain the entire WSDL definition, or it can be the top of a hierarchy of files, each of which can import further files via import elements. An import element specifies the location of the resource to import with a location attribute

The importer attempts to resolve all relative import locations relative to the specified directory; the importer also attempts to resolve any absolute import locations that it encounters. However, avoid using absolute import locations, because any further imports in the hierarchy must use absolute locations after the first time you specify an absolute location.

-data *WorkspacePath*
(Required) The path of the workspace in which your projects are created.

The workspace location is also the default location for projects. Relative paths are interpreted as being relative to the directory from which the command was started.

-f *<WSDL file name>*
(Required) The file name of the top-level WSDL file to be imported.

Where a path is required to fully identify the filename, the path should be specified using the **-d** parameter.

-log *ReportFilePathName*
(Optional) Absolute or relative path name of the report file; if omitted, the report is written to the default log file and is named *Wsdl-file-name*.wsdl.report.txt. *Wsdl-file-name* is the name of the WSDL definition you are importing and it is placed in the directory from which the command is invoked.

-rmd
(Optional) Replaces an existing message definition file of the same name.

Note:

1. If you omit this flag, and a message definition file of the same name exists, a warning is returned.
2. The location of the generated message definition file in the message set is determined by the target namespace.

-rmp
(Optional) Replaces the existing project of the same name.

-v
(Optional) Verbose report.

-vfa
(Optional) Validation failure action. Specifies the required action if WS-I compliance checking detects a problem in the WSDL to be imported. The default is set to fail. Select from:

- fail: If the WSDL definition is not WS-I compliant, the import process stops, and errors are written to the log file.
- warn: If the WSDL definition is not WS-I compliant, the import process writes warning errors to the log file.
- ignore: If the WSDL definition is not WS-I compliant, the import process ignores them and informational messages of how this WSDL definition is not compliant to the WS-I profile are written to the logfile.

Authorization:
On Windows 7, Windows Vista, and Windows Server 2008 systems, the user ID used to run this command must be running with elevated privileges on the local system:

- The user ID must be a member of the group Administrators.
- The command must be started from an environment that has **Run as Administrator** authority.

If you do not run the command from a privileged environment, you are asked to confirm that you want to continue. When you click **OK**, a new privileged

command console is created and the command completed, but all responses are written to the privileged environment and are lost when that console closes when the command completes.

On other platforms, no specific authority is required to run this command.

Examples:
In the following example, the WSDL document `service.wsdl` which exists in the directory `wsdlfiles`, is to be imported into the project myProject and overwrite the project if it exists.

```
mqsicreatemsgdefsfromwsdl -p myProject -d .\wsdlfiles -f service.wsdl -rmd -data .\wsdlfilewspc
```

In the following example, the WSDL document `service.wsdl` which exists in the directory `wsdlfiles`, is to be imported to create a new message set project (newProj) based on an existing project (existingProj).

```
mqsicreatemsgdefsfromwsdl -p newProj -base existingProj -d .\wsdlfiles -f service.wsdl -data .\wsdlfilewspc
```

Related concepts:

"Message definition files" on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

Related tasks:

"Creating a message definition file" on page 2865
Creating an empty message definition file to contain your message model objects.

"Importing from WSDL" on page 2946
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to create a new message definition from WSDL.

"Importing WSDL definitions from the command line" on page 2948
WSDL definitions can be imported using the (**mqsicreatemsgdefsfromwsdl**) command.

Runtime commands

The topics in this section describe the WebSphere Message Broker runtime commands.

These commands are available on all computers on which you have installed the broker.

See "Commands" on page 3672 for a list of all the WebSphere Message Broker commands.

Related reference:

"Commands" on page 3672
All WebSphere Message Broker Toolkit and runtime commands that are provided on distributed systems are listed, grouped by function, with references to command details.

"Syntax diagrams" on page 3677

mqsiaddbrokerinstance command:

Use the **mqsiaddbrokerinstance** command to create a multi-instance broker on a server where WebSphere Message Broker has been installed.

Supported platforms:

- Windows

- Linux and UNIX systems

Purpose:
Use the **mqsiaddbrokerinstance** command to add a broker instance to any additional server on which you require multi-instance support. You must first create a multi-instance enabled broker on one server by using the **mqsicreatebroker** command.

If you add a broker instance with the **mqsiaddbrokerinstance** command, you cannot use this command to start and stop a multi-instance broker as a WebSphere MQ service; you must use the "**mqsichangebroker** command" on page 3723 instead.

Select the appropriate link for details of this command on the platform, or platforms, that your enterprise uses:

- "**mqsiaddbrokerinstance** command - Linux and UNIX systems" on page 3717
- "**mqsiaddbrokerinstance** command - Windows platforms" on page 3718

Authorization:
For information about platform-specific authorizations, see the following topics:

- "Security requirements for Linux and UNIX platforms" on page 3648
- "Security requirements for Windows systems" on page 3651

If you have enabled broker administration security, you must also set up the authority detailed in "Tasks and authorizations for broker administration security" on page 3645.

On all Windows platforms, you must install WebSphere MQ on a domain server, and create a user to own WebSphere MQ resources. The user must either be a member of domain group `mqm` or a member of another global domain group which is directly or indirectly a member of `mqm`. Make this user the owner of the shared queue manager and log files. The `sid` of the user who owns the queue manager and log files is then the same as the `sid` of the user that runs instances of the queue manager.

On Linux and UNIX systems, the user ID used to run this command must be a member of both the mqbrkrs group and the mqm group. Additionally, you need to make the `uid` and `gid` for this user ID the same on all the systems, and the user ID needs to be the same one that created the first instance of the multi-instance broker, using the **mqsicreatebroker** command.

You might need to edit the `/etc/passwd` file on each system to set a common `uid` and `gid` for the user ID being used to create and add broker instances; then reboot your systems.

You should change the `uid` and `gid` in the Linux and UNIX environments with caution, as it affects the permission levels of files on the system.

Changing a `uid` or `gid` causes the ownership of all the files previously owned by that user or group to change to the actual integer of the previous owner of the file. Therefore, you must ensure that your system administrator manually restores the ownerships of the affected files and directories.

Related reference:

"Syntax diagrams" on page 3677

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

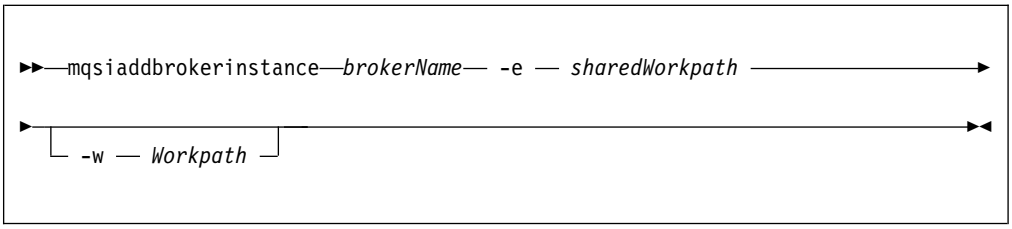
"**mqsideletebroker** command" on page 3863
Use the **mqsideletebroker** command to delete a named broker. The command also deletes the queues on the associated queue manager (created when the broker was created). You can also specify that the queue manager is to be deleted.

"**mqsiremovebrokerinstance** command" on page 3918
Use the **mqsiremovebrokerinstance** command to remove a multi-instance broker from a server where WebSphere Message Broker has been installed.

mqsiaddbrokerinstance *command - Linux and UNIX systems:*

Use the **mqsiaddbrokerinstance** command to add a multi-instance broker on Linux and UNIX systems.

Syntax:



Parameters:

brokerName
(Required) The name of the broker instance that you are adding. You must specify the name as the first parameter and the name is case-sensitive. The broker instance name must match that of a multi-instance enabled broker previously created using the **mqsicreatebroker** command.

For restrictions on the character set that you can use, see "Characters allowed in commands" on page 3680.

-e ***sharedWorkpath***
(Required) The value represents the directory in which globally accessible working files for this broker are located in shared network storage (NFS or NAS).

You must ensure the broker has access to this network storage location before you start the broker, and that the queue manager for the broker has been configured as a WebSphere MQ multi-instance queue manager.

The information stored in this shared directory includes:

- The broker registry
- Component directories
- Internal broker tables and files for deployed message flows
- Configurable service properties.

-w ***Workpath***
(Optional) The directory in which working files specific to this broker instance are stored locally on the server where the broker instance is going to run. If you do not specify this parameter, files are stored in the default Work path,

which is the one you specified when the product was installed. If you specify this parameter, you must create this directory before you start the broker.

This directory is also used for trace records that are created when tracing is active. These records are written to a subdirectory, `log`, which you must create before you start the broker.
Error logs that are written by the broker when a process ends abnormally are stored in this directory.
The error log is unbounded and continues to grow. Check this directory periodically and clear out old error information.
You cannot change this parameter using the **mqsichangebroker** command. To specify or change the work path, delete and re-create the broker.
Specifying this parameter creates a separate working directory for the broker. This working directory is a subset of the default working directory structure that contains fewer subdirectories and no `common\profiles` subdirectory.

Examples:
The following example adds a broker instance on broker `Mybroker` using queue manager `MyQmgr` on the shared work path `MyNetworkSharedWorkpath`:

```
mqsiaddbrokerinstance MyBroker –e /MyNetworkSharedWorkpath
```

Related reference:

"Syntax diagrams" on page 3677

"**mqsiaddbrokerinstance** command" on page 3715
Use the **mqsiaddbrokerinstance** command to create a multi-instance broker on a server where WebSphere Message Broker has been installed.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

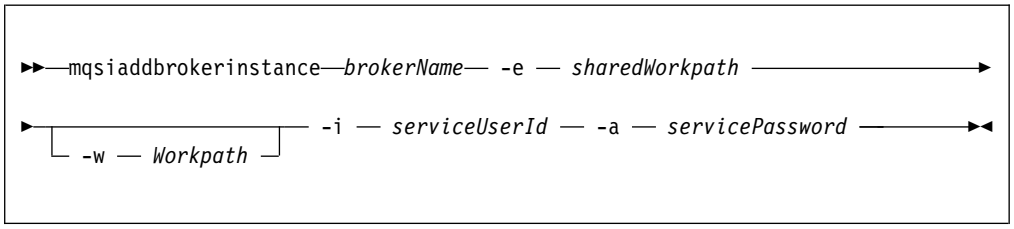
"**mqsideletebroker** command" on page 3863
Use the **mqsideletebroker** command to delete a named broker. The command also deletes the queues on the associated queue manager (created when the broker was created). You can also specify that the queue manager is to be deleted.

"**mqsiremovebrokerinstance** command" on page 3918
Use the **mqsiremovebrokerinstance** command to remove a multi-instance broker from a server where WebSphere Message Broker has been installed.

mqsiaddbrokerinstance *command - Windows platforms:*

Use the **mqsiaddbrokerinstance** command to add a multi-instance broker on Windows platforms.

Syntax:



Parameters:

brokerName
(Required) The name of the broker instance that you are adding. You must

specify the name as the first parameter and the name is case-sensitive. The broker instance name must match that of a multi-instance enabled broker previously created using the **mqsicreatebroker** command.

For restrictions on the character set that you can use, see "Characters allowed in commands" on page 3680.

-e ***sharedWorkpath***
(Required) The value represents the directory in which globally accessible working files for this broker are located in shared network storage (NFS or NAS).

You must ensure the broker has access to this network storage location before you start the broker, and that the queue manager for the broker has been configured as a WebSphere MQ multi-instance queue manager.

The information stored in this shared directory includes:

- The broker registry
- Component directories
- Internal broker tables and files for deployed message flows
- Configurable service properties.

-w ***Workpath***
(Optional) The directory in which working files specific to this broker instance are stored locally on the server where the broker instance is going to run. If you do not specify this parameter, files are stored in the default Work path, which is the one you specified when the product was installed. If you specify this parameter, you must create this directory before you start the broker.

This directory is also used for trace records that are created when tracing is active. These records are written to a subdirectory, `log`, which you must create before you start the broker.
Error logs that are written by the broker when a process ends abnormally are stored in this directory.
The error log is unbounded and continues to grow. Check this directory periodically and clear out old error information.
You cannot change this parameter using the **mqsichangebroker** command. To specify or change the work path, delete and re-create the broker.
Specifying this parameter creates a separate working directory for the broker. This working directory is a subset of the default working directory structure that contains fewer subdirectories and no `common\profiles` subdirectory.

-i *serviceUserId*
(Required) The user ID under which the broker runs.

You can specify the *serviceUserId* in any valid user name syntax:

- `\\server\username`
- `.\username`
- `username`

Do not use a domain name as part of the *serviceUserId* parameter.
If you use the unqualified form for this user ID (`username`), the operating system searches for the user ID throughout its domain, starting with the local system. This search might take some time to complete.
The *serviceUserId* that you specify must be a direct or indirect member of the mqbrkrs local group. The *serviceUserId* must also be authorized to access the home directory (where WebSphere Message Broker has been installed), and the working directory (if specified by the **-w** parameter).
If you specify that the broker is to run as a WebSphere MQ trusted application

(**-t** parameter), you must also add the service user ID to the mqm group. The security requirements for the *serviceUserId* are described in "Security requirements for Windows systems" on page 3651.

-a *servicePassword*
(Required) The password for the *serviceUserId*.

For compatibility with existing systems, you can specify `<password>`. However, if you do not specify a password with this parameter when you run the command, you are prompted to enter a password. You must enter the password a second time to verify that you have entered it correctly.

Examples:
The following example adds a broker instance on broker `Mybroker` on the shared work path `MyNetworkSharedWorkpath`:

```
mqsiaddbrokerinstance MyBroker –e /MyNetworkSharedWorkpath -i Fred -a Test
```

Related reference:

"Syntax diagrams" on page 3677

"**mqsiaddbrokerinstance** command" on page 3715
Use the **mqsiaddbrokerinstance** command to create a multi-instance broker on a server where WebSphere Message Broker has been installed.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsideletebroker** command" on page 3863
Use the **mqsideletebroker** command to delete a named broker. The command also deletes the queues on the associated queue manager (created when the broker was created). You can also specify that the queue manager is to be deleted.

"**mqsiremovebrokerinstance** command" on page 3918
Use the **mqsiremovebrokerinstance** command to remove a multi-instance broker from a server where WebSphere Message Broker has been installed.

mqsibackupbroker command:

Use the **mqsibackupbroker** command to back up the current configuration of a broker.

Supported platforms:

- Windows
- Linux and UNIX systems
- z/OS. Run this command by customizing and submitting `BIPBUBK`; see "Contents of the broker PDSE" on page 3991.

Purpose:
The **mqsibackupbroker** command creates a record of the current configuration of a broker in a file, in compressed file format. You can use the file to restore the broker at a later time, if required.

Usage notes:
The command backs up of the following persistent configuration data associated with the broker:

- Deployed resources; message flows, dictionaries, JAR files, and other runtime resources that you have previously deployed in a BAR file.
- Execution groups.

- Broker configuration; for example, configurable services.

The command does not back up the following resources:

- Transient information; for example, inflight aggregations or collections.
- Executable code, including resources that are associated with user-defined extensions (nodes, parsers, and exits).
- Resources required by message flows to function correctly; for example, WebSphere MQ queues and data stored in user databases.

Use the backup file to restore a broker on a computer that has an identical configuration; the operating system must be at the same level, and the broker and queue manager names must be identical.

You can run this command for a broker that is active. However, you must not take a backup while the broker is processing configuration changes and deployments; the backup file created might contain incomplete information. If the file contains partial records, you cannot use it to restore the broker at a later time.

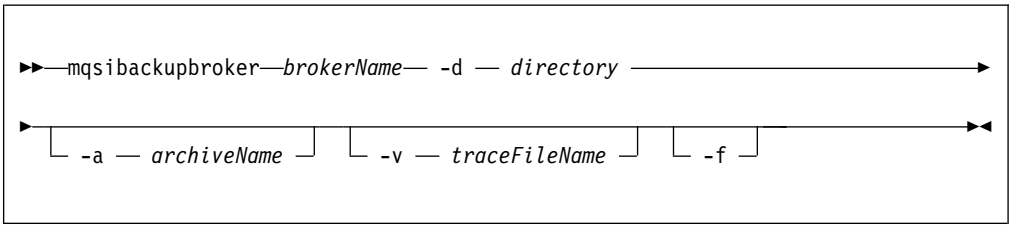
To ensure that the backup is complete and correct, either take a backup when the broker is not processing a configuration change (such as a deployment or change property) or when the broker is stopped.

Authorization:
For information about platform-specific authorizations, see the following topics:

- "Security requirements for Linux and UNIX platforms" on page 3648
- "Security requirements for Windows systems" on page 3651
- "Security requirements for z/OS" on page 3655

If you have enabled broker administration security, you must also set up the authority detailed in "Tasks and authorizations for broker administration security" on page 3645.

Syntax:



Parameters:

brokerName
(Required) The name of the broker that you want to back up. You must specify the name as the first parameter.

-d *directory*
(Required) The directory in which the backup file is created. You must specify a directory that is on a file system that can be accessed by the computer on which you run this command.

-a *archiveName*
(Optional) The name of the backup (archive) file. The file is created in compressed file format. If you do not specify this parameter, the default name *brokerName_yyMMdd_HHmmss*.zip is used.

-v *traceFileName*
(Optional) The location of a trace file that records details of the actions taken by the command.

-f
(Optional) Force incomplete backup files to be created if one or more parts of the broker configuration cannot be read.

The command creates a backup file that contains all available information; if it cannot access some configuration detail, that subset of data is not saved. For example, if a message flow required exclusive access to a deployed resource, and the message flow is running, the deployed resource might not be included in the backup file.

If you do not specify this option, the command fails if it cannot create complete backup file.

Specify this option only if directed to do so by your IBM service representative.

Examples:
The following example backs up broker MB7BROKER on Windows:

```
mqsibackupbroker MB7BROKER -d C:\MQSI\BACKUP -v C:\MQSI\BACKUP\trace.log
```

Related tasks:

"Backing up the broker" on page 1013
Back up the broker configuration and all associated resources.

"Command environment: Linux and UNIX systems" on page 310
Set up the Linux or UNIX environment to run WebSphere Message Broker commands.

"Command environment: Windows systems" on page 306
Set up the Windows environment to run WebSphere Message Broker commands.

Related reference:

"Contents of the broker PDSE" on page 3991
After you have successfully customized the broker, the broker PDSE members have been set up.

"Syntax diagrams" on page 3677

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsideletebroker** command" on page 3863
Use the **mqsideletebroker** command to delete a named broker. The command also deletes the queues on the associated queue manager (created when the broker was created). You can also specify that the queue manager is to be deleted.

"**mqsirestorebroker** command" on page 3952
Use the **mqsirestorebroker** command to restore the broker configuration from a backup file.

"Sample BIPBUBK file" on page 4004
The sample BIPBUBK file that is shipped with WebSphere Message Broker is

included here for your reference.

mqsichangebroker command:

Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

Note: You can use the **mqsireportbroker** command to report the current values of the configuration parameters of the broker before you change anything, and to confirm any changes that you made. See "**mqsireportbroker** command" on page 3919 for more information.

Supported platforms:

- Windows
- Linux and UNIX systems
- z/OS. Run this command either as a console command, or by customizing and submitting BIPCHBK; see "Contents of the broker PDSE" on page 3991

Purpose:
You can modify the value of many, but not all, of the parameters that you set when you created the broker. For example, after you change a password, you must run the **mqsichangebroker** command. You can also use this command to set the **UserExitPath** property or to enable administrative authority.

You can use the **mqsichangebroker** command with the **-c** parameter on WebSphere Message Broker Version 7.0 in the same way that you did for WebSphere Message Broker .

However, the converter file name must follow the WebSphere Message Broker Version 7.0 form, and the location given must specify the path up to, but not including, the *icudt38b* component; see "Generating a new code page converter" on page 824 for further information.

You must stop the broker before you issue this command, and restart the broker for the changes to take effect:

- On Windows, Linux, and UNIX systems, use the **mqsistop** and **mqsistart** commands to stop and restart the broker.
- On z/OS, you must have started the original broker control process by using the `/S` option. You must stop the broker components using the `/F broker,PC` option and start the broker components again using the `/F broker,SC` option.

On Windows systems, Linux, and UNIX systems, use this command to specify whether the broker can start and stop as a WebSphere MQ service.

Use the **-f** (`function level`) parameter to specify functions that become available in WebSphere Message Broker fix packs. See the version of the command that is applicable to your enterprise for more information.

For details of this command on the operating system that your enterprise uses, see the appropriate topic:

- "**mqsichangebroker** command - Linux and UNIX systems" on page 3724
- "**mqsichangebroker** command - Windows systems" on page 3729
- "**mqsichangebroker** command - z/OS" on page 3733

Authorization:
For information about platform-specific authorizations, see the following topics:

- "Security requirements for Linux and UNIX platforms" on page 3648
- "Security requirements for Windows systems" on page 3651
- "Security requirements for z/OS" on page 3655

If you have enabled broker administration security, you must also set up the authority detailed in "Tasks and authorizations for broker administration security" on page 3645.

Related concepts:

"User exits" on page 3015
A user exit is user-provided custom software, written in C, to track data passing through message flows.

Related tasks:

Chapter 8, "Administering brokers and broker resources," on page 899
Administering brokers and associated broker resources includes the tasks that you perform frequently to activate and manage those resources. Choose the method you prefer to administer your brokers and associated resources.

"Generating a new code page converter" on page 824
Generate a code page converter to handle conversions of data that belongs to a code page that is not in the default set of code pages provided by WebSphere Message Broker.

"Developing a user exit" on page 3114
Develop a user exit by declaring it, implementing its behavior, then compiling it.

"Deploying a user exit" on page 3116
Deploy your user exit to the broker.

Related reference:

"Security requirements for Linux and UNIX platforms" on page 3648
View a summary of the authorizations in a Linux or UNIX environment.

"Security requirements for Windows systems" on page 3651
Security requirements depend on the administrative task that you want to perform.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsideletebroker** command" on page 3863
Use the **mqsideletebroker** command to delete a named broker. The command also deletes the queues on the associated queue manager (created when the broker was created). You can also specify that the queue manager is to be deleted.

"**mqsistart** command" on page 3965
Use the **mqsistart** command to start the specified broker if all initial verification tests complete successfully.

"**mqsistop** command" on page 3972
Use the **mqsistop** command to stop the specified component.

"START and STOP commands on z/OS" on page 3981

mqsichangebroker *command - Linux and UNIX systems:*

Use the **mqsichangebroker** command on Linux and UNIX systems to modify your broker.

Syntax:

Parameters:

BrokerName

(Required) This parameter must be the first parameter. Specify the name of the broker to modify.

-t (Optional) The broker runs as a WebSphere MQ trusted application.

For more details about using WebSphere MQ trusted applications, see the *Intercommunication* section of the WebSphere MQ Version 7 Information Center online.

-n

(Optional) The broker ceases to run as a WebSphere MQ trusted application.

-l *userLilPath*

(Optional) A list of paths (directories) from which the broker loads Loadable implementation libraries (LIL files) for user-defined message processing nodes.

On Linux and UNIX systems, directory names are case sensitive. You must include the names in single quotation marks if they contain mixed case characters.
Do not include environment variables in the path; the broker ignores them.

Create your own directory for storing your `.lil` or `.jar` files. Do not save them in the WebSphere Message Broker installation directory.
If you specify more than one additional directory, each directory must be separated by the default platform-specific path separator.

-g *configurationChangeTimeout*
(Optional) The maximum time (in seconds) that is allowed for a user configuration request to be processed. It defines the length of time taken within the broker to apply to an execution group a configuration change that you have initiated. For example, if you deploy a configuration from the WebSphere Message Broker Toolkit, the broker must respond to the Configuration Manager within this time.

A message flow cannot respond to a configuration change while it is processing an application message. An execution group returns a negative response to the deployed configuration message if any one of its message flows does not finish processing an application message and apply the configuration change within this timeout.

Specify the value in seconds, in the range 10 - 3600. The default is 300.

For information about how to set the value for this timeout, see "Setting configuration timeout values" on page 3258.

-k *internalConfigurationTimeout*
(Optional) The maximum time (in seconds) that is allowed for an internal configuration change to be processed. For example, it defines the length of time taken within the broker to start an execution group.

The response time of each execution group differs according to system load and the load of its own processes. The value must reflect the longest response time that any execution group takes to respond. If the value is too low, the broker returns a negative response, and might issue error messages to the local error log.
Specify the value in seconds, in the range 10 - 3600. The default is 60.
For information about how to set the value for this timeout, see "Setting configuration timeout values" on page 3258.

-P *httpListenerPort*
(Optional) Enter the number of the port on which the Web services support is listening.

The broker starts this listener when a message flow that includes HTTP nodes or Web services support is started; the default is 7080.

Ensure that the port that you specify has not been specified for any other purpose.

-v *statisticsMajorInterval*
(Optional) The time interval (in minutes) at which statistics and accounting archive records are written. The valid range is from 1 minute to 43200 minutes; the default value is 60.

-y *ldapPrincipal*
(Optional, but mandatory when *ldapCredentials* is provided.) The user principal for access to an optional LDAP directory that holds the JNDI administered Initial Context for the JMS provider.

-z *ldapCredentials*
(Optional, but mandatory when *ldapPrincipal* is provided.) The user password for access to LDAP.

-c *ICU converter path*
(Optional) A set of directories to search for additional code page converters, delimited by a colon (:).

The code page converters must be either of the form *codepagename*.cnv, or in an ICU data package called icudt38.dat. The code page converters must be located in a sub-directory named icudt38_<*platform_suffix*> of the specified directory where <*platform_suffix*> is one of the following values:

- l for little-endian ASCII platforms
- b for big-endian ASCII platforms
- e for EBCDIC platforms

-x ***userExitPath***
(Optional) The path that contains the location of all user exits to be loaded for execution groups in this broker. This path is added to the system library search path (PATH,LIBPATH,LD_LIBRARY_PATH,SHLIBPATH) for the execution group process only.

-e ***activeUserExits***
(Optional) Active user exits. By default, user exits are inactive. Adding a *userExit* name to this colon-separated list changes its default state to active for this broker. You can use the **mqsichangeflowuserexits** command to override the default state at the execution group or message flow level. If you specify a user exit name, and no library is found to provide that user exit when the execution group starts, a BIP2314 message is written to the system log. The execution group fails to start.

-o *operationMode*
(Optional) Use this parameter to set the mode of your broker; see "Operation modes" on page 48. Valid values that you can set are enterprise (the full package, enterprise mode), entry (Entry Edition mode), starter (Starter Edition mode), and adapter (Remote Adapter Deployment mode). The default is enterprise unless you have the Trial Edition, when the default is trial. If no **-o** parameter is set, the default is set automatically.

-f *function level*
(Optional) Use this parameter to enable function that becomes available in WebSphere Message Broker fix packs. Valid values that you can set are all, which enables all functions, or a version string of the form V.R.M.F (for example, 7.0.0.7), which indicates the maximum level of feature function to enable. You can use this parameter to revert to a previous fix pack level, but you must first remove any flows that are using the new functions.

This command does not disable all new features, and it is not possible to use this flag to run the broker at a different major version.

-s ***security_status***
(Optional) Specify this parameter to change the administrative security status of the broker.

- If you specify active, administrative security is enabled for the broker. The command creates authorization queues that are required, if they do not exist.
- If you specify inactive, administrative security is disabled for the broker. The command does not delete or clear the security queues that are associated with this broker and its execution groups.

If you omit this parameter, the security status is unchanged.

For further information about using security, see "Broker administration security overview" on page 362 and "Authorizing users for broker administration" on page 371.

-d (Optional) Enables a broker to be started and stopped as a WebSphere MQ service when the queue manager starts and stops.

This option is an alternative to starting a multi-instance broker in standby mode using the **mqsistart** command.

If you specify **-d defined**, the WebSphere MQ service is defined to the queue manager and the broker starts and stops when the queue manager starts and stops.

If you specify **-d undefined**, the WebSphere MQ service is not defined to the queue manager and the broker does not start and stop when the queue manager starts and stops. This is the default setting.

To change other broker properties, first delete and re-create the broker, and then use the WebSphere Message Broker Toolkit to redeploy the broker configuration. If you want to update the user ID credentials that the broker uses to access one or more databases from deployed message flows, use the **mqsisetdbparms** command. For more information, see "Accessing databases from message flows" on page 2112.

Examples:
Define the path to user-defined exits:

```
mqsichangebroker MB7BROKER -x /opt/3rdparty/wmbexit
```

Enable the new function that is supplied in WebSphere Message Broker Version 7.0.0.7:

```
mqsichangebroker MB7BROKER -f 7.0.0.7
```

Set the broker's security status to `active`, so that the broker checks that the user has the correct authority to complete actions that are requested after this change.

```
mqsichangebroker MB7BROKER -s active
```

Set the WebSphere MQ status of the broker to `defined`, so that the broker starts and stops when its associated queue manager starts and stops.

```
mqsichangebroker MB7BROKER -d defined
```

Set the WebSphere MQ service status to `undefined`, so that the broker no longer starts and stops when its associated queue manager starts and stops. The broker starts only when using the **mqsistart** command.

```
mqsichangebroker MB7BROKER -d undefined
```

Related tasks:

"Setting configuration timeout values" on page 3258
Change timeout values that affect configuration tasks in the broker.

"Modifying a broker on Windows, Linux, and UNIX systems" on page 632
Use the **mqsichangebroker** command on Windows, Linux, and UNIX to modify your broker.

Related reference:

"Syntax diagrams" on page 3677

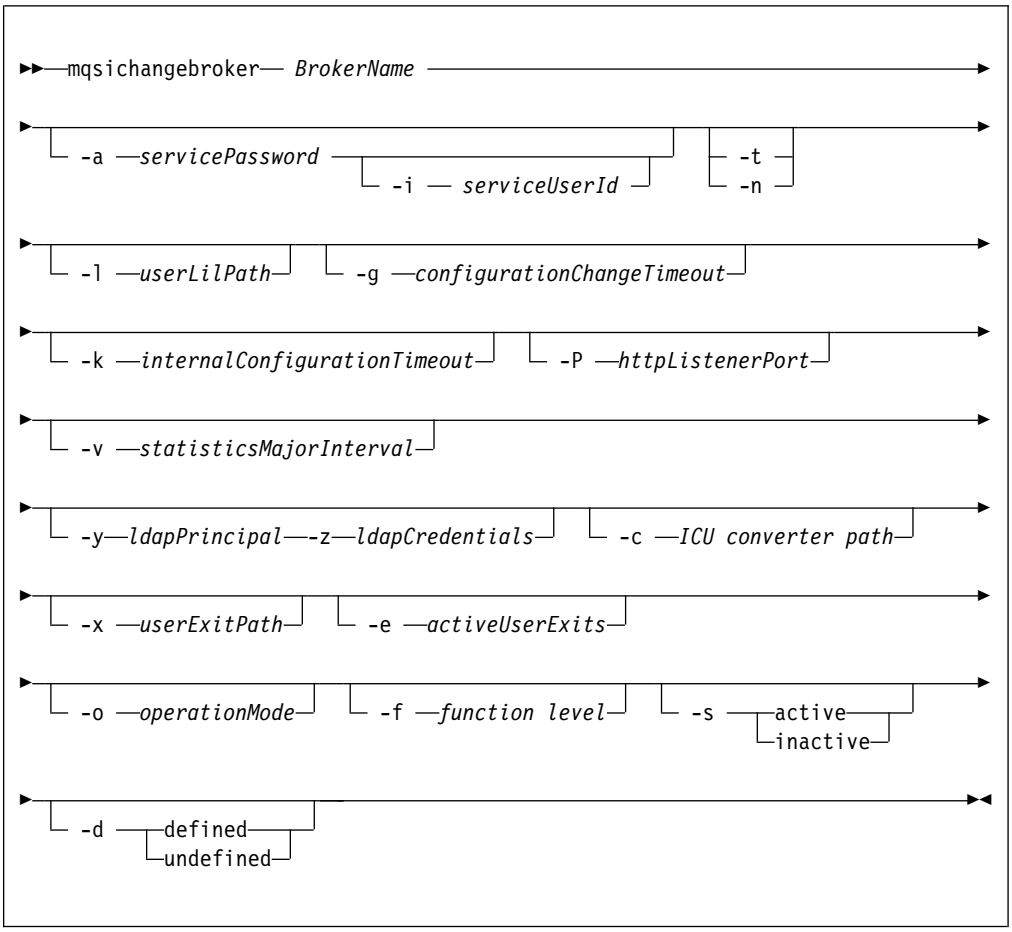
"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

mqsichangebroker *command - Windows systems:*

Use the **mqsichangebroker** command on Windows systems to modify your broker.

Syntax:



Parameters:

BrokerName

(Required) This parameter must be the first parameter. Specify the name of the broker to modify.

-a *servicePassword*
(Optional) The password for the *serviceUserId*.

-i *serviceUserId*
(Optional) The user ID under which the broker runs. You must also change the password (**-a**) if you change this value.

You can specify the *serviceUserId* in any valid user name syntax:

- `\\server\username`
- `.\username`
- `username`

Do not use a domain name as part of the *serviceUserId* parameter.
If you use the unqualified form for this user ID (`username`), the operating system searches for the user ID throughout its domain, starting with the local

system. This search might take some time to complete.
The *serviceUserId* that you specify must be a direct or indirect member of the mqbrkrs local group. The *serviceUserId* must also be authorized to access the home directory (where WebSphere Message Broker has been installed), and the working directory (if specified by the **-w** parameter).
If you specify that the broker is to run as a WebSphere MQ trusted application (**-t** parameter), you must also add the service user ID to the mqm group.
The security requirements for the *serviceUserId* are described in "Security requirements for Windows systems" on page 3651.

-t (Optional) The broker runs as a WebSphere MQ trusted application.

For more details about using WebSphere MQ trusted applications, see the *Intercommunication* section of the WebSphere MQ Version 7 Information Center online.

-n

(Optional) The broker ceases to run as a WebSphere MQ trusted application.

-l *userLilPath*
(Optional) A list of paths (directories) from which the broker loads Loadable implementation libraries (LIL files) for user-defined message processing nodes.

Create your own directory for storing your `.lil` or `.jar` files. Do not save them in the WebSphere Message Broker installation directory.
If you specify more than one additional directory, each directory must be separated by the default platform-specific path separator.

-g *configurationChangeTimeout*
(Optional) The maximum time (in seconds) that is allowed for a user configuration request to be processed. It defines the length of time taken within the broker to apply to an execution group a configuration change that you have initiated. For example, if you deploy a configuration from the WebSphere Message Broker Toolkit, the broker must respond to the Configuration Manager within this time.

A message flow cannot respond to a configuration change while it is processing an application message. An execution group returns a negative response to the deployed configuration message if any one of its message flows does not finish processing an application message and apply the configuration change within this timeout.

Specify the value in seconds, in the range 10 - 3600. The default is 300.

For information about how to set the value for this timeout, see "Setting configuration timeout values" on page 3258.

-k *internalConfigurationTimeout*
(Optional) The maximum time (in seconds) that is allowed for an internal configuration change to be processed. For example, it defines the length of time taken within the broker to start an execution group.

The response time of each execution group differs according to system load and the load of its own processes. The value must reflect the longest response time that any execution group takes to respond. If the value is too low, the broker returns a negative response, and might issue error messages to the local error log.
Specify the value in seconds, in the range 10 - 3600. The default is 60.
For information about how to set the value for this timeout, see "Setting configuration timeout values" on page 3258.

-P *httpListenerPort*
(Optional) Enter the number of the port on which the Web services support is listening.

The broker starts this listener when a message flow that includes HTTP nodes or Web services support is started; the default is 7080.

Ensure that the port that you specify has not been specified for any other purpose.

-v *statisticsMajorInterval*
(Optional) The time interval (in minutes) at which statistics and accounting archive records are written. The valid range is from 1 minute to 43200 minutes; the default value is 60.

-y *ldapPrincipal*
(Optional, but mandatory when *ldapCredentials* is provided.) The user principal for access to an optional LDAP directory that holds the JNDI administered Initial Context for the JMS provider.

-z *ldapCredentials*
(Optional, but mandatory when *ldapPrincipal* is provided.) The user password for access to LDAP.

-c *ICU converter path*
(Optional) A set of directories to search for additional code page converters, delimited by a semicolon (;).

The code page converters must be either of the form *codepagename*`.cnv`, or in an ICU data package called `icudt38.dat`. The code page converters must be located in a sub-directory named `icudt38_`*<platform_suffix>* of the specified directory where *<platform_suffix>* is one of the following values:

- l for little-endian ASCII platforms
- b for big-endian ASCII platforms
- e for EBCDIC platforms

-x *userExitPath*
(Optional) The path that contains the location of all user exits to be loaded for execution groups in this broker. This path is added to the system library search path (`PATH`,`LIBPATH`,`LD_LIBRARY_PATH`,`SHLIBPATH`) for the execution group process only.

-e *activeUserExits*
(Optional) Active user exits. By default, user exits are inactive. Adding a *userExit* name to this colon-separated list changes its default state to active for this broker. You can use the **mqsichangeflowuserexits** command to override the default state at the execution group or message flow level. If you specify a user exit name, and no library is found to provide that user exit when the execution group starts, a BIP2314 message is written to the system log. The execution group fails to start.

Note you can use double quotation marks to remove all the user exits from the active list:

```
mqsichangebroker <brokername> ""
```

-o *operationMode*
(Optional) Use this parameter to set the mode of your broker; see “Operation modes” on page 48. Valid values that you can set are `enterprise` (the full package, enterprise mode), `entry` (Entry Edition mode), `starter` (Starter Edition mode), and `adapter` (Remote Adapter Deployment mode). The default

value is `enterprise`, unless you have the Trial Edition, when the default value is `trial`. If you do not set the **-o** parameter, the default value is used automatically.

-f *function level*
(Optional) Use this parameter to enable function that becomes available in WebSphere Message Broker fix packs. Valid values that you can set are `all`, which enables all functions, or a version string of the form V.R.M.F (for example, `7.0.0.7`), which indicates the maximum level of feature function to enable. You can use this parameter to revert to a previous fix pack level, but you must first remove any flows that are using the new functions.

This command does not disable all new features, and it is not possible to use this flag to run the broker at a different major version.

-s ***security_status***
(Optional) Specify this parameter to change the administrative security status of the broker.

- If you specify `active`, administrative security is enabled for the broker. The command creates authorization queues that are required, if they do not exist.
- If you specify `inactive`, administrative security is disabled for the broker. The command does not delete or clear the security queues that are associated with this broker and its execution groups.

If you omit this parameter, the security status is unchanged.

For further information about using security, see "Broker administration security overview" on page 362 and "Authorizing users for broker administration" on page 371.

-d (Optional) Enables a broker to be started and stopped as a WebSphere MQ service when the queue manager starts and stops.

This option is an alternative to starting a multi-instance broker in standby mode using the **mqsistart** command.

If you specify **-d defined**, the WebSphere MQ service is defined to the queue manager and the broker starts and stops when the queue manager starts and stops.

If you specify **-d undefined**, the WebSphere MQ service is not defined to the queue manager and the broker does not start and stop when the queue manager starts and stops. This is the default setting.

To change other broker properties, first delete and re-create the broker, and then use the WebSphere Message Broker Toolkit to redeploy the broker configuration. If you want to update the user ID credentials that the broker uses to access one or more databases from deployed message flows, use the **mqsisetdbparms** command. For more information, see "Accessing databases from message flows" on page 2112.

Examples:
Define the path to user-defined exits:

```
mqsichangebroker MB7BROKER -x /opt/3rdparty/wmbexit
```

Enable the new function that is supplied in WebSphere Message Broker Version 7.0.0.7:

```
mqsichangebroker MB7BROKER -f 7.0.0.7
```

Set the broker's security status to `active`, so that the broker checks that the user has the correct authority to complete actions that are requested after this change.

```
mqsichangebroker MB7BROKER -s active
```

Set the broker's WebSphere MQ status to `defined`, so that the broker starts and stops when its associated queue manager starts and stops.

```
mqsichangebroker MB7BROKER -d defined
```

Set the WebSphere MQ service status to `undefined`, so that the broker no longer starts and stops when its associated queue manager starts and stops. The broker starts only when using the **mqsistart** command.

```
mqsichangebroker MB7BROKER -d undefined
```

Related tasks:

"Setting configuration timeout values" on page 3258
Change timeout values that affect configuration tasks in the broker.

"Modifying a broker on Windows, Linux, and UNIX systems" on page 632
Use the **mqsichangebroker** command on Windows, Linux, and UNIX to modify your broker.

Related reference:

"Syntax diagrams" on page 3677

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

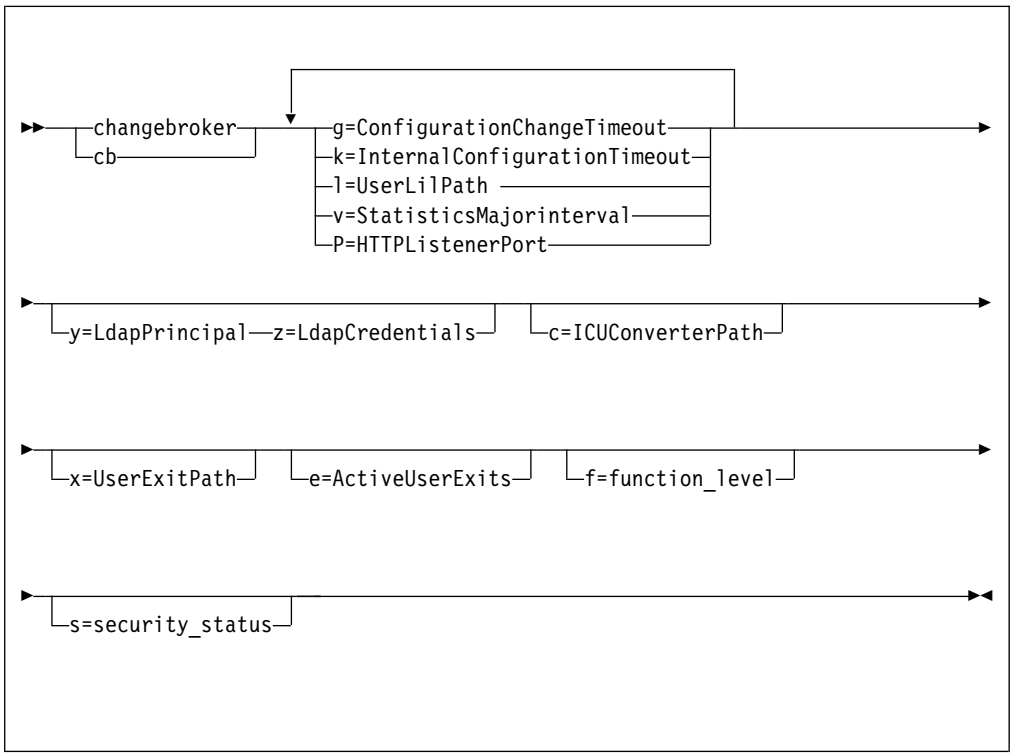
mqsichangebroker *command - z/OS:*

Use the **mqsichangebroker** command on z/OS to modify your broker.

Syntax:

z/OS command - BIPCHBK:

z/OS console command:
Synonym: cb



Parameters:

BrokerName
(Required) This parameter must be the first parameter. Specify the name of the broker to modify.

This parameter is implied in the console form of the command.

-l ***UserLilPath***
(Optional) A list of paths (directories) from which the broker loads Loadable implementation libraries (LIL files) for user-defined message processing nodes.

This name is case sensitive; enclose the names in single quotation marks if they are in mixed case.
Do not include environment variables in this path; WebSphere Message Broker ignores them.

Create your own directory for storing your `.lil` or `.jar` files. Do not save them in the WebSphere Message Broker installation directory.
If you specify more than one additional directory, each directory must be separated by the default platform-specific path separator.

-g ***ConfigurationChangeTimeout***
(Optional) The maximum time (in seconds) that is allowed for a user configuration request to be processed. It defines the length of time taken within the broker to apply to an execution group a configuration change that you have initiated. For example, if you deploy a configuration from the WebSphere Message Broker Toolkit, the broker must respond to the Configuration Manager within this time.

A message flow cannot respond to a configuration change while it is processing an application message. An execution group returns a negative response to the deployed configuration message if any one of its message flows does not finish processing an application message and apply the configuration change within this timeout.

Specify the value in seconds, in the range 10 - 3600. The default is 300.

For information about how to set the value for this timeout, see "Setting configuration timeout values" on page 3258.

-k ***InternalConfigurationTimeout***
(Optional) The maximum time (in seconds) that is allowed for an internal configuration change to be processed. For example, it defines the length of time taken within the broker to start an execution group.

The response time of each execution group differs according to system load and the load of its own processes. The value must reflect the longest response time that any execution group takes to respond. If the value is too low, the broker returns a negative response, and might issue error messages to the local error log.
Specify the value in seconds, in the range 10 - 3600. The default is 60.
For information about how to set the value for this timeout, see "Setting configuration timeout values" on page 3258.

-P ***HTTPListenerPort***
(Optional) Enter the number of the port on which the Web services support is listening.

The broker starts this listener when a message flow that includes HTTP nodes or Web services support is started; the default is 7080.

Ensure that the port that you specify has not been specified for any other purpose.

-v *StatisticsMajorInterval*
(Optional) Specify the interval (in minutes) at which statistics and accounting archive records are to be written. The valid range is from 1 minute to 43200 minutes; the default value is 60.

An interval of zero minutes indicates that the operating system has an external method of notification (the ENF timer), and is not using an internal timer within WebSphere Message Broker.

-y *LdapPrincipal*
(Optional, but mandatory when *ldapCredentials* is provided.) The user principal for access to an optional LDAP directory that holds the JNDI administered Initial Context for the JMS provider.

-z *LdapCredentials*
(Optional, but mandatory when *ldapPrincipal* is provided.) The user password for access to LDAP.

-c *ICUConverterPath*
(Optional) A delimited set of directories to search for additional code page converters; the delimiter is a period (.).

The code page converters must be either of the form *codepagename*`.cnv`, or in an ICU data package called `icudt38.dat`. The code page converters must be located in a sub-directory named `icudt38_`*<platform_suffix>* of the specified directory where *<platform_suffix>* is one of the following values:

- l for little-endian ASCII platforms
- b for big-endian ASCII platforms
- e for EBCDIC platforms

Do not use this parameter to set the converter path if you are using a converter that matches one of the built-in converters that are provided with Version 6.0, and that converter is the local code page for the broker. Use the **ICU_DATA** environment variable instead.

-x *UserExitPath*
(Optional) The path that contains the location of all user exits to be loaded for execution groups in this broker. This path is added to the system library search path (`PATH,LIBPATH,LD_LIBRARY_PATH,SHLIBPATH`) for the execution group process only.

-e *ActiveUserExits*
(Optional) Active user exits. By default, user exits are inactive. Adding a *userExit* name to this colon-separated list changes its default state to active for this broker. You can use the **mqsichangeflowuserexits** command to override the default state at the execution group or message flow level. If you specify a user exit name, and no library is found to provide that user exit when the execution group starts, a BIP2314 message is written to the system log. The execution group fails to start.

-f *function level*
(Optional) Use this parameter to enable function that becomes available in WebSphere Message Broker fix packs. Valid values that you can set are `all`, which enables all functions, or a version string of the form V.R.M.F (for example, `7.0.0.7`), which indicates the maximum level of feature function to enable. You can use this parameter to revert to a previous fix pack level, but you must first remove any flows that are using the new functions.

This command does not disable all new features, and it is not possible to use this flag to run the broker at a different major version.

-s ***security_status***
(Optional) Specify this parameter to change the administrative security status of the broker.

- If you specify `active`, administrative security is enabled for the broker. The command creates authorization queues that are required, if they do not exist.
- If you specify `inactive`, administrative security is disabled for the broker. The command does not delete or clear the security queues that are associated with this broker and its execution groups.

If you omit this parameter, the security status is unchanged.

For further information about using security, see "Broker administration security overview" on page 362 and "Authorizing users for broker administration" on page 371.

To change other broker properties, first delete and re-create the broker, and then use the WebSphere Message Broker Toolkit to redeploy the broker configuration. If you want to update the user ID credentials that the broker uses to access one or more databases from deployed message flows, use the **mqsisetdbparms** command. For more information, see "Accessing databases from message flows" on page 2112.

Examples:
When you run the console command, you must use a comma between each command option, as shown in these examples.

Change the configuration timeout parameters:

```
F MQP1BRK,cb g=100,k=200
```

Specify both the **x** and **e** parameters to set the user exit path and set an exit to active state:

```
/f MA05BRK,cb x='/u/test/wbi/MsgFlowTrackingUserExit/zOS',e='MqsiStrUserExit02:MqsiStrUserExit03'
```

Enable the new function that is supplied in WebSphere Message Broker Version 7.0.0.7:

```
F MQP1BRK,cb f=7.0.0.7
```

Set the broker's security status to `active`, so that the broker checks that the user has the correct authority to complete actions that are requested after this change.

```
mqsichangebroker MB7BROKER -s active
```

Related tasks:

"Setting configuration timeout values" on page 3258
Change timeout values that affect configuration tasks in the broker.

"Modifying a broker on z/OS" on page 634
Use the **mqsichangebroker** command on z/OS to modify your broker.

Related reference:

"Syntax diagrams" on page 3677

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

mqsichangeflowmonitoring command:

Use the **mqsichangeflowmonitoring** command to enable monitoring of message flows.

Supported operating systems:

- Windows
- Linux and UNIX systems
- z/OS: Run this command either as a console command, or by customizing and submitting BIPCHME; see "Contents of the broker PDSE" on page 3991.

Purpose:
Use the **mqsichangeflowmonitoring** command to carry out the following tasks:

- Activate or deactivate monitoring for a message flow
- Enable or disable an event source within a message flow
- Associate a configurable service with a message flow by setting its `monitoringProfile` property.

This facility can be used to change the state (enabled or disabled) of any event in the message flow, whether it was configured using the node's monitoring properties, or a monitoring profile configurable service. The new state takes effect immediately (but safely).

The state change persists, and so it survives a restart of the message flow or execution group. For an event configured using the node's monitoring properties, the state change is lost if the message flow is redeployed, unless the node property is also changed.

Select the appropriate link for details of the **mqsichangeflowmonitoring** command on the platform that your enterprise uses:

- "**mqsichangeflowmonitoring** command - Windows, Linux and UNIX systems" on page 3739
- "**mqsichangeflowmonitoring** command - z/OS" on page 3741

Authorization:
For information about platform-specific authorizations, see the following topics:

- "Security requirements for Linux and UNIX platforms" on page 3648
- "Security requirements for Windows systems" on page 3651
- "Security requirements for z/OS" on page 3655

If you have enabled broker administration security, you must also set up the authority detailed in "Tasks and authorizations for broker administration security" on page 3645.

Related tasks:

"Business-level monitoring" on page 3319
You can configure your message flow to emit event messages that can be used to support transaction monitoring and auditing, and business process monitoring.

Related reference:

"Syntax diagrams" on page 3677

"**mqsichangeflowmonitoring** command"
Use the **mqsichangeflowmonitoring** command to enable monitoring of message flows.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

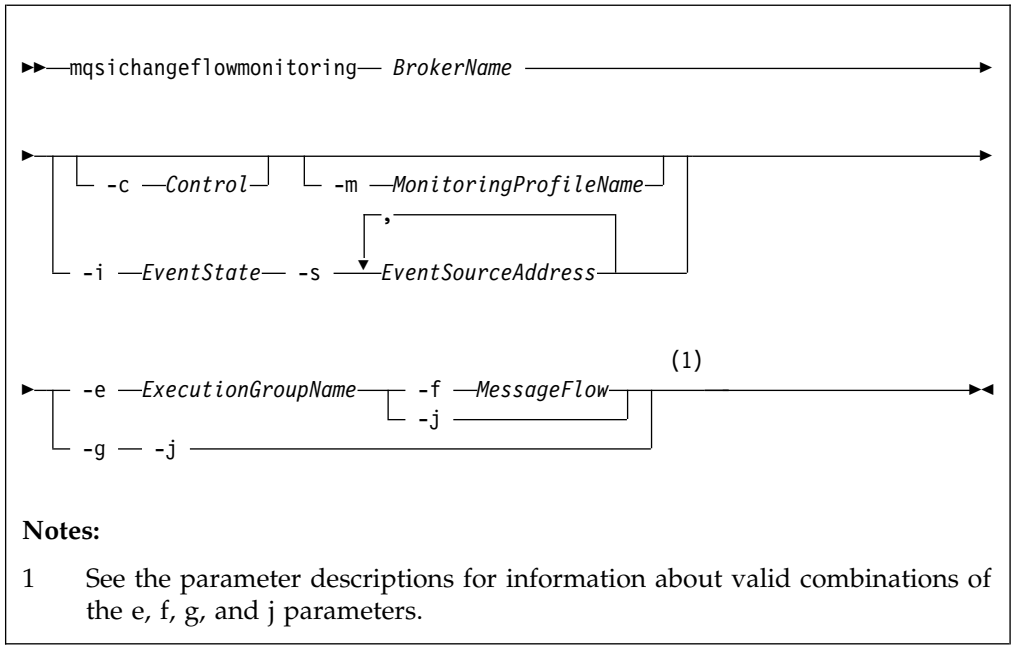
"Security requirements for Linux and UNIX platforms" on page 3648
View a summary of the authorizations in a Linux or UNIX environment.

"Security requirements for Windows systems" on page 3651
Security requirements depend on the administrative task that you want to perform.

mqsichangeflowmonitoring *command - Windows, Linux and UNIX systems:*

Use the **mqsichangeflowmonitoring** command on Windows, Linux and UNIX systems to enable monitoring of message flows.

Syntax:



Notes:

1 See the parameter descriptions for information about valid combinations of the e, f, g, and j parameters.

Parameters:

BrokerName
(Required) Specify the label of the broker to which the message flows that you want to be monitored are deployed.

-c *Control*
(Optional) Specify the string value that controls monitoring for the specified message flows. Possible values are:

`active` - activate monitoring

`inactive` - deactivate monitoring

-f *MessageFlow*
(Required) Specify the label for the message flow, for which the monitoring options are to be activated or updated.

You must specify either **-f** or **-j**. If you do not specify one of these arguments you receive an error message.

-g

(Required) Specifies that the command applies to **all** execution groups that belong to the broker.

You must specify either **-e** or **-g**. If you do not specify one of these arguments you receive an error message.

-i EventState

(Optional) Specify the string value that controls monitoring for the specified event source. Valid only when used with the -e and -f parameters. Possible values are:

`enable` - enable monitoring for the specified event sources.

`disable` - disable monitoring for the specified event sources.

-j

(Required) Specifies that the command applies to **all** message flows that belong to the execution group.

You must specify either **-f** or **-j**. If you do not specify one of these arguments you receive an error message.

Note: If you set the **-g** option for all execution groups, you must use **-j** instead of **-f**.

-m *MonitoringProfileName*

(Optional) Specify the name of the monitoring profile which the specified message flows should use.

If there is no monitoring profile with the specified name on the specified broker, the command completes successfully, and the message flows attempt to use the specified monitoring profile. Each message flow logs a warning in the User Trace to indicate that it was instructed to use a nonexistent monitoring profile. No event message is created. If a monitoring profile with the specified name is later deployed to the broker, the message flows do not immediately begin to use it. A refresh of the monitoring state can be triggered by issuing the command again with the –c option to activate or reactivate monitoring.

-s EventSourceAddress

(Optional) Comma-separated list of the event sources to be enabled or disabled. Valid only when used with the -e and -f parameters. This value takes the form <*node name*>.<*event source*>, where <*event source*> is one of the following values:

`'terminal.`<*terminal name*>'

`'transaction.Start'`

`'transaction.End'`

`'transaction.Rollback'`

If a message flow contains two or more nodes with identical names, the event sources on those nodes cannot be accurately addressed. If this is attempted, behavior is undefined.

<*node name*> is the label of the node as known by the broker runtime components. If the node is in a subflow, the label reflects this. For example, flow A contains an instance of flow B as a subflow that is labeled 'myB'. Flow B contains an instance of a Compute node that is labeled 'myCompute'. The <*node name*> for the Compute node is 'myB.myCompute'.

If you issue this command with a comma separated list to enable or disable individual event sources, and you have not already issued the command with the **-c** parameter, monitoring is not activated for these event sources. To enable monitoring, reissue the command with the **-c** parameter set to `active`.

Examples:
Assign *monitoringProfile1* to *messageFlow1* in execution group *default*:

```
mqsichangeflowmonitoring WBRK_BROKER -e default
     -f messageFlow1 -m monitoringProfile1
```

Activate monitoring for all message flows in all execution groups:

```
mqsichangeflowmonitoring WBRK_BROKER -c active -g -j
```

Enable individual event sources:

```
mqsichangeflowmonitoring WBRK_BROKER
     -e default
     -f myMessageFlow
     -s "SOAP Input1.terminal.out,MQOutput1.terminal.in"
     -i enable
```

Related tasks:

"Business-level monitoring" on page 3319
You can configure your message flow to emit event messages that can be used to support transaction monitoring and auditing, and business process monitoring.

Related reference:

"Syntax diagrams" on page 3677

"**mqsichangeflowmonitoring** command" on page 3738
Use the **mqsichangeflowmonitoring** command to enable monitoring of message flows.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

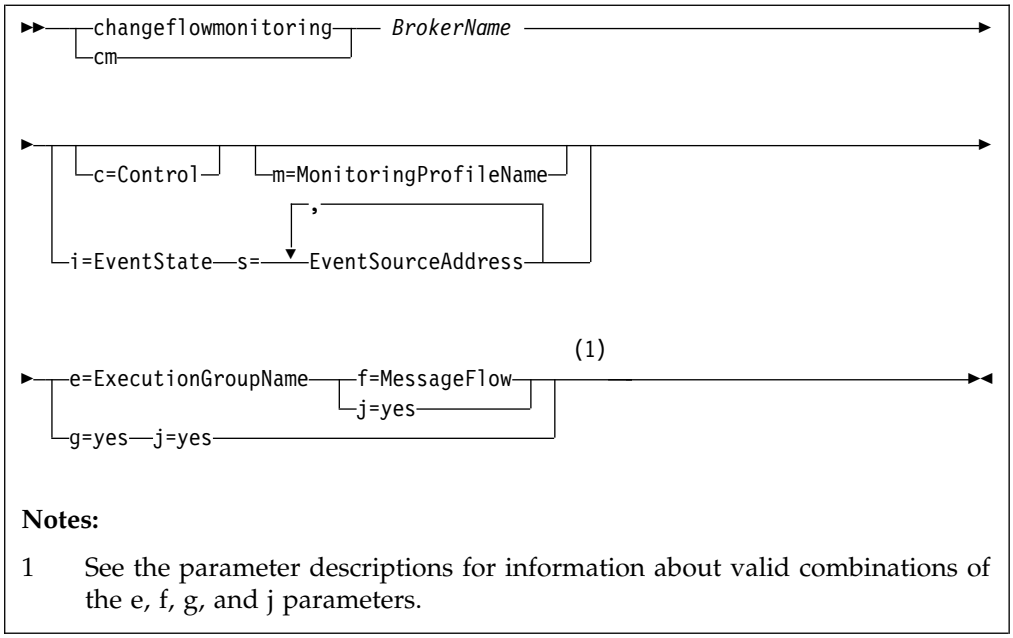
"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

mqsichangeflowmonitoring *command - z/OS:*

Use the **mqsichangeflowmonitoring** command on z/OS to enable monitoring of message flows.

Syntax:

z/OS console command:
Synonym: cm



Notes:

1 See the parameter descriptions for information about valid combinations of the e, f, g, and j parameters.

Parameters:

BrokerName
(Required) Specify the label of the broker to which the message flows that you want to be monitored are deployed.

This parameter is implied on the console form of the command.

c=Control
(Optional) Specify the string value that controls monitoring for the specified message flows. Possible values are:

`active` - activate monitoring

`inactive` - deactivate monitoring

e=ExecutionGroupName
(Required) Specify the name for the execution group to which the message flows that you want to be monitored are deployed.

You must specify either **-e** or **-g**. If you do not specify one of these arguments you receive an error message.

f=MessageFlow
(Required) Specify the label for the message flow, for which the monitoring options are to be activated or updated.

You must specify either **-f** or **-j**. If you do not specify one of these arguments you receive an error message.

g (Required) Specifies that the command applies to **all** execution groups that belong to the broker.

You must specify either **-e** or **-g**. If you do not specify one of these arguments you receive an error message.

i=EventState
(Optional) Specify the string value that controls monitoring for the specified event source. Valid only when used with the -e and -f parameters. Possible values are:

`enable` - enable monitoring for the specified event sources.

`disable` - disable monitoring for the specified event sources.

j (Required) Specifies that the command applies to **all** message flows that belong to the execution group.

You must specify either **-f** or **-j**. If you do not specify one of these arguments you receive an error message.

Note: If you set the **-g** option for all execution groups, you must use **-j** instead of **-f**.

m=MonitoringProfileName
(Optional) Specify the name of the monitoring profile which the specified message flows should use.

If there is no monitoring profile with the specified name on the specified broker, the command completes successfully, and the message flows attempt to use the specified monitoring profile. Each message flow logs a warning in the User Trace to indicate that it was instructed to use a nonexistent monitoring profile. No event message is created. If a monitoring profile with the specified name is later deployed to the broker, the message flows do not immediately begin to use it. A refresh of the monitoring state can be triggered by issuing the command again with the –c option to activate or reactivate monitoring.

s=EventSourceAddress
(Optional) Comma-separated list of the event sources to be enabled or disabled. Valid only when used with the -e and -f parameters. This value takes the form <*node name*>.<*event source*>, where <*event source*> is one of the following values:

`'terminal.`<*terminal name*>'

`'transaction.Start'`

`'transaction.End'`

`'transaction.Rollback'`

If a message flow contains two or more nodes with identical names, the event sources on those nodes cannot be accurately addressed. If this is attempted, behavior is undefined.

<*node name*> is the label of the node as known by the broker runtime components. If the node is in a subflow, the label reflects this. For example, flow A contains an instance of flow B as a subflow that is labeled 'myB'. Flow B contains an instance of a Compute node that is labeled 'myCompute'. The <*node name*> for the Compute node is 'myB.myCompute'.

If you issue this command with a comma separated list to enable or disable individual event sources, and you have not already issued the command with the **-c** parameter, monitoring is not activated for these event sources. To enable monitoring, reissue the command with the **-c** parameter set to `active`.

Example:
Activate monitoring on all message flows in all execution groups:

```
F MI10BRK,cm c=active,g=yes,j=yes
```

Related tasks:

“Business-level monitoring” on page 3319
You can configure your message flow to emit event messages that can be used to support transaction monitoring and auditing, and business process monitoring.

Related reference:

“Syntax diagrams” on page 3677

"**mqsichangeflowmonitoring** command" on page 3738
Use the **mqsichangeflowmonitoring** command to enable monitoring of message flows.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

mqsichangeflowstats command:

Use the **mqsichangeflowstats** command to control the accumulation of statistics about message flow operation.

Supported Platforms:

- Windows
- Linux and UNIX systems
- z/OS. Run this command in one of two ways - as a console command, or by customizing and submitting BIPCHMS; see "Contents of the broker PDSE" on page 3991

Purpose:
Use the **mqsichangeflowstats** command to:

- Turn on or off accounting and statistics snapshot publication, or archive record output.
- Specify that the command is applied to a specific flow message flow, or all flows in an execution group, or all execution groups belonging to a broker.
- Modify the granularity of the data collected in addition to the standard message flow accounting and statistics. This extra data can include thread related data, node related data, node terminal related data, or a mixture of this data.

The options set using this command remain active until modified by a subsequent **mqsichangeflowstats** command, or until affected by a subsequent deploy.

Select the appropriate link for details of this command on the platform, or platforms, that your enterprise uses:

- "mqsichangeflowstats command - Windows, Linux and UNIX systems" on page 3745
- "mqsichangeflowstats command - z/OS" on page 3748

Authorization:
For information about platform-specific authorizations, see the following topics:

- "Security requirements for Linux and UNIX platforms" on page 3648
- "Security requirements for Windows systems" on page 3651
- "Security requirements for z/OS" on page 3655

If you have enabled broker administration security, you must also set up the authority detailed in "Tasks and authorizations for broker administration security" on page 3645.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an

input message is received.

Related tasks:

"Optimizing message flow throughput" on page 587
Each message flow that you design must provide a complete set of processing for messages received from a certain source. This design might result in very complex message flows that include large numbers of nodes that can cause a performance overhead, and might create potential bottlenecks. You can increase the number of message flows that process your messages to provide the opportunity for parallel processing and therefore improved throughput.

"Optimizing message flow response times" on page 3264
You can use different solutions to improve message flow response times.

Related reference:

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

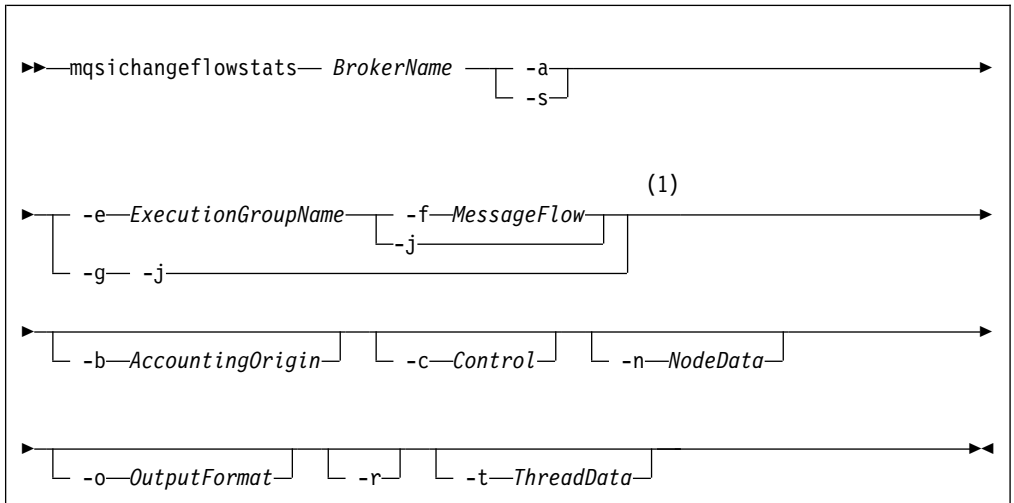
"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsireportflowstats** command" on page 3929
Use the **mqsireportflowstats** command to display the current options for accounting and statistics that have been set using the **mqsichangeflowstats** command.

mqsichangeflowstats command - Windows, Linux and UNIX systems:

Use the **mqsichangeflowstats** command on Windows, Linux and UNIX systems to control the accumulation of statistics about message flow operation.

Syntax:



Notes:

1 See the parameter descriptions for information on valid combinations of the e, f, g, and j parameters.

Parameters:

BrokerName
(Required) Specify the label of the broker for which accounting and statistics are to be changed.

-a (Required) Specify that the command modifies archive accounting and statistics collection.

You must specify either **-a** or **-s**. If you do not specify one of these arguments you receive a warning message.

-b *AccountingOrigin*
(Optional) Specifies that the environment tree path *Broker.Accounting.Origin* is used to partition the collected statistics into distinct outputs. Possible values are:

- none - do not partition statistics according to accounting origin data
- basic - partition statistics according to accounting origin data

-c *Control*
(Optional) Specify the string value that controls the level of the action to be applied to accounting and statistics collection for snapshot or archiving. Possible values are:

- active - turn on snapshot or archiving
- inactive - turn off snapshot or archiving.

-e *ExecutionGroupName*
(Required) Specify the name for the execution group, for which accounting and statistics options are to be changed.

You must specify either **-e** or **-g**. If you do not specify one of these arguments you receive a warning message.

-f *MessageFlow*
(Required) Specify the label for the message flow, for which accounting and statistics options are to be changed.

You must specify either **-f** or **-j**. If you do not specify one of these arguments you receive a warning message.

-g
(Required) Specifies that the command applies to **all** execution groups that belong to the broker.

You must specify either **-e** or **-g**. If you do not specify one of these arguments you receive a warning message.

-j
(Required) Specifies that the command applies to **all** message flows that belong to the execution group.

You must specify either **-f** or **-j**. If you do not specify one of these arguments you receive a warning message.

Note: If you set the **-g** option for all execution groups, you must use **-j** instead of **-f**.

-n *NodeData*
(Optional) Specify a string value to modify the collection of node statistics data for a message flow. Possible values are:

- none - exclude node related data in the statistics
- basic - include node related statistics in the statistics
- advanced - include node related and terminal related data in the statistics

-o *OutputFormat*
(Optional) Specify the output destination for the statistics reports. Possible values are:

- usertrace - this is the default and writes "bip" messages to usertrace, which can be post processed in the normal way using the **mqsireadlog** and **mqsiformatlog** commands
- xml - the statistics reports are generated as XML documents and published by the broker running the message flow.

  The topic on which the data is published has the following structure:

  ```
  $SYS/Broker/<brokerName>/StatisticsAccounting/<recordType>
  /<executionGroupLabel>/<messageFlowLabel>
  ```

  where recordType is set to Snapshot or Archive, and broker, execution group, and message flow names are specified according to the subscriber's requirements.

-r (Optional) This parameter applies only to archive data and specifies that archive data is to be reset.

This results in the clearing out of accounting and statistics data accumulated so far for this interval, and restarts collection from this point. All archive data for all flows in the execution group, or groups, is reset.

The archive interval timer is only reset if the **-v** option (*statistics archive interval*) of **mqsicreatebroker** or **mqsichangebroker** is non zero.

That is, the interval timer is set only if the internal interval notification mechanism is being used, and not an external method.

-s (Required) Specify that the command modifies snapshot accounting and statistics collection.

You must specify either **-a** or **-s**. If you do not specify one of these arguments you receive a warning message.

-t *ThreadData*
(Optional) Specify a string value to modify the collection of thread statistics data for a message flow Possible values are:

- none - exclude thread related data from the statistics
- basic - include thread related data in the statistics

Examples:
Turn on snapshot statistics for the message flow "myFlow1" in all execution groups of BrokerA and specify that the statistics are not to be partitioned according to accounting origin data:

```
mqsichangeflowstats BrokerA -s -g -j -b none -c active
```

Turn off the collection of archive statistics for message flow "MyFlow1" in execution group "EGRP2" for BrokerA, and at the same time modify the granularity of data that is to be collected (when next activated) to include thread related data.

```
mqsichangeflowstats BrokerA -a -e EGRP2 -f MyFlow1 -c inactive -t basic
```

Turn off snapshot data for all message flows in all execution groups for Broker A.

```
mqsichangeflowstats BrokerA -s -g -j -c inactive
```

Related reference:

"Syntax diagrams" on page 3677

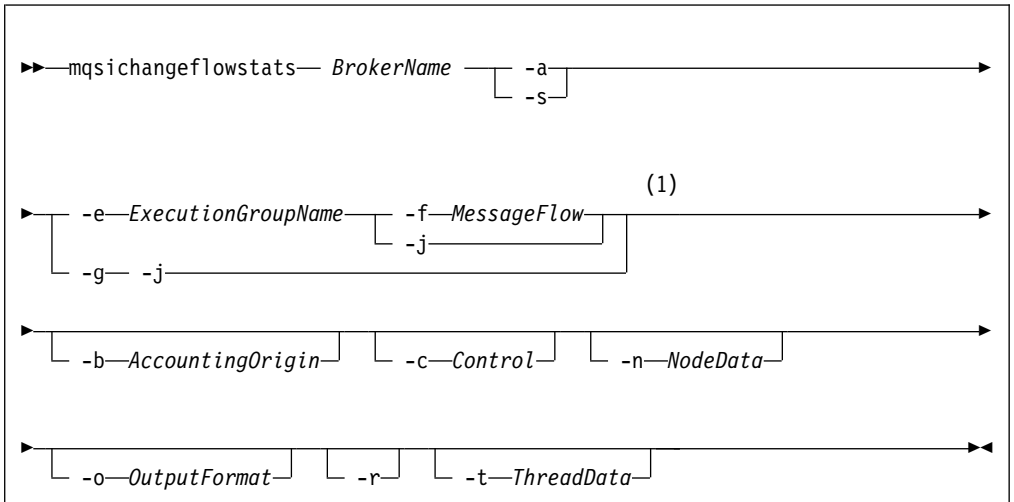
"**mqsichangeflowstats** command" on page 3744
Use the **mqsichangeflowstats** command to control the accumulation of statistics about message flow operation.

mqsichangeflowstats command - z/OS:

Use the **mqsichangeflowstats** command on z/OS to control the accumulation of statistics about message flow operation.

Syntax:

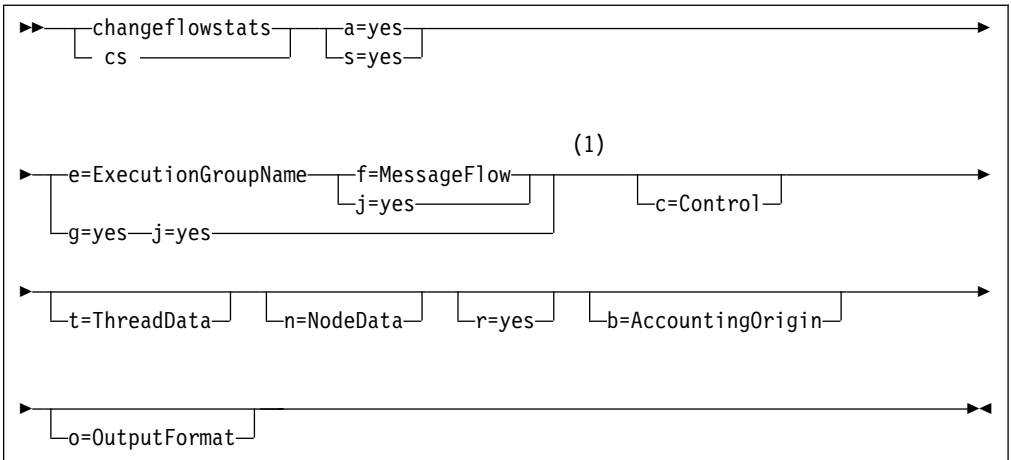
z/OS command - BIPCHMS:



Notes:

1 See the parameter descriptions for information on valid combinations of the e, f, g, and j parameters.

z/OS console command:
Synonym: cs



Notes:

1 See the parameter descriptions for information on valid combinations of the e,f, g, and j parameters.

Parameters:

BrokerName
(Required) Specify the label of the broker for which accounting and statistics are to be changed.

This parameter is implied on the console form of the command.

-a (Required) Specify that the command modifies archive accounting and statistics collection.

You must specify either **-a** or **-s**. If you do not specify one of these arguments you receive a warning message.

-b *AccountingOrigin*
(Optional) Specifies that the environment tree path *Broker.Accounting.Origin* is used to partition the collected statistics into distinct outputs. Possible values are:

- none - do not partition statistics according to accounting origin data
- basic - partition statistics according to accounting origin data

-c *Control*
(Optional) Specify the string value that controls the level of the action to be applied to accounting and statistics collection for snapshot or archiving. Possible values are:

- active - turn on snapshot or archiving
- inactive - turn off snapshot or archiving.

-e *ExecutionGroupName*
(Required) Specify the name for the execution group, for which accounting and statistics options are to be changed.

You must specify either **-e** or **-g**. If you do not specify one of these arguments you receive a warning message.

-f *MessageFlow*
(Required) Specify the label for the message flow, for which accounting and statistics options are to be changed.

You must specify either **-f** or **-j**. If you do not specify one of these arguments you receive a warning message.

-g
(Required) Specifies that the command applies to **all** execution groups that belong to the broker.

You must specify either **-e** or **-g**. If you do not specify one of these arguments you receive a warning message.

-j
(Required) Specifies that the command applies to **all** message flows that belong to the execution group.

You must specify either **-f** or **-j**. If you do not specify one of these arguments you receive a warning message.

Note: If you set the **-g** option for all execution groups, you must use **-j** instead of **-f**.

-n *NodeData*
(Optional) Specify a string value to modify the collection of node statistics data for a message flow. Possible values are:

- none - exclude node related data in the statistics
- basic - include node related statistics in the statistics
- advanced - include node related and terminal related data in the statistics

-o *OutputFormat*
(Optional) Specify the output destination for the statistics reports. Possible values are:

- usertrace - this is the default and writes "bip" messages to usertrace, which can be post processed in the normal way using the **mqsireadlog** and **mqsiformatlog** commands
- xml - the statistics reports are generated as XML documents and published by the broker running the message flow.

  The topic on which the data is published has the following structure:

  ```
  $SYS/Broker/<brokerName>/StatisticsAccounting/<recordType>
  /<executionGroupLabel>/<messageFlowLabel>
  ```

  where recordType is set to Snapshot or Archive, and broker, execution group, and message flow names are specified according to the subscriber's requirements.
- smf - statistics reports are output as SMF type 117 records.

-r (Optional) This parameter applies only to archive data and specifies that archive data is to be reset.

This results in the clearing out of accounting and statistics data accumulated so far for this interval, and restarts collection from this point. All archive data for all flows in the execution group, or groups, is reset.

The archive interval timer is only reset if the **-v** option (*statistics archive interval*) of **mqsicreatebroker** or **mqsichangebroker** is non zero.

That is, the interval timer is set only if the internal interval notification mechanism is being used, and not an external method.

-s (Required) Specify that the command modifies snapshot accounting and statistics collection.

You must specify either **-a** or **-s**. If you do not specify one of these arguments you receive a warning message.

-t *ThreadData*
(Optional) Specify a string value to modify the collection of thread statistics data for a message flow Possible values are:

- none - exclude thread related data from the statistics
- basic - include thread related data in the statistics

Examples:
Using the command BIPCHMS:

- Turn on snapshot statistics for the message flow "myFlow1" in all execution groups and specify that the statistics are not to be partitioned according to accounting origin data:

  ```
  mqsichangeflowstats BrokerA -s -g -j -b none
  ```

- Turn off the collection of archive statistics for message flow "MyFlow1" in execution group "EGRP2" , and at the same time modify the granularity of data that is to be collected (when next activated) to include thread related data.

  ```
  mqsichangeflowstats BrokerA -a -e "EGRP2" -f MyFlow1 -c inactive -t basic
  ```

The following example uses the console form of the command. Turn on archive accounting for all the message flows in all the execution groups that belong to the broker and output the report as SMF records.

```
F VCP2BRK,CS A=YES,G=YES,J=YES,C=ACTIVE,O=SMF
```

Related reference:

"Syntax diagrams" on page 3677

"**mqsichangeflowstats** command" on page 3744
Use the **mqsichangeflowstats** command to control the accumulation of statistics about message flow operation.

mqsichangeflowuserexits command:

Use the **mqsichangeflowuserexits** command to set the list of active or inactive user exits. A list of active and a list of inactive user exits is maintained for each execution group and message flow. The effective state of user exits for a given flow is decided when the flow starts.

Supported operating systems:

- Windows
- Linux and UNIX systems
- z/OS. Run this command in one of two ways - as a console command, or by customizing and submitting the BIPCHUE utility; see "Contents of the broker PDSE" on page 3991

Purpose:
The order of precedence is message flow, execution group, then broker default. The active list takes precedence over the inactive list in the message flow and execution group settings.

If the state for a given user exit is not set for the message flow, its state is taken from the execution group setting. If its state is not set for the message flow or

execution group, it takes the default state which is implicitly inactive, or can be explicitly defined as active by the broker property *activeUserExits*, through the **mqsichangebroker** command.

If a particular user exit name is present in both the active and inactive lists for a message flow or execution group, the active list takes precedence and the user exit is active for that level. Therefore, if you want to change a user exit from active to inactive you must specify it as part of the inactive list, by using the **-i** flag and also remove it from the active list by re-specifying the new active list by using the **-a** flag.

When multiple exits are active for a given flow, they are invoked in a defined order. Those exits in the message flow's active list are invoked first in the order in which they were specified on the **-a** flag.

After those have been invoked, the exits in the execution group's active list (which were in neither the message flow's active nor inactive list) are invoked. These exits are invoked in the order in which they were specified on the **-a** flag.

Note, that to clear the user exit lists, double quotation marks must be used as an argument with the **-i** or **-a** flag, depending on which user exit list is to be cleared.

All user exits that are not mentioned in the execution group's or message flow's active or inactive list, but are in the broker's active list, are invoked in the order in which they were specified when the broker property *activeUserExits* was set.

If any of the user exits specified in either the active or inactive list are not registered for the target execution group, the command fails with a BIP8858 error.

After successful command completion, if a user exit becomes invalid, the following action is taken, depending on which list the user exit appeared in:

- If the user exit was specified in the message flow's active or inactive list, the flow fails to start and a BIP2315 message is written to system log.
- If the user exit was specified in the execution group's active or inactive list, the execution group fails to start and a BIP2314 message is written to system log.

A user exit might become invalid for one of the following reasons:

- The broker or execution group is restarted after you change the MQSI_USER_EXIT_PATH variable by removing the directory containing the user exit library.
- The broker or execution group is restarted after you change the *userExitPath* broker property by removing the directory containing the user exit library.
- The user exit library (or one of its dependencies) is removed, or the broker is unable to load it.

Select the appropriate link for details of this command on the platform, or platforms, that your enterprise uses:

- “mqsichangeflowuserexits command - Windows, Linux, and UNIX systems” on page 3753
- “mqsichangeflowuserexits command - z/OS” on page 3754

Authorization:
For information about platform-specific authorizations, see the following topics:

- “Security requirements for Linux and UNIX platforms” on page 3648

- "Security requirements for Windows systems" on page 3651
- "Security requirements for z/OS" on page 3655

If you have enabled broker administration security, you must also set up the authority detailed in "Tasks and authorizations for broker administration security" on page 3645.

Related concepts:

"User exits" on page 3015
A user exit is user-provided custom software, written in C, to track data passing through message flows.

Related tasks:

"Developing a user exit" on page 3114
Develop a user exit by declaring it, implementing its behavior, then compiling it.

"Deploying a user exit" on page 3116
Deploy your user exit to the broker.

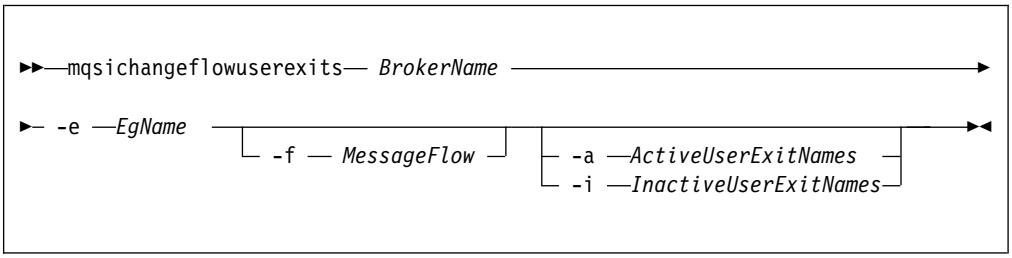
Related reference:

"**mqsireportflowuserexits** command" on page 3933
Use the **mqsireportflowuserexits** command to report the list of active and inactive user exits for the specified broker, execution group, or message flow.

mqsichangeflowuserexits command - Windows, Linux, and UNIX systems:

Use the **mqsichangeflowuserexits** command on Windows, Linux, and UNIX systems to set the list of active or inactive user exits. A list of active and a list of inactive user exits is maintained for each execution group and message flow. The effective state of user exits for a given flow is decided when the flow starts.

Syntax:



Parameters:

BrokerName
(Required). The name of the broker.

-a *ActiveUserExitNames*
(Optional). A list of the names, separated by colons, of the active user exits. These are the names registered by the user exits when they were loaded. If any of the user exits listed are not registered for the target execution group, then the command fails with a BIP8858 error.

-e *EgName*
(Required). The name of the execution group.

-f *MessageFlow*
(Optional). The name of the message flow.

If you supply this value, the user exit is changed for that message flow; if you do not, the user exit is set at the execution group level.

-i *InactiveUserExitNames*
(Optional). A list of the names, separated by colons, of the inactive user exits. These are the names registered by the user exits when they were loaded. If any of the user exits listed are not registered for the target execution group, then the command fails with a BIP8858 error.

Examples:
Setting active exits at flow level

```
mqsichangeflowuserexits MB7BROKER -e default -f myFlow -a exit2
```

Setting inactive exits at flow level

```
mqsichangeflowuserexits MB7BROKER -e default -f myFlow -i exit1
```

Setting active exits at execution group level:

```
mqsichangeflowuserexits MB7BROKER -e default -a exit3,exit1
```

Setting inactive exits at execution group level:

```
mqsichangeflowuserexits MB7BROKER -e default -1 exit2
```

Changing exit1 to inactive and leaving exit2 active at flow level (A command had previously been issued with "-a exit1:exit2" to set them both active):

```
mqsichangeflowuserexits MB7BROKER -e default -f myFlow -i exit1 -a exit2
```

Clearing the active user exit list at flow level:

```
mqsichangeflowuserexits MB7BROKER -e default -f myFlow -a ""
```

Related reference:

"Syntax diagrams" on page 3677

"**mqsichangeflowuserexits** command" on page 3751

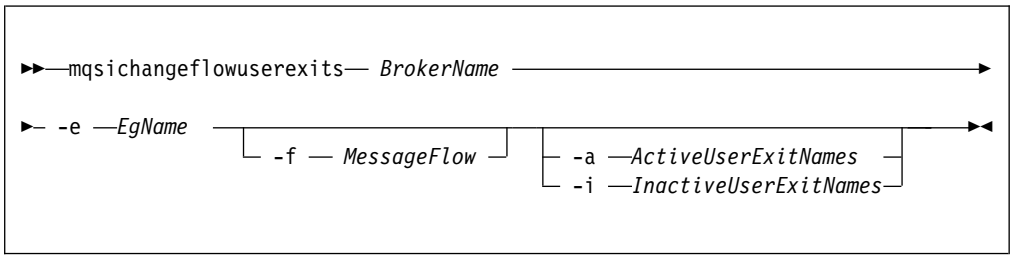
Use the **mqsichangeflowuserexits** command to set the list of active or inactive user exits. A list of active and a list of inactive user exits is maintained for each execution group and message flow. The effective state of user exits for a given flow is decided when the flow starts.

mqsichangeflowuserexits command - z/OS:

Use the **mqsichangeflowuserexits** command on z/OS to set the list of active or inactive user exits. A list of active and a list of inactive user exits is maintained for each execution group and message flow. The effective state of user exits for a given flow is decided when the flow starts.

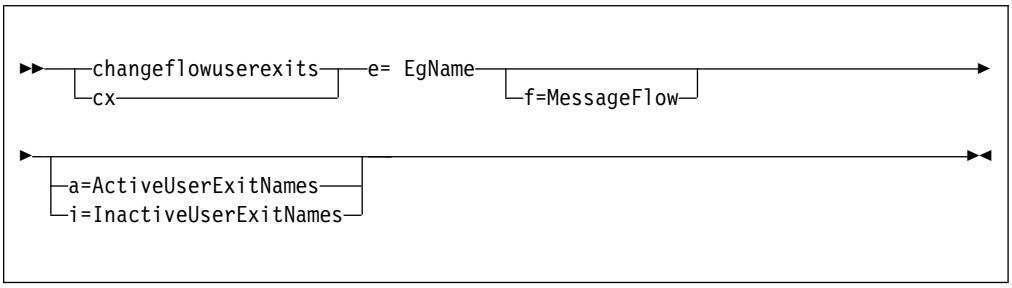
Syntax:

z/OS command - BIPCHUE:



z/OS console command:

Synonym: cx



Parameters:

BrokerName
(Required). The name of the broker.

-a *ActiveUserExitNames*
(Optional). A list of the names, separated by colons, of the active user exits. These are the names registered by the user exits when they were loaded. If any of the user exits listed are not registered for the target execution group, then the command fails with a BIP8858 error.

-e *EgName*
(Required). The name of the execution group.

-f *MessageFlow*
(Optional). The name of the message flow.

If you supply this value, the user exit is changed for that message flow; if you do not, the user exit is set at the execution group level.

-i *InactiveUserExitNames*
(Optional). A list of the names, separated by colons, of the inactive user exits. These are the names registered by the user exits when they were loaded. If any of the user exits listed are not registered for the target execution group, then the command fails with a BIP8858 error.

Examples:
Setting active exits at flow level

```
mqsichangeflowuserexits MB7BROKER -e default -f myFlow -a exit2
```

Setting inactive exits at flow level

```
mqsichangeflowuserexits MB7BROKER -e default -f myFlow -i exit1
```

Setting active exits at execution group level

```
mqsichangeflowuserexits MB7BROKER -e default -a exit3,exit1
```

Setting inactive exits at execution group level

```
mqsichangeflowuserexits MB7BROKER -e default -1 exit2
```

Changing exit1 to inactive and leaving exit2 active at flow level (A command had previously been issued with "-a exit1:exit2" to set them both active)

```
mqsichangeflowuserexits MB7BROKER -e default -f myFlow -i exit1 -a exit2
```

Related reference:
"Syntax diagrams" on page 3677

"**mqsichangeflowuserexits** command" on page 3751
Use the **mqsichangeflowuserexits** command to set the list of active or inactive user exits. A list of active and a list of inactive user exits is maintained for each execution group and message flow. The effective state of user exits for a given flow is decided when the flow starts.

mqsichangeproperties command:

Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

Supported platforms:

- Windows systems.
- Linux and UNIX systems.
- z/OS. Run this command by customizing and submitting the BIPCHPR utility; see "Contents of the broker PDSE" on page 3991.

Purpose:
Use the **mqsichangeproperties** command to change properties that are associated with a broker:

- Properties that affect the whole broker; for example, an HTTP listener or Service Federation Management
- Properties that affect one or more execution groups; for example, the broker registry
- Properties that affect a configurable service; for example, a JMS provider

You can also use the WebSphere Message Broker Explorer or the Administration API for WebSphere Message Broker (also known as the CMP API) to change properties.

Use the **mqsireportproperties** command to view properties that are associated with a broker.

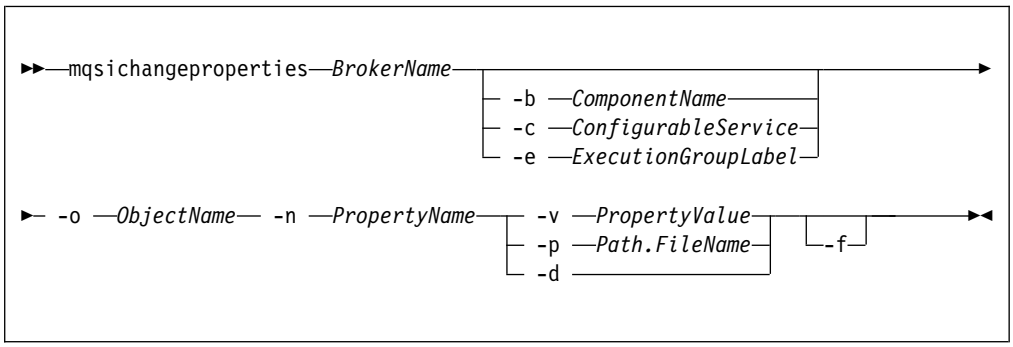
Usage notes:

Before you run the **mqsichangeproperties** command, ensure that the broker is running.

If you change one or more values, stop and start the broker again for the change to take effect.

When a message flow that includes HTTP nodes is started, the broker starts the HTTP and HTTPS listeners, unless this listener has been disabled.

Syntax:



Parameters:

BrokerName
(Required) The name of the broker to modify. This parameter must be the first parameter.

-b *ComponentName*
(Optional) The name of the component. Valid values are *httplistener*, *securitycache*, and *servicefederation*. The properties associated with these components affect the whole broker.

-c *ConfigurableService*
(Optional) The type of configurable service. The type is predefined; for example, JMSProviders. You can define additional services of all defined types by using the WebSphere Message Broker Explorer or the **mqsicreateconfigurableservice** command.

The valid resource types are listed in "Configurable services properties" on page 3766.

-e *ExecutionGroupLabel*
(Optional) The label of the execution group for which you want to change properties.

-o *ObjectName*
(Required) The name of the broker resource for which you want to change the properties.

You must also specify **-b**, **-e**, or **-c** with **-o**, except if you specify the object name `BrokerRegistry`, or the object name `ComIbmJVMManager` to change a property related to the heap size.

For compatibility with previous versions, you can also specify the value `ComIbmXmlParserFactory` for the **ObjectName**.

-n *PropertyName*
(Required) The name of the property to be changed. The available property names differ according to the component or configurable service that you have specified. All property names start with a lowercase character.

The property names for predefined configurable services are described in "Configurable services properties" on page 3766.

If you specify a Configurable Service of type UserDefined, the PropertyName specified is created if it does not exist.

-v *PropertyValue*
(Required) The value that is assigned to the property that is specified by the **-n** parameter.

You can specify more than one property name together with a corresponding value, by using commas as separators, if you use a valid value for the corresponding property; for example, `-n Name1,Name2 -v Value1,Value2`.

Do not leave a space after each comma in the list of names and corresponding values. Use "" to specify an empty *PropertyValue* string.

If the property value contains a comma, enclose the value with escaped double quotation marks (\" and \"); for example, `-n Name1,Name2 -v Value1,\"Value21,Value22\"`.

UNIX On UNIX, if the **-v** parameter contains a semicolon (;), enclose the entire string in quotation marks, as shown in the following example:

```
mqsichangeproperties MB7BROKER -c JDBCProviders -o DB2EXTRA -n connectionUrlFormat
-v "jdbc:db2://[serverName]:[portNumber]/[databaseName]:user=[user];password=[password];"
```

If you set the **-c** parameter to `EISProviders` or `JMSProviders`, and the **-n** parameter to `jarsURL`, the expected value is a URL that specifies the file location of the EIS or JMS provider JAR files, but omits the `file://` part of the URL. (On Windows, the file location cannot be a mapped network drive on a remote Windows computer; the directory must be local or on a Storage Area Network (SAN) disk.)

-p *Path.FileName*
(Optional) The location and name of a file from which the command reads the property value. Use this command as an alternative to **-v** where the value of the property is complex and is defined in a file, such as an XML file.

The following conditions apply to the use of this parameter:

- **-p** can be used to set a single property only. Therefore, the **-n** parameter must have a single property name, not a comma-separated list.
- White space characters (including line feed, carriage return, and end of file characters) read from the file are preserved by the command.

Use this parameter for policy sets and bindings.

Use this parameter for monitoring profiles; the XML file must conform to the monitoring profile schema.

-d (Optional) This parameter is valid only if you specify a configurable service of type UserDefined. If the property that you specify on the **-n** parameter exists, it is deleted.

If you specify this parameter for other configurable services or components, an error is generated.

-f (Optional) This parameter is valid only if you specify a property of an execution group level object. If the property that you specify on the **-n** parameter exists, its value is changed to the *PropertyValue* specified by the **-v** parameter.

You can use this parameter only when the execution group is in the stopped state. You are advised to use this parameter only under the direction of IBM support to recover from execution group startup failures.

If you specify this parameter for other configurable services or components, an error is generated.

For detailed information about valid components, configurable services, object names, properties, and values, select the appropriate topic:

- "Broker registry object parameter values" on page 3765
- "Configurable services properties" on page 3766
- "Content based filtering component parameter values" on page 3805
- "Execution group HTTP listener parameters (SOAP and HTTP nodes)" on page 3805
- "Broker-wide HTTP listener parameters" on page 3809
- "JVM parameter values" on page 3813
- "Parameter values for the securitycache component" on page 3815
- "Parameter values for the servicefederation component" on page 3816

- "ServiceFederationManager object property values" on page 3818

Authorization:
For information about platform-specific authorizations, see the following topics:
- "Security requirements for Linux and UNIX platforms" on page 3648
- "Security requirements for Windows systems" on page 3651
- "Security requirements for z/OS" on page 3655

If you have enabled broker administration security, you must also set up the authority detailed in "Tasks and authorizations for broker administration security" on page 3645.

Examples:
Always enter the command on a single line; in some examples, line breaks have been added to enhance readability.

Changes to broker components

The following examples specify the **-b** parameter to identify a particular broker component.

- Disable the broker-wide HTTP listener:

```
mqsichangeproperties MB7BROKER -b httplistener -o HTTPListener
   -n startListener -v false
```

- Enable the broker-wide HTTP listener:

```
mqsichangeproperties MB7BROKER -b httplistener -o HTTPListener
   -n startListener -v true
```

- Enable the HTTPSConnector for the HTTP nodes deployed to the specified broker that use the broker-wide listener:

```
 mqsichangeproperties MB7BROKER -b httplistener -o HTTPListener
   -n enableSSLConnector -v true
```

- Change the default SSL protocol from SSLv3 to TLS for the HTTP nodes that are deployed to the specified broker:

```
mqsichangeproperties MB7BROKER  -b httplistener -o HTTPSConnector
   -n sslProtocol -v TLS
```

- Change the securitycache timeout to 200 seconds:

```
mqsichangeproperties MB7BROKER -b securitycache -o SecurityCache
   -n cacheTimeout -v 200
```

- Enable Service Federation Management (SFM) and specify a host name and contact name:

```
mqsichangeproperties MB7BROKER -b servicefederation -o scmp
-n enabled,hostname,contactName -v true,mbhost.ibm.com,John.Smith
```

- Change the JVM heap size for the broker:

```
mqsichangeproperties MB7BROKER -o ComIbmJVMManager
-n jvmMaxHeapSize -v size_in_bytes
```

Changes to properties that are associated with execution groups

The following examples include the **-e** parameter to specify the execution group to change.

- Configure the execution group so that all deployed HTTP nodes use the embedded listener:

```
mqsichangeproperties MB7BROKER -e exgroup1 -o ExecutionGroup
 -n httpNodesUseEmbeddedListener -v true
```

If you disable the broker-wide listener (as shown in a previous example), you do not have to change the execution group configuration as shown in this command; if the broker-wide listener is not active, all execution groups use the embedded listener by default for all HTTP message handling.

- Configure the execution group properties so that deployed HTTP nodes do not use the embedded listener:

```
mqsichangeproperties MB7BROKER -e exgroup1 -o ExecutionGroup
 -n httpNodesUseEmbeddedListener -v false
```

- Set the port number when changing properties for execution groups:

```
mqsichangeproperties MB7BROKER -e exgroup1 -o HTTPSConnector
  -n explicitlySetPortNumber -v 7777
```

- Set the list of ciphers allowed by the HTTPS listener of the execution group:

```
mqsichangeproperties MB7BROKER -e default
-o HTTPSConnector -n ciphers
-v \"SSL_RSA_WITH_RC4_128_MD5,SSL_RSA_WITH_RC4_128_SHA\"
```

  On distributed systems, if the property value contains a comma, enclose the value with escaped double quotation marks (\" and \"); for example:

```
-v \"SSL_RSA_WITH_RC4_128_MD5,SSL_RSA_WITH_RC4_128_SHA\"
```

  On z/OS, if the property value contains a comma, enclose the value with double quotation marks (" and "); for example:

```
-v "SSL_RSA_WITH_RC4_128_MD5,SSL_RSA_WITH_RC4_128_SHA"
```

- Set the JVM port number to activate message flow debugging:

```
mqsichangeproperties MB7BROKER -e exgroup1 -o ComIbmJVMManager
 -n jvmDebugPort -v 8018
```

- Change the JVM heap size for the execution group:

```
mqsichangeproperties MB7BROKER -e exgroup1 -o ComIbmJVMManager
-n jvmMaxHeapSize -v size_in_bytes
```

- Set the type of server keystore:

```
mqsichangeproperties MB7BROKER -e AddressSampleProvider
 -o ComIbmJVMManager
 -n keystoreType -v JKS
```

- Set a host name and port to be used in the endpoint address of SFM proxies created in an execution group that is to be used as an SFM Connectivity Provider:

```
mqsichangeproperties MB7BROKER -e exgroup1
 -o ServiceFederationManager
 -n proxyURLHostName,port,securePort -v mbhost.ibm.com,8811,8844
```

- Set the coordination queue manager to *QM2* for execution group myExecutionGroup in broker MB7BROKER:

```
mqsichangeproperties MB7BROKER -e myExecutionGroup
-o FTEAgent -n coordinationQMgr -v QM2
```

  Note: After running this command, you must reload the execution group for the change to take effect.

- Configure the expiry compensation to prevent messages from immediately expiring when put:

```
mqsichangeproperties MB7BROKER -e default
-o ComIbmMQConnectionManager -n expiryCompensation -v option
```

  where *option* specifies if the expiry compensation is active or not:

  – *false* This option is the default value. No adjustment is applied to the expiry

– *true* This option causes the expiry to be reduced by the amount of time that the message has spent in the flow at the time that the MQPUT call is made by the MQOutput Node.

Changes to the BrokerRegistry object

- Set the HTTPConnector Port Range in the broker registry:

```
mqsichangeproperties MB7BROKER -o BrokerRegistry
 -n httpConnectorPortRange -v 7777-8888
```

Changes to configurable services

The following examples include the **-c** parameter to specify the type of configurable service to change.

- Change an Aggregation configurable service:

```
mqsichangeproperties MB7BROKER -c Aggregation
 -o myAggregationService
 -n timeoutSeconds -v 120
```

  This command changes all nodes that are configured to use the myAggregationService configurable service by setting the timeout of an aggregation to 120 seconds.

- Change a CICSConnection configurable service:

```
mqsichangeproperties MB7BROKER -c CICSConnection -o myCICSConnectionService
 -n connectionTimeoutSecs -v 120
```

  This command changes the CICSRequest node that is configured to use the myCICSConnectionService configurable service by setting the Connection timeout to 120 seconds.

- Change a Collector configurable service:

```
mqsichangeproperties MB7BROKER -c Collector
 -o myCollectorService
 -n collectionExpirySeconds -v 120
```

  This command changes all nodes that are configured to use the myCollectorService configurable service by setting the expiry time of collections to 120 seconds.

- Change the location of the object reference for all nodes that are configured to use the myCORBAService configurable service. After you run this command, all nodes look for the object reference name "Europe.region/Market.object" in the naming service on the local host on port 2809.

```
mqsichangeproperties MB7BROKER -c CORBA -o myCORBAService -n namingService,objectReferenceName
```

- Change an EmailServer configurable service:

```
mqsichangeproperties MB7BROKER -c EmailServer -o
myEmailConfigurableServiceName -n securityIdentity -v
myNewSecurityIdentityObjectName
```

  This command changes the EmailInput node that is configured to use the myEmailConfigurableServiceName configurable service by renaming the securityIdentity object to myNewSecurityIdentityObjectName.

- Change the *myCDServer* configurable service to use a specific staging directory `-/tmp/cdtransfer`.

  Note that you can use any string for the -o *Objectname* parameter.

```
mqsichangeproperties MB7BROKER -c CDServer -o myCDServer -n brokerPathToStagingDir
 -v /tmp/cdtransfer
```

- Make the JAR files and shared libraries available to the WebSphere Adapter for SAP:

```
mqsichangeproperties MB7BROKER -c EISProviders -o SAP
 -n jarsURL,nativeLibs
      -v c:\sapjco\jars,c:\sapjco\bin
```

- For the FtpServer configurable service called TEST1, change the protocol to SFTP and change the server name to winlnx58:

```
mqsichangeproperties MB7BROKER -c FtpServer -o TEST1
-n protocol,serverName,scanDelay,remoteDirectory,securityIdentity,
cipher,compression,strictHostKeyChecking
-v  SFTP,winlnx58,30,.,chbatey,blowfish-cbc,9,no
```

- Change all the nodes that are configured to use the *myIMSConnectService* configurable service. After you run this command, all nodes connect to the production system (`production.ims.ibm.com`) instead of the test system (`test.ims.ibm.com`).

```
mqsichangeproperties MB7BROKER -c IMSConnect
 -o myIMSConnectService
 -n Hostname -v production.ims.ibm.com
```

- Update the security identity for the JDBCProvider service for Oracle:

```
mqsichangeproperties MB7BROKER -c JDBCProviders -o Oracle

 -n securityIdentity -v OracleDSN
```

  OracleDSN is the DSN with which you have associated a user ID and password using the **mqsisetdbparms** command.

- Change the location of the JAR files for the IBM WebSphere MQ JMS client:

```
mqsichangeproperties MB7BROKER -c JMSProviders -o WebSphere_MQ
      -n jarsURL -v file://D:\SIBClient\Java
```

- Change the value of the properties `proprietaryAPIAttr2` and `proprietaryAPIAttr3` for a user-defined JMS provider configurable service definition called BEA_Weblogic, where the properties represent the URL of the BEA WebLogic bindings and the DNS name of the JMS server:

```
mqsichangeproperties MB7BROKER -c JMSProviders -o BEA_Weblogic
      -n proprietaryAPIAttr2, proprietaryAPIAttr3
      -v t3://9.20.94.16:7001,BEAServerName
```

- Change the value of the `jndiEnvironmentParms` property in the definition of a user-defined JMS provider configurable service called *myJMSprovider*:

```
mqsichangeproperties MB7BROKER -c JMSProviders -o myJMSprovider
      -n jndiEnvironmentParms
      -v domainName=myDomain;timeout=6000
```

- Set the properties of monitoring profile 'mp1' by using the contents of file `mp1.xml`:

```
mqsichangeproperties MB7BROKER -c MonitoringProfiles -o mp1
-n profileProperties -p mp1.xml
```

- Change all connections that are used by the adapter *myPeopleSoftAdapter.outadapter*. After you run this command, all adapters connect to the production system (`my.peoplesoft.production.com`) instead of the test system (`my.peoplesoft.qa.com`).

```
mqsichangeproperties MB7BROKER -c PeopleSoftConnection

 -o myPeopleSoftAdapter.outadapter -n hostName
 -v "my.peoplesoft.production.com"
```

- When you create a policy set, its properties are always set to default values. Use this command to change the property values.

  Import a policy set to a broker from a file:

```
mqsichangeproperties MB7BROKER -c PolicySets -o Policy_2
      -n ws-security -p policyset.xml
```

This command reads file policyset.xml and sets its contents as Policy_2 in broker MB7BROKER. The command is used to move policy sets between brokers, or to restore from a backup.

- Import a policy set bindings to a broker from a file:

```
mqsichangeproperties MB7BROKER -c PolicySetBindings -o Bindings_2
      -n ws-security -p bindings.xml
```

This command reads file bindings.xml and sets its contents as Bindings_2 in broker MB7BROKER. The command is used to move policy set bindings between brokers, or to restore from a backup.

- Change a Resequence configurable service:

```
mqsichangeproperties MB7BROKER -c Resequence -o myResequenceService
 -n missingMessageTimeoutSeconds -v 120
```

This command changes all nodes that are configured to use the myResequenceService configurable service, by setting them to propagate message sequences that have missing messages after waiting for 120 seconds.

- Change all connections that are used by the adapter *mySAPAdapter.outadapter*. After you run this command, all adapters connect to the production system (production.sap.ibm.com) instead of the test system (test.sap.ibm.com).

```
mqsichangeproperties MB7BROKER -c SAPConnection -o mySAPAdapter
-n applicationServerHost -v production.sap.ibm.com
```

- Change a SecurityProfiles configurable service to enable identity mapping using the WS-Trust v1.3 STS provider interface to Tivoli Federated Identity Manager (TFIM) V6.2:

```
mqsichangeproperties MB7BROKER -c SecurityProfiles
  -o TFIMv62MapSecProfile
  -n mapping,mappingConfig
  -v "WS-Trust v1.3 STS",
   http://wstrusthost1.ibm.com:9080/TrustServerWST13/services/RequestSecurityToken
```

- Change the connection timeout for queries issued by the WebSphere Service Registry and Repository nodes to 180 seconds:

```
mqsichangeproperties MB7BROKER -c ServiceRegistries -o DefaultWSRR
      -n connectionTimeout -v 180
```

- Change the connection that is used by the adapter *myAdapter.outadapter*. After you run this command, all message flows in all execution groups that use this adapter connect to the production system (my.siebel.production.com) instead of the test system (my.siebel.qa.com). If you are using different adapters in the message flow, run the **mqsichangeproperties** command for each named adapter.

```
mqsichangeproperties MB7BROKER -c SiebelConnection
 -o mySiebelAdapter.outadapter
 -n connectString
 -v "siebel://my.siebel.production.com/SBA_80/SSEObjMgr_enu"
```

- Update the TCPIPClient configurable service so that it does not create client connections until they are required:

```
mqsichangeproperties MB7BROKER -c TCPIPClient
 -o ClientPort1452HostnameJsmith
      -n MinimumConnections -v 0
```

- Change the connection expiry time on TCPIPServer connections to 30 seconds:

```
mqsichangeproperties MB7BROKER -c TCPIPServer -o ServerPort1452
      -n ExpireConnectionSec -v 30
```

- Change the TCPIPClient configurable service to use alternative addresses:

```
mqsichangeproperties MB7BROKER -c TCPIPClient -o MyTCPIPClient
      -n AlternativeAddresses -v smith6:1111;jones7
```

- Change the TCPIPClient configurable service to use a connection pool other than the main pool:

```
mqsichangeproperties MB7BROKER -c TCPIPClient -o MyTCPIPClient
      -n UseUniqueConnectionPool -v true
```

- Change a TCPIPClient or TCPIPServer configurable service to use SSL:

  "Configuring TCP/IP client nodes to use SSL" on page 551

  "Configuring TCP/IP server nodes to use SSL" on page 553

- Change a Timer configurable service:

```
mqsichangeproperties MB7BROKER -c Timer -o myTimerService
 -n timeoutIntervalSeconds -v 1
```

  This command changes all TimeoutNotification nodes that are configured to use the myTimerService configurable service, by configuring them to generate events every second if the node is configured in Automatic mode.

- Select content based filtering on the default execution group:

```
mqsichangeproperties MB7BROKER -e default -o ContentBasedFiltering
  -n cbfEnabled -v true
```

- Select content based filtering on the default execution group and increase the number of evaluation threads to five:

```
mqsichangeproperties MB7BROKER -e default -o ContentBasedFiltering
  -n validationThreads -v 5
```

- Change a JavaClassLoader service:

```
mqsichangeproperties MB7BROKER -c JavaClassLoader
  -o myJavaClassLoader
  -n sharedJarPath
  -v /var/app2/jars
```

- Add a property to a configurable service of type UserDefined:

```
mqsichangeproperties MB7BROKER -c UserDefined
  -o MyService1 -n VerifyRequestTimeout -v 60
```

- Delete a property of a configurable service of type UserDefined:

```
mqsichangeproperties MB7BROKER -c UserDefined
  -o HTTP_Timeout -n VerifyRequestTimeout -d
```

Related concepts:

"Using the WebSphere Message Broker Explorer to work with configurable services" on page 644
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view, add, modify and delete configurable services.

"HTTP listeners" on page 1589
You can choose between broker-wide listeners and execution group (embedded) listeners to manage HTTP messages in your HTTP flows. Learn about the two types of listener, how ports are assigned to them, and how you can switch from one to the other for individual execution groups.

Related tasks:

Chapter 8, "Administering brokers and broker resources," on page 899
Administering brokers and associated broker resources includes the tasks that you perform frequently to activate and manage those resources. Choose the method you prefer to administer your brokers and associated resources.

"Setting up a JDBC provider for type 4 connections" on page 684
Use the **mqsicreateconfigurableservice** or the **mqsichangeproperties** command to configure a JDBC provider service.

Related reference:

"Syntax diagrams" on page 3677

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

Related information:

Administration API for WebSphere Message Broker (CMP API)

Broker registry object parameter values:

Select the names of the properties and values that you want to change for the broker registry object.

To change these properties, you must specify the broker name and the **ObjectName** `BrokerRegistry`. You do not have to specify **-b**, **-c**, or **-e** to change properties associated with this object.

The following properties and values are valid:

-n brokerKeystorePass
: The password used to access the server certificate from the specified keystore file.

  Set the password by using **mqsisetdbparms** when the broker is stopped.

  - Value type - string
  - Initial value - brokerKeystore::password

-n brokerKeystoreFile
: The path to the keystore file where the server certificate, which is to be loaded, has been stored. The HTTP listener expects a file with the default name `.keystore` in the home directory of the user who started the broker.

  - Value type - string
  - Initial value - *default value* (described previously)

-n brokerKeystoreType
: The type of keystore file to be used for the server certificate.

  - Value type - string
  - Initial value - `JKS`

-n brokerTruststorePass
: The password used to access the server certificate from the specified keystore file.

  Set the password by using **mqsisetdbparms** when the broker is stopped.

  - Value type - string
  - Initial value - brokerKeystore::password

-n brokerTruststoreFile
The path to the keystore file where the server certificate, which is to be loaded, has been stored. The HTTP listener expects a file with the default name `.keystore` in the home directory of the user who started the broker.

- Value type - string
- Initial value - *default value* (described previously)

-n brokerTruststoreType
The type of keystore file to be used for the server certificate.

- Value type - string
- Initial value - `JKS`

-n httpConnectorPortRange
The numeric range of ports available to the HTTPConnector object that is associated with SOAP nodes.

- Value type - integer
- Initial value - `7800-7842`

-n httpsConnectorPortRange
The numeric range of ports available to the HTTPSConnector object that is associated with SOAP nodes.

- Value type - integer
- Initial value - `7843-7884`

See the "**mqsichangeproperties** command" on page 3756 for examples of how to change `BrokerRegistry` parameters. Other examples are provided for particular tasks:

"Viewing and setting keystore and truststore runtime properties at broker level" on page 780

"Accessing a secure WSRR repository" on page 1884

Related tasks:

"Viewing and setting keystore and truststore runtime properties at broker level" on page 780
Configure the message broker to refer to a keystore, a truststore, or both, before deploying any message flows that require policy set or bindings for signature, encryption, or X.509 Authentication.

Related reference:

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"Execution group HTTP listener parameters (SOAP and HTTP nodes)" on page 3805
Select the resources and properties associated with the HTTPInput, HTTPReply, SOAPInput, SOAPReply, and SOAPAsyncResponse nodes that you want to change.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

Configurable services properties:

The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

Use the WebSphere Message Broker Explorer to view, create, and modify configurable services. You can also delete custom-named configurable services; however, you cannot delete IBM-defined configurable services. For more information, see "Using the WebSphere Message Broker Explorer to work with configurable services" on page 644.

You can create your own configurable services by using the **mqsicreateconfigurableservice** command. For examples of how to use this command to create each configurable service, see "**mqsicreateconfigurableservice** command" on page 3849. You can also delete services that you have created by using the **mqsideleteconfigurableservice** command. For examples of how to use this command to delete each configurable service, see "**mqsideleteconfigurableservice** command" on page 3866.

If you want to change the properties for a configurable service, use the **mqsichangeproperties** command and specify the broker name and **-c** *ConfigurableService*. Set the **ObjectName** to the name of the service for which you want to change properties; the name can be a predefined name (one of the names shown for the service type in the following tables), or the name of a configurable service that you have created yourself. For examples of how to use this command, see "**mqsichangeproperties** command" on page 3756.

To display one or more of the defined configurable services, use the **mqsireportproperties** command. The following example displays all configurable services that are available for a single broker:

```
mqsireportproperties brokerName -c AllTypes -o AllReportableEntityNames -r
```

Specify the appropriate parameters from those parameters shown in the following tables on the **mqsichangeproperties** and **mqsireportproperties** commands:

- Specify **-c** to identify the configurable service type.
- Specify **-o** to identify the name of the configurable service object.
- Specify **-n** to identify the properties of the service.
- Specify **-v** to identify the values of the properties specified.

Follow the link to the configurable service that you want to use to view the available properties:

Aggregation configurable service

Supplied configurable services that are created for each broker	Properties for each configurable service that is defined	Description of properties
None	queuePrefix	The queue prefix used to specify the storage queues that are generated for use by the aggregation nodes. This property is optional. Five queues are required: • SYSTEM.BROKER.AGGR.*QueuePrefix*.CONTROL • SYSTEM.BROKER.AGGR.*QueuePrefix*.REPLY • SYSTEM.BROKER.AGGR.*QueuePrefix*.REQUEST • SYSTEM.BROKER.AGGR.*QueuePrefix*.UNKNOWN • SYSTEM.BROKER.AGGR.*QueuePrefix*.TIMEOUT The prefix can contain any characters that are valid in a WebSphere MQ queue name, but must be no longer than eight characters and must not begin or end with a period (.). For example, `SET.1` is valid, but `.SET1` and `SET1.` are invalid. Multiple configurable services can use the same queue prefix. If no queue prefix is specified, the aggregation nodes use the default queues that are generated when the broker is created: • SYSTEM.BROKER.AGGR.CONTROL • SYSTEM.BROKER.AGGR.REPLY • SYSTEM.BROKER.AGGR.REQUEST • SYSTEM.BROKER.AGGR.UNKNOWN • SYSTEM.BROKER.AGGR.TIMEOUT
	timeoutSeconds	The timeout value used by the AggregateControl node to set the expiry time (in seconds) of an aggregation. The value can be any positive integer. This property is optional; if it is not set, the value set on the node is used. The value of this property overrides the value set on the node, but it can be overridden by a value in the local environment.
	timeoutThreads	The number of threads used by the AggregateReply node to process expired messages. The value can be any positive integer. This property is optional. When the property is set, a unique `queuePrefix` property must be set on the Aggregation configurable service, and this Aggregation configurable service must be used only in a single execution group. If the property is not set, a single thread is used to process expired messages.

CICSConnection configurable service

Supplied configurable services that are created for each broker	Properties for each configurable service that is defined	Description of properties
None	cicsServer	The value that is used by the CICSRequest node to determine the connection to CICS Transaction Server for z/OS. One of the following connection methods can be specified: **A direct connection to CICS (two-tier):** If making a direct connection from WebSphere Message Broker to CICS, the `cicsServer` property must contain the URL of the CICS Transaction Server for z/OS region where the program exists. The URL allows you to specify a protocol, host name, and port number, which is the minimum information you need to connect to the CICS target region. The URL must be made up of the following structure: *protocol*`://`*hostname*`:`*port* Where: • *protocol* can be `tcp` or `ssl`. • *hostname* is the Internet Protocol version 4 (IPv4) TCP/IP address or DNS-resolvable host name of the CICS host. • *port* is the port number of the TCPIPSERVICE listener in CICS that is listening for IP InterCommunications (IPIC) protocol requests over TCP/IP or Secure Sockets Layer (SSL) protocol. You can enter an integer in the range 1 - 65535. For example: `tcp://mycicsregion.com:12345` or `ssl://mycicsregion.com:56789`. You can obtain the *hostname* and *port* values from the IPIC TCPIPSERVICE definition in the CICS target region. **A connection to CICS through a CICS Transaction Gateway (three-tier):** If you make connections to CICS through a Transaction Gateway (CICS TG), • The `gatewayURL` property must contain the URL for the CICS TG to use for the connection, and • The `cicsServer` property must contain the *CICS server definition* that you have specified in the CICS TG (`.INI`) initialization file. For example, the property can specify the particular CICS region that you want the transaction gateway to connect to. For more information about the two-tier and three-tier connection models, see “CICS Transaction Server for z/OS overview” on page 2173 for a high-level overview, or “CICS Transaction Server for z/OS two-tier connectivity” on page 2177 and “CICS Transaction Server for z/OS three-tier connectivity” on page 2181 for detailed conceptual information.
	clientApplid	The APPLID by which WebSphere Message Broker is known to the CICS region. This is not the APPLID of the CICS region. You can enter a maximum of 8 characters. This property is optional.
	clientQualifier	The APPLID qualifier by which WebSphere Message Broker is known to the CICS region. You can enter a maximum of 8 characters. This property is optional.
	securityIdentity	The name of the security identity object that is created and

Collector configurable service

Supplied configurable services that are created for each broker	Properties for each configurable service that is defined	Description of properties
None	queuePrefix	The queue prefix used to specify the storage queues that are generated for use by the Collector node. This property is optional. Two queues are required: • SYSTEM.BROKER.EDA.*QueuePrefix*.EVENTS • SYSTEM.BROKER.EDA.*QueuePrefix*.COLLECTIONS The prefix can contain any characters that are valid in a WebSphere MQ queue name, but must be no longer than eight characters and must not begin or end with a period (.). For example, `SET.1` is valid, but `.SET1` and `SET1.` are invalid. Multiple configurable services can use the same queue prefix. If no queue prefix is specified, the Collector node uses the default queues that are generated when the broker is created: • SYSTEM.BROKER.EDA.EVENTS • SYSTEM.BROKER.EDA.COLLECTIONS These default queues are also used by the Resequence node.
	collectionExpirySeconds	The value used by the Collector node to set the expiry time (in seconds) of a collection. The value can be any positive integer. This property is optional; if it is not set, the value set on the node is used. The value of this property (if it is set) overrides the value set on the node.

CDServer configurable service

For information about FtpServer configurable service properties, see "*CDServer* configurable service properties" on page 3798.

CORBA configurable service

Supplied configurable services that are created for each broker	Properties for each configurable service that is defined	Description of properties
None	namingService	The host name of the naming service from which to get the object reference. The format of this value is host:port, where port is optional. For example, localhost:2809. You can obtain this value from the administrator of the CORBA application that you are calling.
	objectReferenceName	The name of the reference to the object in the naming service. You can obtain this value from the CORBA server that you are calling. For more information about how to specify the object reference name, see "CORBA naming service" on page 2154.

EmailServer configurable service

Supplied configurable services that are created for each broker	Properties for each configurable service that is defined	Description of properties
None	serverName	The URL of the email server where the email messages exist for retrieving. The URL allows you to specify a protocol, host name, and port number, which is the minimum information you need to connect to the email server. The URL must be made up of the following structure: *protocol*://*hostname*:*port* Where: • *protocol* can be pop3 or imap. • *hostname* is the Internet Protocol version 4 (IPv4) TCP/IP address or DNS-resolvable host name of the email host. • *port* is the port number that the email server is listening on for connections over Post Office Protocol 3 (POP3) or Internet Message Access Protocol (IMAP). You can enter an integer in the range 1 - 65535. For example: pop3://myemailserver.com:12345 or imap://myemailserver.com:56789. There is no default value for this property, however this property is mandatory, and therefore must be configured with a URL. You can obtain the *hostname* and *port* values from the email server or email server administrator.
	securityIdentity	The name of the security identity object that is created and configured by the **mqsisetdbparms** command, which contains the user ID and password to be used by the broker to authenticate the connection to the email server. Use the **mqsisetdbparms** command to set the security identity user ID and password to be accessed by the broker. This property is optional. There is no default value for this property. If a value is not entered, the security identity object is taken from the Security identity property value that is configured on the EmailInput node. For more information about email server security identity support, see "**mqsisetdbparms** command" on page 3954.

EISProviders configurable service

Supplied configurable services that are created for each broker	Properties for each configurable service that is defined	Description of properties
PeopleSoft SAP Siebel Twineball	jarsURL	A URL that specifies the file location of the EIS provider JAR files. Omit `file://` from the URL. On Windows, the file location cannot be a mapped network drive on a remote Windows computer; the directory must be local or on a storage area network (SAN) disk. If you do not set the **-n** parameter on the **mqsicreateconfigurableservice** command, the default location for the EIS provider JAR files is the broker shared-classes directories.
	nativeLibs	The file location of any libraries that the EIS provider owns. If you do not set the **-n** parameter on the **mqsicreateconfigurableservice** command, the default location for any libraries that the EIS provider owns, is the broker LILPATH.
Siebel	siebelPropertiesURL	A URL that specifies the file location of a `siebel.properties` file. To connect to a Siebel server in a clustered environment, use the `siebel.conmgr.virtualhosts` property in the `siebel.properties` file to list the Siebel servers in the cluster. The `siebel.conmgr.virtualhosts` property is used to list groups of servers with the same function. Servers are listed by using a comma-separated list in the format *hostname:port*. The only property in the `siebel.properties` file that WebSphere Message Broker supports is `siebel.conmgr.virtualhosts`. Information message BIP3427 indicates if a `siebel.properties` file is being used, and error message BIP3428 indicates if a problem occurs when attempting to access the `siebel.properties` file.

FtpServer configurable service

For more information about FtpServer configurable service properties, see "FtpServer configurable service properties" on page 3794.

Supplied configurable services that are created for each broker	Properties for each configurable service that is defined	Description of properties
None	accountInfo	Some FTP servers require an account name during the FTP logon procedure. If this property is specified, its value is the account name that is supplied when it is requested during FTP logon. If this property is not specified, and the server requests an account name, the FTP transfer fails.
	cipher	This property identifies the cipher that is used for encryption. This property takes the form of one or more of the following values, separated by plus signs (+): • `blowfish-cbc` • `3des-cbc` • `aes128-cbc` The cipher that you use for encryption depends on your SSH implementation. List the values in order of preference. This property is valid only when SFTP is specified as the protocol. If FTP is used, this property is ignored. If no value is specified, the following default is used: `blowfish-cbc+3des-cbc+aes128-cbc`
	compression	This property specifies the level of compression to be used. Valid values are integers in the range 0 - 9, where 0 specifies no compression and 9 specifies maximum compression. This property is valid only when SFTP is specified as the protocol. If FTP is used, this property is ignored. If no value is specified, the default value of 0 is used.
	connectionType	This property identifies the FTP data socket connection. Valid values are ACTIVE or PASSIVE. This property is valid only when FTP is specified as the protocol. If SFTP is specified, this property is ignored.
	knownHostsFile	This property specifies the location of the known hosts file. The value must be the fully qualified path to a valid known hosts file. The host information is stored in a `known_hosts` file in the standard OpenSSH format. This property is mandatory if the `strictHostKeyChecking` property is set to Yes. If the `strictHostKeyChecking` property is set to No, this property is ignored. This property is valid only when SFTP is specified as the protocol. If FTP is used, this property is ignored.
	mac	This property specifies the Message Authentication Code. This property takes the form of one or more of the following values, separated by plus signs (+): • `hmac-md5` • `hmac-sha1` The MAC that you use depends on your SSH implementation. List the values in order of preference. This property is valid only when SFTP is specified as the protocol. If FTP is used, this property is ignored. If no value is specified, the following default is used: `hmac-md5+hmac-sha1`
	preserveRemoteFileDate	This property specifies whether files that are retrieved from a remote server by the FileInput node retain the last modified date on the server.

IMSConnect configurable service

Supplied configurable services that are created for each broker	Properties for each configurable service that is defined	Description of properties
None	Hostname	The host name of the computer on which the IMS Connect instance is running. This property is mandatory; if you leave it blank, the node cannot connect to IMS Connect.
	PortNumber	The port number on which the IMS Connect instance is listening. This value must be a positive integer. This property is mandatory; if you leave it blank, the node cannot connect to IMS Connect.
	DataStoreName	The data store name against which the IMS Connect instance is running. This property is not mandatory, but if you do not set it, an exception is issued by any node that attempts to use this configurable service.
	SocketTimeoutSec	The socket timeout is the maximum amount of time that the IMS Connector for Java waits for a response from IMS Connect before it disconnects the socket and returns an exception to the broker. If network problems or routing failures occur, this property prevents the client that is using the IMS Resource Adapter from waiting indefinitely for a response from IMS Connect. This property is based on the TCP/IP sockets that IMS Connect and the IMS Resource Adapter use to communicate; therefore, it is not applicable with the Local option. This property is optional, and the default value is 0 (zero), indicating that the socket never times out. You can enter an integer in the range 0 - 2147483. You set the socket timeout independently of the execution timeout on the configurable service. The socket timeout is used to respond to network problems (such as a loss of connection), while the execution timeout is used to recover from a non-responsive IMS program. The socket timeout is typically longer than the execution timeout.
	ExecutionTimeoutSec	The execution timeout is the maximum amount of time that the IMS Connector for Java waits for a response from a transaction. This property is not mandatory, and the default value is 60, indicating that the IMS Connector for Java waits for 60 seconds. You can enter an integer in the range 1 - 2147483.
	ConnectionIdleTimeoutSec	The connection idle timeout is the maximum amount of time, in seconds, that a connection can be idle before it expires. This property is not mandatory, and the default value is 0 (zero), indicating that the connection never times out. You can enter an integer in the range 0 - 2147483. Any IMS connection that is idle for more than the value specified by this property is not reused, and it is removed from the connection pool.
	UseSSL	The value that defines whether the TCP/IP connection to IMS Connect uses Secure Sockets Layer (SSL) encryption. Valid values are `True` or `False`. Both values are not case-sensitive. If no value is specified, the default value of `False` is used.
	SSLEncryptionType	The value of the cipher that is used for encryption. This value takes effect only when property `UseSSL` is set to `True`. Valid encryption values are `Strong`, `Weak`, or `ENULL`. These values are case-sensitive. Value definition, where: • *ENULL* allows for authentication during the SSL handshaking process. When the handshaking process for a socket has completed including authentication as required, all messages then flow in plain text over that authenticated socket. This

JavaClassLoader configurable service

Supplied configurable services that are created for each broker	Properties for each configurable service that is defined	Description of properties
None	includedDeployedJars	The JAR files that are loaded by the classloader. The JAR files must have been deployed to the execution group. Specify one of the following values: • `none` if no deployed JAR files are to be loaded. • `automatic` if only the JAR file that contains the class that implements a JavaCompute node is to be loaded. • a semicolon separated list of JAR file names that are to be loaded. The JAR file names must end with `.jar`. No wildcard characters are supported. You can include `automatic` in the list to also load the JAR file that contains the class required by a JavaCompute node. The default value is `automatic`. Java classes in the JAR files in `includedDeployedJars` are searched and loaded before classes in the JAR files in the `sharedJarPath`.
	sharedJarPath	The fully qualified file path of a single additional directory to be searched by the classloader. The directory must not be in the broker file path or work path, as defined by the `MQSI_FILEPATH` and `MQSI_WORKPATH` environment variables. If the directory is specified, it is used to resolve additional classes. If the directory is not specified, the broker `shared-classes` directories are used to resolve additional classes. Java classes in the JAR files in `includedDeployedJars` are searched and loaded before classes in the JAR files in the `sharedJarPath`.

JDBCProviders configurable service

Supplied configurable services that are created for each broker	Properties for each configurable service that is defined	Description of properties
DB2 Informix Informix_With_Date_Format Microsoft_SQL_Server Oracle Sybase_JConnect6_05	connectionUrlFormat	A pattern that represents the connection URL definition, which is specific to a particular database type. For example, the pattern for DB2 is defined with the following fixed content: `jdbc:db2://[serverName]:[portNumber]/[databaseName] :user=[user];password=[password];` Do not use the **mqsichangeproperties** command to change the pattern itself; changes made to the pattern might cause unpredictable results.
	connectionUrlFormatAttr1 connectionUrlFormatAttr2 connectionUrlFormatAttr3 connectionUrlFormatAttr4 connectionUrlFormatAttr5	If the specified URL format contains non-standard JDBC data source properties, such as a server identifier, specify one of five general-purpose connection attributes to define these additional properties. For example, if `connectionURLFormat = jdbc:oracle:thin:[user]/[password]@[serverName]:[portNumber]:[connectionUrlFormatAttr1]`, connectionUrlFormatAttr1 must contain an Oracle server identifier, which you must supply by defining the value for the property connectionUrlFormatAttr1 on the **mqsicreateconfigurableservice** or **mqsichangeproperties** command. The broker can then substitute all the required values into the required pattern.
	databaseName	The name of the database to which the data source entry enables connections; for example, employees.
	databaseType	The database type, for example, DB2.
	databaseVersion	The database version; for example, 9.1.
	description	An optional property to describe the data source definition.
	environmentParms	DB2 only. An optional property specifying a list of data source properties of the form `name=value` each separated by a semicolon.
	jarsURL	The local directory path on the system on which the broker is running, where the JAR file that contains the type 4 driver class is located.
	portNumber	The port number on which the database server is listening; for example, 50000.
	securityIdentity	A unique security key to perform a second broker registry lookup to find an entry under the broker DSN entries, which store the encrypted password for the user on their associated host system; for example, `jdbc::mysecurityIdentity`. Use the **mqsisetdbparms** command to create a DSN entry, as described in "Securing a JDBC type 4 connection" on page 689.
	serverName	The name of the server; for example, `host1`.
	type4DatasourceClassName	The name of the JDBC type 4 data source class name that is used to establish a type 4 connection to a remote database and for coordinated transaction support; for example, `com.ibm.db2.jcc.DB2XADataSource`.
	type4DriverClassName	The name of the JDBC driver class name that is used to establish a connection; for example, `com.ibm.db2.jcc.DB2Driver`.
	jdbcProviderXASupport	This property is optional. Setting this property to true, indicates that the selected JDBC provider supports XA coordinated transactions, and the database server is enabled to use the XA transaction protocol. Setting this property to false, indicates that the selected JDBC provider either does not support XA coordinated transactions, or the database server is not enabled to use the XA transaction protocol. The default value for this property is true.

JDEdwardsConnection configurable service

Supplied configurable services that are created for each broker	Properties for each configurable service that is defined	Description of properties
None	assuredOnceDelivery	Specifies whether to provide assured once-only delivery for inbound events. This property applies only to `.inadapter` files. The default value is an empty string, which indicates that behavior is controlled by the `.inadapter` file.
	Environment	The JD Edwards EnterpriseOne environment to which to connect. A JD Edwards EnterpriseOne environment is a user-defined pointer that indicates the location of data and objects on a JD Edwards EnterpriseOne server. Users can be authorized to use multiple JD Edwards EnterpriseOne environments on a single JD Edwards EnterpriseOne server.
	Role	The role that is applicable to the JD Edwards connection. Roles define what authority users have. Users can have multiple roles. A user's access to applications, forms, table columns, data sources, and so on, is based on one or more roles to which the user is assigned. Roles are created and named by the JD Edwards EnterpriseOne administrator.

JMSProviders configurable service

Supplied configurable services that are created for each broker	Properties for each configurable service that is defined	Description of properties
ActiveMQ BEA_Weblogic FioranoMQ Generic_File Generic_LDAP JBoss JOnAS Joram OpenJMS Oracle_OEMS SeeBeyond SonicMQ SwiftMQ Tibco_EMS WebSphere_MQ WebSphere_WAS_Client	clientAckBatchSize	This property is optional. Set this property to configure JMS message flows to send a batch acknowledgment of non-transactional messages that have been received. The value of this property is an integer that represents the threshold number of messages received before the batch acknowledgment is sent. This property can be used in combination with the **clientAckBatchTime** property. If both properties are set, and the time interval set on **clientAckBatchTime** has expired, the batch acknowledgment is sent even if the **clientAckBatchSize** threshold for the number of received messages has not been reached. A batch acknowledgment is also sent when: • there are no more input messages on the JMS server • an error occurs during message processing • the message flow stops. The minimum non-zero value is 2. To disable batch acknowledgment, set **clientAckBatchSize** to 0 and ensure that **clientAckBatchTime** is set to 0.
	clientAckBatchTime	This property is optional. Set this property to configure JMS message flows to send a batch acknowledgment of non-transactional messages that have been received. The value of this property is an integer that represents the length, in milliseconds, of a repeating interval. At the end of each interval a batch acknowledgment is sent for all unacknowledged non-transactional JMS messages that were received during the preceding interval. This property can be used in combination with the **clientAckBatchSize** property. If both properties are set, and the threshold set on **clientAckBatchSize** is reached, the batch acknowledgment is sent even if the time interval specified by **clientAckBatchTime** has not yet expired. A batch acknowledgment is also sent when: • there are no more input messages on the JMS server • an error occurs during message processing • the message flow stops. To disable batch acknowledgment, set **clientAckBatchTime** to 0 and ensure that **clientAckBatchSize** is set to 0.
	initialContextFactory	This property is optional. The fully qualified class name of the class used to perform JNDI lookups. If set it overrides the property set on a node that uses this JMS Provider.
	jarsURL	A URL that specifies the file location of the JMS provider JAR files. Omit `file://` from the URL. If you do not set the **-n** parameter on the **mqsicreateconfigurableservice** command, the default location for the JMS provider JAR files is the broker shared-classes directories. To connect to different versions of the same JMS provider (for example, JBoss), create a JMSProviders configurable service for each version of the JMS provider, then set the **jarsURL** property to a unique path.
	jmsAsyncExceptionHandling	This property is optional. If you set this property to true, the broker registers an exception listener on the JMS connection when the connection is created, and handles connection exceptions

MonitoringProfiles configurable service

Supplied configurable services that are created for each broker	Properties for each configurable service that is defined	Description of properties
DefaultMonitoringProfile	profileProperties	The name of the monitoring profile. The monitoring options defined by the monitoring profile can be configured with an XML file that conforms to the monitoring profile schema.
	useParserNameInMonitoringPayload	When a payload is included in a monitoring message, setting this property to TRUE forces the **wmb:applicationData/wmb:complexContent/wmb:elementName** attribute to contain the name of the input node parser if present. This property is optional. Valid values are TRUE and FALSE. The default value is FALSE, which means that the **wmb:applicationData/wmb:complexContent/wmb:elementName** attribute contains the MessageType property value. Setting this property on the default monitoring profile affects the behavior for all nodes that are not explicitly using configurable services.

PeopleSoftConnection configurable service

Supplied configurable services that are created for each broker	Properties for each configurable service that is defined	Description of properties
None	hostName	Identifies, either by name or IP address, the server that hosts PeopleSoft Enterprise. This property is mandatory. By default, the value of this property is an empty string, therefore you must set a valid host name or IP address.
	port	The port number that the adapter uses to access the PeopleSoft Enterprise server. This property is mandatory. By default, the value of this property is an empty string, therefore you must set a valid port number.
	connectionIdleTimeout	The number of seconds for which a connection can be idle before it is closed by WebSphere Message Broker to effectively maintain the connection pool. The default value for this property is 0 (zero) seconds, indicating that no timeout occurs.

PolicySets configurable service

Supplied configurable services that are created for each broker	Properties for each configurable service that is defined	Description of properties
WSS10Default	config	Reserved for future use.
	ws-security	The content of the policy set. The content is edited with the Policy Sets and Policy Set Bindings editor and can be backed up and restored by using the **-p** parameter on the **mqsireportproperties** and **mqsichangeproperties** commands.

PolicySet Bindings configurable service

Supplied configurable services that are created for each broker	Properties for each configurable service that is defined	Description of properties
WSS10Default	associatedPolicySet	The name of the policy set configurable service with which this policy set binding is associated. This value is set by the Policy Set editor when an association is defined. If you are restoring a policy set and binding by using the **mqsichangeproperties** command, ensure that this command refers to the correct associated policy set.
	config	Reserved for future use.
	ws-security	The content of the policy set binding. The content is edited with the Policy Sets and Policy Set Bindings editor and can be backed up and restored by using the **-p** parameter on the **mqsireportproperties** and **mqsichangeproperties** commands.

Resequence configurable service

Supplied configurable services that are created for each broker	Properties for each configurable service that is defined	Description of properties
None	queuePrefix	The queue prefix used to specify the storage queues that are generated for use by the Resequence node. This property is optional. Two queues are required: • SYSTEM.BROKER.EDA.*QueuePrefix*.EVENTS • SYSTEM.BROKER.EDA.*QueuePrefix*.COLLECTIONS The prefix can contain any characters that are valid in a WebSphere MQ queue name, but must be no longer than eight characters and must not begin or end with a period (.). For example, `SET.1` is valid, but `.SET1` and `SET1.` are invalid. Multiple configurable services can use the same queue prefix. If no queue prefix is specified, the Resequence node uses the default queues that are generated when the broker is created: • SYSTEM.BROKER.EDA.EVENTS • SYSTEM.BROKER.EDA.COLLECTIONS These default queues are also used by the Collector node.
	missingMessageTimeoutSeconds	The value used by the Resequence node to specify how long the node waits for the next message in a sequence before it is propagated. The value can be any integer that is greater than 1. The value of this property (if it is set) overrides the value set on the node. This property is optional; if it is not set, the value set on the node is used.
	startSequenceSeconds	The value used by the Resequence node to specify how long the node waits before calculating the start number in a sequence of messages. The value can be any positive integer. This value is used only if the `Start of sequence` property of the Resequence node is set to `Automatic`. The value of this property (if it is set) overrides the value set on the node. This property is optional; if it is not set, the value set on the node is used.
	endSequenceSeconds	The value used by the Resequence node to specify how long the node waits for the next message in the sequence to arrive before it closes the sequence. The value can be any positive integer. The value of this property (if it is set) overrides the value set on the node. This property is optional; if it is not set, the value set on the node is used.

SAPConnection configurable service

Supplied configurable services that are created for each broker	Properties for each configurable service that is defined	Description of properties
None	applicationServerHost	Specifies the IP address or the name of the application server host to which the adapter logs on. This property applies to `.inadapter` and `.outadapter` files. This property is mandatory; you must set the value to a valid SAP server host name or IP address. The default value is an empty string, which indicates that behavior is controlled by the `.inadapter` or `.outadapter` file.
	assuredOnceDelivery	Specifies whether to provide assured once-only delivery for inbound events. This property applies only to `.inadapter` files. The default value is an empty string, which indicates that behavior is controlled by the `.inadapter` file.
	client	The client number of the SAP system to which the adapter connects. This property applies to `.inadapter` and `.outadapter` files. Set this property to the required SAP client number, which is a three-digit integer in the range 000 - 999. The default value is an empty string, which indicates that behavior is controlled by the `.inadapter` or `.outadapter` file.
	connectionIdleTimeout	The number of seconds for which a connection can be idle before it is closed by WebSphere Message Broker to effectively maintain the connection pool. The default value for this property is 0 (zero) seconds, indicating that no timeout occurs. New connections to SAP are opened with different user IDs, therefore do not set this property to zero if you are using identity propagation.
	gatewayHost	The host name of the SAP gateway. This property applies to `.inadapter` and `.outadapter` files. This property is not mandatory. The default value is an empty string, which indicates that behavior is controlled by the `.inadapter` or `.outadapter` file.
	gatewayService	The identifier of the gateway on the gateway host that carries out the RFC services. This property applies to `.inadapter` and `.outadapter` files. This property is not mandatory. The default value is an empty string, which indicates that behavior is controlled by the `.inadapter` or `.outadapter` file. The property controls the port number that the broker uses when establishing a TCP/IP connection to the SAP server. Set the property to either of the following values: • The name of the service, which must be defined in the services (`/etc/services` or `C:\Windows\System32\drivers\etc\services`) with the port number that corresponds to the port on which the SAP gateway is listening • The port number The name of the gatewayService property and its corresponding port number typically depend on the system number. The gateway

SecurityProfiles configurable service

For more information about SecurityProfiles configurable services, see "SecurityProfiles configurable service properties" on page 3801.

Supplied configurable services that are created for each broker	Properties for each configurable service that is defined	Description of properties
Default_Propagation	authentication	The type of authentication that is performed on the source identity. Valid values are: • NONE • LDAP • TFIM • WS-Trust V1.3 STS • A user-defined value If you are using TFIM V6.1, specify `TFIM`. If you are using TFIM V6.2, specify `WS-Trust V1.3 STS`.
	authenticationConfig	The information that the broker needs to connect to the provider, specific to the provider. It is a provider-specific configuration string.
	authorization	The types of authorization checks that are performed on the mapped or source identity. Valid values are: • NONE • LDAP • TFIM • WS-Trust V1.3 STS • A user-defined value If you are using TFIM V6.1, specify `TFIM`. If you are using TFIM V6.2, specify `WS-Trust V1.3 STS`.
	authorizationConfig	How the broker connects to the provider, specific to the provider. It is a provider-specific configuration string.
	mapping	The type of mapping that is performed. Valid values are: • NONE • TFIM • WS-Trust V1.3 STS • A user-defined value If you are using TFIM V6.1, specify `TFIM`. If you are using TFIM V6.2, specify `WS-Trust V1.3 STS`.
	mappingConfig	How the broker connects to the provider, specific to the provider. It is a provider-specific configuration string.
	passwordValue	How passwords are treated when they enter a message flow. Valid values are: • PLAIN • MASK • OBFUSCATE
	propagation	Indicates whether identity propagation is performed on output and request nodes. Valid values are: • TRUE • FALSE

Supplied configurable services that are created for each broker	Properties for each configurable service that is defined	Description of properties
	rejectBlankpassword	Indicates whether the security manager rejects authentication of a username that has an empty password token, without passing it to LDAP. Valid values are: • TRUE • FALSE Default is FALSE.

Service Registries configurable service

Supplied configurable services that are created for each broker	Properties for each configurable service that is defined	Description of properties
DefaultWSRR	connectionFactoryName	The name of the WSRR WebSphere Application Server JMS provider JMS connection factory for Cache Notifications. The default value is jms/SRConnectionFactory.
	connectionTimeout	The WSRR connection timeout period in seconds. The default value is 180 seconds (3 minutes).
	enableCacheNotification	The default value is False. Select True to enable WebSphere Message Broker WSRR Cache Notification.
	endpointAddress	The location or endpoint of the WSRR server. The default value for all versions of WSRR is `http://fill.in.your.host.here:9080/WSRRCoreSDO/services/WSRRCoreSDOPort` For information about the specific levels of WSRR that are supported with WebSphere Message Broker, see WebSphere Message Broker Requirements.
	initialContextFactory	The name of the WSRR WebSphere Application Server JMS provider JMS context factory for Cache Notifications. The default value is `com.ibm.websphere.naming.WsnIitialContextFactory`
	locationJNDIBinding	The URL to the WebSphere Application Server JMS provider JNDI bindings. The default value is `iiop://`*`host_name`*`:2809/`
	needCache	The default value is True, indicating that the WebSphere Message Broker WSRR cache is enabled.
	predefinedCacheQueries	A list of semicolon-separated WSRR XPath query expressions with which to initialize the WebSphere Message Broker WSRR cache at start-up. These WSRR XPath query expressions are defined by the WSRR SOAP interface query expression language, and might include an optional depth specifier extension.
	refreshQueriesAfter Notification	When a notification is received from WSRR, if refreshQueriesAfterNotification is set to True, the Cache is updated with the new version of the object immediately; if it is set to False, the Cache is updated on the next request. The default value is True.
	subscriptionTopic	The topic name that is used to receive WebSphere Application Server JMS provider Cache Notifications. The default value is jms/SuccessTopic.
	timeout	The cache expiry time in milliseconds. The default value is 10000000, which is approximately 166 minutes.

SiebelConnection configurable service

Supplied configurable services that are created for each broker	Properties for each configurable service that is defined	Description of properties

None	connectString	The connection URL that is needed to connect to the Siebel server.
		
		This property is mandatory. By default, the value of this property is an empty string, therefore you must set a valid URL. The URL has the following format:
		`Protocol://machinename:port/enterprisename/object manager/server name`
		
		For example:
		• For Siebel 7.0.5 to 7.5x:
		`siebel://IP_ADDRESS/siebel/SSEObjMgr_ENU/sebldev1`
		• For Siebel 7.8:
		`siebel://IP_ADDRESS:2321/Sieb78/SSEObjMgr_enu`
		• For Siebel 8:
		`siebel://IP_ADDRESS:2321/SBA_80/SSEObjMgr_enu`
		
		The default port number is 2320, but in these examples, for Siebel version 7.8 and 8, the port has been set to 2321.

	connectionIdleTimeout	The number of seconds for which a connection can be idle before it is closed by WebSphere Message Broker to effectively maintain the connection pool. The default value for this property is 0 (zero) seconds, indicating that no timeout occurs.

SMTP configurable service

Supplied configurable services that are created for each broker	Properties for each configurable service that is defined	Description of properties
None	serverName	The name of the server; for example, `host1`.
	securityIdentity	The name of a security identity that is defined using the **mqsisetdbparms** command.

TCPIPClient configurable service

Applying TCPIP configurable service changes implies restarting the TCPIP connection managers, so all flows which use configurable services to specify TCPIP parameters can be expected to pick up new TCPIP connections.

Supplied configurable services that are created for each broker	Properties for each configurable service that is defined	Description of properties
Default	Hostname	The host name of the remote system to which to connect with a client connection. A valid value is any IP address or computer name. You cannot change this value if there is already a configurable service with this name using the same port (unless the port is set to 0 (zero)).
	Port	The port number to be used for this configurable service. The default is 0 (zero), which means no port number. By default, the configurable service is disabled and the value of the port provided on the node is used instead. A port number can be assigned to only one configurable service at a time; if you try to assign a port number to more than one configurable service, an error occurs.
	AlternativeAddresses	The list of secondary host names and ports to be tried if the main remote address fails. If no addresses are specified, only the main remote address is used. Addresses must be in the format *hostname:port*. If no port is specified, the main port number is used. Multiple addresses must be separated with semicolons; for example: `jones6:1111;edwards`
	CloseWithUnprocessedData	The default value is false. When the value is set to true, the Broker reads ahead up to 1 MB on the stream to look for TCP/IP close events from the steam. If a close event is encountered, then the connection is allowed to close if it is not owned by an input node. This process allows flows that contain connections that are owned by receive nodes to respond to TCP/IP close events, even if execution never proceeds to another Receive node (for example if the flow's input node is idle when the close occurs).
	MinimumConnections	The minimum number of client connections made by the broker. The broker attempts to establish this number of connections even if no flows are using the connections. The default value is 0 (zero), which means that the broker does not make any client connections until they are required.
	MaximumConnections	The maximum number of client connections that can be made on this port. The default value is 100, which means that, by default, the broker accepts up to 100 server connections.
	MaxReceiveRecordBytes	The maximum size a record can reach before an exception is thrown. The default value is 104857600, which means that, by default, the broker accepts messages with a maximum size of 100 MB. The record size is taken to be the size of the data including all delimiters.
	ExpireConnectionSec	The length of time (in seconds) that a connection is kept open without being used. The value can be any integer. A value of 0 (zero) causes the connection to be closed immediately, and a value of -1 causes the connection to remain open indefinitely (no expiry).
	UseUniqueConnectionPool	Specifies whether the connection pool corresponding to the configurable service is the main connection pool for the specified host name and port. You can have two connection pools for the same remote address, and you can use the UseUniqueConnectionPool property to specify whether the connection pool corresponding to this configurable service is the main connection pool. The nodes in a message flow specify which pool to use by specifying the appropriate configurable service.

TCPIPServer configurable service

Supplied configurable services that are created for each broker	Properties for each configurable service that is defined	Description of properties
Default	CloseWithUnprocessedData	The default value is false. When the value is set to true, the Broker reads ahead up to 1 MB on the stream to look for TCP/IP close events from the steam. If a close event is encountered, then the connection is allowed to close if it is not owned by an input node. This process allows flows that contain connections that are owned by receive nodes to respond to TCP/IP close events, even if execution never proceeds to another Receive node (for example if the flow's input node is idle when the close occurs).
	Port	The port number to be used for this configurable service. The default is 0 (zero), which means no port number. By default, the configurable service is disabled and the value of the port provided on the node is used instead. A port number can be assigned to only one configurable service at a time; if you try to assign a port number to more than one configurable service, an error occurs.
	MaximumConnections	The maximum number of server connections that can be made on this port. The default value is 100.
	MaxReceiveRecordBytes	The maximum size a record can reach before an exception is thrown. The default value is 104857600, which means that, by default, the broker accepts messages with a maximum size of 100 MB. The record size is taken to be the size of the data including any delimiters.
	ExpireConnectionSec	The length of time (in seconds) that a connection is kept open without being used. The value can be any integer. A value of 0 (zero) causes the connection to be closed immediately, and a value of -1 causes the connection to remain open indefinitely (no expiry).
	SO_RCVBUF	The size (in bytes) of the SO_RCVBUF property on the socket. Valid values vary according to the operating system that you are using. This property is a standard TCP/IP property. The default value is 0 (zero), which sets the size of the SO_RCVBUF property to the operating system default.
	SO_SNDBUF	The size (in bytes) of the SO_SNDBUF property on the socket. Valid values vary according to the operating system that you are using. This property is a standard TCP/IP property. The default value is 0 (zero), which sets the size of the SO_SNDBUF property to the operating system default.
	SO_KEEPALIVE	The value of the KEEPALIVE property on the socket. If the value is set to True, the socket checks that it is still connected after a specified time. The length of time depends on the TCP/IP implementation on the operating system, but is typically two hours. Keep alive processing works only if the underlying operating system supports SO_KEEPALIVE. This property is a standard TCP/IP property. The default value is False, which means that no keep alive processing is performed.
	TCP_NODELAY	The value of the TCP_NODELAY property on the socket. If the value is set to True, the socket sends data as soon as it is sent to its buffer. The default value is False.
	TrafficClass	The traffic class on any connection that is established. Valid values are positive integers. The default value is -1, which leaves the TrafficClass set to the platform default.
	SO_LINGER	The SO_LINGER property on any connection that is established.

Timer configurable service

Supplied configurable services that are created for each broker	Properties for each configurable service that is defined	Description of properties
None	queuePrefix	The queue prefix used to specify the storage queue that is generated for use by the Timer nodes. This property is optional. The following queue is required: • SYSTEM.BROKER.TIMEOUT.*QueuePrefix*.QUEUE The prefix can contain any characters that are valid in a WebSphere MQ queue name, but must be no longer than eight characters and must not begin or end with a period (.). For example, `SET.1` is valid, but `.SET1` and `SET1.` are invalid. Multiple configurable services can use the same queue prefix. If no queue prefix is specified, the Timer nodes use the default queue that is generated when the broker is created: • SYSTEM.BROKER.TIMEOUT.QUEUE
	timeoutIntervalSeconds	The value used by the TimeoutNotification node to set the timeout interval (in seconds) when running in automatic mode. The value can be any integer that is greater than 1. This property is optional; if it is not set, the value set on the node is used. The value of this property overrides the value set on the node, but it is ignored if the node is not running in automatic mode.

UserDefined configurable service

A UserDefined configurable service has no predefined properties. You can create, modify, and delete properties for your own purposes; for more information, see "UserDefined configurable service properties" on page 3804.

Related concepts:

"Caching artifacts from the WebSphere Service Registry and Repository" on page 1888
WebSphere Message Broker saves the data it retrieves from the WebSphere Service Registry and Repository (WSRR) in a local cache.

"Using the WebSphere Message Broker Explorer to work with configurable services" on page 644
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view, add, modify and delete configurable services.

Related tasks:

"Creating a security profile" on page 433
You can create a security profile for use with Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2. Support is also provided for TFIM V6.1, for compatibility with previous versions of WebSphere Message Broker. You can create the security profile by using either the **mqsicreateconfigurableservice**

command or an editor in the WebSphere Message Broker Explorer.

"Setting up a JDBC provider for type 4 connections" on page 684
Use the **mqsicreateconfigurableservice** or the **mqsichangeproperties** command to configure a JDBC provider service.

"Business-level monitoring" on page 3319
You can configure your message flow to emit event messages that can be used to support transaction monitoring and auditing, and business process monitoring.

Related reference:

"FtpServer configurable service properties"
Select the properties and values that you want to change for an existing FtpServer configurable service or to create a new service.

"SecurityProfiles configurable service properties" on page 3801
Select the objects and properties that you want to change for the SecurityProfiles configurable service.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

FtpServer configurable service properties:

Select the properties and values that you want to change for an existing FtpServer configurable service or to create a new service.

To change these properties, you must specify the broker name and **-c FtpServer**.

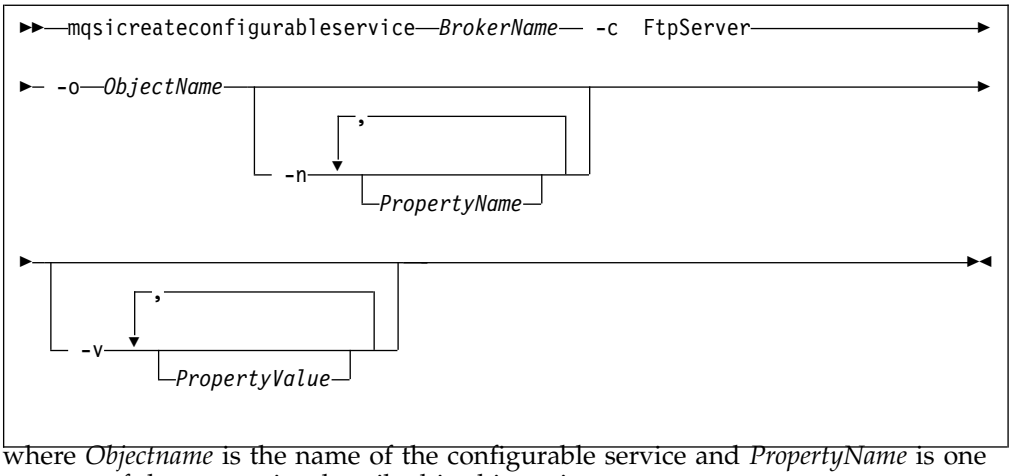
You must also set the **ObjectName** to the name of the configurable service that you have previously created.

See the "**mqsichangeproperties** command" on page 3756 for examples of its use.

49

If you define an FtpServer configurable service by using the **mqsicreateconfigurableservice** command, you can then specify the name of this configurable service in the `Remote server and port` property on the **FTP** tab of the FileInput and FileOutput nodes.

To create an FtpServer configurable service, the command has the following syntax:



where *Objectname* is the name of the configurable service and *PropertyName* is one or more of the properties described in this topic.

If you define an FtpServer configurable service, you **must** specify a value for its serverName property. All the other properties are optional.

serverName
The IP address and, optionally, port number for the remote FTP or SFTP server. The syntax for the property is identical to the syntax that is permitted for the `Remote server and port` property of the FileInput and FileOutput nodes (except that it cannot be the name of an FtpServer configurable service).

accountInfo
Some FTP servers require an account name during the FTP logon procedure. If this property is specified, its value is the account name supplied when requested during FTP logon. If this property is not specified and the server requests an account name, the FTP transfer fails.

cipher
The cipher used for encryption. This property takes the form of a list of one or more of the following values, separated by plus signs (+):
- blowfish-cbc
- 3des-cbc
- aes128-cbc

The cipher that you use for encryption depends on your SSH implementation. List the values in order of preference.

This property is valid only when SFTP is specified as the protocol. If FTP is used, this property is ignored.

If no value is specified, the following default is used: blowfish-cbc+3des-cbc+aes128-cbc

compression
Specifies the level of compression to be used. Valid values are integers in the range 0 - 9, where 0 specifies no compression and 9 specifies maximum compression.

This property is valid only when SFTP is specified as the protocol. If FTP is used, this property is ignored.

If no value is specified, the default value of 0 is used.

connectionType
The FTP data socket connection. It is either `ACTIVE` or `PASSIVE`.

This property is valid only when FTP is specified as the protocol. If SFTP is specified, this property is ignored.

knownHostsFile
The location of the known hosts file. The value must be the fully qualified path to a valid known hosts file.

The host information is stored in a `known_hosts` file in the standard OpenSSH format.

This property is mandatory if the `strictHostKeyChecking` property is set to Yes. If the `strictHostKeyChecking` property is set to No, this property is ignored.

This property is valid only when SFTP is specified as the protocol. If FTP is used, this property is ignored.

mac
Message Authentication Code. This property takes the form of a list of one or more of the following values, separated by plus signs (+):
- hmac-md5
- hmac-sha1

The MAC that you use depends on your SSH implementation. List the values in order of preference.

This property is valid only when SFTP is specified as the protocol. If FTP is used, this property is ignored.

If no value is specified, the following default is used: hmac-md5+hmac-sha1

preserveRemoteFileDate
This property specifies whether files that are retrieved from a remote server by the FileInput node retain the last modified date on the server.

The default value is No. If you select Yes, the FileInput node completes appropriate parsing to determine the remote file date during processing. The MDTM command is used where it is supported by the remote server. Where the MDTM command is not supported, the listing text is used.

When parsing from listing text, the precision is limited to the information that is available in the specific listing format used by the remote server. If the listing format is ambiguous, the FileInput node bases parsing on the locale settings of the broker. If the date cannot be determined reliably from the remote server, an exception is issued and the message is not processed.

protocol
The remote transfer protocol to use. Valid values are FTP or SFTP. If no protocol is specified in the configurable service, the value specified in the node is used.

remoteDirectory
The relative or absolute directory name on the remote FTP server. If set, this property overrides the `Server directory` on the **FTP** tab of the FileInput or FileOutput node that uses this service.

scanDelay
The length of time, in seconds, to wait after a scan of the directory has resulted in no files having been identified for processing. The default is 60 seconds. If set, this property overrides the `Scan delay` on the **FTP** tab of the FileInput node that uses this service.

securityIdentity
The name of a security identity defined using the **mqsisetdbparms** command. If

set, this property overrides `Security identity` on the **FTP** tab of the FileInput or FileOutput node that uses this service. If the value of this property is secId, define the security identity using the following command:

- If you are using FTP:

  ```
  mqsisetdbparms MB7BROKER -n ftp::secId -u userName -p password
  ```

- If you are using SFTP:

  ```
  mqsisetdbparms MB7BROKER -n sftp::secId -u userName -p password
  ```

  or

  ```
  mqsisetdbparms MB7BROKER -n sftp::secId -u userName -i SSHIdentityFile
  -r Passphrase
  ```

strictHostKeyChecking
Specifies how host keys are checked during the connection and authentication phase. Valid values are:

No Specifies that the following action is performed:

- If the connection is to a new host, connect and accept the host's key, and store it
- If the connection is to a host that has been connected to previously, and the host key has changed, issue an exception (FileOutput node).

If you select No, a default known hosts file (managed by the broker) is used.

Yes Connect only to known hosts with valid keys; otherwise issue an exception.

If you select Yes, you must specify your own known hosts file using the `knownHostsFile` property.

The default value is No.

The host information is stored in a `known_hosts` file in the standard OpenSSH format.

This property is valid only when SFTP is specified as the protocol. If FTP is used, this property is ignored.

timeoutSec
The timeout value in seconds to establish a connection to the remote FTP or SFTP server. You can set any valid integer as the `timeoutSec` property value. A timeout of 0 is interpreted as an infinite timeout. If you are using FTP, the default value is 5 seconds. If you are using SFTP, the default value is 20 seconds.

transferMode
The transfer mode of the FTP connection. This value is either `BINARY` or `ASCII`. The default is `BINARY`. If set, this property overrides `Transfer mode` on the **FTP** tab of the FileInput or FileOutput node that uses this service.

This property is valid only when FTP is specified as the protocol. If SFTP is specified, this property is ignored.

By default, none of these properties in the FtpServer configurable service definition is set. The only mandatory property when you define an FtpServer configurable service is serverName.

The following example of a **mqsicreateconfigurableservice** command shows how to create an FtpServer configurable service:

```
mqsicreateconfigurableservice MB7BROKERR -c FtpServer -o Server01
-n serverName,scanDelay,transferMode,connectionType,securityIdentity
-v one.hursley.abc.com:123,20,BINARY,ACTIVE,secId
```

Values set in properties in the FtpServer configurable service definition override the values set in the corresponding properties in the FileInput and FileOutput nodes.

If you set the accountInfo property, it is used during the login protocol when connecting to the FtpServer configurable service after supplying the user identifier and password. This information is sometimes required by FTP servers and requested as part of the login protocol. This setting allows the FileInput and FileOutput nodes to respond appropriately during login.

If you set the connectionType property, it alters the type of data socket that is used to transfer files to or from the FTP server. If you set this property to `ACTIVE`, this refers to a socket that is established by the remote server to the client (the broker message flow). If you set this property to `PASSIVE`, it refers to a socket that is established by the client to the remote server (as is the login or control socket). The default is `PASSIVE`, which is more likely to be tolerated by most types of firewall protection that allow the client to log in. You can set this property to `ACTIVE` if either the FTP server does not support `PASSIVE` connections, or there are special arrangements that your configuration must meet.

Related reference:

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

"FileInput node" on page 4415
Use the FileInput node to process messages that are read from files.

"FileOutput node" on page 4430
Use the FileOutput node to write messages to files.

CDServer configurable service properties:

Select the objects and properties that you want to change for the *CDServer* configurable service.

You can use configurable services if, for example, you have different CDOutput nodes addressing more than one Connect:Direct server.

To change the *CDServer* configurable service properties, you must specify the broker name and **-c CDServer**.

The **-o** parameter is set to the name of the configurable service you are using.

You can edit the predefined configurable service, named `Default`, to alter the default behavior of the CDInput and CDOutput nodes.

Alternatively, you can create your own configurable service, name it (for example, **-o myCDServer**), and reference this configurable service in the `CDServer configurable service` property of the CDInput or CDOutput node; for more information, see "CDInput node" on page 4305 and "CDOutput node" on page 4312.

The *hostname* and *port* properties are mandatory; all the other properties are optional.

-n hostname
The host name of the Connect:Direct server

- Value type - string
- Initial value - localhost
- Other valid value:
  - Host name of the machine running the Connect:Direct server.

-n port
The API port of the Connect:Direct server.

- Value type - string
- Initial value - 1363
- Other valid value:
  - API port on which the Connect:Direct server is listening.

-n brokerPathToStagingDir
The high level path used by the CDOutput node for staging files to transfer. The CDOutput node creates directories and files under this path.

The path given is used by the broker on the machine running the WebSphere Message Broker. The Connect:Direct server must have access to these files and if it is on a different machine, the directory must be mounted on the remote machine.

If the remote path to the mounted directory is different on the IBM Sterling Connect:Direct machine, the *cdPathToStagingDir* must be set to the correct value.

- Value type - string
- Initial value - ""

  If not set, the broker uses the following default location: `$MQSI_WORKPATH/common/CD/Tranfers`
- Other valid value:
  - Any path on the brokers machine to which the broker user has both read and write access.

-n cdPathToStagingDir
The high level path used by the Connect:Direct server node for accessing files staged by the CDOutput node. The CDOutput node creates directories

and files under this path which the Connect:Direct server must have access to. The path given is used by the Connect:Direct server on the machine running the Connect:Direct server.

- Value type - string
- Initial value - ""

  If not set, the value uses the default location set in *brokerPathToStagingDir*
- Other valid value:
  - The path on the Connect:Direct server machine that allows access to the files staged by the CDOutput node.

-n cdPathToInputDir

The directory, including all subdirectories, on the Connect:Direct server machine that contains the files transferred by CD in which this configurable service is interested. Transfers to any other directory are ignored.

For WebSphere Message Broker to be able to process the files transferred to this directory, this directory must be accessible toWebSphere Message Broker.

If the directory is remote, the directory must be mounted to the WebSphere Message Broker machine. If the mounted file system has a different path on the WebSphere Message Broker machine, the *brokerPathToInputDir* must be set to the correct value.

- Value type - string
- Initial value - ""

  The CDInput node uses the file path sent by the Connect:Direct server.
- Other valid value:
  - A path to a directory structure on the Connect:Direct server machine.

-n brokerPathToInputDir

The file path to reach files transferred to *cdPathToInputDir* from the WebSphere Message Broker machine.

- Value type - string
- Initial value - ""

  The CDInput node uses the file path sent by the Connect:Direct server.
- Other valid value:
  - A path to a directory structure on the WebSphere Message Broker machine.

-n queuePrefix

A queue prefix for use when WebSphere Message Broker stores data on WebSphere MQ about this Connect:Direct server. Two queues are used to store state:

- SYSTEM.BROKER.CD.STATS
- SYSTEM.BROKER.CD.TRANSFERS

If you give a prefix, use the following queue name formats:

- SYSTEM.BROKER.CD.<queuePrefix>.STATS
- SYSTEM.BROKER.CD.<queuePrefix>.TRANSFERS
- Value type - string
- Initial value - ""

The SYSTEM.BROKER.CD.STATS queue and SYSTEM.BROKER.CD.TRANSFERS queue are used.

- Other valid value:
  - The queues are created and used with the <queuePrefix> inserted. The <queuePrefix> must:

    Be a maximum length of 16 characters

    Be a valid value for a queue name

    Not begin or end in a period.

-n securityIdentity

A security identity used to connect to the Connect:Direct server.

- Value type - string
- Initial value - `default`

  The default security identity is used.
- Other valid value:
  - Any valid security identity that is defined on the broker. You must define the identity by using the syntax `cd::<name>`.

See the "**mqsichangeproperties** command" on page 3756 for examples of its use.

Related concepts:

"IBM Sterling Connect:Direct overview and concepts" on page 1810
An overview of IBM Sterling Connect:Direct and its terminology when used with WebSphere Message Broker.

Related tasks:

"Initiating a managed file transfer using IBM Sterling Connect:Direct" on page 1873
Use a CDOutput node to send a file from a specified directory on your primary Connect:Direct server (PNODE) to a filename and directory on a secondary Connect:Direct server (SNODE).

Related reference:

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

SecurityProfiles configurable service properties:

Select the objects and properties that you want to change for the SecurityProfiles configurable service.

To change these properties, you must specify the broker name and **-c SecurityProfiles**.

You must also set the **ObjectName** to either `Default_Propagation` or the name of a SecurityProfiles configurable service that you have defined by using the **mqsicreateconfigurableservice**. The properties and values are the same for all services.

This configurable service is independent of the securitycache component.

-n authentication
The type of authentication that is performed on the source identity.

- Value type - enum
- Initial value - NONE
- Other valid values:
  - LDAP
  - TFIM
  - WS-Trust V1.3 STS
  - A user-defined value

  `TFIM` applies to TFIM V6.1 only. If you are using TFIM V6.2, specify `WS-Trust V1.3 STS`.

-n authenticationConfig
The information that the broker needs to connect to the provider, specific to the provider.

- Value type - string
- Initial value - None

-n authorization
The types of authorization checks that are performed on the mapped or source identity.

- Value type - enum
- Initial value - NONE
- Other valid values:
  - LDAP
  - TFIM
  - WS-Trust V1.3 STS
  - A user-defined value

  `TFIM` applies to TFIM V6.1 only. If you are using TFIM V6.2, specify `WS-Trust V1.3 STS`.

-n authorizationConfig
How the broker connects to the provider, specific to the provider.

- Value type - string
- Initial value - None

-n mapping
The type of mapping that is performed.

- Value type - enum
- Initial value - NONE
- Other valid values:
  - TFIM
  - WS-Trust V1.3 STS
  - A user-defined value

`TFIM` applies to TFIM V6.1 only. If you are using TFIM V6.2, specify `WS-Trust V1.3 STS`.

-n mappingConfig
How the broker connects to the provider, specific to the provider.
- Value type - string
- Initial value - None

-n passwordValue
How passwords are treated when they enter a message flow.
- Value type - enum
- Initial value - PLAIN
- Other valid values:
  - MASK
  - OBFUSCATE

-n propagation
Indicates whether identity propagation is performed on output and request nodes.
- Value type - Boolean
- Initial value - TRUE

-n rejectBlankpassword
Indicates whether the security manager rejects a username that has an empty password token, without passing it to LDAP.
- Value type - Boolean
- Initial value - FALSE

See the "**mqsichangeproperties** command" on page 3756 for examples of its use.

Related concepts:

"Security profiles" on page 387
A security profile defines the security operations that are to be performed in a message flow at SecurityPEP nodes and security enabled input and output nodes.

"Identity and security token propagation" on page 426
Identity and security token propagation enables the identity and security tokens (associated with each message) to be propagated throughout a message flow, and on to target applications through output or request nodes.

Related reference:

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"Parameter values for the securitycache component" on page 3815
Select the objects and properties associated with the securitycache component that

you want to change.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

UserDefined configurable service properties:

A UserDefined configurable service has no predefined properties; you can create, change, view, and delete your own values.

To change these properties, you must specify the broker name and **-c UserDefined**.

You must also set the **ObjectName** to the name of a UserDefined configurable service that you have defined by using the **mqsicreateconfigurableservice** command.

Use a UserDefined configurable service to define properties that you can access in your message flows. The properties that you can define are unlimited; you can create properties to serve any purpose that you can think of. Possible uses include:

- Set timeout values for HTTP message processing.
- Set timeouts that are associated with by message category for HTTP messages.

A UserDefined configurable service is accessible by the JavaCompute Node node only, and cannot be accessed by ESQL.

See the "**mqsichangeproperties** command" on page 3756 for examples of its use with a UserDefined configurable service.

Related concepts:

"Security profiles" on page 387
A security profile defines the security operations that are to be performed in a message flow at SecurityPEP nodes and security enabled input and output nodes.

"Identity and security token propagation" on page 426
Identity and security token propagation enables the identity and security tokens (associated with each message) to be propagated throughout a message flow, and on to target applications through output or request nodes.

Related reference:

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"Parameter values for the securitycache component" on page 3815
Select the objects and properties associated with the securitycache component that you want to change.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

Content based filtering component parameter values:

Select the objects and properties associated with the content based filtering component that you want to change.

To change these properties, you must specify the broker name and **-e ExecutionGroupLabel**. You must also set the **ObjectName** to `ContentBasedFiltering`.

-n cbfEnabled
Whether content-based filtering is enabled.
- Value type - Boolean
- Initial value - false

-n evaluationThreads
The number of evaluation threads available.
- Value type - integer
- Initial value - 1
- Other valid values: positive integer in the range 2 - 32.

-n validationThreads
The number of validation threads available.
- Value type - integer
- Initial value - 1
- Other valid values: positive integer in the range 2 - 32.

Note, that if `cbfEnabled` = true and you set either of the thread settings to zero, the value of that thread setting is changed to one automatically.

See the "**mqsichangeproperties** command" on page 3756 for examples of its use.

Related reference:

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

Execution group HTTP listener parameters (SOAP and HTTP nodes):

Select the resources and properties associated with the HTTPInput, HTTPReply, SOAPInput, SOAPReply, and SOAPAsyncResponse nodes that you want to change.

To change these properties, you must specify the broker name and **-e** followed by the name of a single execution group, and **-o** to specify the execution group object. If you specify **-o ExecutionGroup**, you can omit **-e** to change a property that applies to all execution groups on the specified broker.

Because you define all of these properties at the execution group level, they apply to all HTTPInput, HTTPReply, SOAPInput, SOAPReply, and SOAPAsyncResponse nodes that you deploy to the specified execution group.

By default, the HTTP nodes that you deploy to the broker all use the single broker-wide listener. If you prefer, you can change the configuration of the broker so that all execution groups use the embedded execution group. You can also use the broker-wide listener for some execution groups, and the embedded listener in other execution groups. For more information about these options, see "HTTP listeners" on page 1589.

If you want HTTP nodes to use the broker-wide listener, see "Broker-wide HTTP listener parameters" on page 3809 for further information about setting this configuration.

You must restart the execution group for all changes to be implemented.

Choose the **ObjectName** from the following options:
- `ExecutionGroup` for controlling common properties
- `HTTPConnector` for controlling HTTP communication.
- `HTTPSConnector` for controlling HTTPS communication.

The following combinations are valid:

-o ExecutionGroup
The following properties and values are valid:

-n httpNodesUseEmbeddedListener
A value of false, the default value, indicates that all HTTPInput and HTTPReply nodes use the broker-wide HTTP listener. If you set this property to true, all HTTPInput and HTTPReply nodes that are deployed to the specified execution group use the embedded execution group listener.

If you have disabled the broker-wide HTTP listener, all HTTPInput, HTTPReply nodes use the embedded listener regardless of the value of this property.
- Value type - Boolean
- Initial value - False

-o HTTPConnector
The following properties and values are valid for HTTPConnector and HTTPSConnector:

-n address
For servers with more than one IP address, this value specifies which address is used for listening on the specified port. By default, this port is used on all IP addresses associated with the server. If specified, only one address can be used.
- Value type - string
- Initial value - null

-n explicitlySetPortNumber
The TCP/IP port number on which this Connector creates a server socket and awaits incoming connections.

Setting this value disconnects the automatic port-finding capability of the connector; this port is the only one allowed, and the connector fails to start if another program has already used this port.
- Value type - integer
- Initial value - `7800-7842`
- Other valid values - any integer in the range 0-65536.

You must use the **explicitlySetPortNumber** attribute, because the **port** attribute no longer works.

When you set the value to zero, the system enables the automatic port-finding capability again, starting with the last used port, which was saved by the execution group. You can change the default port used when port-scanning starts by explicitly setting a value in the range 7800-7842, then specifying the port number as zero.

The default initial value for HTTPS is 7843 within the range 7843-7884.

-n autoRespondHTTPHEADRequests

Specifies if the connector handles HEAD requests for HTTP traffic in the same way that it does for SOAP traffic. When the value is `True`, if the request is for a URI associated with a flow, the connector returns an HTTP 200 OK response. If not, the connector returns an HTTP 404 Not Found response.

When the value is `False` or left unchanged from the initial value, if the request is for a URI associated with a message flow, the request causes the message flow to be invoked. Otherwise, the connector returns an HTTP 404 Not Found response.

- Value type - boolean
- Initial value - `False`

serverName

Set the value that is set in the "Server" header for all HTTP responses sent by this server.

- Value type - string
- Initial value - Apache-Coyote/1.1

-o HTTPSConnector

The properties listed for object name **HTTPConnector** (`address`, `explicitlySetPortNumber`, `autoRespondHTTPHEADRequests`, and `autoRespondHTTPHEADRequests`) are also valid for this object name. See HTTPConnector for more information.

The following additional properties and values are valid:

Tip: The valid values for `keystoreType`, `sslProtocol`, `sslSessionTimeout`, and `ciphers` are JSSE-implementation specific. These values are in the JSSE provider documentation. For a list of the platforms that use IBM Java, see Appendix A of the IBM JSSE2 Guide, at: http://www.ibm.com/developerworks/java/jdk/security/60/secguides/jsse2Docs/JSSE2RefGuide.html.

-n algorithm

The certificate encoding algorithm to be used.

- Value type - string
- Initial value -
  - Solaris HP-UX SunX509 on Solaris and HP-UX
  - AIX z/OS Linux Windows IbmX509 on other systems (AIX, Linux, Windows, z/OS)

-n ciphers

A comma-separated list of the encryption ciphers that can be used. If not specified (the default), any available cipher can be used. The

client sends a list of ciphers in priority order. The server selects the first acceptable cipher in the list. If none of the ciphers in the list are suitable, the server returns a handshake failure alert and closes the connection.
- Value type - string
- Initial value - null

-n clientAuth
Set to true if the SSL stack requires a valid certificate chain from the client before accepting a connection. A false value (which is the default) does not require a certificate chain.
- Value type - string
- Initial value - false

-n keyAlias
The alias that is used for the server certificate in the keystore. If not specified, the first key that is read in the keystore is used.
- Value type - string
- Initial value - null

-n keystoreFile
The path to the keystore file where the server certificate, which is to be loaded, is stored. By default the HTTP listener expects a file called `.keystore` in the home directory of the user who started the broker.
- Value type - string
- Initial value - *default value* (described previously)

-n keystorePass
The password used to access the server certificate from the specified keystore file.
- Value type - string
- Initial value - changeit

-n keystoreType
The type of keystore file to be used for the server certificate.
- Value type - string
- Initial value - `JKS`

-n sslProtocol

Specifies the version of the SSL protocol to use.
- Value type - string
- Initial value - `TLS`
- Other valid values - `SSL`, `SSLv3`

-n sslSessionTimeout
Specifies the timeout value in seconds, and is set on the SSLSessionContext for SSLSessions created by the **`HTTPConnector`**.
- Value type - integer
- Initial value - 86400 (24 hours)
- Other valid values - any positive integer, 0 means no timeout

See the "**`mqsichangeproperties`** command" on page 3756 for examples of its use.

Related concepts:

"HTTP listeners" on page 1589
You can choose between broker-wide listeners and execution group (embedded) listeners to manage HTTP messages in your HTTP flows. Learn about the two types of listener, how ports are assigned to them, and how you can switch from

one to the other for individual execution groups.

Related reference:

"Broker registry object parameter values" on page 3765
Select the names of the properties and values that you want to change for the broker registry object.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"Broker-wide HTTP listener parameters"
Select the resources and properties associated with the broker-wide HTTP listener that you want to change.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"HTTPInput node" on page 4474
Use the HTTPInput node to receive an HTTP message from an HTTP client for processing by a message flow.

"HTTPReply node" on page 4484
Use the HTTPReply node to return a response from the message flow to an HTTP client. This node generates the response to an HTTP client from which the input message was received by the HTTPInput node, and waits for confirmation that it has been sent.

"SOAPInput node" on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

"SOAPReply node" on page 4819
Use the SOAPReply node to send SOAP messages from the broker to the originating client in response to a message received by a SOAPInput node.

Broker-wide HTTP listener parameters:

Select the resources and properties associated with the broker-wide HTTP listener that you want to change.

To change these properties, you must specify the broker name and **-b httplistener**.

The httplistener component defines properties for the broker that are used for all the HTTPInput and HTTPReply nodes that you deploy to that broker. These properties include the broker-wide listener for all HTTP nodes; if you prefer, you can disable this option so that the HTTP nodes use the listener that is embedded within each execution group.

You must restart the broker for all changes to be implemented.

Choose the **ObjectName** from the following options:

- `HTTPListener` for controlling the HTTPListener process

  You can use the following values with `HTTPListener`:

```
uuid='HTTPListener'
  enableSSLConnector='false'
  threadPoolSize=''
  traceOverrideLevel=''
  traceOverrideSize=''
```

```
  traceLevel='none'
  traceSize='102400 KB'
  javaDebugPort=''
  startListener='true'
```

- `HTTPConnector` for controlling HTTP communication.

  You can use the following values with `HTTPConnector`:

```
 uuid='HTTPConnector'
  address=''
  port='7080'
  allowTrace=''
  maxPostSize=''
  acceptCount=''
  bufferSize=''
  compressableMimeTypes=''
  compression=''
  connectionLinger=''
  connectionTimeout=''
  maxHttpHeaderSize=''
  maxKeepAliveRequests=''
  maxSpareThreads=''
  maxThreads='20'
  minSpareThreads=''
  noCompressionUserAgents=''
  restrictedUserAgents=''
  socketBuffer=''
  tcpNoDelay=''
  enableLookups='false'
```

- `HTTPSConnector` for controlling HTTPS communication.

  You can use the following values with `HTTPSConnector`:

```
uuid='HTTPSConnector'
  algorithm='Platform Default'
  clientAuth='Platform Default'
  keystoreFile='Platform Default'
  keystorePass='********'
  keystoreType='Platform Default'
  sslProtocol='Platform Default'
  ciphers='Platform Default'
  keyAlias=''
  keypass='********'
  address=''
  port=''
  allowTrace=''
  maxPostSize=''
  acceptCount=''
  bufferSize=''
  compressableMimeTypes=''
  compression=''
  connectionLinger=''
  connectionTimeout=''
  maxHttpHeaderSize=''
  maxKeepAliveRequests=''
  maxSpareThreads=''
  maxThreads=''
  minSpareThreads=''
  noCompressionUserAgents=''
  restrictedUserAgents=''
  socketBuffer=''
  tcpNoDelay=''
  enableLookups='false'
```

The following combinations are valid for the httplistener component:

-o HTTPListener
: The following properties and values are valid:

enableSSLConnector

A Boolean value which can be used to enable or disable the HTTPS (SSL) connector. You must set this value to TRUE to start the HTTP listener listening for inbound SSL connections.

- Value type - Boolean
- Initial value - FALSE

-n maxKeepAliveRequests

The value is the maximum number of HTTP requests that can be pipelined until the connection is closed by the server. Setting the attribute to 1 disables HTTP/1.0 keep-alive, as well as HTTP/1.1 keep-alive and pipelining. Setting the value to `-1` allows an unlimited amount of pipelined or keep-alive HTTP requests.

- Value type - integer
- Initial value `-100`

-n maxThreads

The value is the maximum number of request processing threads to be created by this Connector. This value, therefore, determines the maximum number of simultaneous requests that can be handled.

- Value type - integer
- Initial value - `200`

-n startListener

The broker-wide listener that is used by deployed HTTP nodes is started by default. Use this parameter to change its status between active and inactive.

If this listener is active, all HTTP nodes in all execution groups use this listener, unless you have activated the embedded listener in the execution group by using the **mqsichangeproperties** command for that execution group. For information about configuring an execution group so that HTTP nodes can use the embedded listener, see "Execution group HTTP listener parameters (SOAP and HTTP nodes)" on page 3805.

If this listener is disabled, all execution groups use the embedded listener for all nodes.

- Value type - Boolean
- Initial value `-TRUE`

-o HTTPConnector

The following properties and values are valid:

address

For servers with more than one IP address, this value specifies which address is used for listening on the specified port. By default, this port is used on all IP addresses associated with the server. If specified, only one address can be used.

- Value type - string
- Initial value - null

port The TCP port number on which this HTTPConnector creates a server socket and awaits incoming connections.

- Value type - integer
- Initial value - `7080`

serverName

Set the value that is set in the "Server" header for all HTTP responses sent by this server.

- Value type - string
- Initial value - Apache-Coyote/1.1

-o HTTPSConnector

The properties listed for object name **HTTPConnector** are also valid for this object name. The following additional properties and values are valid:

algorithm

The certificate encoding algorithm to be used.

- Value type - string
- Initial value -
  - Solaris HP-UX SunX509 on Solaris and HP-UX
  - AIX z/OS Linux Windows IbmX509 on other systems (AIX, Linux, Windows, z/OS)

ciphers

A comma-separated list of the encryption ciphers that can be used. If not specified (the default), any available cipher can be used.

- Value type - string
- Initial value - null

clientAuth

Set to true if the SSL stack requires a valid certificate chain from the client before accepting a connection. A false value (which is the default) does not require a certificate chain unless the client requests a resource protected by a security constraint that uses CLIENT-CERT authentication.

- Value type - string
- Initial value - false

keyAlias

The alias that is used for the server certificate in the keystore. If not specified, the first key that is read in the keystore is used.

- Value type - string
- Initial value - null

keystoreFile

The path to the keystore file where the server certificate, which is to be loaded, has been stored. By default, the HTTP listener expects a file called `.keystore` in the home directory of the user who started the broker.

- Value type - string
- Initial value - *default value* (described previously)

keystorePass

The password used to access the server certificate from the specified keystore file.

- Value type - string
- Initial value - changeit

keystoreType

The type of keystore file to be used for the server certificate.

- Value type - string
- Initial value - `JKS`

sslProtocol

The version of the SSL protocol to use.

- Value type - string
- Initial value - SSLv3

-n sessionCacheSize

Set the value to the maximum number of sessions that you allow in the SSL Session Cache. These cached sessions are available for SSL session renegotiation.

- Value type - integer
- Initial value - 100

The properties listed for object name **HTTPConnector** are also valid for this object name. The valid values for keystoreType, sslProtocol, and ciphers are JSSE-implementation-specific, and these values are in the JSSE provider documentation.

See the "**mqsichangeproperties** command" on page 3756 for examples of how to change parameters for the httplistener component.

Related concepts:

"HTTP listeners" on page 1589
You can choose between broker-wide listeners and execution group (embedded) listeners to manage HTTP messages in your HTTP flows. Learn about the two types of listener, how ports are assigned to them, and how you can switch from one to the other for individual execution groups.

Related reference:

"Execution group HTTP listener parameters (SOAP and HTTP nodes)" on page 3805
Select the resources and properties associated with the HTTPInput, HTTPReply, SOAPInput, SOAPReply, and SOAPAsyncResponse nodes that you want to change.

"Parameter values for the servicefederation component" on page 3816
Select the objects and properties associated with the servicefederation component that you want to change.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"HTTPInput node" on page 4474
Use the HTTPInput node to receive an HTTP message from an HTTP client for processing by a message flow.

"HTTPReply node" on page 4484
Use the HTTPReply node to return a response from the message flow to an HTTP client. This node generates the response to an HTTP client from which the input message was received by the HTTPInput node, and waits for confirmation that it has been sent.

JVM parameter values:

Select the objects and properties associated with the Java Virtual Machine (JVM) that you want to change.

To change these properties, you must specify the broker name, and set the **ObjectName** to ComIbmJVMManager. If you are setting a debug port number, you must also include **-e** and specify the name of the execution group which will use the port.

-n jvmMinHeapSize
The minimum size of the storage available to the JVM, specified in bytes.

- Value type: integer
- Initial value: -1, which represents 33554432 bytes (32MB)

-n jvmMaxHeapSize
The maximum size of the storage available to the JVM, specified in bytes. If you have configured a publish/subscribe domain, when execution groups retain publications, you might need to increase this value. If you have included the XSLTransform node in one or more message flows, and the node is processing very large XML messages, you might also have to change this parameter.

- Value type: integer
- Initial value: -1, which represents 268435456 bytes (256 MB)

-n jvmDebugPort
The port on which the execution group is listening. You must set a port number to activate debug in the execution group.

- Value type: integer
- Initial value: 0

-n jvmNativeStackSize
The maximum stack size for Java threads.

- Value type: integer
- Initial value: -1, which represents 1048576 bytes (1 MB) on Solaris and Linux platforms, otherwise the value is JVM dependent

-n jvmJavaOSStackSize
The default stack size for Java Operating System threads. If you have a message flow with large number of nodes between two nodes written in Java (for example, JavaCompute nodes or Java user-defined nodes), you might have to increase the size of this parameter.

- Value type: integer
- Initial value: JVM dependent

-n jvmSystemProperty
The value of this property defines Java system properties for an execution group that can be used by JavaCompute nodes. The format is in the form -Dname1=value1. You can set multiple properties in the following way:

```
-v"-Dname1=value1 -Dname2=value2"
```

- Value type: integer
- Initial value: JVM dependent

-n jvmVerboseOption
The value of this property identifies the type of Java verbose information that is displayed for an execution group.

- Value type: string
- Initial value: none
- Other valid values:
  - class: display information about each class loaded

- `jni`: display information about use of native methods and other Java Native Interface activity
- `gc`: display information about each garbage collection event
- `all`: display information for all of the verbose options

See the "**mqsichangeproperties** command" on page 3756 for examples of how to change parameters for the JVM. Other examples are provided for particular tasks:

"Using the Test Client in trace and debug mode" on page 3155

"Attaching the flow debugger to an execution group for debugging" on page 3160

"Setting the JVM heap size" on page 3254

Related concepts:

"JVM heap sizing" on page 3269
The Java virtual machine (JVM) heap is an independent memory allocation that can reduce the capacity of the main memory heap.

Related tasks:

"Debugging a message flow" on page 3157
Use the tasks described in this section of the documentation to manage and work with the flow debugger.

Related reference:

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"XSLTransform node" on page 4968
Use the XSLTransform node to transform an XML message to another form of message, according to the rules provided by an XSL (Extensible Stylesheet Language) style sheet, and to set the `Message domain`, `Message set`, `Message type`, and `Message format` for the generated message.

Parameter values for the securitycache component:

Select the objects and properties associated with the securitycache component that you want to change.

To change these properties, you must specify the broker name and **-b securitycache**. You must also set the **ObjectName** to `SecurityCache`.

-n cacheTimeout

The timeout value for marking entries in the cache as invalid. The time is specified in seconds.

- Value type - integer
- Initial value - 60
- Other valid values: any positive integer.

The **cacheSweepInterval** property, which is displayed when you report the properties of the *securitycache* component, is ignored and has no effect.

See the "**mqsichangeproperties** command" on page 3756 for examples of its use.

Related reference:

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"**mqsireloadsecurity** command" on page 3911
Use the **mqsireloadsecurity** command to force the immediate expiry of some or all the entries in the security cache.

Parameter values for the servicefederation component:

Select the objects and properties associated with the servicefederation component that you want to change.

To change these properties, you must specify the broker name and **-b servicefederation**. Choose the **ObjectName** from the following options:

- `HTTPConnector` for controlling the properties of the HTTP port that processes Service Control Management Protocol (SCMP) Atom requests. Property values and usage are the same as for the properties available on the `HTTPConnector` option on the httplistener component and are described in "Broker-wide HTTP listener parameters" on page 3809. The default HTTP port used for SCMP Atom requests is 7088.
- `HTTPSConnector` for controlling the properties of the secure HTTPS port that processes SCMP Atom requests. Property values and usage are the same as for the properties available on the `HTTPSConnector` option on the httplistener component and are described in "Broker-wide HTTP listener parameters" on page 3809. The default HTTPS port used for SCMP Atom requests is 7089.
- `scmp` for controlling the properties that relate to the implementation of Service Federation Management (SFM) support in WebSphere Message Broker.

You can select these properties when you specify `scmp` as the **ObjectName**:

enabled
A Boolean value to establish whether the broker is to start Service Federation SCMP Atom support so that an SFM console can connect to the broker. You must set this value to true if you want WebSphere Message Broker to start Service Federation and accept a connection from the SFM console.

- Value type - Boolean
- Initial value - false

enableSSL
A Boolean value to establish whether the SFM console needs to use HTTPS to the SCMP Atom port. You must set this value to true if you need to use HTTPS. You must set this value to false if you need to use HTTP. If you specify a value of true, you must configure the `HTTPSConnector` option, ensuring that you set appropriate values in the keystore-related properties. If you specify a value of false, you must configure the `HTTPConnector` option.

- Value type - Boolean
- Initial value - true

hostname
The host name that the SFM console should use to reach the Broker host.

The SFM console might run in a separate network domain so ensure that you specify a fully qualified domain name, for example `brkhost01.location.ibm.com`. If you do not set a value, the value that is supplied is, on many systems, the dot-separated IP address of the host; this address is subject to change should the network be re-configured.

- Value type - string
- Initial value - null

contextRoot
A valid URI path to define the context root of all URIs for SCMP Atom requests.

- Value type - string
- Initial value - /scmp

contactName
The contact property presented in the connectivity provider Atom entry for all execution groups in the broker. This value of the property might, typically, be the contact details of the broker administrator.

- Value type - string
- Initial value - null

locationName
The location property presented in the connectivity provider Atom entry for all execution groups in the broker. This value of the property might, typically, be the location details of the broker.

- Value type - string
- Initial value - null

-n organizationName
This value specifies the organization property presented in the connectivity provider Atom entry for all execution groups in the broker. This value of the property might, typically, be the organization details of the services hosted by the broker.

- Value type - string
- Initial value - null

Related concepts:

"Working with Service Federation Management (SFM)" on page 911
You can use a broker as a connectivity server that can be administered by a Service Federation Management (SFM) console provided with WebSphere Service Registry and Repository (WSRR) V7.0. The SFM console can then configure SFM proxies in the broker.

Related reference:

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"Broker-wide HTTP listener parameters" on page 3809
Select the resources and properties associated with the broker-wide HTTP listener that you want to change.

"ServiceFederationManager object property values" on page 3818
Select the properties associated with the ServiceFederationManager component that you want to change.

ServiceFederationManager object property values:

Select the properties associated with the ServiceFederationManager component that you want to change.

To change these properties, you must specify the execution group name and **-o ServiceFederationManager**.

port The insecure HTTP port that Service Federation Management (SFM) proxy input endpoints that are created in this connectivity provider are to use. You might select a particular port value to make that port available through a firewall, for example. If you do not specify a value or if you specify a value of 0, the broker selects the first unused port in the range 8810 - 8842.

- Value type - integer
- Initial value - 0

securePort
The secure HTTPS port that SFM proxy input endpoints that are created in this connectivity provider are to use. You might select a particular port value to make that port available through a firewall, for example. If you do not specify a value or if you specify a value of 0, the broker selects the first unused port in the range 8843 - 8872.

- Value type - integer
- Initial value - 0

maxWaitTime
The maximum time, in seconds, that an SFM proxy is to wait for the target service to respond to a request that it has sent. If this time is exceeded, the proxy returns a SOAP fault to the calling client.

- Value type - integer
- Initial value - 180

proxyURLHostName
The host name that is to appear in the input endpoint URL for an SFM proxy created in this connectivity provider. You might, typically, set this to a domain name that the service consumer should use to address the proxy, given that the service consumer might be accessing it from a remote network domain. An example is `myhost.location.ibm.com`.

- Value type - string
- Initial value - null

ProxyPathPrefix
The fixed path prefix that is to be appended with unique digits and prefixed to the input endpoint URL for an SFM proxy created in this connectivity provider. If set to null, no prefix is used in the proxy input path and the remainder of the proxy path becomes the target service path. For example, if a proxy is created for a service which has the endpoint URL `http:://host:7080/service` and the initial value `proxy` is used, the proxy URL that is created would be similar to `http:://myhost:8810/proxy1/service`.

- Value type - string
- Initial value - proxy

Related reference:

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"Parameter values for the servicefederation component" on page 3816
Select the objects and properties associated with the servicefederation component that you want to change.

mqsichangeresourcestats command:

Use the **mqsichangeresourcestats** command to control statistics gathering for resources in the broker.

Supported platforms:

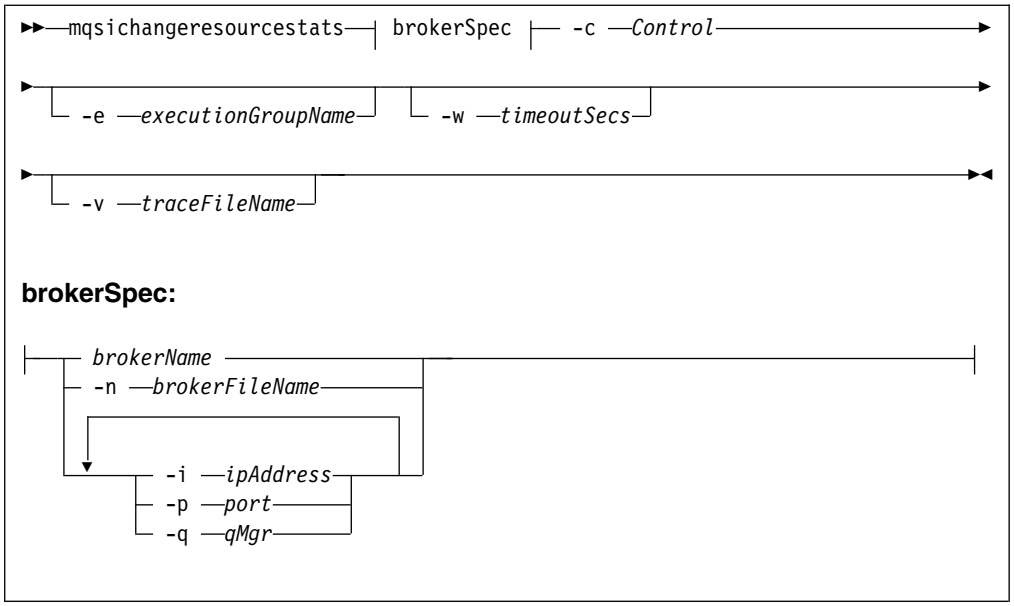
- Windows systems
- Linux and UNIX systems
- z/OS. Run this command by customizing and submitting BIPCHRS; see "Contents of the broker PDSE" on page 3991

Purpose:
Use this command to start or stop statistics collection for the resource types listed in "Resource statistics data" on page 6745.

You can run this command only if the broker is running. If you start statistics collection, then stop and restart the broker, you do not have to rerun this command; the active status is maintained when the broker starts again.

If you start statistics collection for all execution groups on a broker, you can stop collection on all execution groups or on individual execution groups. If you start statistics collection for individual execution groups, you can stop collection for those specific execution groups or for all execution groups.

Syntax:



Parameters:

brokerSpec
(Required) You must specify at least one parameter to identify the target broker for this command, in one of the following forms:

brokerName
This parameter identifies the name of a locally defined broker. You cannot use this option if the broker is on a remote computer.

-n *brokerFileName*
This parameter identifies the name of a file that contains the connection details for a local or remote broker.

Use this option if multiple users want to connect to the same broker, or if you are using advanced connection parameters such as SSL.

To create this file, right-click the broker in the WebSphere Message Broker Explorer and select **Export *.broker file**. When prompted, navigate to the location in which you want to save the file and enter the file name; the extension `.broker` is appended automatically when you click **Save**. Include the location (path) and file name when you specify this parameter. You must ensure that the location is accessible when you run this command.

If you want to run a command that uses SSL to administer a remote broker over a secured channel, you must specify the keystore and truststore password for the connection using the IBM_JAVA_OPTIONS environment variable. See "Resolving problems when running commands" on page 3364 for further information.

-i *ipAddress*, **-p** *port*, **-q** *qMgr*
These parameters identify the connection details for the broker.

Use this option for connections to remote brokers that do not require advanced connection parameters.

If you choose this option, you must specify at least one of these three parameters; the order in which you specify them is insignificant. You cannot specify each parameter more than once.

Parameters that you omit assume default values:

- **-i** *ipAddress*: The host name or the IP address of the computer on which the broker is running. If you do not specify this parameter, a value that represents the local computer is used.
- **-p** *port*: The TCP port on which the broker's queue manager is listening. If you do not specify this parameter, the value 1414 is used.
- **-q** *qMgr*: The name of the broker's queue manager. If you do not specify this parameter, a value that represents the default queue manager on the local computer is used.

You cannot connect to a remote queue manager on z/OS; see "Usage note" on page 3971.

-c *Control*
(Required) The value to define the action to be applied to resource statistics collection:

- Specify `active` to start resource statistics collection
- Specify `inactive` to stop resource statistics collection

-e *executionGroupName*
(Optional) The name of the execution group for which resource statistics collection is started or stopped.

If you do not specify **-e**, resource statistics collection is started or stopped for all execution groups on the broker.

-w *timeoutSecs*
(Optional) This parameter specifies the time in seconds that the utility waits to ensure that the command completed; the default value is 60.

-v *traceFileName*
(Optional) This parameter sends internal debug trace information to the specified file.

Usage note:
If you try to connect to a remote broker by specifying the **i**, **p**, and **q** parameters, or by using a connection parameter file (`.broker`), the command attempts to use WebSphere MQ Java client code. This option is not supported on z/OS, and returns the following error:

```
BIP1046E: Unable to connect with the broker (name)
```

The reported reason code is `MQRC_ENVIRONMENT_ERROR`. You must specify a local queue manager.

Authorization:
For information about platform-specific authorizations, see the following topics:

- "Security requirements for Linux and UNIX platforms" on page 3648
- "Security requirements for Windows systems" on page 3651
- "Security requirements for z/OS" on page 3655

If you have enabled broker administration security, you must also set up the authority detailed in "Tasks and authorizations for broker administration security" on page 3645.

Responses:
This command returns the following responses:

0 The command completed successfully.

2 (Failure) The broker received the deployment request but was unable to process it successfully. See the messages issued from the utility (or the Administration log) for more information.

9 (Failure) The request has been submitted to the broker, but no response was received before the timeout expired.

10 (Failure) Another user or application canceled the request operation before the broker was able to process it.

98 The broker is not running.

99 One or more of the parameters that you specified is invalid.

Examples:
Start resource statistics collection for all execution groups on `BrokerA`:

```
mqsichangeresourcestats BrokerA -c active
```

Stop resource statistics collection for the execution group `default` on broker `BrokerA`:

```
mqsichangeresourcestats BrokerA -c inactive -e default
```

:

Related tasks:

"Monitoring resource performance" on page 3305
You can collect statistics to assess the performance of certain resources used by execution groups.

Related reference:

"Syntax diagrams" on page 3677

"**mqsireportresourcestats** command" on page 3944
Use the **mqsireportresourcestats** command to display current statistics gathering options for resources in the broker.

"Example of an XML publication for resource statistics" on page 6746
This example message shows an XML publication that contains resource statistics data.

mqsichangetrace command:

Use the **mqsichangetrace** command to set the tracing characteristics for a broker.

Supported platforms:

- Windows
- Linux and UNIX systems
- z/OS - as a console command

Purpose:
This command is valid for:

- User trace. Specify the **-u** option.
- Service trace. Specify the **-t** option. Use this option only if directed to do so by the action described in a BIPxxxx message, or by your IBM Support Center.
- Trace nodes. Specify the **-n** option to switch Trace nodes on or off. You can significantly improve the performance of a flow by switching Trace nodes off.

You can initiate, modify, or end user (or service) tracing for a broker.

You can switch Trace nodes for a broker on and off, but you cannot use this command to initiate service tracing for the WebSphere Message Broker Toolkit.

If you specify a broker, or any of its resources (execution group or message flow), you must have deployed them before you can start trace. If you want to use the **mqsichangetrace** command to start or stop user trace, you must ensure the broker is running. But for service trace, you can also use the **mqsichangetrace** command to enable or disable trace when the broker is in a shutdown state. This allows you to trace the startup of the broker components.

The output for service and user trace generated by these commands is written to trace files in the `log` subdirectory. When you have completed the work you want to trace, use the **mqsireadlog** command to retrieve the log as an XML format file. Use either the **mqsiformatlog** command (to produce a formatted file), or an XML browser to view the XML records.

When you set tracing on, you cause additional processing to be executed for every activity in the broker that you are tracing. Expect performance to be affected when trace is active.

If you want to trace the command processes themselves, set the environment variables `MQSI_UTILITY_TRACE` and `MQSI_UTILITY_TRACESIZE` before you initiate trace.

You can set `MQSI_UTILITY_TRACE` to `normal` or `debug`, depending whether you want normal or debug tracing, and you should set `MQSI_UTILITY_TRACESIZE` to the maximum size of the trace file that you require in kilobytes (KB), the default being 102400 (100MB).

Ensure that you reset these variables when tracing for the selected command is complete. If you do not do so, all subsequent commands are also traced, and their performance degraded. For more information about MQSI_UTILITY_TRACE and MQSI_UTILITY_TRACESIZE, see "Starting service trace" on page 3534.

You can also start and stop tracing activity for execution groups and message flows using the facilities of the WebSphere Message Broker Toolkit. See "User trace" on page 6873 for more information.

If you want to view the tracing options that are currently in effect, use the **mqsireporttrace** command.

To enable service trace of your CMP applications, take one of the following steps:

- Call the method BrokerProxy.enableAdministrationAPITracing(String filename).
- Before running your CMP application, set the environment variable MQSI_CMP_TRACE to the name of the file to which trace is sent.

Select the appropriate link for details of this command on the platform, or platforms, that your enterprise uses:

- "**mqsichangetrace** command - Windows, Linux, and UNIX systems" on page 3824
- "**mqsichangetrace** command - z/OS" on page 3827

Authorization:
For information about platform-specific authorizations, see the following topics:

- "Security requirements for Linux and UNIX platforms" on page 3648

- "Security requirements for Windows systems" on page 3651
- "Security requirements for z/OS" on page 3655

If you have enabled broker administration security, you must also set up the authority detailed in "Tasks and authorizations for broker administration security" on page 3645.

Related tasks:

"Using trace" on page 3533
You can use different types of trace to help you with problem determination and troubleshooting.

"Switching Trace nodes on and off" on page 3555
Use the **mqsichangetrace** command or the WebSphere Message Broker Explorer to switch Trace nodes on and off.

Related reference:

"User trace" on page 6873
User trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Administration log. User trace is inactive by default; you must activate it explicitly by using a command, or by selecting options in the WebSphere Message Broker Toolkit.

"Service trace" on page 6875
Service trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Event Log or to user trace. Service trace is inactive by default; you must activate it explicitly by using a command.

"**mqsiformatlog** command" on page 3880
Use the **mqsiformatlog** command to process the XML log created by **mqsireadlog**. The command retrieves and formats any messages that the XML log contains into a form suitable for the locale of the user who runs the command.

"**mqsireadlog** command" on page 3905
Use the **mqsireadlog** command to retrieve trace records for the specified component.

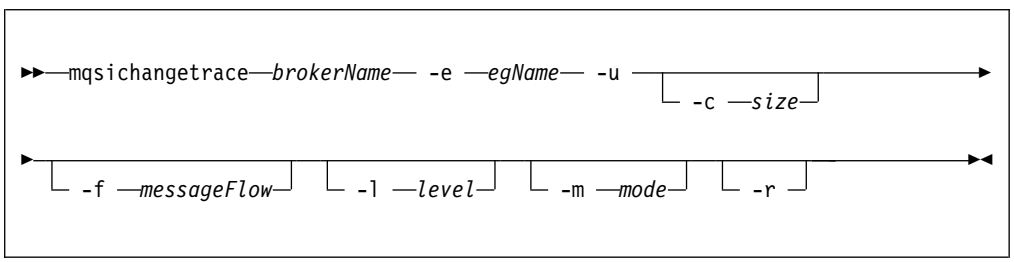
"**mqsireporttrace** command" on page 3947
Use the **mqsireporttrace** command to display the trace options currently in effect. Trace can be run only against a broker, or any of its resources.

mqsichangetrace *command - Windows, Linux, and UNIX systems:*

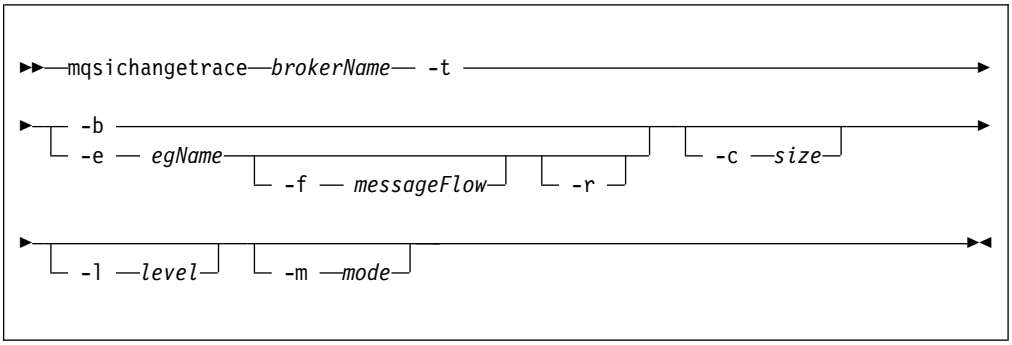
Use the **mqsichangetrace** command to set the tracing characteristics for a broker.

Syntax:

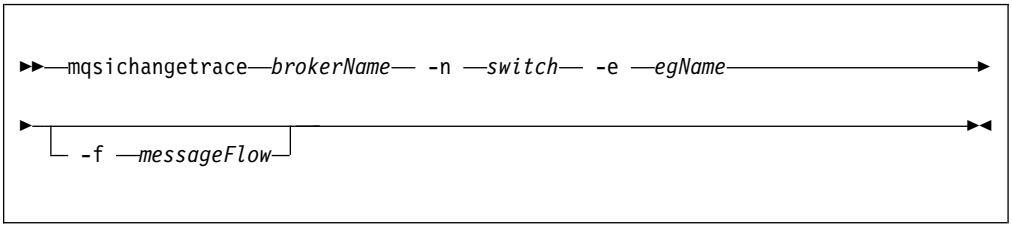
User trace:



Service trace:



Trace nodes:



Parameters:

brokerName
(Required) Specify the name of the broker that you want to trace.. All names are case sensitive on Linux and UNIX systems.

Key words `WebSphere Message Broker Toolkit` and `utility` are reserved, and must not be used as a broker name.

-c *size*

(Optional) The size of the trace file in KB (kilobytes). If you do not specify this parameter, the current value is left unchanged.

Each broker starts with a default value of 102400 KB. Specify this option to reset the value. The maximum value you can specify depends on how you subsequently intend to read the log, by using the **mqsireadlog** command;

- If you use this command with the **-f** option set, the log file is read directly from the file system. In this case, the maximum value that you can specify is 2097151, which allows a trace file up to 2 GB (gigabyte) to be created.
- If you use this command without setting the **-f** option, a WebSphere MQ message is sent to the broker to retrieve the log. In this case, do not allow the trace file to exceed 70 MB (megabytes). The maximum value that you can set is 70000.

On HP-UX, set the *size* value below 500 MB.

However you intend to retrieve the trace file, you might want to keep its size small, either by using a low value for this parameter, or by using the reset (**-r**) option on this command to clear the trace log. The benefit of adopting this approach is that the formatting process (**mqsiformatlog**) is much faster and requires less resource to carry out its task.

If you change this value, it affects tracing for the execution group (if you have specified one), or for the agent component (if you have not specified an execution group).

-e *egName*

(Required for user trace; required for service trace if you do not specify the -b flag)

Identifies the execution group for which trace options are to be modified (for example, started or stopped).

-f *messageFlow*

(Optional) Identifies the message flow for which trace options are to be modified. This option is valid only if you have specified an execution group (flag **-e**).

-l *level*

(Optional) Set the level of the trace. The following options are supported:

- `normal`. This option provides a basic level of trace information.
- `none`. This option switches tracing off.
- `debug`. This option provides a more comprehensive trace.

Each broker is created with a default value of `none`. If you do not specify this parameter, the current value is unchanged. When you have successfully changed this value, it is persistent.

-m *mode*

(Optional) Indicate the way trace information is to be buffered:

- `safe`. This mode causes trace entries to be written to file when they are generated.
- `fast`. This mode causes trace entries to be buffered, and written to file in batches.

Each broker starts with a default value of `safe`. If you do not specify this parameter, the current value is unchanged.

If you change this value, it affects tracing for the execution group (if you have specified one), or for the agent component (if you have not specified an execution group).

-r

(Optional) This option requests that the trace log is reset: that is, all current records are discarded. Use this option when you start a new trace to ensure that all records in the log are unique to the new trace.

This option is valid only if you have specified an execution group (flag **-e**).

-u (Required for user trace)

Specifies that user trace options are to be modified.

Additional parameters exclusive to service trace:
Use these options only when directed to do so by your IBM Support Center, or by a BIPxxxx message.

-b

(Required) Specifies that service trace options for the `agent` subcomponent of the broker specified are to be modified (for example, started or stopped). You can specify this flag only if **-t** is also specified.

You must specify the -b flag or the -e flag, but not both.

-m *mode*

(Optional) In addition to the `safe` and `fast` modes available for user trace, there is a third `temp` mode available with execution group service trace only:

- `temp`. This mode is the same as safe mode but trace is automatically switched off when the component restarts. You cannot specify a mode of `temp` if **-f** is also specified.

-t

(Required) Specifies that service trace options are to be modified.

Additional parameters exclusive to Trace nodes:

-n *switch*
(Required) Specifies the mode for trace flow. Valid values are `on` and `off`.

Examples:
To collect and process a user trace for the default execution group use the command:

```
mqsichangetrace MB7BROKER -u -e default -l normal -c 5000
```

To collect and process a service trace for flow f1 in the default execution group use the command:

```
mqsichangetrace MB7BROKER -t -e default -m fast
```

To collect and process a service trace for an agent use the command:

```
mqsichangetrace MB7BROKER -t -b -m -l normal
```

To switch off Trace nodes in the default execution group, use the command:

```
mqsichangetrace MB7BROKER –n off -e default
```

Related tasks:

"Switching Trace nodes on and off" on page 3555
Use the **mqsichangetrace** command or the WebSphere Message Broker Explorer to switch Trace nodes on and off.

Related reference:

"Syntax diagrams" on page 3677

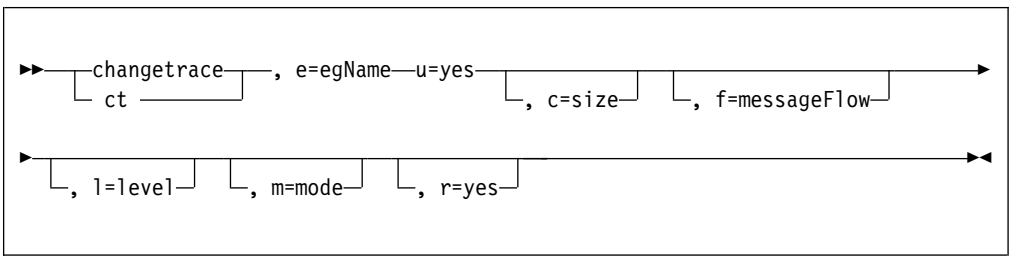
"**mqsichangetrace** command" on page 3822
Use the **mqsichangetrace** command to set the tracing characteristics for a broker.

mqsichangetrace *command - z/OS:*

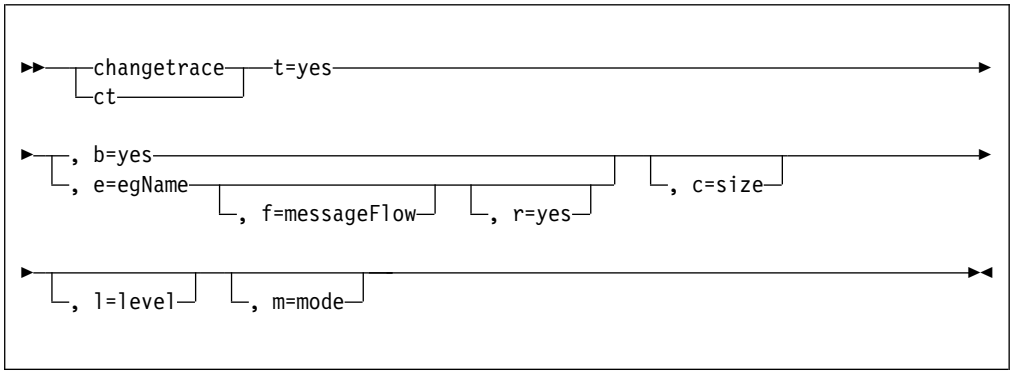
Use the **mqsichangetrace** command to set the tracing characteristics for a broker.

Syntax:

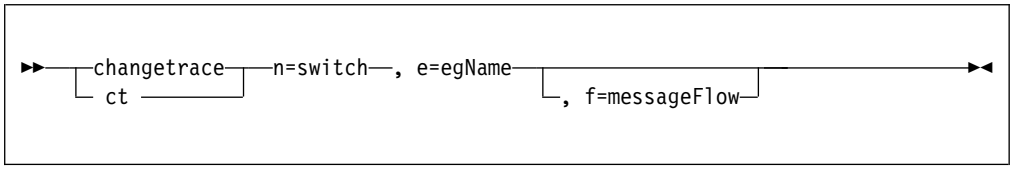
z/OS console command:
User trace



Service trace



Trace nodes:



Parameters:

-c *size*

(Optional) The size of the trace file in KB (kilobytes). If you do not specify this parameter, the current value is left unchanged.

Each broker starts with a default value of 102400 KB. Specify this option to reset the value. The maximum value you can specify depends on how you subsequently intend to read the log, by using the **mqsireadlog** command;

- If you use this command with the **-f** option set, the log file is read directly from the file system. In this case, the maximum value that you can specify is 2097151, which allows a trace file up to 2 GB (gigabyte) to be created.
- If you use this command without setting the **-f** option, a WebSphere MQ message is sent to the broker to retrieve the log. In this case, do not allow the trace file to exceed 70 MB (megabytes). The maximum value that you can set is 70000.

However you intend to retrieve the trace file, you might want to keep its size small, either by using a low value for this parameter, or by using the reset (**-r**) option on this command to clear the trace log. The benefit of adopting this approach is that the formatting process (**mqsiformatlog**) is much faster and requires less resource to carry out its task.

If you change this value, it affects tracing for the execution group (if you have specified one), or for the agent component (if you have not specified an execution group).

-e *egName*

(Required for user trace; optional for service trace)

Identifies the execution group for which trace options are to be modified (for example, started or stopped).

This name is case sensitive; you must include names in single quotes if they contains mixed case characters.

-f *messageFlow*

(Optional) Identifies the message flow for which trace options are to be modified. This option is valid only if you have specified an execution group (flag **-e**).

This name is case sensitive; you must include names in single quotes if they contain mixed case characters.

-l *level*

(Optional) Set the level of the trace. The following options are supported:

- `normal`. This option provides a basic level of trace information.
- `none`. This option switches tracing off.
- `debug`. This option provides a more comprehensive trace.

Each broker is created with a default value of `none`. If you do not specify this parameter, the current value is unchanged. When you have successfully changed this value, it is persistent.

-m *mode*

(Optional) Indicate the way trace information is to be buffered:

- `safe`. This mode causes trace entries to be written to file when they are generated.
- `fast`. This mode causes trace entries to be buffered, and written to file in batches.

Each broker starts with a default value of `safe`. If you do not specify this parameter, the current value is unchanged.

If you change this value, it affects tracing for the execution group (if you have specified one), or for the agent component (if you have not specified an execution group).

-r

(Optional) This option requests that the trace log is reset: that is, all current records are discarded. Use this option when you start a new trace to ensure that all records in the log are unique to the new trace.

This option is valid only if you have specified an execution group (flag **-e**).

-u (Required for user trace)

Specifies that user trace options are to be modified.

Additional parameters exclusive to service trace:
Use these options only when directed to do so by your IBM Support Center, or by a BIPxxxx message.

-b

(Required) Specifies that service trace options for the `agent` subcomponent of the broker specified are to be modified (for example, started or stopped). You can specify this flag only if **-t** is also specified.

-m *mode*

(Optional) In addition to the `safe` and `fast` modes available for user trace, there is a third `temp` mode available with execution group service trace only:

- `temp`. This mode is the same as safe mode but trace is automatically switched off when the component restarts. You cannot specify a mode of `temp` if **-f** is also specified.

-t

(Required) Specifies that service trace options are to be modified.

Additional parameters exclusive to Trace nodes:

n=*switch*

(Required) Specifies the mode for trace flow. Valid values are `on` and `off`.

Examples:
To collect and process a user trace for the default execution group use the command:

```
F MQP1BRK,ct U=YES, E='default', L=NORMAL, C=5000
```

and in the PDSE member BIPRELG, set the option for **mqsireadlog** to

```
-u -e default -f
```

To collect and process a service trace for flow f1 in the default execution group use the command:

```
F MQP1BRK,ct T=YES, E='default', F='F1', M=FAST, L=DEBUG
```

and in the PDSE member BIPRELG, set the option for **mqsireadlog** to

```
 -t -e default -f
```

To collect and process a service trace for an agent use the command:

```
F MQP1BRK,ct T=YES, B=YES, M=FAST, L=DEBUG
```

and in the PDSE member BIPRELG, set the option for **mqsireadlog** to

```
-t -b agent -f
```

To switch off Trace nodes for the default execution group, use the command:

```
F MQP1BRK,ct n='off', e='default'
```

Related tasks:

"Switching Trace nodes on and off" on page 3555
Use the **mqsichangetrace** command or the WebSphere Message Broker Explorer to switch Trace nodes on and off.

Related reference:

"Syntax diagrams" on page 3677

"**mqsichangetrace** command" on page 3822
Use the **mqsichangetrace** command to set the tracing characteristics for a broker.

mqsicommandconsole command:

Use the **mqsicommandconsole** command to launch an elevated command console from which commands that require elevation on Windows can be run.

Supported platforms:

- Windows

Purpose:
The **mqsicommandconsole** command is required for the following commands:

- **mqsiaddbrokerinstance**
- **mqsicreatebroker**
- **mqsimigratecomponents**

- **mqsisetsecurity**
- **mqsimanagexalinks**

Syntax:

Parameters:
None

Authorization:
On Windows XP and Windows Server 2003 systems, the user ID used to run this command must be a member of the group Administrators on the local system.

On Windows 7, Windows Vista, and Windows Server 2008 systems, the user ID used to run this command must be running with elevated privileges on the local system:

- The user ID must be a member of the group Administrators.
- The command must be run from an elevated command prompt. To obtain an elevated command prompt, run the mqsicommandconsole command and follow the Windows prompt to launch a new elevated command console.

  This console starts with the correct mqsiprofile environment from which commands that require elevation can be run.

If you do not run the command from within a privileged command console, the command will not process successfully.

Related concepts:

"Security overview" on page 351
When you are designing a WebSphere Message Broker application it is important to consider the security measures that are needed to protect the information in the system.

Related reference:

"Syntax diagrams" on page 3677

"Security requirements for Windows systems" on page 3651
Security requirements depend on the administrative task that you want to perform.

mqsicreatebroker command:

Use the **mqsicreatebroker** command to create a broker and its associated resources.

Supported platforms:

- Windows
- Linux and UNIX systems
- z/OS. Run this command by customizing and submitting BIPCRBK; see "Contents of the broker PDSE" on page 3991

Purpose:
The **mqsicreatebroker** command completes the following tasks.

1. The command checks whether the specified WebSphere MQ queue manager exists:
   - If the queue manager does not exist, the following behavior occurs.
     - If you run this command on z/OS, the action fails and an error is issued.
     - If you run this command on Linux, UNIX, or Windows, a queue manager is created.

       If WebSphere MQ Version 7.1, or later, has been selected for the queue manager, the channel auth security is automatically disabled.

       The queues that are created include a dead letter queue (DLQ), SYSTEM.DEAD.LETTER.QUEUE. The security settings are the same as for other broker-specific WebSphere MQ queues.

       If a message received by a message flow cannot be processed, it is typically backed out onto the input queue. If it cannot be backed out, or the message flow is not configured to back out messages, or to complete alternative error processing, the broker puts the message to the DLQ.

       The **mqsideletebroker** command does not delete the default DLQ (unless the queue manager is deleted).
   - If the queue manager does exist, check that the queue manager has a DLQ defined; the queue is not created by this command on an existing queue manager, but is required because the broker puts messages that cannot be processed to the DLQ.

     If you use WebSphere MQ clusters in your domain, define the queue manager before you run this command, and configure the queue manager in the cluster to benefit from reduced administration and increased availability.
     - If you run this command on Linux, UNIX, or Windows and the existing queue manager is WebSphere MQ Version 7.1, or later, it is assumed that the user has applied the appropriate security configuration to meet their requirements, and therefore channel auth security is not disabled.
2. The command starts the WebSphere MQ queue manager, if it is not already running, except on z/OS.

   If the command creates the queue manager on Windows, the queue manager is not started as a service. If you log off, the queue manager stops. Therefore, you must either remain logged on, or change the startup status of the queue manager service. If you lock your workstation, the WebSphere MQ queue manager does not stop.
3. The command connects to the associated queue manager.
4. The command creates the WebSphere MQ queues that are required by the broker, if they do not exist.

   If you enable administrative security, the queues required for this support are also created by this command.
5. On Windows only, the command installs a service under which the broker runs.
6. The command creates the broker in one of the available modes. If the full package is installed, the default mode is `enterprise`. If the trial package is installed, the default mode is `trial`. For more information, see "Operation modes" on page 48.
7. The command creates a record for the component in the registry.
8. On Windows systems, Linux, and UNIX systems, the command allows you to specify, when the broker is created, whether the broker can be started and stopped as a WebSphere MQ service.

Brokers can access only local queue managers, so you cannot create a broker on a queue manager that is on a remote system.

Usage notes:
If you have migrated from Version 6.1 or Version 6.0, the following restrictions apply.

In previous versions of WebSphere Message Broker, parameters that you specified on this command were used by the broker to provide default authorization for database access: either the database user ID and password (if specified) or the service user ID and password. Because the Version 7.0 broker does not use a database for its own purposes, the database user ID and password parameters have been deprecated. The broker service ID and password parameters have also been deprecated, except on Windows.

If you migrate a broker from a previous release, the associated values are stored and used for default user database access as though you had entered these values by using the **mqsisetdbparms** command.

When you create a Version 7.0 broker, you must use the **mqsisetdbparms** command to set up database access authorization.

When you create a multi-instance broker, the resources are stored on a shared file system. The user that issues the **mqsicreatebroker** command must have the correct permissions on the shared file system to create these resources.

For details of this command on the operating system that your enterprise uses, see the appropriate topic.

- "**mqsicreatebroker** command - Linux and UNIX systems" on page 3835
- "**mqsicreatebroker** command - Windows systems" on page 3840
- "**mqsicreatebroker** command - z/OS" on page 3845

Authorization:
For information about platform-specific authorizations, see the following topics:

- "Security requirements for Linux and UNIX platforms" on page 3648
- "Security requirements for Windows systems" on page 3651
- "Security requirements for z/OS" on page 3655

If you have enabled broker administration security, you must also set up the authority detailed in "Tasks and authorizations for broker administration security" on page 3645.

WebSphere MQ queues created:

- SYSTEM.BROKER.ADAPTER.FAILED
- SYSTEM.BROKER.ADAPTER.INPROGRESS
- SYSTEM.BROKER.ADAPTER.NEW
- SYSTEM.BROKER.ADAPTER.PROCESSED
- SYSTEM.BROKER.ADAPTER.UNKNOWN
- SYSTEM.BROKER.ADMIN.QUEUE
- SYSTEM.BROKER.ADMIN.REPLYTODM
- SYSTEM.BROKER.AGGR.CONTROL
- SYSTEM.BROKER.AGGR.REPLY
- SYSTEM.BROKER.AGGR.REQUEST
- SYSTEM.BROKER.AGGR.TIMEOUT
- SYSTEM.BROKER.AGGR.UNKNOWN

- SYSTEM.BROKER.AUTH
- SYSTEM.BROKER.CONTROL.QUEUE
- SYSTEM.BROKER.DEPLOY.QUEUE
- SYSTEM.BROKER.DEPLOY.REPLY
- SYSTEM.BROKER.EDA.COLLECTIONS
- SYSTEM.BROKER.EDA.EVENTS
- SYSTEM.BROKER.EXECUTIONGROUP.QUEUE
- SYSTEM.BROKER.EXECUTIONGROUP.REPLY
- SYSTEM.BROKER.INTER.BROKER.COMMUNICATIONS
- SYSTEM.BROKER.MODEL.QUEUE
- SYSTEM.BROKER.TIMEOUT.QUEUE
- SYSTEM.BROKER.WS.ACK
- SYSTEM.BROKER.WS.INPUT
- SYSTEM.BROKER.WS.REPLY

Access authority is granted for the WebSphere Message Broker group mqbrkrs to all these queues. If the DLQ is enabled, it also has the same authority.

Responses:
If you run the **mqsicreatebroker** command and it fails, resolve the problem that caused the failure:

- Check responses; see "Responses to commands" on page 3682.
- Check the error logs; see "Local error logs" on page 6867.
- Check the error messages in the error log; you can search for error messages in the information center.

When you run the same command again, you might receive a series of messages specifying items that cannot be created. Receiving these messages does not indicate a problem with the **mqsicreatebroker** command itself.

Related concepts:

"Operation modes" on page 48
The operation mode that you use for your broker is determined by the license that you purchase.

"User exits" on page 3015
A user exit is user-provided custom software, written in C, to track data passing through message flows.

"Logs" on page 6864
If an error is reported by a WebSphere Message Broker component, start your investigations into its causes by looking at the product and systems logs to which information is written during component operation.

Related tasks:

"Creating a broker" on page 611
You can create brokers on every platform that is supported by WebSphere Message Broker. The broker runs as a 64-bit application on all platforms except Linux on x86 and Windows on x86.

Related reference:

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

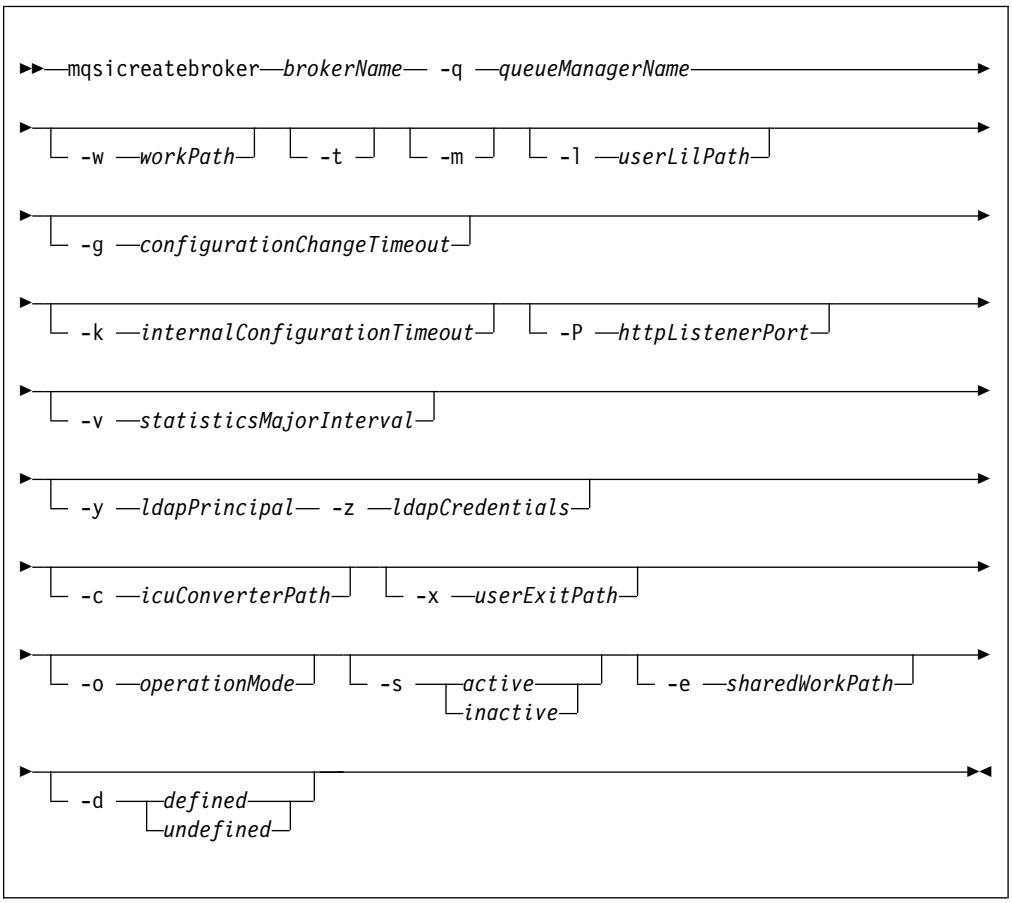
"**mqsideletebroker** command" on page 3863
Use the **mqsideletebroker** command to delete a named broker. The command also deletes the queues on the associated queue manager (created when the broker was created). You can also specify that the queue manager is to be deleted.

"**mqsireportbroker** command" on page 3919
Use the **mqsireportbroker** command to display broker registry entries.
"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.
"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

mqsicreatebroker *command - Linux and UNIX systems:*

Use the **mqsicreatebroker** command to create a broker on a Linux or UNIX systems.

Syntax:



Parameters:

brokerName
(Required) The name of the broker that you are creating. This parameter must be the first parameter, and it is case sensitive.

For restrictions on the character set that you can use, see "Characters allowed in commands" on page 3680.

-q *queueManagerName*
(Required) The name of the queue manager, (or multi-instance queue manager

if creating a multi instance broker) that is associated with this broker instance. Use the same name for your broker and the queue manager to simplify the organization and administration of your network. Queue manager names are limited to 48 characters in length, and they are case sensitive.

The default behavior is to create a queue manager using the default `mqm` path, if the queue manager does not already exist on the same server.

When creating a multi-instance broker where the queue manager does not exist on the server, a multi-instance queue manager is created beneath the multi-instance broker shared work path using the WebSphere MQ **crtmqm** command as follows:

```
crtmqm –md /<broker sharedWorkPath>/mqm/qmdata
       -ld //<broker sharedWorkPath>/mqm/qmlog queueManagerName
```

If this shared queue manager path is not appropriate, create the multi-instance queue manager on the server before you run this command.

The queue manager attribute `MAXMSGLEN` (the maximum length of messages that can be put to queues) is updated to 100 MB. This attribute is updated regardless of whether the queue manager is created by this command.

For restrictions on the character set that you can use, see "Characters allowed in commands" on page 3680.

-w *workPath*
(Optional) The directory in which working files for this broker are stored. If you do not specify this parameter, files are stored in the default work path, which was specified when the product was installed. If you specify this parameter, you must create this directory before you start the broker.

When a broker has been enabled for multi-instance mode using the –e flag, the broker `workPath` is divided between data that is specific to this broker instance, and that which is shared between this broker and any of its instances created using the **mqsiaddbrokerinstance** command. Data specific to the multi-instance enabled broker is stored in the `workPath` directory on the local server, whereas the shared data is held in a directory on network storage at the location specified using the –e flag.

This directory is also used for trace records that are created when tracing is active. These records are written to a subdirectory, `log`, which you must create before you start the broker.
Error logs that are written by the broker when a process ends abnormally are stored in this directory.
The error log is unbounded and continues to grow. Check this directory periodically and clear out old error information.
You cannot change this parameter using the **mqsichangebroker** command. To specify or change the work path, delete and re-create the broker.
Specifying this parameter creates a separate working directory for the broker. This working directory is a subset of the default working directory structure that contains fewer subdirectories and no `common\profiles` subdirectory.

-t (Optional) The broker runs as a WebSphere MQ trusted application.

If you specify this parameter on HP-UX and Solaris, specify the *serviceUserId* as `mqm`.

For more details about using WebSphere MQ trusted applications, see the *Intercommunication* section of the WebSphere MQ Version 7 Information Center online.

-l *userLilPath*
(Optional) A list of paths (directories) from which the broker loads Loadable implementation libraries (LIL files) for user-defined message processing nodes.

Directory names are case sensitive, and you must include the names in single quotation marks if they contain mixed case characters.
Do not include environment variables in the path; the broker ignores them.

Create your own directory for storing your `.lil` or `.jar` files. Do not save them in the WebSphere Message Broker installation directory.
If you specify more than one directory, separate directories by using a colon (`:`).

-g *configurationChangeTimeout*
(Optional) The maximum time (in seconds) that is allowed for a user configuration request to be processed. It defines the length of time taken within the broker to apply to an execution group a configuration change that you have initiated. For example, if you deploy a configuration from the WebSphere Message Broker Toolkit, the broker must respond to the Configuration Manager within this time.

A message flow cannot respond to a configuration change while it is processing an application message. An execution group returns a negative response to the deployed configuration message if any one of its message flows does not finish processing an application message and apply the configuration change within this timeout.

Specify the value in seconds, in the range 10 - 3600. The default is 300.

For information about how to set the value for this timeout, see "Setting configuration timeout values" on page 3258.

-k *internalConfigurationTimeout*
(Optional) The maximum time (in seconds) that is allowed for an internal configuration change to be processed. For example, it defines the length of time taken within the broker to start an execution group.

The response time of each execution group differs according to system load and the load of its own processes. The value must reflect the longest response time that any execution group takes to respond. If the value is too low, the broker returns a negative response, and might issue error messages to the local error log.
Specify the value in seconds, in the range 10 - 3600. The default is 60.
For information about how to set the value for this timeout, see "Setting configuration timeout values" on page 3258.

-P *httpListenerPort*
(Optional) Enter the number of the port on which the Web services support is listening.

The broker starts this listener when a message flow that includes HTTP nodes or Web services support is started; the default is 7080.

Ensure that the port that you specify has not been specified for any other purpose.

-v *statisticsMajorInterval*
(Optional) Specify the interval (in minutes) at which statistics and accounting archive records are to be written. The valid range is from 1 minute to 43200 minutes; the default value is 60.

-y *ldapPrincipal*
(Optional, but mandatory when *ldapCredentials* is provided.) The user principal for access to an optional LDAP directory that holds the JNDI administered Initial Context for the JMS provider.

-z *ldapCredentials*
(Optional, but mandatory when *ldapPrincipal* is provided.) The user password for access to LDAP.

-c *icuConverterPath*
(Optional) A set of directories to search for additional code page converters, delimited by a colon (:).

Do not use this parameter to set the converter path if you are using a converter that matches one of the built-in converters that are provided, and that converter is the local code page for the broker. Use the **ICU_DATA** environment variable instead.

-x *userExitPath*
(Optional) The path that contains the location of all user exits to be loaded for execution groups in this broker. This path is added to the system library search path (`PATH,LIBPATH,LD_LIBRARY_PATH,SHLIBPATH`) for the execution group process only.

-o *operationMode*
(Optional) Use this parameter to set the mode of your broker; for more information, see "Operation modes" on page 48. Valid values are `enterprise` (the full package, enterprise mode), `starter` (Starter Edition mode), `entry` (Entry Edition mode), and `adapter` (Remote Adapter Deployment mode). The default value is `enterprise`, unless you have the Trial Edition, in which case the default value is `trial`. If you do not set the **-o** parameter, the default value is used automatically.

-s (Optional) Specify the administrative security status for the broker.

If you specify **-s active**, administration security is enabled. Only user IDs that you authorize are permitted to complete actions on the broker. Read, write, and execute authority is always granted on the security queue SYSTEM.BROKER.AUTH to all user IDs that belong to the security group mqbrkrs. When the broker has been created, you can add further user ID authorizations to this queue.

When you create an execution group on a broker for which administrative security is enabled, the queue SYSTEM.BROKER,AUTH.*egroup_name* is created. Populate the queue with the appropriate user authorization.

If you specify **-s inactive**, or omit this parameter, broker administration security is not enabled. All users are able to complete all actions against the broker and all execution groups.

For further information about using security, see "Broker administration security overview" on page 362 and "Authorizing users for broker administration" on page 371.

-e *sharedWorkPath*
(Optional) Setting this value enables the broker for the multi-instance mode of operation.

You must ensure the broker has access to this network storage location before you start the broker, and that the queue manager for the broker has been configured as a WebSphere MQ multi-instance queue manager. The information stored in this shared directory includes:

- The broker registry
- Component directories
- Internal broker tables and files for deployed message flows
- Configurable service properties.

-d (Optional) Specify whether you enable a broker to be started and stopped as a WebSphere MQ service when the queue manager starts and stops.

This option is an alternative to starting a multi-instance broker in standby mode using the **mqsistart** command.

If you specify **-d defined**, the WebSphere MQ service is defined to the queue manager and the broker starts and stops when the queue manager starts and stops.

If you specify **-d undefined**, the WebSphere MQ service is not defined to the queue manager and the broker does not start and stop when the queue manager starts and stops. This is the default setting.

-i *serviceUserId*
(Deprecated) This parameter is included for compatibility with earlier versions.

-a *servicePassword*
(Deprecated) This parameter is included for compatibility with earlier versions.

Examples:
Create a broker to run as a trusted application:

```
mqsicreatebroker MB7BROKER
 -q MB7QMGR -t
```

Create a broker that references user exits:

```
mqsicreatebroker MB7BROKER
 -q MB7QMGR -x /opt/3rdparty/wmbexits
```

Create a broker with administrative security enabled:

```
mqsicreatebroker MB7BROKER
 -q MB7QMGR -s active
```

Create a broker, by using the multi-instance queue manager `MyQMGR` and the shared work path `MyNetworkSharedWorkpath`, where the broker starts as a WebSphere MQ service:

```
mqsicreatebroker MB7BROKER
 -q MyQmgr -e MyNetworkSharedWorkpath -d defined
```

Related concepts:

"Logs" on page 6864
If an error is reported by a WebSphere Message Broker component, start your investigations into its causes by looking at the product and systems logs to which information is written during component operation.

Related tasks:

"Setting configuration timeout values" on page 3258
Change timeout values that affect configuration tasks in the broker.

Related reference:

"Syntax diagrams" on page 3677

"**mqsiaddbrokerinstance** command" on page 3715
Use the **mqsiaddbrokerinstance** command to create a multi-instance broker on a server where WebSphere Message Broker has been installed.

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

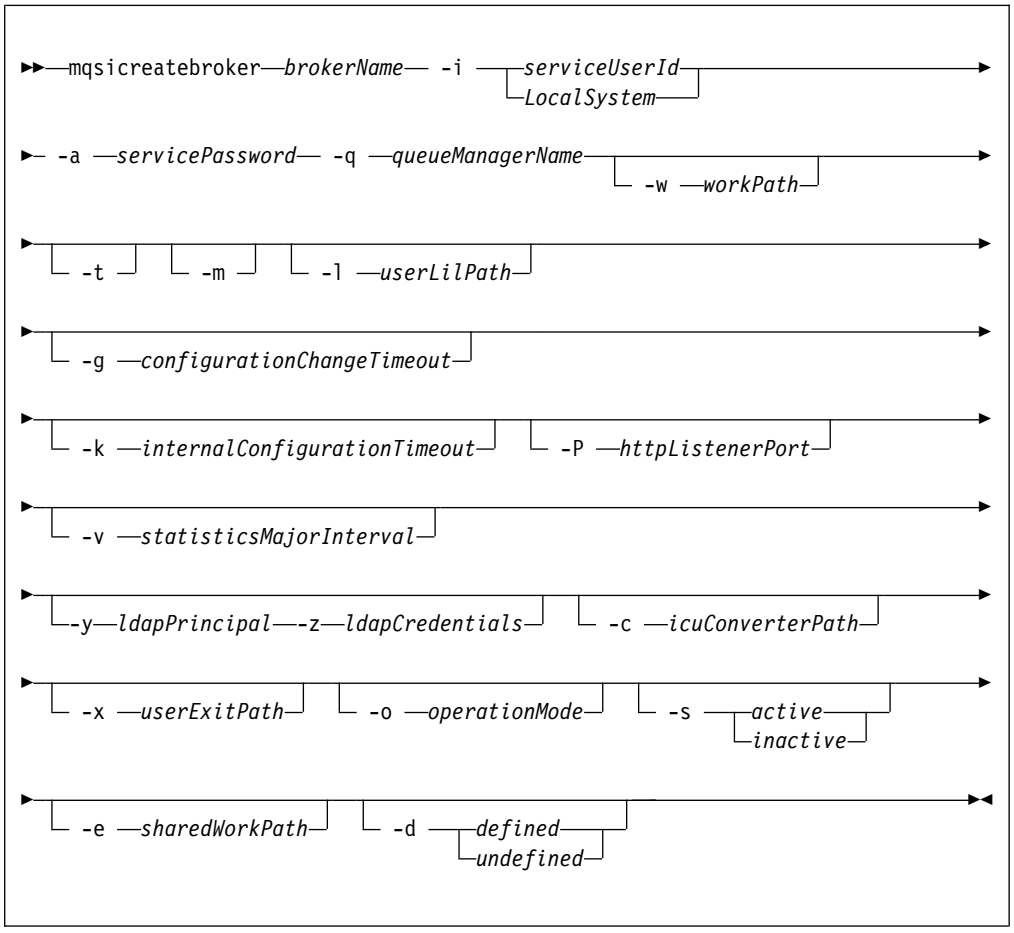
"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsideletebroker** command" on page 3863
Use the **mqsideletebroker** command to delete a named broker. The command also deletes the queues on the associated queue manager (created when the broker was created). You can also specify that the queue manager is to be deleted.

mqsicreatebroker *command - Windows systems:*

Use the **mqsicreatebroker** command to create a broker on a Windows system.

Syntax:



Parameters:

brokerName
(Required) The name of the broker that you are creating. This parameter must be the first parameter. If you create a broker with an uppercase name, you must also specify the name in uppercase in the WebSphere Message Broker Toolkit.

For restrictions on the character set that you can use, see "Characters allowed in commands" on page 3680.

-i *serviceUserId*
(Optional) The user ID under which the broker runs.

You can specify the *serviceUserId* in any valid user name syntax:

- `\\server\username`
- `.\username`
- `username`

Do not use a domain name as part of the *serviceUserId* parameter.
If you use the unqualified form for this user ID (`username`), the operating system searches for the user ID throughout its domain, starting with the local system. This search might take some time to complete.
The *serviceUserId* that you specify must be a direct or indirect member of the mqbrkrs local group. The *serviceUserId* must also be authorized to access the home directory (where WebSphere Message Broker has been installed), and the working directory (if specified by the **-w** parameter).
If you specify that the broker is to run as a WebSphere MQ trusted application (**-t** parameter), you must also add the service user ID to the mqm group.
The security requirements for the *serviceUserId* are described in "Security requirements for Windows systems" on page 3651.

-i *LocalSystem*
(Optional) You can specify *LocalSystem* instead of *serviceUserId*.

If you specify *LocalSystem*, you must use the *servicePassword* parameter, however, the value of the *servicePassword* parameter is ignored.

If you specify the **-e** parameter, *LocalSystem* should not be used. When specifying the **-e** parameter, you must use a real userid for the **-i** option.

For Windows, only the **-i** *LocalSystem* option is available. If you specify the **-e** parameter for a multi-instance broker, the system issues an error (BIP8022E: Invalid service userid and password combination supplied).

Note: Either the *Localsystem* or *serviceUserId* option must be specified for the **-i** parameter.

-a *servicePassword*
(Required) The password for the *serviceUserId*.

For compatibility with existing systems, you can specify `<password>`. However, if you do not specify a password with this parameter when you run the command, you are prompted to enter a password. You must enter the password a second time to verify that you have entered it correctly.

-q *queueManagerName*
(Required) The name of the queue manager, (or multi-instance queue manager if creating a multi instance broker) that is associated with this broker instance. Use the same name for your broker and the queue manager to simplify the organization and administration of your network. Queue manager names are limited to 48 characters in length, and they are case sensitive.

The default behavior is to create a queue manager using the default `mqm` path, if the queue manager does not already exist on the same server.

When creating a multi-instance broker where the queue manager does not exist on the server, a multi-instance queue manager is created beneath the multi-instance broker shared work path using the WebSphere MQ **crtmqm** command as follows:

```
crtmqm –md /<broker sharedWorkPath>/mqm/qmdata
       -ld //<broker sharedWorkPath>/mqm/qmlog queueManagerName
```

If this shared queue manager path is not appropriate, create the multi-instance queue manager on the server before you run this command.

The queue manager attribute `MAXMSGLEN` (the maximum length of messages that can be put to queues) is updated to 100 MB. This attribute is updated regardless of whether the queue manager is created by this command.

For restrictions on the character set that you can use, see "Characters allowed in commands" on page 3680.

-w *workPath*
(Optional) The directory in which working files for this broker are stored. If you do not specify this parameter, files are stored in the default work path, which was specified when the product was installed. If you specify this parameter, you must create this directory before you start the broker.

When a broker has been enabled for multi-instance mode using the –e flag, the broker `workPath` is divided between data that is specific to this broker instance, and that which is shared between this broker and any of its instances created using the **mqsiaddbrokerinstance** command. Data specific to the multi-instance enabled broker is stored in the `workPath` directory on the local server, whereas the shared data is held in a directory on network storage at the location specified using the –e flag.

This directory is also used for trace records that are created when tracing is active. These records are written to a subdirectory, `log`, which you must create before you start the broker.
Error logs that are written by the broker when a process ends abnormally are stored in this directory.
The error log is unbounded and continues to grow. Check this directory periodically and clear out old error information.
You cannot change this parameter using the **mqsichangebroker** command. To specify or change the work path, delete and re-create the broker.
Specifying this parameter creates a separate working directory for the broker. This working directory is a subset of the default working directory structure that contains fewer subdirectories and no `common\profiles` subdirectory.

-t (Optional) The broker runs as a WebSphere MQ trusted application.

If you specify this parameter, add the *serviceUserId* (identified by the **-i** parameter) to the mqm group.

For more details about using WebSphere MQ trusted applications, see the *Intercommunication* section of the WebSphere MQ Version 7 Information Center online.

-l *userLilPath*
(Optional) A list of paths (directories) from which the broker loads Loadable implementation libraries (LIL files) for user-defined message processing nodes.

Do not include environment variables in the path; the broker ignores them.

Create your own directory for storing your `.lil` or `.jar` files. Do not save them in the WebSphere Message Broker installation directory.
If you specify more than one directory, separate directories by using a semicolon (`;`).

-g *configurationChangeTimeout*
(Optional) The maximum time (in seconds) that is allowed for a user configuration request to be processed. It defines the length of time taken within the broker to apply to an execution group a configuration change that you have initiated. For example, if you deploy a configuration from the WebSphere Message Broker Toolkit, the broker must respond to the Configuration Manager within this time.

A message flow cannot respond to a configuration change while it is processing an application message. An execution group returns a negative response to the deployed configuration message if any one of its message flows does not finish processing an application message and apply the configuration change within this timeout.

Specify the value in seconds, in the range 10 - 3600. The default is 300.

For information about how to set the value for this timeout, see "Setting configuration timeout values" on page 3258.

-k *internalConfigurationTimeout*
(Optional) The maximum time (in seconds) that is allowed for an internal configuration change to be processed. For example, it defines the length of time taken within the broker to start an execution group.

The response time of each execution group differs according to system load and the load of its own processes. The value must reflect the longest response time that any execution group takes to respond. If the value is too low, the broker returns a negative response, and might issue error messages to the local error log.
Specify the value in seconds, in the range 10 - 3600. The default is 60.
For information about how to set the value for this timeout, see "Setting configuration timeout values" on page 3258.

-P *httpListenerPort*
(Optional) Enter the number of the port on which the Web services support is listening.

The broker starts this listener when a message flow that includes HTTP nodes or Web services support is started; the default is 7080.

Ensure that the port that you specify has not been specified for any other purpose.

-v *statisticsMajorInterval*
(Optional) Specify the interval (in minutes) at which statistics and accounting archive records are to be written. The valid range is from 1 minute to 43200 minutes; the default value is 60.

-y *ldapPrincipal*
(Optional, but mandatory when *ldapCredentials* is provided.) The user principal for access to an optional LDAP directory that holds the JNDI administered Initial Context for the JMS provider.

-z *ldapCredentials*
(Optional, but mandatory when *ldapPrincipal* is provided.) The user password for access to LDAP.

-c *icuConverterPath*
(Optional) A delimited set of directories to search for additional code page converters. On Windows systems, the delimiter is a semicolon (;). On UNIX and Linux systems, the delimiter is a colon (:).

Do not use this parameter to set the converter path if you are using a converter that matches one of the built-in converters that are provided, and that converter is the local code page for the broker. Use the **ICU_DATA** environment variable instead.

-x *userExitPath*
(Optional) The path that contains the location of all user exits to be loaded for execution groups in this broker. This path is added to the system library search path (`PATH,LIBPATH,LD_LIBRARY_PATH,SHLIBPATH`) for the execution group process only.

-o *operationMode*
(Optional) Use this parameter to set the mode of your broker; see "Operation modes" on page 48. Valid values that you can set are `enterprise` (the full package, enterprise mode), `entry` (Entry Edition mode), `starter` (Starter Edition mode), and `adapter` (Remote Adapter Deployment mode). The default value is `enterprise`, unless you have the Trial Edition, in which case the default value is `trial`. If you do not set the **-o** parameter, the default value is used automatically.

-s (Optional) Specify the administrative security status for the broker.

If you specify **-s active**, administration security is enabled. Only user IDs that you authorize are permitted to complete actions on the broker. Read, write, and execute authority is always granted on the security queue SYSTEM.BROKER.AUTH to all user IDs that belong to the security group mqbrkrs. When the broker has been created, you can add further user ID authorizations to this queue.

When you create an execution group on a broker for which administrative security is enabled, the queue SYSTEM.BROKER,AUTH.*egroup_name* is created. Populate the queue with the appropriate user authorization.

If you specify **-s inactive**, or omit this parameter, broker administration security is not enabled. All users are able to complete all actions against the broker and all execution groups.

For further information about using security, see "Broker administration security overview" on page 362 and "Authorizing users for broker administration" on page 371.

-e *sharedWorkPath*
(Optional) Setting this value enables the broker for the multi-instance mode of operation.

You must ensure the broker has access to this network storage location before you start the broker, and that the queue manager for the broker has been configured as a WebSphere MQ multi-instance queue manager. The information stored in this shared directory includes:

- The broker registry
- Component directories
- Internal broker tables and files for deployed message flows
- Configurable service properties.

-d (Optional) Specify whether you enable a broker to be started and stopped as a WebSphere MQ service when the queue manager starts and stops.

This option is an alternative to starting a multi-instance broker in standby mode using the **mqsistart** command.

If you specify **-d defined**, the WebSphere MQ service is defined to the queue manager and the broker starts and stops when the queue manager starts and stops.

If you specify **-d undefined**, the WebSphere MQ service is not defined to the queue manager and the broker does not start and stop when the queue manager starts and stops. This is the default setting.

Examples:
Create a broker to run as a trusted application:

```
mqsicreatebroker MB7BROKER -i wbrkuid -a wbrkpw -q MB7QMGR -t
```

Create a broker that references user exits:

```
mqsicreatebroker MB7BROKER -i wbrkuid -a wbrkpw
 -q MB7QMGR -x /opt/3rdparty/wmbexits
```

Create a broker with administrative security enabled:

```
mqsicreatebrokerMB7BROKER -q MB7QMGR -s active
```

Related concepts:

"Logs" on page 6864
If an error is reported by a WebSphere Message Broker component, start your investigations into its causes by looking at the product and systems logs to which information is written during component operation.

Related tasks:

"Setting configuration timeout values" on page 3258
Change timeout values that affect configuration tasks in the broker.

Related reference:

"Syntax diagrams" on page 3677

"**mqsiaddbrokerinstance** command" on page 3715
Use the **mqsiaddbrokerinstance** command to create a multi-instance broker on a server where WebSphere Message Broker has been installed.

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

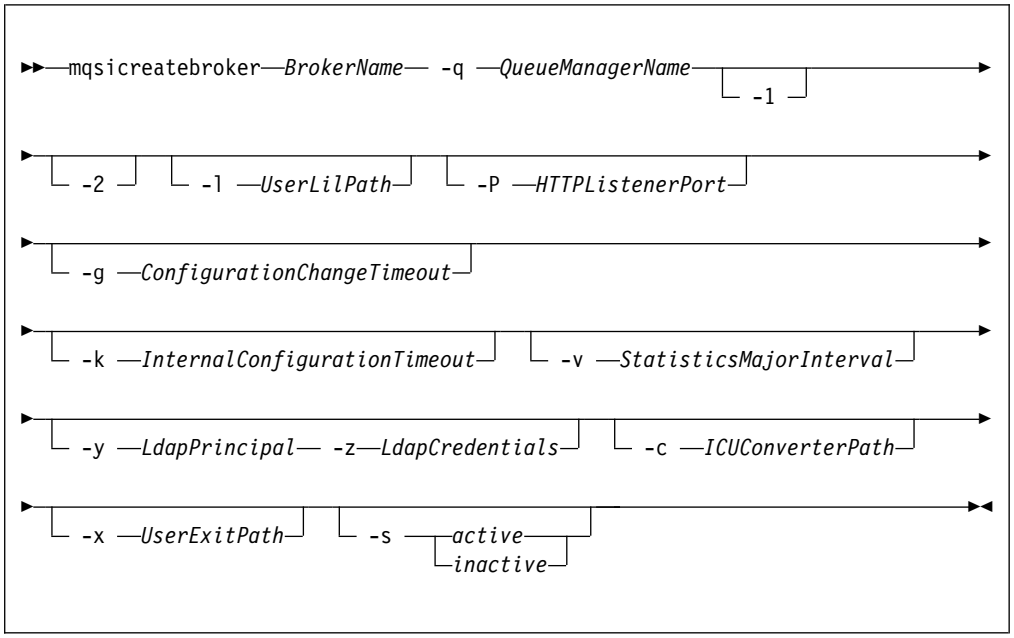
"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsideletebroker** command" on page 3863
Use the **mqsideletebroker** command to delete a named broker. The command also deletes the queues on the associated queue manager (created when the broker was created). You can also specify that the queue manager is to be deleted.

mqsicreatebroker *command - z/OS:*

Use the **mqsicreatebroker** command to create a broker on a z/OS system.

Syntax:
z/OS command - BIPCRBK



Parameters:

BrokerName
(Required) The name of the broker that you are creating. This parameter must be the first parameter. If you create a broker with an uppercase name, you must also specify the name in uppercase in the WebSphere Message Broker Toolkit.

For restrictions on the character set that you can use, see "Characters allowed in commands" on page 3680.

-q *QueueManagerName*
(Required) The name of the queue manager, (or multi-instance queue manager if creating a multi instance broker) that is associated with this broker instance. Use the same name for your broker and the queue manager to simplify the organization and administration of your network. Queue manager names are limited to 48 characters in length, and they are case sensitive.

The default behavior is to create a queue manager using the default `mqm` path, if the queue manager does not already exist on the same server.

When creating a multi-instance broker where the queue manager does not exist on the server, a multi-instance queue manager is created beneath the multi-instance broker shared work path using the WebSphere MQ **crtmqm** command as follows:

```
crtmqm –md /<broker sharedWorkPath>/mqm/qmdata
       -ld //<broker sharedWorkPath>/mqm/qmlog queueManagerName
```

If this shared queue manager path is not appropriate, create the multi-instance queue manager on the server before you run this command.

The queue manager attribute `MAXMSGLEN` (the maximum length of messages that can be put to queues) is updated to 100 MB. This attribute is updated regardless of whether the queue manager is created by this command.

For restrictions on the character set that you can use, see "Characters allowed in commands" on page 3680.

-l *UserLilPath*
(Optional) A list of paths (directories) from which the broker loads Loadable implementation libraries (LIL files) for user-defined message processing nodes.

This name is case sensitive; enclose the names in single quotation marks if they are in mixed case.
Do not include environment variables in this path; WebSphere Message Broker ignores them.

You must create your own directory for storing your `.lil` or `.jar` files. Do not save these files in the WebSphere Message Broker install directory.

-P *HTTPListenerPort*
(Optional) Enter the number of the port on which the Web services support is listening.

The broker starts this listener when a message flow that includes HTTP nodes or Web services support is started; the default is 7080.

Ensure that the port that you specify has not been specified for any other purpose.

-g *ConfigurationChangeTimeout*
(Optional) The maximum time (in seconds) that is allowed for a user configuration request to be processed. It defines the length of time taken within the broker to apply to an execution group a configuration change that you have initiated. For example, if you deploy a configuration from the WebSphere Message Broker Toolkit, the broker must respond to the Configuration Manager within this time.

A message flow cannot respond to a configuration change while it is processing an application message. An execution group returns a negative response to the deployed configuration message if any one of its message flows does not finish processing an application message and apply the configuration change within this timeout.

Specify the value in seconds, in the range 10 - 3600. The default is 300.

For information about how to set the value for this timeout, see "Setting configuration timeout values" on page 3258.

-k *InternalConfigurationTimeout*
(Optional) The maximum time (in seconds) that is allowed for an internal configuration change to be processed. For example, it defines the length of time taken within the broker to start an execution group.

The response time of each execution group differs according to system load and the load of its own processes. The value must reflect the longest response time that any execution group takes to respond. If the value is too low, the broker returns a negative response, and might issue error messages to the local error log.
Specify the value in seconds, in the range 10 - 3600. The default is 60.
For information about how to set the value for this timeout, see "Setting configuration timeout values" on page 3258.

-v *StatisticsMajorInterval*
(Optional) Specify the interval (in minutes) at which statistics and accounting archive records are to be written. The valid range is from 1 minute to 43200 minutes; the default value is 60.

An interval of zero minutes indicates that the operating system has an external method of notification (the ENF timer), and is not using an internal timer within WebSphere Message Broker.

-1

(Optional) The registry pass, which creates only the broker registry.

You must run the command with option **1** before you run the command with option **2**, otherwise the command fails.

-2

(Optional) The WebSphere MQ pass, which creates only the broker WebSphere MQ queues.

-y *LdapPrincipal*

(Optional, but mandatory when *ldapCredentials* is provided.) The user principal for access to an optional LDAP directory that holds the JNDI administered Initial Context for the JMS provider.

-z *LdapCredentials*

(Optional, but mandatory when *ldapPrincipal* is provided.) The user password for access to LDAP.

-c *ICUConverterPath*

(Optional) A delimited set of directories to search for additional code page converters.

The code page converters must be either of the form *codepagename*`.cnv`, or in an ICU data package called `icudt38.dat`. The code page converters must be located in a sub-directory named `icudt38_`*<platform_suffix>* of the specified directory where <*platform_suffix*> is one of the following values:

- l for little-endian ASCII platforms
- b for big-endian ASCII platforms
- e for EBCDIC platforms

Do not use this parameter to set the converter path if you are using a converter that matches one of the built-in converters that are provided with Version 6.0, and that converter is the local code page for the broker. Use the **ICU_DATA** environment variable instead.

-x *UserExitPath*

(Optional) The path that contains the location of all user exits to be loaded for execution groups in this broker. This path is added to the system library search path (`PATH,LIBPATH,LD_LIBRARY_PATH,SHLIBPATH`) for the execution group process only.

-s (Optional) Specify the administrative security status for the broker.

If you specify **-s active**, administration security is enabled. Only user IDs that you authorize are permitted to complete actions on the broker. Read, write, and execute authority is always granted on the security queue SYSTEM.BROKER.AUTH to all user IDs that belong to the security group mqbrkrs. When the broker has been created, you can add further user ID authorizations to this queue.

When you create an execution group on a broker for which administrative security is enabled, the queue SYSTEM.BROKER,AUTH.*egroup_name* is created. Populate the queue with the appropriate user authorization.

If you specify **-s inactive**, or omit this parameter, broker administration security is not enabled. All users are able to complete all actions against the broker and all execution groups.

For further information about using security, see “Broker administration security overview” on page 362 and “Authorizing users for broker administration” on page 371.

Examples:
Create a broker on z/OS by using a single command:

```
mqsicreatebroker CSQ1BRK -q CSQ1
```

Create a broker with administrative security enabled:

```
mqsicreatebroker MB7BROKER -q MB7QMGR -s active
```

Related tasks:

“Setting configuration timeout values” on page 3258
Change timeout values that affect configuration tasks in the broker.

Related reference:

“Syntax diagrams” on page 3677

“**mqsichangebroker** command” on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

“**mqsicreatebroker** command” on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

“**mqsideletebroker** command” on page 3863
Use the **mqsideletebroker** command to delete a named broker. The command also deletes the queues on the associated queue manager (created when the broker was created). You can also specify that the queue manager is to be deleted.

mqsicreateconfigurableservice command:

Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

You can also use the WebSphere Message Broker Explorer to create configurable services. For more information, see “Using the WebSphere Message Broker Explorer to work with configurable services” on page 644.

Supported platforms:

- Windows.
- Linux and UNIX systems.
- z/OS. Run this command by customizing and submitting BIPCRCS. For more information, see “Contents of the broker PDSE” on page 3991.

Purpose:
For configurable services that you add by using the **mqsicreateconfigurableservice** command:

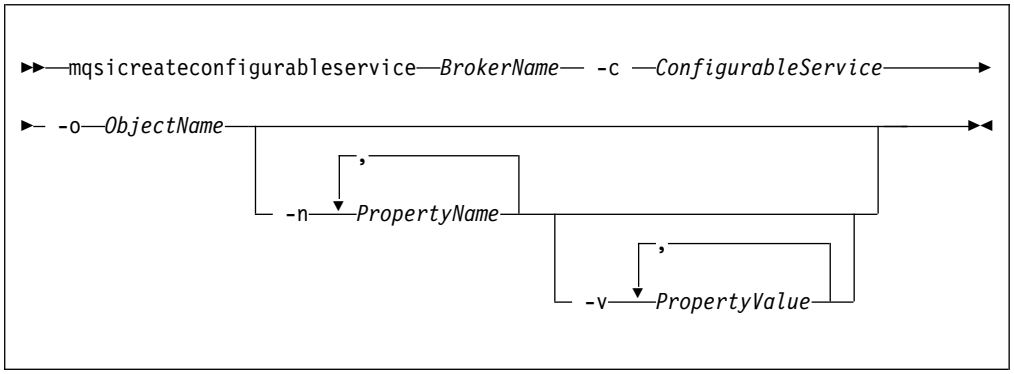
- Use the **mqsireportproperties** command to view the configurable services.
- Use the **mqsichangeproperties** command to modify the configurable services.
- Use the **mqsideleteconfigurableservice** command to delete configurable services.

You do not have to use the **mqsicreateconfigurableservice** command to create EIS providers, because definitions are supplied for you. Use the **mqsichangeproperties** command to modify EIS providers.

Usage notes:

- Before you run this command, ensure that the broker is running.
- Stop and restart the broker before you use any new broker resources and properties.

Syntax:



Parameters:

BrokerName
(Required) The name of the broker to modify. This parameter must be the first parameter.

-c *ConfigurableService*
(Required) The type of external resource (configurable service). Use the **mqsireportproperties** command to view the list of valid values.

The valid resource types are listed in "Configurable services properties" on page 3766.

-o *ObjectName*
(Required) The name of the object whose properties you want to change.

For example, if the **-c** parameter is set to JDBCProviders, the expected object name is either an IBM-defined JDBC provider name, or a user-defined JDBC provider name. Default services are provided for the supported databases to which you can connect over JDBC type 4 connections. Use the supplied services as a template when you create a service by using this command. Use the **mqsireportproperties** command to view the list of default provider names.

-n *PropertyName*
(Optional) The name of the property that is being changed.

The valid property names are listed in "Configurable services properties" on page 3766.

-v *PropertyValue*
(Optional, but required if the **-n** parameter is specified) The value that is assigned to the property that is specified by the **-n** parameter. You can specify more than one property name and corresponding value by using commas as separators; for example, `-n Name1,Name2 -v Value1,Value2`.

If the property value contains a comma, enclose the value with escaped double quotation marks (\" and \"); for example, `-n Name1,Name2 -v Value1,\"Value21,Value22\"`.

UNIX On UNIX systems, if the **-v** parameter contains a semi-colon (;), enclose the entire string in quotation marks, as shown in the following example:

```
mqsicreateconfigurableservice MB7BROKER -c JDBCProviders -o DB2EXTRA -n connectionUrlFormat
-v "jdbc:db2://[serverName]:[portNumber]/[databaseName]:user=[user];password=[password];"
```

The property values are described in "Configurable services properties" on page 3766.

Authorization:
For information about platform-specific authorizations, see the following topics:

- "Security requirements for Linux and UNIX platforms" on page 3648
- "Security requirements for Windows systems" on page 3651
- "Security requirements for z/OS" on page 3655

If you have enabled broker administration security, you must also set up the authority detailed in "Tasks and authorizations for broker administration security" on page 3645.

Responses:
This command returns the following responses:

- BIP8011 Unable to create configuration data
- BIP8012 Unable to connect to system components
- BIP8014 Component cannot be created
- BIP8073 Invalid broker name
- BIP8983 Configurable service already exists
- BIP8984 Configurable service was not found

Examples:
Create an Aggregation configurable service that uses a set of queues that is prefixed with SYSTEM.BROKER.AGGR.SET1, and with a timeout of 60 seconds:

```
mqsicreateconfigurableservice MB7BROKER -c Aggregation -o myAggregationService
-n queuePrefix,timeoutSeconds -v SET1,60
```

Create a CICSConnection configurable service for the CICS instance that is running at `tcp://test.cics.ibm.com` port 12345. The broker is identified by APPLID *BRKApp* and qualifier *BRKQual*. The connection timeout is 10 seconds and the request timeout is 5 seconds in this example:

```
mqsicreateconfigurableservice MB7BROKER -c CICSConnection -o myCICSConnectionService
-n cicsServer,clientApplid,clientQualifier,connectionTimeoutSecs,
requestTimeoutSecs -v tcp://test.cics.ibm.com:12345,BRKApp,BRKQual,10,5
```

Create a Collector configurable service that uses queues that are prefixed with SYSTEM.BROKER.EDA.SET1, and with a collection expiry of 60 seconds:

```
mqsicreateconfigurableservice MB7BROKER -c Collector -o myCollectorService
-n queuePrefix,collectionExpirySeconds -v SET1,60
```

Create a Connect:Direct server configurable service on object `test`, with a host name of `9.122.17.58` and port number `1369`:

```
mqsicreateconfigurableservice MB7BROKER -c CDServer -o test
-n Hostname,Port -v 9.122.17.58,1369
```

Create a CORBA configurable service that specifies the location of the object reference:

```
mqsicreateconfigurableservice MB7BROKER -c CORBA -o myCORBAService
-n namingService,objectReferenceName -v localhost:2809,Europe.region/Market.object
```

Create an EmailServer configurable service to which the EmailInput node or message flow can refer at run time to connect to an email server. The server is running at pop3://test.email.server.ibm.com on port 12345. In this example, the security identity is identified by *mySecurityIdentityObjectName*.

```
mqsicreateconfigurableservice MB7BROKER -c EmailServer -o myEmailConfigurableServiceName
-n serverName,securityIdentity -v pop3://test.email.server.ibm.com
:12345,mySecurityIdentityObjectName
```

Create an FtpServer configurable service:

```
mqsicreateconfigurableservice MB7BROKER -c FtpServer -o Server01
-n serverName,scanDelay,transferMode,connectionType,securityIdentity
-v one.hursley.abc.com:123,20,Binary,ACTIVE,secId
```

Create an FtpServer configurable service to use SFTP without strict host key checking:

```
mqsicreateconfigurableservice MB7BROKER -c FtpServer -o TEST1
-n protocol,serverName,scanDelay,remoteDirectory,securityIdentity,cipher,compression,strictHostKeyChecking
-v  SFTP,winlnx58,30,.,chbatey,blowfish-cbc,9,no
```

Create an IMSConnect configurable service for the IMS instance IMSA that is running on test.ims.ibm.com port 9999:

```
mqsicreateconfigurableservice MB7BROKER -c IMSConnect -o myIMSConnectService
-n Hostname,PortNumber,DataStoreName -v test.ims.ibm.com,9999,IMSA
```

Create a JavaClassLoader service:

```
mqsicreateconfigurableservice MB7BROKER -c JavaClassLoader
-o myJavaClassLoader
-n includedDeployedJars,sharedJarPath
-v "jcnapp1.jar","/var/app1/jars"
```

Add a JMS provider that is called "MyProviderXYZ":

```
mqsicreateconfigurableservice MB7BROKER -c JMSProviders -o JMS_MyProviderXYZ
```

Add a JMS provider that is called "ProviderABC", with default values for the resource properties:

```
mqsicreateconfigurableservice MB7BROKER -c JMSProviders -o JMS_ProviderABC
```

Add a JMS provider that is called "BEAV91", specifying the name of an IBM supplied Java class that is called com.ibm.broker.apihandler.BEAWebLogicAPIHandler to handle vendor-specific API calls:

```
mqsicreateconfigurableservice MB7BROKER -c JMSProviders -o BEAV91
-n proprietaryAPIHandler,proprietaryAPIAttr1,proprietaryAPIAttr2,proprietaryAPIAttr3
–v com.ibm.broker.apihandler.BEAWebLogicAPIHandler,weblogic.jndi.WLInitialContextFactory,
t3://19.21.194.126:7001,BEAServerName
```

Create a monitoring profile with the name 'mp1' to broker MB7BROKER:

```
mqsicreateconfigurableservice MB7BROKER -c MonitoringProfiles -o mp1
```

Create a PeopleSoftConnection configurable service for the PeopleSoft instance that is running on *my.peoplesoft.qa.com*:

```
mqsicreateconfigurableservice MB7BROKER -c PeopleSoftConnection
-o myPeopleSoftAdapter.outadapter -n hostName,port -v "my.peoplesoft.qa.com",9000
```

Create a Resequence configurable service that uses a set of queues that are prefixed with SYSTEM.BROKER.EDA.SET1, and with a missing message timeout of 60 seconds:

```
mqsicreateconfigurableservice MB7BROKER -c Resequence -o myResequenceService
-n queuePrefix,missingMessageTimeoutSeconds -v SET1,60
```

Create a SAPConnection configurable service for the SAP adapter *mySAPAdapter.outadapter* that connects to the SAP host `test.sap.ibm.com`, and uses client *001* for the connections into that server:

```
mqsicreateconfigurableservice MB7BROKER -c SAPConnection -o mySAPAdapter.outadapter
-n applicationServerHost,client -v test.sap.ibm.com,001
```

You can set the user name and password for an SAP adapter by using the **mqsisetdbparms** command.

Create a security profile for a WS-Trust V1.3 STS provider by using Tivoli Federated Identity Manager (TFIM) V6.2:

```
mqsicreateconfigurableservice MB7BROKER -c SecurityProfiles -o myWSTrustTFIMv62Profile
-n authentication,mapping,authorization,propagation,mappingConfig
-v "WS-Trust v1.3 STS","WS-Trust v1.3 STS","WS-Trust v1.3 STS",TRUE,
http://wstrusthost1.ibm.com:9080/TrustServerWST13/services/RequestSecurityToken
```

The Default Propagation profile is a predefined profile that requests only identity propagation. For more information about creating a security profile for a WS-Trust V1.3 STS provider, LDAP, or TFIM V6.1, see "Creating a security profile" on page 433.

Create a SiebelConnection configurable service for the Siebel adapter *mySiebelAdapter.outadapter* that connects to the Siebel server *"siebel://my.siebel.qa.com/SBA_80/SSEObjMgr_enu"*.

```
mqsicreateconfigurableservice MB7BROKER -c SiebelConnection -o mySiebelAdapter.outadapter
-n connectString -v "siebel://my.siebel.qa.com/SBA_80/SSEObjMgr_enu"
```

You can set the user name and password for a Siebel adapter by using the **mqsisetdbparms** command.

Create a TCPIPServer configurable service:

```
mqsicreateconfigurableservice MB7BROKER -c TCPIPServer -o ServerPort1452
-n Port,MaximumConnections,ExpireConnectionSec -v 1452,1000,15
```

Create a TCPIPClient configurable service:

```
mqsicreateconfigurableservice MB7BROKER -c TCPIPClient
-o ClientPort1452HostnameJsmith
-n Port,Hostname,AlternativeAddresses,MinimumConnections,MaximumConnections
-v 1452,jsmith.hursley.ibm.com,jones:1111;edwards,5,10
```

Create a TCP/IP configurable service that uses SSL:

"Configuring TCP/IP client nodes to use SSL" on page 551

"Configuring TCP/IP server nodes to use SSL" on page 553

Create a Timer configurable service that uses a queue that is prefixed with SYSTEM.BROKER.TIMEOUT.SET1, and has a timeout of 5 seconds:

```
mqsicreateconfigurableservice MB7BROKER -c Timer -o myTimerService
-n queuePrefix,timeoutIntervalSeconds -v SET1,5
```

Create a UserDefined service:

```
mqsicreateconfigurableservice MB7BROKER -c UserDefined
-o HTTP_Timeout -n VerifyRequestTimeout -v 60
```

Related concepts:

"Using the WebSphere Message Broker Explorer to work with configurable services" on page 644
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view, add, modify and delete configurable services.

Related tasks:

"Setting up a JDBC provider for type 4 connections" on page 684
Use the **mqsicreateconfigurableservice** or the **mqsichangeproperties** command to configure a JDBC provider service.

"Configuring the broker to enable a JMS provider's proprietary API" on page 748
Some JMS providers provide an alternative interface to the standard JMS specification for particular JMS API calls. In these cases, IBM supplies a Java class to interface with that proprietary API.

Related reference:

"Contents of the broker PDSE" on page 3991
After you have successfully customized the broker, the broker PDSE members have been set up.

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

"FtpServer configurable service properties" on page 3794
Select the properties and values that you want to change for an existing FtpServer configurable service or to create a new service.

mqsicreateexecutiongroup command:

Use the **mqsicreateexecutiongroup** command to add a new execution group to a broker.

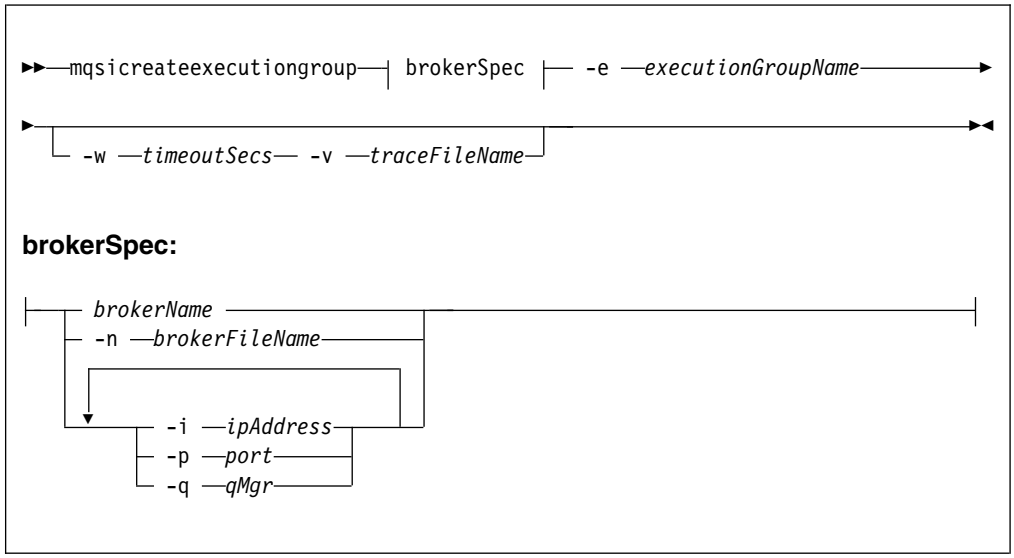
Supported platforms:

- Windows
- Linux and UNIX systems

- z/OS. Run this command by customizing and submitting BIPCREG; see "Contents of the broker PDSE" on page 3991

Purpose:
You must start the broker before you run the **mqsicreateexecutiongroup** command.

Syntax:



Parameters:

brokerSpec
(Required) You must specify at least one parameter to identify the target broker for this command, in one of the following forms:

brokerName
This parameter identifies the name of a locally defined broker. You cannot use this option if the broker is on a remote computer.

-n *brokerFileName*
This parameter identifies the name of a file that contains the connection details for a local or remote broker.

Use this option if multiple users want to connect to the same broker, or if you are using advanced connection parameters such as SSL.

To create this file, right-click the broker in the WebSphere Message Broker Explorer and select **Export *.broker file**. When prompted, navigate to the location in which you want to save the file and enter the file name; the extension `.broker` is appended automatically when you click **Save**. Include the location (path) and file name when you specify this parameter. You must ensure that the location is accessible when you run this command.

If you want to run a command that uses SSL to administer a remote broker over a secured channel, you must specify the keystore and truststore password for the connection using the IBM_JAVA_OPTIONS environment variable. See "Resolving problems when running commands" on page 3364 for further information.

-i *ipAddress*, **-p** *port*, **-q** *qMgr*
These parameters identify the connection details for the broker.

Use this option for connections to remote brokers that do not require advanced connection parameters.

If you choose this option, you must specify at least one of these three parameters; the order in which you specify them is insignificant. You cannot specify each parameter more than once.

Parameters that you omit assume default values:

- **-i** *ipAddress*: The host name or the IP address of the computer on which the broker is running. If you do not specify this parameter, a value that represents the local computer is used.
- **-p** *port*: The TCP port on which the broker's queue manager is listening. If you do not specify this parameter, the value 1414 is used.
- **-q** *qMgr*: The name of the broker's queue manager. If you do not specify this parameter, a value that represents the default queue manager on the local computer is used.

You cannot connect to a remote queue manager on z/OS; see "Usage note."

-e *executionGroupName*
(Required) The name of the execution group to create.

-w *timeoutSecs*
(Optional) This parameter specifies the time in seconds that the utility waits to ensure that the command completed; the default value is 60.

-v *traceFileName*
(Optional) This parameter sends internal debug trace information to the specified file.

Usage note:
If you try to connect to a remote broker by specifying the **i**, **p**, and **q** parameters, or by using a connection parameter file (`.broker`), the command attempts to use WebSphere MQ Java client code. This option is not supported on z/OS, and returns the following error:

```
BIP1046E: Unable to connect with the broker (name)
```

The reported reason code is `MQRC_ENVIRONMENT_ERROR`. You must specify a local queue manager.

Authorization:
For information about platform-specific authorizations, see the following topics:

- "Security requirements for Linux and UNIX platforms" on page 3648
- "Security requirements for Windows systems" on page 3651
- "Security requirements for z/OS" on page 3655

If you have enabled broker administration security, you must also set up the authority detailed in "Tasks and authorizations for broker administration security" on page 3645.

Responses:
This command returns the following responses:

0 The command completed successfully.

2 (Failure) The broker received the deployment request but was unable to

process it successfully. See the messages issued from the utility (or the Administration log) for more information.

9 (Failure) The request has been submitted to the broker, but no response was received before the timeout expired.

10 (Failure) Another user or application canceled the request operation before the broker was able to process it.

98 The broker is not running.

99 One or more of the parameters that you specified is invalid.

Examples:
Create an execution group called `EG1` on the broker that is hosted by the queue manager `QMGR`, which is listening on `fred.abc.com:1414`.

```
mqsicreateexecutiongroup -i fred.abc.com -p 1414 -q QMGR –e EG1
```

Create an execution group called `EG2` on the broker that is defined by the connection parameters in file `BROKER.broker`

```
mqsicreateexecutiongroup –n BROKER.broker –e EG2
```

Create an execution group `EG3` on the broker that is defined by the connection parameters in file `FRED.broker`. Wait 5 minutes for the broker to respond, and send output to `trace.txt`.

```
mqsicreateexecutiongroup –n FRED.broker –e EG3 –w 300 –v trace.txt
```

Related tasks:

"Enabling broker administration security" on page 368
Enable broker administration security on a broker to control which users can complete specific tasks against that broker and its resources.

Related reference:

"**mqsideleteexecutiongroup** command" on page 3869
Use the **mqsideleteexecutiongroup** command to remove an execution group from a broker.

"**mqsistart** command" on page 3965
Use the **mqsistart** command to start the specified broker if all initial verification tests complete successfully.

"Syntax diagrams" on page 3677

mqsicvp command:

Use the **mqsicvp** command to perform verification tests on a broker, or to verify ODBC connections.

Supported platforms:

- Windows
- Linux and UNIX systems

When you start a broker by using the **mqsistart** command, this command is run automatically to verify the component.

On z/OS, the same verification procedures are run automatically when you start a broker.

You can run this command against a broker that is running, or is not running. If the broker is not running, the verification tests are performed, but the broker is not started.

Purpose:
The **mqsicvp** command completes the following actions:

- The command checks that the broker environment is set up correctly; for example, that the installed level of Java is supported.
- The command verifies that the WebSphere MQ queues are defined and accessible.
- Linux UNIX On Linux and UNIX systems, the command verifies that the ODBC environment (if specified) is configured correctly.

  If the ODBCINI environment variable is set, the **mqsicvp** command writes warning messages to the syslog in the following situation:
  - If the file to which the ODBCINI environment variable points does not exist, or the broker does not have access to read it or write to it

  If the environment variable ODBCUOINI is set, the **mqsicvp** command writes warning messages to the syslog in the following situations:
  - If the file that is referenced by the ODBCUOINI environment variable does not exist, or the broker does not have access to read or write to the file
  - If ODBCSYSINI is not set
  - If the directory that is referenced by the ODBCSYSINI environment variable does not contain a file called `odbcinst.ini`, or the broker does not have access to read or write to this file
  - If the IE02_PATH environment variable is not set
- Linux UNIX On Linux and UNIX systems, if the ODBC environment check is successful, the command connects to all data sources that are listed in the `odbc.ini` files, and that were associated with the broker by using the **mqsisetdbparms** command.

Use of the command to provide information about user data sources:
Extra invocations of the **mqsicvp** command have been added to provide ODBC Test Tool function. This function provides useful information and diagnostics about a user data source, or Compares two user data sources for equivalence. These invocations are discrete from the existing invocation of the **mqsicvp** command, and are not run at broker startup.

When you use the **mqsicvp** command as an ODBC Test Tool, the command issues an informational message on successful connection, providing the name of the datasource, database type, and version. If a secondary datasource is supplied, the **mqsicvp** command issues a second informational message on successful connection to that datasource, with the same information concerning the secondary datasource, and informing you that a comparison will be made.

When the tool is run against one data source, it completes several checks against the ODBC interface to determine which data types and functions are supported, together with information on the names and sizes of those data types. If any data types or functions are not supported, they are summarized in a final informational message.

When these tests are run against two data sources, they are run against both data sources, and both results are shown. A final informational message states whether

the two data sources are equivalent and eligible to be used within the same message processing node; for more information, see “Accessing databases from ESQL” on page 2115.

Syntax:

Invocation to verify the broker

When you run the **mqsicvp** command with only the broker name, and no other parameters, the command completes the following checks:

- Checks that the broker environment is set up correctly (for example, the installed level of Java is supported).
- Verifies that the WebSphere MQ queues are defined and accessible.
- Linux UNIX On Linux and UNIX systems only, the command verifies that the ODBC environment (if specified) is configured correctly. If the ODBCINI environment variable is not set, ODBC environment verification is skipped. If the ODBC environment check is completed successfully, the command then tries to connect to all data sources that are listed in the `odbc.ini` files where the **mqsisetdbparms** command was used to associate the data source with the broker.

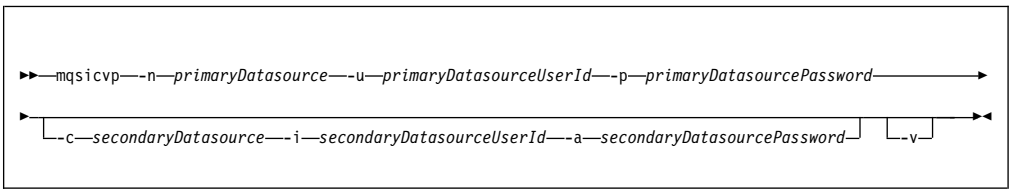
Syntax:

Invocation where data source names have not been associated with the broker

When you run the **mqsicvp** command with parameters shown in the following syntax diagram, the command provides ODBC test tool function. You can use the command to display useful information about a user data source, or compare two user data sources. On Linux and UNIX systems only, invocations of the command that use these parameters also verify that the ODBC environment (if specified) is configured correctly. In these cases, the command checks that the ODBCINI, ODBCSYSINI, ODBCUOINI, and IE02_PATH variables are set.

Each data source name (DSN) is fully specified with a user name and password.

This invocation does not rely on an association between the broker and the data source, and returns information about the data source even when there is no association with a broker.



Syntax:

Invocation where data source names were associated with the broker

This invocation requires an association between the broker and the data source name (DSN). In order to successfully use this invocation, you must first run the **mqsisetdbparms** command to identify a specific user ID and password for the broker to use when connecting to the data source. The output from this invocation is the same as the output from the invocation where data source names have not been associated with the broker.

When using the **mqsisetdbparms** command, you can either specify the data source name directly, or add one of the following prefixes:

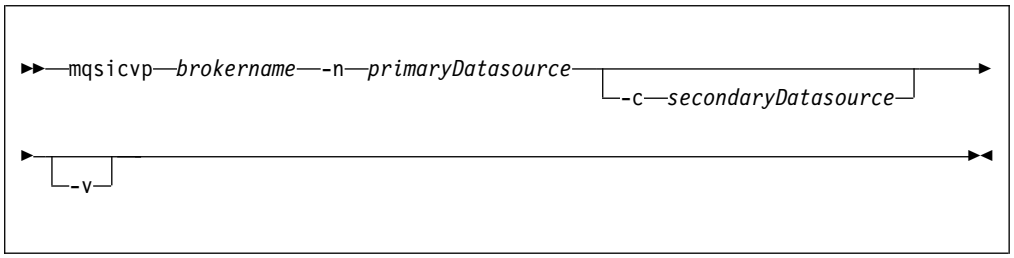
- `dsn::`
- `odbc::`

These prefixes identify the type of identity being created in **mqsisetdbparms**, and must be omitted when using this identity as the data source in **mqsicvp**. For example:

- `mqsisetdbparms MB7BROKER -n dsn::myDsn -u username -p password`
- `mqsicvp MB7BROKER -n myDsn`

If you use **mqsisetdbparms** to set an execution group level identity for a data source, the identity cannot be used in **mqsicvp**. Additionally, **mqsicvp** cannot use a default user ID and password that has been set for all data sources.

Linux UNIX On Linux and UNIX systems only, invocations of the command that use the parameters in the following syntax diagram also verify that the ODBC environment (if specified) is configured correctly. In these cases, the command checks that the ODBCINI, ODBCSYSINI, ODBCUOINI, and IE02_PATH variables are set.



Parameters:

brokername
(Required - if you are using an invocation containing *brokername*) Specify a broker name to verify, or the broker name with which the `primaryDatasource` is associated.

Linux UNIX All names are case sensitive on Linux and UNIX systems.

-n *primaryDatasource*
(Required if you are using an invocation containing *primaryDatasource*) Name of the ODBC connection to verify.

-u *primaryDatasourceUserId*
(Required if you have not previously associated the data source name with the broker) User name with which to connect to the *primaryDatasource*.

-p *primaryDatasourcePassword*
(Required if you have not previously associated the data source name with the broker) Password used with the *primaryDatasourceUserId*.

-c *secondaryDatasource*
(Optional) If two data sources are being compared for equivalence, this is the second ODBC connection name.

-i *secondaryDatasourceUserId*
(Optional) User name with which to connect to the *secondaryDatasource*.

-a *secondaryDatasourcePassword*
(Optional) Password used with the *secondaryDatasourceUserId*.

-v (Optional) Causes extra, untranslated, diagnostics related to supported CASTS to be output by the command.

Authorization:
For information about authorizations that are specific to operating systems, see the following topics:

- Linux UNIX "Security requirements for Linux and UNIX platforms" on page 3648
- Windows "Security requirements for Windows systems" on page 3651

If you enabled broker administration security, you must also set up the authority that is described in "Tasks and authorizations for broker administration security" on page 3645.

Responses:
- BIP8040W: No database access (unable to connect).
- BIP8267W: Warning, there might be issues using this data source. See the preceding messages for more information.
- BIP8268I: The two data sources supplied are compatible, and can be used in the same Compute node.
- BIP8269W: The two data sources supplied are not compatible, and must not be used in the same Compute node.
- BIP8270I: Connected to datasource <..multiple inserts..>
- BIP8271I: Connected to second datasource <..multiple inserts..> for comparison.
- BIP8272W: Datasource specified has not been associated with the broker.
- BIP8273I: The following datatypes and functions are not natively supported by datasource '&1': <..multiple inserts..>
- BIP8274W: The following datatypes and functions might cause problems when using datasource '&1' with WebSphere Message Broker: <..multiple inserts..>
- BIP8288W: Unable to read ODBCINI file '*file_name*'. Check that this file exists, and that the broker user ID has permission to read and write the file.
- BIP8289W: Unable to read ODBCSYSINI file in specified directory '*directory_name*'. Check that this file exists, and that the broker user ID has permission to read and write the file.
- BIP8290I: Verification passed for the ODBC environment.
- BIP8291W: The IE02_PATH environment variable is not set.
- BIP8292I: '*insert1*' user data sources were not verified because they do not have **mqsisetdbparms** credentials.
- BIP8293W: Unable to read ODBCUOINI file '*file_name*'. Check that this file exists, and that the broker user ID has permission to read and write to the file.

- BIP8294I: ODBC environment verification was skipped because both the ODBCINI and ODBCUOINI environment variables are not set.
- BIP8295E: ODBCINI environment variables have not been set, or are invalid.
- BIP8296W: The ODBCSYSINI environment variable is not set.
- BIP8297W: '*environment_variable_name*' file '*file_name*' is empty.
- BIP8873I: Starting the component verification for broker '*broker_name*'.
- BIP8874I: The component verification for '*broker_name*' has finished successfully.
- BIP8875W: The component verification for '*broker_name*' has finished, but one or more checks failed.
- BIP8876I: Starting the environment verification for broker '*broker_name*'.
- BIP8877W: The environment verification for broker '*broker_name*' has finished, but one or more checks failed.
- BIP8878I: The environment verification for broker '*broker_name*' has finished successfully.
- BIP8882I: Starting the WebSphere MQ verification for broker '*broker_name*'.
- BIP8883W: The WebSphere MQ verification for broker '*broker_name*' has finished, but one or more checks failed.
- BIP8884I: The WebSphere MQ verification for broker '*broker_name*' has finished successfully.
- BIP8885E: Verification failed. Failed to connect to queue manager '*queue_manager_name*'. MQRC: *return_code* MQCC: *completion_code*
- BIP8886I: Verification passed for queue '*queue_name*' on queue manager '*queue_manager_name*'.
- BIP8887E: Verification failed for queue '*queue_name*' on queue manager '*queue_manager_name*' while issuing '*operation*'. MQRC: *return_code* MQCC: *completion_code*
- BIP8888E: Verification failed. Failed to disconnect from queue manager '*queue_manager_name*'. MQRC: *return_code* MQCC: *completion_code*
- BIP8892E: Verification failed. The installed Java level '*level_installed*' does not meet the required Java level '*level_supported*'.
- BIP8893E: Verification failed for environment variable '*variable_name*'. Unable to access file '*file_name*' with user ID '*user_ID*'. Additional information for IBM support: *data1 data2*.
- BIP8894I: Verification passed for '*broker_name*'.
- BIP8895E: Verification failed. Environment variable '*variable_name*' is incorrect or missing.
- BIP8896E: Verification failed. Unable to access the registry with user ID '*user_ID*'. Additional information for IBM support: *data1 data2*
- BIP8897E: Verification failed. Environment variable '*variable_name*' does not match the broker name '*broker_name*'.
- BIP8900I: APF Authorization check successful for file '*file_name*'.
- BIP8903E: Verification failed. The APF Authorization check failed for file '*file_name*'.
- BIP8904E: Verification failed. Failed to stat file '*file_name*1' with return code '*return_code*' and errno '*error_number*'.

Examples:
Run verification checks on the broker named MB7BROKER:

```
mqsicvp MB7BROKER
```

Data Source Name (DSN) MyDB is associated with the broker MB7BROKER using the **mqsisetdbparms** command:

```
mqsicvp MB7BROKER -n MyDB
```

DSNMyDB is compared against a secondary DSN MyDB2 associated with the broker MB7BROKER using the **mqsisetdbparms** command:

```
mqsicvp MB7BROKER -n MyDB -c MyDB2
```

The fully qualified DSN MyDB is compared against a secondary fully qualified DSN MyDB2 using the primary and secondary user IDs and passwords:

```
mqsicvp –n MyDB –u username –p password –c MyDB2 –i username2 –a password2
```

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related tasks:

"Accessing databases from ESQL" on page 2115
Configure your broker and your database to support connections from message flows.

Related reference:

"Syntax diagrams" on page 3677

"**mqsistart** command" on page 3965
Use the **mqsistart** command to start the specified broker if all initial verification tests complete successfully.

"**mqsistop** command" on page 3972
Use the **mqsistop** command to stop the specified component.

mqsideletebroker command:

Use the **mqsideletebroker** command to delete a named broker. The command also deletes the queues on the associated queue manager (created when the broker was created). You can also specify that the queue manager is to be deleted.

Supported platforms:

- Windows
- Linux and UNIX systems
- z/OS. Run this command by customizing and submitting BIPDLBK; see "Contents of the broker PDSE" on page 3991

Purpose:
The **mqsideletebroker** command:

- On Windows platforms, stops the service that runs the broker.
- Stops and deletes the WebSphere MQ queue manager for the broker, if requested.
- Deletes the administrative security queues that are associated with the broker, if requested.
- Removes the record for the component in the registry.

If you delete a broker that has WebSphere MQ publish/subscribe broker neighbors, you must also run the **clrmqbrk** on each of these neighbors. You must identify the broker that you are deleting on this command.

Select the appropriate link for details of this command on the platform, or platforms, that your enterprise uses:

- "**mqsideletebroker** command - Windows, Linux, and UNIX systems"
- "**mqsideletebroker** command - z/OS" on page 3865

Usage notes:
If you run the command against a component that does not exist (for example, the component has already been deleted, or you have mistyped the component name), the command returns with a successful completion message. The command does not inform you that the component does not exist.

You cannot delete a broker to which you have an open connection; the command fails. For example, if your WebSphere Message Broker Explorer session is currently connected to a broker, you cannot delete that broker until you disconnect it.

Authorization:
For information about platform-specific authorizations, see the following topics:

- "Security requirements for Linux and UNIX platforms" on page 3648
- "Security requirements for Windows systems" on page 3651
- "Security requirements for z/OS" on page 3655

If you have enabled broker administration security, you must also set up the authority detailed in "Tasks and authorizations for broker administration security" on page 3645.

Related tasks:

"Deleting a broker" on page 930
Delete a broker using the command line on the system where the broker component is installed.

Related reference:

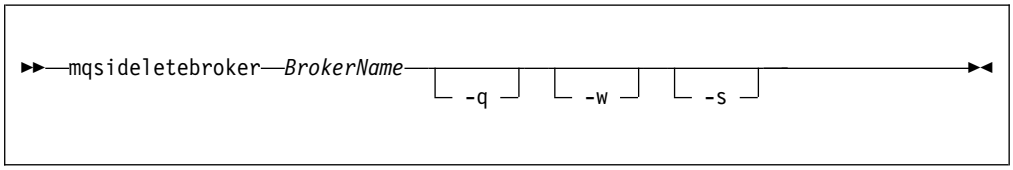
"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

mqsideletebroker *command - Windows, Linux, and UNIX systems:*

Use the **mqsideletebroker** command to delete a broker on Linux, UNIX, or Windows systems.

Syntax:



Parameters:

BrokerName
(Required) The name of the broker that you want to delete. You must specify this parameter first.

-q (Optional) Specifies that the broker's queue manager should be deleted. If you do not specify this parameter, only the WebSphere MQ queues and broker's data are deleted.

-w (Optional) Deletes all files related to this broker from the work path.

-s (Optional) Specify this option to delete all administrative security queues for this broker when the broker is deleted. The queue SYSTEM.BROKER.AUTH and the queue for every defined execution group (SYSTEM.BROKER.AUTH.*egroup_name*) are deleted.

If you do not specify this option, the security queues are retained and can be reused if you re-create the broker.

This parameter is ignored if you specify **-q**, which deletes the queue manager and all its queues.

Examples:
Delete the broker and its associated queue manager:

```
mqsideletebroker MB7BROKER -q
```

Delete the broker and all its security queues that are defined on the associated queue manager:

```
mqsideletebroker MB7BROKER -s
```

Related reference:

"Syntax diagrams" on page 3677

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

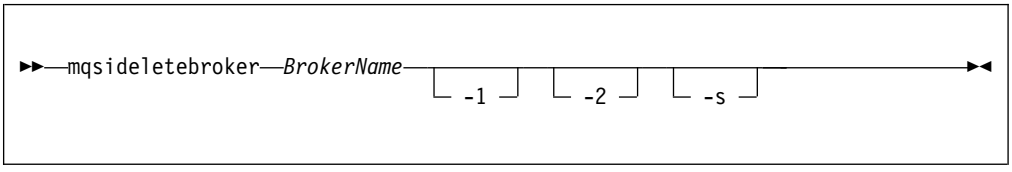
"**mqsideletebroker** command" on page 3863
Use the **mqsideletebroker** command to delete a named broker. The command also deletes the queues on the associated queue manager (created when the broker was created). You can also specify that the queue manager is to be deleted.

mqsideletebroker *command - z/OS:*

Use the **mqsideletebroker** command to delete a broker on a z/OS system.

Syntax:

z/OS command - BIPDLBK:



Parameters:

BrokerName
(Required) The name of the broker that you want to delete. You must specify this parameter first.

-1
(Optional) Deletes only the broker registry.

-2
(Optional) Deletes only the broker WebSphere MQ queues.

-s (Optional) Specify this option to delete all administrative security queues for this broker when the broker is deleted. The queue SYSTEM.BROKER.AUTH and the queue for every defined execution group (SYSTEM.BROKER.AUTH.*egroup_name*) are deleted.

If you do not specify this option, the security queues are retained and can be reused if you re-create the broker.

Examples:
Delete the broker registry for the broker `CSQ1BRK` on z/OS:

```
mqsideletebroker CSQ1BRK -1
```

Delete the broker `CSQ2BRK` and all its associated resources:

```
mqsideletebroker CSQ2BRK
```

Delete the broker and all its security queues that are defined on the associated queue manager:

```
mqsideletebroker CSQ1BRK -s
```

Related reference:

"Syntax diagrams" on page 3677

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

"**mqsideletebroker** command" on page 3863
Use the **mqsideletebroker** command to delete a named broker. The command also deletes the queues on the associated queue manager (created when the broker was created). You can also specify that the queue manager is to be deleted.

mqsideleteconfigurableservice command:

Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

You can also use the WebSphere Message Broker Explorer to delete custom-named configurable services; however, you cannot delete IBM-defined configurable services. See "Using the WebSphere Message Broker Explorer to work with configurable services" on page 644 for more information.

Supported platforms:
- Windows systems
- Linux and UNIX systems

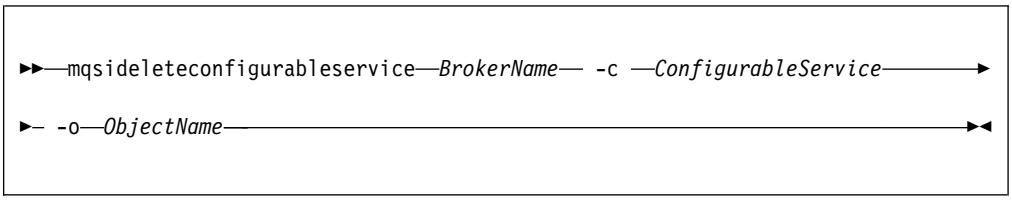
- z/OS. Run this command by customizing and submitting BIPDLCS; see "Contents of the broker PDSE" on page 3991

Purpose:
Use this command to delete a configurable service. Use the **mqsireportproperties** command to view the configurable services that are defined.

Usage notes:

- Before you run this command, ensure that the broker is running.
- After you have run this command, stop and restart the broker to ensure that deleted broker resources and properties are not being used.

Syntax:



Parameters:

BrokerName
(Required) The name of the broker to modify. This parameter must be the first parameter.

-c *ConfigurableService*
(Required) The type of configurable service. Use the **mqsireportproperties** command to view the list of all defined services.

For a list of supplied configurable services, and their properties and values, see "Configurable services properties" on page 3766.

-o *ObjectName*
(Required) The name of the object for which you want to delete the properties. Use the **mqsireportproperties** command to view the list of properties that you can delete.

Authorization:
For information about platform-specific authorizations, see the following topics:

- "Security requirements for Linux and UNIX platforms" on page 3648
- "Security requirements for Windows systems" on page 3651
- "Security requirements for z/OS" on page 3655

If you have enabled broker administration security, you must also set up the authority detailed in "Tasks and authorizations for broker administration security" on page 3645.

Responses:
This command returns the following responses

- BIP8012 Unable to connect to system components
- BIP8014 Component cannot be created
- BIP8073 Invalid broker name
- BIP8984 Configurable service was not found

Examples:
Delete a CICSConnection configurable service for broker MB7BROKER:

```
mqsideleteconfigurableservice MB7BROKER -c CICSConnection
-o myCICSConnectionService
```

Delete a CORBA configurable service for broker MB7BROKER:

```
mqsideleteconfigurableservice MB7BROKER -c CORBA -o myCORBAService
```

Delete an EmailServer configurable service for broker MB7BROKER called *myEmailConfigurableServiceName* that the EmailInput node or message flow is referring to at runtime:

```
mqsideleteconfigurableservice MB7BROKER -c EmailServer
-o myEmailConfigurableServiceName
```

Delete an FtpServer configurable service for broker MB7BROKER:

```
mqsideleteconfigurableservice MB7BROKER -c FtpServer -o Server01
```

Delete an IMS configurable service called *myIMSconnectService*:

```
mqsideleteconfigurableservice MB7BROKER -c IMSConnect
-o myIMSconnectService
```

Delete a JavaClassLoader configurable service:

```
mqsideleteconfigurableservice MB7BROKER -c JavaClassLoader -o myJavaClassLoader
```

Delete a JMS provider configurable service called *MyProviderXYZ*:

```
mqsideleteconfigurableservice MB7BROKER -c JMSProviders -o JMS_MyProviderXYZ
```

Delete a monitoring profile:

```
mqsideleteconfigurableservice MB7BROKER -c MonitoringProfiles -o myMonitoringProfile
```

Delete the PeopleSoftConnection configurable service that is associated with *myPeopleSoftAdapter.outadapter*:

```
mqsideleteconfigurableservice MB7BROKER -c PeopleSoftConnection
-o myPeopleSoftAdapter.outadapter
```

Delete a security profile for LDAP use:

```
mqsideleteconfigurableservice MB7BROKER -c SecurityProfiles -o MyLDAPProfile
```

Delete the SiebelConnection configurable service that is associated with *mySiebelAdapter.outadapter*:

```
mqsideleteconfigurableservice MB7BROKER -c SiebelConnection -o mySiebelAdapter.outadapter
```

Delete a TCPIPClient configurable service:

```
mqsideleteconfigurableservice MB7BROKER -c TCPIPClient
-o ClientPort1452HostnameJsmith
```

Delete a TCPIPServer configurable service:

```
mqsideleteconfigurableservice MB7BROKER -c TCPIPServer -o ServerPort1452
```

Related concepts:

"Using the WebSphere Message Broker Explorer to work with configurable services" on page 644
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view, add, modify and delete configurable services.

Related reference:

"Contents of the broker PDSE" on page 3991
After you have successfully customized the broker, the broker PDSE members have been set up.

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

mqsideleteexecutiongroup command:

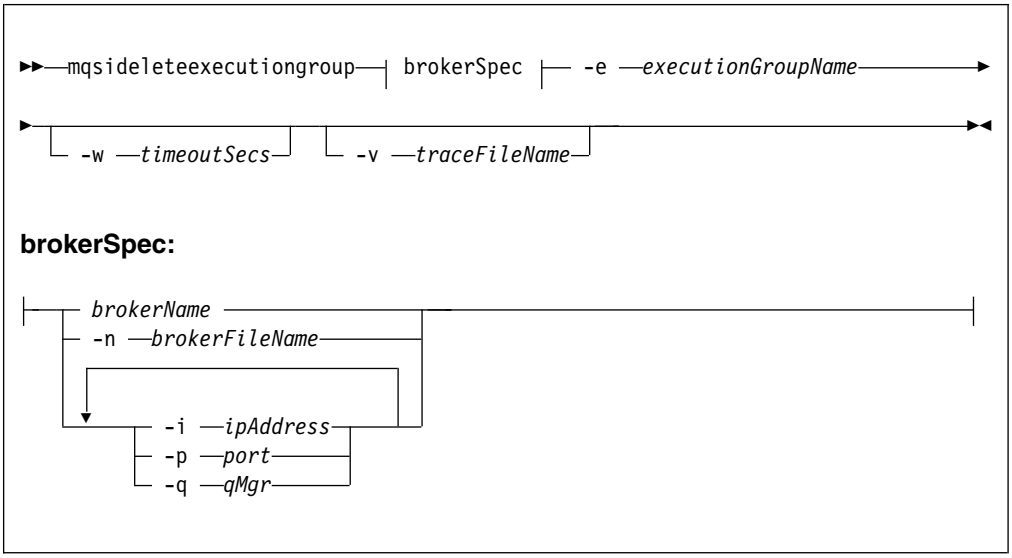
Use the **mqsideleteexecutiongroup** command to remove an execution group from a broker.

Supported platforms:

- Windows
- Linux and UNIX systems
- z/OS. Run this command by customizing and submitting BIPDLEG; see "Contents of the broker PDSE" on page 3991

Purpose:
You must start the broker before you can issue this command.

Syntax:



Parameters:

brokerSpec

(Required) You must specify at least one parameter to identify the target broker for this command, in one of the following forms:

brokerName

This parameter identifies the name of a locally defined broker. You cannot use this option if the broker is on a remote computer.

-n *brokerFileName*

This parameter identifies the name of a file that contains the connection details for a local or remote broker.

Use this option if multiple users want to connect to the same broker, or if you are using advanced connection parameters such as SSL.

To create this file, right-click the broker in the WebSphere Message Broker Explorer and select **Export *.broker file**. When prompted, navigate to the location in which you want to save the file and enter the file name; the extension `.broker` is appended automatically when you click **Save**. Include the location (path) and file name when you specify this parameter. You must ensure that the location is accessible when you run this command.

If you want to run a command that uses SSL to administer a remote broker over a secured channel, you must specify the keystore and truststore password for the connection using the IBM_JAVA_OPTIONS environment variable. See "Resolving problems when running commands" on page 3364 for further information.

-i *ipAddress*, **-p** *port*, **-q** *qMgr*

These parameters identify the connection details for the broker.

Use this option for connections to remote brokers that do not require advanced connection parameters.

If you choose this option, you must specify at least one of these three parameters; the order in which you specify them is insignificant. You cannot specify each parameter more than once.

Parameters that you omit assume default values:

- **-i** *ipAddress*: The host name or the IP address of the computer on which the broker is running. If you do not specify this parameter, a value that represents the local computer is used.
- **-p** *port*: The TCP port on which the broker's queue manager is listening. If you do not specify this parameter, the value 1414 is used.
- **-q** *qMgr*: The name of the broker's queue manager. If you do not specify this parameter, a value that represents the default queue manager on the local computer is used.

You cannot connect to a remote queue manager on z/OS; see "Usage note" on page 3871.

-e *executionGroupName*

(Required) The name of the execution group to delete.

-w *timeoutSecs*

(Optional) This parameter specifies the time in seconds that the utility waits to ensure that the command completed; the default value is 60.

-v *traceFileName*
(Optional) This parameter sends internal debug trace information to the specified file.

Usage note:
If you try to connect to a remote broker by specifying the **i**, **p**, and **q** parameters, or by using a connection parameter file (`.broker`), the command attempts to use WebSphere MQ Java client code. This option is not supported on z/OS, and returns the following error:

```
BIP1046E: Unable to connect with the broker (name)
```

The reported reason code is `MQRC_ENVIRONMENT_ERROR`. You must specify a local queue manager.

Authorization:
For information about platform-specific authorizations, see the following topics:

- "Security requirements for Linux and UNIX platforms" on page 3648
- "Security requirements for Windows systems" on page 3651
- "Security requirements for z/OS" on page 3655

If you have enabled broker administration security, you must also set up the authority detailed in "Tasks and authorizations for broker administration security" on page 3645.

Responses:
This command returns the following responses:

0 The command completed successfully.

2 (Failure) The broker received the deployment request but was unable to process it successfully. See the messages issued from the utility (or the Administration log) for more information.

9 (Failure) The request has been submitted to the broker, but no response was received before the timeout expired.

10 (Failure) Another user or application canceled the request operation before the broker was able to process it.

98 The broker is not running.

99 One or more of the parameters that you specified is invalid.

Examples:
Delete an execution group called `EG1` on the broker defined by connection parameters specified by the file `BKR1.broker`.

```
mqsideleteexecutiongroup –n BKR1.broker –e EG1
```

Delete an execution group `EG2` on the broker specified by the file `FRED.broker`. Wait 5 minutes for the broker to tidy up related resources, and send output to `trace.txt`.

```
mqsideleteexecutiongroup –n FRED.broker –e EG2 –w 300 –v trace.txt
```

Delete the execution group `EG3` on the locally defined broker **MYBROKER**.

```
mqsideleteexecutiongroup MYBROKER –e EG3
```

Related tasks:

"Enabling broker administration security" on page 368
Enable broker administration security on a broker to control which users can

complete specific tasks against that broker and its resources.

Related reference:

"**mqsicreateexecutiongroup** command" on page 3854
Use the **mqsicreateexecutiongroup** command to add a new execution group to a broker.

"**mqsistart** command" on page 3965
Use the **mqsistart** command to start the specified broker if all initial verification tests complete successfully.

"Syntax diagrams" on page 3677

mqsideploy command:

Use the **mqsideploy** command to make a deployment request to the broker.

Supported platforms:

- Windows
- Linux and UNIX systems
- z/OS. Run this command in one of two ways - as a console command, or by customizing and submitting BIPDPLY; see "Contents of the broker PDSE" on page 3991

Purpose:
Use the **mqsideploy** command to make deployment requests of all types from a batch command script, without the need for manual interaction.

Select the appropriate link for details of this command on the platform, or platforms, that your enterprise uses:

- "**mqsideploy** command - Windows, Linux, and UNIX systems" on page 3873
- "**mqsideploy** command - z/OS" on page 3876

Authorization:
For information about platform-specific authorizations, see the following topics:

- "Security requirements for Linux and UNIX platforms" on page 3648
- "Security requirements for Windows systems" on page 3651
- "Security requirements for z/OS" on page 3655

If you have enabled broker administration security, you must also set up the authority detailed in "Tasks and authorizations for broker administration security" on page 3645.

Responses:
This command returns the following responses:

0 The command completed successfully.

2 (Failure) The broker received the deployment request but was unable to process it successfully. See the messages issued from the utility (or the Administration log) for more information.

9 (Failure) The request has been submitted to the broker, but no response was received before the timeout expired.

10 (Failure) Another user or application canceled the request operation before the broker was able to process it.

98 The broker is not running.

99 One or more of the parameters that you specified is invalid.

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

"Packaging and deployment overview" on page 3210
Deployment is the process of transferring data to an execution group on a broker so that it can take effect in the broker. Message flows and associated resources are packaged in broker archive (BAR) files for deployment.

Related tasks:

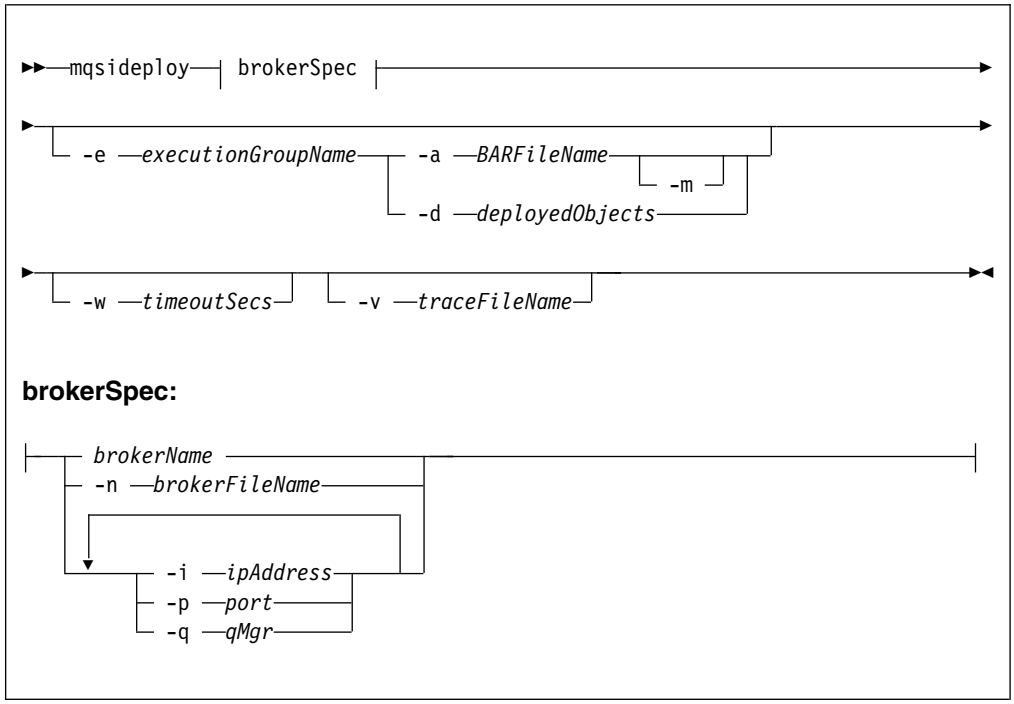
"Enabling broker administration security" on page 368
Enable broker administration security on a broker to control which users can complete specific tasks against that broker and its resources.

Chapter 11, "Packaging and deploying," on page 3209
Package resources that you create in the WebSphere Message Broker Toolkit, such as message flows, and deploy them to execution groups on brokers.

mqsideploy *command - Windows, Linux, and UNIX systems:*

Use the **mqsideploy** command on Windows, Linux, and UNIX systems to make a deployment request to the broker.

Syntax:



Parameters:

brokerSpec
(Required) You must specify at least one parameter to identify the target broker for this command, in one of the following forms:

brokerName
This parameter identifies the name of a locally defined broker. You cannot use this option if the broker is on a remote computer.

-n *brokerFileName*
This parameter identifies the name of a file that contains the connection details for a local or remote broker.

Use this option if multiple users want to connect to the same broker, or if you are using advanced connection parameters such as SSL.

To create this file, right-click the broker in the WebSphere Message Broker Explorer and select **Export *.broker file**. When prompted, navigate to the location in which you want to save the file and enter the file name; the extension `.broker` is appended automatically when you click **Save**. Include the location (path) and file name when you specify this parameter. You must ensure that the location is accessible when you run this command.

If you want to run a command that uses SSL to administer a remote broker over a secured channel, you must specify the keystore and truststore password for the connection using the IBM_JAVA_OPTIONS environment variable. See "Resolving problems when running commands" on page 3364 for further information.

-i *ipAddress*, **-p** *port*, **-q** *qMgr*
These parameters identify the connection details for the broker.

Use this option for connections to remote brokers that do not require advanced connection parameters.

If you choose this option, you must specify at least one of these three parameters; the order in which you specify them is insignificant. You cannot specify each parameter more than once.

Parameters that you omit assume default values:

- **-i** *ipAddress*: The host name or the IP address of the computer on which the broker is running. If you do not specify this parameter, a value that represents the local computer is used.
- **-p** *port*: The TCP port on which the broker's queue manager is listening. If you do not specify this parameter, the value 1414 is used.
- **-q** *qMgr*: The name of the broker's queue manager. If you do not specify this parameter, a value that represents the default queue manager on the local computer is used.

-a *BARFileName*
(Optional) This parameter specifies the name of the broker archive (BAR) file that is to be used for deployment of the message flow and other resources. You must also specify the **-e** parameter with this option.

-d *deployedObjects*
(Optional) This parameter describes the set of objects that you want to remove from the execution group. You can specify multiple files to delete by separating the filenames with a colon (:).

You can specify objects of all types, but if you specify an ambiguous object name (for example, "top", when both "top.dictionary" and "top.cmf" are deployed to the same execution group), the entire command fails with the message BIP1089. In these circumstances, you must specify the fully qualified name of the objects to remove; for example, "top.dictionary:top.cmf".

-e *executionGroupName*
(Optional) This parameter specifies the name of the execution group to which to deploy. You must also specify the **-a** parameter with this option.

-m
(Optional) This parameter specifies deployment of complete information:

The default operation is a delta or incremental deployment. Use the **-m** parameter to override the default operation and run a complete deployment.

- For a *BAR file deployment*, **-m** removes all currently-deployed message flows and message sets from the execution group as part of the deployment. If you do not set **-m**, the contents of the BAR file are deployed in addition to what is already deployed to the execution group. Any deployed objects with the same name as an item inside the BAR file are replaced by the version inside the BAR file.
- For a *remove message flow or message set operation*, the **-m** parameter is ignored.

-v *traceFileName*
(Optional) This parameter sends internal debug trace information to the specified file.

-w *timeoutSecs*
(Optional) This parameter specifies the maximum time in seconds that the command waits for the broker to complete the request before returning.

You can set this parameter to a value in the range 1 - 2 145 336 164. If you do not provide a *timeoutValue* value, or you set a value less than 1 or greater than 2 145 336 164 is specified, an error is returned.

Set this parameter to a value greater than the sum of the configuration timeout parameters **ConfigurationChangeTimeout** and **InternalConfigurationTimeout** that you specified for the broker, if you want to ensure that a response is received within the *timeoutValue* period. If you set a smaller value, the response returned might indicate that the state of the deploy request is unknown.

Examples:
The following examples show the use of the **-n** parameter to define the connection parameters for the broker; if you prefer, you can specify the **i**, **p**, and **q** parameters. If the broker is on the local computer, you can specify it by name.

Deploy a BAR file to the broker identified by the connection parameters in the file `b1.broker`, and remove all currently-deployed message flows and message sets from the execution group as part of the deployment. Allow 10 minutes for the broker to reply.

```
mqsideploy -n b1.broker -e default -a mybar.bar -m -w 600
```

Remove the message flow `top` and the dictionary `bar` from the execution group `default` on the broker identified by the connection parameters in the file `b1.broker`.

```
mqsideploy –n b1.broker –e default –d top.cmf:bar.dictionary
```

For information about using this command with SSL protected channels, see You want to run a command that uses SSL to administer a remote broker over a secured channel

Related concepts:

"Packaging and deployment overview" on page 3210
Deployment is the process of transferring data to an execution group on a broker so that it can take effect in the broker. Message flows and associated resources are

packaged in broker archive (BAR) files for deployment.

Related tasks:

Chapter 11, “Packaging and deploying,” on page 3209
Package resources that you create in the WebSphere Message Broker Toolkit, such as message flows, and deploy them to execution groups on brokers.

Related reference:

“Syntax diagrams” on page 3677

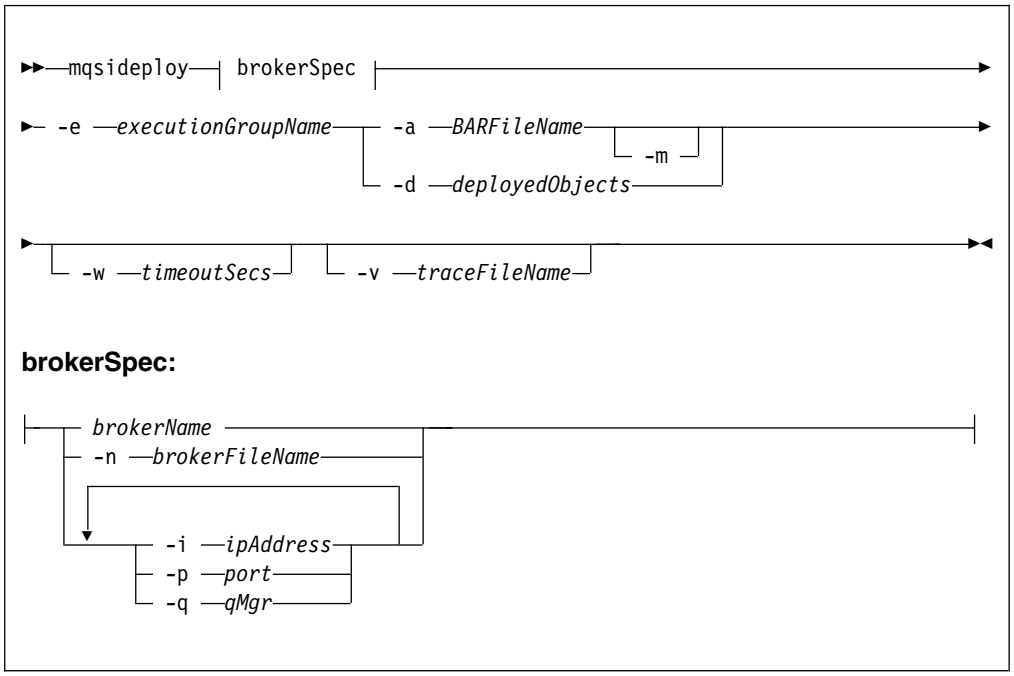
“**mqsideploy** command” on page 3872
Use the **mqsideploy** command to make a deployment request to the broker.

mqsideploy *command - z/OS:*

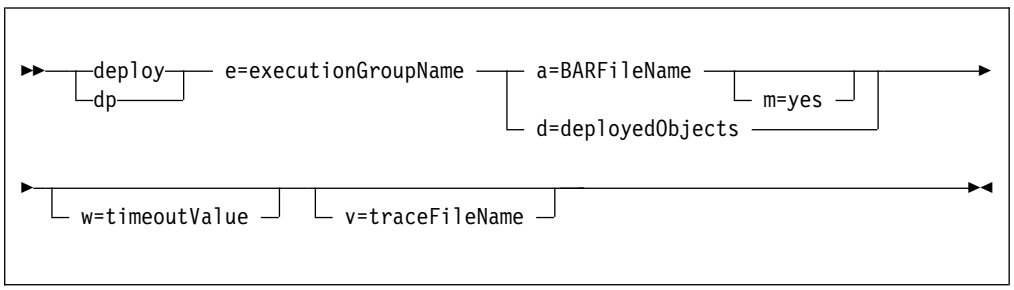
Use the **mqsideploy** command on z/OS to make a deployment request to a local broker.

Syntax:

z/OS command - BIPDPLY:



z/OS console command:
Synonym: dp



Parameters:

brokerSpec

You must specify at least one parameter to identify the target broker for this command, in one of the following forms:

brokerName
This parameter identifies the name of a locally defined broker.

-n *brokerFileName*
This parameter identifies the name of a file that contains the connection details for a local broker.

Use this option if multiple users want to connect to the same broker, or if you are using advanced connection parameters such as SSL.

To create this file, right-click the broker in the WebSphere Message Broker Explorer and select **Export *.broker file**. When prompted, navigate to the location in which you want to save the file and enter the file name; the extension `.broker` is appended automatically when you click **Save**. Include the location (path) and file name when you specify this parameter. You must ensure that the location is accessible when you run this command.

-i *ipAddress*, **-p** *port*, **-q** *qMgr*
These parameters identify the connection details for the broker. If you specify a queue manager, it must be defined on the local computer.

If you choose this option, you must specify at least one of these three parameters; the order in which you specify them is irrelevant. You cannot specify each parameter more than once.

Parameters that you omit assume default values:

- **-i** *ipAddress*: The host name or the IP address of the computer on which the broker is running. If you do not specify this parameter, a value that represents the local computer is used.
- **-p** *port*: The TCP port on which the broker's queue manager is listening. If you do not specify this parameter, the value 1414 is used.
- **-q** *qMgr*: The name of the broker's queue manager. If you do not specify this parameter, a value that represents the default queue manager on the local computer is used.

You cannot connect to a remote queue manager on z/OS; see "Usage note" on page 3878.

-a *BARFileName*
(Optional) This parameter specifies the name of the broker archive (BAR) file that is to be used for deployment of the message flow and other resources. You must also specify the **-e** parameter with this option.

The BAR file can be in a local or remote file system, if the user ID or the broker that is running the command can access the file and read it.

-d *deployedObjects*
(Optional) This parameter describes the set of objects that you want to remove from the execution group. You can specify multiple files to delete by separating the filenames with a colon (:).

You can specify objects of all types, but if you specify an ambiguous object name (for example, "top", when both "top.dictionary" and "top.cmf" are deployed to the same execution group), the entire command fails with the

message BIP1089. In these circumstances, you must specify the fully qualified name of the objects to remove; for example, "top.dictionary:top.cmf".

-e *executionGroupName*
(Optional) This parameter specifies the name of the execution group to which to deploy. You must also specify the **-a** parameter with this option.

-m (Optional) This parameter specifies deployment of complete information:

The default operation is a delta or incremental deployment. Use the **-m** parameter to override the default operation and run a complete deployment.

- For a *BAR file deployment*, **-m** removes all currently-deployed message flows and message sets from the execution group as part of the deployment. If you do not set **-m**, the contents of the BAR file are deployed in addition to what is already deployed to the execution group. Any deployed objects with the same name as an item inside the BAR file are replaced by the version inside the BAR file.
- For a *remove message flow or message set operation*, the **-m** parameter is ignored.

-v *traceFileName*
(Optional) This parameter sends internal debug trace information to the specified file.

-w *timeoutSecs*
(Optional) This parameter specifies the maximum time in seconds that the command waits for the broker to complete the request before returning.

You can set this parameter to a value in the range 1 - 2 145 336 164. If you do not provide a *timeoutValue* value, or you set a value less than 1 or greater than 2 145 336 164 is specified, an error is returned.

Set this parameter to a value greater than the sum of the configuration timeout parameters **ConfigurationChangeTimeout** and **InternalConfigurationTimeout** that you specified for the broker, if you want to ensure that a response is received within the *timeoutValue* period. If you set a smaller value, the response returned might indicate that the state of the deploy request is unknown.

Usage note:
If you try to connect to a remote broker by specifying the **i**, **p**, and **q** parameters, or by using a connection parameter file (`.broker`), the command attempts to use WebSphere MQ Java client code. This option is not supported on z/OS, and returns the following error:

```
BIP1046E: Unable to connect with the broker (name)
```

The reported reason code is `MQRC_ENVIRONMENT_ERROR`. You must specify a local queue manager.

Examples:
The following examples show the use of the **-n** parameter to define the connection parameters for the broker. If the broker is on the local computer, you can specify it by name.

Deploy a BAR file to the broker identified by the connection parameters in the file `b1.broker`, and remove all deployed message flows and message sets from the execution group as part of the deployment. Allow 10 minutes for the broker to reply.

```
mqsideploy -n broker1.broker -e default -a mybar.bar -m -w 600
```

Remove the message flow `top` and the dictionary `bar` from the execution group `default` on the broker identified by the connection parameters in the file `b1.broker`.

```
mqsideploy –n b1.broker –e default –d top.cmf:bar.dictionary
```

Deploy a BAR file by using the console command:

```
F MQ01BRK,dp e=default,a=flows.bar
```

Related concepts:

"Packaging and deployment overview" on page 3210
Deployment is the process of transferring data to an execution group on a broker so that it can take effect in the broker. Message flows and associated resources are packaged in broker archive (BAR) files for deployment.

Related tasks:

Chapter 11, "Packaging and deploying," on page 3209
Package resources that you create in the WebSphere Message Broker Toolkit, such as message flows, and deploy them to execution groups on brokers.

Related reference:

"Syntax diagrams" on page 3677

"**mqsideploy** command" on page 3872
Use the **mqsideploy** command to make a deployment request to the broker.

mqsiexplain command:

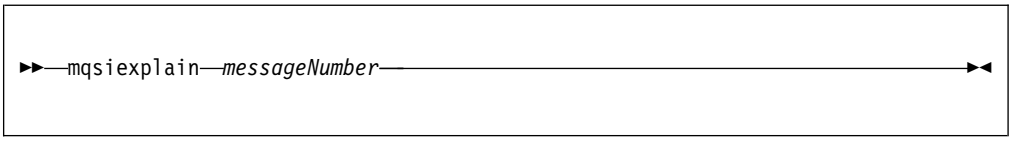
Use the **mqsiexplain** command to display the contents of a WebSphere Message Broker BIP message.

Supported platforms:

- Windows
- Linux and UNIX systems

Purpose:
The **mqsiexplain** command returns the full details of a BIP message that you specify, including the user response and explanation sections.

Syntax:



Parameters:

messageNumber
(Required) The BIP message for which you want to display details. You can identify the message by using a number only (for example, 2393), or you can precede the number with "BIP" (for example, BIP2393). You can also include the severity marker (I, E, or W) at the end of the number (for example, BIP2393E).

Authorization:
The user ID that is used to run this command must have Administrator authority on the local system.

Responses:
This command shows the full contents of the specified message, including the user response and explanation sections. Where inserts are expected in the message, this command substitutes them with "insert#".

Examples:
The following example is used to display the full content of error message BIP2393:

```
mqsiexplain 2393
```

The command returns the following information:

```
BIP2393E: Database error: ODBC return code 'insert1' from data source 'insert2' using ODBC driver mar
The broker received an error when processing a database operation. The ODBC return code was 'insert1
Use the following messages to determine the cause of the error. Typical problems are an incorrect dat

BIP8071I: Successful command completion.
```

The following example is used to display the full content of error message BIP3595:

```
mqsiexplain BIP3595E
```

The command returns the following information:

```
BIP3595E: The connection with the ID: 'insert1' on Hostname: 'insert2' and Port: 'insert3' exists but
There was a connection available with the specified details but no data arrived within the permitted
Ensure that the end application is running correctly and sending data on this connection.

BIP8071I: Successful command completion.
```

Related tasks:

"Are there any error messages or return codes that explain the problem?" on page 3350
You can find details of error messages and return codes in several places.

Related reference:

Diagnostic messages
Diagnostic messages are listed in this section in numeric order, grouped according to the component to which they relate.

"Troubleshooting" on page 6864
Use the reference information in this section to help you diagnose errors in WebSphere Message Broker.

"Syntax diagrams" on page 3677

mqsiformatlog command:

Use the **mqsiformatlog** command to process the XML log created by **mqsireadlog**. The command retrieves and formats any messages that the XML log contains into a form suitable for the locale of the user who runs the command.

Supported platforms:

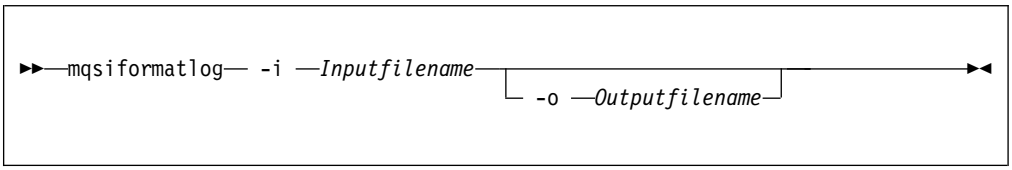
- Windows
- Linux and UNIX systems
- z/OS. Run this command by customizing and submitting BIPFMLG; see "Contents of the broker PDSE" on page 3991

Purpose:
The **mqsiformatlog** command interprets an input log file that has been created on any system in a platform-independent code page, `utf-8`. Use this command to produce formatted output from input log files transferred from other systems to

the system on which you issue the command. If you use this facility, ensure that you use a file transfer program that does not convert the data (for example, by specifying a binary transfer option).

You can direct the output to a file, or to the command shell.

Syntax:



Parameters:

-i *Inputfilename*
(Required) The filename of the XML log file that is to be formatted. This file is created by the **mqsireadlog** command; it is encoded in utf-8.

-o *Outputfilename*
(Optional) The filename of the file into which the formatted log output is to be written. If this is not specified, the formatted log data is written to stdout.

Output written by this command (to file or stdout) is written in a code page suitable for the current user locale.

Authorization:
For information about platform-specific authorizations, see the following topics:

- "Security requirements for Linux and UNIX platforms" on page 3648
- "Security requirements for Windows systems" on page 3651
- "Security requirements for z/OS" on page 3655

If you have enabled broker administration security, you must also set up the authority detailed in "Tasks and authorizations for broker administration security" on page 3645.

Examples:

```
mqsiformatlog -i trace.xml -o formattrace.log
```

The following extract shows the output that is generated by this command:

```
Timestamps are formatted in local time, local time is GMT.
.
.
.
2003-02-12 12:57:21.895999    388  UserTrace   BIP2638E:
MQPUT to queue 'SYSTEM.BROKER.EXECUTIONGROUP.REPLY' on queue manager
'MB7QMGR': MQCC=0, MQRC=0; node ConfigurationMessageFlow.outputNode'.
The node 'ConfigurationMessageFlow.outputNode' attempted
to write a message to the specified queue 'SYSTEM.BROKER.EXECUTIONGROUP.REPLY'
connected to queue manager 'MB7QMGR'.
The MQCC was 0 and the MQRC was 0.
No user action required.

2003-02-12 12:57:21.895999    388  UserTrace   BIP2622I:
Message successfully output by output node 'ConfigurationMessageFlow.outputNode'
to queue 'SYSTEM.BROKER.EXECUTIONGROUP.REPLY' on queue manager
'MB7QMGR'. The WebSphere MQ output node ConfigurationMessageFlow.outputNode'
successfully wrote an output message to the specified queue
```

```
SYSTEM.BROKER.EXECUTIONGROUP.REPLY connected to queue manager MB7QMGR.
No user action required.
.
.
.
Threads encountered in this trace:  335  388
```

Related tasks:

"Using logs" on page 3526
There are a variety of logs that you can use to help with problem determination and troubleshooting.

Related reference:

"Syntax diagrams" on page 3677

"**mqsichangetrace** command" on page 3822
Use the **mqsichangetrace** command to set the tracing characteristics for a broker.

"**mqsireadlog** command" on page 3905
Use the **mqsireadlog** command to retrieve trace records for the specified component.

"**mqsireporttrace** command" on page 3947
Use the **mqsireporttrace** command to display the trace options currently in effect. Trace can be run only against a broker, or any of its resources.

mqsilist command:

Use the **mqsilist** command to list installed brokers and their associated resources.

Supported platforms:

- Windows
- Linux and UNIX systems
- z/OS. Run this command in one of two ways - as a console command, or by customizing and submitting BIPLIST; see "Contents of the broker PDSE" on page 3991

Purpose:
The **mqsilist** command reports on the configuration of one or more brokers. Your choice of parameters on the command determines which resources are included in the response, and the level of detail that the response provides.

The command returns information about the following resources:

- Local and remote brokers
- Execution groups defined on the brokers
- All resources that you have deployed to each execution group, including message flows and message sets
- Runtime versioning information for those resources, if applicable

You can also specify that you want the command to recursively examine resources and return information, so that you need only specify a single resource to get details about its children.

You can choose the level of detail that you want returned for each resource requested:

- A list of resource names. This level of output is compatible with previous versions of WebSphere Message Broker.

- A one line summary for each resource, including name and active or inactive status.
- A detailed view of each resource, including build level and platform for brokers, and resources that have been deployed to each execution group.

On Windows platforms, Linux, and UNIX systems, the command also reports whether a broker is configured as a WebSphere MQ service.

Only a subset of this information is returned if the broker is not running.

On Windows, Linux, and UNIX systems, the output is directed to STDOUT.

Select the appropriate link for details of this command on the platform, or platforms, that your enterprise uses:

- "**mqsilist** command - Windows, Linux, and UNIX systems"
- "**mqsilist** command - z/OS" on page 3888

Authorization:
For information about platform-specific authorizations, see the following topics:

- "Security requirements for Linux and UNIX platforms" on page 3648
- "Security requirements for Windows systems" on page 3651
- "Security requirements for z/OS" on page 3655

If you have enabled broker administration security, you must also set up the authority detailed in "Tasks and authorizations for broker administration security" on page 3645.

Related tasks:

"Starting and stopping a broker" on page 921
Run the appropriate command to start or stop a broker.

"Verifying brokers" on page 630
Use the **mqsilist** command to display the brokers that you have created on your computer.

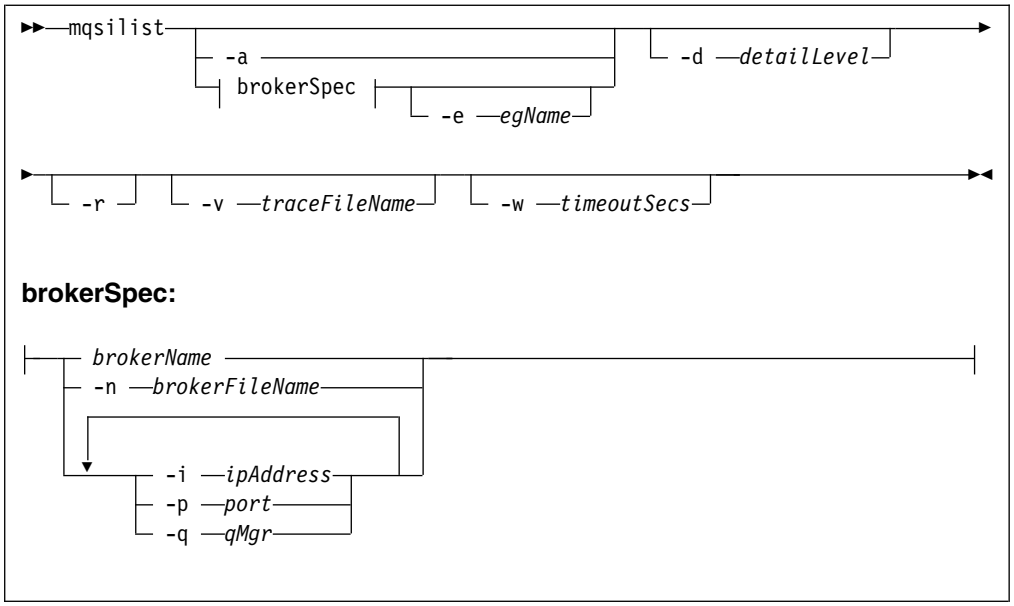
Related reference:

"Syntax diagrams" on page 3677

mqsilist *command - Windows, Linux, and UNIX systems:*

Use the **mqsilist** command to list information about one or more brokers and their deployed resources.

Syntax:



If you specify this command with no parameters, a summary is returned for broker that you have created in the current installation on this computer.

The current installation is associated with the command console that you have opened (on Windows), or the **mqsiprofile** that is active (on Linux and UNIX systems).

Parameters:

brokerSpec
(Required) You must specify at least one parameter to identify the target broker for this command, in one of the following forms:

brokerName
This parameter identifies the name of a locally defined broker. You cannot use this option if the broker is on a remote computer.

-n *brokerFileName*
This parameter identifies the name of a file that contains the connection details for a local or remote broker.

Use this option if multiple users want to connect to the same broker, or if you are using advanced connection parameters such as SSL.

To create this file, right-click the broker in the WebSphere Message Broker Explorer and select **Export *.broker file**. When prompted, navigate to the location in which you want to save the file and enter the file name; the extension `.broker` is appended automatically when you click **Save**. Include the location (path) and file name when you specify this parameter. You must ensure that the location is accessible when you run this command.

If you want to run a command that uses SSL to administer a remote broker over a secured channel, you must specify the keystore and truststore password for the connection using the IBM_JAVA_OPTIONS environment variable. See "Resolving problems when running commands" on page 3364 for further information.

-i *ipAddress*, **-p** *port*, **-q** *qMgr*

These parameters identify the connection details for the broker.

Use this option for connections to remote brokers that do not require advanced connection parameters.

If you choose this option, you must specify at least one of these three parameters; the order in which you specify them is insignificant. You cannot specify each parameter more than once.

Parameters that you omit assume default values:

- **-i** *ipAddress*: The host name or the IP address of the computer on which the broker is running. If you do not specify this parameter, a value that represents the local computer is used.
- **-p** *port*: The TCP port on which the broker's queue manager is listening. If you do not specify this parameter, the value 1414 is used.
- **-q** *qMgr*: The name of the broker's queue manager. If you do not specify this parameter, a value that represents the default queue manager on the local computer is used.

-a (Optional) List all the brokers installed on the local computer, in all installations.

If you specify this option on a computer on which you have installed versions earlier than Version 7.0, only brokers are listed for those versions, if they exist; the level of detail corresponds to **-d 0** for these earlier brokers, even if you have specified (or allowed to default) a different level of detail on this command. If you want to view information about other components that you have created in earlier versions, you must use the corresponding version of the **mqsilist** command.

You cannot use this option to list information about remote brokers and their resources.

-d *detailLevel*

(Optional) Specify the level of detail to be returned; the default value is **1**.

- **0** returns only broker name and the names of their associated queue managers (this information is the same as the detail provided in previous versions). This list of local brokers is returned without connecting to a queue manager, and remote broker options are not supported.

  Note that while they are starting, execution groups might report a PID of `0`.
- **1** returns a one line summary of each resource.
- **2** returns detailed information about each resource.

-e *egName*

(Optional) Selects an execution group within a broker. Specify the label of the execution group for which you want to list message flows. The command returns a list of message flows assigned to the specified execution group within the broker.

The specified broker must be active for message flow information to be returned.

-r (Optional) Run the command recursively; display information about subcomponents.

-v *traceFileName*

(Optional) This parameter sends internal debug trace information to the specified file.

-w *timeoutSecs*
(Optional) This parameter specifies the time in seconds that the utility waits to ensure that the command completed; the default value is 60.

Examples:
The following example lists all brokers created in the current installation:

```
mqsilist -d0
```

The following responses are generated:

```
BIP8099I: Broker: MB7BROKER  -  MB7QMGR
BIP8099I: Broker: test  -  testqm

BIP8071I: Successful command completion.
```

The following example requests a summary of the execution groups that are defined on a specific broker. (The **-d** parameter is not specified and therefore has the default value of 1.)

```
mqsilist MB7BROKER
```

The following responses are generated:

```
BIP1286I: Execution group 'ello' on broker 'MB7BROKER' is running.

BIP8071I: Successful command completion.
```

Display detailed information about all resources for brokers on the local computer:

```
mqsilist -a -r -d2
```

The following responses are generated:

```
BIP1280I: The number of runtime installations on this machine is '1'.

 7.0.0.0 : C:\Program Files\IBM\MQSI\7.0

==================================
BIP1284I: Broker 'MB7BROKER' on queue manager 'MB7QMGR' is running.

Broker version: '7000' (build 'S000-L90708')
Platform: 'Microsoft Windows XP', '5.1 build 2600 Service Pack 3', 'x86'
Process ID: '3116'
Short description: ''
Long description: ''

----------------------------------
BIP1286I: Execution group 'ello' on broker 'MB7BROKER' is running.

Number of message flows that are enabled to run: '2'.
Number of applications that are enabled to run: '1'.
Process ID: '4220'
UUID: '45ef67ac-2201-0000-0080-f364270ba85e'
Short description: ''
Long description: ''

--------
BIP1288I: Message flow 'simpleflow' on execution group 'ello' is running.

Additional thread instances: '0'
Deployed: '24/07/09 16:37' in Bar file 'C:\Documents and Settings\Matt\My Documents\BAR Files\test.ba
Last edited: '08/08/07 17:42'
User-defined property names:
Keywords:
  Author = 'Matt'
  Information = 'This flow simply removes messages from SYSTEM.DEFAULT.LOCAL.QUEUE'
```

```
  Usage = 'This usage is buried inside the CMF'
  VERSION = 'v1.1'
--------
BIP1288I: Message flow 'MyApplicationFlow' on execution group 'ello' is running.

Additional thread instances: '0'
Deployed: '27/07/09 20:15' in Bar file 'C:\Documents and Settings\Matt\My Documents\BAR Files\over
Last edited: '12/06/07 17:06'
User-defined property names:
Keywords:

--------
BIP1290I: File 'Swift_2002_MT103.dictionary' is deployed to execution group 'ello'.

Deployed: '27/07/09 16:47' in Bar file 'C:\Documents and Settings\Matt\My Documents\BAR Files\SWIF
Last edited: '06/09/05 15:17'.
Keywords:

==================================
BIP1285I: Broker 'test' on queue manager 'testqm' is stopped.

Start the broker for more information.

BIP8071I: Successful command completion.
```

The following response is generated when a broker is a multi-instance broker in Standby mode:

```
BIP1280I: The number of runtime installations on this machine is '1'.

 7.0.0.0 : C:\Program Files\IBM\MQSI\7.0

==================================
BIP1292I: Broker 'MIBROKER1' is a multi-instance broker .
The broker instance is running in Standby-mode on multi-instance
Queue manager 'MIQMGR1'

More information will be available when the broker instance is active.

BIP8071I: Successful command completion
```

The following response is generated when a broker is a multi-instance broker in Active mode:

```
BIP1280I: The number of runtime installations on this machine is '1'.

 7.0.0.0 : C:\Program Files\IBM\MQSI\7.0

==================================
BIP1293I: Broker 'MIBROKER1' is a multi instance broker .
The broker instance is active and running on multi-instance
Queue manager 'MIQMGR1'

----------------------------------
BIP1286I: Execution group 'default' on broker 'MIBROKER1' is running.

Number of message flows that are enabled to run: '1'.
Number of applications that are enabled to run: '1'.
Process ID: '4876'
UUID: '45ef7be4-2203-0000-0080-b764270fa96c'
Short description: ''
Long description: ''

--------
BIP1288I: Message flow 'jmsflow' on execution group 'default' is running.

Additional thread instances: '0'
```

```
Deployed: '26/09/09 18:21' in Bar file 'C:\Documents and Settings\JDoe\My Documents\
  BAR Files\test.bar'
Last edited: '24/09/09 12:16'
User-defined property names:
Keywords:
  Author = 'JDoe'
  Information = 'This flow runs a simple jms scenario'
  Usage = 'Takes JMS input and writes to an MQOutput node'
  VERSION = 'v2.3'
--------

BIP8071I: Successful command completion.
```

The following response is generated when a broker is a multi-instance broker being started as a WebSphere MQ service:

```
BIP1296I: Broker 'HABK1' is a multi-instance broker that will be started as a WebSphere MQ service.
Multi-instance queue manager 'HAQM1' is stopped.
BIP1298I: Broker 'BK3' will be started as a WebSphere MQ service.
Queue manager 'QM3' is stopped.
```

:

Related reference:

"**mqsilist** command" on page 3882
Use the **mqsilist** command to list installed brokers and their associated resources.

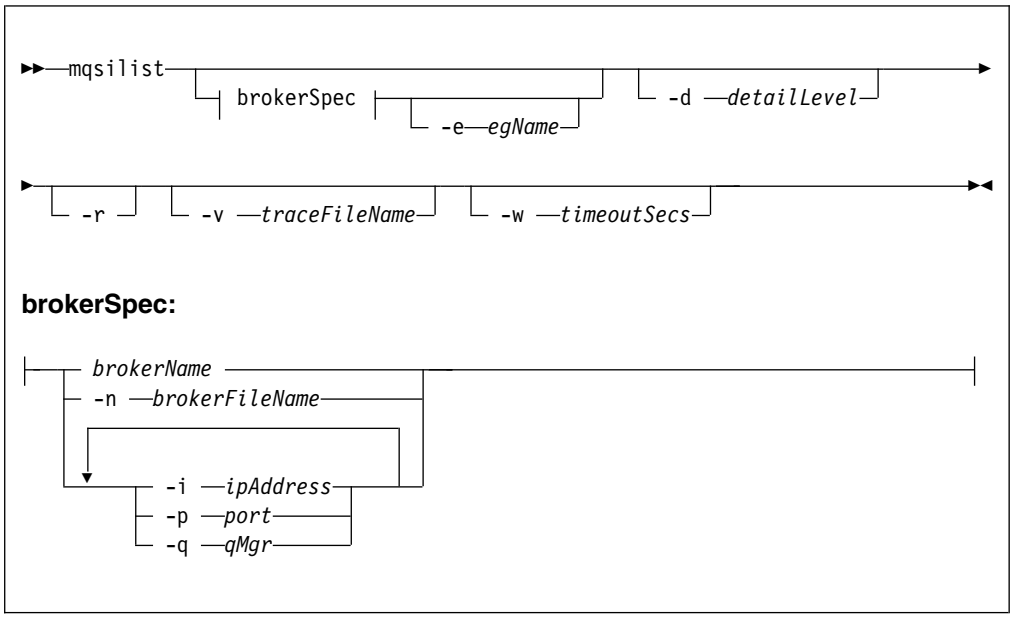
"Syntax diagrams" on page 3677

mqsilist *command - z/OS:*

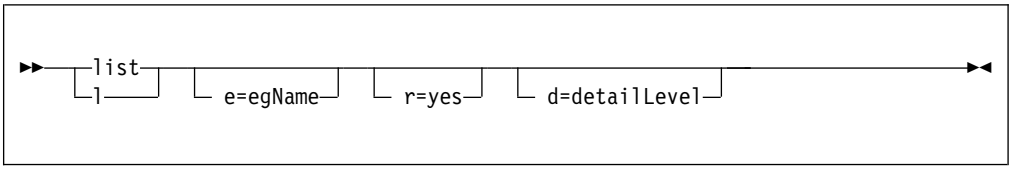
Use the **mqsilist** command to list information about one or more brokers and their deployed resources.

Syntax:

z/OS command - BIPLIST:



z/OS console command:
Synonym: l



If you specify this command with no parameters, a list of the execution groups is displayed.

Parameters:

brokerSpec
You must specify at least one parameter to identify the target broker for this command, in one of the following forms:

brokerName
This parameter identifies the name of a locally defined broker.

-n *brokerFileName*
This parameter identifies the name of a file that contains the connection details for a local broker.

Use this option if multiple users want to connect to the same broker, or if you are using advanced connection parameters such as SSL.

To create this file, right-click the broker in the WebSphere Message Broker Explorer and select **Export *.broker file**. When prompted, navigate to the location in which you want to save the file and enter the file name; the extension `.broker` is appended automatically when you click **Save**. Include the location (path) and file name when you specify this parameter. You must ensure that the location is accessible when you run this command.

-i *ipAddress*, **-p** *port*, **-q** *qMgr*
These parameters identify the connection details for the broker. If you specify a queue manager, it must be defined on the local computer.

If you choose this option, you must specify at least one of these three parameters; the order in which you specify them is irrelevant. You cannot specify each parameter more than once.

Parameters that you omit assume default values:

- **-i** *ipAddress*: The host name or the IP address of the computer on which the broker is running. If you do not specify this parameter, a value that represents the local computer is used.
- **-p** *port*: The TCP port on which the broker's queue manager is listening. If you do not specify this parameter, the value 1414 is used.
- **-q** *qMgr*: The name of the broker's queue manager. If you do not specify this parameter, a value that represents the default queue manager on the local computer is used.

You cannot connect to a remote queue manager on z/OS; see "Usage note" on page 3890.

-d *detailLevel*
(Optional) Specify the level of detail to be returned; the default value is **1**.

- **0** returns only broker name and the names of their associated queue managers (this information is the same as the detail provided in previous versions).

  Note that while they are starting, execution groups might report a PID of `0`.
- **1** returns a one line summary of each resource.
- **2** returns detailed information about each resource.

-e *egName*
(Optional) Selects an execution group within a broker. Specify the label of the execution group for which you want to list message flows. The command returns a list of message flows assigned to the specified execution group within the broker.

The specified broker must be active for message flow information to be returned.

-r (Optional) Run the command recursively; display information about subcomponents.

-v *traceFileName*
(Optional) This parameter sends internal debug trace information to the specified file.

-w *timeoutSecs*
(Optional) This parameter specifies the time in seconds that the utility waits to ensure that the command completed; the default value is 60.

Usage note:
If you try to connect to a remote broker by specifying the **i**, **p**, and **q** parameters, or by using a connection parameter file (`.broker`), the command attempts to use WebSphere MQ Java client code. This option is not supported on z/OS, and returns the following error:

```
BIP1046E: Unable to connect with the broker (name)
```

The reported reason code is `MQRC_ENVIRONMENT_ERROR`. You must specify a local queue manager.

Examples:
Use the console command to request a summary of the execution groups for the broker. (The **d=** parameter is not specified and therefore has the default value of 1.)

```
F MQP1BRK,list
```

The following responses are generated:

```
BIP1286I: Execution group 'ello' on broker 'MQP1BRK' is running.

BIP8071I: Successful command completion.
```

Use the console command to list the message flows in the specified execution group:

```
F MQP1BRK,list e='exgrp1'
```

The responses generated are in the following form:

```
BIP1288I: Message flow 'simpleflow' on execution group 'ello' is running.

BIP8071I: Successful command completion
```

For examples of running the command by using BIPLIST, and the responses that are generated, see "**mqsilist** command - Windows, Linux, and UNIX systems" on page 3883.

:

Related reference:

"**mqsilist** command" on page 3882
Use the **mqsilist** command to list installed brokers and their associated resources.

"Syntax diagrams" on page 3677

mqsimanagexalinks command:

Use the **mqsimanagexalinks** command to set up links for a supported database on and UNIX systems for XA coordination under the control of WebSphere MQ.

Supported platforms:

- Linux and UNIX systems

Purpose:
The command sets up or removes the links required by WebSphere MQ to include databases in XA transactions.

You can also use the **mqsimanagexalinks** command to return a list of supported ODBC drivers.

Run the **mqsimanagexalinks** command before you create the database that you want the broker to connect to. You can run the command before or after you install your database; specify either the intended or the actual database installation directory where indicated.

Select the appropriate link for details of this command on the platform, or platforms, that your enterprise uses:

- "**mqsimanagexalinks** command - Linux and UNIX systems"

Related tasks:

"Working with databases" on page 2109
Create and configure databases to use with your message flow applications.

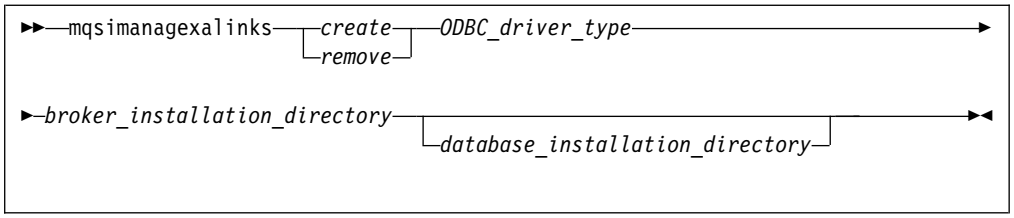
"Configuring databases for global coordination of transactions" on page 665
If your message flow interacts with a user database, and you want to globally coordinate the updates made to the database with other actions within the message flow, configure your databases for global coordination.

"Configuring ODBC connections for globally coordinated transactions" on page 699
Configure the definition of your ODBC databases to the transaction manager (the queue manager).

mqsimanagexalinks *command - Linux and UNIX systems:*

Set up links for a supported database on Linux and UNIX systems for XA coordination under the control of WebSphere MQ.

Syntax:



Parameters:

create
(Required) Specify that the required links are created.

remove
(Required) Specify that the defined links are removed.

ODBC_driver_type
(Required) The ODBC driver type. See the list command for the list of ODBC drivers that are supported.

This parameter is required if you specify **create**, and is optional if you specify **remove**. If you specify **remove** and omit this parameter, all links are removed.

broker_installation_directory
(Required) The full path to your broker installation.

This parameter is required only if you specify **create**.

database_installation_directory
(Optional). The full path to your database installation.

This parameter is required only if you specify **create**, and have created a DB2 database.

list
(Required) Display the list of supported ODBC driver versions. Use the output to check what values you can specify for **ODBC_driver_type**.

For example, the following output is generated on AIX:

Supported information for the **create** option.

Supported version of the ODBC drivers	Version number for **mqsimanagexalinks**
DB2	DB2
DataDirect Connect for ODBC V6.0	DD60

Supported information for the **remove** option.

Supported version of the ODBC drivers	Version number for **mqsimanagexalinks**
DB2	DB2
DataDirect Connect for ODBC V5.2	DD52
DataDirect Connect for ODBC V5.3	DD53
DataDirect Connect for ODBC V6.0	DD60

Links created in /var/mqm/exits:

Links are only created in `/var/mqm/exits` on Linux on x86. The links created for each supported database are shown here.

DB2

database_install_dir/lib32/libdb2.*so*
database_install_dir/lib32/libdb2.*so.1*
broker_install_dir/sample/xatm/db2swit

Oracle

broker_install_dir/ODBC/V6.0/lib/libUKicu24.*so*
broker_install_dir/ODBC/V6.0/lib/UKora24.*so*
broker_install_dir/ODBC/V6.0/lib/UKoradtc24.*so*
broker_install_dir/ODBC/V6.0/lib/libodbcinst.*so*

Sybase

broker_install_dir/ODBC/V6.0/lib/libUKicu24.*so*
broker_install_dir/ODBC/V6.0/lib/UKase24.*so*
broker_install_dir/ODBC/V6.0/lib/UKasedtc24.*so*
broker_install_dir/ODBC/V6.0/lib/libodbcinst.*so*

Links created in `/var/mqm/exits64`:
Links are created in `/var/mqm/exits64` on all Linux and UNIX systems except on Linux on x86. The links created are shown later in this section for each supported database. Not all links are created on all platforms; the links created are determined by the database support on each platform. In the links shown, `x` represents the relevant library extension on your platform.

DB2

database_install_dir/lib64/libdb2.*x*
database_install_dir/lib64/libdb2.*x.1*
broker_install_dir/sample/xatm/db2swit

Where *.x* is the library extension on your platform: `.a` on AIX or `.so` on all other platforms.

Oracle

broker_install_dir/ODBC/V6.0/lib/libUKicu24.*x*
broker_install_dir/ODBC/V6.0/lib/UKora24.*so*
broker_install_dir/ODBC/V6.0/lib/UKoradtc24.*so*
broker_install_dir/ODBC/V6.0/lib/libodbcinst.*x*

Where *.x* is the library extension on your platform: `.a` on AIX or `.so` on all other platforms.

Sybase

broker_install_dir/ODBC/V6.0/lib/libUKicu24.*x*
broker_install_dir/ODBC/V6.0/lib/UKase24.*so*
broker_install_dir/ODBC/V6.0/lib/UKasedtc24.*so*
broker_install_dir/ODBC/V6.0/lib/libodbcinst.*x*

Where *.x* is the library extension on your platform: `.a` on AIX or `.so` on all other platforms.

Authorization:
The user ID with which you run this command must be **root**.

Examples:
To create the links required for DB2:

```
mqsimanagexalinks create DB2 /opt/IBM/mqsi/V7.0 /opt/IBM/db2/V9.1
```

To create the links required for Oracle and Sybase databases (these databases use the DataDirect V5.3 drivers):

```
mqsimanagexalinks create DD60 /opt/IBM/mqsi/V7.0
```

To remove the links required for Oracle and Sybase databases (these databases previously used the DataDirect V5.3 drivers):

```
mqsimanagexalinks remove DD53
```

To list the supported database drivers:

```
mqsimanagexalinks list
```

Related tasks:

"Working with databases" on page 2109
Create and configure databases to use with your message flow applications.

"Configuring databases for global coordination of transactions" on page 665
If your message flow interacts with a user database, and you want to globally coordinate the updates made to the database with other actions within the message flow, configure your databases for global coordination.

"Configuring ODBC connections for globally coordinated transactions" on page 699
Configure the definition of your ODBC databases to the transaction manager (the queue manager).

Related reference:

"Syntax diagrams" on page 3677

`mqsimigratecomponents` command:

Use the **mqsimigratecomponents** command to migrate a component from a previously installed version of the product to another version on the same computer.

Supported platforms:

- Windows.
- Linux and UNIX systems.
- z/OS. Run this command by customizing and submitting BIPMGCMP.

Purpose:
Migrate components to WebSphere Message Broker Version 7.0 from Version 6.1 or from Version 6.0:

- If you are migrating from Version 6.1 you must have installed Version 6.1.0.3 (Fix Pack 3) or later.

  If you are using WebSphere Application Server with WebSphere Message Broker, or you have publish/subscribe applications that use the SubIdentity option, you must upgrade WebSphere Message Broker to Fix Pack 4 before you can migrate to WebSphere Message Broker Version 7.0.
- If you are migrating from Version 6.0, you must have installed Version 6.0.0.9 (Fix Pack 9) or later.

  If you are using WebSphere Application Server with WebSphere Message Broker, or you have publish/subscribe applications that use the SubIdentity option, you

must upgrade WebSphere Message Broker Version 6.0 or WebSphere Event Broker Version 6.0 to Fix Pack 10 before you can migrate to WebSphere Message Broker Version 7.0.

When you migrate a broker to Version 7.0, its configuration is read from the broker database, and migrated to an internal repository that is maintained by the migrated broker. The broker database used by the Version 6.1 or Version 6.0 broker is no longer required by the Version 7.0 broker. When you have successfully completed migration, you can delete the database, or uninstall the database product, if you have no further use of it.

You can also use the **mqsimigratecomponents** command to return a broker from Version 7.0 to an earlier version to reverse the effects of forward migration; however, the broker is restored with the configuration that was active at the time of the forward migration, and updates that you have made after migration are not reflected in the database that is associated with the broker at the earlier version. You must retain the relevant broker database to be able to use the broker at the earlier version.

You must run this command from whichever version of the installed product is the later, regardless of whether it is the source version or the target version.

You must have an installation of the product at both target and source versions, with the required component code installed, to issue this command successfully.

Before you start migration, stop the broker and all active debug sessions in the WebSphere Message Broker Toolkit. You cannot migrate message flows that are being debugged.

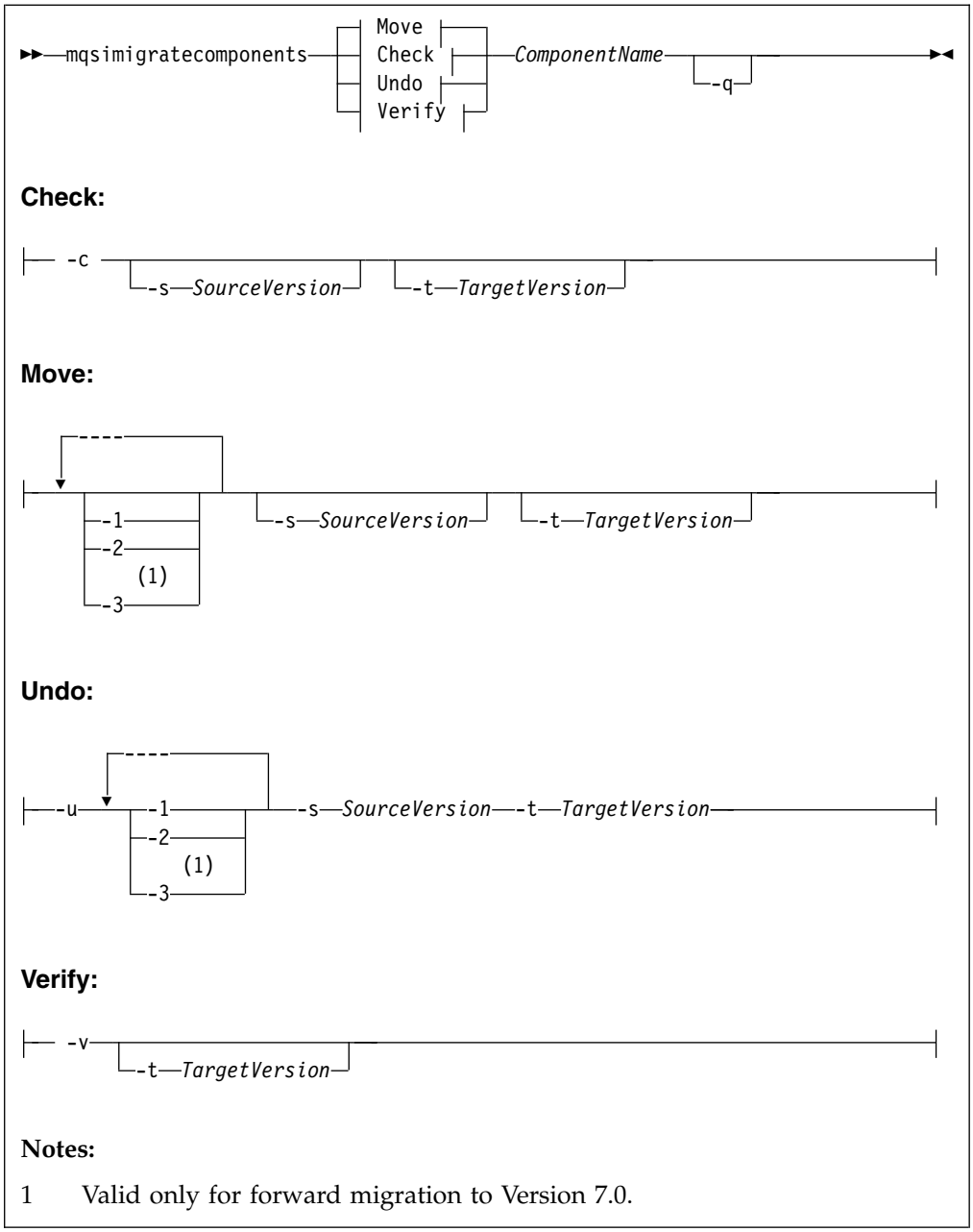
Specify appropriate options on this command to perform one of the following actions:

- Check that the component is suitable for the required migration, without changing that component (**-c**).
- Move a component to a different version, in full or part (**-s** and **-t**).
- Undo a failed migration step (**-u**).
- Verify that a move has been successful (**-v**).

Usage notes:
If you have specified a data source user ID and password on the **mqsicreatebroker** command for the broker that you are migrating, the values of these parameters are also migrated and saved in the format that is used by the **mqsisetdbparms** command. The values are used by the broker to access user databases for which you have not set alternative values by using the **mqsisetdbparms** command. After migration, if you want to change the user IDs or passwords that the broker uses to access user databases, you can use only the **mqsisetdbparms** command.

If you update the user ID and password values, and you migrate the broker back to the previous version, the new values are also migrated back to the original broker.

Syntax:



Parameters:

-c

(Optional) Check a specified component before migration, to ensure that:

- The auto-detected version of the broker matches any version specified on the command line.
- Any database tables accessed in a previous release do not contain any rows that are incorrectly indexed.

You can check a running component. The check does not affect the component, apart from a slight degradation of performance.

The check command either succeeds or fails, and prints a message about whether the migration will succeed, but no modifications are made during the process.

The **-c** and **-v** parameters are mutually exclusive. Additionally, if you specify either of these parameters, you cannot specify any other parameter when you run this command.

-v

(Optional) Check a specified component after migration, to ensure that:

- The registry is in the correct format for the new version.
- The correct queues exist for the new version.

The **-c** and **-v** parameters are mutually exclusive. Additionally, if you specify either of these parameters, you cannot specify any other parameter when you run this command.

-q

(Optional) Print fewer status messages during the operation.

-1

(Optional) Do only registry and file system work.

- When you migrate to Version 7.0, use the **-1** parameter before the **-2** or **-3** parameters.
- When you migrate backwards from Version 7.0 to a previous version, use the **-2** parameter before the **-1** parameter.

-2

(Optional) Do only WebSphere MQ work.

-3 (Optional) Do only database work.

This option is valid only for forward migration to Version 7.0. If you specify this parameter for backwards migration, it is ignored; changes that you have made to the broker state cannot be migrated back and applied to the database belonging to the broker at the earlier version.

-u (Optional) Undo a failed migration step; you must also specify at least one of **-1**, **-2**, or **-3**. Use this option only when migration has failed, and also failed to auto-recover (for example, if a failure occurs during split migration).

-3 is valid only for forward migration to Version 7.0.

-s *SourceVersion*

(Optional) The previous version of the component.

- If not specified, this value is detected automatically.
- When you perform split migration to Version 7.0, the **-s** parameter is mandatory after you run the **mqsimigratecomponents** command with the **-1** parameter, as shown in the split migration example.
- See “Purpose” on page 3894 for the restrictions to the version numbers of the product that are supported.

-t *TargetVersion*

(Optional) The destination version of the component.

- If not specified, this value is assumed to be the current version.
- When you perform split migration from Version 7.0 to a previous version, the **-t** parameter is mandatory.
- See “Purpose” on page 3894 for the restrictions to the version numbers of the product that are supported.

ComponentName

(Required) The name of the component to migrate.

Authorization:
For information about platform-specific authorizations, see the following topics:

- "Security requirements for Linux and UNIX platforms" on page 3648
- "Security requirements for Windows systems" on page 3651
- "Security requirements for z/OS" on page 3655

If you have enabled broker administration security, you must also set up the authority detailed in "Tasks and authorizations for broker administration security" on page 3645.

The **mqsimigratecomponents** command updates your registry, file system, and WebSphere MQ definitions.

If the user ID used to run this command does not have the authority to perform all these steps, you can run the command one part at a time. Different users can run the part for which they are authorized in order to achieve the overall result. This approach is referred to as *split migration*, and is performed by using the **-1**, **-2**, and **-3** parameters.

If you run single-step migration, your user ID must be able to:

- Write to the registry and the file system for the product
- Modify queue definitions
- Read from the database that is associated with the broker

If you run split migration, your user ID must always be able to read from the registry for the product.

If you are doing a backward split migration of the registry, an integrity check to verify the levels of both WebSphere MQ and the database are done that requires your user ID to have

- SELECT authority on the broker database tables
- WebSphere MQ authority to open the `SYSTEM.BROKER.*` queues.

The following specific authorization for each step is required in order to succeed:

- **-1** requires the ability to write to the registry and the file system for the product, SELECT from the broker database tables, and open the `SYSTEM.BROKER.*` queues
- **-2** requires the ability to modify queue definitions
- **-3** requires the ability to read from databases associated with the broker

Responses:
This command can produce many possible responses, depending on the results of the various operations. This command differs from other commands in the way it produces messages: they are displayed when they are generated, rather than being reported in a batch at the end of the program.

When you migrate database tables, z/OS produces more output than distributed systems. Use the **-q** parameter to reduce the number of messages displayed.

Examples:
The following example shows a split migration from Version 6.1 to Version 7.0:

```
mqsimigratecomponents BROKER1 -1
mqsimigratecomponents BROKER1 -2
mqsimigratecomponents BROKER1 -3
```

The following example shows a migration from Version 7.0 back to Version 6.1:

```
mqsimigratecomponents MYBROKER -t 6.1.0.3
```

Related tasks:

Chapter 3, "Migrating and upgrading," on page 137
To migrate a broker domain to WebSphere Message Broker Version 7.0, plan your migration strategy, perform pre-migration tasks, migrate your domain components, and then complete post-migration tasks.

Related reference:

"Migration and upgrade" on page 3579
Consider the factors involved in the migration of components and resources from Version 6.0 or Version 6.1 to Version 7.0.

mqsimode command:

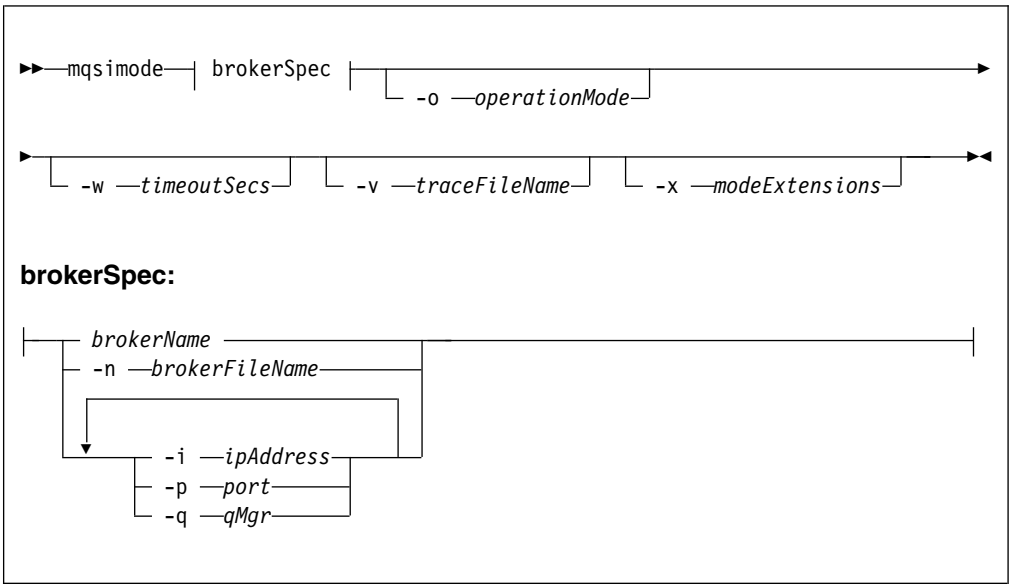
Use the **mqsimode** command to configure and retrieve operation mode information.

Supported platforms:

- Windows
- Linux and UNIX systems
- z/OS. Run this command by customizing and submitting BIPMODE; see "Contents of the broker PDSE" on page 3991

Purpose:
Use the **mqsimode** command to change the operation mode of a broker, or to retrieve information about the mode in which the broker is currently working.

Syntax:



Parameters:

brokerSpec

(Required) You must specify at least one parameter to identify the target broker for this command, in one of the following forms:

brokerName

This parameter identifies the name of a locally defined broker. You cannot use this option if the broker is on a remote computer.

-n *brokerFileName*

This parameter identifies the name of a file that contains the connection details for a local or remote broker.

Use this option if multiple users want to connect to the same broker, or if you are using advanced connection parameters such as SSL.

To create this file, right-click the broker in the WebSphere Message Broker Explorer and select **Export *.broker file**. When prompted, navigate to the location in which you want to save the file and enter the file name; the extension `.broker` is appended automatically when you click **Save**. Include the location (path) and file name when you specify this parameter. You must ensure that the location is accessible when you run this command.

If you want to run a command that uses SSL to administer a remote broker over a secured channel, you must specify the keystore and truststore password for the connection using the IBM_JAVA_OPTIONS environment variable. See "Resolving problems when running commands" on page 3364 for further information.

-i *ipAddress*, **-p** *port*, **-q** *qMgr*

These parameters identify the connection details for the broker.

Use this option for connections to remote brokers that do not require advanced connection parameters.

If you choose this option, you must specify at least one of these three parameters; the order in which you specify them is insignificant. You cannot specify each parameter more than once.

Parameters that you omit assume default values:

- **-i** *ipAddress*: The host name or the IP address of the computer on which the broker is running. If you do not specify this parameter, a value that represents the local computer is used.
- **-p** *port*: The TCP port on which the broker's queue manager is listening. If you do not specify this parameter, the value 1414 is used.
- **-q** *qMgr*: The name of the broker's queue manager. If you do not specify this parameter, a value that represents the default queue manager on the local computer is used.

You cannot connect to a remote queue manager on z/OS; see "Usage note" on page 3901.

-o *operationMode*

(Optional) This parameter sets the mode of the target broker. Valid values are `enterprise` (the full edition), `starter` (Starter Edition), `entry` (Entry Edition), and `adapter` (Remote Adapter Deployment mode). If you do not specify the **-o** parameter, the command displays the mode in which the broker is running.

-x *modeExtensions*

(Optional) This parameter uses a comma separated list of to specify the mode

extensions to which the broker is entitled. You can also use the **-x** parameter to switch off all mode extensions (See "Examples").

-w *timeoutSecs*
(Optional) This parameter specifies the time in seconds that the utility waits to ensure that the command completed; the default value is 60.

-v *traceFileName*
(Optional) This parameter sends internal debug trace information to the specified file.

Usage note:
If you try to connect to a remote broker by specifying the **i**, **p**, and **q** parameters, or by using a connection parameter file (`.broker`), the command attempts to use WebSphere MQ Java client code. This option is not supported on z/OS, and returns the following error:

```
BIP1046E: Unable to connect with the broker (name)
```

The reported reason code is `MQRC_ENVIRONMENT_ERROR`. You must specify a local queue manager.

Authorization:
For information about platform-specific authorizations, see the following topics:

- "Security requirements for Linux and UNIX platforms" on page 3648
- "Security requirements for Windows systems" on page 3651
- "Security requirements for z/OS" on page 3655

If you have enabled broker administration security, you must also set up the authority detailed in "Tasks and authorizations for broker administration security" on page 3645.

Responses:
This command returns the following responses:

0 The command completed successfully.

2 (Failure) The broker received the deployment request but was unable to process it successfully. See the messages issued from the utility (or the Administration log) for more information.

9 (Failure) The request has been submitted to the broker, but no response was received before the timeout expired.

10 (Failure) Another user or application canceled the request operation before the broker was able to process it.

98 The broker is not running.

99 One or more of the parameters that you specified is invalid.

Examples:
Running the `mqsimode` command specifying the `-o` parameter

When you run the **mqsimode** command for broker `BROKER1`, and specify the **-o** parameter, the mode is updated, and you receive a report about all errors. For example, if you run the following **mqsimode** command to change your broker to the Remote Adapter Deployment mode, and your broker is in the following state:

- *Broker_Name1* is the name of your broker.
- *Message_Flow* is the name of your message flow.

- *Execution_Group* is the name of your execution group.
- The command changes *Broker_Name1* to the Remote Adapter Deployment mode, but contains a node *Node_Type* that is not valid in this mode.

```
mqsimode BROKER1 –o adapter
```

You receive the following messages:

```
BIP1044: Connecting to the broker's queue manager...
BIP1045: Connecting to the broker...
BIP1805: The mode for broker 'Broker_Name1' has been changed to 'adapter'.
BIP1823: WARNING: Broker 'Broker_Name1' has a message flow called 'Message_Flow'
 in execution group 'Execution_Group', which contains one or more nodes that are not
 valid in this mode: Node_Type.
BIP8229: The command completed with the following number of warnings: 1.
```

Running the mqsimode command without the -o parameter

When you run the **mqsimode** command without the **-o** parameter, you receive a report about the mode being used by your broker, a report about all mode violations, and a report for any mode extensions that are required and not set. For example, if you run the following **mqsimode** command, and your broker is in the following state:

- *Broker_Name* is the name of your broker.
- Your broker is in Starter Edition mode.
- Your broker has no violations.

```
mqsimode Broker_Name
```

You receive the following messages:

```
BIP1044: Connecting to the broker's queue manager...
BIP1807: Discovering mode information from broker 'Broker_Name'...
BIP1802: Broker 'Broker_Name' is in 'starter' mode.
BIP8071: Successful command completion.
```

Switching off all mode extensions

To switch off all mode extensions, run the **mqsimode** command with the **-x** parameter:

```
mqsimode Broker_Name –x
```

where *Broker_Name* is the name of your broker.

Running the mqsimode command specifying the -o parameter against a broker that is running a version before V6.1

When you run the **-o** parameter against a broker that is running a version before V6.1 you receive a report with an appropriate message. For example, where *Broker_Name* is the name of your broker:

```
BIP1044: Connecting to the broker's queue manager...
BIP1045: Connecting to the broker...
BIP1808: Broker 'Broker_Name' is not at the required software level to
 change the operation mode.
 BIP8229: The command completed with the following number of warnings: 1.
```

Related concepts:

"Operation modes" on page 48
The operation mode that you use for your broker is determined by the license that you purchase.

Related tasks:

"Checking the operation mode of your broker" on page 657
Use the **mqsimode** command to find out the operation mode of your broker.

"Changing the operation mode of your broker" on page 655
Change the operation mode in which your broker is working by using the **mqsimode** command.

"Enabling broker administration security" on page 368
Enable broker administration security on a broker to control which users can complete specific tasks against that broker and its resources.

"Example: Changing the operation mode of your broker" on page 3904
You want to convert from the Starter Edition (`starter` mode) to the full package (`enterprise` mode).

"Example: Changing the Trial Edition to the full edition"
You want to convert all of your brokers from the Trial Edition mode to the full edition (`enterprise` mode).

Related reference:

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

Example: Changing the Trial Edition to the full edition:

You want to convert all of your brokers from the Trial Edition mode to the full edition (`enterprise` mode).

About this task

Contact your IBM representative to upgrade your license, and change the brokers that you have created to conform to your new license.

Before, or after, the Trial Edition expires complete the following steps:

Procedure

1. Open a command prompt.
   - Linux On Linux and UNIX, run the **mqsiprofile** command to initialize the command environment.
   - Windows On Windows, click **Start** > **Programs** > **IBM WebSphere Message Broker 7.0** > **Command Console** to open a command console.
2. Run the following **mqsimode** command for every broker:

   ```
   mqsimode –i localhost –p 2414 –q MB7QMGR –o enterprise
   ```

   When this command completes successfully, the broker is fully functional, but all new brokers are still created in `trial` mode by default.
3. To ensure that all new brokers start in `enterprise` mode, uninstall the trial package and reinstall the full package; see "Moving from Trial Edition" on page 656.

Related concepts:

"Operation modes" on page 48
The operation mode that you use for your broker is determined by the license that you purchase.

Related tasks:

"Checking the operation mode of your broker" on page 657
Use the **mqsimode** command to find out the operation mode of your broker.

"Changing the operation mode of your broker" on page 655
Change the operation mode in which your broker is working by using the **mqsimode** command.

"Moving from Trial Edition" on page 656
You want to convert from Trial Edition mode to an alternative edition.

"Setting up a command environment" on page 213
After you have installed the product on one of the distributed systems, you must initialize the environment before you can use a runtime component or command.

Related reference:

"Restrictions that apply in each operation mode" on page 3657
The operation mode in which your broker is working defines how many execution groups you can use, and which nodes are available.

"**mqsimode** command" on page 3899
Use the **mqsimode** command to configure and retrieve operation mode information.

Example: Changing the operation mode of your broker:

You want to convert from the Starter Edition (`starter` mode) to the full package (`enterprise` mode).

About this task

Contact your IBM representative to upgrade your license, and change the brokers that you have created to conform to your new license.

Procedure

1. Open a command prompt.
   - Linux On Linux and UNIX, run the **mqsiprofile** command to initialize the command environment.
   - Windows On Windows, click **Start** > **Programs** > **IBM WebSphere Message Broker 7.0** > **Command Console** to open a command console.
2. Run the following **mqsimode** command to change the mode of your broker from `starter` to `enterprise` (where *Broker_Name* is the name of your broker):

   ```
   mqsimode –i localhost –p 1414 –q MB7QMGR –o enterprise
   ```

   The following messages are displayed:

   ```
   BIP1044: Connecting to the broker's queue manager...
   BIP1809: Deploying 'enterprise' mode on broker 'Broker_Name'...
   BIP1805: The mode for broker 'Broker_Name' has been changed to 'enterprise'.
   ```

Related concepts:

"Operation modes" on page 48
The operation mode that you use for your broker is determined by the license that you purchase.

Related tasks:

"Checking the operation mode of your broker" on page 657
Use the **mqsimode** command to find out the operation mode of your broker.

"Changing the operation mode of your broker" on page 655
Change the operation mode in which your broker is working by using the **mqsimode** command.

"Setting up a command environment" on page 213
After you have installed the product on one of the distributed systems, you must initialize the environment before you can use a runtime component or command.

Related reference:

"Restrictions that apply in each operation mode" on page 3657
The operation mode in which your broker is working defines how many execution groups you can use, and which nodes are available.

"**mqsimode** command" on page 3899
Use the **mqsimode** command to configure and retrieve operation mode information.

mqsireadlog command:

Use the **mqsireadlog** command to retrieve trace records for the specified component.

Supported platforms:

- Windows
- Linux and UNIX systems
- z/OS. Run this command by customizing and submitting BIPRELG; see "Contents of the broker PDSE" on page 3991

Purpose:
The **mqsireadlog** command is valid for:

User trace
Specify the **-u** option.

Service trace
Specify the **-t** option. You are recommended to use this option only if directed to do so by the action described in a BIPxxxx message, or by your IBM Support Center.

You can specify the output to be directed to file, or to STDOUT. The trace records returned by this command are in XML format and can be browsed with an XML browser. If you direct output to file, the data is written in code page `utf-8`. The file is therefore platform-independent, and can be transferred to other systems for browsing or formatting using the **mqsiformatlog** command.

On HP-UX, set the `size` parameter of the **mqsichangetrace** command to be less than 500 MB because the size of the XML generated files is often half as much again as the original trace file, and setting the value of the `size` parameter to be greater than 500 MB can cause problems.

If you transfer this file to another system, ensure that you use a file transfer program that does not convert the data (for example, by specifying a binary transfer option).

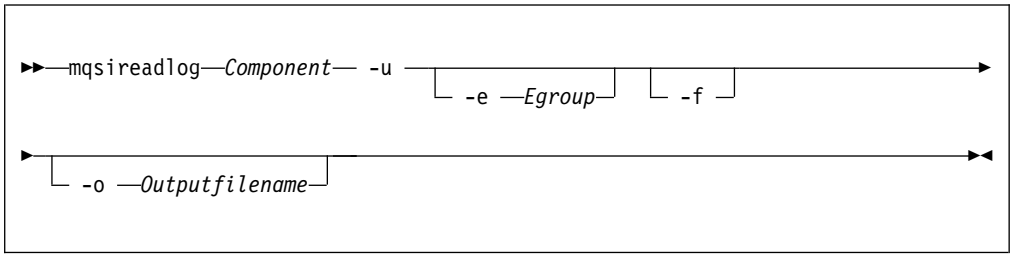
If you specify a broker, or any of its resources (execution group or message flow) you must have deployed them before you can start trace and read the log files.

To enable service trace of your CMP applications, take one of the following steps:

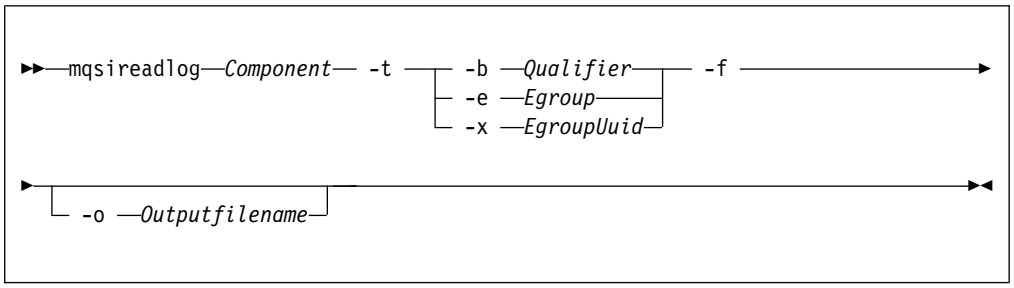
- Call the method BrokerProxy.enableAdministrationAPITracing(String filename).
- Before running your CMP application, set the environment variable MQSI_CMP_TRACE to the name of the file to which trace is sent.

Syntax:

User trace:



Service trace:



Parameters common to user trace and service trace:

Component
(Required) The name of the component for which the log is to be read. The component can be either a broker name, or the fixed values, `WebSphere Message Broker Toolkit`, or `utility` (all are case sensitive on Linux and UNIX systems, and on z/OS).

-e *Egroup*
(Optional) The label of the execution group for which log information is to be read.

-o *Outputfilename*
(Optional) The name of the file into which to write the log data. If you specify a full path name, the file is created in the directory specified. If you specify just the filename, the file is created in the current working directory. The contents of the file are written in code page `utf-8`, which is platform-independent and preserves data such as DBCS characters.

You must specify a file name if you want to format the log by using the **mqsiformatlog** command. If you do not specify a filename, the contents of the log are written to stdout. Use a file extension of `.xml`, which represents the format of the data.

-f (Optional for User trace; required for Service trace). Read the log file directly from the file system. If you do not specify this option, the command sends an XML message to the component to request the log contents. If you have specified **-t** (service trace), you must specify this flag as well.

If you specify this option, stop tracing (by using **mqsichangetrace**) before you run the **mqsireadlog** command. If the log file is in use when you issue this command with this flag specified, partial XML records might be returned. Specify **-m safe** on the **mqsichangetrace** command to reduce the risk of partial

records. If the component being traced has itself stopped, you do not then need to issue a **mqsichangetrace** command.

If you do not stop tracing before you issue this command, check the contents of the log file created, and remove any partial records from the end by using a text editor before running the **mqsiformatlog** command, as partial records cannot be read by the format command.

Additional parameter exclusive to user trace:

-u (Required) Read the log contents from the user trace log. This option is valid **only** if you select the `broker` component.

Additional parameters exclusive to service trace:
Use these options only when directed to do so by your IBM Support Center or by a BIPxxxx message.

-t (Required) Read the log contents from the service trace log.

-b *Qualifier*
(Required) Read the contents of the log for the broker `agent`, or for the specified command utility program. This option is valid only if you have specified **-t** (service trace).

The following table shows the valid combinations of `qualifier` and `component` for service trace.

This option is generally used to trace the commands themselves. If you want to trace a particular command, run that command with environment variable MQSI_UTILITY_TRACE set to `debug` or `normal` before you issue this command to read the trace output generated.

Enter these values exactly as shown.

The `agent` trace is initiated when you specify the **-b** flag on the **mqsichangetrace** command. Do this only when directed to do so by a WebSphere Message Broker error message, or when instructed to do so by your IBM Support Center.

The `service` trace is initiated when you specify the **-b** flag on the **mqsichangetrace** command. The format of the command is:

```
mqsireadlog brokername -t -b service -f -o service.xml
```

Run this command only when directed to do so by a WebSphere Message Broker error message, or when instructed to do so by your IBM Support Center.

-x*EgroupUuid*
(Optional) Read the contents of the log for the execution group UUID (Universally Unique Identifier).

This option is valid only if you have specified **-t** (service trace).

Use this option when the execution group label is not available. The execution group UUID can be obtained from the BIP2201I or BIP2208I message that is written to the system log when the execution group starts up.

The format of the command is:

```
mqsireadlog brokername -t
    -x ce40b73e-2701-0000-0080-85557ff4a8ad -f -o service.xml
```

Qualifier	Component= *broker_name*	Component= WebSphere Message Broker Toolkit	Component= utility
mqsichangebroker	x		
mqsichangeflowmonitoring	x		
mqsichangeflowstats	x		
mqsichangeflowuserexits	x		
mqsichangeproperties	x		
mqsichangetrace	x		
mqsicreatebroker	x		
mqsicreateconfigurableservice	x		
mqsicvp	x		
mqsideletebroker	x		
mqsideleteconfigurableservice	x		
mqsideploy	x		
mqsiformatlog[1]			x
mqsimigratecomponents	x		
mqsireadlog	x		x
mqsireload	x		
mqsireloadsecurity	x		
mqsireportbroker	x		
mqsireportflowmonitoring	x		
mqsireportflowstats	x		
mqsireportflowuserexits	x		
mqsireportproperties	x		
mqsireporttrace	x		
mqsisetdbparms	x		
mqsistart	x		
mqsistop	x		
agent	x		
service	x		
WebSphere Message Broker Toolkit		x	
httplistener	x		

Notes:

1. Because this command does not have a component parameter, trace information is recorded in, and retrieved from, the *utility* component trace files. For further details see the **mqsichangetrace** command.

Authorization:
For information about platform-specific authorizations, see the following topics:

- "Security requirements for Linux and UNIX platforms" on page 3648
- "Security requirements for Windows systems" on page 3651

- "Security requirements for z/OS" on page 3655

If you have enabled broker administration security, you must also set up the authority detailed in "Tasks and authorizations for broker administration security" on page 3645.

Examples:
Retrieve the user trace for broker MB7BROKER:

```
mqsireadlog MB7BROKER -u -e default -o trace.xml
```

Retrieve service trace for utility **mqsiformatlog**:

```
mqsireadlog utility -t -b agent -f -o trace.xml
```

You can format the log file (`trace.xml` in the above examples) by using the command **mqsiformatlog**, or view it using an XML editor or viewer.

Related tasks:

"Using logs" on page 3526
There are a variety of logs that you can use to help with problem determination and troubleshooting.

Related reference:

"Syntax diagrams" on page 3677

"**mqsichangetrace** command" on page 3822
Use the **mqsichangetrace** command to set the tracing characteristics for a broker.

"**mqsiformatlog** command" on page 3880
Use the **mqsiformatlog** command to process the XML log created by **mqsireadlog**. The command retrieves and formats any messages that the XML log contains into a form suitable for the locale of the user who runs the command.

"**mqsireporttrace** command" on page 3947
Use the **mqsireporttrace** command to display the trace options currently in effect. Trace can be run only against a broker, or any of its resources.

mqsireload command:

Use the **mqsireload** command to request the broker to stop and restart execution groups.

Supported platforms:

- Windows
- Linux and UNIX systems
- z/OS - as a console command

Purpose:
When you issue this command, a message is sent to the broker, which stops and restarts all its execution groups. You can specify a single execution group to be reloaded, but if you use the default form of this command to reload all execution groups you ensure state and data integrity is preserved.

Because an execution group does not stop until all message flows within it terminates, the ability of the broker to reload quickly depends on the processing time for the longest running message flow. This affects the performance of this command, therefore review any long-running message flows before running this command.

If you have included a user-defined node or parser within a message flow on the broker, these are deleted by this command, and the relevant termination functions called. When message flows are restarted, the resources used by user-defined nodes and parsers are re-accessed and reacquired. However, it is better programming practice to ensure that user-defined nodes and parsers provide their own mechanism to reload persistent state and data dynamically, and do not rely on the use of this command.

Select the appropriate link for details of this command on the platform, or platforms, that your enterprise uses:

- "mqsireload command - Windows, Linux, and UNIX systems"
- "mqsireload command - z/OS" on page 3911

Authorization:
For information about platform-specific authorizations, see the following topics:

- "Security requirements for Linux and UNIX platforms" on page 3648
- "Security requirements for Windows systems" on page 3651
- "Security requirements for z/OS" on page 3655

If you have enabled broker administration security, you must also set up the authority detailed in "Tasks and authorizations for broker administration security" on page 3645.

Responses:
No additional responses are returned.

Related tasks:

"Deleting a broker" on page 930
Delete a broker using the command line on the system where the broker component is installed.

Related reference:

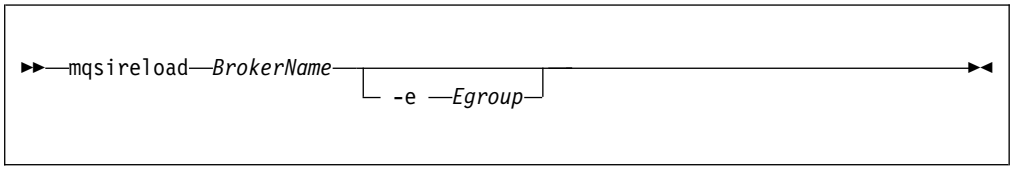
"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

mqsireload command - Windows, Linux, and UNIX systems:

Use the **mqsireload** command on Windows, Linux, and UNIX systems to request the broker to stop and restart execution groups.

Syntax:



Parameters:

BrokerName
(Required) The name of the broker to which the reload request is sent.

-e *Egroup*
(Optional) The name of the execution group that is to be reloaded. If this parameter is not specified, all execution groups on the specified broker are stopped and restarted.

Examples:

```
mqsireload broker1
```

Related reference:

"Syntax diagrams" on page 3677

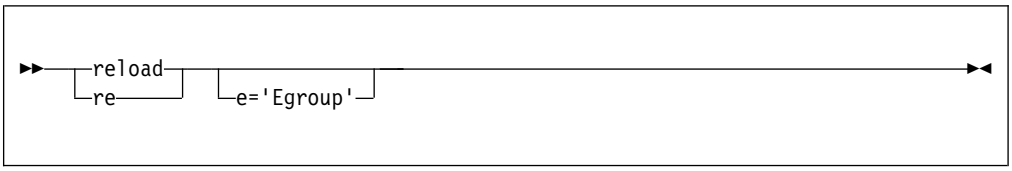
"**mqsireload** command" on page 3909
Use the **mqsireload** command to request the broker to stop and restart execution groups.

mqsireload command - z/OS:

Use the **mqsireload** command on z/OS to request the broker to stop and restart execution groups.

Syntax:

z/OS console command:
Synonym: re



Parameters:

e= *Egroup*
(Optional) The name of the execution group that is to be reloaded. If this parameter is not specified, all execution groups on the specified broker are stopped and restarted.

Examples:

```
F MQP1BRK,re e='EgName'
```

Related reference:

"Syntax diagrams" on page 3677

"**mqsireload** command" on page 3909
Use the **mqsireload** command to request the broker to stop and restart execution groups.

mqsireloadsecurity command:

Use the **mqsireloadsecurity** command to force the immediate expiry of some or all the entries in the security cache.

Supported platforms:

- Windows
- Linux and UNIX systems

- z/OS. Run this command in one of two ways - as a console command, or by customizing and submitting BIPRLSEC; see "Contents of the broker PDSE" on page 3991

Purpose:
Entries in the security cache are valid for a specified length of time, after which the entries are marked as 'expired'. When an entry is marked as expired, it must be reauthenticated with the security provider before it can be reused, and its expiry time must be reset. If reauthentication fails, the entry remains marked as expired. All entries in the security cache marked as expired are removed when the next sweep of the cache is performed.

Use the **mqsichangeproperties** command to set the time for which entries in the cache are valid, and also the value for the sweep of the security cache. When the entries in the security cache have expired, you must reauthenticate them.

Select the appropriate link for details of this command on the operating system that is used by your enterprise:

- "**mqsireloadsecurity** command - Windows, Linux, and UNIX systems" on page 3913
- "**mqsireloadsecurity** command - z/OS" on page 3915

Authorization:
For information about platform-specific authorizations, see the following topics:

- "Security requirements for Linux and UNIX platforms" on page 3648
- "Security requirements for Windows systems" on page 3651
- "Security requirements for z/OS" on page 3655

If you have enabled broker administration security, you must also set up the authority detailed in "Tasks and authorizations for broker administration security" on page 3645.

Responses:
This command returns the following responses:

0 The command completed successfully.

2 (Failure) The broker received the deployment request but was unable to process it successfully. See the messages issued from the utility (or the Administration log) for more information.

9 (Failure) The request has been submitted to the broker, but no response was received before the timeout expired.

10 (Failure) Another user or application canceled the request operation before the broker was able to process it.

98 The broker is not running.

99 One or more of the parameters that you specified is invalid.

Related concepts:

Chapter 5, "Security," on page 351
Security is an important consideration for both developers of WebSphere Message Broker applications, and for system administrators configuring WebSphere Message Broker authorities.

"Message flow security overview" on page 383
WebSphere Message Broker provides a security manager, which enables you to control access to individual messages in a message flow, using the identity of the

message.

Related tasks:

"Enabling broker administration security" on page 368
Enable broker administration security on a broker to control which users can complete specific tasks against that broker and its resources.

"Diagnosing security problems" on page 496
This topic explains how to find out why access to a secured flow has been denied.

Related reference:

"Parameter values for the securitycache component" on page 3815
Select the objects and properties associated with the securitycache component that you want to change.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsireloadsecurity** command - Windows, Linux, and UNIX systems"
Use the **mqsireloadsecurity** command on Windows, Linux, and UNIX to force the immediate expiry of some or all of the entries in the security cache.

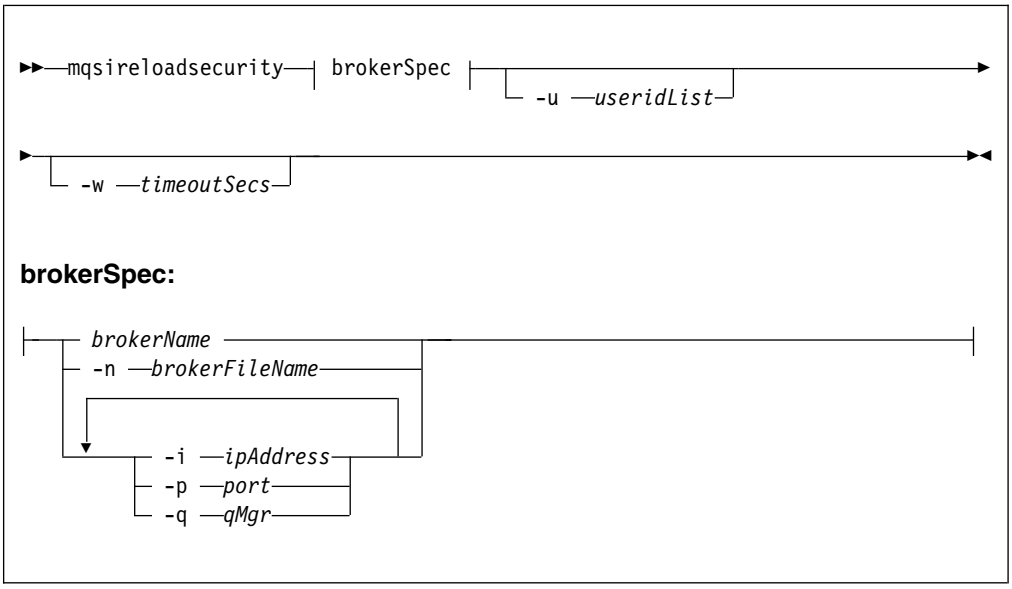
"**mqsireloadsecurity** command - z/OS" on page 3915
Use the **mqsireloadsecurity** command on z/OS to force the immediate expiry of some or all the entries in the security cache.

"Syntax diagrams" on page 3677

mqsireloadsecurity *command - Windows, Linux, and UNIX systems:*

Use the **mqsireloadsecurity** command on Windows, Linux, and UNIX to force the immediate expiry of some or all of the entries in the security cache.

Syntax:



Parameters:

brokerSpec
(Required) You must specify at least one parameter to identify the target broker for this command, in one of the following forms:

brokerName
This parameter identifies the name of a locally defined broker. You cannot use this option if the broker is on a remote computer.

-n *brokerFileName*
This parameter identifies the name of a file that contains the connection details for a local or remote broker.

Use this option if multiple users want to connect to the same broker, or if you are using advanced connection parameters such as SSL.

To create this file, right-click the broker in the WebSphere Message Broker Explorer and select **Export *.broker file**. When prompted, navigate to the location in which you want to save the file and enter the file name; the extension `.broker` is appended automatically when you click **Save**. Include the location (path) and file name when you specify this parameter. You must ensure that the location is accessible when you run this command.

If you want to run a command that uses SSL to administer a remote broker over a secured channel, you must specify the keystore and truststore password for the connection using the IBM_JAVA_OPTIONS environment variable. See "Resolving problems when running commands" on page 3364 for further information.

-i *ipAddress*, **-p** *port*, **-q** *qMgr*
These parameters identify the connection details for the broker.

Use this option for connections to remote brokers that do not require advanced connection parameters.

If you choose this option, you must specify at least one of these three parameters; the order in which you specify them is insignificant. You cannot specify each parameter more than once.

Parameters that you omit assume default values:

- **-i** *ipAddress*: The host name or the IP address of the computer on which the broker is running. If you do not specify this parameter, a value that represents the local computer is used.
- **-p** *port*: The TCP port on which the broker's queue manager is listening. If you do not specify this parameter, the value 1414 is used.
- **-q** *qMgr*: The name of the broker's queue manager. If you do not specify this parameter, a value that represents the default queue manager on the local computer is used.

-u *useridList*
(Optional) This parameter reloads all entries in the security cache for the specified list of users (separated by colons). If you do not specify this parameter, all entries in the security cache are reloaded. For cached certificates, this value is the common name element value.

-w *timeoutSecs*
(Optional) This parameter specifies the time in seconds that the utility waits to ensure that the command completed; the default value is 60.

Examples:
Reload the cache for all users on the specified broker. :

```
mqsireloadsecurity BROKER1
```

Reload the cache for a single user on the specified broker. The connection parameters for the broker are defined in the file BROKER1.broker.

```
mqsireloadsecurity -n BROKER1.broker -u user1
```

Reload the cache for a list of users on the specified broker, and wait for five seconds before returning:

```
mqsireloadsecurity My_Broker -u user1:user2:user3 -w 5
```

Related concepts:

"Message flow security overview" on page 383
WebSphere Message Broker provides a security manager, which enables you to control access to individual messages in a message flow, using the identity of the message.

Chapter 5, "Security," on page 351
Security is an important consideration for both developers of WebSphere Message Broker applications, and for system administrators configuring WebSphere Message Broker authorities.

"Security overview" on page 351
When you are designing a WebSphere Message Broker application it is important to consider the security measures that are needed to protect the information in the system.

Related tasks:

"Diagnosing security problems" on page 496
This topic explains how to find out why access to a secured flow has been denied.

Related reference:

"**mqsireloadsecurity** command" on page 3911
Use the **mqsireloadsecurity** command to force the immediate expiry of some or all the entries in the security cache.

"Syntax diagrams" on page 3677

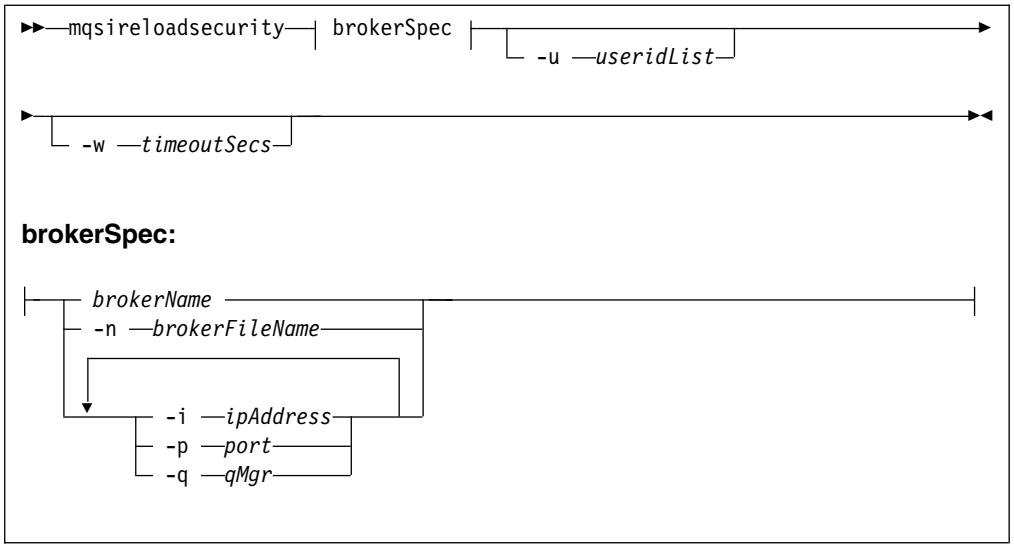
"Parameter values for the securitycache component" on page 3815
Select the objects and properties associated with the securitycache component that you want to change.

mqsireloadsecurity *command - z/OS:*

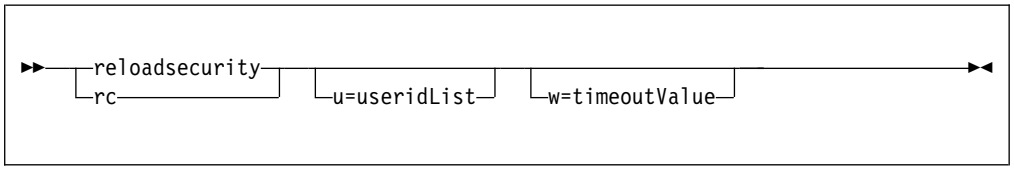
Use the **mqsireloadsecurity** command on z/OS to force the immediate expiry of some or all the entries in the security cache.

Syntax:

z/OS command - BIPRLSEC:



z/OS console command:



Parameters:

brokerSpec
You must specify at least one parameter to identify the target broker for this command, in one of the following forms:

brokerName
This parameter identifies the name of a locally defined broker.

-n *brokerFileName*
This parameter identifies the name of a file that contains the connection details for a local broker.

Use this option if multiple users want to connect to the same broker, or if you are using advanced connection parameters such as SSL.

To create this file, right-click the broker in the WebSphere Message Broker Explorer and select **Export *.broker file**. When prompted, navigate to the location in which you want to save the file and enter the file name; the extension `.broker` is appended automatically when you click **Save**. Include the location (path) and file name when you specify this parameter. You must ensure that the location is accessible when you run this command.

-i *ipAddress*, **-p** *port*, **-q** *qMgr*
These parameters identify the connection details for the broker. If you specify a queue manager, it must be defined on the local computer.

If you choose this option, you must specify at least one of these three parameters; the order in which you specify them is irrelevant. You cannot specify each parameter more than once.

Parameters that you omit assume default values:

- **-i** *ipAddress*: The host name or the IP address of the computer on which the broker is running. If you do not specify this parameter, a value that represents the local computer is used.
- **-p** *port*: The TCP port on which the broker's queue manager is listening. If you do not specify this parameter, the value 1414 is used.
- **-q** *qMgr*: The name of the broker's queue manager. If you do not specify this parameter, a value that represents the default queue manager on the local computer is used.

You cannot connect to a remote queue manager on z/OS; see "Usage note."

-u *useridList*
(Optional) This parameter reloads all entries in the security cache for the specified list of users (separated by colons). If you do not specify this parameter, all entries in the security cache are reloaded. For cached certificates, this value is the common name element value.

-w *timeoutValue*
(Optional) This parameter specifies the time in seconds that the utility waits to ensure that the command completed; the default value is 60.

Usage note:
If you try to connect to a remote broker by specifying the **i**, **p**, and **q** parameters, or by using a connection parameter file (`.broker`), the command attempts to use WebSphere MQ Java client code. This option is not supported on z/OS, and returns the following error:

```
BIP1046E: Unable to connect with the broker (name)
```

The reported reason code is `MQRC_ENVIRONMENT_ERROR`. You must specify a local queue manager.

Examples:
Reload the cache for a single user on the specified broker. The connection parameters for the broker are defined in the file `BROKER1.broker`.

```
mqsireloadsecurity -n BROKER1.broker -u user1
```

Reload the cache for all users by using the console command:

```
F MQP1BRK,rc
```

Reload the cache for a single user:

```
F MQP1BRK,rc u=user1
```

Reload the cache for a list of users, and wait for 5 seconds before returning:

```
F MQP1BRK,rc u=user1:user2:user3,w=5
```

Related concepts:

"Message flow security overview" on page 383
WebSphere Message Broker provides a security manager, which enables you to control access to individual messages in a message flow, using the identity of the message.

Chapter 5, "Security," on page 351
Security is an important consideration for both developers of WebSphere Message Broker applications, and for system administrators configuring WebSphere Message Broker authorities.

"Security overview" on page 351
When you are designing a WebSphere Message Broker application it is important to consider the security measures that are needed to protect the information in the system.

Related tasks:

"Diagnosing security problems" on page 496
This topic explains how to find out why access to a secured flow has been denied.

Related reference:

"Syntax diagrams" on page 3677

"**mqsireloadsecurity** command" on page 3911
Use the **mqsireloadsecurity** command to force the immediate expiry of some or all the entries in the security cache.

"Parameter values for the securitycache component" on page 3815
Select the objects and properties associated with the securitycache component that you want to change.

mqsiremovebrokerinstance command:

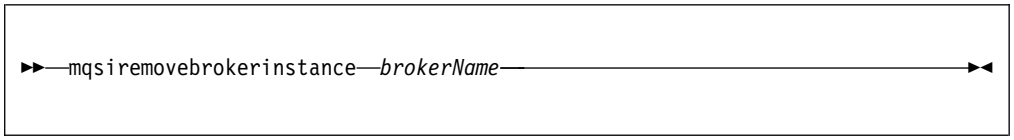
Use the **mqsiremovebrokerinstance** command to remove a multi-instance broker from a server where WebSphere Message Broker has been installed.

Supported platforms:

- Windows
- Linux and UNIX systems

Purpose:
Use the **mqsiremovebrokerinstance** command to remove a broker instance from any additional server on which you no longer require multi-instance support. You must first create a multi-instance enabled broker on one server using the **mqsicreatebroker** command.

Syntax:



Parameters:

brokerName
(Required) The name of the broker instance that you are deleting; the name is case-sensitive. The broker instance name must match that of a multi-instance enabled broker previously created by using the **mqsicreatebroker** command.

Authorization:
For information about platform-specific authorizations, see the following topics:

- "Security requirements for Linux and UNIX platforms" on page 3648
- "Security requirements for Windows systems" on page 3651

If you have enabled broker administration security, you must also set up the authority detailed in "Tasks and authorizations for broker administration security" on page 3645.

Examples:
The following example removes a broker instance from broker `Mybroker`:

```
mqsiremovebrokerinstance MyBroker
```

Related reference:

"Syntax diagrams" on page 3677

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsideletebroker** command" on page 3863
Use the **mqsideletebroker** command to delete a named broker. The command also deletes the queues on the associated queue manager (created when the broker was created). You can also specify that the queue manager is to be deleted.

"**mqsiaddbrokerinstance** command" on page 3715
Use the **mqsiaddbrokerinstance** command to create a multi-instance broker on a server where WebSphere Message Broker has been installed.

mqsireportbroker command:

Use the **mqsireportbroker** command to display broker registry entries.

Supported platforms:

- Windows
- Linux and UNIX systems
- z/OS. Run this command by customizing and submitting BIPRPBK; see "Contents of the broker PDSE" on page 3991

Purpose:
You can use the **mqsireportbroker** command to view the property values set when you create or change the broker.

On Windows platforms, Linux, and UNIX systems, the **mqsireportbroker** command also reports whether a broker is configured as a WebSphere MQ service.

Select the appropriate link for details of this command on the platform, or platforms, that your enterprise uses:

- "**mqsireportbroker** command - Windows, Linux, and UNIX systems" on page 3921
- "**mqsireportbroker** command - z/OS" on page 3923

Usage notes:
The following rules apply on the report:

- You can run this command if the broker is stopped or running.
- If values for a password are set, eight asterisks are displayed.
- Units are shown in the output for all numeric values.

Responses:
The following table shows the output returned when you run the **mqsireportbroker** command.

Property (Units)	Notes
brokerName	1
Service userId	2, 3, 4

Property (Units)	Notes
Service password	2, 3, 4
Queue manager	1
Work path	1
Trusted (fastpath) queue manager application - true	2, 3
Trusted (fastpath) queue manager application - false	3, 5
User LIL path	2
Configuration change timeout (seconds)	2
Internal configuration timeout (seconds)	2
Administration security enabled (true or false)	2
Statistics major interval (minutes)	2
LDAP principal	2
LDAP credentials	2
ICU converter path	2
User exit path	2
Operation mode	2, 3
Fix pack capability level	5
Active user exits	5
Broker UUID	7
Install path	8
Process ID	7

Notes:

1. This value is set by the **mqsicreatebroker** command.
2. This value is set by the **mqsicreatebroker** command or later modified by the **mqsichangebroker** command.
3. This option is applicable only on distributed systems.
4. This option is not relevant on Windows, because this value is stored in the service.
5. This option can be set only by the **mqsichangebroker** command.
6. (Not used.)
7. This value is generated automatically when the broker is started.
8. You cannot set this value by using a command.

Authorization:
For information about platform-specific authorizations, see the following topics:

- "Security requirements for Linux and UNIX platforms" on page 3648
- "Security requirements for Windows systems" on page 3651
- "Security requirements for z/OS" on page 3655

If you have enabled broker administration security, you must also set up the authority detailed in "Tasks and authorizations for broker administration security" on page 3645.

Related concepts:

"User exits" on page 3015
A user exit is user-provided custom software, written in C, to track data passing through message flows.

Related tasks:

Chapter 8, "Administering brokers and broker resources," on page 899
Administering brokers and associated broker resources includes the tasks that you perform frequently to activate and manage those resources. Choose the method you prefer to administer your brokers and associated resources.

"Generating a new code page converter" on page 824
Generate a code page converter to handle conversions of data that belongs to a code page that is not in the default set of code pages provided by WebSphere Message Broker.

"Developing a user exit" on page 3114
Develop a user exit by declaring it, implementing its behavior, then compiling it.

"Deploying a user exit" on page 3116
Deploy your user exit to the broker.

Related reference:

"Security requirements for administrative tasks" on page 3644
You can configure access permissions to govern which users and groups can manipulate objects in the broker network. Security requirements for administrative tasks depend on the platform that you use.

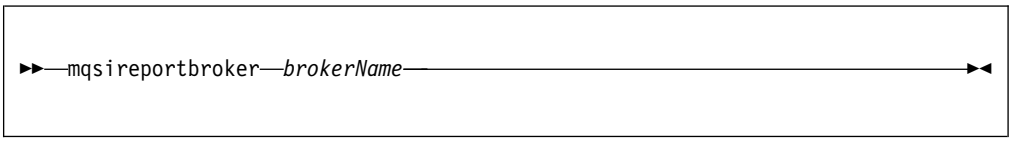
"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

mqsireportbroker *command - Windows, Linux, and UNIX systems:*

Use the **mqsireportbroker** command to display broker registry entries.

Syntax:



Parameters:

brokerName
(Required) The label of the broker for which registry entries are to be reported.

Examples: Windows
Display registry entries of the specified broker on Windows:

```
mqsireportbroker SOAPBR
```

The following output is returned:

```
BIP8927I: Broker Name 'SOAPBR'
Install path = 'C:\Program Files\IBM\MQSI\8.0.0.0'
Work path = 'C:\Documents and Settings\All Users\Application Data\IBM\MQSI'
Broker UUID = '91c31c2a-1a01-0000-0080-d9cfce7fc2e8'
Process id = '6928'
```

```
Queue manager = 'SOAPQM'
User lil path = 'C:\Documents and Settings\All Users\Application Data\lilPath'
User exit path = 'C:\Documents and Settings\All Users\Application Data\userExits'
Active user exits = 'activeUE'
LDAP principal = 'ldapuser'
LDAP credentials = '********'
ICU converter path = 'C:\Documents and Settings\All Users\Application Data\converters'
Trusted (fathpath) queue manager application = 'true'
Configuration change timeout = '320' seconds
Internal configuration timeout = '70' seconds
Statistics major interval = '61' minutes
Operation mode = 'enterprise'
Fixpack capability level = 'all' (effective level '8.0.0.0')
Broker registry format = 'v8.0'
Administration security = 'inactive'
Multi-instance Broker = 'true'
Shared Work Path = ' \\?\UNC\Server1\HAShared'
Start As MQ Service = 'defined'
HTTP listener port = '24'
```

Linux UNIX

Display registry entries of the specified broker on Linux:

```
mqsireportbroker SOAPBR
```

The following output is returned:

```
BIP8926I: Broker Name 'SOAPBR'
Install path = 'usr/lpp/ibm/mqsi/8.0.0.0/'
Work path = '/var/mqsi'
Broker UUID = '91c31c2a-1a01-0000-0080-d9cfce7fc2e8'
Process id = '6928'
Queue manager = 'SOAPQM'
User lil path = '/usr/lpp/ibm/mqsi/8.0.0.0/userlilpath'
User exit path = '/usr/lpp/ibm/mqsi/8.0.0.0/userexitpath'
Active user exits = 'activeUE'
LDAP principal = 'ldapuser'
LDAP credentials = '********'
ICU converter path = '/usr/lpp/ibm/mqsi/8.0.0.0/converterPath'
Trusted (fathpath) queue manager application = 'true'
Configuration change timeout = '320' seconds
Internal configuration timeout = '70' seconds
Statistics major interval = '61' minutes
Operation mode = 'enterprise'
Fixpack capability level = 'all' (effective level '8.0.0.0')
Broker registry format = 'v8.0'
Administration security = 'inactive'
Multi-instance Broker = 'true'
Shared Work Path = ' //?/UNC/Server1/HAShared'
Start As MQ Service = 'defined'
HTTP listener port = '24'
```

Related tasks:

"Modifying a broker on Windows, Linux, and UNIX systems" on page 632
Use the **mqsichangebroker** command on Windows, Linux, and UNIX to modify your broker.

Related reference:

"Syntax diagrams" on page 3677

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

mqsireportbroker *command - z/OS:*

Use the **mqsireportbroker** command to display broker registry entries.

Syntax:

Parameters:

brokerName
(Required) The label of the broker for which registry entries are to be reported.

Examples: z/OS
Display registry entries of the specified broker on z/OS:

```
mqsireportbroker MQ81BRK
```

The following output is returned:

```
BIP8928I: Broker Name 'MQ81BRK'
Install path = '/usr/lpp/ibm/mqsi/7.0.0.0/'
Work path = '/u/wmqi81/broker'
Broker UUID = '91c31c2a-1a01-0000-0080-d9cfce7fc2e8'
Process id = '6928'
Queue manager = 'MQ81'
User lil path = '/u/wmqi81/broker/userlilpath'
User lil path64 = '/u/wmqi81/broker/userlilpath64'
User exit path = '/u/wmqi81/broker/userexitpath'
User exit path64 = '/u/wmqi81/broker/userexitpath64'
Active user exits = 'activeUE'
LDAP principal = 'wmqi81'
LDAP credentials = '********'
ICU converter path = '/u/wmqi81/broker/converterPath'
Trusted (fathpath) queue manager application = 'true'
Configuration change timeout = '320' seconds
Internal configuration timeout = '70' seconds
Statistics major interval = '61' minutes
Operation mode = 'enterprise'
Fixpack capability level = 'all' (effective level '7.0.0.0')
Broker registry format = 'v7.0'
```

Related tasks:

"Modifying a broker on z/OS" on page 634
Use the **mqsichangebroker** command on z/OS to modify your broker.

Related reference:

"Syntax diagrams" on page 3677

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

mqsireportflowmonitoring command:

Use the **mqsireportflowmonitoring** command to display the current options for monitoring that have been set using the **mqsichangeflowmonitoring** command.

Supported platforms:

- Windows
- Linux and UNIX systems
- z/OS: Run this command either as a console command, or by customizing and submitting BIPRPME; see "Contents of the broker PDSE" on page 3991.

Purpose:
Select the appropriate link for details of this command on the platform, or platforms, that your enterprise uses:

- "mqsireportflowmonitoring command - Windows, Linux and UNIX systems"
- "mqsireportflowmonitoring command - z/OS" on page 3927

Authorization:
For information about platform-specific authorizations, see the following topics:

- "Security requirements for Linux and UNIX platforms" on page 3648
- "Security requirements for Windows systems" on page 3651
- "Security requirements for z/OS" on page 3655

If you have enabled broker administration security, you must also set up the authority detailed in "Tasks and authorizations for broker administration security" on page 3645.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Business-level monitoring" on page 3319
You can configure your message flow to emit event messages that can be used to support transaction monitoring and auditing, and business process monitoring.

"Configuring monitoring event sources using a monitoring profile" on page 762
You can create a monitoring profile and use the **mqsichangeflowmonitoring** command to configure your message flows to emit monitoring events.

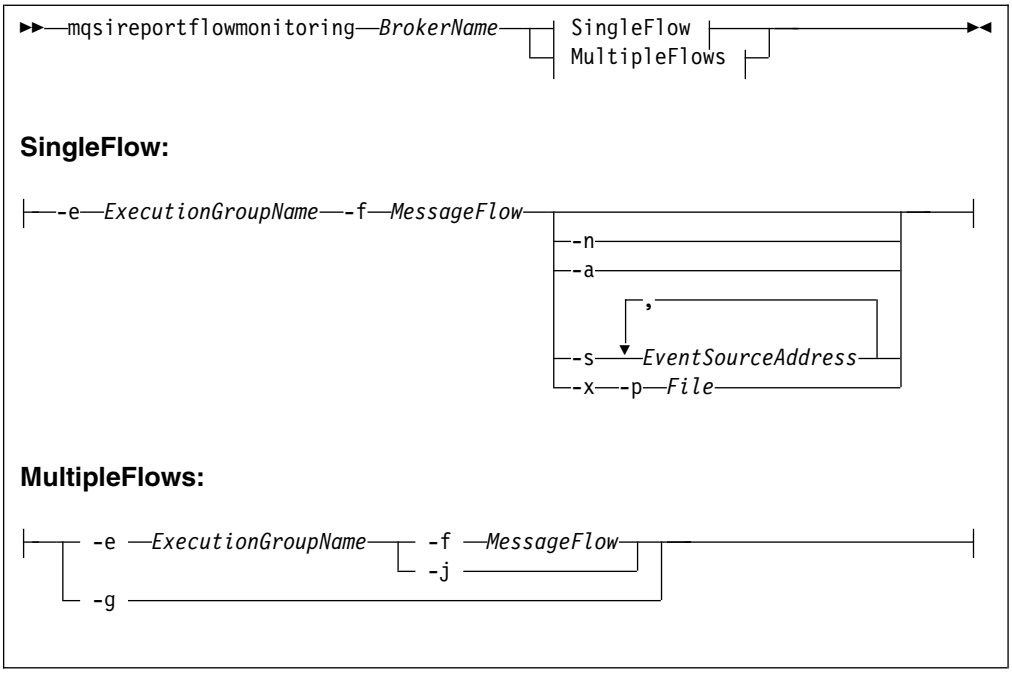
Related reference:

"**mqsichangeflowmonitoring** command" on page 3738
Use the **mqsichangeflowmonitoring** command to enable monitoring of message flows.

mqsireportflowmonitoring command - Windows, Linux and UNIX systems:

Use the **mqsireportflowmonitoring** command on Windows, Linux and UNIX systems to display the current options for monitoring that have been set by the **mqsichangeflowmonitoring** command.

Syntax:



Parameters:

BrokerName
(Required) Specify the label of the broker, for which monitoring options are to be reported.

-a
(Optional) Report the flow-level monitoring properties and the properties of all event sources within the flow, whether configured or not.

-e *ExecutionGroupName*
(Required) Specify the name for the execution group, for which monitoring options are to be reported.

You must specify either **-e** or **-g**. If you do not specify one of these arguments you receive an error message.

-f *MessageFlow*
(Required) Specify the label for the message flow, for which monitoring options are to be reported.

You must specify either **-f** or **-j**. If you do not specify one of these arguments you receive an error message.

-g
(Required) Specifies that the command applies to **all** execution groups that belong to the broker.

You must specify either **-e** or **-g**. If you do not specify one of these arguments you receive an error message.

-j
(Required) Specifies that the command applies to **all** message flows that belong to the execution group.

You must specify either **-f** or **-j**. If you do not specify one of these arguments you receive an error message.

Note: If you set the **-g** option for all execution groups, you must use **-j** instead of **-f**.

-n
(Optional) Report the flow-level monitoring properties and the properties of any configured event sources within the flow.

-p (Optional unless **-x** is specified.) The file name to which the monitoring profile will be written, in XML format.

-s *EventSourceAddress*
(Optional) Report the flow-level monitoring properties and the properties of the specified event sources within the flow, whether configured or not.

You must provide a comma-separated list of the event source addresses for which monitoring options are to be reported. An event source address takes the form <*node name*>.<*event source*>, where <*event source*> is one of the following values:

'terminal.<*terminal name*>'

'transaction.Start'

'transaction.End'

'transaction.Rollback'

If a message flow contains two or more nodes with identical names, the event sources on those nodes cannot be accurately addressed. If this is attempted, behavior is undefined.

Note: <*node name*> is the label of the node as known by the broker runtime components. If the node is in a subflow the label reflects this. For example, flow A contains an instance of flow B as a subflow labeled 'myB'; flow B contains an instance of a Compute node labeled 'myCompute'. The <*node name*> for the Compute node is 'myB.myCompute'.

-x
(Optional; if you specify -x you must also specify the -p parameter.) Outputs the current monitoring properties for the specified message flow as a monitoring profile XML file.

Note:

- If all flags are omitted, only the flow-level properties are reported.
- Flags -n, -a, -s and -x can be used only when -f is present.
- Flags -n, -a, -s and -x are alternatives and cannot be used together.

Examples:
Request a report of the monitoring options for message flow "MyFlow1" in the execution group "default" for broker "BrokerA":

```
mqsireportflowmonitoring BrokerA -e default -f MyFlow1
```

Request a report of the current monitoring options for all message flows in all execution groups for broker "BrokerA" :

```
mqsireportflowmonitoring BrokerA -g -j
```

Related tasks:

“Reporting monitoring settings” on page 3343
Use the **mqsireportflowmonitoring** command to report monitoring settings for a flow.

Related reference:

"Syntax diagrams" on page 3677

"**mqsireportflowmonitoring** command" on page 3924
Use the **mqsireportflowmonitoring** command to display the current options for monitoring that have been set using the **mqsichangeflowmonitoring** command.

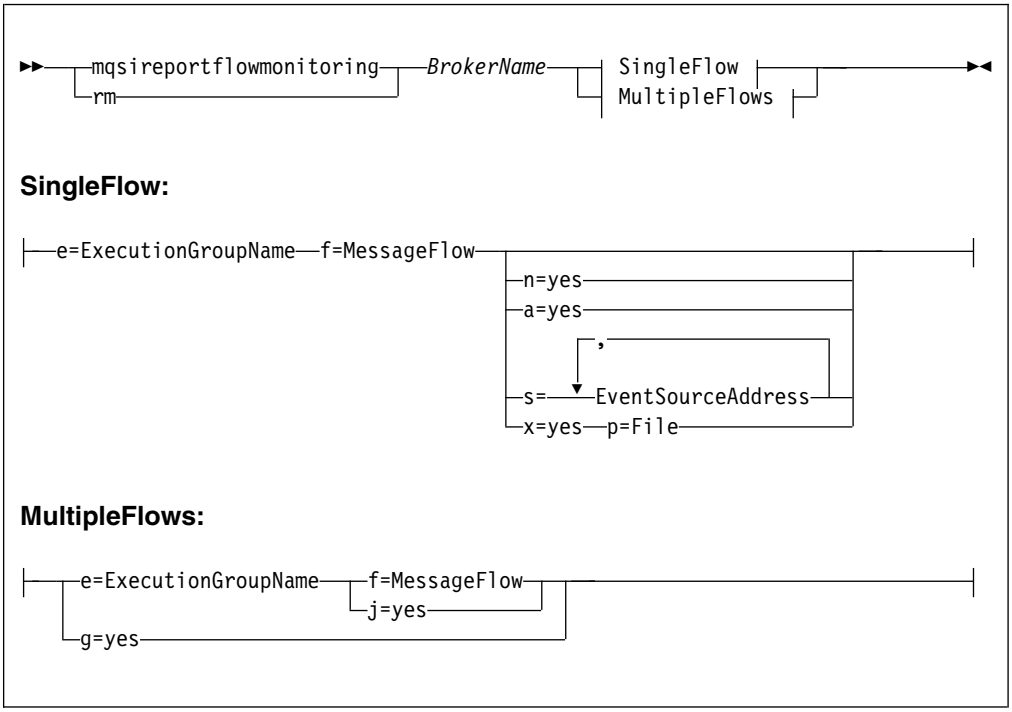
"**mqsichangeflowmonitoring** command" on page 3738
Use the **mqsichangeflowmonitoring** command to enable monitoring of message flows.

mqsireportflowmonitoring command - z/OS:

Use the **mqsireportflowmonitoring** command on z/OS to display the current options for monitoring that have been set by the **mqsichangeflowmonitoring** command.

Syntax:

z/OS console command:
Synonym: rm



Parameters:

BrokerName
(Required) Specify the label of the broker, for which monitoring options are to be reported.

a (Optional) Report the flow-level monitoring properties and the properties of all event sources within the flow, whether configured or not.

e *ExecutionGroupName*
(Required) Specify the name for the execution group, for which monitoring options are to be reported.

You must specify either **-e** or **-g**. If you do not specify one of these arguments you receive an error message.

f *MessageFlow*
(Required) Specify the label for the message flow, for which monitoring options are to be reported.

You must specify either **-f** or **-j**. If you do not specify one of these arguments you receive an error message.

g (Required) Specifies that the command applies to **all** execution groups that belong to the broker.

You must specify either **-e** or **-g**. If you do not specify one of these arguments you receive an error message.

j (Required) Specifies that the command applies to **all** message flows that belong to the execution group.

You must specify either **-f** or **-j**. If you do not specify one of these arguments you receive an error message.

Note: If you set the **-g** option for all execution groups, you must use **-j** instead of **-f**.

n (Optional) Report the flow-level monitoring properties and the properties of any configured event sources within the flow.

p (Optional unless `-x` is specified.) The file name to which the monitoring profile will be written, in XML format.

s ***EventSourceAddress***
(Optional) Report the flow-level monitoring properties and the properties of the specified event sources within the flow, whether configured or not.

You must provide a comma-separated list of the event source addresses for which monitoring options are to be reported. An event source address takes the form <*node name*>.<*event source*>, where <*event source*> is one of the following values:

`'terminal.`<*terminal name*>'

`'transaction.Start'`

`'transaction.End'`

`'transaction.Rollback'`

If a message flow contains two or more nodes with identical names, the event sources on those nodes cannot be accurately addressed. If this is attempted, behavior is undefined.

Note: <*node name*> is the label of the node as known by the broker runtime components. If the node is in a subflow the label reflects this. For example, flow A contains an instance of flow B as a subflow labeled 'myB'; flow B contains an instance of a Compute node labeled 'myCompute'. The <*node name*> for the Compute node is 'myB.myCompute'.

x (Optional; if you specify -x you must also specify the -p parameter.) Outputs the current monitoring properties for the specified message flow as a monitoring profile XML file.

Note:

- If all flags are omitted, only the flow-level properties are reported.

- Flags -n, -a, -s and -x can be used only when -f is present.
- Flags -n, -a, -s and -x are alternatives and cannot be used together.

Examples:
Request a report of monitoring options for message flow "MFlow1" in the execution group "default":

```
F MI10BRK,rm e='default',f='MFlow1'
```

Request a report of the current monitoring options for all message flows in all execution groups:

```
F MI10BRK,rm g=yes,j=yes
```

Related tasks:

"Reporting monitoring settings" on page 3343
Use the **mqsireportflowmonitoring** command to report monitoring settings for a flow.

Related reference:

"Syntax diagrams" on page 3677

"**mqsireportflowmonitoring** command" on page 3924
Use the **mqsireportflowmonitoring** command to display the current options for monitoring that have been set using the **mqsichangeflowmonitoring** command.

"**mqsichangeflowmonitoring** command" on page 3738
Use the **mqsichangeflowmonitoring** command to enable monitoring of message flows.

mqsireportflowstats command:

Use the **mqsireportflowstats** command to display the current options for accounting and statistics that have been set using the **mqsichangeflowstats** command.

Supported platforms:
- Windows
- Linux and UNIX systems
- z/OS. Run this command in one of two ways - as a console command, or by customizing and submitting BIPRPMS; see "Contents of the broker PDSE" on page 3991

Purpose:
Select the appropriate link for details of this command on the platform, or platforms, that your enterprise uses:
- "mqsireportflowstats command - Windows, Linux and UNIX systems" on page 3930
- "mqsireportflowstats command - z/OS" on page 3931

Authorization:
For information about platform-specific authorizations, see the following topics:
- "Security requirements for Linux and UNIX platforms" on page 3648
- "Security requirements for Windows systems" on page 3651
- "Security requirements for z/OS" on page 3655

If you have enabled broker administration security, you must also set up the authority detailed in "Tasks and authorizations for broker administration security" on page 3645.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Optimizing message flow throughput" on page 587
Each message flow that you design must provide a complete set of processing for messages received from a certain source. This design might result in very complex message flows that include large numbers of nodes that can cause a performance overhead, and might create potential bottlenecks. You can increase the number of message flows that process your messages to provide the opportunity for parallel processing and therefore improved throughput.

"Optimizing message flow response times" on page 3264
You can use different solutions to improve message flow response times.

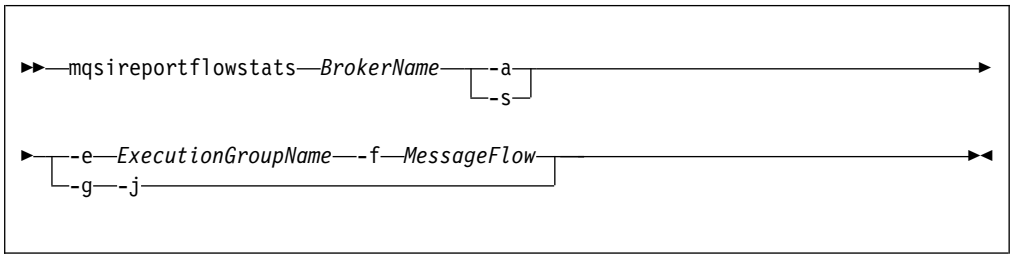
Related reference:

"**mqsichangeflowstats** command" on page 3744
Use the **mqsichangeflowstats** command to control the accumulation of statistics about message flow operation.

mqsireportflowstats command - Windows, Linux and UNIX systems:

Use the **mqsireportflowstats** command on Windows, Linux and UNIX systems to display the current options for accounting and statistics that have been set by the **mqsichangeflowstats** command.

Syntax:



Parameters:

BrokerName
(Required) Specify the label of the broker for which the previously stored accounting and statistics options are to be reported.

-a (Required) Specify that the command reports the stored settings for the archive accounting and statistics collection.

You must specify **-a** or **-s**, or both arguments. If you do not specify at least one of these arguments you receive a warning message.

-e *ExecutionGroupName*
(Required) Specify the name for the execution group, for which accounting and statistics options are to be reported.

You must specify either **-e** or **-g**. If you do not specify one of these arguments you receive a warning message.

-f *MessageFlow*
(Required) Specify the label for the message flow, for which accounting and statistics options are to be reported.

You must specify either **-f** or **-j**. If you do not specify one of these arguments you receive a warning message.

-g
(Required) Specifies that the command applies to **all** execution groups that belong to the broker.

You must specify either **-e** or **-g**. If you do not specify one of these arguments you receive a warning message.

-j
(Required) Specifies that the command applies to **all** message flows that belong to the execution group.

You must specify either **-f** or **-j**. If you do not specify one of these arguments you receive a warning message.

Note: If you set the **-g** option for all execution groups, you must use **-j** instead of **-f**.

-s (Required) Specify that the command reports the stored settings for the snapshot accounting and statistics collection.

You must specify **-a** or **-s**, or both arguments. If you do not specify at least one of these arguments you receive a warning message.

Examples:
Request a report for message flow "MyFlow1" in the execution group "default" for broker "BrokerA" for both archive and snapshot statistics collection:

```
mqsireportflowstats BrokerA -s -a -e default -f MyFlow1
```

Request a report of the snapshot options that are currently stored for all message flows in all execution groups for broker "BrokerA" :

```
mqsireportflowstats BrokerA -s -g -j
```

Related reference:

"Syntax diagrams" on page 3677

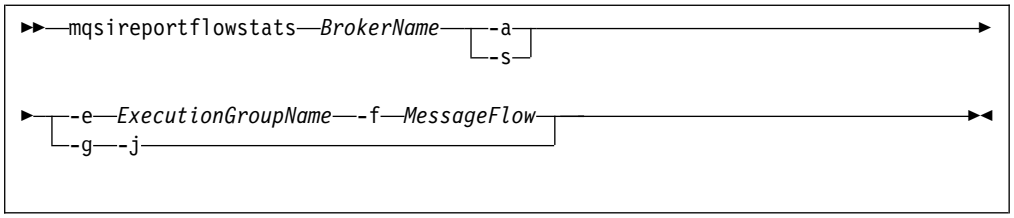
"**mqsireportflowstats** command" on page 3929
Use the **mqsireportflowstats** command to display the current options for accounting and statistics that have been set using the **mqsichangeflowstats** command.

mqsireportflowstats command - z/OS:

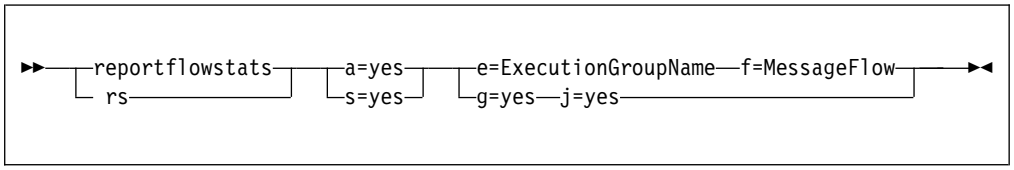
Use the **mqsireportflowstats** command on z/OS to display the current options for accounting and statistics that have been set by the **mqsichangeflowstats** command.

Syntax:

z/OS command - BIPRPMS:



z/OS console command:
Synonym: rs



Parameters:

BrokerName
(Required) Specify the label of the broker for which the previously stored accounting and statistics options are to be reported.

-a (Required) Specify that the command reports the stored settings for the archive accounting and statistics collection.

You must specify **-a** or **-s**, or both arguments. If you do not specify at least one of these arguments you receive a warning message.

-e *ExecutionGroupName*
(Required) Specify the name for the execution group, for which accounting and statistics options are to be reported.

You must specify either **-e** or **-g**. If you do not specify one of these arguments you receive a warning message.

-f *MessageFlow*
(Required) Specify the label for the message flow, for which accounting and statistics options are to be reported.

You must specify either **-f** or **-j**. If you do not specify one of these arguments you receive a warning message.

-g
(Required) Specifies that the command applies to **all** execution groups that belong to the broker.

You must specify either **-e** or **-g**. If you do not specify one of these arguments you receive a warning message.

-j
(Required) Specifies that the command applies to **all** message flows that belong to the execution group.

You must specify either **-f** or **-j**. If you do not specify one of these arguments you receive a warning message.

Note: If you set the **-g** option for all execution groups, you must use **-j** instead of **-f**.

-s (Required) Specify that the command reports the stored settings for the snapshot accounting and statistics collection.

You must specify **-a** or **-s**, or both arguments. If you do not specify at least one of these arguments you receive a warning message.

Examples:
Using the command BIPRPMS:

- Request a report for message flow "MyFlow1" in the execution group "default" for both archive and snapshot statistics collection:

```
mqsireportflowstats  BrokerA -s -a -e default -f MyFlow1
```

- Request a report of the snapshot options that are currently stored for all message flows in all execution groups:

```
mqsireportflowstats BrokerA -s -g -j
```

Using the console form of the command, request a report for message flow "MyFlow1" in the execution group "default" for both archive and snapshot statistics collection:

```
mqsireportflowstats  s= a= e=default f=MyFlow1
```

Related reference:

"Syntax diagrams" on page 3677

"**mqsireportflowstats** command" on page 3929
Use the **mqsireportflowstats** command to display the current options for accounting and statistics that have been set using the **mqsichangeflowstats** command.

mqsireportflowuserexits command:

Use the **mqsireportflowuserexits** command to report the list of active and inactive user exits for the specified broker, execution group, or message flow.

Supported platforms:

- Windows
- Linux and UNIX systems
- z/OS. Run this command in one of two ways - as a console command, or by customizing and submitting BIPRPUE; see "Contents of the broker PDSE" on page 3991

Purpose:
Use the **mqsireportflowuserexits** command to generate a report about the status of user exits:

- If you specify only the broker name, the command returns a list of all user exits that you have set to be active at the broker level by using the **mqsichangebroker** command.
- If you specify the broker and execution group names, the command returns the list returned for the broker name only, plus:
  - A list of all user exits that you have set to be active at the execution group level by using the **mqsichangeflowuserexits** command.
  - A list of all user exits that you have set to be inactive at the execution group level by using the **mqsichangeflowuserexits** command.
- If you specify the broker, execution group, and message flow names, the command returned the list returned for the broker and execution group names, plus:

– A list of all user exits that you have set to be active at the message flow level by using the **mqsichangeflowuserexits** command.
– A list of all user exits that you have set to be inactive at the message flow level by using the **mqsichangeflowuserexits** command.

Select the appropriate link for details of this command on the platform, or platforms, that your enterprise uses:

- "mqsireportflowuserexits command - Windows, Linux, and UNIX systems"
- "mqsireportflowuserexits command - z/OS" on page 3935

Authorization:
For information about platform-specific authorizations, see the following topics:

- "Security requirements for Linux and UNIX platforms" on page 3648
- "Security requirements for Windows systems" on page 3651
- "Security requirements for z/OS" on page 3655

If you have enabled broker administration security, you must also set up the authority detailed in "Tasks and authorizations for broker administration security" on page 3645.

Related concepts:

"User exits" on page 3015
A user exit is user-provided custom software, written in C, to track data passing through message flows.

Related tasks:

"Developing a user exit" on page 3114
Develop a user exit by declaring it, implementing its behavior, then compiling it.

"Deploying a user exit" on page 3116
Deploy your user exit to the broker.

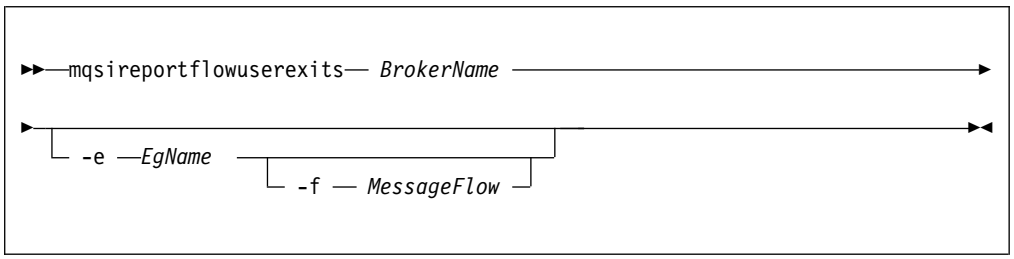
Related reference:

"**mqsichangeflowuserexits** command" on page 3751
Use the **mqsichangeflowuserexits** command to set the list of active or inactive user exits. A list of active and a list of inactive user exits is maintained for each execution group and message flow. The effective state of user exits for a given flow is decided when the flow starts.

mqsireportflowuserexits command - Windows, Linux, and UNIX systems:

Use the **mqsireportflowuserexits** command on distributed systems to report the list of active and inactive user exits for the specified broker, execution group, or message flow.

Syntax:



Parameters:

BrokerName
(Required). The name of the broker.

-e *EgName*
(Optional). The name of the execution group.

-f *MessageFlow*
(Optional). The name of the message flow.

The user exits that are reported by this command depend on the parameters that you specify.

- If you specify only the broker name, the command lists user exits that have been set to active at the broker level by using the **mqsichangebroker** command.
- If you specify the broker name and execution group name, the command lists the following user exits:
  - User exits that have been set to active at the broker level by using the **mqsichangebroker** command
  - User exits that have been set to active at the execution group level by using the **mqsichangeflowuserexits** command
  - User exits that have been set to inactive at the execution group level by using the **mqsichangeflowuserexits** command
- If you specify the broker name, execution group name, and a message flow name, the command lists the following user exits:
  - User exits that have been set to active at the broker level by using the **mqsichangebroker** command
  - User exits that have been set to active at the execution group level by using the **mqsichangeflowuserexits** command
  - User exits that have been set to inactive at the execution group level by using the **mqsichangeflowuserexits** command
  - User exits that have been set to active at the message flow level by using the **mqsichangeflowuserexits** command
  - User exits that have been set to inactive at the message flow level by using the **mqsichangeflowuserexits** command

Examples:

```
mqsireportflowuserexits MB7BROKER -e default -f MYFLOW
BIP8859 User Exits active for broker MYBROKER: exit1, exit2
BIP8854 User Exits active for Execution Group default: exit1,exit3
BIP8855 User Exits inactive for Execution Group default: exit2
BIP8856 User Exits active for Message Flow MYFLOW: exit2
BIP8857 User Exits inactive for Message Flow MYFLOW: exit1
```

Related reference:

"Syntax diagrams" on page 3677

"**mqsireportflowuserexits** command" on page 3933

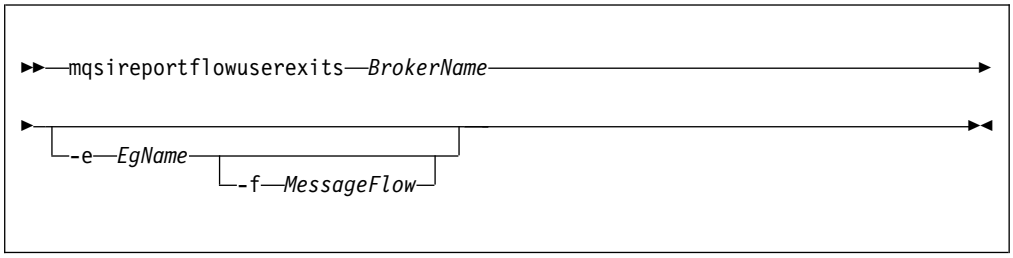
Use the **mqsireportflowuserexits** command to report the list of active and inactive user exits for the specified broker, execution group, or message flow.

mqsireportflowuserexits command - z/OS:

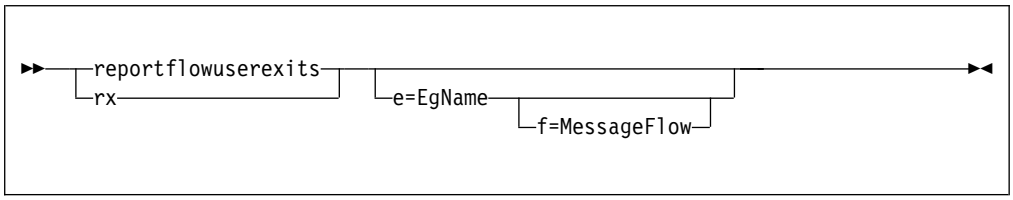
Use the **mqsireportflowuserexits** command on z/OS to report the list of active and inactive user exits for the specified broker, execution group, or message flow.

Syntax:

z/OS command - BIPRPUE:



z/OS console command:
Synonym: rx



Parameters:

BrokerName
(Required). The name of the broker.

-e *EgName*
(Optional). The name of the execution group.

-f *MessageFlow*
(Optional). The name of the message flow.

The user exits that are reported by this command depend on the parameters that you specify.

- If you specify only the broker name, the command lists user exits that have been set to active at the broker level by using the **mqsichangebroker** command.
- If you specify the broker name and execution group name, the command lists the following user exits:
  - User exits that have been set to active at the broker level by using the **mqsichangebroker** command
  - User exits that have been set to active at the execution group level by using the **mqsichangeflowuserexits** command
  - User exits that have been set to inactive at the execution group level by using the **mqsichangeflowuserexits** command
- If you specify the broker name, execution group name, and a message flow name, the command lists the following user exits:
  - User exits that have been set to active at the broker level by using the **mqsichangebroker** command
  - User exits that have been set to active at the execution group level by using the **mqsichangeflowuserexits** command

- User exits that have been set to inactive at the execution group level by using the **mqsichangeflowuserexits** command
- User exits that have been set to active at the message flow level by using the **mqsichangeflowuserexits** command
- User exits that have been set to inactive at the message flow level by using the **mqsichangeflowuserexits** command

Examples:

```
mqsireportflowuserexits MB7BROKER -e default -f MYFLOW
BIP8859 User Exits active for broker MYBROKER: exit1, exit2
BIP8854 User Exits active for Execution Group default: exit1,exit3
BIP8855 User Exits inactive for Execution Group default: exit2
BIP8856 User Exits active for Message Flow MYFLOW: exit2
BIP8857 User Exits inactive for Message Flow MYFLOW: exit1
```

Related reference:

"Syntax diagrams" on page 3677

"**mqsireportflowuserexits** command" on page 3933

Use the **mqsireportflowuserexits** command to report the list of active and inactive user exits for the specified broker, execution group, or message flow.

mqsireportproperties command:

Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

You can also use the WebSphere Message Broker Explorer to create and view configurable services. See "Using the WebSphere Message Broker Explorer to work with configurable services" on page 644 for more information.

Supported platforms:

- Windows
- Linux and UNIX systems
- z/OS. Run this command by customizing and submitting BIPRPPR; see "Contents of the broker PDSE" on page 3991.

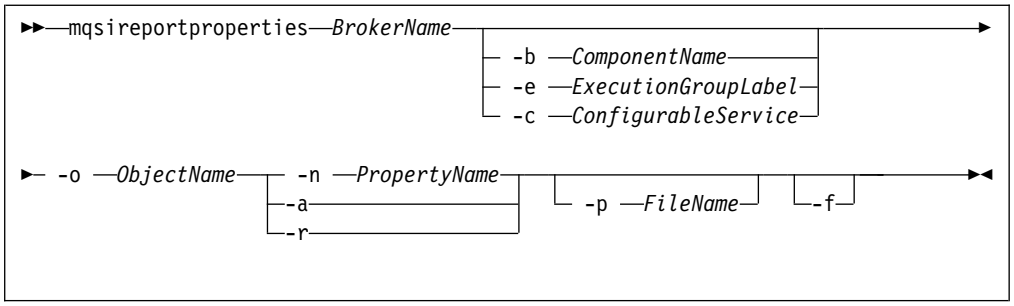
Purpose:

Use the **mqsireportproperties** command to examine the values of properties or broker resources that are set by using the **mqsichangeproperties** command, or created by using the **mqsicreateconfigurableservice** command.

Usage notes:

- Before you run the **mqsireportproperties** command, ensure that the broker is running.
- If you use the **mqsichangeproperties** command to change any value, the change is applied only after you stop and restart the broker or execution group.
- If you run the **mqsireportproperties** command after changing properties, but before you restart the broker or execution group, the command shows the changed value, even though that value is not yet in effect.

Syntax:



Parameters:

BrokerName
(Required) The name of the broker.

-b *ComponentName*
(Optional) The name of the component selected. Valid values are *httplistener*, *securitycache*, and *servicefederation*.

-c *ConfigurableService*
(Optional) The type of the configurable service. Specify a value of `AllTypes` to report on all configurable service types.

For a list of supplied configurable services, and their properties and values, see "Configurable services properties" on page 3766.

-e *ExecutionGroupLabel*
(Optional) The label of the execution group for which a report is required.

-o *ObjectName*
(Required) The name of the object whose properties you want to read.

You must also specify **-b**, **-e**, or **-c** after **-o**, except if you specify the object name `BrokerRegistry`.

Set *ObjectName* to match other parameters:

- Specify `BrokerRegistry` for broker registry parameters.
- Specify the name of a configurable service (predefined or user-defined) of a type that you have specified with **-c**.

  For example:

  – **-c EISProviders** with `SAP`, `Siebel`, or `PeopleSoft` for a predefined WebSphere Adapters configurable service.

  – **-c JMSProviders** with the name of a predefined or user-defined service, for example, `WebSphere_MQ`.
- Specify a communications object for the `httplistener` component that you have specified with **-b**; one of `HTTPListener`, `HTTPConnector`, or `HTTPSConnector`. Values are defined for all HTTP nodes that you have deployed to the broker.

  You can set a parameter to disable or enable the broker-wide listener for HTTP nodes; by default, this listener is active.
- Specify a communications object for the execution group that you have specified with **-e**; one of `HTTPListener`, `HTTPConnector`, or `HTTPSConnector`. Values are defined for all SOAP nodes that you have deployed to the specified execution group.

  You can configure the execution group such that all HTTP nodes use the embedded listener instead of the broker-wide listener. For more information, see "HTTP listeners" on page 1589.

- Specify `DynamicSubscriptionEngine` for inter-broker communications properties for the execution group that you have specified with **-e**. These properties apply to brokers that you have configured in collectives, or cloned.
- Specify `SecurityCache` for properties for the `securitycache` component that you have specified with **-b**.
- Specify an object for the `servicefederation` component that you have specified with **-b**; one of `scmp`, `HTTPConnector`, or `HTTPSConnector`. The properties apply to the broker HTTP(S) port that processes Service Federation Management (SFM) SCMP Atom requests.
- Specify `ServiceFederationManager` for properties involved in the creation of an SFM proxy for the execution group that you have specified with **-e**.

Specify a value of `AllReportableEntityNames` to return a list of all valid object names. If you run the **mqsireportproperties** command on the command line without any properties, `AllReportableEntityNames` is used.

-n *PropertyName*
(Optional) Display only the named property.

You must select only one option from **-n**, **-a**, and **-r**.

-a
(Optional) Indicates that all property values of the object are displayed, and does not act in a recursive manner on properties that have child values.

-r
(Optional) Indicates that all property values of the object are displayed and, additionally, displays the child values for all properties that have child values.

Note, that this option does not show all the possible values for `AllReportableEntityNames`. Furthermore, not all entities support this parameter.

-p *FileName*
(Optional) The location and name of the file to which the command writes all selected properties. If you do not specify **-n**, the property values, but not the property names, are written.

-f
(Optional) This parameter is valid only if you specify an execution group level object.

You can use this parameter only when the execution group is in the stopped state, and you must also specify **-e** and **-o**.

If you specify this parameter for other configurable services or components, an error is generated.

For more information about objects, properties, and values, and valid combinations of these parameters, see "**mqsichangeproperties** command" on page 3756.

For the *httplistener* component, the **mqsireportproperties** command does not report those properties that have not been explicitly set with the **mqsichangeproperties** command, even if those properties have a default setting.

For example, the default HTTPSConnector port that is used (unless it has been changed) is 7083. However, this value is not reported by the **mqsireportproperties** command unless you have changed it from this default by using the

mqsichangeproperties command. To see the default values for all properties that the **mqsireportproperties** command can report, see the **mqsichangeproperties** command description.

Authorization:
For information about platform-specific authorizations, see the following topics:

- "Security requirements for Linux and UNIX platforms" on page 3648
- "Security requirements for Windows systems" on page 3651
- "Security requirements for z/OS" on page 3655

If you have enabled broker administration security, you must also set up the authority detailed in "Tasks and authorizations for broker administration security" on page 3645.

Responses:
Responses are of the form:

- `HTTPConnector`
  - PortNumber = 7800
- `HTTPSConnector`
  - PortNumber = 7843

Examples:
Always enter the command on a single line; in some examples, a line break has been added to enhance readability.

Displaying properties associated with broker components

The following examples include the **-b** parameter to specify the component to view.

- Check if the broker-wide listener is active for deployed HTTP nodes in one or more execution groups:

```
mqsireportproperties MB7BROKER -b httplistener -o HTTPListener -n startListener
```

- Display all the current HTTPListener settings associated with HTTP nodes (defined in the httplistener component):

  ```
  mqsireportproperties MB7BROKER -b httplistener -o HTTPListener -a
  ```

- Display the HTTPSConnector port setting for the HTTP nodes (defined in the httplistener component):

  ```
  mqsireportproperties MB7BROKER -b httplistener -o HTTPSConnector -n port
  ```

- Display all Service Federation object properties and their values:

```
mqsichangeproperties MB7BROKER -b servicefederation -o AllReportableEntityNames -r
```

Displaying properties associated with execution groups

The following example includes the **-e** parameter to specify the execution group to view.

- Display the FTE agent name for execution group FTESAMPLE in broker MB7BROKER:

  ```
  mqsireportproperties MB7BROKER -e FTESAMPLE -o FTEAgent -n agentName
  ```

  If the agent has been created, the command returns the agent name. If the agent has not been created, the command returns an empty string.

- Check if the deployed HTTP nodes are using the execution group embedded listener:

```
mqsireportproperties MB7BROKER -e exgroup1 -o ExecutionGroup -n httpNodesUseEmbeddedListener
```

- Display the properties that control the creation of an SFM proxy for an execution group that is to be used as an SFM Connectivity Provider:

  ```
  mqsireportproperties MB7BROKER -e exgroup1 -o ServiceFederationManager -a
  ```

  The result of this command has the following format:

```
ServiceFederationManager
  uuid='ServiceFederationManager'
  userTraceLevel='none'
  traceLevel='none'
  userTraceFilter='none'
  traceFilter='none'
  port='8811'
  securePort='8844'
  maxWaitTime='180'
  proxyURLHostName=' mbhost.ibm.com '
  proxyPathPrefix='proxy'
  proxyPathPrefixesEnabled='TRUE'
  creationTime='2009-09-11 15:37:52.639219'
  nextProxyPathPrefixCount='8'
  ownedProxyGroupCount='2'
```

Displaying properties associated with configurable services

The following examples include the **-c** parameter to specify the configurable service to view.

- Display all Aggregation configurable services:

  ```
  mqsireportproperties MB7BROKER -c Aggregation -o AllReportableEntityNames -r
  ```

- Display all CICSConnection configurable services:

  ```
  mqsireportproperties MB7BROKER -c CICSConnection -o AllReportableEntityNames -r
  ```

- Display all configurable services for Connect:Direct server (that you have defined to this broker with the **mqsicreateconfigurableservice** command)

  ```
  mqsireportproperties MB7BROKER -c CDServer -o AllReportableEntityNames -r
  ```

  The result of this command has the following format:

```
CDServer
  Default
    hostname='localhost'
    port='1363'
    brokerPathToStagingDir=''
    cdPathToStagingDir=''
    brokerPathToInputDir=''
    cdPathToInputDir=''
    queuePrefix= ''
    securityIdentity="Default"
```

- Display all Collector configurable services:

  ```
  mqsireportproperties MB7BROKER -c Collector -o AllReportableEntityNames -r
  ```

- Display all CORBA configurable services:

  ```
  mqsireportproperties MB7BROKER -c CORBA -o AllReportableEntityNames -r
  ```

- Display all EmailServer configurable services that EmailInput nodes or message flows are referring to at runtime:

  ```
  mqsireportproperties MB7BROKER -c EmailServer -o AllReportableEntityNames -r
  ```

- Display all properties of the FtpServer configurable service called TEST1:

  ```
  mqsireportproperties MB7BROKER -c FtpServer -o TEST1 -r
  ```

- Display the **protocol** property setting of the FtpServer configurable service called TEST1:

```
mqsireportproperties MB7BROKER -c FtpServer -o TEST1 -n protocol
```

- Display all IMSConnect configurable services:

```
mqsireportproperties MB7BROKER -c IMSConnect -o AllReportableEntityNames -r
```

- Report the properties of the Oracle JDBCProvider configurable service:

```
mqsireportproperties MB7BROKER -c JDBCProviders -o Oracle -r
```

- Display the properties for all the broker's JMS provider resources (default JMS provider resources and those configurable services that you have defined by using the **mqsicreateconfigurableservice** command):

```
mqsireportproperties MB7BROKER -c JMSProviders
     -o AllReportableEntityNames -r
```

- Display the properties for all the JMS provider resources of WebSphere MQ.

```
mqsireportproperties MB7BROKER -c JMSProviders
     -o WebSphere_MQ -r
```

- Display the properties for all the JMS provider resources (default JMS provider resources and those configurable services that you have defined to this broker with the **mqsicreateconfigurableservice** command):

```
mqsireportproperties MB7BROKER -c JMSProviders
     -o BEA_WebLogic –r
```

  The result of this command has the following format:

```
ReportableEntityName=''
JMSProviders
  BEA_Weblogic=''
    jarsURL='default_Path'
    nativeLibs='default_Path'
    proprietaryAPIAttr1='weblogic.jndi.WLInitialContextFactory'
    proprietaryAPIAttr2='t3://9.20.94.16:7001'
    proprietaryAPIAttr3='BEAServerName'
    proprietaryAPIAttr4='default_none'
    proprietaryAPIAttr5='default_none'proprietaryAPIHandler='BEAWebLogicAPIHandler.jar'
```

- Display all PeopleSoftConnection configurable services:

```
mqsireportproperties MB7BROKER -c PeopleSoftConnection -o AllReportableEntityNames -r
```

- Display all policy sets for broker MB7BROKER:

```
mqsireportproperties MB7BROKER -c PolicySets
     -o AllReportableEntityNames -a
```

- Display all policy set bindings for broker MB7BROKER:

```
mqsireportproperties MB7BROKER -c PolicySetBindings
     -o AllReportableEntityNames -a
```

- Export policy set Policy_2 in broker MB7BROKER to file `policyset.xml`:

```
mqsireportproperties MB7BROKER -c PolicySets
     -o Policy_2 -n ws-security -p policyset.xml
```

  You can use the output file to move policy sets between brokers, and for backup.

- Export a policy set bindings from a broker to a file:

```
mqsireportproperties MB7BROKER -c PolicySetBindings
     -o Bindings_2 -n ws-security -p bindings.xml
```

  This command writes the Policy Set Bindings file Bindings_2 in broker MB7BROKER to file `bindings.xml`. You can use this file to move policy set bindings between brokers, and for backup.

- Display all Resequence configurable services:

```
mqsireportproperties MB7BROKER -c Resequence -o AllReportableEntityNames -r
```

- Report the dependencies for the WebSphere Adapter for SAP:

```
mqsireportproperties MB7BROKER -c EISProviders -o SAP -r
```

- Display all SAPConnection configurable services:

  ```
  mqsireportproperties MB7BROKER -c SAPConnection -o AllReportableEntityNames -r
  ```

- Display the properties for all the security profiles (default security profiles and any that you have defined on this broker by using the **mqsicreateconfigurableservice** command):

  ```
  mqsireportproperties MB7BROKER -c SecurityProfiles
       -o AllReportableEntityNames -r
  ```

  The result of this command has the following format:

  ```
  ReportableEntityName=''
  SecurityProfiles

   Default_Propagation=''
    Authentication = 'NONE'
    AuthenticationConfig = ''
    Mapping = 'NONE'
    MappingConfig = ''
    Authorization = 'NONE'
    AuthorizationConfig = ''
    Propagation = 'TRUE'
    passwordValue = 'PLAIN'
  ```

- Display the properties for the security profile called MyFirstSecurityProfile:

  ```
  mqsireportproperties MB7BROKER -c SecurityProfiles
       -o MyFirstSecurityProfile -r
  ```

  The result of this command has the following format:

  ```
  ReportableEntityName=''
  SecurityProfiles
   MyFirstSecurityProfile=''
    Authentication = 'LDAP'
    AuthenticationConfig = 'ldap://localhost:389/ou=users,o=ibm'
    Mapping = 'TFIM'
    MappingConfig = 'http://tfimhost1:80'
    Authorization = 'NONE'
    AuthorizationConfig = ''
    Propagation = 'TRUE'
    passwordValue = 'PLAIN'
  ```

- Display all SiebelConnection configurable services:

  ```
  mqsireportproperties MB7BROKER -c SiebelConnection -o AllReportableEntityNames -r
  ```

- Display all TCPIPclient configurable services:

  ```
  mqsireportproperties MB7BROKER -c TCPIPClient
        -o AllReportableEntityNames -r
  ```

- Display all TCPIPServer configurable services:

  ```
  mqsireportproperties MB7BROKER -c TCPIPServer
        -o AllReportableEntityNames -r
  ```

- Display all Timer configurable services:

  ```
  mqsireportproperties MB7BROKER -c Timer -o AllReportableEntityNames -r
  ```

- Display all JavaClassLoader services:

  ```
  mqsireportproperties MB7BROKER -c JavaClassLoader
        -o AllReportableEntityNames -r
  ```

Related concepts:

"HTTP listeners" on page 1589
You can choose between broker-wide listeners and execution group (embedded) listeners to manage HTTP messages in your HTTP flows. Learn about the two types of listener, how ports are assigned to them, and how you can switch from one to the other for individual execution groups.

"Using the WebSphere Message Broker Explorer to work with configurable services" on page 644
Configurable services are used to define properties that are related to external services on which the broker relies. Use the WebSphere Message Broker Explorer to view, add, modify and delete configurable services.

Related tasks:

Chapter 8, "Administering brokers and broker resources," on page 899
Administering brokers and associated broker resources includes the tasks that you perform frequently to activate and manage those resources. Choose the method you prefer to administer your brokers and associated resources.

"Setting up a JDBC provider for type 4 connections" on page 684
Use the **mqsicreateconfigurableservice** or the **mqsichangeproperties** command to configure a JDBC provider service.

Related reference:

"Syntax diagrams" on page 3677

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

"Parameter values for the servicefederation component" on page 3816
Select the objects and properties associated with the servicefederation component that you want to change.

"ServiceFederationManager object property values" on page 3818
Select the properties associated with the ServiceFederationManager component that you want to change.

mqsireportresourcestats command:

Use the **mqsireportresourcestats** command to display current statistics gathering options for resources in the broker.

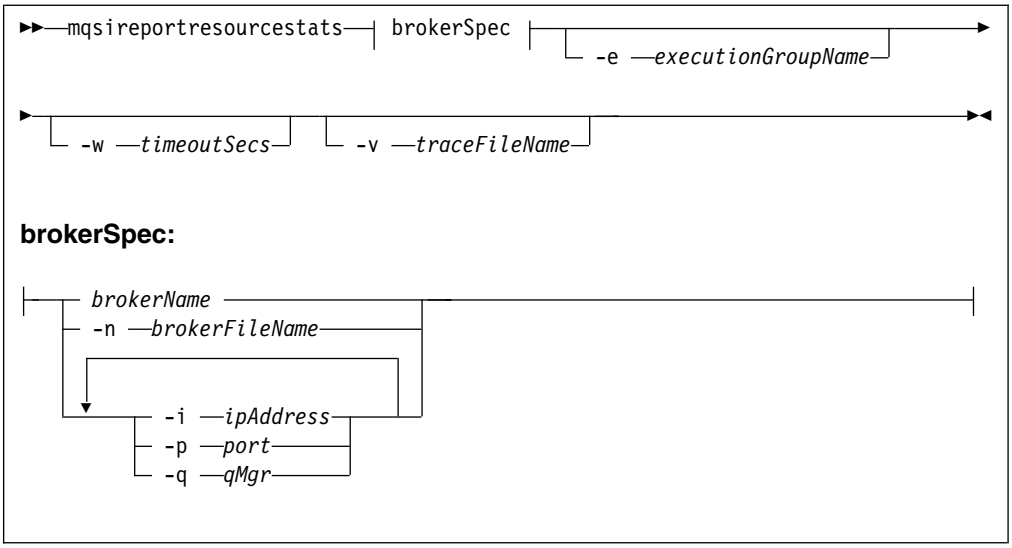
Supported platforms:

- Windows systems
- Linux and UNIX systems
- z/OS. Run this command by customizing and submitting BIPRPRS; see "Contents of the broker PDSE" on page 3991

Purpose:
Use this command to view the status of statistics collection for the resource types listed in "Resource statistics data" on page 6745.

You can run this command only if the broker is running.

Syntax:



Parameters:

brokerSpec

(Required) You must specify at least one parameter to identify the target broker for this command, in one of the following forms:

brokerName

This parameter identifies the name of a locally defined broker. You cannot use this option if the broker is on a remote computer.

-n *brokerFileName*

This parameter identifies the name of a file that contains the connection details for a local or remote broker.

Use this option if multiple users want to connect to the same broker, or if you are using advanced connection parameters such as SSL.

To create this file, right-click the broker in the WebSphere Message Broker Explorer and select **Export *.broker file**. When prompted, navigate to the location in which you want to save the file and enter the file name; the extension `.broker` is appended automatically when you click **Save**. Include the location (path) and file name when you specify this parameter. You must ensure that the location is accessible when you run this command.

If you want to run a command that uses SSL to administer a remote broker over a secured channel, you must specify the keystore and truststore password for the connection using the IBM_JAVA_OPTIONS environment variable. See "Resolving problems when running commands" on page 3364 for further information.

-i *ipAddress*, **-p** *port*, **-q** *qMgr*

These parameters identify the connection details for the broker.

Use this option for connections to remote brokers that do not require advanced connection parameters.

If you choose this option, you must specify at least one of these three parameters; the order in which you specify them is insignificant. You cannot specify each parameter more than once.

Parameters that you omit assume default values:

- **-i** *ipAddress*: The host name or the IP address of the computer on which the broker is running. If you do not specify this parameter, a value that represents the local computer is used.
- **-p** *port*: The TCP port on which the broker's queue manager is listening. If you do not specify this parameter, the value 1414 is used.
- **-q** *qMgr*: The name of the broker's queue manager. If you do not specify this parameter, a value that represents the default queue manager on the local computer is used.

You cannot connect to a remote queue manager on z/OS; see "Usage note."

-e *executionGroupName*
(Optional) The name of the execution group for which resource statistics collection status is reported.

If you do not specify **-e**, resource statistics status is reported for all execution groups on the broker.

-w *timeoutSecs*
(Optional) This parameter specifies the time in seconds that the utility waits to ensure that the command completed; the default value is 60.

-v *traceFileName*
(Optional) This parameter sends internal debug trace information to the specified file.

Usage note:
If you try to connect to a remote broker by specifying the **i**, **p**, and **q** parameters, or by using a connection parameter file (`.broker`), the command attempts to use WebSphere MQ Java client code. This option is not supported on z/OS, and returns the following error:

```
BIP1046E: Unable to connect with the broker (name)
```

The reported reason code is `MQRC_ENVIRONMENT_ERROR`. You must specify a local queue manager.

Authorization:
For information about platform-specific authorizations, see the following topics:

- "Security requirements for Linux and UNIX platforms" on page 3648
- "Security requirements for Windows systems" on page 3651
- "Security requirements for z/OS" on page 3655

If you have enabled broker administration security, you must also set up the authority detailed in "Tasks and authorizations for broker administration security" on page 3645.

Responses:
This command returns the following responses:

0 The command completed successfully.

2 (Failure) The broker received the deployment request but was unable to process it successfully. See the messages issued from the utility (or the Administration log) for more information.

9 (Failure) The request has been submitted to the broker, but no response was received before the timeout expired.

10 (Failure) Another user or application canceled the request operation before the broker was able to process it.

98 The broker is not running.

99 One or more of the parameters that you specified is invalid.

If the command completes successfully, one report message per resource type is returned for each execution group specified by the command. Each message has the general format:

```
BIP8940I Statistics settings for resource type type in execution group execution_group_name is sta
```

where:

- *type* is the resource type. For more information about the resource types for which statistics can be requested, see "Resource statistics" on page 3306.
- *state* can be `active` or `inactive`
- *measurement* is a repeating field that depends on the resource type; for example, `TotalSocketsOpened` is returned for resource type `Sockets`. Details of all the measurements reported are provided in "Resource statistics data" on page 6745.

Examples:
Request a report for the execution group `default` on broker `BrokerA` for statistics collection for all resources:

```
mqsireportresourcestats BrokerA -e default
```

Request a report for all execution groups on broker `BrokerA` for statistics collection for all resources:

```
mqsireportresourcestats BrokerA
```

:

Related concepts:

"Resource statistics" on page 3306
Resource statistics are collected by a broker to record performance and operating details of resources that are used by execution groups.

Related tasks:

"Monitoring resource performance" on page 3305
You can collect statistics to assess the performance of certain resources used by execution groups.

Related reference:

"Syntax diagrams" on page 3677

"**mqsichangeresourcestats** command" on page 3819
Use the **mqsichangeresourcestats** command to control statistics gathering for resources in the broker.

"Resource statistics data" on page 6745
Learn about the measurements for which data is returned when you activate resource statistics collection.

"Example of an XML publication for resource statistics" on page 6746
This example message shows an XML publication that contains resource statistics data.

mqsireporttrace command:

Use the **mqsireporttrace** command to display the trace options currently in effect. Trace can be run only against a broker, or any of its resources.

Supported platforms:

- Windows
- Linux and UNIX systems
- z/OS as a console command

Purpose:
The **mqsireporttrace** command can be used to report on the current trace options that are currently set for an active broker. The broker name is a mandatory parameter, but if no other parameters are specified, the **mqsireporttrace** lists all **active** user and service trace settings for that broker. Alternatively, you can use the following options for more specific reports on the current trace settings:

- User trace. Specify the **-u** option.
- Service trace. Specify the **-t** option. Use this option only if directed to do so by the action described in a BIPxxxx message, or by your IBM Support Center.
- Trace nodes. Specify the **-n** option.

If you specify a broker, or any of its resources (execution group or message flow), you must have deployed those resources, and the broker must be running, before you can query trace settings.

Select the appropriate link for details of this command on the platform, or platforms, that your enterprise uses:

- "**mqsireporttrace** command - Windows, Linux, and UNIX systems" on page 3949
- "**mqsireporttrace** command - z/OS" on page 3951

Authorization:
For information about platform-specific authorizations, see the following topics:

- "Security requirements for Linux and UNIX platforms" on page 3648
- "Security requirements for Windows systems" on page 3651
- "Security requirements for z/OS" on page 3655

If you have enabled broker administration security, you must also set up the authority detailed in "Tasks and authorizations for broker administration security" on page 3645.

Related tasks:

"Using trace" on page 3533
You can use different types of trace to help you with problem determination and troubleshooting.

Related reference:

"User trace" on page 6873
User trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Administration log. User trace is inactive by default; you must activate it explicitly by using a command, or by selecting options in the WebSphere Message Broker Toolkit.

"Service trace" on page 6875
Service trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Event Log or to user trace. Service trace is inactive by default; you must activate it explicitly by using a command.

"Trace node" on page 4942
Use the Trace node to generate trace records that you can use to monitor the behavior of a message flow.

"**mqsichangetrace** command" on page 3822
Use the **mqsichangetrace** command to set the tracing characteristics for a broker.

"**mqsiformatlog** command" on page 3880
Use the **mqsiformatlog** command to process the XML log created by **mqsireadlog**. The command retrieves and formats any messages that the XML log contains into a form suitable for the locale of the user who runs the command.

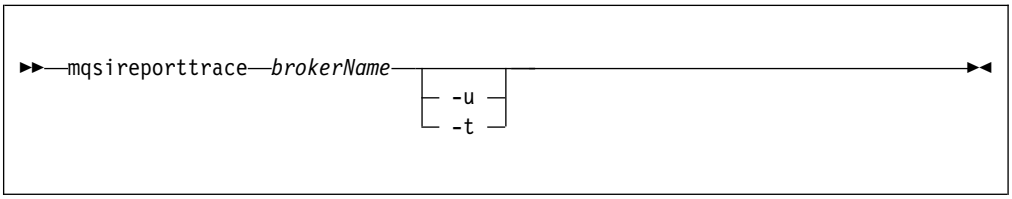
"**mqsireadlog** command" on page 3905
Use the **mqsireadlog** command to retrieve trace records for the specified component.

mqsireporttrace *command - Windows, Linux, and UNIX systems:*

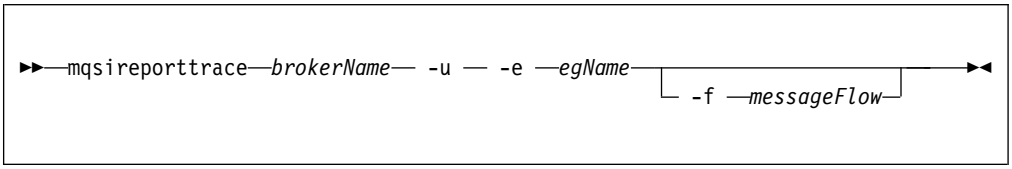
Use the **mqsireporttrace** command to display the trace options currently in effect. Trace can be run only against a broker, or any of its resources.

Syntax:

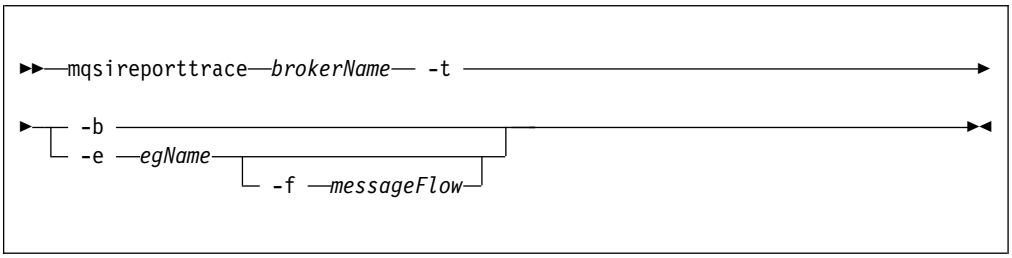
Report broker-wide active user and service trace settings:



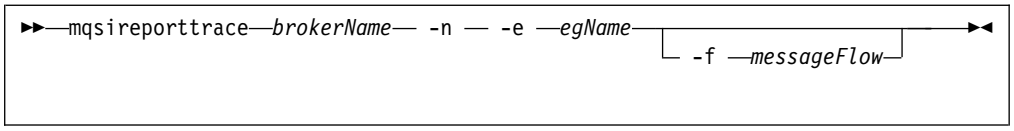
Report individual component user trace settings:



Report individual component service trace settings:



Report Trace nodes:



Parameters:

brokerName
(Required) The name of a broker; this name is case sensitive on Linux and UNIX systems.

-e *egName*
(Required for component user trace, otherwise optional) The label of the execution group for which a report is required.

-f *messageFlow*
(Optional) The label of the message flow for which a report is required. This option is valid only if you have specified an execution group.

-u (Required for component user trace, otherwise optional) Derive report information from the user trace.

Additional parameters exclusive to service trace:
Use these options only when directed to do so by your IBM Support Center, or by a `BIPxxxx` message.

-b (Alternative to **-e** on all platforms) Request a report for `agent` function.

-t (Required for component service trace, otherwise optional) Derive report information from the service trace.

Additional parameters exclusive to Trace nodes:

-n Report the setting of the Trace node switch. One BIP message is reported for each message flow.

Examples:
To derive report information from service trace for execution group exgrp1 in broker MB7BROKER, enter the command:

```
mqsireporttrace MB7BROKER -t -e "exgrp1"
```

To report the setting of the Trace node switch for execution group exgrp1 in broker MB7BROKER, enter the command:

```
mqsireporttrace MB7BROKER -n -e "exgrp1"
```

To report all active trace settings in broker MB7BROKER, enter the command:

```
mqsireporttrace MB7BROKER
```

Related reference:

"Trace node" on page 4942
Use the Trace node to generate trace records that you can use to monitor the behavior of a message flow.

"Syntax diagrams" on page 3677

"**mqsichangetrace** command" on page 3822
Use the **mqsichangetrace** command to set the tracing characteristics for a broker.

"**mqsireporttrace** command" on page 3947
Use the **mqsireporttrace** command to display the trace options currently in effect. Trace can be run only against a broker, or any of its resources.

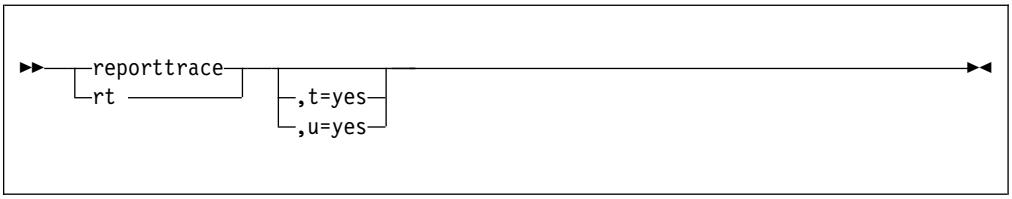
mqsireporttrace *command - z/OS:*

Use the **mqsireporttrace** command to display the trace options currently in effect.

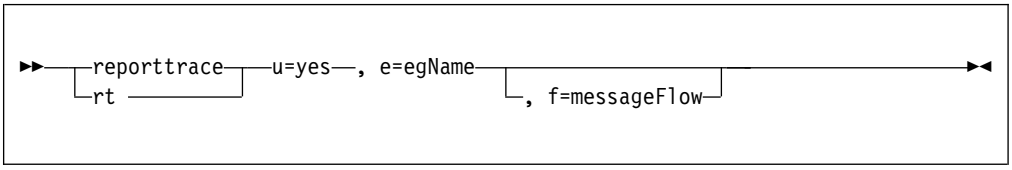
Syntax:

z/OS console command:

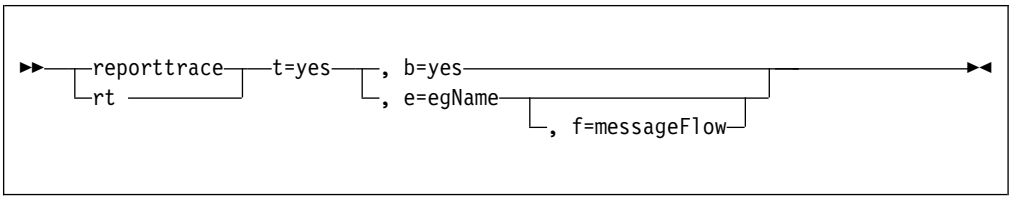
Report broker-wide active user and service trace settings:



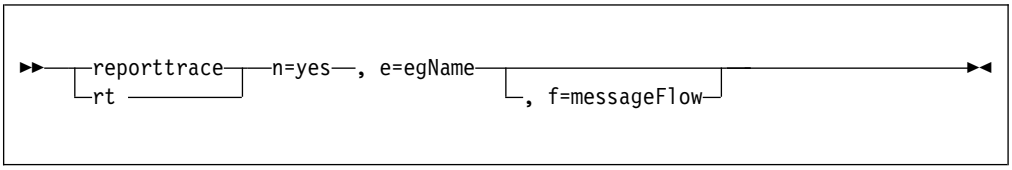
Report individual component user trace settings:



Report individual component service trace settings:



Report Trace nodes:



Parameters:

-e *egName*
(Required for component user trace, otherwise optional) The label of the execution group for which a report is required.

This name is case sensitive; include the name in single quotes if it contains mixed-case characters.

-f *messageFlow*
(Optional) The label of the message flow for which a report is required. This option is valid only if you have specified an execution group.

This name is case sensitive; include the name in single quotes if it contains mixed-case characters.

-u (Required for component user trace, otherwise optional) Derive report information from the user trace.

Additional parameters exclusive to service trace:
Use these options only when directed to do so by your IBM Support Center or by a BIPxxxx message.

-b (Alternative to **-e** on all platforms) Request a report for `agent` function.

-t (Required for component service trace, otherwise optional) Derive report information from the service trace.

Additional parameters exclusive to Trace nodes:

n=yes
Report the setting of the Trace node switch. One BIP message is reported for each message flow.

Examples:
To report information from service trace for execution group exgrp1, enter the command:

```
F MQP1BRK,rt t=yes, e='exgrp1'
```

To report the setting of the Trace node switch for execution group exgrp1, enter the command:

```
F MQP1BRK,rt n=yes, e='exgrp1'
```

To report all active trace settings in broker MQP1BRK, enter the command:

```
F MQP1BRK,rt
```

Related reference:

"Trace node" on page 4942
Use the Trace node to generate trace records that you can use to monitor the behavior of a message flow.

"Syntax diagrams" on page 3677

"**mqsichangetrace** command" on page 3822
Use the **mqsichangetrace** command to set the tracing characteristics for a broker.

"**mqsireporttrace** command" on page 3947
Use the **mqsireporttrace** command to display the trace options currently in effect. Trace can be run only against a broker, or any of its resources.

mqsirestorebroker command:

Use the **mqsirestorebroker** command to restore the broker configuration from a backup file.

Supported platforms:

- Windows
- Linux and UNIX systems
- z/OS. Run this command by customizing and submitting `BIPRSBK`; see "Contents of the broker PDSE" on page 3991.

Purpose:

The **mqsirestorebroker** command restores the broker configuration from a backup file that you have created by using the **mqsibackupbroker** command. You can restore a broker only on a computer that has an identical configuration; the operating system must be at the same level, and the broker and queue manager names must be identical.

Usage notes:
The command restores the following persistent configuration data associated with the broker that is contained in the backup file:

- Deployed resources; message flows, dictionaries, JAR files, and other runtime resources that you have previously deployed in a BAR file.
- Execution groups.
- Broker configuration; for example, configurable services.

Always stop the broker before you run this command. If you specify **-c** to restore common configuration data that is shared with other brokers, you must also stop all brokers with which this broker shares that data. If you run this command when a relevant broker is active, the results are unpredictable.

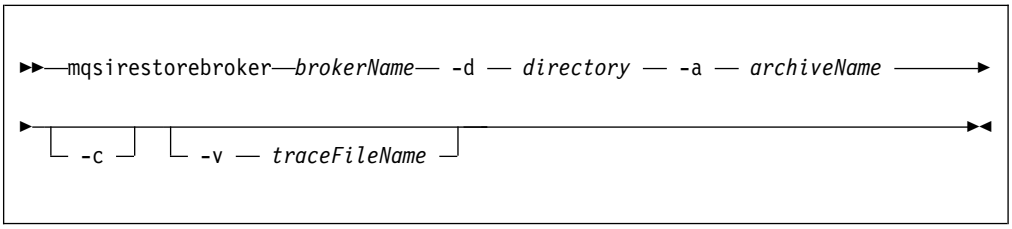
If the **mqsibackupbroker** command that created the backup file includes information about files that were locked when the backup was taken, and therefore cannot be backed up, the **mqsirestorebroker** command returns the names of the files affected.

Authorization:
For information about platform-specific authorizations, see the following topics:

- "Security requirements for Linux and UNIX platforms" on page 3648
- "Security requirements for Windows systems" on page 3651
- "Security requirements for z/OS" on page 3655

If you have enabled broker administration security, you must also set up the authority detailed in "Tasks and authorizations for broker administration security" on page 3645.

Syntax:



Parameters:

brokerName
(Required) The name of the broker that you want to restore. You must specify the name as the first parameter. If the name that you specify does not match the name of the broker associated with the backup file, the command fails.

-d *directory*
(Required) The directory in which the backup file is stored. The file must be stored in a file system that can be accessed by the computer on which you run this command.

-a *archiveName*
(Required) The name of the backup (archive) file.

-c
(Optional) Specify this parameter to restore all configuration data that is shared with other brokers on the same computer; for example, profiles.

-v *traceFileName*
(Optional) The location of a trace file that records details of the actions taken by the command.

Examples:
The following example restores broker MB7BROKER on Windows:

```
mqsirestorebroker MB7BROKER -d C:\MQSI\BACKUP -a 20090101.zip
```

Related tasks:

"Restoring the broker" on page 1015
Restore a broker configuration that you backed up previously.

"Command environment: Linux and UNIX systems" on page 310
Set up the Linux or UNIX environment to run WebSphere Message Broker commands.

"Command environment: Windows systems" on page 306
Set up the Windows environment to run WebSphere Message Broker commands.

Related reference:

"Contents of the broker PDSE" on page 3991
After you have successfully customized the broker, the broker PDSE members have been set up.

"Syntax diagrams" on page 3677

"**mqsibackupbroker** command" on page 3720
Use the **mqsibackupbroker** command to back up the current configuration of a broker.

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsideletebroker** command" on page 3863
Use the **mqsideletebroker** command to delete a named broker. The command also deletes the queues on the associated queue manager (created when the broker was created). You can also specify that the queue manager is to be deleted.

"Sample BIPRSBK file" on page 4012
The sample BIPRSBK file that is shipped with WebSphere Message Broker is included here for your reference.

mqsisetdbparms command:

Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

Supported platforms:

- Windows
- Linux and UNIX systems
- z/OS. Run this command by customizing and submitting BIPSDBP.

Purpose:
Use the **mqsisetdbparms** command to set security credentials for a specific user ID and password (or SSH identity file) for the following resources:

- A CICSConnection configurable service
- A data source name (DSN) that is accessed from a message flow
- An EmailServer configurable service
- An FtpServer configurable service
- An IMSConnect configurable service
- A JDBCProvider configurable service
- A JMS or JNDI resource, for example a JMSProviders configurable service
- Kerberos Key Distribution Center (KDC) client credentials for SOAPRequest nodes with a WS-Security policy set and bindings that specify Kerberos
- Lightweight Directory Access Protocol (LDAP) bind credentials for the broker security manager
- The Service Federation Manager (SFM) user ID and password credentials to authorize Service Control Management Protocol (SCMP) Atom requests
- An SMTP configurable service
- The broker keystore password
- An account name, with a user name and password, for the WebSphere Adapters
- A WebSphere Service Registry and Repository (WSRR) configurable service

The user ID and password pair is created in the DSN folder under the broker registry folder.

You can run the **mqsisetdbparms** command while the broker is running. However, you must stop and start each execution group that uses a particular DSN, before that information is read and used by that execution group.

This behavior is different from the default behavior in WebSphere Message Broker where the broker must be stopped to issue this command.

If you are using the **mqsisetdbparms** command on Linux or a UNIX console, add an escape character if you use one or more of the reserved characters. For example, you must specify these values:

```
mqsisetdbparms DUMMYBROKER -n ftp::DUMMYFTP -u dummy\\user -p abcdef
```

Do not use the following format:

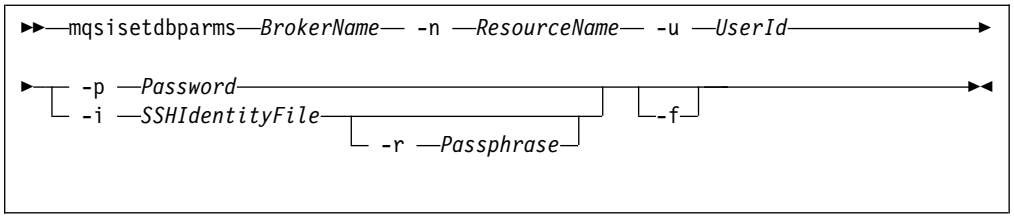
```
mqsisetdbparms DUMMYBROKER -n ftp::DUMMYFTP -u dummy\user -p abcdef
```

If you use the latter format, the backslash character (\) in the user ID or password is ignored. The example causes the FTP connection through the FileInput node to fail with incorrect user credentials.

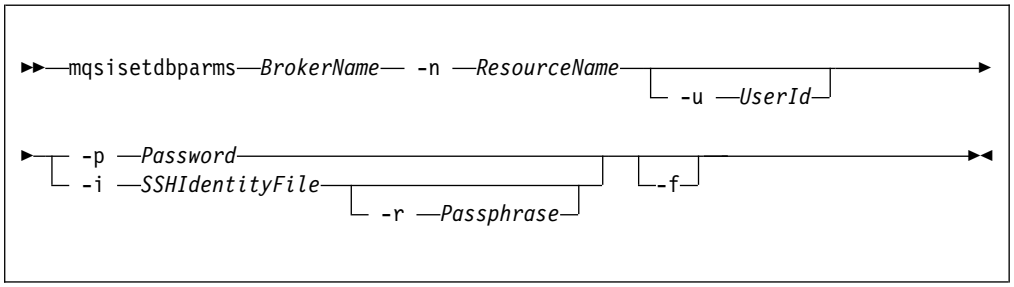
For a full list of reserved characters, and the rules that are associated with those characters when you use quotation marks and escape characters, see the documentation that is supplied with the shell.

Syntax:

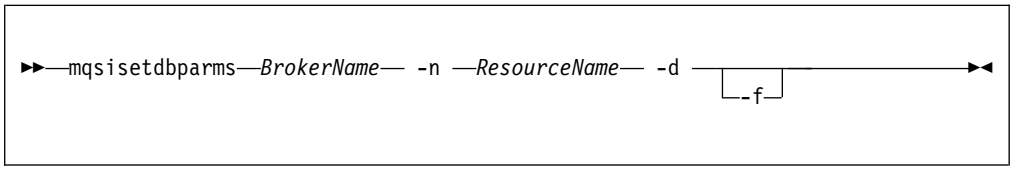
Create:



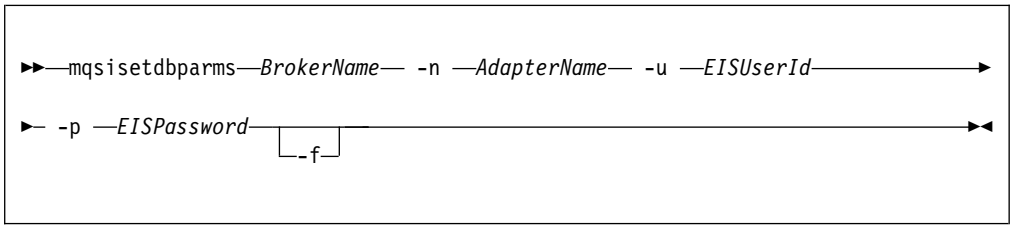
Alter:



Delete:



Adapter connection:



Parameters:

BrokerName
(Required) The name of the broker for which settings are to be created, altered, or deleted.

-n *ResourceName* **or** *AdapterName*
(Required) This parameter identifies one of the following resources:

- The ODBC data source for which the user ID and password pair are to be created or modified. The *ResourceName* takes the form `dsn::`*datasource_name*, where *datasource_name* identifies the data source name for the database to which you want to connect.

  Data source names are used by the following nodes:

  – Compute
  – Database
  – DatabaseRetrieve

- DatabaseRoute
- DataDelete
- DataInsert
- DataUpdate
- Filter
- Mapping
- Warehouse

If you use the same DSN to refer to the same database instance from multiple nodes, the same user ID and password pairing is used.

To define default values for user ID and password for the broker to use for all data source names for which you have not set specific values, specify `dsn::DSN`.

If you migrated the broker from a previous version, the values that you define on this command replace the values that you set on the **mqsicreatebroker** or **mqsichangebroker** commands before migration; the relevant parameters on those commands are deprecated in WebSphere Message Broker Version 7.0.

- The name of the security identity that is used to connect an IBM Sterling Connect:Direct CDOutput or node to itsConnect:Direct server. The *ResourceName* takes the form `cd::secId`, where `secId` is specified as the value of the security identity property on a `CDServer` configurable service. Change security identity `cd::default` to alter the default user ID and password.
- The name of the security identity that is used to authenticate a CICS Transaction Server for z/OS connection. The *ResourceName* takes the form `cics::secId`, where `secId` is specified as the value of the `Security identity` property on the CICSRequest node or in the **-n securityIdentity** property of the associated CICSConnection configurable service.
- The name of the security identity that the EmailInput node or EmailServer configurable service use to authenticate with an email server to retrieve email messages. The *ResourceName* takes the form `email::secId`, where `secId` is specified as the value of the `Security identity` property on the EmailInput node or in the **-n securityIdentity** property of the associated EmailServer configurable service.
- The name of the security identity that is used to authenticate a JDBC type 4 connection. The *ResourceName* takes the form `jdbc::secId`, where `secId` is specified as the value of the **-n securityIdentity** property of the associated JDBCProvider configurable service on the **mqsicreateconfigurableservice** or **mqsichangeproperties** command.

  Specify `jdbc::JDBC` to define default values for user ID and password for the broker to use for all JDBC connections for which you have not set specific values.
- The name of the security identity that is used to authenticate a connection to a JMS or JNDI resource. The *ResourceName* takes the form `jms::secId` or `jndi::secId`, where `secId` is specified as the value.
- The name of the security identity that is used for retrieving client credentials from the Kerberos Key Distribution Center (KDC) by a SOAPRequest node with a policy set and binding specifying Kerberos.
- The name of the security identity that is used to authenticate an LDAP directory.

Specify `ldap::<servername>` to define credentials for an individual server. If you want the broker to bind anonymously to this server, specify `anonymous` as the user ID.

Specify `ldap::LDAP` to define a default setting. The broker uses the specified user ID and password values for all servers that do not have an explicit `ldap::<servername>` entry. Therefore, all servers that previously used anonymous bind by default start to use the details defined in an `ldap::LDAP` entry.

- The name of the adapter connection to the external EIS. The *AdapterName* takes the form `eis::adapterName`, where `adapterName` is specified as the value.
- The name of the IMS connection. The *ResourceName* takes the form `ims::secId`, where `secId` is the same as the value of the `Security identity` property on the IMSRequest node or in the **-n securityIdentity** property of the associated IMSConnect configurable service.
- The name of the security identity that is used to authenticate an SMTP server.
- The name of the security identity that is used to authenticate a connection to an FTP server. The *ResourceName* takes the form `ftp::secId`, where `secId` is specified as the value of the `Security identity` property of the FileInput or FileOutput node, or in the **-n securityIdentity** property of the associated FtpServer configurable service on the **mqsicreateconfigurableservice** or **mqsichangeproperties** command.
- The name of the security identity that is used to authenticate a connection to an SFTP server. The security identity is used to locate the user name and password or the Secure Shell (SSH) identity file. The *ResourceName* takes the form `sftp::secId`, where `secId` is specified as the value of the `Security identity` property of the FileInput or FileOutput node, or in the **-n securityIdentity** property of the associated FtpServer configurable service on the **mqsicreateconfigurableservice** or **mqsichangeproperties** command.
- The name of the security identity that is used to authenticate a broker keystore.
- The name of the security identity that is used to authenticate a WSRR configurable service.
- The name of the security identity for SFM. Specify `sfm::scmp` to define the basic access authentication credentials that must be present in every SCMP Atom request that is received on the SFM HTTP(S) port.

  The user ID and password that are configured in this way must match the user ID and password that the SFM console can provide. It is good practice to configure SFM to use SSL when you set basic access authentication.

-u *UserId* **or** *EISUserId*
(Required for Create and adapter connection; Optional for Alter) The user ID to be associated with this resource or EIS.

-p *Password*
(Required for Create, Alter, and adapter connection) The password to be associated with this resource or EIS.

For compatibility with existing systems, you can still specify `<password>`. However, if you do not specify a password with this parameter when you run the command, you are prompted to enter a password during its invocation, and to enter the password a second time to verify that you have entered it correctly.

On z/OS only, this parameter is optional with the `dsn::DSN` resource type. If you omit this parameter, the broker uses the started task user ID to connect to DB2. The broker uses the user ID that you specified with the **-u** parameter when it constructs fully qualified SQL statements; for example, for stored procedures. If you create fully qualified SQL statements, the broker uses these statements as created.

This parameter is required with the `ftp::` resource type, but is optional with the `sftp::` resource type. However, if you do not specify a password with an `sftp::` resource, you must specify the *SSHIdentityFile* parameter.

-i *SSHIdentityFile*
(Optional) The name of an identity file, in the OpenSSH format, to be used for authentication with SFTP, in place of a password. You must specify either a password or an identity file, but not both. If you specify an identity file, you can also specify a pass phrase with the *Passphrase* parameter.

On z/OS systems, known hosts files and SSH identity files are stored in EBCDIC format, and on other operating systems they are stored in ASCII format.

-r *Passphrase*
(Optional) The pass phrase that is used for authentication with SFTP. This parameter is valid only when the *SSHIdentityFile* parameter is also specified. The pass phrase is used during decryption of the identity file.

-d (Required for Delete) This parameter deletes completely the resource from the broker registry.

-f (Optional) Specify this parameter to process the **mqsisetdbparms** command only when the broker itself is stopped.

Authorization:
For information about platform-specific authorizations, see the following topics:

- "Security requirements for Linux and UNIX platforms" on page 3648
- "Security requirements for Windows systems" on page 3651
- "Security requirements for z/OS" on page 3655

If you have enabled broker administration security, you must also set up the authority detailed in "Tasks and authorizations for broker administration security" on page 3645.

Ensure that the registry is appropriately secured to prevent unauthorized access.

Examples:
CICS connections

Use the **mqsisetdbparms** command in the following format to associate a user ID and password pair with CICS.

```
mqsisetdbparms broker name -n ResourceName -u userID -p password
```

For example:

```
mqsisetdbparms MB7BROKER -n cics::mySecurityIdentity -u myUserID -p myPassword
```

Data source names

The following example show use of the command for a specific data source name (no Universal Record Identifier (URI) prefix is required for this purpose):

```
mqsisetdbparms MB7BROKER -n Database1 -u MQUserId -p password
```

The following example deletes all the values that are defined for a specific data source name from the broker registry:

```
mqsisetdbparms MB7BROKER -n ClientDB -d
```

The following example shows how to set up a default user ID and password for the broker to use for all data source names where no explicit values have been set:

```
mqsisetdbparms MB7BROKER -n dsn::DSN -u UserId1 -p password1
```

The following examples show the use of the additional prefix `odbc::`. Use this option to set the user ID and password at the execution group level, and at the broker level:

```
mqsisetdbparms MB7BROKER -n odbc::DSN -u myuserid
  -p mypassword
mqsisetdbparms MB7BROKER -n odbc::DSN::default -u myuserid -p mypassword
```

Email server connections

Use the **mqsisetdbparms** command in the following format to associate a user ID and password pair with an email server for the EmailInput node or the EmailServer configurable service to use to retrieve email messages.

```
mqsisetdbparms broker name -n ResourceName -u userID -p password
```

For example:

```
mqsisetdbparms MB7BROKER -n email::mySecurityIdentityObjectName
-u myUserID -p myPassword
```

IBM Sterling Connect:Direct

Use the **mqsisetdbparms** command in the following format to associate a user ID and password pair with a Connect:Direct server.

```
mqsisetdbparms broker name -n ResourceName -u userID -p password
```

For example:

```
mqsisetdbparms MB7BROKER -n cd::default -u mqbroker -p xxxxxxx
```

JDBC type 4 connections

Use the **mqsisetdbparms** command to associate a user ID and password pair with a JDBC type 4 connection. The value that you specify for the **-n ResourceName** must have a prefix of `jdbc::`, followed by the value that matches the **-n securityIdentity** property of the associated JDBCProvider configurable service.

```
mqsisetdbparms broker name -n resource_name -u userID -p password
```

For example:

```
mqsisetdbparms MB7BROKER -n jdbc::mySecurityIdentity -u myuserid -p secretpw
```

The following example shows how to set up a default user ID and password for the broker to use for all JDBC connections for which you have not set specific values:

```
mqsisetdbparms MB7BROKER -n jdbc::JDBC -u UserId2 -p password2
```

JMS and JNDI resource names

The following examples show the use of the command when the URI for a JMS or JNDI resource name is substituted for the **-n ResourceName** parameter.

For a JMS resource, the URL prefix is "jms::"; for JNDI, the prefix is "jndi::".

On Linux and UNIX systems, if the parameter string includes a backslash (\) character, you must escape from this character by using a second backslash character (\\) when you enter the **mqsisetdbparms** command.

For example, to specify a user ID of myuserid and password secret for JMS topic connection factory tcf1, use the following syntax:

```
mqsisetdbparms MB7BROKER -n jms::tcf1 -u myuserid -p secret
```

Similarly, to specify the same security for a JNDI initial context com.sun.jndi.fscontext.RefFSContextFactory, enter the following command:

```
mqsisetdbparms MB7BROKER -n jndi::com.sun.jndi.fscontext.RefFSContextFactory
     -u myuserid -p secret
```

JMS node account names

The preceding examples describe how to configure security for JMS and JNDI resources for *all* JMS nodes that use those resources in a broker.

To increase the degree of control that you have in the security of JMS nodes, you can associate a resource with an *account name*. The account name comprises the message flow name that is concatenated with the node label by using the underscore character (_):

Message Flow Name_Node label

For example, where the message flow name is MyJMSFlow1, and you require a specific user ID and password for JMSInput node MyJMSInput1, the resulting account name is:

MyJMSFlow1_MyJMSInput1

You can then specify the account name string in the **-n ResourceName** parameter on the **mqsisetdbparms** command by prefixing the account name with the *resource type*, and concatenating the account name with an at sign (@) character followed by the resource name:

resource typeaccount name@resource name

Therefore, assuming a JMS resource name of tcf1, used by JMSInput node MyJMSInput1 in message flow MyJMSFlow1, the following resource name is used:

jms::MyJMSFlow1_MyJMSInput1@tcf1

To specify a user ID of myuserid, a password of secret, and the resource name that is created from the account name, use the following syntax:

```
mqsisetdbparms MB7BROKER -n jms::MyJMSFlow1_MyJMSInput1@tcf1
                 -u myuserid -p secret
```

LDAP servers

Use the **mqsisetdbparms** command to set up authorization credentials for the LDAP server ldap.mydomain.com:

```
mqsisetdbparms MB7BROKER -n ldap::ldap.mydomain.com -u ldapuid -p ********
```

To set up authorization for other servers, use the command to set up default credentials:

```
mqsisetdbparms MB7BROKER -n ldap::LDAP -u ldapother -p ********
```

If you want the broker to bind anonymously to an LDAP server, specify the server name and the user ID `anonymous`:

```
mqsisetdbparms MB7BROKER -n ldap::ldap.mydomain2.com -u anonymous -p ********
```

For the user ID `anonymous`, the password is always ignored.

WebSphere Adapters account names

Use the **mqsisetdbparms** command in the following format to configure an account name with a user name and password for the WebSphere Adapters.

```
mqsisetdbparms broker name -n adapter name -u user name -p password
```

For example:

```
mqsisetdbparms MB7BROKER  -n eis::SAPCustomerInbound.inadapter -u sapuid -p ********
mqsisetdbparms MB7BROKER  -n eis::TwineballInbound.inadapter -u mqbroker -p ********
```

IMS connections

Use the **mqsisetdbparms** command in the following format to associate a user ID and password pair with an IMS Connect connection.

```
mqsisetdbparms broker name -n resource_name -u userID -p password
```

For example:

```
mqsisetdbparms MB7BROKER -n ims::mySecurityIdentity -u myuserid -p mypassword
```

FTP and SFTP server connections

Use the **mqsisetdbparms** command to associate a user ID and password with an FTP server connection:

```
mqsisetdbparms MB7BROKER -n ftp::identityA -u user1 -p MyPassword
```

Use the **mqsisetdbparms** command to associate a user ID and password with an SFTP server connection:

```
mqsisetdbparms MB7BROKER -n sftp::identityB -u user2 -p MyPassword
```

Use the **mqsisetdbparms** command to associate a user ID and SSH identity file with an SFTP server connection:

```
mqsisetdbparms MB7BROKER -n sftp::identityC -u user3 -i C:\key_rsa_no_pp
```

Use the **mqsisetdbparms** command to associate a user ID, SSH identity file, and pass phrase with an SFTP server connection:

```
mqsisetdbparms MB7BROKER -n sftp::identityD -u user4 -i C:\key_rsa_pp -r MyPassPhrase
```

Service Federation Management

Use the **mqsisetdbparms** command to set the SCMP enablement to be secured with basic access authentication by using a user ID and password:

```
mqsisetdbparms MB7BROKER  -n sfm::scmp -u user1 -p MyPassword
```

Use the **mqsisetdbparms** command to clear any previous settings and specify that the SCMP enablement does not require basic access authentication:

```
mqsisetdbparms MB7BROKER -n sfm::scmp -d
```

Kerberos

Use the **mqsisetdbparms** command to provide the broker with the Kerberos client credentials for accessing the Kerberos Key Distribution Center (KDC). These credentials (which are required for SOAPRequest nodes) can also be provided in the broker properties tree.

To set KDC credentials for a specific realm that is used by SOAPRequest nodes in a specific execution group:

```
mqsisetdbparms MB7BROKER -n kerberos::realm1::ExecutionGroup1 -u clientId -p ClientPassword
```

To set KDC credentials for a specific realm that is used by SOAPRequest nodes in any execution group:

```
mqsisetdbparms MB7BROKER -n kerberos::realm1 -u clientId -p ClientPassword
```

To set KDC credentials for any realm that is used by SOAPRequest nodes in any execution group:

```
mqsisetdbparms MB7BROKER -n kerberos::kerberos -u clientId -p ClientPassword
```

Related tasks:

"Accessing a secure WSRR repository" on page 1884
To access a secure WebSphere Service Registry and Repository (WSRR) repository, set the configuration parameters by using the **mqsichangeproperties** command.

"Configuring authorization with LDAP" on page 472
This topic describes how to configure a message flow to perform authorization on an identity using Lightweight Directory Access Protocol (LDAP).

"Setting up a JDBC provider for type 4 connections" on page 684
Use the **mqsicreateconfigurableservice** or the **mqsichangeproperties** command to configure a JDBC provider service.

"Securing a JDBC type 4 connection" on page 689
Set up security for the JDBC connection if required by the database provider.

"Viewing and setting keystore and truststore runtime properties at broker level" on page 780
Configure the message broker to refer to a keystore, a truststore, or both, before deploying any message flows that require policy set or bindings for signature, encryption, or X.509 Authentication.

Related reference:

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

"CICSRequest node" on page 4321
Use the CICSRequest node to call CICS Transaction Server for z/OS programs over TCP/IP-based IP InterCommunications (IPIC) protocol.

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"EmailInput node" on page 4394
Use the EmailInput node to retrieve an email, with or without attachments, from an email server that supports Post Office Protocol 3 (POP3) or Internet Message Access Protocol (IMAP).

"FileInput node" on page 4415
Use the FileInput node to process messages that are read from files.

"FileOutput node" on page 4430
Use the FileOutput node to write messages to files.

"PeopleSoftInput node" on page 4630
Use the PeopleSoftInput node to interact with a PeopleSoft application.

"PeopleSoftRequest node" on page 4635
Use the PeopleSoftRequest node to interact with a PeopleSoft application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

"TwineballRequest node" on page 4955
Use the TwineballRequest node to discover out how WebSphere Adapters nodes work.

"Syntax diagrams" on page 3677

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"FtpServer configurable service properties" on page 3794
Select the properties and values that you want to change for an existing FtpServer configurable service or to create a new service.

mqsisetsecurity command:

Use the **mqsisetsecurity** command to create the Windows groups that WebSphere Message Broker requires for secure access to its runtime libraries and data.

Supported platforms:

- Windows

Purpose:
The **mqsisetsecurity** command runs automatically as part of the installation process of the Broker component.

The installation wizard calls the **mqsisetsecurity** command which creates a new security group called mqbrkrs, adds your current (logged on) user ID to the group mqbrkrs, and adds your current user ID to the group mqm, if that group exists. If WebSphere MQ is installed after WebSphere Message Broker, you can issue this command to add your account to the group mqm.

Syntax:

Parameters:
None

Authorization:
For information about platform-specific authorizations, see the following topics:

- "Security requirements for Linux and UNIX platforms" on page 3648
- "Security requirements for Windows systems" on page 3651

If you have enabled broker administration security, you must also set up the authority detailed in "Tasks and authorizations for broker administration security" on page 3645.

In addition, this command is supported on Linux and UNIX systems where it only creates the group mqbrkrs.

Related concepts:

"Security overview" on page 351
When you are designing a WebSphere Message Broker application it is important to consider the security measures that are needed to protect the information in the system.

Related reference:

"Security requirements for Windows systems" on page 3651
Security requirements depend on the administrative task that you want to perform.

`mqsistart` command:

Use the **`mqsistart`** command to start the specified broker if all initial verification tests complete successfully.

Supported platforms:

- Windows
- Linux and UNIX systems
- z/OS as a console command.

Purpose:
The **`mqsistart`** command:

- On distributed systems, starts the associated queue manager if it is not running.
- Checks that the broker environment is set up correctly; for example, that the installed level of Java is supported.
- Verifies that the WebSphere MQ queues are defined and accessible.

  The command completes these checks, and reports all errors in the system log, or to the command line, or both. If one or more checks fail, the command does not start the component.
- On Linux and UNIX systems, where the **`mqsisetdbparms`** command has been used to associate the datasource with the broker, the **`mqsistart`** command:
  - Checks broker connectivity to, and
  - Performs tests against

  any user data sources where the WebSphere Message Broker Database Extender (IE02) SupportPac is being used for extended database support.

  If any issues are identified, a warning is reported to the system log and command line. However, this step does not prevent the broker from starting cleanly.
- Checks the publish/subscribe mode (attribute PSMODE) of the queue manager associated with the broker. If the PSMODE is set to DISABLED, it is changed to

COMPAT and this is reported to the system log. If the PSMODE is already set to COMPAT or ENABLED, the setting is not changed.

Select the appropriate link for details of this command on the platform, or platforms, that your enterprise uses:

- "**mqsistart** command - Windows, Linux, and UNIX platforms"
- "**mqsistart** command - z/OS" on page 3968

Authorization:
For information about platform-specific authorizations, see the following topics:

- "Security requirements for Linux and UNIX platforms" on page 3648
- "Security requirements for Windows systems" on page 3651
- "Security requirements for z/OS" on page 3655

If you have enabled broker administration security, you must also set up the authority detailed in "Tasks and authorizations for broker administration security" on page 3645.

The broker starts only if the user ID under which it will run is authorized to access both the following locations. On Windows, the user ID is defined by the *ServiceUserID* specified on **mqsicreatebroker**; on Linux and UNIX systems, it is the user ID with which you run this command:

- Home directory, where WebSphere Message Broker has been installed.
- Working directory, if specified by the **-w** flag on the **mqsicreatebroker** command.

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related reference:

"Security requirements for Linux and UNIX platforms" on page 3648
View a summary of the authorizations in a Linux or UNIX environment.

"Security requirements for Windows systems" on page 3651
Security requirements depend on the administrative task that you want to perform.

"Security requirements for z/OS" on page 3655
View a summary of the authorizations in a z/OS environment.

"Syntax diagrams" on page 3677

"**mqsicvp** command" on page 3857
Use the **mqsicvp** command to perform verification tests on a broker, or to verify ODBC connections.

"**mqsistop** command" on page 3972
Use the **mqsistop** command to stop the specified component.

mqsistart *command - Windows, Linux, and UNIX platforms:*

Use the **mqsistart** command to start a broker.

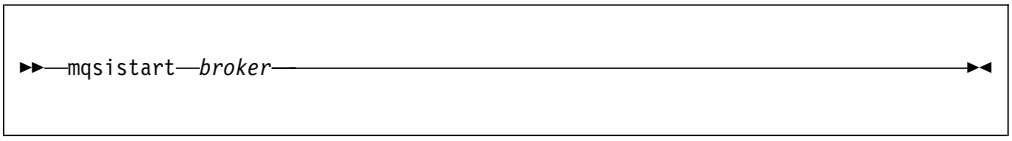
Purpose:
If the queue manager associated with this broker (defined by the **mqsicreatebroker** command) is not already running, it is also started by this command. However, no listeners, channels, or channel initiators associated with the started broker are started. Use WebSphere MQ Explorer to start any required listeners, channels, or channel initiators.

Successful completion of this command indicates that the Windows service, or Linux or UNIX daemon, has started successfully, and that the broker startup has been initiated. Check the Windows system event log or the Linux or UNIX syslog to determine if the broker and all related software have started successfully, are initially active, and remain in an active state.

All errors that have prevented successful startup, that are detected by the broker, are recorded in the log. Continue to monitor the Windows system event log or Linux or UNIX syslog.

On Windows platforms, the queue manager is not started as a service, and stops if you log off. To avoid this happening, either remain logged on, or change the startup status of the queue manager service as described in "Starting a WebSphere MQ queue manager as a Windows service" on page 929. (If you lock your workstation, the queue manager does not stop).

Syntax:



Parameters:

broker
(Required) Specify the name of the broker that you want to stop.

On Linux and UNIX systems, all names are case sensitive.

Responses:

- BIP8012 Unable to connect to system components
- BIP8013 Broker does not exist
- BIP8015 Broker cannot be started
- BIP8018 Broker running
- BIP8024 Unable to locate executable
- BIP8025 Broker disabled
- BIP8026 Unable to start broker
- BIP8027 Unable to start WebSphere MQ
- BIP8028 WebSphere MQ unavailable
- BIP8030 Unable to modify user privileges
- BIP8048 Unable to start queue manager
- BIP8056 Unknown queue manager
- BIP8093 Queue manager being created
- BIP8094 Queue manager stopping
- BIP8116 Starts a broker

Examples:
To start the broker MB7BROKER:

```
mqsistart MB7BROKER
```

Related reference:

"Syntax diagrams" on page 3677

"**mqsistart** command" on page 3965
Use the **mqsistart** command to start the specified broker if all initial verification tests complete successfully.

"**mqsistop** command" on page 3972
Use the **mqsistop** command to stop the specified component.

mqsistart *command - z/OS:*

Use the **startcomponent** command to start a broker when its controller (control process) is already running, or the **start (/S)** command to bring a broker into a state in which you can run the appropriate **change** command.

Purpose:
The following table distinguishes between the **start (/S)** and **startcomponent** commands, and lists the available options.

Command	Description
/S *Broker started task name*	Starts the broker.
/F *Broker started task name*,SC	Starts the broker from a 'stop component' state.

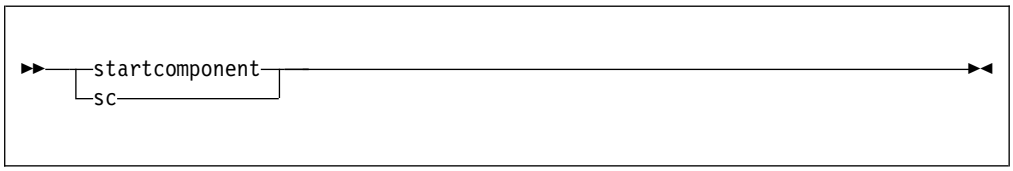
When the controller address space is started, the broker starts automatically. You can change this behavior by using an optional start parameter in the started task. If you set the parameter to MAN, the broker does not start automatically; the default is AUTO.

Issuing commands against the controller means issuing start, stop, or modify commands from the console to the controller address space. Two scenarios apply when you use this command:

1. The controller is started with the parameter MAN instead of AUTO.
2. After a **stopcomponent** command, you must restart the broker.

Syntax:

z/OS console command - startcomponent:
Synonym: sc



Examples:

```
/F MQ00BRK,sc
```

Related tasks:

"Starting and stopping a broker" on page 921
Run the appropriate command to start or stop a broker.

Related reference:

"Syntax diagrams" on page 3677

"**mqsistart** command" on page 3965
Use the **mqsistart** command to start the specified broker if all initial verification

tests complete successfully.

"**mqsistop** command" on page 3972
Use the **mqsistop** command to stop the specified component.

mqsistartmsgflow command:

Use the **mqsistartmsgflow** command to start execution groups and message flows.

Supported operating systems:

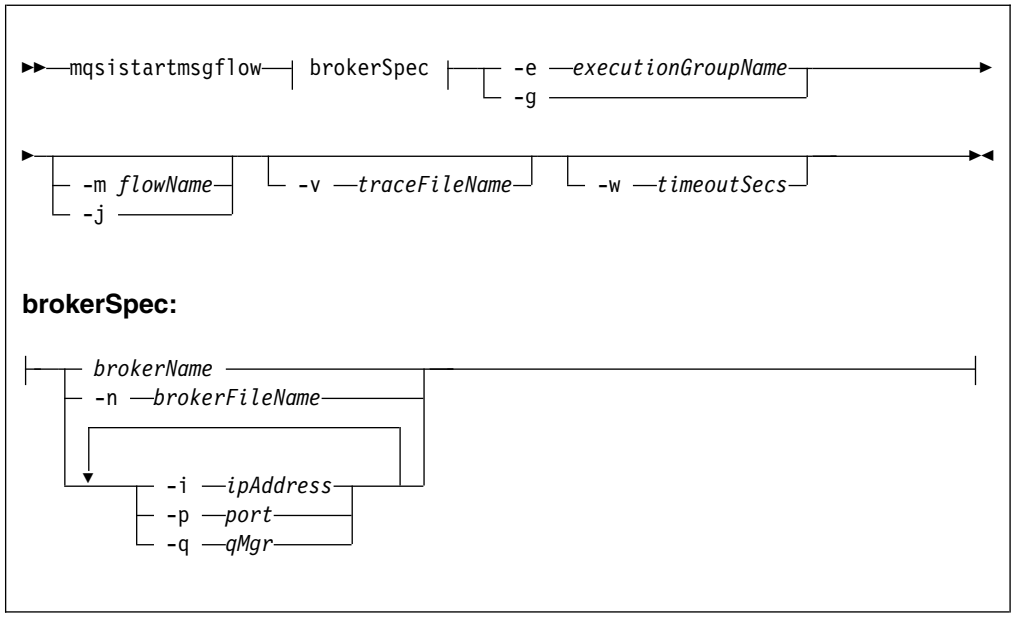
- Windows
- Linux and UNIX systems
- z/OS. Run this command by customizing and submitting BIPSTMF; see "Contents of the broker PDSE" on page 3991

Purpose:
You can use the **mqsistartmsgflow** command for the following purposes:

- To start a specific message flow in an execution group
- To start all message flows in the execution group
- To start a specific execution group
- To start all execution groups

To use this command, you must have already deployed message flows, if specified, to the broker in a broker archive (BAR) file. You can start message flows only if the execution group to which the message flow is deployed is running.

Syntax:



Parameters:

brokerSpec
(Required) You must specify at least one parameter to identify the target broker for this command, in one of the following forms:

brokerName
This parameter identifies the name of a locally defined broker. You cannot use this option if the broker is on a remote computer.

-n *brokerFileName*
This parameter identifies the name of a file that contains the connection details for a local or remote broker.

Use this option if multiple users want to connect to the same broker, or if you are using advanced connection parameters such as SSL.

To create this file, right-click the broker in the WebSphere Message Broker Explorer and select **Export *.broker file**. When prompted, navigate to the location in which you want to save the file and enter the file name; the extension `.broker` is appended automatically when you click **Save**. Include the location (path) and file name when you specify this parameter. You must ensure that the location is accessible when you run this command.

If you want to run a command that uses SSL to administer a remote broker over a secured channel, you must specify the keystore and truststore password for the connection using the IBM_JAVA_OPTIONS environment variable. See "Resolving problems when running commands" on page 3364 for further information.

-i *ipAddress*, **-p** *port*, **-q** *qMgr*
These parameters identify the connection details for the broker.

Use this option for connections to remote brokers that do not require advanced connection parameters.

If you choose this option, you must specify at least one of these three parameters; the order in which you specify them is insignificant. You cannot specify each parameter more than once.

Parameters that you omit assume default values:

- **-i** *ipAddress*: The host name or the IP address of the computer on which the broker is running. If you do not specify this parameter, a value that represents the local computer is used.
- **-p** *port*: The TCP port on which the broker's queue manager is listening. If you do not specify this parameter, the value 1414 is used.
- **-q** *qMgr*: The name of the broker's queue manager. If you do not specify this parameter, a value that represents the default queue manager on the local computer is used.

You cannot connect to a remote queue manager on z/OS; see "Usage note" on page 3971.

-e *executionGroupName*
(Optional) The name of the execution group for which to start message flows. You must specify either **-e** or **-g**.

If you specify **-e** without **-m** or **-j**, the execution group is started. Message flows are started only if they were running when the execution group was stopped.

If you specify **-e** with **-m** or **-j**, the execution group must be running. If the execution group is stopped, the request is rejected.

-g (Optional) The specified message flow or flows are started on all execution groups on the specified broker. You must specify either **-e** or **-g**.

If you specify **-g** without **-m** or **-j**, all execution groups are started, but message flows are started only if they were running when the execution group was stopped.

If you specify **-g** with **-m** or **-j**, the specified message flow or flows are started only if the execution group is running. The request checks all execution groups, and starts message flows on execution groups that are currently running. Error BIP2851 is reported for each execution group that is not running.

-j

(Optional) All message flows in the specified execution group are started.

If you do not specify **-j** or **-m**, the execution group identified by **-e** (if specified), or all execution groups on this broker, are started. Message flows that were running when the execution group was last stopped are also restarted.

-m *flowName*
(Optional) The name of the message flow being started.

If you do not specify **-m** or **-j**, the execution group identified by **-e** (if specified), or all execution groups on this broker, are started. Message flows that were running when the execution group was last stopped are also restarted.

-v *traceFileName*
(Optional) This parameter sends internal debug trace information to the specified file.

-w *timeoutSecs*
(Optional) This parameter specifies the time in seconds that the utility waits to ensure that the command completed; the default value is 60.

Usage note:
If you try to connect to a remote broker by specifying the **i**, **p**, and **q** parameters, or by using a connection parameter file (`.broker`), the command attempts to use WebSphere MQ Java client code. This option is not supported on z/OS, and returns the following error:

```
BIP1046E: Unable to connect with the broker (name)
```

The reported reason code is `MQRC_ENVIRONMENT_ERROR`. You must specify a local queue manager.

Authorization:
For information about platform-specific authorizations, see the following topics:

- "Security requirements for Linux and UNIX platforms" on page 3648
- "Security requirements for Windows systems" on page 3651
- "Security requirements for z/OS" on page 3655

If you have enabled broker administration security, you must also set up the authority detailed in "Tasks and authorizations for broker administration security" on page 3645.

Responses:
This command returns the following responses:

0 The command completed successfully.

2 (Failure) The broker received the deployment request but was unable to

process it successfully. See the messages issued from the utility (or the Administration log) for more information.

9 (Failure) The request has been submitted to the broker, but no response was received before the timeout expired.

10 (Failure) Another user or application canceled the request operation before the broker was able to process it.

98 The broker is not running.

99 One or more of the parameters that you specified is invalid.

Examples:
Start a named message flow that you have deployed to execution group `eg1` on the broker MB7BROKER:

```
mqsistartmsgflow MB7BROKER -e eg1 -m simpleflow
```

Ensure that all message flows are running on the local broker MB7BROKER:

```
mqsistartmsgflow MB7BROKER -g -j
```

:

Related tasks:

"Enabling broker administration security" on page 368
Enable broker administration security on a broker to control which users can complete specific tasks against that broker and its resources.

Chapter 11, "Packaging and deploying," on page 3209
Package resources that you create in the WebSphere Message Broker Toolkit, such as message flows, and deploy them to execution groups on brokers.

Related reference:

"**mqsideploy** command" on page 3872
Use the **mqsideploy** command to make a deployment request to the broker.

"**mqsistopmsgflow** command" on page 3975
Use the **mqsistopmsgflow** command to stop execution groups and message flows.

"Syntax diagrams" on page 3677

mqsistop command:

Use the **mqsistop** command to stop the specified component.

Supported platforms:

- Windows
- Linux and UNIX systems
- z/OS as a console command

Select the appropriate link for details of this command on the platform, or platforms, that your enterprise uses:

- "**mqsistop** command - Windows, Linux, and UNIX systems" on page 3973
- "**mqsistop** command - z/OS" on page 3974

Authorization:
For information about platform-specific authorizations, see the following topics:

- "Security requirements for Linux and UNIX platforms" on page 3648
- "Security requirements for Windows systems" on page 3651

- "Security requirements for z/OS" on page 3655

If you have enabled broker administration security, you must also set up the authority detailed in "Tasks and authorizations for broker administration security" on page 3645.

Related reference:

"Syntax diagrams" on page 3677

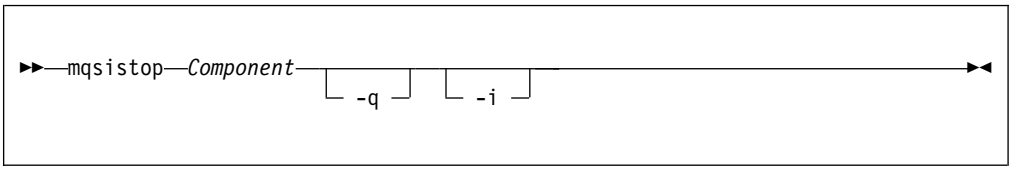
"**mqsistart** command" on page 3965
Use the **mqsistart** command to start the specified broker if all initial verification tests complete successfully.

mqsistop *command - Windows, Linux, and UNIX systems:*

How to use the **mqsistop** command on Windows, Linux, and UNIX systems.

Purpose:
Use the **mqsistop** command to stop a WebSphere Message Broker component.

Syntax:



Parameters:

Component
(Required) Specify the name of the broker that you want to stop.

On Linux and UNIX systems, all names are case sensitive.

-q (Optional) Stops the WebSphere MQ queue manager associated with this component.

Specify this flag only if the WebSphere Message Broker component is the last (or only) WebSphere Message Broker component active on this queue manager. The **mqsistop** command initiates a controlled shutdown of the queue manager, and informs other users of the queue manager that it is closing.

Stop other WebSphere Message Broker components that use this queue manager before you issue the **mqsistop** command with this option; alternatively stop them afterward or restart the queue manager.

If you use this option, be aware that any listeners associated with this queue manager are not stopped with the queue manager. Stop these listeners manually after issuing the **mqsistop** command.

-i (Optional) Immediately stops the broker.

Specify this flag only if you have already tried, and failed, to stop the broker in a controlled fashion by using the **mqsistop** command without the **-i** flag.

Responses:

- BIP8012 Unable to connect to system components
- BIP8013 Component does not exist
- BIP8016 Component cannot be stopped

- BIP8019 Component stopped
- BIP8030 Unable to modify user privileges
- BIP8049 Unable to stop queue manager
- BIP8093 Queue manager being created
- BIP8094 Queue manager stopping

Examples:
To stop the broker, *mybroker*, and the WebSphere MQ queue manager associated with it:

```
mqsistop mybroker -q
```

Related reference:

"Syntax diagrams" on page 3677

"**mqsistart** command" on page 3965
Use the **mqsistart** command to start the specified broker if all initial verification tests complete successfully.

"**mqsistop** command" on page 3972
Use the **mqsistop** command to stop the specified component.

mqsistop *command - z/OS:*

Use the **mqsistop** command to stop a WebSphere Message Broker broker; the controller must be running.

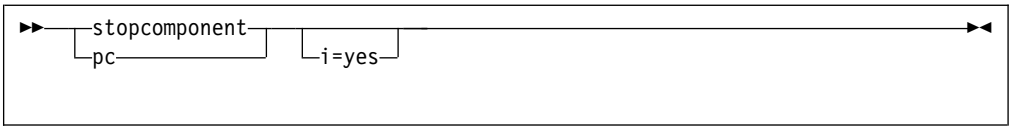
Purpose:
The following table distinguishes between the **stop (/P)** and **stopcomponent** commands, and lists the available options:

- Use the **stop (/P)** command to bring a component into a state in which you can run the appropriate **change** command.
- Use the **stopcomponent** command to stop a broker when its controller (control process) is already running.

Component	Command	Description
Broker	/P <Broker started task name> /F <Broker started task name>,P /F <Broker started task name>,PC	Stop broker. Stop broker; this is the same as /P. You can also use /F <broker started task name>,STOP Stop broker component. This stops the broker process (including any execution group address spaces), but leaves the message broker console command server running inside the controller address space. This allows you to run the "**mqsichangebroker** command" on page 3723 console command. Restart the broker afterward by running the "**mqsistart** command" on page 3965 (SC) console command.

Syntax:

z/OS console command - stopcomponent:
Synonym: pc



Parameters:

-i (Optional) Immediately stop the broker.

Specify this flag only if you have already tried, and failed, to stop the broker in a controlled fashion by using the **mqsistop** command without the **-i** flag.

This command is rejected by the WebSphere Message Broker console command server if a previous stop command has failed to complete. This can occur, for example, if one or more execution group address spaces cannot shutdown.

Examples:

```
F MQ00BRK,pc
```

Related reference:

"Syntax diagrams" on page 3677

"**mqsistart** command" on page 3965
Use the **mqsistart** command to start the specified broker if all initial verification tests complete successfully.

"**mqsistop** command" on page 3972
Use the **mqsistop** command to stop the specified component.

mqsistopmsgflow command:

Use the **mqsistopmsgflow** command to stop execution groups and message flows.

Supported operating systems:

- Windows
- Linux and UNIX systems
- z/OS. Run this command by customizing and submitting BIPSPMF; see "Contents of the broker PDSE" on page 3991

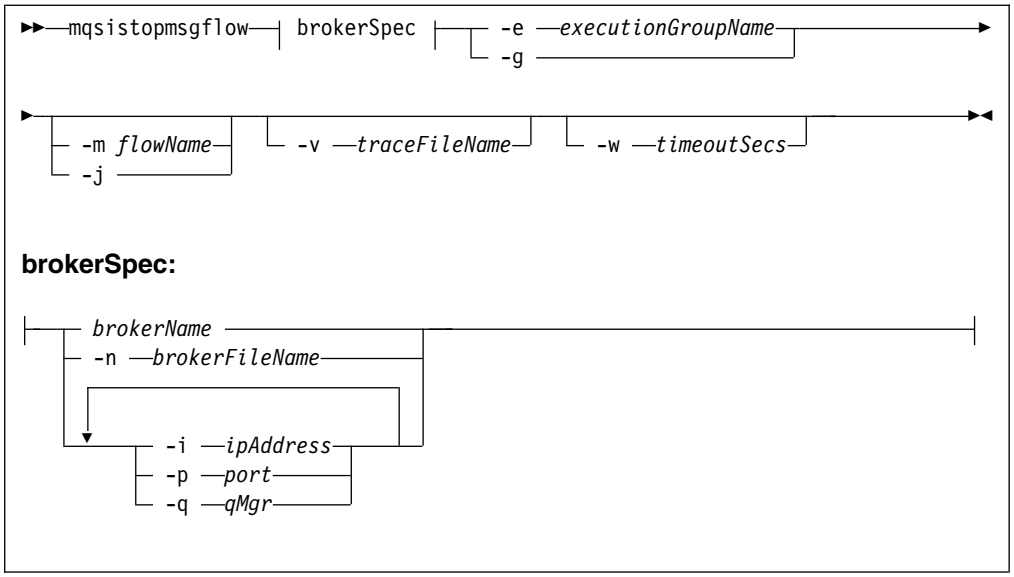
Purpose:
You can use the **mqsistopmsgflow** command for the following purposes:

- To stop a specific message flow in an execution group
- To stop all message flows in the execution group
- To stop a specific execution group
- To stop all execution groups

To use this command, you must have already deployed message flows, if specified, to the broker in a broker archive (BAR) file.

The broker processes all inflight messages and associated transactions for each message flow before stopping it. For information about how outstanding units of work are handled in this situation, see "Message flow transactions" on page 1281.

Syntax:



Parameters:

brokerSpec
(Required) You must specify at least one parameter to identify the target broker for this command, in one of the following forms:

brokerName
This parameter identifies the name of a locally defined broker. You cannot use this option if the broker is on a remote computer.

-n *brokerFileName*
This parameter identifies the name of a file that contains the connection details for a local or remote broker.

Use this option if multiple users want to connect to the same broker, or if you are using advanced connection parameters such as SSL.

To create this file, right-click the broker in the WebSphere Message Broker Explorer and select **Export *.broker file**. When prompted, navigate to the location in which you want to save the file and enter the file name; the extension `.broker` is appended automatically when you click **Save**. Include the location (path) and file name when you specify this parameter. You must ensure that the location is accessible when you run this command.

If you want to run a command that uses SSL to administer a remote broker over a secured channel, you must specify the keystore and truststore password for the connection using the IBM_JAVA_OPTIONS environment variable. See "Resolving problems when running commands" on page 3364 for further information.

-i *ipAddress*, **-p** *port*, **-q** *qMgr*
These parameters identify the connection details for the broker.

Use this option for connections to remote brokers that do not require advanced connection parameters.

If you choose this option, you must specify at least one of these three parameters; the order in which you specify them is insignificant. You cannot specify each parameter more than once.

Parameters that you omit assume default values:

- **-i** *ipAddress*: The host name or the IP address of the computer on which the broker is running. If you do not specify this parameter, a value that represents the local computer is used.
- **-p** *port*: The TCP port on which the broker's queue manager is listening. If you do not specify this parameter, the value 1414 is used.
- **-q** *qMgr*: The name of the broker's queue manager. If you do not specify this parameter, a value that represents the default queue manager on the local computer is used.

You cannot connect to a remote queue manager on z/OS; see "Usage note" on page 3978.

-e *executionGroupName*
(Optional) The name of the execution group for which to stop message flows. You must specify either **-e** or **-g**.

If you specify **-e** without **-m** or **-j**, the execution group is stopped. The state of every message flow (stopped or started) is retained, and started message flows are started when the execution group is started again.

If you specify **-e** with **-m** or **-j**, the execution group must be running. If the execution group is stopped, the request is rejected.

-g (Optional) The specified message flow or flows are stopped on all execution groups on the specified broker. You must specify either **-e** or **-g**.

If you specify **-g** without **-m** or **-j**, all execution groups are stopped. The state of every message flow (stopped or started) is retained for each execution group, and started message flows are started when the execution group is started again.

If you specify **-g** with **-m** or **-j**, the specified message flow or flows are stopped only if the execution group is running. The request checks all execution groups, and stops message flows on execution groups that are currently running. Error BIP2851 is reported for each execution group that is not running.

-j

(Optional) All message flows in the specified execution group are stopped; the execution group is not stopped.

If you do not specify **-j** or **-m**, the execution group identified by **-e** (if specified), or all execution groups on this broker, are stopped.

-m *flowName*
(Optional) The name of the message flow being stopped.

You can specify only one message flow in a single command. If you want to stop all message flows deployed to the broker, or to the execution group identified by **-e**, specify **-j**.

If you do not specify **-m** or **-j**, the execution group identified by **-e** (if specified), or all execution groups on this broker, are stopped.

-v *traceFileName*
(Optional) This parameter sends internal debug trace information to the specified file.

-w *timeoutSecs*
(Optional) This parameter specifies the time in seconds that the utility waits to ensure that the command completed; the default value is 60.

Usage note:
If you try to connect to a remote broker by specifying the **i**, **p**, and **q** parameters, or by using a connection parameter file (`.broker`), the command attempts to use WebSphere MQ Java client code. This option is not supported on z/OS, and returns the following error:

```
BIP1046E: Unable to connect with the broker (name)
```

The reported reason code is `MQRC_ENVIRONMENT_ERROR`. You must specify a local queue manager.

Authorization:
For information about platform-specific authorizations, see the following topics:

- "Security requirements for Linux and UNIX platforms" on page 3648
- "Security requirements for Windows systems" on page 3651
- "Security requirements for z/OS" on page 3655

If you have enabled broker administration security, you must also set up the authority detailed in "Tasks and authorizations for broker administration security" on page 3645.

Responses:
This command returns the following responses:

0 The command completed successfully.

2 (Failure) The broker received the deployment request but was unable to process it successfully. See the messages issued from the utility (or the Administration log) for more information.

9 (Failure) The request has been submitted to the broker, but no response was received before the timeout expired.

10 (Failure) Another user or application canceled the request operation before the broker was able to process it.

98 The broker is not running.

99 One or more of the parameters that you specified is invalid.

Examples:
Stop a named execution group on broker MB7BROKER:

```
mqsistopmsgflow MB7BROKER -e eg2
```

Stop all execution group processes on the local broker MB7BROKER:

```
mqsistopmsgflow MB7BROKER -g
```

:

Related tasks:

"Enabling broker administration security" on page 368
Enable broker administration security on a broker to control which users can complete specific tasks against that broker and its resources.

Chapter 11, "Packaging and deploying," on page 3209
Package resources that you create in the WebSphere Message Broker Toolkit, such as message flows, and deploy them to execution groups on brokers.

Related reference:

"**mqsideploy** command" on page 3872
Use the **mqsideploy** command to make a deployment request to the broker.

"**mqsistartmsgflow** command" on page 3969
Use the **mqsistartmsgflow** command to start execution groups and message flows.

"Syntax diagrams" on page 3677

z/OS configuration and administration specific information

The following links provide more information:
- "Administration in z/OS"
- "z/OS customization" on page 3983
- "z/OS JCL variables" on page 3994

Related concepts:

"z/OS customization overview" on page 592
After you have used SMP/E to install WebSphere Message Broker for z/OS, the installed executable code is located inside the file system. JCL samples are located in the PDS `<hlq>.SBIPSAMP`, the JCL procedures are located in the PDS `<hlq>.SBIPPROC`, and load module for synchronizing statistics with SMF are located in the PDS `<hlq>.SBIPAUTH`.

Related tasks:

Chapter 7, "Configuring brokers for test and production environments," on page 579
Create one or more brokers on one or more computers, and configure them on your test and production systems to process messages that contain your business data.

Chapter 8, "Administering brokers and broker resources," on page 899
Administering brokers and associated broker resources includes the tasks that you perform frequently to activate and manage those resources. Choose the method you prefer to administer your brokers and associated resources.

Administration in z/OS

In the z/OS environment, commands are issued through the console and others in batch jobs.

- *hlq*.`SBIPSAMP` has all the JCL samples to customize.
- *hlq*.`SBIPPROC` has all the JCL procedures to customize.

STEPNAME

The processes `BIPSERVICE` and `BIPBROKER` are in the same address space (control address space). After an Execution Group address space is started on z/OS, the STEPNAME of the address space has the following value:

- The last eight characters are taken from the Execution Group label.
- Any lowercase characters are folded to uppercase.
- Any non alphanumeric characters are changed to the character @.
- If the first character is not an alpha character, it is changed to `A`.

The STEPNAME is not guaranteed to be a unique value. However, you are strongly recommended to ensure that the last eight characters of any Execution Group labels that are deployed to a z/OS broker are unique, and contain only alphanumeric characters; note that an Execution Group label has a maximum size of 3000 bytes.

See the following topics for more information:

- "Issuing commands to the z/OS console"
- "Guidance for issuing console commands in z/OS" on page 3981
- "START and STOP commands on z/OS" on page 3981

Related tasks:

"Creating an execution group specific environment file" on page 627
You can create an execution group specific environment file, which is used instead of the default broker environment file when you restart the execution group.

Related reference:

"Issuing commands to the z/OS console"
You operate the broker using the z/OS START, STOP, and MODIFY commands.

"Guidance for issuing console commands in z/OS" on page 3981

"Commands" on page 3672
All WebSphere Message Broker Toolkit and runtime commands that are provided on distributed systems are listed, grouped by function, with references to command details.

Issuing commands to the z/OS console

You operate the broker using the z/OS START, STOP, and MODIFY commands.

You can issue commands, and get responses back from:

- The z/OS operator console
- The TSO CONSOLE facility
- The CONSOLE interface from REXX
- Products such as SDSF
- z/OS automation products, such as NetView

However, you are likely to need to issue commands with mixed case, because execution group names are often in mixed case. You can issue commands with mixed case on the z/OS console, using the REXX CONSOLE interface, with products like SDSF V2.10 and higher, and NetView. Releases of SDSF before V2.10, and the TSO CONSOLE facility, do not support passing mixed case data.

If you do not have support for mixed case, you can submit the commands through a batch job. For example:

```
//MI01CMD JOB MSGCLASS=H
//  COMMAND 'f MQP1BRK,ct t=yes,e='default',l=debug,f='lowmfl''
//STEP1 EXEC PGM=IEFBR14
```

If your product for issuing console commands supports mixed case input, you might need to take special actions to use mixed case. For example in SDSF on OS/390 V2.10 and above, you can enter mixed case keyword values by typing `/`, pressing `ENTER`, and entering the command in the popup window. If you enter `/` followed by the command, the command is translated to uppercase. In NetView, prefix the command with `NETVASIS` to get lowercase support.

Related reference:

"Administration in z/OS" on page 3979
In the z/OS environment, commands are issued through the console and others in batch jobs.

"Guidance for issuing console commands in z/OS" on page 3981

Guidance for issuing console commands in z/OS

These examples use a broker called MQP1BRK. Start and stop the broker using the MVS START (S) and STOP (P) commands. If you want to pass information to the broker while it is running, use the MVS MODIFY command (F) to issue commands. For example:

```
F MQP1BKR,rt
```

This list summarizes the rules you must follow when issuing console commands:

- Each command starts with a verb followed by zero or more keyword=value pairs.
- There must be one or more blanks between the verb and the first keyword.
- All characters are converted to lowercase, unless they are within quotation marks.
- Multiple keywords are separated by , with no blanks.
- The keyword is never case sensitive.
- The value is case sensitive, unless specified otherwise (for example Yes/No, trace-modes).
- Each keyword must be followed by the equals sign = and parameter value. Enter parameters in any order in the form:

  ```
  flag=value
  ```

  and separate them with commas. Repeated parameters are not allowed.
- Strings that contain blanks or special characters must be enclosed in single quotation marks ('). If the string itself contains a quotation mark, the quotation mark is represented by two single quotation marks.
- The verb and keywords are not case sensitive.
- On mixed case consoles the case of values in single quotation marks (') is not changed.
- For YES/NO flags, all case combinations are allowed.
- The maximum length of the MODIFY command is 126 characters, including

  ```
  F taskname,
  ```

An example console command:

```
F MQP1BRK,changetrace u=Yes,l=normal,e='myExecutionGroup'
```

Related reference:

"Administration in z/OS" on page 3979
In the z/OS environment, commands are issued through the console and others in batch jobs.

"Issuing commands to the z/OS console" on page 3980
You operate the broker using the z/OS START, STOP, and MODIFY commands.

START and STOP commands on z/OS

These examples use a broker called MQP1BRK. Start and stop the broker using the MVS START (S) and STOP (P) commands. If you want to pass information to the broker while it is running, use the MVS MODIFY command (F) to issue commands. For example:

```
F MQP1BRK,rt
```

The MVS command START (S) starts a broker on MVS. This initiates the control address space, plus other address spaces as needed, to run that component.

If you have problems starting the broker, use the command:

```
S <broker name>,STRTP=MAN
```

This command starts the administration task but not the execution groups.

The MVS command STOP (P) stops a server component completely, including its control address space.

For administration purposes you can bring the server component into another state, where the control address space is still running, but all other components are stopped.

For example, this is needed in order to change broker startup parameters, see "**mqsichangebroker** command" on page 3723. You can do this by using the MODIFY (/F) command on the started component, and using the startcomponent (SC) or stopcomponent (PC) options. See "**mqsistart** command" on page 3965 and "**mqsistop** command" on page 3972 for more information.

Related reference:

"Administration in z/OS" on page 3979
In the z/OS environment, commands are issued through the console and others in batch jobs.

"Issuing commands to the z/OS console" on page 3980
You operate the broker using the z/OS START, STOP, and MODIFY commands.

"**mqsistart** command" on page 3965
Use the **mqsistart** command to start the specified broker if all initial verification tests complete successfully.

"**mqsistop** command" on page 3972
Use the **mqsistop** command to stop the specified component.

Usage data on z/OS

This topic introduces SMF 89 subtype 1 records as a method of recording accumulated usage data on z/OS.

Whenever an execution group (DataFlowEngine) address space starts on z/OS, the system registers the address space for usage data collection; the system writes this usage data information to SMF 89 subtype 1 records.

When the execution group address space stops, the system deregisters the address space. The data that is collected in the SMF 89 subtype 1 records does not correspond to the data that is collected by the Accounting and Statistics SMF 117 records.

The system writes the accumulated usage data at a scheduled interval, the maximum period of which is one hour. The timing of the interval is set by the INTERVAL value that is specified for the SMF address space.

The system writes any usage data that is produced by other products to the same SMF 89 subtype 1 record.

On registering for usage data collection, the system writes the following message to the system log:

```
BIP9272I MQ05BRK default 0 THE DATAFLOWENGINE PROCESS HAS REGISTERED SMF 89
         SUBTYPE 1 RECORD COLLECTION. RETURN CODE '0', : ImbMain(316)
```

On deregistering from usage data collection, the system writes the following message to the system log:

```
BIP9273I MQ05BRK default 0 THE DATAFLOWENGINE PROCESS HAS DEREGISTERED SMF 89
         SUBTYPE 1 RECORD COLLECTION. RETURN CODE '0', : ImbMain(1079)
```

The values set in the SMF 89 subtype 1 records are defined in the following table..

Name	Description	WebSphere Message Broker value
SMF89UPO	Product owner	"IBM CORP"
SMF89UPN	Product name	"WMB" WebSphere Message Broker "WRF" WebSphere Message Broker Rules and Formatter
SMF89UPV	Product version	"NOTUSAGE"
SMF89UPQ	Product qualifier	""
SMF89UPI	Product ID	"5655-74" for WMB "5697-J09" for WRF

In all cases, the values are left-aligned and padded with blanks.

All execution groups that are started and stopped on a single system provide the same values when registering and deregistering. Therefore, the usage data that is written at the end of an interval in the SMF 89 subtype 1 record is an accumulation of all execution group address spaces.

For further information about SMF 89 subtype 1 records, see the *MVS System Management Facilities (SMF)* manual.

Related reference:

"Administration in z/OS" on page 3979
In the z/OS environment, commands are issued through the console and others in batch jobs.

Related information:

z/OS V1R7.0 LibraryCenter

z/OS customization

This is an introduction topic for a number of reference topics in the area of z/OS customization. See the links under *Related reference information*.

Related tasks:

"Customizing the z/OS environment" on page 591
If you are planning to use a z/OS environment, consider whether to create your brokers on z/OS. You must also complete a number of tasks to configure your environment.

"Creating a broker on z/OS" on page 620
Create the broker component and the other resources on which it depends.

Related reference:

"Naming conventions for WebSphere Message Broker for z/OS" on page 3984
Decide upon a naming convention for your WebSphere Message Broker for z/OS broker to make customizing, operating, and administering easier.

"Customization tasks and roles on z/OS" on page 3984
System programmers do most of the customization of WebSphere Message Broker.

"Customization planning checklist for z/OS" on page 3991
Use the information contained in the following tables to make a note of the values to use when you customize your system variables on z/OS.

"Disk space requirements on z/OS" on page 3586
The installation of WebSphere Message Broker for z/OS uses approximately 400 MB of disk space; plan on using 500 MB to allow for the component directories, and for new service fixes to be applied.

"Contents of the broker PDSE" on page 3991
After you have successfully customized the broker, the broker PDSE members have been set up.

Naming conventions for WebSphere Message Broker for z/OS

Decide upon a naming convention for your WebSphere Message Broker for z/OS broker to make customizing, operating, and administering easier.

Each broker requires its own queue manager. Base your broker name on the queue manager name. For example, append `BRK` to the queue manager name of `MQP1`, to give `MQP1BRK`. This naming convention has the following advantages:

- It is easy to associate the broker with the queue manager, because they both begin with the same characters.
- The started task name has the same name as the broker.

Using this convention, you might have the following broker name `MQP1BRK` for a queue manager `MQP1`:

- A started task name of `MQP1BRK`.
- A started task user ID of `MQP1BRK`.
- A PDSE containing definitions called `hlq.MQP1BRK.xxx`.
- A UNIX System Services directory structure like `/xxx/yyy/MQP1BRK`.

Plan for expansion by selecting names that allow growth.

If your broker name is in uppercase, ensure that the broker name is also in uppercase in the WebSphere Message Broker Toolkit and WebSphere Message Broker Explorer.

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related tasks:

"Creating a broker on z/OS" on page 620
Create the broker component and the other resources on which it depends.

Customization tasks and roles on z/OS

System programmers do most of the customization of WebSphere Message Broker.

Tasks that need to be completed by other people in your organization are identified in the following table.

Role	Task
z/OS system programmer	"Customizing the z/OS environment" on page 591
	"Setting up z/OS security" on page 556
	"Summary of required access (z/OS)" on page 3985

Role	Task
	"Customizing UNIX System Services on z/OS" on page 598
	"Creating the broker component" on page 629
	"Starting and stopping a broker on z/OS" on page 924
	"Checking APF attributes of `bipimain` on z/OS" on page 607
WebSphere MQ administrator	"Setting up WebSphere MQ" on page 558
	"Summary of required access (z/OS)"
	"WebSphere MQ planning for z/OS" on page 601
	"Creating the broker component" on page 629
	"Defining the started tasks to z/OS Workload Manager (WLM)" on page 602
WebSphere Message Broker administrator	"Setting up WebSphere Message Broker Toolkit access on z/OS" on page 559
	"Summary of required access (z/OS)"
	"Creating a broker on z/OS" on page 620
	"Creating the broker component" on page 629
	"Checking APF attributes of `bipimain` on z/OS" on page 607
Performance specialist	"Defining the started tasks to z/OS Workload Manager (WLM)" on page 602
Security administrator	"Setting up z/OS security" on page 556
	"Summary of required access (z/OS)"
Data administrator	"Setting up z/OS security" on page 556
	"Summary of required access (z/OS)"
	"Disk space requirements on z/OS" on page 3586
	"Starting and stopping a broker on z/OS" on page 924
	"Using the file system on z/OS" on page 596

Some tasks, for example defining queue security, overlap two different roles.

Related reference:

"Summary of required access (z/OS)"
The professionals in your organization require access to components and resources on z/OS.

Summary of required access (z/OS):

The professionals in your organization require access to components and resources on z/OS.

Authorizations required for the WebSphere Message Broker started-task user ID:
The following directory authorizations are required for all brokers:

- *READ* and *EXECUTE* access to `<INSTPATH>`, where `<INSTPATH>` is the directory where WebSphere Message Broker for z/OS is installed by SMP/E.

- *READ*, *WRITE*, and *EXECUTE* access to the component directory ++COMPONENTDIRECTORY++.
- *READ* and *WRITE* access to the home directory.
- *READ* and *WRITE* access to the directory identified by ++HOME++.
- In UNIX System Services, the started task user ID and the WebSphere Message Broker administrator user ID must both be members of the groups that have access to the installation and component directories, because they both need privileges over these resources. The owner of these directories must give the appropriate permissions to this group.

All brokers need the following RACF authorizations:

- *READ* and *WRITE* access to RACF class BPX.SMF, when you need to create SMF 117 records for accounting and statistics.
- *READ* access to the CSFRNG resource in the CSFSERV class.

READ access to the component PDSE is required.

WebSphere MQ authorizations

Enable WebSphere MQ security to protect your WebSphere MQ resources. If all WebSphere MQ security switches are enabled, define the following profiles, and give the started task user ID the listed access to each profile. For each profile access listed, <MQ_QMNAME> represents the WebSphere MQ queue manager that the WebSphere Message Broker component is connected to, and TASKID represents the started task user ID.

- Connection security: *READ* access to profile <MQ_QMNAME>.BATCH of class MQCONN. For example, for queue manager MQP1 and started task ID TASKID, use the RACF commands:

  ```
  RDEFINE MQCONN MQP1.BATCH UACC(NONE)
  PERMIT MQP1.BATCH CLASS(MQCONN) ID(TASKID) ACCESS(READ)
  ```

- Connection security when content-based filtering with publish/subscribe is used: *UPDATE* access to profile <MQ_QMNAME>.BATCH of class MQCONN. For example, for queue manager MQP1 and started task ID TASKID, use the RACF commands:

  ```
  RDEFINE MQCONN MQP1.BATCH UACC(NONE)
  PERMIT MQP1.BATCH CLASS(MQCONN) ID(TASKID) ACCESS(UPDATE)
  ```

- Queue security: *UPDATE* access to profile <MQ_QMNAME>.queue of class MQQUEUE for all queues. Consider creating profiles for the following queues:
  - All component queues, by using the generic profile SYSTEM.BROKER.**
  - All transmissions queues that you have defined between component queue managers.
  - All queues that you have specified in message flows.
  - Dead-letter queues.
  - Model queues.

  For example, for queue manager MQP1 and started task ID TASKID, use the following RACF commands to restrict access to the component queues:

  ```
  RDEFINE MQQUEUE MQP1.SYSTEM.BROKER.** UACC(NONE)
  PERMIT MQP1.SYSTEM.BROKER.** CLASS(MQQUEUE) ID(TASKID) ACCESS(UPDATE)
  ```

- Context security: *CONTROL* access to profile <MQ_QMNAME>.CONTEXT of class MQADMIN. For example, for queue manager MQP1 and started task ID TASKID, use the following RACF commands:

  ```
  RDEFINE MQADMIN MQP1.CONTEXT UACC(NONE)
  PERMIT MQP1.CONTEXT.** CLASS(MQADMIN) ID(TASKID) ACCESS(CONTROL)
  ```

- Alternate user security: Define the alternate user authority as: *UPDATE* access to profile <MQ_QMNAME>.ALTERNATE.USER.id of class MQADMIN, where id represents the start task ID of the broker component. For example, for queue manager MQP1, started task ID TASKID, and configuration service ID CFGID, use the following RACF commands:

  ```
  RDEFINE MQADMIN MQP1.ALTERNATE.USER.CFGID UACC(NONE)
  PERMIT MQP1.ALTERNATE.USER.CFGID  CLASS(MQADMIN) ID(TASKID) ACCESS(UPDATE)
  ```

  UPDATE access to profile <MQ_QMNAME>.ALTERNATE.USER.id of class MQADMIN, where id represents the user ID of, for example, a publish/subscribe request.
- Process and namelist security: If you have WebSphere MQ security switches enabled in your system for process and namelist security, you do not have to define access profiles in a WebSphere Message Broker default configuration.

For users connecting remotely from the WebSphere Message Broker Explorer, the WebSphere Message Broker Toolkit or from a CMP API application to the broker on z/OS, the following authorizations are required. CMP applications include the commands that use that interface; **mqsichangeresourcestats**, **mqsicreateexecutiongroup**, **mqsideleteexecutiongroup**, **mqsideploy**, **mqsilist**, **mqsimode**, **mqsireloadsecurity**, **mqsireportresourcestats**, **mqsistartmsgflow**, and **mqsistopmsgflow**.

- Connection security: *READ* access to profile <MQ_QMNAME>.CHIN of class MQCONN. For example, for queue manager MQP1 and started task ID TASKID, use the following RACF commands:

  ```
  RDEFINE MQCONN MQP1.CHIN UACC(NONE)
  PERMIT MQP1.CHIN CLASS(MQCONN) ID(TASKID) ACCESS(READ)
  ```
- Alternate user security: Define the alternate user authority as: *UPDATE* access to profile <MQ_QMNAME>.ALTERNATE.USER.id of class MQADMIN, where id represents the user ID of the WebSphere Message Broker Toolkit or CMP API application. For example, for queue manager MQP1, started task ID TASKID, and user ID USERID, use the following RACF commands:

  ```
  RDEFINE MQADMIN MQP1.ALTERNATE.USER.USERID UACC(NONE)
  PERMIT MQP1.ALTERNATE.USER.USERID  CLASS(MQADMIN) ID(TASKID) ACCESS(UPDATE)
  ```

Authorizations required for the WebSphere Message Broker administrator:
The broker administrator requires the following authorizations:

- *ALTER* access to the component PDSE.
- *READ*, *WRITE*, and *EXECUTE* access to the component directory ++COMPONENTDIRECTORY++.
- *READ* and *EXECUTE* access to <INSTPATH>, where <INSTPATH> is the directory where WebSphere Message Broker for z/OS is installed by SMP/E.
- *READ* and *WRITE* access to the directory identified by ++HOME++.
- In UNIX System Services, the started task user ID and the WebSphere Message Broker administrator user ID must both be members of the groups that have access to the installation and component directories, because they both need privileges over these resources. The owner of these directories needs to give the appropriate permissions to this group. In addition, the WebSphere Message Broker administrator must be a member of the group that is the primary group of the started task user ID.

Authorizations required for the WebSphere MQ administrator:
If the WebSphere MQ administrator runs the WebSphere MQ pass when creating a broker, the administrator user ID requires the following authorizations.

Alternatively, you can grant authorization to the WebSphere Message Broker administrator to run the WebSphere MQ pass.

- *ALTER* access to the component PDSE.
- Directory authorizations:
  - *READ* and *EXECUTE* access to <INSTPATH>, where <INSTPATH> is the directory where WebSphere Message Broker for z/OS is installed by SMP/E.
  - *READ*, *WRITE*, and *EXECUTE* access to the component directory ++COMPONENTDIRECTORY++.
  - *READ* and *WRITE* access to the directory identified by ++HOME++.

Enable WebSphere MQ security to protect your WebSphere MQ resources. If all WebSphere MQ security switches are enabled, define the following profiles and give the WebSphere MQ administrator the listed access to each profile in order to run the WebSphere MQ configurations jobs. For each profile access listed, MQ_QMNAME represents the WebSphere MQ queue manager that the WebSphere Message Broker component is connected to, and MQADMIN represents the WebSphere MQ administrator ID:

- Connection security: *READ* access to profile <MQ_QMNAME>.BATCH of class MQCONN. For example, for queue manager MQP1 and WebSphere MQ administrator ID MQADMIN, use the following RACF commands:

  ```
  RDEFINE MQCONN MQP1.BATCH UACC(NONE)
  PERMIT MQP1.BATCH CLASS(MQCONN) ID(MQADMIN) ACCESS(READ)
  ```

- Queue security: *UPDATE* access to profile <MQ_QMNAME>.queue of class MQQUEUE for component queues created or deleted. You can create a generic profile SYSTEM.BROKER.** For example, for queue manager MQP1 and WebSphere MQ administrator ID MQADMIN, use the following RACF commands to restrict access to the component queues:

  ```
  RDEFINE MQQUEUE MQP1.SYSTEM.BROKER.** UACC(NONE)
  PERMIT MQP1.SYSTEM.BROKER.** CLASS(MQQUEUE) ID(MQADMIN) ACCESS(UPDATE)
  ```

- System command server: *UPDATE* access to profile <MQ_QMNAME>.queue of class MQQUEUE for SYSTEM.COMMAND.**. For example, for queue manager MQP1 and WebSphere MQ administrator ID MQADMIN, use the following RACF commands to restrict access to the system command server:

  ```
  RDEFINE MQQUEUE MQP1.SYSTEM.COMMAND.** UACC(NONE)
  PERMIT MQP1.SYSTEM.COMMAND.** CLASS(MQQUEUE) ID(MQADMIN) ACCESS(UPDATE)
  ```

  UPDATE access to profile <MQ_QMNAME>.queue of class MQQUEUE for some system queues used during the create/delete job. You can create a generic profile <MQ_QMNAME>.**

- Command security:
  - To run the WebSphere MQ pass when creating a component you need:
    - *ALTER* access to <MQ_QMNAME>.DEFINE.QLOCAL of class MQCMDS.
    - *ALTER* access to <MQ_QMNAME>.DEFINE.QMODEL of class MQCMDS.
    - *ALTER* access to <MQ_QMNAME>.DEFINE.CHANNEL of class MQCMDS.
  - To run the WebSphere MQ pass when deleting a component you need:
    - *ALTER* access to <MQ_QMNAME>.DELETE.QLOCAL of class MQCMDS.
    - *ALTER* access to <MQ_QMNAME>.DELETE.QMODEL of class MQCMDS.
    - *ALTER* access to <MQ_QMNAME>.DELETE.CHANNEL of class MQCMDS.

  For queue manager MQP1 and WebSphere MQ administrator ID MQADMIN, use the following RACF commands:

```
RDEFINE MQCMDS MQP1.DELETE.QLOCAL UACC(NONE)
PERMIT MQP1.DELETE.QLOCAL CLASS(MQCMDS) ID(MQADMIN) ACCESS(ALTER)
```

- Resource command security: *ALTER* access to `MQP1.QUEUE.queue` of class `MQADMIN` for each queue created or deleted. You can create a generic profile SYSTEM.BROKER.**. For example, for queue manager `MQP1` and WebSphere MQ administrator ID `MQADMIN`, use the RACF commands:

  ```
  RDEFINE MQADMIN MQP1.QUEUE.SYSTEM.BROKER.** UACC(NONE)
  PERMIT MQP1.QUEUE.SYSTEM.BROKER.** CLASS(MQADMIN) ID(MQADMIN) ACCESS(ALTER)
  ```

- Process and namelist security: If you have WebSphere MQ security switches enabled in your system for process and namelist security, you do not need to define any access profiles in a WebSphere Message Broker default configuration.

For a description of how to implement WebSphere MQ security using RACF, see "Setting up WebSphere MQ" on page 558.

Disk space requirements on z/OS

The installation of WebSphere Message Broker for z/OS uses approximately 400 MB of disk space; plan on using 500 MB to allow for the component directories, and for new service fixes to be applied.

When you apply service, if you do not replace your existing installation (for example, you apply the new fix pack level alongside your existing installation), you must plan the same amount of disk space for the higher service level libraries.

If you are transferring the files by using *tar* to package them, you need approximately 200 MB of space for the `.tar` file.

You can check how much space is used and how much is free in a file system by using the OMVS command:

```
df -P /pathname
```

100 MB is 3 276 800 512 byte sectors.

The following table gives guidance on the space required for a minimum installation (base installation and verification test) of WebSphere Message Broker for each component implemented on z/OS.

Component	Space required	Purpose
Component directory	20 MB	Holds the runtime-deployed configuration information and output directories for the component. This information includes all deployment information, such as ESQL, JAR files, message sets, XSLT files, and so on. This information also includes WebSphere Message Broker trace files and other user problem determination data, which might become large. Consideration must be given to the potential size of deployments to the WebSphere Message Broker runtime environment and, therefore, the size of this directory (including sub directories).

Component	Space required	Purpose
Component PDSE	1 MB	Holds the customization and administration jobs, procedures, and data for the component. The data set must be allocated with a fixed record length of 80 (LRECL=80) and a format of FB 80. Reserve directory space for 50 members, or use a PDSE if possible.
Started task user ID home directory	8 GB	Collects diagnostic materials: for example, dumps. Dumps are usually more than 500 MB in size. 8 GB of space must be available in the file system, but many user IDs can have their home directory in the file system.

The Component directory and the Started task user ID home directory must be separate to ensure that, when dumps are taken in the Started task user ID home directory, they do not cause problems with the runtime broker that still has to write to the Component directory.

Related tasks:

"Customizing the z/OS environment" on page 591
If you are planning to use a z/OS environment, consider whether to create your brokers on z/OS. You must also complete a number of tasks to configure your environment.

"Using the file system on z/OS" on page 596
If you have more than one MVS image, consider how you will use the file system. You can share files in a file system with different members of a sysplex. The file system is mounted on one MVS image and requests to the file are routed to the owning system using XCF from systems which do not have it mounted.

Binding a DB2 plan to use data-sharing groups on z/OS

During customization, you can specify which plan name to use, or use the default DSNACLI. If you are using XPLINK, the default plan is called DSNACLX. If you want your broker to access DB2 data-sharing groups other than its own, the DSNACLI plan must be bound in a special way. If the broker uses one data sharing group, but might want to access tables on DSNONE and DSNTWO, which are in different data-sharing groups, amend the DB2 supplied job DSNTIJCL to do the following:

```
BIND PACKAGE (DSNAOCLI)MEMBER(DSNCLICS)ISOLATION(CS)
BIND PACKAGE (DSNAOCLI)MEMBER(DSNCLINC)ISOLATION(NC)
BIND PACKAGE (DSNAOCLI)MEMBER(DSNCLIRR)ISOLATION(RR)
BIND PACKAGE (DSNAOCLI)MEMBER(DSNCLIRS)ISOLATION(RS)
BIND PACKAGE (DSNAOCLI)MEMBER(DSNCLIUR)ISOLATION(UR)
BIND PACKAGE (DSNAOCLI)MEMBER(DSNCLIC1)
BIND PACKAGE (DSNAOCLI)MEMBER(DSNCLIC2)
BIND PACKAGE (DSNAOCLI)MEMBER(DSNCLIF4)
BIND PACKAGE (DSNAOCLI)MEMBER(DSNCLIMS)
BIND PACKAGE (DSNAOCLI)MEMBER(DSNCLIQR)
BIND PACKAGE (DSNONE.DSNAOCLI)MEMBER(DSNCLICS)ISOLATION(CS)
BIND PACKAGE (DSNONE.DSNAOCLI)MEMBER(DSNCLINC)ISOLATION(NC)
BIND PACKAGE (DSNONE.DSNAOCLI)MEMBER(DSNCLIRR)ISOLATION(RR)
BIND PACKAGE (DSNONE.DSNAOCLI)MEMBER(DSNCLIRS)ISOLATION(RS)
BIND PACKAGE (DSNONE.DSNAOCLI)MEMBER(DSNCLIUR)ISOLATION(UR)
BIND PACKAGE (DSNONE.DSNAOCLI)MEMBER(DSNCLIC1)
BIND PACKAGE (DSNONE.DSNAOCLI)MEMBER(DSNCLIC2)
BIND PACKAGE (DSNONE.DSNAOCLI)MEMBER(DSNCLIF4)
BIND PACKAGE (DSNONE.DSNAOCLI)MEMBER(DSNCLIMS)
```

```
BIND PACKAGE (DSNONE.DSNAOCLI)MEMBER(DSNCLIQR)
BIND PACKAGE (DSNTWO.DSNAOCLI)MEMBER(DSNCLICS)ISOLATION(CS)
BIND PACKAGE (DSNTWO.DSNAOCLI)MEMBER(DSNCLINC)ISOLATION(NC)
BIND PACKAGE (DSNTWO.DSNAOCLI)MEMBER(DSNCLIRR)ISOLATION(RR)
BIND PACKAGE (DSNTWO.DSNAOCLI)MEMBER(DSNCLIRS)ISOLATION(RS)
BIND PACKAGE (DSNTWO.DSNAOCLI)MEMBER(DSNCLIUR)ISOLATION(UR)
BIND PACKAGE (DSNTWO.DSNAOCLI)MEMBER(DSNCLIC1)
BIND PACKAGE (DSNTWO.DSNAOCLI)MEMBER(DSNCLIC2)
BIND PACKAGE (DSNTWO.DSNAOCLI)MEMBER(DSNCLIF4)
BIND PACKAGE (DSNTWO.DSNAOCLI)MEMBER(DSNCLIMS)
BIND PACKAGE (DSNTWO.DSNAOCLI)MEMBER(DSNCLIQR)
BIND PLAN(DSNACLI)-
PKLIST(*.DSNAOCLI.DSNCLICS -
*.DSNAOCLI.DSNCLINC -
*.DSNAOCLI.DSNCLIRR -
*.DSNAOCLI.DSNCLIRS -
*.DSNAOCLI.DSNCLIUR -
*.DSNAOCLI.DSNCLIC1 -
*.DSNAOCLI.DSNCLIC2 -
*.DSNAOCLI.DSNCLIF4 -
*.DSNAOCLI.DSNCLIMS -
*.DSNAOCLI.DSNCLIQR )
```

Related concepts:

"XPLink on z/OS" on page 595

Related tasks:

"Using the file system on z/OS" on page 596
If you have more than one MVS image, consider how you will use the file system. You can share files in a file system with different members of a sysplex. The file system is mounted on one MVS image and requests to the file are routed to the owning system using XCF from systems which do not have it mounted.

Customization planning checklist for z/OS

Use the information contained in the following tables to make a note of the values to use when you customize your system variables on z/OS.

For further information, see the following topics:

- "Installation information - broker" on page 621
- "Component information - broker" on page 622

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related tasks:

"Collecting the information required to create a broker" on page 620
This is part of the larger task of creating a broker on z/OS.

Contents of the broker PDSE

After you have successfully customized the broker, the broker PDSE members have been set up.

Broker PDSE members originating in <hlq>.SBIPSAMP

The PDSE members are described in the following table.

Description	Name
Broker profile	BIPBPROF

Description	Name
Broker DB2 dsnaoini file (for user databases) "Sample BIPDSNAO file" on page 4008 lists the shipped BIPDSNAO file.	BIPDSNAO
Execution group specific profile	BIPEPROF

Broker PDSE members originating in <hlq>.SBIPPROC

The PDSE members are described in the following table.

Description	Name
Commented sample job to identify WebSphere MQ resources, including queues for performance tuning purposes	BIPALMQ
Job to run the "**mqsibackupbroker** command" on page 3720. "Sample BIPBUBK file" on page 4004 lists the shipped BIPBUBK file.	BIPBUBK
Job to run the **mqsibrowse** command. Use this command only at the request of IBM service.	BIPBRWS
Job to run the "**mqsichangebroker** command" on page 3723	BIPCHBK
Job to run the "**mqsichangeflowmonitoring** command" on page 3738	BIPCHME
Job to run the "**mqsichangeflowstats** command" on page 3744	BIPCHMS
Job to run the "**mqsichangeflowuserexits** command" on page 3751	BIPCHUE
Job to run the "**mqsichangeproperties** command" on page 3756	BIPCHPR
Job to run the "**mqsichangeresourcestats** command" on page 3819	BIPCHRS
Job to run the "**mqsicreatebroker** command" on page 3831 to: • Create the WebSphere MQ resources • Create the broker registry "Sample BIPCRBK file" on page 4006 lists the shipped BIPCRBK file.	BIPCRBK
Job to run the "**mqsicreateconfigurableservice** command" on page 3849	BIPCRCS
Job to run the "**mqsicreateexecutiongroup** command" on page 3854	BIPCREG
Job to run the "**mqsideletebroker** command" on page 3863	BIPDLBK
Job to run the "**mqsideleteconfigurableservice** command" on page 3866	BIPDLCS
Job to run the "**mqsideleteexecutiongroup** command" on page 3869	BIPDLEG
Job to run the "**mqsideploy** command" on page 3872	BIPDPLY
Edit macro for customization. Rename BIPEDIT to a unique name that identifies it to the current component; for example, MQ01EDBK "Sample BIPEDIT file" on page 4009 lists the shipped BIPEDIT file.	BIPEDIT (MQP1BRK)
PROC that is called by BIPGEN to convert BIPBPROF, and an execution group specific profile (renamed BIPEPROF), to a valid execution group specific ENVFILE.	BIPEGEN
Job to run the "**mqsiexplain** command" on page 3879	BIPEXPL

Description	Name
Job to run the "**mqsiformatlog** command" on page 3880	BIPFMLG
Job to convert BIPBPROF profile to a valid ENVFILE. "Sample BIPGEN file" on page 4010 lists the shipped BIPGEN file.	BIPGEN
Job to run the "**mqsilist** command" on page 3882	BIPLIST
Job to run the "**mqsimigratecomponents** command" on page 3894	BIPMGCMP
Job to run the "**mqsimode** command" on page 3899	BIPMODE
Job to run the "**mqsiapplybaroverride** command" on page 3684	BIPOBAR
Job to run the "**mqsireadbar** command" on page 3697	BIPRBAR
Job to run the "**mqsireadlog** command" on page 3905	BIPRELG
Job to run the "**mqsireloadsecurity** command" on page 3911	BIPRLSEC
Job to run the "**mqsireportbroker** command" on page 3919	BIPRPBK
Job to run the "**mqsireportflowmonitoring** command" on page 3924	BIPRPME
Job to run the "**mqsireportflowstats** command" on page 3929	BIPRPMS
Job to run the "**mqsireportflowuserexits** command" on page 3933	BIPRPUE
Job to run the "**mqsireportproperties** command" on page 3937	BIPRPPR
Job to run the "**mqsireportresourcestats** command" on page 3944	BIPRPRS
Job to run the "**mqsirestorebroker** command" on page 3952. "Sample BIPRSBK file" on page 4012 lists the shipped BIPRSBK file.	BIPRSBK
Job to run the "**mqsisetdbparms** command" on page 3954 to define a data source, user ID, and password for user data sources.	BIPSDBP
(started task). Rename BIPBRKP to **the same name as the ++STARTEDTASKNAME++**. "Sample BIPBRKP file" on page 4000 lists the shipped BIPBRKP file.	BIPBRKP
Job to run the "**mqsistartmsgflow** command" on page 3969	BIPSTMF
Job to run the "**mqsistopmsgflow** command" on page 3975	BIPSPMF

Use the standard z/OS IPCS facilities to take a dump. You require access to the following resources:

- COMPONENT_PDSE
- ComponentDirectory
- SYS1.MIGLIB
- SYS1.SBLSCLI0
- SYS1.PARMLIB
- IPCS

Related concepts:

"Component PDSE on z/OS" on page 595
On z/OS, the *component PDSE* contains jobs customized for a single component. These jobs are used to create and administer the component.

Related tasks:

"Customizing the broker component data set" on page 624
This is part of the larger task of creating a broker on z/OS.

z/OS JCL variables

Customize JCL variables in BIPEDIT to modify the configuration of your brokers.

The following table lists the JCL variables in alphabetic order, together with a description and an example value. For JCL variables that apply to specific commands, see the information that relates to that command.

JCL variable	Description	Example value	Corresponding value in BIPEDIT
++COMPONENTDATASET++	The data set where all JCL relevant to a particular component is saved.	TESTDEV.MQP1BRK.BROKER	componentdataset_value
++COMPONENTDIRECTORY++	The file system directory where the component exists. This directory includes subdirectories, for example, `/log` and `/registry`	`mqsi/brokers/MQP1BRK`	compdir_value
++COMPONENTNAME++	The name you give the component when you create it.	MQP1BRK	MQ01BRK
++HOME++	The file system home directory for the user ID of this component. This value is required to dynamically generate the `ENVFILE`. You must have the appropriate external security manager authorities to write to this file system directory when submitting JCL to run a command. For example, use RACF to define authorities.	`/u/mqp1usr/mqp1brk`	`/u/mq01brk`
++INSTALL++	The directory where you install the product.	`/usr/lpp/mqsi`	install_value
++JAVA++	Location of your Java installation.	`/usr/lpp/java/IBM/J1.5`	`/usr/lpp/java/IBM/J1.5`
++LANGLETTER++	The letter for the language in which you want messages shown.	E (English)	E
++LOCALE++	Locale of environment where commands are run by submitting JCL.	C	C
++MQPATH++	WebSphere MQ location.	`/usr/lpp/mqm`	`/usr/lpp/mqm`
++OPTIONS++	Many commands submitted by JCL require additional options. See the reference material for additional information about options specific to each command.	N/A	options_value
++QUEUEMANAGER++	The name of the queue manager associated with the component for which you submit the JCL.	MQP1	MQ01
++STARTEDTASKNAME++	Name of the Started Task JCL. This name can be a maximum of eight characters.	MQP1BRK	MQ01BRK
++TIMEZONE++	Time zone of environment where commands are run by submitting JCL.	GMT0BST	GMT0BST

JCL variable	Description	Example value	Corresponding value in BIPEDIT
++WMQHLQ++	WebSphere MQ high-level-qualifier	MQM.V700	MQM.V700

Related tasks:

"Collecting the information required to create a broker" on page 620
This is part of the larger task of creating a broker on z/OS.

Related reference:

"Sample BIPEDIT file" on page 4009
The sample BIPEDIT file that is shipped with WebSphere Message Broker is included here for your reference.

z/OS sample files supplied

Several samples files are provided for you to customize on z/OS.

About this task

- "Sample BIPBPROF file"
- "Sample BIPBRKP file" on page 4000
- "Sample BIPBUBK file" on page 4004
- "Sample BIPCRBK file" on page 4006
- "Sample BIPDSNAO file" on page 4008
- "Sample BIPEDIT file" on page 4009
- "Sample BIPGEN file" on page 4010
- "Sample BIPRSBK file" on page 4012

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related reference:

"z/OS configuration and administration specific information" on page 3979

Sample BIPBPROF file

The sample BIPBPROF file that is shipped with WebSphere Message Broker is included here for your reference.

```
#*****************************************************************
#*                                                               *
#* @START_COPYRIGHT@                                            *
#*                                                               *
#* Licensed Materials - Property of IBM;                         *
#* 5655-G97 (c) Copyright IBM Corp. 2004;                        *
#* All Rights Reserved;                                          *
#* US Government Users Restricted Rights - use,                  *
#* duplication or disclosure restricted by GSA                   *
#* ADP Schedule Contract with IBM Corp.;                         *
#* See Copyright Instructions                                    *
#*                                                               *
#* @END_COPYRIGHT@                                              *
#*                                                               *
#*****************************************************************
#*              IBM WebSphere Message Broker                     *
#*                                                               *
#* Sample profile for a broker.                                  *
#*                                                               *
#*****************************************************************
```

```
#* MORE INFORMATION - See:                                              *
#*                                                                      *
#*    WebSphere Message Brokers Information Centre.                     *
#*                                                                      *
#* IMPORTANT:                                                           *
#*                                                                      *
#*     You must submit BIPGEN each time you update this profile!        *
#*                                                                      *
#**********************************************************************
#* CUSTOMIZE HERE FOR YOUR INSTALLATION
#* YOU MUST DO GLOBAL CHANGES ON THESE PARAMETERS USING YOUR EDITOR
#*                                                                      *
#*    Replace   ++INSTALL++
#*                      WMB installation directory.
#*                      e.g. '/usr/lpp/mqsi'
#*
#*    Replace   ++COMPONENTDIRECTORY++
#*                      Broker directory.
#*                      e.g. '/mqsi/brokers/MQ01BRK'
#*
#*    Replace   ++COMPONENTDATASET++
#*                      Component dataset.
#*                      e.g. 'BIP.BROKER.MQ01BRK'
#*
#*    Replace   ++COMPONENTNAME++
#*                      Broker name.
#*                      e.g. 'MQ01BRK'
#*
#*    Replace   ++LOCALE++
#*                      Locale.
#*                      e.g. 'C'
#*
#*    Replace   ++TIMEZONE++
#*                      Time zone.
#*                      e.g. 'GMT0BST'
#*
#*    Replace   ++JAVA++
#*                      Java location.
#*                      e.g. '/usr/lpp/java/IBM/J1.6'
#*
#*    Replace   ++DB2CONVERSION++
#*                      Specifies the DB2 Converter.
#*                      e.g. 'SINGLE'
#*                      (ONLY REQUIRED FOR USER DATABASES)
#*
#*    Replace   ++MQPATH++
#*                      MQ location.
#*                      e.g. '/usr/lpp/mqm'
#*
#**********************************************************************
# 1. Component Settings
#**********************************************************************
#
# 1.1 MQSI_REGISTRY references the component path. Also needed by
#     commands.
#     e.g. MQSI_REGISTRY=/mqsi/brokers/MQ01BRK
#
# 1.2 MQSI_COMPONENT_NAME is set to the broker name.
#     e.g. MQSI_COMPONENT_NAME=MQ01BRK
#
# 1.3 MQSI_FILEPATH needed by commands to reference the component
#     version install path.
#     e.g. MQSI_FILEPATH=usr/lpp/mqsi/V7R0M0
#
# 1.4 MQSI_LILPATH needed by components to find LILs
#     e.g. MQSI_LILPATH=usr/lpp/mqsi/V7R0M0/lil
#
```

```
# 1.5 MQSI_SECURITY_PROVIDER_PATH needed by components to find LSLs
#     e.g. MQSI_SECURITY_PROVIDER_PATH=usr/lpp/mqsi/V7R0M0/SecurityProviders
#
export MQSI_REGISTRY=++COMPONENTDIRECTORY++
export MQSI_COMPONENT_NAME=++COMPONENTNAME++
export MQSI_FILEPATH=++INSTALL++
export MQSI_LILPATH=$MQSI_FILEPATH/lil
export MQSI_SECURITY_PROVIDER_PATH=$MQSI_FILEPATH/SecurityProviders


#*****************************************************************
# 2. NLS Settings
#*****************************************************************
#
# 2.1 LANG and LC_ALL determine in which locale the component
#     will run.
#     e.g. LANG=Ja_JP.IBM-939 and LC_ALL=Ja_JP.IBM-939 for
#          japanese locale.
#          LANG=C, LC_ALL=C for US English locale.
#
# 2.2 TZ has the timezone setting in which you are located.
#     e.g. TZ=EST5                    for USA Eastern Standard Time
#          TZ=MEZ-1MES,M3.5.0,M10.5.0 for Central Europe
#          TZ=GMT0BST                 for the UK
#          Please refer to the IBM Manual
#          "Unix System Services Command Reference SC28-1892.
#
# 2.3 NLSPATH contains the location of the message catalog(s).
#     (NO NEED TO CHANGE FROM DEFAULT!)
#
# 2.4 MQSI_CONSOLE_NLSPATH is used to locate the messages for
#     the console.
#     For Japanese or S-Chinese messages, change En_US to
#     Ja_JP or Zh_CN below. For English messages these can be
#     displayed in mixed or upper case only. (see MC_MESSAGES)
#     Note that MQSI_CONSOLE_NLSPATH does not use %L or %N
#
export LANG=++LOCALE++
export LC_ALL=++LOCALE++
export TZ=++TIMEZONE++
export NLSPATH=$MQSI_FILEPATH/messages/%L/%N
export NLSPATH=$NLSPATH:$MQSI_FILEPATH/nnsy/MIF/messages/%N
export MQSI_CONSOLE_NLSPATH=$MQSI_FILEPATH/messages/En_US


#*****************************************************************
# 3. Automatic Restart Management (ARM) Settings
#*****************************************************************
#
# 3.1 MQSI_USE_ARM specifies whether to use ARM.
#     e.g. MQSI_USE_ARM=YES for ARM enabled.
#          MQSI_USE_ARM=NO  for ARM not enabled.
#
# 3.2 MQSI_ARM_ELEMENTNAME required if ARM enabled.
#
# 3.3 MQSI_ARM_ELEMENTTYPE required if ARM enabled.
#
export MQSI_USE_ARM=NO
export MQSI_ARM_ELEMENTNAME=++COMPONENTNAME++
export MQSI_ARM_ELEMENTTYPE=


#*****************************************************************
# 4. DB2 Settings
#*****************************************************************
#
# 4.1 MQSI_DB2_ALWAYS_PREPARE
```

```
#     (NO NEED TO CHANGE FROM DEFAULT!)
#
# 4.2 MQSI_DB2_CONVERSION specifies the DB2 Converter.
#     (ONLY REQUIRED FOR USER DATABASES)
#     e.g. MQSI_DB2_CONVERSION=SINGLE
#                     WBIMB uses a SQL_EBCDIC_SCCSID to determine
#                     the DB2 converter.
#                     Note that this setting reflects the current
#                     WBIMB behavior.
#          MQSI_DB2_CONVERSION=MIXED
#                     WBIMB uses SQL_EBCDIC_MCCSID to determine
#                     the DB2 converter.
#                     Note that this setting requires that DB2 is
#                     configured to accept mixed byte data. This
#                     setting should be used when the customer
#                     wants data to be stored in the configured
#                     DB2 EBCDIC mixed byte code page.
#          MQSI_DB2_CONVERSION=LOCAL
#                     WBIMB uses localConverter identified by
#                     LC_ALL/LANG settings. This complies to what
#                     is done on distributed. This setting
#                     requires that WBIMB and DB2 are using the same
#                     codepage, otherwise only WBIMB can read DB2
#                     data correctly, it gets unreadable for other
#                     ( non-WBIMB ) applications that want to read
#                     the data.
#
# 4.3 DSNAOINI references the component dsnaoini file.
#     (NO NEED TO CHANGE FROM DEFAULT!)
#
export MQSI_DB2_ALWAYS_PREPARE=NO
#export MQSI_DB2_CONVERSION=++DB2CONVERSION++
export DSNAOINI=//\'++COMPONENTDATASET++\(BIPDSNAO\)\'

#*********************************************************************
# 5. Java Settings
#*********************************************************************
#
# 5.1 JAVAHOME contains the root directory of the JAVA install.
#     e.g. JAVAHOME=/usr/lpp/java/IBM/J1.6
#     Note that the Java version must be at least 1.6.0
#
export JAVAHOME=++JAVA++


#*********************************************************************
# 6. WebSphere Message Broker Settings
#*********************************************************************
#
# 6.1 _BPX_BATCH_SPAWN
#     (MUST NOT CHANGE FROM DEFAULT!)
#
# 6.2 MQSI_MC_MESSAGES determines if messages should appear in
#     mixed case or upper case.
#     e.g. MQSI_MC_MESSAGES=YES for mixed case
#          MQSI_MC_MESSAGES=NO for upper case
#
# 6.3 MQSI_COMMAND_DATABASE_ECHO if defined, mqsi commands
#     display information when creating DB2 tables/indexes.
#
# 6.4 MQSI_COMMAND_ZOS_MQ_ECHO if defined, mqsi commands display
#     information returned from the MQ command server when
#     creating/deleting queues.
#
# 6.5 MQSI_COMMAND_ZOS_MQ_ECHO_RC if defined, mqsi commands display
#     reason and return codes from the MQ command server when
#     creating/deleting queues.
```

```
#
# 6.6 MQSI_DEPLOY_PROGRESS if defined, shows deployment progress
#     by the execution group
#
# 6.7 STEPLIB
#     (MUST NOT CHANGE FROM DEFAULT!)
#
# 6.8 MQSI_COMMAND_ZOS_ERROR_RC if defined, mqsi commands return
#     this value as the return code if an error occurs.
#
# 6.9 MQSI_FAD current FAD level.
#
export _BPX_BATCH_SPAWN=NO
export MQSI_MC_MESSAGES=NO
export MQSI_COMMAND_DATABASE_ECHO=1
export MQSI_COMMAND_ZOS_MQ_ECHO=1
export MQSI_COMMAND_ZOS_MQ_ECHO_RC=1
export MQSI_DEPLOY_PROGRESS=1
export STEPLIB=CURRENT
export MQSI_COMMAND_ZOS_ERROR_RC=16
export MQSI_FAD=5


#*****************************************************************
# 7. Other Settings
#*****************************************************************
#
# NO NEED TO CHANGE FROM DEFAULT!
#

CP=++MQPATH++/java/lib/com.ibm.mq.jar
CP=$CP:++MQPATH++/java/lib/connector.jar
CP=$CP:++MQPATH++/java/lib/com.ibm.mq.commonservices.jar
CP=$CP:++MQPATH++/java/lib/com.ibm.mq.headers.jar
CP=$CP:++MQPATH++/java/lib/com.ibm.mq.jmqi.jar
CP=$CP:++MQPATH++/java/lib/com.ibm.mq.pcf.jar
CP=$CP:++MQPATH++/java/lib/com.ibm.mqjms.jar
CP=$CP:$MQSI_FILEPATH/classes
CP=$CP:$MQSI_FILEPATH/classes/deploymgr.jar
CP=$CP:$MQSI_FILEPATH/classes/ConfigManagerProxy.jar
CP=$CP:$MQSI_FILEPATH/sample/ConfigManagerProxy/ConfigManagerProxySamples.jar
CP=$CP:$MQSI_FILEPATH/classes/brokerutil.jar
CP=$CP:$JAVAHOME/lib
CP=$CP:$MQSI_FILEPATH/messages
export CLASSPATH=$CP
export ICU_DATA=$MQSI_FILEPATH/nnsy/lib:$MQSI_FILEPATH/icudata
LP=++MQPATH++/java/lib
LP=$LP:$MQSI_FILEPATH/lib/wbirf
LP=$LP:$MQSI_FILEPATH/lib/wbimb
LP=$LP:$MQSI_FILEPATH/lib
LP=$LP:$MQSI_FILEPATH/nnsy/lib
LP=$LP:$MQSI_FILEPATH/nnsy/MIF/lib
LP=$LP:$JAVAHOME/lib/s390x
LP=$LP:$JAVAHOME/lib/s390x/classic
export LIBPATH=$LP
MIBDIRS=$MQSI_FILEPATH/snmp-mib
export MIBDIRS
export PATH=$MQSI_FILEPATH/bin
export PATH=$PATH:$MQSI_FILEPATH/nnsy/bin
export PATH=$PATH:/bin
export PATH=$PATH:$JAVAHOME/bin
```

Related reference:

"Contents of the broker PDSE" on page 3991
After you have successfully customized the broker, the broker PDSE members have been set up.

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

"**mqsideletebroker** command" on page 3863
Use the **mqsideletebroker** command to delete a named broker. The command also deletes the queues on the associated queue manager (created when the broker was created). You can also specify that the queue manager is to be deleted.

Sample BIPBRKP file

The sample BIPBRKP file that is shipped with WebSphere Message Broker is included here for your reference.

```
//*************************************************************
//*                                                           *
//* @START_COPYRIGHT@                                         *
//*                                                           *
//* Licensed Materials - Property of IBM;                     *
//* ProgIds: 5724-J06 5724-J05 5724-J04 5697-J09 5655-M74 5655-M75 5655-G97
//* (C) Copyright IBM Corporation 2004,2010
//* All Rights Reserved;                                      *
//* US Government Users Restricted Rights - use,              *
//* duplication or disclosure restricted by GSA               *
//* ADP Schedule Contract with IBM Corp.;                     *
//* See Copyright Instructions                                *
//*                                                           *
//* @END_COPYRIGHT@                                           *
//*                                                           *
//*************************************************************
//*              IBM WebSphere Message Broker                 *
//*                                                           *
//* Sample job to start a broker.                             *
//*                                                           *
//*************************************************************
//* MORE INFORMATION - See:                                   *
//*                                                           *
//*    WebSphere Message Broker Information Centre.           *
//*                                                           *
//*************************************************************
//* CUSTOMIZE THIS JCL HERE FOR YOUR INSTALLATION
//* YOU MUST DO GLOBAL CHANGES ON THESE PARAMETERS USING YOUR EDITOR
//*                                                           *
//*    Replace   ++HOME++
//*                      Unique home directory where ENVFILE
//*                      will be created.
//*                      e.g. '/u/mq01usr/mq01brk'
//*
//*    Replace   ++INSTALL++
//*                      WMB installation directory.
//*                      e.g. '/usr/lpp/mqsi'
//*
//*    Replace   ++QUEUEMANAGER++
//*                      Queue manager name.
//*                      e.g. 'MQ01'
//*
//*    Replace   ++COMPONENTDIRECTORY++
//*                      Broker directory.
//*                      e.g. '/mqsi/brokers/MQ01BRK'
//*
//*    Replace   ++STARTEDTASKNAME++
//*                      Started Task Name (max 8 chars uppercase).
//*                      e.g. 'MQ01BRK'
//*
//*    Replace   ++COMPONENTNAME++
//*                      Component name.
//*                      e.g. 'MQ01BRK'
```

```
//*
//*    Replace   ++COMPONENTDATASET++
//*                      Broker dataset.
//*                      e.g. 'BIP.BROKER.MQ01BRK'
//*
//*    Replace   ++DB2HLQ++
//*                      DB2 high-level-qualifier.
//*                      e.g. 'SYS2.DB2.V910'
//*                      (Database nodes require DB2 in order
//*                       to connect to a datasource.)
//*
//*    Replace   ++WMQHLQ++
//*                      WebSphere MQ high-level-qualifier.
//*                      e.g. 'MQM.V701'
//*
//*    Replace   ++WMBHLQ++
//*                      WebSphere Message Broker
//*                      high-level-qualifier.
//*
//*                      ONLY NEEDED IF USING EVENT NOTIFICATION
//*
//*                      e.g. 'MB.V7R0M0'
//*
//*    Replace   ++LANGLETTER++
//*                      The letter for the language that
//*                      you want messages shown in.
//*                      e.g. 'E' for English
//*
//*****************************************************************
//*
//* Following variables are changed when starting a DataFlowEngine:
//*     Semaphore ID
//*       SE=<Semaphore ID>(default is '')
//*     Shared Memory Segment ID
//*       SH=<Shared Memory Segment ID>(default is '')
//*     Component Unique Name
//*       COMPK='' (default is ++STARTEDTASKNAME++)
//*     Start Parameter
//*       STRTP='' (default is 'AUTO')
//*     Execution Group name
//*       E='' (default is '')
//*
//*****************************************************************
//*
//++STARTEDTASKNAME++ PROC COMPK='++STARTEDTASKNAME++',
//         INSTP='++INSTALL++',
//         MAINP='bipimain',
//         SRVMP='bipservice',
//         COMDS='++COMPONENTDATASET++',
//         STRTP='AUTO',
//*        COMPDIR='++COMPONENTDIRECTORY++',
//         SE='',
//         SH='',
//         HOME='++HOME++',
//         E='',
//*        DB2HLQ='++DB2HLQ++',
//         WMQHLQ='++WMQHLQ++'
//*
//*
//*****************************************************************
//* Test to see if the ENVFILE exists.
//* The base ENVFILE called ENVFILE should exist.
//* Execution Group specifc ENVFILEs (ENVFILE.<EGLabel>) may
//* exist if specified. If not then use the base ENVFILE.
//*****************************************************************
//*
//CHECKENV EXEC PGM=BPXBATSL,REGION=0M,TIME=NOLIMIT,
```

```
//         PARM='PGM /bin/test -e &HOME./ENVFILE.&E.'
//STDOUT   DD SYSOUT=* //STDERR   DD SYSOUT=*
//*
//*****************************************************************
//* Copy ENVFILE to SYSOUT
//*****************************************************************
//*
//         IF (CHECKENV.RC EQ 0) THEN
//COPYENV1 EXEC PGM=IKJEFT01,
//         PARM='OCOPY INDD(BIPFROM) OUTDD(ENVFILE)'
//SYSTSPRT DD DUMMY
//BIPFROM  DD PATHOPTS=(ORDONLY),
//            PATH='&HOME./ENVFILE.&E.'
//ENVFILE  DD SYSOUT=*,DCB=(RECFM=V,LRECL=8192)
//SYSTSIN  DD DUMMY
//         ELSE
//COPYENV2 EXEC PGM=IKJEFT01,
//         PARM='OCOPY INDD(BIPFROM) OUTDD(ENVFILE)'
//SYSTSPRT DD DUMMY
//BIPFROM  DD PATHOPTS=(ORDONLY),
//            PATH='&HOME./ENVFILE'
//ENVFILE  DD SYSOUT=*,DCB=(RECFM=V,LRECL=8192)
//SYSTSIN  DD DUMMY
//         ENDIF
//*
//*****************************************************************
//* Copy DSNAOINI to SYSOUT
//*****************************************************************
//*
//COPYDSN  EXEC PGM=IKJEFT01,
//         PARM='OCOPY INDD(BIPFROM) OUTDD(DSNAOINI)'
//SYSTSPRT DD DUMMY
//BIPFROM  DD DISP=SHR,DSN=&COMDS.(BIPDSNAO)
//DSNAOINI DD SYSOUT=*,DCB=(RECFM=V,LRECL=256)
//SYSTSIN  DD DUMMY
//*
//*****************************************************************
//* Test to see if starting a DataFlowEngine address space.
//* Should return RC=0 if starting a Control address space or
//* RC=12 if starting a DataFlowEngine address space.
//*****************************************************************
//*
//*
//CHECKDFE EXEC PGM=IKJEFT01,
//         PARM='LISTDS ''&COMDS.&SE.'''
//SYSTSIN  DD DUMMY
//SYSTSPRT DD DUMMY
//*
//         IF (CHECKDFE.RC=0) THEN
//*
//*****************************************************************
//* Broker MQ and environment verification
//*****************************************************************
//*
//VERIFY   EXEC PGM=BPXBATSL,REGION=0M,TIME=NOLIMIT,
//         PARM='PGM &INSTP./bin/mqsicvp ++COMPONENTNAME++'
//*        MQSeries Runtime Libraries
//STEPLIB  DD DSN=&WMQHLQ..SCSQANL++LANGLETTER++,DISP=SHR
//         DD DSN=&WMQHLQ..SCSQAUTH,DISP=SHR
//         DD DSN=&WMQHLQ..SCSQLOAD,DISP=SHR
//STDENV   DD PATH='&HOME./ENVFILE'
//STDOUT   DD SYSOUT=*
//STDERR   DD SYSOUT=*
//*
//*****************************************************************
//* Step to delete residual locks
//* (this is only needed if the broker is ARM enabled)
```

```
//*************************************************************
//*
//*RMLOCKS  EXEC PGM=BPXBATCH,REGION=0M,TIME=NOLIMIT,
//*         PARM='SH rm -f &COMPDIR./common/locks/*'
//*
//          ENDIF
//*
//*************************************************************
//* Check RCs from previous steps and call correct program
//* with the correct ENVFILE
//*************************************************************
//*
//         IF (CHECKDFE.RC EQ 0) AND (VERIFY.RC EQ 0) THEN
//*
//*************************************************************
//* Start Control Address Space with base ENVFILE
//* (bipimain, bipservice and bipbroker)
//*************************************************************
//*
//BROKER   EXEC PGM=BPXBATA8,REGION=0M,TIME=NOLIMIT,
//         PARM='PGM &INSTP./bin/&MAINP. &SRVMP. &COMPK. &STRTP.'
//*        MQSeries Runtime Libraries
//STEPLIB  DD DSN=&WMQHLQ..SCSQANL++LANGLETTER++,DISP=SHR
//         DD DSN=&WMQHLQ..SCSQAUTH,DISP=SHR
//         DD DSN=&WMQHLQ..SCSQLOAD,DISP=SHR
//STDENV   DD PATH='&HOME./ENVFILE'
//STDOUT   DD SYSOUT=* //STDERR   DD SYSOUT=*
//*
//         ELSE
//         IF (CHECKENV.RC EQ 0) AND (CHECKDFE.RC NE 0) THEN
//*
//*************************************************************
//* Start DataFlowEngine Address Space with specific ENVFILE
//* (bipimain and DataFlowEngine)
//*************************************************************
//*
//EGENV    EXEC PGM=BPXBATA8,REGION=0M,TIME=NOLIMIT,
//         PARM='PGM &INSTP./bin/&MAINP. &SRVMP. &SE. &SH. &COMPK.
//             &STRTP.'
//*        MQSeries Runtime Libraries
//STEPLIB  DD DSN=&WMQHLQ..SCSQANL++LANGLETTER++,DISP=SHR
//         DD DSN=&WMQHLQ..SCSQAUTH,DISP=SHR
//         DD DSN=&WMQHLQ..SCSQLOAD,DISP=SHR
//*        DB2 Runtime Libraries
//*        Database nodes require DB2 to connect to a datasource.
//*        Note:
//*        DB2 must be included in the STEPLIB if database
//*        nodes are deployed to the broker. Also change EGNOENV.
//*        DD DISP=SHR,DSN=&DB2HLQ..SDSNEXIT
//*        DD DISP=SHR,DSN=&DB2HLQ..SDSNLOAD
//*        DD DISP=SHR,DSN=&DB2HLQ..SDSNLOD2
//*        APF Authorized Library of Message Broker
//*        Required if using Event Notification
//*        (All librarys in concatenation
//*         need to be APF authorized)
//*        DD DISP=SHR,DSN=++WMBHLQ++.SBIPAUTH
//STDENV   DD PATH='&HOME./ENVFILE.&E.'
//STDOUT   DD SYSOUT=*
//STDERR   DD SYSOUT=*
//* DD to enable DB2 ODBC trace for each Execution Group
//*APPLTRC  DD PATHOPTS=(OWRONLY,OCREAT,OAPPEND),
//*            PATHMODE=(SIRWXU,SIRWXG),
//*            PATH='&COMPDIR./output/db2appltrace.&SE..&E.'
//*
//         ELSE
//         IF (CHECKDFE.RC NE 0) THEN
//*
```

```
//*************************************************************
//* Start DataFlowEngine Address Space with base ENVFILE
//* (bipimain and DataFlowEngine)
//*************************************************************
//*
//EGNOENV  EXEC PGM=BPXBATA8,REGION=0M,TIME=NOLIMIT,
//         PARM='PGM &INSTP./bin/&MAINP. &SRVMP. &SE. &SH. &COMPK.
//             &STRTP.'
//*        MQSeries Runtime Libraries
//STEPLIB  DD DSN=&WMQHLQ..SCSQANL++LANGLETTER++,DISP=SHR
//         DD DSN=&WMQHLQ..SCSQAUTH,DISP=SHR
//         DD DSN=&WMQHLQ..SCSQLOAD,DISP=SHR
//*        DB2 Runtime Libraries
//*        Database nodes require DB2 to connect to a datasource.
//*        Note:
//*        DB2 must be included in the STEPLIB if database
//*        nodes are deployed to the broker. Also change EGENV.
//*        DD DISP=SHR,DSN=&DB2HLQ..SDSNEXIT
//*        DD DISP=SHR,DSN=&DB2HLQ..SDSNLOAD
//*        DD DISP=SHR,DSN=&DB2HLQ..SDSNLOD2
//*        APF Authorized Library of Message Broker
//*        Required if using Event Notification
//*        (All librarys in concatenation
//*         need to be APF authorized)
//*        DD DISP=SHR,DSN=++WMBHLQ++.SBIPAUTH
//STDENV   DD PATH='&HOME./ENVFILE'
//STDOUT   DD SYSOUT=*
//STDERR   DD SYSOUT=*
//* DD to enable DB2 ODBC trace for each Execution Group
//*APPLTRC  DD PATHOPTS=(OWRONLY,OCREAT,OAPPEND),
//*            PATHMODE=(SIRWXU,SIRWXG),
//*            PATH='&COMPDIR./output/db2appltrace.&SE..&E.'
//*
//         ENDIF
//         ENDIF
//         ENDIF
//*
//*-------------------------------------------------------------
//         PEND
//*-------------------------------------------------------------
//*
//
```

Related reference:

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

"**mqsideletebroker** command" on page 3863
Use the **mqsideletebroker** command to delete a named broker. The command also deletes the queues on the associated queue manager (created when the broker was created). You can also specify that the queue manager is to be deleted.

Sample BIPBUBK file

The sample BIPBUBK file that is shipped with WebSphere Message Broker is included here for your reference.

```
//BIPBUBK JOB
//*************************************************************
//*                                                           *
//* @START_COPYRIGHT@                                         *
//*                                                           *
//* Licensed Materials - Property of IBM;                     *
//* ProgIds: 5724-J06 5724-J05 5724-J04 5697-J09 5655-M74 5655-M75 5655-G97
//* (C) Copyright IBM Corporation 2008.
//* All Rights Reserved;                                      *
//* US Government Users Restricted Rights - use,              *
```

```
//* duplication or disclosure restricted by GSA                    *
//* ADP Schedule Contract with IBM Corp.;                          *
//* See Copyright Instructions                                     *
//*                                                                *
//* @END_COPYRIGHT@                                               *
//*                                                                *
//*****************************************************************
//*              IBM WebSphere Message Broker                      *
//*                                                                *
//* Sample job to backup a broker                                  *
//* (mqsibackupbroker)                                             *
//*                                                                *
//*****************************************************************
//* MORE INFORMATION - See:                                        *
//*                                                                *
//*    WebSphere Message Broker Information Centre.                *
//*                                                                *
//*****************************************************************
//* CUSTOMIZE THIS JCL HERE FOR YOUR INSTALLATION
//* YOU MUST DO GLOBAL CHANGES ON THESE PARAMETERS USING YOUR EDITOR
//*                                                                *
//*    Replace   ++HOME++
//*                      Unique home directory where ENVFILE
//*                      will be created.
//*                      e.g. '/u/mq01usr/mq01brk'
//*
//*    Replace   ++INSTALL++
//*                      WMB installation directory.
//*                      e.g. '/usr/lpp/mqsi'
//*
//*    Replace   ++COMPONENTNAME++
//*                      Broker Name.
//*                      e.g. 'MQ01'
//*
//*    Replace   ++DIRECTORYPATHNAME++
//*                      Directory/Path name where the backup is
//*                      placed.
//*                      e.g. '/u/mq01usr/mq01brk/db'
//*
//*    Replace   ++ARCHIVENAME++
//*                      Archive name.
//*                      e.g. '-a archive1'
//*
//*    Replace   ++WMQHLQ++
//*                      WebSphere MQ high-level-qualifier.
//*                      e.g. 'MQM.V701'
//*
//*    Replace   ++LANGLETTER++
//*                      The letter for the language that
//*                      you want messages shown in.
//*                      e.g. 'E' for English
//*
//*****************************************************************
//*
//*****************************************************************
//* Copy ENVFILE to SYSOUT
//*****************************************************************
//*
//COPYENV  EXEC PGM=IKJEFT01,
//         PARM='OCOPY INDD(BIPFROM) OUTDD(ENVFILE)'
//SYSTSPRT DD DUMMY
//BIPFROM  DD PATHOPTS=(ORDONLY),
//            PATH='++HOME++/ENVFILE'
//ENVFILE  DD SYSOUT=*,DCB=(RECFM=V,LRECL=256)
//SYSTSIN  DD DUMMY
//*
//*****************************************************************
```

```
//* Run mqsibackupbroker command
//*****************************************************************
//*
//BIPBUBK  EXEC PGM=IKJEFT01,REGION=0M
//*        MQSeries Runtime Libraries
//STEPLIB  DD DISP=SHR,DSN=++WMQHLQ++.SCSQANL++LANGLETTER++
//         DD DISP=SHR,DSN=++WMQHLQ++.SCSQAUTH
//         DD DISP=SHR,DSN=++WMQHLQ++.SCSQLOAD
//STDENV   DD PATHOPTS=(ORDONLY),
//            PATH='++HOME++/ENVFILE'
//STDOUT   DD SYSOUT=*
//STDERR   DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
BPXBATSL PGM -
  ++INSTALL++/bin/-
mqsibackupbroker -
  ++COMPONENTNAME++ -
  -d ++DIRECTORYPATHNAME++ -
  -a ++ARCHIVENAME++
/*
//
```

Related tasks:

"Customizing the broker component data set" on page 624
This is part of the larger task of creating a broker on z/OS.

Related reference:

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsibackupbroker** command" on page 3720
Use the **mqsibackupbroker** command to back up the current configuration of a broker.

"**mqsirestorebroker** command" on page 3952
Use the **mqsirestorebroker** command to restore the broker configuration from a backup file.

Sample BIPCRBK file

The sample BIPCRBK file that is shipped with WebSphere Message Broker is included here for your reference.

```
//BIPCRBK JOB
//*****************************************************************
//*                                                               *
//* @START_COPYRIGHT@                                            *
//*                                                               *
//* Licensed Materials - Property of IBM;                         *
//* ProgIds: 5724-J06 5724-J05 5724-J04 5697-J09 5655-M74 5655-M75 5655-G97
//* (C) Copyright IBM Corporation 2004.
//* All Rights Reserved;                                          *
//* US Government Users Restricted Rights - use,                  *
//* duplication or disclosure restricted by GSA                   *
//* ADP Schedule Contract with IBM Corp.;                         *
//* See Copyright Instructions                                    *
//*                                                               *
//* @END_COPYRIGHT@                                              *
//*                                                               *
//*****************************************************************
//*              IBM WebSphere Message Broker                     *
//*                                                               *
//* Sample job to create a broker (mqsicreatebroker).             *
//*                                                               *
//*****************************************************************
//* MORE INFORMATION - See:                                       *
```

```
//*                                                                   *
//*    WebSphere Message Broker Information Centre.                   *
//* Topic "an07080"                                                   *
//*                                                                   *
//*********************************************************************
//* CUSTOMIZE THIS JCL HERE FOR YOUR INSTALLATION
//* YOU MUST DO GLOBAL CHANGES ON THESE PARAMETERS USING YOUR EDITOR
//*                                                                   *
//*    Replace   ++HOME++
//*                      Unique home directory where ENVFILE
//*                      will be created.
//*                      e.g. '/u/mq01usr/mq01brk'
//*
//*    Replace   ++INSTALL++
//*                      WMB installation directory.
//*                      e.g. '/usr/lpp/mqsi'
//*
//*    Replace   ++COMPONENTNAME++
//*                      Broker name.
//*                      e.g. 'MQ01BRK'
//*
//*    Replace   ++QUEUEMANAGER++
//*                      Queue manager name.
//*                      e.g. 'MQ01'
//*
//*    Replace   ++OPTIONS++
//*                      Options for mqsicreatebroker command.
//*                      e.g. '-1'
//*
//*                      z/OS specific options are
//*                          -1 Registry pass only
//*                             This creates the broker directory.
//*                          -2 MQ pass only
//*                             This creates the broker MQ queues.
//*
//*                      Please see documentation for other options.
//*
//*    Replace   ++WMQHLQ++
//*                      WebSphere MQ high-level-qualifier.
//*                      e.g. 'MQM.V701'
//*
//*    Replace   ++LANGLETTER++
//*                      The letter for the language that
//*                      you want messages shown in.
//*                      e.g. 'E' for English
//*
//*********************************************************************
//*
//*********************************************************************
//* Copy ENVFILE to SYSOUT
//*********************************************************************
//*
//COPYENV  EXEC PGM=IKJEFT01,
//         PARM='OCOPY INDD(BIPFROM) OUTDD(ENVFILE)'
//SYSTSPRT DD DUMMY
//BIPFROM  DD PATHOPTS=(ORDONLY),
//            PATH='++HOME++/ENVFILE'
//ENVFILE  DD SYSOUT=*,DCB=(RECFM=V,LRECL=256)
//SYSTSIN  DD DUMMY
//*
//*********************************************************************
//* Run mqsicreatebroker command
//*********************************************************************
//*
//BIPCRBK  EXEC PGM=IKJEFT01,REGION=0M
//*        MQSeries Runtime Libraries
//STEPLIB  DD DISP=SHR,DSN=++WMQHLQ++.SCSQANL++LANGLETTER++
```

```
//         DD DISP=SHR,DSN=++WMQHLQ++.SCSQAUTH
//         DD DISP=SHR,DSN=++WMQHLQ++.SCSQLOAD
//STDENV   DD PATHOPTS=(ORDONLY),
//            PATH='++HOME++/ENVFILE'
//STDOUT   DD SYSOUT=*
//STDERR   DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
BPXBATSL PGM -
  ++INSTALL++/bin/-
mqsicreatebroker -
  ++COMPONENTNAME++ -
  -q ++QUEUEMANAGER++ -
  ++OPTIONS++
/*
//
```

Related reference:

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

Sample BIPDSNAO file

The sample BIPDSNAO file that is shipped with WebSphere Message Broker is included here for your reference.

```
;*********************************************************************
;*                                                                   *
;* @START_COPYRIGHT@                                                *
;*                                                                   *
;* Licensed Materials - Property of IBM;                             *
;* 5655-G97 (c) Copyright IBM Corp. 2004;                            *
;* All Rights Reserved;                                              *
;* US Government Users Restricted Rights - use,                      *
;* duplication or disclosure restricted by GSA                       *
;* ADP Schedule Contract with IBM Corp.;                             *
;* See Copyright Instructions                                        *
;*                                                                   *
;* @END_COPYRIGHT@                                                  *
;*                                                                   *
;*********************************************************************
;*              IBM WebSphere Message Broker                         *
;*                                                                   *
;* Sample dsnaoini for a broker.                                     *
;*                                                                   *
;* (ONLY REQUIRED FOR USER DATABASES)                                *
;*                                                                   *
;*********************************************************************
;* MORE INFORMATION - See:                                           *
;*                                                                   *
;*    WebSphere Message Broker Information Centre.                   *
;*                                                                   *
;*********************************************************************
;* CUSTOMIZE HERE FOR YOUR INSTALLATION
;* YOU MUST DO GLOBAL CHANGES ON THESE PARAMETERS USING YOUR EDITOR
;*                                                                   *
;*    Replace   ++DB2SUBSYSTEM++
;*                      DB2 SSID.
;*                      e.g. 'DFK4'
;*
;*    Replace   ++DB2LOCATION++
;*                      DB2 location.
;*                      e.g. 'DSN810PK'
;*
;*    Replace   ++DB2CURRENTSQLID++
;*                      CURRENT SQLID user ID.
;*                      e.g. 'MQ01GRP'
```

```
;*
;*    Replace   ++DB2DSNACLIPLAN++
;*                      DB2 plan name for dsnacli.
;*                      e.g. 'DSNACLI'
;*
;*****************************************************************
[COMMON]
APPLTRACE=0
APPLTRACEfilename="DD:APPLTRC"
TRACETIMESTAMP=3
CONNECTTYPE=2
DIAGTRACE=0
DIAGTRACE_NO_WRAP=0
MAXCONN=0
MULTICONTEXT=0
MVSDEFAULTSSID=++DB2SUBSYSTEM++

; SUBSYSTEM
[++DB2SUBSYSTEM++]
MVSATTACHTYPE=RRSAF
PLANNAME=++DB2DSNACLIPLAN++

; DATASOURCES
[++DB2LOCATION++]
CURRENTSQLID=++DB2CURRENTSQLID++
```

Related tasks:

"Connecting to a user database on z/OS" on page 696
Complete these tasks to connect to your user databases on z/OS.

Sample BIPEDIT file

The sample BIPEDIT file that is shipped with WebSphere Message Broker is included here for your reference.

```
/* REXX  */
/*********************************************************************/
/*                                                                   */
/*  @START_COPYRIGHT@                                                */
/*                                                                   */
/*  Licensed Materials - Property of IBM;                            */
/* ProgIds: 5724-J06 5724-J05 5724-J04 5697-J09 5655-M74 5655-M75 5655-G97*/
/* (C) Copyright IBM Corporation 2004.                               */
/*  All Rights Reserved;                                             */
/*  US Government Users Restricted Rights - use,                     */
/*  duplication or disclosure restricted by GSA                      */
/*  ADP Schedule Contract with IBM Corp.;                            */
/*  See Copyright Instructions                                       */
/*                                                                   */
/*  @END_COPYRIGHT@                                                  */
/*                                                                   */
/*********************************************************************/
/*              IBM WebSphere Message Broker                         */
/*                                                                   */
/* REXX utility to customize JCL.                                    */
/*                                                                   */
/*********************************************************************/
/* MORE INFORMATION - See:                                           */
/*                                                                   */
/*    WebSphere Message Broker Information Centre.                   */
/*                                                                   */
/*********************************************************************/
/*                                                                   */
/*  This edit macro can be used to assist you in configuring your    */
/*  Broker configuration files.                                      */
/*  1. Edit this file to alter the change commands so the second     */
/*     parameter is the value for your installaton                   */
/*  2. Save the file                                                 */
```

```
/*  3. Rename it to a broker related file, for example VCP1EDIT for     */
/*     Broker called VCP1BRK                                            */
/*  4. In TSO or TSO command shell ( often ISPF option 6) execute       */
/*     ALTLIB ACTIVATE APPLICATION(EXEC) DA('WMB.V7.SBIPPROC')          */
/*     where WMB.V7.SBIPPROC is the name of the PDS contaning this      */
/*     macro                                                            */
/*  5. Using the same ISPF session, edit a configuration file           */
/*  6. Type the macro name at the commands line and press enter         */
/*     This will execute the macro.                                     */
/*     If you get Command not found, then check you issued the command  */
/*     in 4. in the same ISPF session                                   */
/*  7. If the macro has any errors, cancel from the file being edited   */
/*     resolve the errors in the macro and retry                        */
/*                                                                      */
/************************************************************************/
/*   See the product docmentation on the meaning of the fields below:   */
/************************************************************************/
ISREDIT MACRO NOPROCESS
ADDRESS ISREDIT
"change ++INSTALL++ install_value                       all"
"change ++COMPONENTDIRECTORY++ compdir_value            all"
"change ++COMPONENTNAME++ MQ01BRK                       all"
"change ++HOME++ /u/mq01brk                             all"
"change ++OPTIONS++ options_value                       all"
"change ++LOCALE++ C                                    all"
"change ++TIMEZONE++ GMT0BST                            all"
"change ++JAVA++ /usr/lpp/java/IBM/J1.6                 all"
"change ++WMQHLQ++ MQM.V701                             all"
"change ++LANGLETTER++ E                                all"
"change ++QUEUEMANAGER++ MQ01                           all"
"change ++COMPONENTDATASET++ componentdataset_value     all"
"change ++STARTEDTASKNAME++ MQ01BRK                     all"
"change ++MQPATH++ /usr/lpp/mqm                         all"
/************************************************************************/
/* Database nodes and mqsimigratecomponents (BIPMGCMP) require DB2 in   */
/* order to connect to a datasource.                                    */
/************************************************************************/
/*"change ++DB2HLQ++ SYS2.DB2.V910                         all"        */
```

Related tasks:

"Customizing the broker JCL" on page 625
This subtask is part of the larger task of creating a broker on z/OS.

Related reference:

"Contents of the broker PDSE" on page 3991
After you have successfully customized the broker, the broker PDSE members have been set up.

Sample BIPGEN file

The sample BIPGEN file that is shipped with WebSphere Message Broker is included here for your reference.

```
//BIPGEN JOB
//*******************************************************************
//*                                                                 *
//* @START_COPYRIGHT@                                               *
//*                                                                 *
//* Licensed Materials - Property of IBM;                           *
//* ProgIds: 5724-J06 5724-J05 5724-J04 5697-J09 5655-M74 5655-M75 5655-G97
//* (C) Copyright IBM Corporation 2004,2010
//* All Rights Reserved;                                            *
//* US Government Users Restricted Rights - use,                    *
//* duplication or disclosure restricted by GSA                     *
//* ADP Schedule Contract with IBM Corp.;                           *
//* See Copyright Instructions                                      *
//*                                                                 *
```

```
//* @END_COPYRIGHT@                                                *
//*                                                               *
//*****************************************************************
//*              IBM WebSphere Message Broker                     *
//*                                                               *
//* Copy component profile to the file system and generate an     *
//* ENVFILE.                                                      *
//*                                                               *
//* IMPORTANT:                                                    *
//*                                                               *
//*     You must submit BIPGEN each time you update a profile!    *
//*                                                               *
//*****************************************************************
//* MORE INFORMATION - See:                                       *
//*                                                               *
//*    WebSphere Message Broker Information Centre.               *
//*                                                               *
//*****************************************************************
//* CUSTOMIZE THIS JCL HERE FOR YOUR INSTALLATION
//* YOU MUST DO GLOBAL CHANGES ON THESE PARAMETERS USING YOUR EDITOR
//*                                                               *
//*    Replace   ++HOME++
//*                      Unique home directory where ENVFILE
//*                      will be created.
//*                      e.g. '/u/mq01usr/mq01brk'
//*
//*    Replace   ++COMPONENTDATASET++
//*                      Component dataset.
//*                      e.g. 'BIP.BROKER.MQ01BRK'
//*
//*****************************************************************
//*
//PROCLIB  JCLLIB ORDER=(++COMPONENTDATASET++)
//*
//*****************************************************************
//* Copy BIPBPROF to file system
//*****************************************************************
//*
//COPYPROF EXEC PGM=IKJEFT01,
//         PARM='OCOPY INDD(BIPFROM) OUTDD(BIPPROF)'
//SYSTSPRT DD DUMMY
//BIPFROM  DD DISP=SHR,DSN=++COMPONENTDATASET++(BIPBPROF)
//BIPPROF  DD PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//            PATHMODE=(SIRWXU,SIRWXG),
//            PATH='++HOME++/bipprof'
//SYSTSIN  DD DUMMY
//*
//*****************************************************************
//* Copy BIPPROF to SYSOUT
//*****************************************************************
//*
//COPYENV  EXEC PGM=IKJEFT01,
//         PARM='OCOPY INDD(BIPFROM) OUTDD(BIPPROF)'
//SYSTSPRT DD DUMMY
//BIPFROM  DD PATHOPTS=(ORDONLY),
//            PATH='++HOME++/bipprof'
//BIPPROF  DD SYSOUT=*,DCB=(RECFM=V,LRECL=256)
//SYSTSIN  DD DUMMY
//*
//*****************************************************************
//* Create ENVFILE from BIPPROF
//*****************************************************************
//*
//CREATENV EXEC PGM=IKJEFT01,REGION=0M
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
  BPXBATCH SH -
```

```
  . ++HOME++/bipprof; -
  /bin/printenv > -
  ++HOME++/ENVFILE
//*
//*****************************************************************
//* Copy ENVFILE to SYSOUT
//*****************************************************************
//*
//COPYENV  EXEC PGM=IKJEFT01,
//         PARM='OCOPY INDD(BIPFROM) OUTDD(ENVFILE)'
//SYSTSPRT DD DUMMY
//BIPFROM  DD PATHOPTS=(ORDONLY),
//            PATH='++HOME++/ENVFILE'
//ENVFILE  DD SYSOUT=*,DCB=(RECFM=V,LRECL=256)
//SYSTSIN  DD DUMMY
//*
//*****************************************************************
//* For any execution group specific profiles, add additional steps
//* below.
//* These must call BIPEGEN to create the execution group specific
//* ENVFILEs, passing in the name of the EG (last 8 characters as
//* defined in the infocenter). Also a member of this name must
//* exist in the component dataset.
//*
//* See the Message Broker infocenter for more information on
//* execution group specifc profiles on z/OS.
//*****************************************************************
//*BIPEG01 EXEC PROC=BIPEGEN,EG=DEFAULT
//*
//
```

Related reference:

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

"**mqsideletebroker** command" on page 3863
Use the **mqsideletebroker** command to delete a named broker. The command also deletes the queues on the associated queue manager (created when the broker was created). You can also specify that the queue manager is to be deleted.

Sample BIPRSBK file

The sample BIPRSBK file that is shipped with WebSphere Message Broker is included here for your reference.

```
//BIPRSBK JOB
//*****************************************************************
//*                                                               *
//* @START_COPYRIGHT@                                            *
//*                                                               *
//* Licensed Materials - Property of IBM;                         *
//* ProgIds: 5724-J06 5724-J05 5724-J04 5697-J09 5655-M74 5655-M75 5655-G97
//* (C) Copyright IBM Corporation 2008.
//* All Rights Reserved;                                          *
//* US Government Users Restricted Rights - use,                  *
//* duplication or disclosure restricted by GSA                   *
//* ADP Schedule Contract with IBM Corp.;                         *
//* See Copyright Instructions                                    *
//*                                                               *
//* @END_COPYRIGHT@                                              *
//*                                                               *
//*****************************************************************
//*              IBM WebSphere Message Broker                     *
//*                                                               *
//* Sample job to restore a broker                                *
//* (mqsirestorebroker)                                           *
//*                                                               *
```

```
//*****************************************************************
//* MORE INFORMATION - See:                                       *
//*                                                               *
//*    WebSphere Message Broker Information Centre.               *
//*                                                               *
//*****************************************************************
//* CUSTOMIZE THIS JCL HERE FOR YOUR INSTALLATION
//* YOU MUST DO GLOBAL CHANGES ON THESE PARAMETERS USING YOUR EDITOR
//*                                                               *
//*    Replace   ++HOME++
//*                      Unique home directory where ENVFILE
//*                      will be created.
//*                      e.g. '/u/mq01usr/mq01brk'
//*
//*    Replace   ++INSTALL++
//*                      Message Broker installation directory.
//*                      e.g. '/usr/lpp/mqsi'
//*
//*    Replace   ++COMPONENTNAME++
//*                      Broker Name.
//*                      e.g. 'MQ01'
//*
//*    Replace   ++DIRECTORYPATHNAME++
//*                      Directory/Path name where the backup is.
//*                      e.g. '/u/mq01usr/mq01brk/db'
//*
//*    Replace   ++ARCHIVENAME++
//*                      Archive name.
//*                      e.g. '-a archive1'
//*
//*    Replace   ++WMQHLQ++
//*                      WebSphere MQ high-level-qualifier.
//*                      e.g. 'MQM.V701'
//*
//*    Replace   ++LANGLETTER++
//*                      The letter for the language that
//*                      you want messages shown in.
//*                      e.g. 'E' for English
//*
//*****************************************************************
//*
//*****************************************************************
//* Copy ENVFILE to SYSOUT
//*****************************************************************
//*
//COPYENV  EXEC PGM=IKJEFT01,
//         PARM='OCOPY INDD(BIPFROM) OUTDD(ENVFILE)'
//SYSTSPRT DD DUMMY
//BIPFROM  DD PATHOPTS=(ORDONLY),
//            PATH='++HOME++/ENVFILE'
//ENVFILE  DD SYSOUT=*,DCB=(RECFM=V,LRECL=256)
//SYSTSIN  DD DUMMY
//*
//*****************************************************************
//* Run mqsirestorebroker command
//*****************************************************************
//*
//BIPRSBK  EXEC PGM=IKJEFT01,REGION=0M
//*        MQSeries Runtime Libraries
//STEPLIB  DD DISP=SHR,DSN=++WMQHLQ++.SCSQANL++LANGLETTER++
//         DD DISP=SHR,DSN=++WMQHLQ++.SCSQAUTH
//         DD DISP=SHR,DSN=++WMQHLQ++.SCSQLOAD
//STDENV   DD PATHOPTS=(ORDONLY),
//            PATH='++HOME++/ENVFILE'
//STDOUT   DD SYSOUT=*
//STDERR   DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
```

```
//SYSTSIN  DD *
BPXBATSL PGM -
  ++INSTALL++/bin/-
mqsirestorebroker -
  ++COMPONENTNAME++ -
  -d ++DIRECTORYPATHNAME++ -
  -a ++ARCHIVENAME++
/*
//
```

Related tasks:

"Customizing the broker component data set" on page 624
This is part of the larger task of creating a broker on z/OS.

Related reference:

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsibackupbroker** command" on page 3720
Use the **mqsibackupbroker** command to back up the current configuration of a broker.

"**mqsirestorebroker** command" on page 3952
Use the **mqsirestorebroker** command to restore the broker configuration from a backup file.

Data sources on z/OS

The `Data Source` name in the Compute and Database nodes identifies the location of the table referred to in the respective node's ESQL. Data sources on z/OS correspond to DB2 subsystems rather than DB2 databases. The DB2 owning region for a particular database table is identified using a combination of the DSNAOINI file and DB2 subsystem configuration.

The MVSDEFAULTSSID parameter in the DSNAOINI file identifies the local DB2 subsystem to which the broker is connected. This subsystem is used to locate the data source which is either a local or remote DB2. The mapping between a particular data source and DB2 subsystem is shown in the DSNTIPR installation panel of the default DB2 subsystem and SYSIBM.LOCATIONS table.

When you access remote DB2 subsystems, ensure that the DBRMs for ODBC are bound at the remote subsystem. For more information, refer to the 'Programming for ODBC' topics in the DB2 Information Management Software Information Center for z/OS Solutions .

If you need to access databases that are not on DB2 on z/OS, you can use the DB2 Distributed Data Facility (DDF) and Distributed Relational Architecture (DRDA) to incorporate a remote unit of work within a message flow.

Related tasks:

"Accessing databases from message flows" on page 2112
Create and configure message flows to access user databases.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

Message flow development

View the reference material associated with developing message flow applications.

- "Message flows"
-
- "Built-in nodes" on page 4293
- "Transformation interfaces" on page 4980
- "User-defined patterns" on page 5344
- "Message model reference information" on page 5366
- "Publish/subscribe" on page 6395
- "User-defined extensions" on page 6411
- "Web services external standards" on page 6696

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Developing user-defined extensions" on page 2970
A user-defined extension is a component that you design and implement to extend the function of WebSphere Message Broker.

Related tasks:

"Processing messages" on page 1021
Process your business messages and data by interacting with a broker, which you can configure to provide services and to communicate with other applications and systems.

"Transforming and enriching messages" on page 2227
Transform and enrich messages by using one or more of the techniques described in this section.

Message flows

Use the reference information in this section to develop your message flows and related resources.

The following message flow reference information is available:

- "Message flow preferences" on page 4016
- "Description properties for a message flow" on page 4016
- "Configurable message flow properties" on page 4020
- "WebSphere Adapters properties" on page 4024
- "Validation properties" on page 4169
- "Parsing on demand" on page 4173
- "Impact analysis: reference" on page 4174
- "User-defined nodes" on page 6415
- "Supported code pages" on page 4176
- "WebSphere MQ connections" on page 4222
- "Data integrity in message flows" on page 4223
- "Exception list structure" on page 4224
- "Message flow porting" on page 4233
- "Coordinated message flows" on page 4234
- "Element definitions for message parsers" on page 4237
- "XML constructs" on page 4257

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Using message mappings" on page 2228
Message mappings define the blueprint for creating a message.

"The Message Flow editor experiences problems when opening a message flow, and opens in error mode" on page 3435

Message flow preferences

You can change preferences that determine properties of message flows when you create them.

To edit message flow preferences, open the Preferences dialog box by clicking **Window** > **Preferences**. Select **Message Flow** from the list of categories.

For a new message flow, you can set the default version tag, which is described in the following table.

Property	Type	Meaning
Default version tag	String	Provide the default version information that you want to be set in the message flow `Version` property when you create a message flow.

Related concepts:

"Message flow version and keywords" on page 1445
When you are developing a message flow, you can define the version of the message flow as well as other key information that you want to be associated with it.

"Version and keyword information for deployable objects" on page 1443
Use the Broker Archive editor to view the version and keyword information of deployable objects.

Related tasks:

"Configuring CVS to run with the WebSphere Message Broker Toolkit" on page 573
Install CVS as a normal program by following the usual prompts. Not all versions of *CVSNT* are supported by Eclipse.

Related reference:

"Description properties for a message flow"
The description properties for a message flow include the `Version`, `Short Description` and `Long Description`. To view and edit the properties of a message flow click **Flow** > **Properties**.

Description properties for a message flow

The description properties for a message flow include the `Version`, `Short Description` and `Long Description`. To view and edit the properties of a message flow click **Flow** > **Properties**.

Property	Type	Meaning
Version	String	You can enter a version for the message flow in this field. This allows the version of the message flow to be displayed using the Eclipse properties view. A default for this field can be set in the messages flow preferences.
Short Description	String	You can enter a short description of the message flow in this field.
Long Description	String	You can add information to enhance the understanding of the message flow's function in this field. It is a string field and any standard alphanumeric characters can be used. You can also use this field to define a keyword and its value that will display for the deployed message flow in the properties view of Eclipse. An example is: `$MQSI Author=Fred MQSI$` When the properties of the deployed message flow are displayed, this will add a row to the display showing "Author" as the property name and "Fred" as its value. For information on keywords see "Guidance for defining keywords."

Related concepts:

"Version and keyword information for deployable objects" on page 1443
Use the Broker Archive editor to view the version and keyword information of deployable objects.

Related reference:

"Guidance for defining keywords"
You can add extra information to an object in the form of one or more keywords.

Guidance for defining keywords:

You can add extra information to an object in the form of one or more keywords.

This information can display information about an object after the object has been deployed. The default information that is displayed is the time the object was deployed and the last time the object was modified.

You can define custom keywords, and their values that the WebSphere Message Broker Toolkit interprets as additional information to be displayed, in the properties view. For example, you can define keywords for "Author" and "Subflow 1 Version":

```
$MQSI Author=John Smith MQSI$
$MQSI Subflow 1 Version=v1.3.2 MQSI$
```

The following table contains the information that is displayed by the WebSphere Message Broker Toolkit:

Object name	Example
Deployment Time	28-Aug-2009 15:04
Modification Time	28-Aug-2009 14:27
Version	v1.0
Author	John Smith

Object name	Example
Subflow 1 Version	v1.3.2

In this display, the version information has been defined by using the **Version** property of the object. If the version information had not been defined, it would be omitted from this display.

Use the following syntax to define a keyword and its associated value:

```
$MQSI KeywordName = KeywordValue MQSI$
```

Where:

$MQSI
$MQSI opens the definition. It can be followed by an optional underscore or white-space character that is ignored.

KeywordName
The name of the keyword for which you are setting the value. It is made up of a sequence of alphanumeric characters apart from the equals (=) sign. It can contain white-space characters, but leading or trailing white-space characters are omitted.

= The equals (=) sign is the delimiter between the keyword and the value that you are setting it to.

KeywordValue
The value to which the keyword is set. It is made up of a sequence of alphanumeric characters. It can contain white-space characters, but leading or trailing white-space characters are omitted.

MQSI$
MQSI$ closes the keyword definition.

Examples

Example definitions	Interpreted keyword and value	Comments
$MQSIAuthor=JohnMQSI$ or $MQSI Author=John MQSI$ or $MQSI Author = John MQSI$	Keyword = "Author" Value = "John"	Each of these examples shows what can be set and that the leading and trailing white-space characters for the name and value parameters are ignored.
$MQSI_Author = John MQSI$	Keyword = "Author" Value = "John"	The first character after $MQSI can be an underscore character. The underscore character is omitted in the interpreted keyword. If a second underscore character appears, this forms part of the keyword name.
$MQSI Flow designer = John Smith MQSI$	Keyword = "Flow designer" Value = "John Smith"	White-space characters are accepted for each parameter value.
$MQSI bar = MQSI$	Keyword = "bar" Value = ""	The keyword value can be set to an empty ("") string.

Example definitions	Interpreted keyword and value	Comments
$MQSI_mqsitag=$MQSI$MQSI$	Keyword = "mqsitag" Value = "$"	This example is a poorly formatted definition. After defining the keyword name, the parser is looking to find the delimiters that form the boundary of the value to be set. In this case, the only character before the MQSI$ that closes the definition is a '$', and that is set as the keyword value.
$MQSI=barMQSI$		This pattern is ignored because the keyword name cannot be an empty string.
$MQSItagbarMQSI$		This pattern is ignored because there is not a separator (=) between the keyword name and the keyword value.

Do not use the following keywords:

VERSION
When you use the WebSphere Message Broker Toolkit to edit message flows and dictionaries, it is possible to set the **Version** property in the Properties pane, which you can then view in the Broker Archive file editor. If you set this property, a keyword called VERSION is added to the resulting `.cmf` or dictionary file. For this reason, do not add `$MQSI_VERSION=...MQSI$` to these files.

BAR The BAR keyword is associated with each object automatically when it is deployed and it contains the full path name of the broker archive file that deployed the object.

The values of both keywords are defined programmatically in the class `com.ibm.broker.config.proxy.DeployedObject`.

Restrictions within keywords

Do not use the following characters within keywords because they cause unpredictable behavior:

```
^ $ . | \ < > ? + * = & [ ] ( )
```

You can use these characters in the values that are associated with keywords; for example:

- `$MQSI RCSVER=$id$ MQSI$` is acceptable
- `$MQSI $name=Fred MQSI$` is not acceptable

Related concepts:

"Message flow version and keywords" on page 1445
When you are developing a message flow, you can define the version of the message flow as well as other key information that you want to be associated with it.

"Adding keywords to ESQL files" on page 2486
You can add keywords to ESQL files to contain information that you want to associate with a message flow.

"Keywords in subflows" on page 1447
You can embed keywords in each subflow that you use in a message flow.

"Version and keyword information for deployable objects" on page 1443
Use the Broker Archive editor to view the version and keyword information of deployable objects.

"Message set version and keywords" on page 1169
When you develop a message set, you can define the version of the message set, and other key information that you want to be associated with it.

Related tasks:

"Adding keywords to JAR files" on page 2660
If a BAR file contains JAR files, you can associate keywords with the JAR files.

Related reference:

"Message flow preferences" on page 4016
You can change preferences that determine properties of message flows when you create them.

"Adding keywords to XSL style sheets" on page 4975
Embedded keywords in an XSL style sheet; their location is not restricted. You can also add a keyword as an XML comment.

"Description properties for a message flow" on page 4016
The description properties for a message flow include the `Version`, `Short Description` and `Long Description`. To view and edit the properties of a message flow click **Flow** > **Properties**.

Configurable message flow properties

When you add a message flow to a broker archive (BAR) file in preparation for deploying it to a broker, you can set additional properties that influence its run time operation. These properties are available for review and update when you select the **Manage and Configure** tab for the broker archive file.

Additional Instances

This property specifies the number of additional threads that the broker can use to service the message flow. These additional threads are created only if there are sufficient input messages. You can use up to 256 threads. The default value is 0. Additional threads can increase the throughput of a message flow but you must consider the potential effect on message order.

If the message flow processes WebSphere MQ messages, you can configure the message flow to control the message order. Set the `Order Mode` property on the MQInput node accordingly. You might also need to set the `Commit by Message Group` and `Logical Order` properties.

An MQInput node opens the input queue with MQOO_INPUT_AS_Q_DEF, which uses the DEFSOPT property of the input queue. Therefore, you must ensure that the input queue has been defined with DEFSOPT(SHARED) and with the SHARE property set to enable multiple broker threads to read from the input queue. If these properties are not set in this way, the message flow threads report that the queue is in use (MQRC=2042), and the message flow might stop processing messages on the input queue.

If you have multiple input nodes in your message flow, the available additional threads might not be allocated evenly between the different input nodes. In an extreme case, all the threads might be allocated to a single input node, and only one aspect of message flow throughput is improved. To avoid this problem, you can use the `Additional Instances`

Pool property, together with the Additional Instances property, to allocate a pool of additional instance threads for each input node.

Commit Count
For WebSphere MQ messages, this property specifies how many input messages are processed by a message flow before a sync point is taken (by issuing an MQCMIT).

The default value of 1 is also the minimum permitted value. Change this property to avoid frequent MQCMIT calls when messages are being processed quickly and the lack of an immediate commit can be tolerated by the receiving application.

Use the Commit Interval to ensure that a commit is performed periodically when not enough messages are received to fulfill the Commit Count.

This property has no effect if the message flow does not process WebSphere MQ messages.

Commit Interval
This property specifies the maximum time interval between the last WebSphere MQ message being received, and a sync point being taken. If no message is received in the specified time interval, a sync point is taken (by issuing an MQCMIT command), even if the number of messages processed does not fulfill the value of the Commit Count property.

The time interval is specified in seconds, as a decimal number with a maximum of three decimal places (millisecond granularity). The value must be in the range 0.000 through 60.000. The default value is 0.

This property has no effect if the message flow does not process WebSphere MQ messages, or if the value of the Commit Count is 1

Consumer policy set
This property specifies the consumer policy to use to authenticate, encrypt, and sign messages for the SOAPRequest, SOAPAsyncRequest, and SOAPAsyncResponse nodes.

Consumer policy set bindings
This property associates a policy set binding with a consumer policy set and contains information that is specific to the environment and operating system, such as information about keys.

Coordinated Transaction
This property controls whether the message flow is processed as a global transaction, coordinated by WebSphere MQ. Such a message flow is said to be fully globally coordinated. The default value is No.

Use coordinated transactions only where you need to process the message and any database updates that are performed by the message flow in a single unit-of-work, using a two-phase commit protocol. In this case, both the message is read and the database updates are performed, or neither is done.

If you change this value, ensure that the queue manager for the broker is configured correctly. If you do not set up the queue manager correctly, the broker generates a message when the message flow receives a message to indicate that, although the message flow is to be globally coordinated, the queue manager configuration does not support coordination.

See “Supported databases” on page 3591 for information about which databases are supported as participants in a global transaction, and the

System Administration section of the WebSphere MQ Version 7 Information Center online for how to configure WebSphere MQ and the database managers.

This property has no effect if the message flow does not process WebSphere MQ messages.

Monitoring profile name
This property specifies the name of the monitoring profile configurable service to apply to one or more message flows in a BAR file. The monitoring profile is used to configure your message flows to emit monitoring events.

For more information, see "Configuring monitoring event sources using a monitoring profile" on page 762.

Provider policy set
This property specifies the provider policy to use to authenticate, encrypt, and sign messages for the SOAPInput and SOAPReply nodes.

Provider policy set bindings
This property associates a policy set binding with a provider policy set and contains information that is specific to the environment and operating system, such as information about keys.

Security profile name
This property specifies a security profile that has authorization enabled so that a message flow can complete authorization with Tivoli Federated Identity Manager (TFIM). You can set a security profile on a message flow or on individual input nodes. If no security profile is set for the input nodes, the setting is inherited from the setting on the message flow.

For more information, see "Configuring authorization with TFIM V6.1" on page 483.

User-defined properties
The initial value of a user-defined property (UDP) can be modified at design time by the "Message Flow editor" on page 6810, or overridden at deployment time by the "Broker Archive editor" on page 6794. The advantage of UDPs is that their values can be changed by operational staff at deployment time. If, for example, you use UDPs to hold configuration data, you can configure a message flow for a particular computer, task, or environment at deployment time, without having to change the code at the node level. You can also query and set the values of user-defined properties at run time by using the Administration API for WebSphere Message Broker (also known as the CMP API). For example, a systems monitoring tool might use the CMP API to modify the value of a user-defined property at run time to change the behavior of the message flow.

For introductory information about UDPs and dynamic UDPs, see "User-defined properties in ESQL" on page 2376 and "User-defined properties" on page 1147.

For information about configuring UDPs at deployment time, see "Configuring a message flow at deployment time with user-defined properties" on page 2626.

For information about configuring UDPs at run time, see "Setting message flow user-defined properties at run time in a CMP application" on page 985.

You can view and update other configurable properties for the message flow. The properties that are displayed depend on the nodes within the message flow; some have no configurable properties to display. The node properties that are configurable are predominantly system-related properties that are likely to change for each broker to which the message flow is deployed. These properties include data source names and the names of WebSphere MQ queues and queue managers. For full details of configurable properties for a node, see the appropriate node description.

Related concepts:

"User-defined properties in ESQL" on page 2376
Access user-defined properties (UDPs) as variables in your ESQL program by specifying the EXTERNAL keyword on a DECLARE statement. For example, the ESQL statement `DECLARE today EXTERNAL CHARACTER 'monday'` defines a user-defined property called today with an initial value monday.

"User-defined properties" on page 1147
A user-defined property (UDP) is a property that is defined when you construct a message flow by using the Message Flow editor. This property can be used by the ESQL or Java program inside message flow nodes, such as a Compute node.

"Packaging and deployment overview" on page 3210
Deployment is the process of transferring data to an execution group on a broker so that it can take effect in the broker. Message flows and associated resources are packaged in broker archive (BAR) files for deployment.

"The Administration API for WebSphere Message Broker" on page 54
The Administration API for WebSphere Message Broker is a programming interface that your applications can use to control brokers and their resources through a remote interface.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

"Configuring a message flow at deployment time with user-defined properties" on page 2626
Use user-defined properties (UDPs) to configure message flows at deployment and run time, without modifying program code. You can give a UDP an initial value when you declare it in your program, or when you use the Message Flow editor to create or modify a message flow.

"Setting message flow user-defined properties at run time in a CMP application" on page 985
Use the CMP API to query, discover, and set message flow user-defined properties dynamically at run time. You can use the CMP API to set properties with a data type of character.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Related information:

WebSphere MQ Version 7 Information Center online

WebSphere Adapters properties

Reference information about the properties that you set for WebSphere Adapters nodes.

See the properties for the Enterprise Information System (EIS) to which you want to connect:

- "WebSphere Adapter for SAP properties"
- "WebSphere Adapter for Siebel properties" on page 4092
- "WebSphere Adapter for PeopleSoft properties" on page 4122
- "WebSphere Adapter for JD Edwards EnterpriseOne properties" on page 4146

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

"Overview of WebSphere Adapter for Siebel Business Applications" on page 2002
With WebSphere Adapter for Siebel Business Applications, you can create integrated processes that exchange information with a Siebel application, without special coding.

"Overview of WebSphere Adapter for PeopleSoft Enterprise" on page 2013
With the adapter for PeopleSoft Enterprise you can create integrated processes that exchange information with PeopleSoft Enterprise through a standard interface. This interface shields the message flow developer from having to understand the lower level details about implementation of the application or data structures used on the PeopleSoft Enterprise server.

"Overview of WebSphere Adapter for JD Edwards EnterpriseOne" on page 2023
With the WebSphere Adapter for JD Edwards EnterpriseOne, you can create integrated processes that include the exchange of information with a JD Edwards EnterpriseOne server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

WebSphere Adapter for SAP properties:

Reference information to which to refer when you connect to an SAP application.

- "Business object information (SAP)" on page 4025
- "Configuration properties for the WebSphere Adapter for SAP Software" on page 4043
- "SAP options for rediscovery" on page 4090

Related concepts:

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

Related tasks:

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

Related reference:

"WebSphere Adapter for PeopleSoft properties" on page 4122
Reference information to refer to when you connect to a PeopleSoft application.

"WebSphere Adapter for Siebel properties" on page 4092
Reference information to refer to when you connect to a Siebel application.

Business object information (SAP):

A business object contains application-specific information (metadata) about how the adapter processes the business object, and about the operation to be performed on the business object.

The name of the business object is generated by the Adapter Connection wizard in accordance with the naming convention for the adapter.

For more information, see the following topics:

- "Supported data operations (SAP)" on page 4026
- "Naming conventions" on page 4034

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System

(EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

Supported data operations (SAP):

For outbound processing, an operation is the name of the action that is implemented by the adapter so that the message flow can perform the operation on the SAP server.

The adapter uses the application-specific information (ASI) inside the business object definition to implement the operation. The name of the operation typically indicates the type of action to be implemented, such as create or update. For inbound processing, adapters implement an operation by delivering events to their endpoints. For inbound processing, the action that is associated with the event varies depending on the interface (ALE or Advanced event processing). When the interface is ALE, the action is pushed to the adapter and the adapter delivers the event to an endpoint. When the interface is Advanced event processing, the event status is polled by the adapter and processed accordingly.

For more information, see the following topics:

- "Supported data operations on BAPI business objects" on page 4028
- "Supported data operations on ALE business objects" on page 4029

- "Supported data operations of Query interface for SAP Software business objects" on page 4031
- "Supported data operations on Advanced event processing business objects" on page 4032

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

Supported data operations on BAPI business objects:

The operations that are supported by BAPI business objects are the same as those supported by BAPI work units. BAPI result sets support only the RetrieveAll operation.

For BAPI outbound processing, the operation of a BAPI business object is the name of the BAPI call that an adapter issues on the SAP server. The BAPI method determines the operation that is associated with it. The adapter uses the application-specific information (ASI) inside the business object definition to implement the operation. Operations of a business object are called by the component that makes calls to SAP through the adapter. The SAP JCo APIs are used to make the call to the SAP system.

BAPIs and BAPI unit of work

The following table defines operations that the adapter supports for BAPIs and BAPI work units. The definitions in the table are the expected uses for the operations. The action that is taken in the SAP application is based on the meaning of the BAPI itself.

Table 53. Supported operations: BAPI business objects

Operation	Definition
Create	The top-level business object and all contained children are created.
Update	The top-level business object is modified. This operation can include adding and deleting child objects.
Delete	The top-level business object and any contained children are deleted.
Retrieve	The top-level business object and any contained children are retrieved.

For an operation that is not supported, the adapter logs the appropriate error and produces a ResourceException.

Result sets

The following table defines the operation that the adapter supports for BAPI result sets.

Table 54. Supported operation: BAPI result sets

Operation	Definition
RetrieveAll	All the matching records for the BAPI result set are retrieved.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

Supported data operations on ALE business objects:

The operations that are supported by ALE business objects vary, depending on whether the business object is an outbound or inbound object. The adapter uses the application-specific information (ASI) inside the business object definition to implement the operation.

Business objects that are generated with the ALE pass-through IDoc interface are not associated with an operation.

Outbound business objects

The operation of an ALE outbound business object is called by the application component that makes calls to SAP through the adapter. The adapter supports the following outbound operation:

Table 55. Supported operation: ALE outbound business objects

Operation	Definition
Execute	This operation posts the IDoc business object to the SAP application. This operation is one-way and asynchronous. For the `CWYAP_SAPAdapter_TX.rar` version of the adapter, the transaction ID is returned.

Inbound business objects

For ALE inbound business objects, the application-specific information of an operation contains the message type, message code, and message function for an IDoc type. The adapter supports the following inbound operations:

Table 56. Supported operations: ALE inbound business objects

Operation	Definition
Create	The top-level business object and all contained children are created.
Update	The top-level business object is modified. This operation can include adding and deleting child objects.
Delete	The top-level business object and any contained children are deleted.

The adapter uses the IDoc control record field data to determine the operation that is set on the business object before sending it to the endpoint. The following fields in the control record are used to determine the operation:

- Logical_message_type (MESTYP)
- Logical_message_code (MESCOD)
- Logical_message_function(MESFCT)

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

Supported data operations of Query interface for SAP Software business objects:

The SAP Query interface supports the RetrieveAll operation, with which you can have the results of an SAP table returned to you, and the Exists operation, which you use to determine whether data can be found in the SAP table. The adapter uses the application-specific information (ASI) inside the business object definition to implement the operation.

The supported operations for the Query interface for SAP Software are listed in the following table.

Table 57. Supported operations: Query interface for SAP Software business objects

Operation	Description
RetrieveAll	This operation returns a result set in the form of a container of SAP query business objects, which represent the data for each row that is retrieved from the table. If a table business object is sent to the SAP server (instead of a container business object), the rows are returned one at a time.
Exists	This operation provides a means to check for the existence of any records in SAP for defined search criteria. The Exists operation does not return any data; it indicates whether the data exists in SAP. If no data is found, the adapter generates an exception.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

Supported data operations on Advanced event processing business objects:

The operations that are supported by Advanced event processing business objects vary, depending on whether the business object is an outbound or inbound object. The adapter uses the application-specific information (ASI) inside the business object definition to implement the operation.

Outbound business objects

The operation of an Advanced event processing outbound business object is called by the message flow that makes calls to SAP through the adapter. The adapter supports the following outbound operation.

Table 58. Supported operation: Advanced event processing outbound business objects

Operation	Definition
Create	The top-level business object and all contained children are created.
Update	The top-level business object is modified. This operation can include adding and deleting child objects.
Delete	The top-level business object and any contained children are deleted.
Retrieve	The top-level business object and any contained children are retrieved.

Inbound business objects

For Advanced event processing inbound business objects, the application-specific information of an operation contains the message type, message code, and message function for an IDoc type. The adapter supports the following inbound operations.

Table 59. Supported operations: Advanced event processing inbound business objects

Operation	Definition
Create	The top-level business object and all contained children are created.
Update	The top-level business object is modified. This operation can include adding and deleting child objects.
Delete	The top-level business object and any contained children are deleted.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system

administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

Naming conventions:

When the Adapter Connection wizard generates a business object, it provides a name for the business object that is based on the name of the corresponding business function in the SAP server.

The convention that is applied by the SAP server when naming a business object varies depending on whether the name is for a BAPI business object, an ALE business object, an Advanced event processing business object, or a Query interface for SAP Software business object.

For more information, see the following topics:

- "Naming conventions for BAPI business objects" on page 4035
- "Naming conventions for ALE business objects" on page 4038
- "Naming conventions for Query interface for SAP Software business objects" on page 4040
- "Naming conventions for Advanced event processing business objects" on page 4041

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System

(EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

Naming conventions for BAPI business objects:

The Adapter Connection wizard provides names for the business objects for BAPIs, BAPI work units, and BAPI result sets. The business object name reflects the structure of the business function on the SAP server.

BAPIs

When it names business objects for BAPIs, the Adapter Connection wizard adds the prefix `Sap`. The wizard also converts the name of the business function to mixed case, removing any separators such as spaces or underscores, capitalizes the first letter of each word, and it can add an element-specific suffix (for example, `Wrapper` for a top-level business object).

The following table describes the convention that is applied by the Adapter Connection wizard when it names BAPI business objects.

Table 60. Naming conventions for BAPI business objects

Element	Naming convention
Name of the top-level business object	Sap + *Name of the wrapper object that you specify in the Adapter Connection wizard* + Wrapper For example: SapSalesOrderWrapper
Name of the BAPI business object	Sap + *Name of the BAPI interface* For example: SapBapiSalesOrderCreateFromDat1 The top-level object can contain more than one BAPI object.
Name of the child object	Sap + *Name of the Structure/Table* For example: SapReturn

If structures with the same name exist in different BAPIs, or exist in a BAPI (for example, one at the export level and one at the table level), the Adapter Connection wizard adds a unique suffix to differentiate the structures. The first structure is assigned a name (for example, SapReturn) and the second structure is assigned a name such as SapReturn619647890, where 619647890 is the unique identifier that is appended to the name by the wizard.

BAPI work units

The following table describes the convention that is applied by the Adapter Connection wizard when it names a BAPI work unit business object.

Table 61. Naming conventions for BAPI unit of work business objects

Element	Naming convention
Name of the top-level business object	Sap + *Name of the wrapper object that you specify in the Adapter Connection wizard* + Txn For example: SapCustomerTxn
Name of the BAPI business object	Sap + *Name of the BAPI interface* For example: SapCustomer
Name of the child object	Sap + *Name of the Structure/Table* For example: SapReturn

If structures with the same name exist in different BAPIs, or exist in a BAPI (for example, one at the export level and one at the table level), the Adapter Connection wizard adds a unique suffix to differentiate the structures. The first structure is assigned a name (for example, SapReturn) and the second structure is assigned a name such as SapReturn619647890, where 619647890 is the unique identifier that is appended to the name by the wizard.

BAPI result sets

The following table describes the convention that is applied by the Adapter Connection wizard when it names a BAPI result-sets business object.

Table 62. Naming conventions for BAPI result sets

Element	Naming convention
Name of the result set BAPI business object	`Sap + Name of the BAPI interface` For example: `SapBapiCustomerGetDetail`
Name of the child object	`Sap + Name of the Structure/Table` For example: `SapReturn`
Name of the query business object	`Sap + Formatted name of the query BAPI interface` For example: `SapBapiCustomerGetList`

If structures with the same name exist in different BAPIs, or exist in a BAPI (for example, one at the export level and one at the table level), the Adapter Connection wizard adds a unique suffix to differentiate the structures. The first structure is assigned a name (for example, `SapReturn`) and the second structure is assigned a name such as `SapReturn619647890`, where 619647890 is the unique identifier that is appended to the name by the wizard.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection

wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

Naming conventions for ALE business objects:

The Adapter Connection wizard provides names for the ALE business object. The business object name reflects the structure of the business function on the SAP server.

If you are using the ALE pass-through IDoc interface, the following naming conventions apply:

- When you select **Generic IDoc** from the Object Discovery and Selection window, the Adapter Connection wizard creates a business object named `SapGenericIDocObject`. The naming convention described in the following sections does not apply to generic IDocs.
- When you discover an IDoc from the system or from a file, the object is named according to the naming convention for top-level wrapper objects, as described in Table 63. No other objects are generated.

When it names business objects for ALE, the Adapter Connection wizard adds a prefix of `Sap`. The wizard also converts the name of the IDoc and extension to mixed case, removing any separators such as spaces or underscores, capitalizes the first letter of each word, and it can add an element-specific suffix.

The following table describes the conventions that are applied by the Adapter Connection wizard when it names ALE business objects. The *[Name of Extension type IDoc]* in the Naming convention column represents an optional entry. It is included in the name only if the selected IDoc is an Extension Type IDoc.

Table 63. Naming conventions for ALE business objects

Element	Naming convention
Name of the top-level wrapper object	`Sap` + *`Name of IDoc`* + *`[Name of Extension type IDoc]`* For example: `SapAlereq01`
Name of the IDoc business object for basic IDocs	`Sap` + *`Name of IDoc`* + *`BO`* For example, the business object for the IDoc MATMAS03 is: `SapMatmas03BO`
Name of the IDoc business object for extension type IDocs	`Sap` + *`Name of IDoc`* + *`Name of Extension type IDoc`* For example, the business object for the IDoc DELVRY03 and extension SD_DESADV_PDC is: `SapDelvry03SdDesadvPdc`

For an IDoc duplicate name, the Adapter Connection wizard adds a unique suffix to differentiate the business object. If an IDoc packet has two segments with the

same name (for example, segOrder), the first business object is assigned the name `SapSegOrder` and the second business object is assigned a name such as `SapSegOrder619647890`, where 619647890 is the unique identifier that the Adapter Connection wizard appends to the name.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

Naming conventions for Query interface for SAP Software business objects:

The Adapter Connection wizard provides names for the Query interface for SAP Software container, business graph, top-level business object, table object, and query object. The business object name reflects the structure of the business function on the SAP server.

When it names business objects for the Query interface for SAP Software, the Adapter Connection wizard adds the prefix `Sap`. The wizard also converts the name of the business function or SAP table to mixed case, removing any separators such as spaces or underscores, capitalizes the first letter of each word, and it can add an element-specific suffix (for example, `Container` for a container).

The following table describes the convention that is applied by the Adapter Connection wizard when it names a Query interface for SAP Software business object.

Table 64. Naming convention for a Query interface for SAP Software business object

Element	Naming convention
Name of the container	`Sap` + *Name of the object that you specify in the wizard* + `Container` For example: `SapCustomerContainer`
Name of the table object	`Sap` + *Name of the SAP table* For example: `SapKna1`
Name of the query object	`Sap` + *Name of the SAP table* + `Querybo` For example: `SapKna1Querybo`

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system

administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

Naming conventions for Advanced event processing business objects:

The Adapter Connection wizard provides names for the Advanced event processing top-level business object, and the business object. The name of the business object reflects the structure of the business function on the SAP server.

When it names business objects for the Advanced event processing interface, the Adapter Connection wizard adds a prefix of `Sap`. The wizard also converts the name of the IDoc and extension to mixed case, removing any separators such as spaces or underscores, capitalizes the first letter of each word, and it might add an element-specific suffix.

The following table describes the convention that is applied by the Adapter Connection wizard when it names Advanced event processing business objects. The *[Name of Extension type IDoc]* in the Naming convention column represents an optional entry; it is included in the name only if the selected IDoc is an Extension Type IDoc.

Table 65. Naming convention for advanced event processing business objects

Element	Naming convention
Name of the top-level wrapper object	`Sap` + *`Name of IDoc`* + *`[Name of Extension type IDoc]`* For example: `SapAepreq01`
Name of the IDoc business object for basic IDocs	`Sap` + *`Name of IDoc`* + *`BO`* For example, the business object for the IDoc MATMAS03 is: `SapMatmas03BO`
Name of the IDoc business object for extension type IDocs	`Sap` + *`Name of IDoc`* + *`Name of Extension type IDoc`* For example, the business object for the IDoc DELVRY03 and extension SD_DESADV_PDC is: `SapDelvry03SdDesadvPdc`

For an IDoc duplicate name, the Adapter Connection wizard adds a unique suffix to differentiate the business object. If an IDoc packet has two segments with the same name (for example, segOrder), the first business object is assigned the name `SapSegOrder` and the second business object is assigned a name such as `SapSegOrder619647890`, where 619647890 is the unique identifier that is appended to the name by the Adapter Connection wizard.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

Configuration properties for the WebSphere Adapter for SAP Software:

The WebSphere Adapter for SAP Software has several categories of configuration properties, which you set with the Adapter Connection wizard when you generate or create objects and services.

You can change the connection properties for the Adapter Connection wizard, and the inbound and outbound adapter properties. For more information, see the following topics:

- "SAP connection properties for the Adapter Connection wizard" on page 4044
- "Inbound adapter properties for SAP" on page 4054
- "Outbound adapter properties for SAP" on page 4076

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

SAP connection properties for the Adapter Connection wizard:

Connection properties establish a connection between the Adapter Connection wizard, a tool that is used to create business objects, and the SAP server. The properties that you configure in the Adapter Connection wizard specify such things as connection configuration, and tracing and logging options.

After you have established a connection between the Adapter Connection wizard and the SAP server, the wizard can access the metadata that it needs from the SAP server to create business objects.

Some of the properties that you set in the Adapter Connection wizard are used as the initial values for resource adapter, managed connection factory, and activation specification properties that you can specify at a later time in the wizard.

The connection properties and their purpose are described in the following table. A complete description of each property is provided in the sections that follow the table. If you set any of these connection properties using bidirectional script, you must set values that identify the format of the bidirectional script that is entered for that property.

Table 66. Connection properties for the Adapter for SAP Software

Property name	Description
"Bidi direction " on page 4045	The orientation component of the bidi format specification.
"Bidi ordering schema" on page 4046	The ordering schema of the bidi format specification.
"Bidi numeric shaping" on page 4046	The numeric shaping component of the bidi format specification.
"Bidi shaping" on page 4046	The shaping component of the bidi format specification.
"Bidi symmetric swapping" on page 4047	The symmetric swapping component of the bidi format specification.
"Client" on page 4047	The client number of the SAP system to which the adapter connects
"Codepage number (Codepage)" on page 4047	Indicates the numeric identifier of the code page
"Folder for RFC trace files (RfcTracePath)" on page 4048	Sets the fully qualified local path to the folder into which the RFC trace files are written
"Host name (ApplicationServerHost)" on page 4048	Specifies the IP address or the name of the application server host that the adapter logs on to
"Language code" on page 4049	Specifies the language in which the adapter logs on.
"Log file output location property" on page 4049	The location of the log file for enterprise metadata discovery.
"Logging level property" on page 4049	The type error for which logging occurs during enterprise metadata discovery.
"Password" on page 4050	The password of the user account of the adapter on the SAP application server
"RFC trace level (RcfTraceLevel)" on page 4051	Specifies the global trace level

Table 66. Connection properties for the Adapter for SAP Software (continued)

Property name	Description
"RFC trace on (RcfTraceOn)" on page 4051	Specifies whether to generate a text file detailing the RFC activity for each event listener
"SAP interface name" on page 4052	The SAP interface to be used.
"System number (SystemNumber)" on page 4052	The system number of the SAP application server
"User name (userName)" on page 4053	The user account for the adapter on the SAP server

The Adapter Connection wizard uses the bidirectional connection properties to apply the correct bidirectional transformation on the data that is passed to the SAP server.

The bidi properties specify the bidirectional format for data coming from an external application into the adapter in the form of any business object that is supported by this adapter.

Accept the default values for the bidirectional formatting properties on the Adapter Connection wizard that provides SAP server bidirectional format specification. When combined, these bidirectional properties define one single bidirectional format.

The default values for bidirectional formatting properties listed in this section are based on Windows bidirectional formatting. If the Enterprise Information System supports a bidirectional format other than the Windows standard bidirectional format, you must make appropriate changes to the bidi properties that are listed in the following sections.

Bidi direction

This property specifies the orientation component of the bidi format specification.

Table 67. Bidi direction details

Required	No
Possible values	Possible values include: • `LTR` The orientation is left-to-right. • `RTL` The orientation is right-to-left. • `contextualLTR` The orientation is left-to-right because of the context. A character that is not categorized as LTR, and hat is located between two strong characters with a different writing direction, inherits the main context's writing direction (in a LTR document the character becomes LTR). • `contextualRTL` The orientation is right-to-left because of the context. A character that is not categorized as RTL, and is located between two strong characters with a different writing direction, inherits the main context's writing direction (in a RTL document the character becomes RTL).
Default	`LTR`
Property type	String
Usage	Specifies the orientation component of the bidi format specification.
Globalized	Yes

Table 67. Bidi direction details (continued)

Bidi supported	No

Bidi ordering schema

This property specifies the ordering schema of the bidi format specification.

Table 68. Bidi ordering schema details

Required	No
Possible values	`Implicit` `Visual`
Default	`Implicit`
Property type	String
Usage	Specifies the ordering schema of the bidi format specification.
Globalized	Yes
Bidi supported	No

Bidi numeric shaping

This property specifies the numeric shaping component of the bidi format specification.

Table 69. Bidi numeric details

Required	No

Possible values	`Nominal`
	`National`
	`Contextual`

Default	`Nominal`
Property type	String
Usage	Specifies the numeric shaping component of the bidi format specification.
Globalized	Yes
Bidi supported	No

Bidi shaping

This property specifies the shaping component of the bidi format specification.

Table 70. Bidi shaping details

Required	No

Possible values	`Nominal`
	`Shaped`
	`Initial`
	`Middle`
	`Final`
	`Isolated`

Default	`Nominal`
Property type	String
Usage	Specifies the shaping component of the bidi format specification.

Table 70. Bidi shaping details (continued)

Globalized	Yes
Bidi supported	No

Bidi symmetric swapping

This property specifies the symmetric swapping component of the bidi format specification.

Table 71. Bidi symmetric swapping details

Required	No
Possible values	`True` `False`
Default	`True`
Property type	Boolean
Usage	This property specifies the symmetric swapping component of the bidi format specification.
Globalized	Yes
Bidi supported	No

Client

This property is the client number of the SAP system to which the adapter connects.

Table 72. Client details

Required	Yes
Possible values	You can enter a range of values from `000` to `999`.
Default	`100`
Property type	Integer
Usage	When an application attempts to log on to the SAP server, the application must have a Client number associated with it. The Client property value identifies the client (the adapter) that is attempting to log onto the SAP server.
Globalized	No
Bidi supported	No

Codepage number (Codepage)

The numeric identifier of the code page.

Table 73. Codepage number details

Required	No
Possible values	You can enter a range of values from `0000` to `9999`. For a full listing of languages and associated code page numbers that are supported by SAP, see SAP Note 7360.
Default	The default value for this property is conditionally determined by the value set for the **Language code** property.
Property type	Integer

Table 73. Codepage number details (continued)

Usage	The value that is assigned to the **Codepage number** defines the code page to be used and has a one-to-one relationship with the value set for the **Language code** property. The **Codepage number** establishes a connection to the appropriate language. Each language code value has a code page number value associated with it. For example, the language code for English is EN. If you select EN (English) as your language code, the code page number is automatically set to the numeric value that is associated with EN (English). The SAP code page number for EN (English) is `1100`.
Example	If **Language code** is set to `JA` (Japanese), **Codepage number** is set to `8000`.
Globalized	No
Bidi supported	No

Folder for RFC trace files (RfcTracePath)

This property sets the fully qualified local path to the folder in which to write RFC trace files.

Table 74. Folder for RFC trace files details

Required	No
Default	No default value
Property type	String
Usage	Identifies the fully qualified local path into which RFC trace files are written. If **RFC trace on** is set to False (not selected), you are not permitted to set a value in the **Folder for RFC trace files** property.
Example	`c:\temp\rfcTraceDir`
Globalized	Yes
Bidi supported	No

Host name (ApplicationServerHost)

Specifies the IP address or the name of the application server host that the adapter logs on to.

Table 75. Host name details

Required	Yes (when load balancing is not used).
Default	No default value
Property type	String
Usage	When the adapter is configured to run without load balancing, this property specifies the IP address or the name of the application server that the adapter logs on to.
Example	`sapServer`
Globalized	No
Bidi supported	No

Language code

SAP logon language code.

Table 76. Language code details

Required	Yes
Possible values	Each of the supported languages is preceded by a 2 character language code. The language itself is displayed in parentheses. The language codes in the list represent the SAP default set of 41 languages for non Unicode systems plus Arabic. For a full listing of supported language codes and languages, see the SAP documentation.
Default	The default language code will be your current locale. If your current locale is not listed as one of the supported language codes, then a default language code of `EN (English)` is used.
Property type	String
Usage	If you manually enter a language code, you do not need to enter the language in parentheses.
Example	If the system locale is English, the value for this property is `EN (English)`
Globalized	No
Bidi supported	No

Log file output location property

This property specifies the location of the log file for external metadata discovery.

Table 77. Log file output location details

Required	Yes
Default	The `.metadata` directory of the workspace
Property type	String
Usage	Use this directory to hold the log file that lists the errors that occur during the discovery process. The type of discovery errors for which logging occurs is controlled by the **Logging level** property
Example	`C:\IBM\WMBT700\workspace\.metadata\SAPMetadataDiscovery.log`
Globalized	Yes
Bidi supported	No

Logging level property

This property specifies the type error for which logging occurs during enterprise metadata discovery.

Table 78. Logging level details

Required	No

Table 78. Logging level details (continued)

Possible values	FATAL SEVERE WARNING AUDIT INFO CONFIG DETAIL
Default	SEVERE
Property type	String
Usage	Use this property to tailor tracing capabilities. When you specify an error type, you indicate that trace operations occur only for errors of the specified type.
Example	If you accept the default value of SEVERE, trace information is provided on errors that fall into the SEVERE category. Severe errors mean that an operation cannot continue, although the adapter can still function. Severe errors also include error conditions that indicate an impending fatal error, that is, reporting on situations that strongly suggest that resources are on the verge of being depleted. Other error descriptions are listed here: • Fatal Adapter cannot continue. Adapter cannot function • Warning Potential error or impending error, including conditions that indicate a progressive failure (for example, the potential leaking of resources). • Audit Significant event affecting adapter state or resources • Info General information outlining overall operation progress. • Config Configuration change or status. • Detail General information detailing operation progress
Globalized	Yes
Bidi supported	No

Password

This property is the password of the user account of the adapter on the SAP application server.

Table 79. Password details

Required	Yes
Default	No default value
Property type	String

Table 79. Password details (continued)

Usage	The restrictions on the password depend on the version of SAP Web Application Server. • For SAP Web Application Server version 6.40 or earlier, the password: – Must be uppercase – Must be 8 characters in length • For versions of SAP Web Application Server later than 6.40, the password: – Is not case-sensitive – Can be up to 40 characters in length
Globalized	No
Bidi supported	Yes

RFC trace level (RcfTraceLevel)

This property specifies the global trace level.

Table 80. RFC trace level details

Required	No
Possible values	`1` - This is the default RFC trace level. When specified, SAP JCo Java API logging occurs. `3` - When specified, SAP JCo JNI API logging occurs. `5` - When specified, error diagnostic logging occurs.
Default	`1`
Property type	Integer
Usage	If **RFC trace on** is set to `False` (not selected), you cannot set a value in the **RFC trace level** property.
Globalized	No
Bidi supported	No

RFC trace on (RcfTraceOn)

This property specifies whether to generate a text file detailing the RFC activity for each event listener.

Table 81. RFC trace on details

Required	No
Possible values	`True` `False`
Default	`False`
Property type	Boolean
Usage	A value of `True` activates tracing, which generates a text file. This file is created in the directory in which the adapter process was started. The file has a prefix of `rfx` and a file type of trc (for example, rfc03912_02220.trc). Use these text files in a development environment only, because the files can grow rapidly. If **RFC trace on** is set to `False` (not selected), you cannot set values in the **Folder for RFC trace files** or **RFC trace level** properties.

Table 81. RFC trace on details (continued)

Example	Examples of the information in the file are RfcCall FUNCTION BAPI_CUSTOMER_GETLIST, followed by the information for the parameters in the interface, or RFC Info rfctable, followed by the data from one of the interface tables. The trace file is created in the directory where the adapter process has been started. The trace file has a .trc file extension and the file name will start with the letters `rfc` followed by a unique identifier. For example, `rfc03912_02220.trc`.
Globalized	No
Bidi supported	No

SAP interface name

This property indicates whether you are creating business objects for the ALE, BAPI, Advanced event processing, or Query interface for SAP Software.

Table 82. SAP interface name details

Required	Yes

Possible values	For outbound:
	`Advanced event processing (AEP)`
	`ALE`
	`ALE pass-through IDoc`
	`BAPI`
	`BAPI work unit`
	`BAPI result set`
	`Query interface for SAP Software (QSS)`
	
	For inbound:
	`Advanced event processing (AEP)`
	`ALE`
	`ALE pass-through IDoc`
	`BAPI`

Default	For outbound: `BAPI` For inbound:`ALE`
Property type	String
Usage	Specifies the interface used by the adapter. The adapter interacts with the interface to support outbound or inbound processing by enabling the exchange of data in the form of business objects.
Globalized	No
Bidi supported	No

System number (SystemNumber)

This property is the system number of the SAP application server.

Table 83. System number details

Required	Yes
Possible values	You can enter a range of values from `00` to `99`.

Table 83. System number details (continued)

Default	`00`
Property type	Integer
Usage	The system number further identifies the Gateway service.
Globalized	No
Bidi supported	No

User name (userName)

This property is the user account for the adapter on the SAP server.

Table 84. User name details

Required	Yes
Default	No default value
Property type	String
Usage	Maximum length of 12 characters. The user name is not case sensitive. It is recommended that you set up a CPIC user account in the SAP application and that you give this account the necessary privileges to manipulate the data required by the business objects supported by the adapter. For example, if the adapter must perform certain SAP business transactions, the adapter's account in the SAP application must have the permissions set to allow it to perform these transactions.
Example	`SapUser`
Globalized	Yes
Bidi supported	Yes

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

Inbound adapter properties for SAP:

Inbound adapter properties hold the inbound event processing configuration information for a message endpoint. These properties also control the general operation of the adapter. Use the Adapter Connection wizard to set these properties.

These properties are used during endpoint activation to notify the adapter of eligible event listeners. During inbound processing, the adapter uses these event listeners to receive events before it forwards them to the endpoint.

The following table lists the properties for inbound processing that you set by using the Adapter Connection wizard. A more detailed description of each property is provided in the sections that follow the table.

Table 85. Activation specification properties for ALE inbound processing

Property name	Description
Adapter ID to use for logging and tracing (AdapterID)	A specific deployment, or instance, of the adapter
ALE packet audit (alePacketUpdate)	Specifies if the adapter should send ALEAUD per IDoc or per packet (TID)
ALE update status (aleUpdateStatus)	Specifies whether an audit trail is required for all message types
Assured once-only delivery (AssuredOnceDelivery)	Specifies whether to provide assured once-only delivery for inbound events
Auto create event table (EP_CreateTable)	Specifies whether the adapter should create the event recovery table automatically if it does not already exist

Table 85. Activation specification properties for ALE inbound processing (continued)

Property name	Description
Client (Client)	The client number of the SAP system to which the adapter connects
Codepage number (Codepage)	Indicates the numeric identifier of the code page
Database schema name (EP_SchemaName)	The schema that is used for automatically creating the event recovery table
Delivery type (DeliveryType)	Determines the order in which events are delivered by the adapter to the export component.
Enable secure network connection (SncMode)	Indicates whether secure network connection mode is used
Event recovery data source (JNDI) name (EP_SchemaName)	The JNDI name of the data source that is configured for event recovery
Event recovery table name (EP_TableName)	The name of the event recovery table
Event types to process (EventTypeFilter)	A delimited list of event types that the WebSphere Adapter for SAP Software should deliver.
Failure code (aleFailureCode)	The status code for dispatch failure
Failure text (aleFailureText)	The descriptive text for dispatch failure
Folder for RFC trace files (RfcTracePath)	Sets the fully qualified local path to the folder into which the RFC trace files are written
Gateway host (GatewayHost)	The host name of the SAP gateway
Gateway service (GatewayService)	The identifier of the gateway on the gateway host that carries out the RFC services
Host name (ApplicationServerHost)	Specifies the IP address or the name of the application server host that the adapter logs on to
Ignore IDoc packet errors (IgnoreIDocPacketErrors)	Specifies what the adapter does when it encounters an error while processing the IDoc packet
Language code (Language code)	Specifies the Language code in which the adapter logs on to SAP
Logon group name (Group)	An identifier of the name of the group of application server instances that have been defined in transaction SMLG and linked together for logon load balancing
Maximum number of events collected during each poll (PollQuantity)	The number of events that the adapter delivers to the export during each poll period.
Message server host (MessageServerHost)	Specifies the name of the host on which the message server is running
Number of listeners (NumberOfListeners)	Specifies the number of event listeners that are to be started
Partner character set (PartnerCharset)	Specifies PartnerCharset encoding
Password (Password)	The password of the user account of the adapter on the SAP application server
Password used to connect to event data source (EP_Password)	The user password for connecting to the database
RFC program ID (RfcProgramID)	The remote function call identifier under which the adapter registers in the SAP gateway
RFC trace level (RcfTraceLevel)	Specifies the global trace level
RFC trace on (RcfTraceOn)	Specifies whether to generate a text file detailing the RFC activity for each event listener

Table 85. Activation specification properties for ALE inbound processing (continued)

Property name	Description
SAP system ID (SAPSystemID)	Specifies the system ID of the SAP system for which logon load balancing is allowed
Secure network connection library path (SncLib)	Specifies the path to the library that provides the secure network connection service
Secure network connection name (SncMyname)	Specifies the name of the secure network connection
Secure network connection partner (SncPartnername)	Specifies the name of the secure network connection partner
Secure network connection security level (SncQop)	Specifies the level of security for the secure network connection
Selective update (aleSelectiveUpdate)	The IDoc Type and MessageType combinations that are to be updated when the adapter is configured to update a standard SAP status code
Status message code (aleStatusMsgCode)	If required, the message code to use when the adapter posts the ALEAUD Message IDoc (ALEAUD01)
Stop polling on error (StopPollingOnError)	Specifies whether the adapter stops polling for events when it encounters an error during polling.
Success code (aleSuccessCode)	The success status code for Application Document Posted
Success text (aleSuccessText)	The descriptive text for successful Application Document Posted
System number (SystemNumber)	The system number of the SAP application server
Interval between polling periods (PollPeriod)	The length of time that the adapter waits between polling periods.
Retry interval if connection fails (RetryInterval)	The length of time that the adapter waits between attempts to establish a new connection after an error during inbound operations.
User name (userName)	The user account for the adapter on the SAP server
User name used to connect to event data source (EP_UserName)	The user name for connecting to the database
X509 certificate (X509cert)	Specifies the X509 certificate to be used as the logon ticket

Adapter ID to use for logging and tracing (AdapterID)

This property identifies a specific deployment, or instance, of the adapter.

Table 86. Adapter ID to use for logging and tracing details

Required	Yes
Default	`001`
Property type	String

Table 86. Adapter ID to use for logging and tracing details (continued)

Usage	This property identifies the adapter instance in the log and trace files, and also helps identify the adapter instance while monitoring adapters. The adapter ID is used with an adapter-specific identifier, SAPRA, to form the component name used by the Log and Trace Analyzer tool. For example, if the adapter ID property is set to 001, the component ID is SAPRA001. If you run multiple instances of the same adapter, ensure that the first eight characters of the adapter ID property are unique for each instance so that you can correlate the log and trace information to a particular adapter instance. By making the first seven characters of an adapter ID property unique, the component ID for multiple instances of that adapter is also unique, allowing you to correlate the log and trace information to a particular instance of an adapter. For example, when you set the adapter ID property of two instances of WebSphere Adapter for SAP Software to 001 and 002. The component IDs for those instances, SAPRA001 and SAPRA002, are short enough to remain unique, enabling you to distinguish them as separate adapter instances. However, instances with longer adapter ID properties cannot be distinguished from each other. If you set the adapter ID properties of two instances to Instance01 and Instance02, you will not be able to examine the log and trace information for each adapter instance because the component ID for both instances is truncated to SAPRAInstance. For inbound processing, this property is retrieved from the resource adapter properties. For outbound processing, it is retrieved from the managed connection factory properties.
Globalized	Yes
Bidi supported	No

ALE packet audit

ALE update per packet indicates whether the adapter should send ALEAUD audit IDocs per packet or per IDoc.

Table 87. ALE packet audit details

Required	No
Default	`False` (send ALEAUD per IDoc).
Property type	Boolean
Usage	You can enable and disable this property only if the ALE update status property is set to `True`. When you set this property to `true`, the adapter sends one ALEAUD per IDoc packet which contains confirmations for all IDocs in the packet. When you set this property to `false`, the adapter sends one ALEAUD for each IDoc received at the adapter.
Globalized	No
Bidi supported	No

ALE update status (aleUpdateStatus)

This property specifies whether an audit trail is required for all message types.

Table 88. ALE update status details

Required	Yes
Possible values	`True` `False`

Table 88. ALE update status details (continued)

Default	`False`
Property type	Boolean
Usage	Set this property to `True` if you want the adapter to update a standard SAP status code after the ALE module has retrieved an IDoc object for event processing. If you set this value to `True`, you must also set following properties: • ALE failure code • ALE Success code • ALE failure text • ALE success text.
Globalized	No
Bidi supported	No

Assured once-only delivery (AssuredOnceDelivery)

This property specifies whether to provide assured once-only delivery for inbound events.

Table 89. Assured once-only delivery details

Required	Yes
Default	`True`
Property type	Boolean
Usage	When this property is set to `True`, the adapter provides assured once-only event delivery, so that each event is delivered only once. A value of `False` does not provide assured once-only event delivery, but provides better performance. When this property is set to `True`, the adapter attempts to store transaction (XID) information in the event store. If it is set to `False`, the adapter does not attempt to store the information. This property is used only if the export component is transactional. If the export component is not transactional, no transaction can be used, regardless of the value of this property.
Globalized	No
Bidi supported	No

Note: The **Assured once-only delivery** property applies only to asynchronous transactional RFC processing.

Auto create event table (EP_CreateTable)

This property determines if the event table is created automatically.

Table 90. Auto create event table details

Required	Yes if **Assured once-only event delivery** is set to `True`; otherwise, no.
Possible values	`True` `False`
Default	`True`
Property type	Boolean

Table 90. Auto create event table details (continued)

Usage	This property indicates whether the adapter should create the event recovery table automatically if it does not already exist. If you specify a value of `True` to automatically create the table, you must specify information about the event table (such as the event recovery table name). The value that is provided in **Event recovery table name** property is used to create the table.
Globalized	No
Bidi supported	No

Client

This property is the client number of the SAP system to which the adapter connects.

Table 91. Client details

Required	Yes
Possible values	You can enter a range of values from `000` to `999`.
Default	`100`
Property type	Integer
Usage	When an application attempts to log on to the SAP server, the application must have a Client number associated with it. The Client property value identifies the client (the adapter) that is attempting to log onto the SAP server.
Globalized	No
Bidi supported	No

Codepage number (Codepage)

The numeric identifier of the code page.

Table 92. Codepage number details

Required	No
Possible values	You can enter a range of values from `0000` to `9999`. For a full listing of languages and associated code page numbers that are supported by SAP, see SAP Note 7360.
Default	The default value for this property is conditionally determined by the value set for the **Language code** property.
Property type	Integer
Usage	The value that is assigned to the **Codepage number** defines the code page to be used and has a one-to-one relationship with the value set for the **Language code** property. The **Codepage number** establishes a connection to the appropriate language. Each language code value has a code page number value associated with it. For example, the language code for English is EN. If you select EN (English) as your language code, the code page number is automatically set to the numeric value that is associated with EN (English). The SAP code page number for EN (English) is `1100`.
Example	If **Language code** is set to `JA` (Japanese), **Codepage number** is set to `8000`.
Globalized	No
Bidi supported	No

Database schema name (EP_SchemaName)

This property is the schema that is used to create the event recovery table automatically.

Table 93. Database schema name details

Required	No
Default	No default value.
Property type	String
Usage	Specifies the schema name for the database used by the adapter's event persistence feature.
Example	`ALE_SCHEMA`
Globalized	Yes
Bidi supported	No

Delivery type (DeliveryType)

This property specifies the order in which events are delivered by the adapter to the export component.

Table 94. Delivery type details

Required	No
Possible values	`ORDERED` `UNORDERED`
Default	`ORDERED`
Property type	String
Usage	The following values are supported: • `ORDERED`: The adapter delivers events to the export component one at a time. • `UNORDERED`: The adapter delivers all events to the export component at once.
Globalized	No
Bidi supported	No

Enable Secure Network Connection

This property indicates whether secure network connection mode is enabled.

Table 95. Enable Secure Network Connection details

Required	No
Possible values	`0` (off) `1` (on)
Default	`0`
Property type	String

Table 95. Enable Secure Network Connection details (continued)

Usage	Set the value to 1 (on) if you want to use secure network connection. If you set this value to 1, you must also set following properties: • Secure Network Connection library path • Secure Network Connection name • Secure Network Connection partner • Secure Network Connection security level
Globalized	No
Bidi supported	No

Event recovery data source (JNDI) name (EP_DataSource_JNDIName)

This property is the JNDI name of the data source that is configured for event recovery.

Table 96. Event recovery data source (JNDI) name details

Required	Yes
Default	No default value.
Property type	String
Usage	Used in event recovery processing. The data source must be created in WebSphere Message Broker. The adapter uses the data source for persisting the event state.
Example	`jdbc/DB2`
Globalized	No
Bidi supported	No

Event recovery table name (EP_TableName)

This property is the name of the event recovery table.

Table 97. Event recovery table name details

Required	Yes
Default	No default value.
Property type	String
Usage	Used in event recovery processing. Consult database documentation for information on naming conventions. Configure a separate event recovery table for each endpoint. The same data source can be used to hold all of the event recovery tables.
Example	`EVENT_TABLE`
Globalized	No
Bidi supported	No

Event type filter

This property provides a delimited list of business object types for which the adapter should deliver events.

Table 98. Event type filter details

Required	No
Possible values	A comma-delimited (,) list of business object types.
Default	`null`
Property type	String
Usage	If this property is set, the adapter uses the delimited list as a filter, delivering events for only those business object types that are contained in the list. If the list is empty (null), the adapter does not apply filtering, and delivers events for all business object types.
Globalized	No
Bidi supported	No

Failure code (aleFailureCode)

This property determines how the adapter updates the SAP failure status code after the ALE module has retrieved an IDoc object for event processing.

Table 99. ALE failure code details

Required	Yes if AleUpdateStatus is set to `True`; otherwise, no.

Possible values	`68`
	`58`
	`51`

Default	51
Property type	Integer
Usage	Set a value for this property only if you set the value for the ALE update status property to `True`. Specify a value of `68` for this property to cause the adapter to update the SAP failure status code after the ALE module has retrieved an IDoc object for event processing. SAP converts this value to 40 (Application Document not created in receiving system). When you set the AleUpdateStatus property to `True`, the adapter updates a standard SAP status code after the adapter retrieves an IDoc object for event processing. An IDoc that is not sent successfully to the endpoint is considered to be a failure. Use the ALE failure code property to specify the code that is used to signify this failure.
Globalized	No
Bidi supported	No

Failure text (aleFailureText)

The property specifies the text that appears when an IDoc is not sent successfully to the endpoint.

Table 100. ALE failure text details

Required	Yes if AleUpdateStatus is set to `True`; otherwise, no.
Default	No default value.
Property type	String

Table 100. ALE failure text details (continued)

Usage	Use this property only if you set the ALE update status property to `True`. The length of the text string cannot exceed 70 characters. When you set the AleUpdateStatus property to `True`, the adapter updates a standard SAP status code after the adapter retrieves an IDoc object for event processing. An IDoc that is not sent successfully to the endpoint is considered to be a failure. Use the ALE failure text property to specify the descriptive text that is used to signify this failure.
Example	`ALE Dispatch Failed`
Globalized	Yes
Bidi supported	No

Folder for RFC trace files (RfcTracePath)

This property sets the fully qualified local path to the folder in which to write RFC trace files.

Table 101. Folder for RFC trace files details

Required	No
Default	No default value
Property type	String
Usage	Identifies the fully qualified local path into which RFC trace files are written. If **RFC trace on** is set to False (not selected), you are not permitted to set a value in the **Folder for RFC trace files** property.
Example	`c:\temp\rfcTraceDir`
Globalized	Yes
Bidi supported	No

Gateway host (GatewayHost)

This property is the Gateway host name. Enter either the IP address or the name of the Gateway host. Consult your SAP administrator for information on the Gateway host name.

Table 102. Gateway host details

Required	Yes
Default	No default value
Property type	String
Usage	This property is the host name of the SAP gateway. The gateway enables communication between work processes on the SAP system and external programs. The host identified is used as the gateway for the resource adapter. Maximum length of 20 characters. If the computer name is longer than 20 characters, define a symbolic name in the THOSTS table.
Globalized	No
Bidi supported	No

Gateway service (GatewayService)

This property is the identifier of the gateway on the gateway host that carries out the RFC services.

Table 103. Gateway service details

Required	Yes
Default	`sapgw00`
Property type	String
Usage	These services enable communication between work processes on the SAP server and external programs. The service typically has the format of `sapgw00`, where 00 is the SAP system number. Maximum of 20 characters.
Globalized	No
Bidi supported	No

Host name (ApplicationServerHost)

Specifies the IP address or the name of the application server host that the adapter logs on to.

Table 104. Host name details

Required	Yes (when load balancing is not used).
Default	No default value
Property type	String
Usage	When the adapter is configured to run without load balancing, this property specifies the IP address or the name of the application server that the adapter logs on to.
Example	`sapServer`
Globalized	No
Bidi supported	No

Ignore IDoc packet errors (IgnoreIDocPacketErrors)

This property determines whether IDoc packet errors are to be ignored.

Table 105. Ignore IDOC packet errors details

Required	No
Possible values	`True` `False`
Default	`False`
Property type	Boolean

Table 105. Ignore IDOC packet errors details (continued)

Usage	If the adapter encounters an error while processing the IDoc packet, it can behave in two different ways. • When this property is set to `False`, the adapter stops processing further IDocs in that packet and reports an error to the SAP system. • When this property is set to `True`, the adapter logs an error and continues to process the rest of the IDocs in that packet. The status of the transaction is marked as `INPROGRESS`. The adapter log displays the IDoc numbers that failed and you need to resubmit those individual IDocs separately. You need to manually maintain these records in the event recovery table. This property is not used for single IDocs and for non-split IDoc packets.
Globalized	No
Bidi supported	No

Language code

This property specifies the Language code in which the adapter logs on.

Table 106. Language code details

Required	Yes
Possible values	For a full listing of languages and associated code page numbers that are supported by SAP, see SAP Note 7360.
Default	The default value for the **Language code** property is based on the system locale.
Property type	String
Usage	Each of the supported languages is preceded by a two-character language code. The language itself is displayed in parentheses. If you enter a language code manually, you do not need to enter the language in parentheses. The language codes that are listed represent the SAP default set of 41 languages for non-Unicode systems plus Arabic. The value that you select determines the value of the **Codepage number** property.
Example	If the system locale is English, the value for this property is `EN (English)`.
Globalized	No
Bidi supported	No

Logon group name

This property is an identifier for the name of the group of application server instances that have been defined in transaction SMLG and linked together for logon load balancing.

Table 107. Logon group details

Required	Yes (if load balancing is used)
Possible values	Consult SAP documentation for information on creating Logon groups and on calling transaction SMLG.
Default	No default value
Property type	String

Table 107. Logon group details (continued)

Usage	When the adapter is configured for load balancing, this property represents the name of the group of application server instances that have been defined in transaction SMLG and linked together for logon load balancing. Logon load balancing allows for the dynamic distribution of logon connections to application server instances. Maximum of 20 characters. On most SAP systems, the SPACE logon group is reserved by SAP.
Globalized	No
Bidi supported	No

Maximum number of events collected during each poll

This property specifies the number of events that the adapter delivers to the export component during each poll period.

Table 108. Maximum number of events collected during each poll details

Required	Yes
Default	`10`
Property type	Integer
Usage	This value must be greater than 0
Globalized	No
Bidi supported	No

Maximum number of retries in case of system connection failure

This property specifies the number of times the adapter tries to restart the event listeners.

Table 109. Maximum number of retries in case of system failure details

Required	Yes
Default	`0`
Property type	Integer
Usage	When the adapter encounters an error related to the inbound connection (if the SAP application is down for example), this property specifies the number of times the adapter tries to restart the event listeners. A value of 0 indicates an infinite number of retries. **Note:** Configure the **Time between retries in case of system connection failure (milliseconds)** appropriately when retrying infinitely. For each retry attempt, the adapter waits based on the time interval specified in the **Time between retries in case of system connection failure (milliseconds)**. **Note:** If all the retry attempts fail, the adapter logs relevant messages and CEI events and stops attempting to recover the event listener. If you reach this point, you might have to restart the application manually.
Globalized	No
Bidi supported	No

Message server host (MessageServerHost)

This property specifies the name of the host on which the message server is running.

Table 110. Message server host details

Required	Yes (if load balancing is used)
Default	No default value
Property type	String
Usage	This property specifies the name of the host that will inform all the servers (instances) belonging to this SAP system of the existence of the other servers to be used for load balancing. The message server host contains the information about load balancing for RFC clients so that an RFC client can be directed to an appropriate application server.
Example	`SAPERP05`
Globalized	No
Bidi supported	No

Number of listeners

This property specifies the number of listeners that are started by an event.

Table 111. Number of listeners details

Required	No
Default	1
Property type	Integer
Usage	For event sequencing, this property should be set to 1. To improve adapter performance, you can increase the number of listeners. The adapter will not start if the number of listeners is zero (0).
Globalized	No
Bidi supported	No

Partner character set (PartnerCharset)

This property specifies the partner character set encoding.

Table 112. Partner character set details

Required	No
Default	UTF-8
Property type	String
Usage	When an encoding is specified, it is used; otherwise, the default encoding is used.
Globalized	No
Bidi supported	No

Password

This property is the password of the user account of the adapter on the SAP application server.

Table 113. Password details

Required	Yes
Default	No default value
Property type	String
Usage	The restrictions on the password depend on the version of SAP Web Application Server. • For SAP Web Application Server version 6.40 or earlier, the password: – Must be uppercase – Must be 8 characters in length • For versions of SAP Web Application Server later than 6.40, the password: – Is not case-sensitive – Can be up to 40 characters in length
Globalized	No
Bidi supported	Yes

Password used to connect to event data source (EP_Password)

This property is the user password for connecting to the database.

Table 114. Password to connect to event data source details

Required	Yes
Default	No default value.
Property type	String
Usage	This property specifies the password used by event-persistence processing to obtain the database connection from the data source.
Globalized	Yes
Bidi supported	No

RFC program ID

This property is the program identifier under which the adapter registers in the SAP gateway.

Table 115. RFC program ID details

Required	Yes
Possible values	Use the SAP transaction SM59 (Display and Maintain RFC Destinations) to see a list of available RFC program IDs.
Default	No default value.
Property type	String
Usage	The adapter registers with the gateway so that listener threads can process events from RFC-enabled functions. This value must match the program ID registered in the SAP application. The maximum length is 64 characters.
Globalized	No

Table 115. RFC program ID details (continued)

Bidi supported	No

RFC trace level (RcfTraceLevel)

This property specifies the global trace level.

Table 116. RFC trace level details

Required	No
Possible values	`1` - This is the default RFC trace level. When specified, SAP JCo Java API logging occurs. `3` - When specified, SAP JCo JNI API logging occurs. `5` - When specified, error diagnostic logging occurs.
Default	`1`
Property type	Integer
Usage	If **RFC trace on** is set to `False` (not selected), you cannot set a value in the **RFC trace level** property.
Globalized	No
Bidi supported	No

RFC trace on (RcfTraceOn)

This property specifies whether to generate a text file detailing the RFC activity for each event listener.

Table 117. RFC trace on details

Required	No
Possible values	`True` `False`
Default	`False`
Property type	Boolean
Usage	A value of `True` activates tracing, which generates a text file. This file is created in the directory in which the adapter process was started. The file has a prefix of `rfx` and a file type of trc (for example, rfc03912_02220.trc). Use these text files in a development environment only, because the files can grow rapidly. If **RFC trace on** is set to `False` (not selected), you cannot set values in the **Folder for RFC trace files** or **RFC trace level** properties.
Example	Examples of the information in the file are RfcCall FUNCTION BAPI_CUSTOMER_GETLIST, followed by the information for the parameters in the interface, or RFC Info rfctable, followed by the data from one of the interface tables. The trace file is created in the directory where the adapter process has been started. The trace file has a .trc file extension and the file name will start with the letters `rfc` followed by a unique identifier. For example, `rfc03912_02220.trc`.
Globalized	No
Bidi supported	No

SAP system ID (SAPSystemID)

This property specifies the system ID of the SAP system for which logon load balancing is allowed.

Table 118. SAP system ID details

Required	Yes (when load balancing is used)
Default	No default value
Property type	String
Usage	Value must be three characters
Example	`DYL`
Globalized	No
Bidi supported	No

Secure Network Connection library path (SncLib)

This property specifies the path to the library that provides the secure network connection service.

Table 119. Secure Network Connection library path details

Required	Yes, if SncMode is set to 1; no otherwise.
Default	No default value
Property type	String
Usage	If the SncMode property is set to 1 (indicating that you are using a secure network connection), specify the path to the library that provides the service.
Example	`/WINDOWS/system32/gssapi32.dll`
Globalized	No
Bidi supported	No

Secure Network Connection name (SncMyname)

This property specifies the name of the secure network connection.

Table 120. Secure Network Connection name details

Required	Yes, if SncMode is set to 1; no otherwise.
Default	No default value
Property type	String
Usage	If the SncMode property is set to 1 (indicating that you are using a secure network connection), specify a name for the connection.
Example	`DOMAINNAME/USERNAME`
Globalized	No
Bidi supported	No

Secure Network Connection partner (SncPartnername)

This property specifies the name of the secure network connection partner.

Table 121. Secure Network Connection partner details

Required	Yes, if SncMode is set to 1; no otherwise.
Default	No default value
Property type	String
Usage	If the SncMode property is set to 1 (indicating that you are using a secure network connection), specify a name for the connection partner.
Example	`CN=sap00.saperpdev, OU=Adapter, O=IBM, C=US`
Globalized	No
Bidi supported	No

Secure Network Connection security level (SncQop)

This property specifies the level of security for the secure network connection.

Table 122. Secure Network Connection security level details

Required	Yes, if SncMode is set to 1; no otherwise.
Possible values	`1` (Authentication only) `2` (Integrity protection) `3` (Privacy protection) `8` (Use the value from `snc/data_protection/use` on the application server) `9` (Use the value from `snc/data_protection/max` on the application server)
Default	`3` (Privacy protection)
Property type	String
Usage	If the SncMode property is set to 1 (indicating that you are using a secure network connection), specify a value to indicate the level of security for the connection.
Globalized	No
Bidi supported	No

Selective update (aleSelectiveUpdate)

This property specifies which IDoc Type and MessageType combinations are to be updated.

Table 123. ALE selective update details

Required	No
Default	No default value
Property type	String

Table 123. ALE selective update details (continued)

Usage	You can set values for this property only if the ALE update status property is set to `True`. When you set the AleUpdateStatus property to `True`, the adapter updates a standard SAP status code after the adapter retrieves an IDoc object for event processing. Use the ALE selective update property to specify which IDoc Type and MessageType combinations are to be updated. The syntax for this property is: `IDocType: MessageType [;IDocType: MessageType [;...]]` where a slash (/) delimiter separates each IDoc Type and MessageType, and a semicolon (;) delimiter separates entries in a set.
Example	The following example illustrates two sets. In the example, MATMAS03 and DEBMAS03 are the IDocs, and MATMAS and DEBMAS are the message types: `MATMAS03/MATMAS;DEBMAS03/DEBMAS`
Globalized	No
Bidi supported	No

Status message code (aleStatusMsgCode)

This property specifies the message code to use when the adapter posts the ALEAUD01 IDoc with message type ALEAUD.

Table 124. ALE status message code details

Required	No
Possible values	For a list of available codes, refer to the SAP table TEDS1.
Default	No default value.
Property type	String
Usage	You can set a value for this property only if the ALE update status property has been set to `True`. You must configure this message code in the receiving partner profile on SAP.
Globalized	No
Bidi supported	No

Stop the adapter when an error is encountered while polling (StopPollingOnError)

This property specifies whether the adapter stops polling for events when it encounters an error during polling.

Table 125. Stop the adapter when an error is encountered while polling details

Required	No
Possible values	`True` `False`
Default	`False`
Property type	Boolean
Usage	If this property is set to `True`, the adapter stops polling when it encounters an error. If this property is set to `False`, the adapter logs an exception when it encounters an error during polling and continues polling.
Globalized	No

Table 125. Stop the adapter when an error is encountered while polling details (continued)

Bidi supported	No

Success code (aleSuccessCode)

This property specifies the ALE success code for the successful posting of an IDoc.

Table 126. ALE success code details

Required	Yes if AleUpdateStatus is set to `True`; otherwise, no.
Possible values	`52` `53`
Default	No default value
Property type	Integer
Usage	Use this property only if you set the ALE update status property to `True`. When you set the AleUpdateStatus property to `True`, the adapter updates a standard SAP status code after the adapter retrieves an IDoc object for event processing. Use the ALE success code property to specify the code for IDoc posted as 53. After the IDoc is sent to the endpoint, the IDoc status remains as 03 (IDoc posted to port) in SAP. After posting the IDoc, the adapter posts the audit IDoc with the current IDoc number and status as 53. SAP converts the current IDoc status to 41 (Application Document Created in Receiving System).
Globalized	No
Bidi supported	No

Success text (aleSuccessText)

This property specifies the text that appears when an application document is posted successfully.

Table 127. ALE success text details

Required	Yes if AleUpdateStatus is set to `True`; otherwise, no.
Default	No default value.
Property type	String
Usage	Use this property only if you set the ALE status update property to `True`. When you set the AleUpdateStatus property to `True`, the adapter updates a standard SAP status code after the adapter retrieves an IDoc object for event processing. Use the ALE success text property to specify the descriptive text that is used to signify Application Document Posted.
Example	`ALE Dispatch OK`
Globalized	Yes
Bidi supported	No

System number (SystemNumber)

This property is the system number of the SAP application server.

Table 128. System number details

Required	Yes

Table 128. System number details (continued)

Possible values	You can enter a range of values from `00` to `99`.
Default	`00`
Property type	Integer
Usage	The system number further identifies the Gateway service.
Globalized	No
Bidi supported	No

Time between polling for events (milliseconds)

This property specifies the length of time that the adapter waits between polling periods.

Table 129. Time between polling for events (milliseconds)

Required	Yes
Possible values	Integers greater than or equal to `0`.
Default	`2000`
Unit of measure	Milliseconds
Property type	Integer
Usage	The time interval between polling events is established at a fixed rate, which means that if running the poll cycle is delayed for any reason (for example, if a prior poll cycle takes longer than expected to complete), the next poll cycle occurs immediately to make up for the time that is lost because of the delay.
Globalized	No
Bidi supported	No

Time between retries in case of system connection failure (milliseconds) (ConnectionRetryInterval)

This property specifies the time interval between attempts to restart the event listeners.

Table 130. Time between retries in case of system connection failure details

Required	No
Possible values	Positive integers
Default	`60000`
Unit of measure	Milliseconds
Property type	Integer
Usage	When the adapter encounters an error related to the inbound connection, this property specifies the time interval the adapter waits in between attempts to restart the event listeners.
Globalized	No
Bidi supported	No

User name (userName)

This property is the user account for the adapter on the SAP server.

Table 131. User name details

Required	Yes
Default	No default value
Property type	String
Usage	Maximum length of 12 characters. The user name is not case sensitive. It is recommended that you set up a CPIC user account in the SAP application and that you give this account the necessary privileges to manipulate the data required by the business objects supported by the adapter. For example, if the adapter must perform certain SAP business transactions, the adapter's account in the SAP application must have the permissions set to allow it to perform these transactions.
Example	`SapUser`
Globalized	Yes
Bidi supported	Yes

User name used to connect to event data source (EP_UserName)

This property is the user name for connecting to the database.

Table 132. User name to connect to event data source details

Required	Yes
Default	No default value.
Property type	String
Usage	User name used by event persistence for getting the database connection from the data source. Consult database documentation for information on naming conventions.
Globalized	Yes
Bidi supported	No

X509 certificate (X509cert)

This property specifies the X509 certificate to be used as the logon ticket.

Table 133. X509 certificate details

Required	No
Default	No default value
Property type	String
Usage	If the SncMode property is set to 1 (indicating that you are using a secure network connection), you can provide a value for the X509 certificate.
Globalized	No
Bidi supported	No

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

Outbound adapter properties for SAP:

Outbound adapter properties define how the adapter creates an outbound connection instance with the SAP server, and how operations are run on the server. Use the Adapter Connection wizard to set these properties.

The following table describes the outbound adapter properties. A more detailed description of each property is provided in the sections that follow the table.

Table 134. Interaction specification properties for Adapter for SAP Software

Property name	Description
ABAP debug (ABAPDebug)	The ABAP debugger property
Client (Client)	The client number of the SAP system to which the adapter connects
Codepage number (Codepage)	Indicates the numeric identifier of the code page
Custom retrieve function name (customFunctionName)	The name of a custom function to be used by the Query interface to SAP Software to retrieve data from an SAP table.
Enable secure network connection (SncMode)	Indicates whether secure network connection mode is used
Folder for RFC trace files (RfcTracePath)	Sets the fully qualified local path to the folder into which the RFC trace files are written
Function name (functionName)	The function name for a specific SAP interface.
Gateway host (GatewayHost)	The host name of the SAP gateway
Gateway service (GatewayService)	The identifier of the gateway on the gateway host that carries out the RFC services
Host name (ApplicationServerHost)	Specifies the IP address or the name of the application server host that the adapter logs on to
Ignore errors in BAPI return (IgnoreBAPIReturn)	Specifies if errors in BAPI return are ignored.
Language code (Language code)	Specifies the Language code in which the adapter logs on to SAP
Maximum number of hits for the discovery (ResultSetLimit)	The maximum number of result sets to return during a `RetrieveAll` operation.
Message server host (MessageServerHost)	Specifies the name of the host on which the message server is running
Partner character set (PartnerCharset)	Specifies PartnerCharset encoding
Password (Password)	The password of the user account of the adapter on the SAP application server
RFC trace level (RcfTraceLevel)	Specifies the global trace level
RFC trace on (RcfTraceOn)	Specifies whether to generate a text file detailing the RFC activity for each event listener
SAP system ID (SAPSystemID)	Specifies the system ID of the SAP system for which logon load balancing is allowed
Secure network connection library path (SncLib)	Specifies the path to the library that provides the secure network connection service
Secure network connection name (SncMyname)	Specifies the name of the secure network connection
Secure network connection partner (SncPartnername)	Specifies the name of the secure network connection partner
Secure network connection security level (SncQop)	Specifies the level of security for the secure network connection
Select the queue name (QRFCQueueName)	The name of a customer-defined queue on the SAP server.
System number (SystemNumber)	The system number of the SAP application server
Use wait parameter before calling BAPI commit	Specifies whether the WAIT parameter is set on a BAPI_TRANSACTION_COMMIT.
User name (userName)	The user account for the adapter on the SAP server

Table 134. Interaction specification properties for Adapter for SAP Software (continued)

Property name	Description
X509 certificate (X509cert)	Specifies the X509 certificate to be used as the logon ticket

ABAP debug

This property specifies whether the adapter invokes the ABAP Debugger for the appropriate function module when the adapter starts to process a business object.

Table 135. ABAP debug details

Required	No
Possible values	`True` `False`
Default	`False`
Property type	Boolean
Usage	When the property is set to `True`, the adapter opens the SAP GUI in debug mode. You must have appropriate authorization to use the debugger. Create a dialog user ID because a CPI-C user ID cannot open an SAP GUI session. You must have authorization to run in debug mode as well as any authorizations that are required by the ABAP code that is being debugged. For example, if a BAPI_CUSTOMER_CREATEFROMDATA1 is being debugged, you must have authorization to create customers. You can add breakpoints only after the debugger opens. Always set this property to `False` in a production environment. This property is supported on Windows systems only.
Globalized	No
Bidi supported	No

Client

This property is the client number of the SAP system to which the adapter connects.

Table 136. Client details

Required	Yes
Possible values	You can enter a range of values from `000` to `999`.
Default	`100`
Property type	Integer
Usage	When an application attempts to log on to the SAP server, the application must have a Client number associated with it. The Client property value identifies the client (the adapter) that is attempting to log onto the SAP server.
Globalized	No
Bidi supported	No

Codepage number (Codepage)

The numeric identifier of the code page.

Table 137. Codepage number details

Required	No
Possible values	You can enter a range of values from `0000` to `9999`. For a full listing of languages and associated code page numbers that are supported by SAP, see SAP Note 7360.
Default	The default value for this property is conditionally determined by the value set for the **Language code** property.
Property type	Integer
Usage	The value that is assigned to the **Codepage number** defines the code page to be used and has a one-to-one relationship with the value set for the **Language code** property. The **Codepage number** establishes a connection to the appropriate language. Each language code value has a code page number value associated with it. For example, the language code for English is EN. If you select EN (English) as your language code, the code page number is automatically set to the numeric value that is associated with EN (English). The SAP code page number for EN (English) is `1100`.
Example	If **Language code** is set to `JA` (Japanese), **Codepage number** is set to `8000`.
Globalized	No
Bidi supported	No

Custom retrieve function name (customFunctionName)

For the Query interface for SAP Software, this property specifies the name of a custom function that must be used to retrieve data from an SAP table.

Table 138. Custom retrieve function name details

Required	No
Default	No default value
Property type	String
Usage	This property applies to Query interface for SAP Software only. On non-Unicode systems, the default function used to retrieve data from SAP tables (RFC_READ_TABLE) might produce an exception. To avoid the problem, you can create another function on the SAP server and indicate, during configuration, that the adapter must use this custom function to retrieve data. This property specifies the name of the custom function. You must create the function on the SAP server before you specify this property on the Adapter Connection wizard. Follow the steps listed in SAP note 758278 to make a copy of RFC_READ_TABLE and modify the copy in line with the note.
Globalized	No
Bidi supported	No

Enable Secure Network Connection (SncMode)

This property indicates whether secure network connection mode is enabled.

Table 139. Enable Secure Network Connection details

Required	No
Possible values	`0` (off) `1` (on)
Default	`0`
Property type	String
Usage	Set the value to 1 (on) if you want to use secure network connection. If you set this value to 1, you must also set following properties: • Secure Network Connection library path (SncLib) • Secure Network Connection name (SncMyname) • Secure Network Connection partner (SncPartnername) • Secure Network Connection security level (SncQop)
Globalized	No
Bidi supported	No

Folder for RFC trace files (RfcTracePath)

This property sets the fully qualified local path to the folder in which to write RFC trace files.

Table 140. Folder for RFC trace files details

Required	No
Default	No default value
Property type	String
Usage	Identifies the fully qualified local path into which RFC trace files are written. If **RFC trace on** is set to False (not selected), you are not permitted to set a value in the **Folder for RFC trace files** property.
Example	`c:\temp\rfcTraceDir`
Globalized	Yes
Bidi supported	No

Function name (functionName)

The `functionName` interaction specification property controls the interaction by associating operations with the correct interface.

Table 141. Function name details

Required	Yes
Possible values	`True` `False`
Default	Null
Property type	String

Table 141. Function name details (continued)

Usage	The BAPI/RFC supports the following values for the functionName interaction specification property:
	`WBIInteractionSpec.CREATE`
	`WBIInteractionSpec.UPDATE`
	`WBIInteractionSpec.RETRIEVE`
	`WBIInteractionSpec.DELETE`
	
	The BAPI result set supports the following value for the functionName interaction specification property:
	
	`WBIInteractionSpec.RETRIEVEALL`
	
	The ALE outbound interface supports the following value:
	
	`WBIInteractionSpec.EXECUTE`
	
	The ALE inbound interface supports the following values for the functionName interaction specification property:
	
	`WBIInteractionSpec.CREATE`
	`WBIInteractionSpec.UPDATE`
	`WBIInteractionSpec.RETRIEVE`
	`WBIInteractionSpec.DELETE`
	
	The Query interface for SAP software (QISS) interface supports the following values for the functionName interaction specification property:
	• `WBIInteractionSpec.EXISTS`
	Throws the exceptions `NotExistsException` and `QISSQueryFailedException`
	• `WBIInteractionSpec.RETRIEVEALL`
	Throws the exceptions `QISSQueryFailedException`
	
	The Advanced event processing interface for inbound processing supports the following values for the functionName interaction specification property:
	
	`WBIInteractionSpec.CREATE`
	`WBIInteractionSpec.UPDATE`
	`WBIInteractionSpec.DELETE`
	
	The Advanced event processing interface for outbound processing supports the following values for the functionName interaction specification property:
	
	`WBIInteractionSpec.CREATE`
	`WBIInteractionSpec.UPDATE`
	`WBIInteractionSpec.RETRIEVE`
	`WBIInteractionSpec.DELETE`

Globalized	No
Bidi supported	No

Gateway host (GatewayHost)

This property is the Gateway host name. Enter either the IP address or the name of the Gateway host. Consult your SAP administrator for information on the Gateway host name.

Table 142. Gateway host details

Required	Yes

Table 142. Gateway host details (continued)

Default	No default value
Property type	String
Usage	This property is the host name of the SAP gateway. The gateway enables communication between work processes on the SAP system and external programs. The host identified is used as the gateway for the resource adapter. Maximum length of 20 characters. If the computer name is longer than 20 characters, define a symbolic name in the THOSTS table.
Globalized	No
Bidi supported	No

Gateway service (GatewayService)

This property is the identifier of the gateway on the gateway host that carries out the RFC services.

Table 143. Gateway service details

Required	Yes
Default	`sapgw00`
Property type	String
Usage	These services enable communication between work processes on the SAP server and external programs. The service typically has the format of `sapgw00`, where 00 is the SAP system number. Maximum of 20 characters.
Globalized	No
Bidi supported	No

Host name (ApplicationServerHost)

Specifies the IP address or the name of the application server host that the adapter logs on to.

Table 144. Host name details

Required	Yes (when load balancing is not used).
Default	No default value
Property type	String
Usage	When the adapter is configured to run without load balancing, this property specifies the IP address or the name of the application server that the adapter logs on to.
Example	`sapServer`
Globalized	No
Bidi supported	No

Ignore errors in BAPI return (IgnoreBAPIReturn)

This property indicates whether to ignore errors that are specified in a BAPI return operation. The return structure can be data or a table.

Table 145. Ignore errors in BAPI return details

Required	No
Possible values	`True` `False`
Default	`False`
Property type	Boolean
Usage	This property applies to BAPI outbound synchronous RFC processing only. When you set this property to `True`, the Adapter for SAP Software ignores the checking of error codes in the BAPI RETURN structure after BAPI has run, and returns this structure to the user as it is. The RETURN structure is part of every BAPI and contains the status of the BAPI execution. If you accept the default value of `False`, the Adapter for SAP Software processes the RETURN structure and throws an exception if an error code is found.
Globalized	No
Bidi supported	No

Language code

This property specifies the Language code in which the adapter logs on.

Table 146. Language code details

Required	Yes
Possible values	For a full listing of languages and associated code page numbers that are supported by SAP, see SAP Note 7360.
Default	The default value for the **Language code** property is based on the system locale.
Property type	String
Usage	Each of the supported languages is preceded by a two-character language code. The language itself is displayed in parentheses. If you enter a language code manually, you do not need to enter the language in parentheses. The language codes that are listed represent the SAP default set of 41 languages for non-Unicode systems plus Arabic. The value that you select determines the value of the **Codepage number** property.
Example	If the system locale is English, the value for this property is `EN (English)`.
Globalized	No
Bidi supported	No

Maximum number of hits for the discovery (ResultSetLimit)

For the Query interface for SAP Software, this property specifies the maximum number of result sets, which represents data for each row that is retrieved from a table through a RetrieveAll operation.

Table 147. Result set limit details

Required	Yes
Default	`100`
Property type	Integer

Table 147. Result set limit details (continued)

Usage	This property applies to Query interface for SAP Software only. If the number of hits in the table on the SAP server exceeds the value of the ResultSetLimit property, the adapter returns the error: `MatchesExceededLimitException`. The adapter uses this property to help avoid out-of-memory issues.
Globalized	No
Bidi supported	No

Message server host (MessageServerHost)

This property specifies the name of the host on which the message server is running.

Table 148. Message server host details

Required	Yes (if load balancing is used)
Default	No default value
Property type	String
Usage	This property specifies the name of the host that will inform all the servers (instances) belonging to this SAP system of the existence of the other servers to be used for load balancing. The message server host contains the information about load balancing for RFC clients so that an RFC client can be directed to an appropriate application server.
Example	`SAPERP05`
Globalized	No
Bidi supported	No

Partner character set (PartnerCharset)

This property specifies the partner character set encoding.

Table 149. Partner character set details

Required	No
Default	UTF-8
Property type	String
Usage	When an encoding is specified, it is used; otherwise, the default encoding is used.
Globalized	No
Bidi supported	No

Password

This property is the password of the user account of the adapter on the SAP application server.

Table 150. Password details

Required	Yes
Default	No default value
Property type	String

Table 150. Password details (continued)

Usage	The restrictions on the password depend on the version of SAP Web Application Server. • For SAP Web Application Server version 6.40 or earlier, the password: – Must be uppercase – Must be 8 characters in length • For versions of SAP Web Application Server later than 6.40, the password: – Is not case-sensitive – Can be up to 40 characters in length
Globalized	No
Bidi supported	Yes

RFC trace level (RcfTraceLevel)

This property specifies the global trace level.

Table 151. RFC trace level details

Required	No
Possible values	`1` - This is the default RFC trace level. When specified, SAP JCo Java API logging occurs. `3` - When specified, SAP JCo JNI API logging occurs. `5` - When specified, error diagnostic logging occurs.
Default	`1`
Property type	Integer
Usage	If **RFC trace on** is set to `False` (not selected), you cannot set a value in the **RFC trace level** property.
Globalized	No
Bidi supported	No

RFC trace on (RcfTraceOn)

This property specifies whether to generate a text file detailing the RFC activity for each event listener.

Table 152. RFC trace on details

Required	No
Possible values	`True` `False`
Default	`False`
Property type	Boolean
Usage	A value of `True` activates tracing, which generates a text file. This file is created in the directory in which the adapter process was started. The file has a prefix of `rfx` and a file type of trc (for example, rfc03912_02220.trc). Use these text files in a development environment only, because the files can grow rapidly. If **RFC trace on** is set to `False` (not selected), you cannot set values in the **Folder for RFC trace files** or **RFC trace level** properties.

Table 152. RFC trace on details (continued)

Example	Examples of the information in the file are RfcCall FUNCTION BAPI_CUSTOMER_GETLIST, followed by the information for the parameters in the interface, or RFC Info rfctable, followed by the data from one of the interface tables. The trace file is created in the directory where the adapter process has been started. The trace file has a .trc file extension and the file name will start with the letters `rfc` followed by a unique identifier. For example, `rfc03912_02220.trc`.
Globalized	No
Bidi supported	No

SAP system ID (SAPSystemID)

This property specifies the system ID of the SAP system for which logon load balancing is allowed.

Table 153. SAP system ID details

Required	Yes (when load balancing is used)
Default	No default value
Property type	String
Usage	Value must be three characters
Example	`DYL`
Globalized	No
Bidi supported	No

Secure Network Connection library path (SncLib)

This property specifies the path to the library that provides the secure network connection service.

Table 154. Secure Network Connection library path details

Required	Yes, if SncMode is set to 1; no otherwise.
Default	No default value
Property type	String
Usage	If the SncMode property is set to 1 (indicating that you are using a secure network connection), specify the path to the library that provides the service.
Example	`/WINDOWS/system32/gssapi32.dll`
Globalized	No
Bidi supported	No

Secure Network Connection name (SncMyname)

This property specifies the name of the secure network connection.

Table 155. Secure Network Connection name details

Required	Yes, if SncMode is set to 1; no otherwise.
Default	No default value
Property type	String

Table 155. Secure Network Connection name details (continued)

Usage	If the SncMode property is set to 1 (indicating that you are using a secure network connection), specify a name for the connection.
Example	`DOMAINNAME/USERNAME`
Globalized	No
Bidi supported	No

Secure Network Connection partner (SncPartnername)

This property specifies the name of the secure network connection partner.

Table 156. Secure Network Connection partner details

Required	Yes, if SncMode is set to 1; no otherwise.
Default	No default value
Property type	String
Usage	If the SncMode property is set to 1 (indicating that you are using a secure network connection), specify a name for the connection partner.
Example	`CN=sap00.saperpdev, OU=Adapter, O=IBM, C=US`
Globalized	No
Bidi supported	No

Secure Network Connection security level (SncQop)

This property specifies the level of security for the secure network connection.

Table 157. Secure Network Connection security level details

Required	Yes, if SncMode is set to 1; no otherwise.
Possible values	`1` (Authentication only) `2` (Integrity protection) `3` (Privacy protection) `8` (Use the value from `snc/data_protection/use` on the application server) `9` (Use the value from `snc/data_protection/max` on the application server)
Default	`3` (Privacy protection)
Property type	String
Usage	If the SncMode property is set to 1 (indicating that you are using a secure network connection), specify a value to indicate the level of security for the connection.
Globalized	No
Bidi supported	No

Select the queue name (QRFCQueueName)

For BAPI outbound processing, when Asynchronous queued RFC is selected, this property specifies the name of a queue on the SAP server to which BAPIs will be delivered.

Table 158. Select the queue name details

Required	No

Table 158. Select the queue name details (continued)

Default	The first queue defined on the SAP server. If no queue is defined on the SAP server, no default value exists.
Property type	String
Usage	This property applies to BAPI outbound asynchronous queued RFC processing only. When you want to deliver BAPI calls to a queue on the SAP server, you must specify the name of the queue. During configuration, you select an existing queue from a drop-down list. If no queues exist on the SAP server, you can type the name of a queue.
Globalized	No
Bidi supported	No

System number (SystemNumber)

This property is the system number of the SAP application server.

Table 159. System number details

Required	Yes
Possible values	You can enter a range of values from `00` to `99`.
Default	`00`
Property type	Integer
Usage	The system number further identifies the Gateway service.
Globalized	No
Bidi supported	No

Use wait parameter before calling BAPI commit

This property indicates whether the adapter calls a BAPI_TRANSACTION_COMMIT with the WAIT parameter set.

Table 160. Use wait parameter before calling BAPI commit

Required	No
Possible values	`True` `False`
Default	`False`
Property type	Boolean
Usage	This property applies to BAPI outbound synchronous RFC processing only. When you set this property to `True`, the Adapter for SAP Software calls a BAPI_TRANSACTION_COMMIT with the WAIT parameter set, so that other resource managers are committed only when the update has been completed in SAP. If you accept the default value of `False`, message flow processing might continue before the update has been completed in SAP. Therefore, after calling a BAPI, the updated data might not be available when you next access the system.
Globalized	No
Bidi supported	No

User name (userName)

This property is the user account for the adapter on the SAP server.

Table 161. User name details

Required	Yes
Default	No default value
Property type	String
Usage	Maximum length of 12 characters. The user name is not case sensitive. It is recommended that you set up a CPIC user account in the SAP application and that you give this account the necessary privileges to manipulate the data required by the business objects supported by the adapter. For example, if the adapter must perform certain SAP business transactions, the adapter's account in the SAP application must have the permissions set to allow it to perform these transactions.
Example	`SapUser`
Globalized	Yes
Bidi supported	Yes

X509 certificate (X509cert)

This property specifies the X509 certificate to be used as the logon ticket.

Table 162. X509 certificate details

Required	No
Default	No default value
Property type	String
Usage	If the SncMode property is set to 1 (indicating that you are using a secure network connection), you can provide a value for the X509 certificate.
Globalized	No
Bidi supported	No

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you

must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

SAP options for rediscovery:

In WebSphere Message Broker Version 7.0, you can add newly discovered objects into an existing adapter component without modifying any existing objects. This facility is known as *iterative discovery*. You can rediscover certain objects for an inbound or outbound adapter.

SAP outbound

The following table identifies the options that you can use to rediscover objects for an outbound adapter.

Discovery option	Existing XSD files affected	Description
BAPI	None	Rediscovery is possible. The addition of another BAPI does not require any existing XSD file to be replaced.
BAPI (Wrapper)	SAP*BusinessObjectName*Wrapper.xsd	By default, this option is cleared. Rediscovery is possible provided that you do not change the name of the business object.
BAPI (Work Unit)	SAP*BusinessObjectName*Txn.xsd	Rediscovery is possible provided that you do not change the name of the business object.

Discovery option	Existing XSD files affected	Description
BAPI (Result Set)	All XSD files need to be replaced	Incremental rediscovery is not possible because you can specify only two objects (Query and Response).
ALE passthrough IDoc	None	Rediscovery is possible provided that you do not change configuration parameters such as qRFC, which changes ASI annotations on top-level objects.
ALE	None	Rediscovery is possible provided that you do not change configuration parameters such as qRFC, which changes ASI annotations on top-level objects.
Advanced event processing (AEP)	None	Rediscovery is possible provided that you do not change configuration parameters such as the function module name, which changes ASI annotations on top-level objects.
Query interface for SAP Software (QISS)	None	Rediscovery is possible provided that you do not change configuration parameters such as custom function name, which changes ASI annotations on top-level objects.

SAP inbound

The following table identifies the options that you can use to rediscover objects for an inbound adapter.

Discovery option	Existing XSD files affected	Description
ALE events: create, update, or delete	None	Rediscovery is possible if you add a new IDoc but do not change Create, Read, Update, or Delete operations on previously discovered IDocs.
ALE passthrough	All XSD files need to be replaced	Rediscovery is possible, but you can specify only one IDoc. The IDoc is treated as opaque binary. Ensure that you select the option to replace all files.
AEP	None	Rediscovery is possible because the addition of another IDoc does not change existing XSD files.
BAPI	None	Rediscovery is possible if you add a new BAPI and do not change Create, Read, Update, or Delete operations on previously discovered BAPIs.

Related concepts:

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

Related tasks:

"Enhancing existing adapters with newly discovered objects" on page 2063
In WebSphere Message Broker Version 7.0, you can take an adapter component that was created by using the Adapter Connection wizard, and update it with newly discovered objects from the Enterprise Information System (EIS). This facility is known as *iterative discovery*. You can either add the new objects without modifying existing objects, or replace existing objects.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

Related reference:

"Configuration properties for the WebSphere Adapter for SAP Software" on page 4043
The WebSphere Adapter for SAP Software has several categories of configuration properties, which you set with the Adapter Connection wizard when you generate or create objects and services.

WebSphere Adapter for Siebel properties:

Reference information to refer to when you connect to a Siebel application.

- "Business object information (Siebel)" on page 4093
- "Supported data operations (Siebel)" on page 4094
- "Naming conventions for business objects representing Siebel business services" on page 4095
- "Configuration properties for the WebSphere Adapter for Siebel Business Applications" on page 4098
- "Siebel connection properties for the Adapter Connection wizard" on page 4099
- "Inbound adapter properties for Siebel" on page 4107
- "Outbound adapter properties for Siebel" on page 4116

Related concepts:

"Overview of WebSphere Adapter for Siebel Business Applications" on page 2002
With WebSphere Adapter for Siebel Business Applications, you can create integrated processes that exchange information with a Siebel application, without special coding.

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

Related tasks:

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

Related reference:

"WebSphere Adapter for PeopleSoft properties" on page 4122
Reference information to refer to when you connect to a PeopleSoft application.

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

Business object information (Siebel):

A business object is a structure that contains application-specific information (metadata) about how the adapter should process the business object as well as the operation to be performed on the business object.

The name of the business object is generated by the Adapter Connection wizard in accordance with the naming convention for the adapter.

The Siebel business objects are created with long names by default. To generate business objects with shorter names, select **Generate business objects with shorter names** on the Configure Objects screen of the Adapter Connection wizard.

For more information about business objects, see the following topics.

- "Naming conventions for business objects representing Siebel business services" on page 4095
- "Supported data operations (Siebel)" on page 4094

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for Siebel Business Applications" on page 2002
With WebSphere Adapter for Siebel Business Applications, you can create integrated processes that exchange information with a Siebel application, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for Siebel properties" on page 4092
Reference information to refer to when you connect to a Siebel application.

"SiebelInput node" on page 4740
Use the SiebelInput node to interact with a Siebel application.

"SiebelRequest node" on page 4745
Use the SiebelRequest node to interact with a Siebel application.

Supported data operations (Siebel):

An operation is the action that an adapter can perform on the Siebel server during outbound processing. The name of the operation typically indicates the type of action that the adapter takes, such as create or update.

Table 163. Supported operations of business objects

Operation	Definition
Create	Creates the business component
Delete	Deletes the business component and its children
Exists	Checks for the existence of incoming business objects
Retrieve	Retrieves the values of the business component
Retrieve all	Retrieves multiple instances of the same business component
Update	Updates the Siebel application with the incoming object

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for Siebel Business Applications" on page 2002
With WebSphere Adapter for Siebel Business Applications, you can create integrated processes that exchange information with a Siebel application, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for Siebel properties" on page 4092
Reference information to refer to when you connect to a Siebel application.

"SiebelInput node" on page 4740
Use the SiebelInput node to interact with a Siebel application.
"SiebelRequest node" on page 4745
Use the SiebelRequest node to interact with a Siebel application.

Naming conventions for business objects representing Siebel business services:

When the Adapter Connection wizard generates a business object, it provides a name for the business object based on the name of the object in the Siebel application that it uses to build the business object.

Naming conventions for business objects that represent Siebel business services and integration components

The naming conventions for business objects that represent Siebel business services are the same for both inbound and outbound processing. The names comprise the concatenation of several words, including prefix, business service name, integration object, and method name.

The following table describes the naming conventions that the Adapter Connection wizard uses to name business objects that represent Siebel business services and integration components.

Table 164. Business object naming conventions for Siebel business services and integration components

Element	Naming convention
Name of the top-level business object	*<Prefix><Business Service Name><Method Name><Names of all the integration objects selected for the Input and InputOutput complex type arguments>* • If no Input or InputOutput arguments exist, the names of all the output arguments are: *<Prefix><Business Service Name><Method Name><Names of all the integration objects selected for the output complex type arguments>* • If the method contains no complex arguments in the method, the naming convention is: *<Prefix><Business Service Name><Method Name>*
Name of the inbound object that is generated against integration components	'IO' + *<Name of Integration Object>* + 'IC' + *<Name of integration component>* For example: `IOAccountInterfaceICAccount`
Name of the outbound object that is generated against integration components	'IO' + *<Name of Integration Object>* + 'IC' + *<Name of integration component>* The name of an account interface integration object with the integration component account, as shown in the following example: `IOAccountInterfaceICAccount`

Optional: Shorter naming conventions for business objects that are generated against Siebel business services and integration components

The naming conventions for business objects that are generated against Siebel business services and integration components are valid if the optional 'Generate business objects with shorter names' property is specified on the configuration objects pane in the Adapter Connection wizard.

If this optional property is used, you should set the 'Folder' property with a unique value to avoid overwriting existing xsds that were previously generated. For example, if you select 'EAI Siebel Adapter', and click **Query** in two different Adapter Connection wizard runs for the integration objects, 'Account (PRM ANI)' and 'ATP Check Interface', the top-level object is named `EAISiebelAdapter.xsd`.

The name comprises the concatenation of several words, including prefix, business service name, and integration component name.

The following table describes the naming conventions that the Adapter Connection wizard uses to name business objects that are generated against Siebel business services and integration components.

Table 165. Shorter business object naming conventions for business objects that are generated against Siebel business services and integration components

Element	Naming convention
Name of the inbound and outbound child business objects that are generated against integration components	*<Prefix>+<Name of the Siebel Integration Component>* The Siebel business object and integration component names are stripped of all non-alphanumeric characters before being added to the business object name. If the resulting names are not unique, a counter is added to the end of the names.
Name of the inbound top-level business object that is generated against business services and integration components	*<Prefix>+<Name of the Siebel Integration Component>* The Siebel business object and integration component names are stripped of all non-alphanumeric characters before being added to the business object name. If the resulting names are not unique, a counter is added to the end of the names.
Name of the outbound top-level business object that is generated against business services and integration components	*<Prefix>+<Name of the Siebel Business Service>* The Siebel business object and integration component names are stripped of all non-alphanumeric characters before being added to the business object name. If the resulting names are not unique, a counter is added to the end of the names.

Naming conventions for business objects that represent Siebel business objects

The naming conventions for business objects that represent Siebel business objects are the same for both inbound and outbound processing. The name comprises the concatenation of several words, including prefix, business object name, and business component name.

The following table describes the naming conventions that are used by the Adapter Connection wizard to name business objects that represent Siebel business objects.

Table 166. Business object naming conventions for Siebel business objects

Element	Naming convention
Name of the business object	<*Prefix*>+<BO>+<*Business Object Name*>+<BC>+<*Business Component Name*>. The Siebel business object and component names are stripped of all non-alphanumeric characters before being added to the business object name. If the resulting names are not unique, a counter is added to the end of the names. For example, if two business objects have the name, `SiebelBOAccountBCBusinessAddress`, a counter is added as a suffix to make them unique, as shown in this example: `SiebelBOAccountBCAddress1 and SiebelBOAccountBCAddress2`
Name of the container business object that is generated for the Exists operation	<SiebelExistsResult>
Name of the container business object that is generated for the Retrieve All operation	<*Prefix*>+BO+<*Business Object Name*>+<BC>+<*Business Component Name*>+Container
Name of the top-level business object	<*Prefix*>+<BO>+<*Business Object Name*>+<BC>+<*Business Component Name*>

Optional: Shorter naming conventions for business objects that are generated against Siebel business components

The naming conventions for business objects that are generated against Siebel business components are valid if the optional 'Generate business objects with shorter names' property is specified on the configuration objects pane of the Adapter Connection wizard.

If this optional property is used, set the 'Folder' property with a unique value to avoid overwriting existing xsds that were previously generated. For example, **Siebel business object** > **Siebel business component** combination of **Account-ESP** > **Account** and Account (as the top-level object) is named `Account.xsd`.

The name comprises the concatenation of several words, including prefix and business component name.

The following table describes the naming conventions that the Adapter Connection wizard uses to name business objects that are generated against Siebel business components.

Table 167. Shorter business object naming conventions for business objects that are generated against Siebel business components

Element	Naming convention
Name of the top-level business object that is generated against business components	<*Prefix*>+<*Name of the Siebel Business Component*> The Siebel business object and integration component names are stripped of all non-alphanumeric characters before being added to the business object name. If the resulting names are not unique, a counter is added to the end of the names.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for Siebel Business Applications" on page 2002
With WebSphere Adapter for Siebel Business Applications, you can create integrated processes that exchange information with a Siebel application, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for Siebel properties" on page 4092
Reference information to refer to when you connect to a Siebel application.

"SiebelInput node" on page 4740
Use the SiebelInput node to interact with a Siebel application.

"SiebelRequest node" on page 4745
Use the SiebelRequest node to interact with a Siebel application.

Configuration properties for the WebSphere Adapter for Siebel Business Applications:

WebSphere Adapter for Siebel Business Applications has several categories of configuration properties, which you set with the Adapter Connection wizard when you generate or create objects and services.

You can change the connection properties for the Adapter Connection wizard, and the inbound and outbound adapter properties. For more information, see the following topics.

- "Siebel connection properties for the Adapter Connection wizard" on page 4099
- "Inbound adapter properties for Siebel" on page 4107
- "Outbound adapter properties for Siebel" on page 4116

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and

PeopleSoft.

"Overview of WebSphere Adapter for Siebel Business Applications" on page 2002
With WebSphere Adapter for Siebel Business Applications, you can create integrated processes that exchange information with a Siebel application, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for Siebel properties" on page 4092
Reference information to refer to when you connect to a Siebel application.

"SiebelInput node" on page 4740
Use the SiebelInput node to interact with a Siebel application.

"SiebelRequest node" on page 4745
Use the SiebelRequest node to interact with a Siebel application.

Siebel connection properties for the Adapter Connection wizard:

Set Adapter Connection wizard properties to establish a connection between the wizard, a tool that is used to create business objects, and the Siebel server. The properties that you configure in the Adapter Connection wizard specify such things as connection configuration, and logging and tracing options.

After you have established a connection between the Adapter Connection wizard and the Siebel server, the Adapter Connection wizard is able to access the metadata that it needs from the Siebel server to create business objects.

Some of the properties that you set in the Adapter Connection wizard are used as the initial value for resource adapter, managed connection factory, and activation specification properties that you can specify at a later time in the wizard.

The following table describes the Adapter Connection wizard properties and their purpose. A complete description of each property is provided in the sections that follow the table.

If you set any of these connection properties using bidirectional script, you must set values that identify the format of the bidirectional script that is entered for that property.

Table 168. Adapter Connection wizard properties

Property name in the wizard	Description
Adapter style	The service type that is associated with the adapter module
Business object namespace	The namespace value has been added as a prefix to the business object name to keep the business object schemas separated
Code page	The code page that the adapter uses to connect to the Siebel application
Connection URL	The connection URL that you need to connect to the Siebel application
Delimiter for keys in the event store	Specifies that the delimiter that is used between two name value pairs contains the object key name and value
Folder	The location of the generated business object
Generate business objects with shorter names	Ensures that the adapter generates shorter business object names, which are based on the Siebel integration components, business services, and business components rather than the concatenation of several words (which is the default)
Language code	The language code that is used to log on to the Siebel server
Method name	The name of the business service method to be implemented
Password	The password for the corresponding user name
Prefix for business object names	The prefix for the business object name
Siebel business object name for event store	The name of the business object in the event store where events are stored for inbound processing.
Siebel repository name	The name of the Siebel repository from which the objects are to be discovered
Siebel server view mode	Specifies the Siebel server mode and controls the kind of data to be retrieved and what actions can be performed
Type of Siebel objects to discover	The type of Siebel objects (business objects or business services) that need to be discovered and listed
Use resonate support for load balancing on Siebel server	Specifies that if resonate support is installed on the Siebel server, and the value is set to `True`, the adapter takes advantage of the load balancing feature to connect to the Siebel server more efficiently
User name	The user name that is used to log into the Siebel application

Adapter style (AdapterStyle)

This property specifies the service type that is associated with the adapter module.

Table 169. Service type details

Required	Yes
Default	Outbound

Table 169. Service type details (continued)

Property type	List of values
Possible values	`Outbound` `Inbound`
Usage	Specifies the service type associated with the adapter module
Globalized	No
Bidi supported	No

Business object namespace (BusinessObjectNameSpace)

This property specifies that the namespace value has been added as a prefix to the business object name to keep the business object schemas separated.

Table 170. Business object namespace details

Required	Yes
Default	`http://www.ibm.com/xmlns/prod/wbi/j2ca/siebel`
Property type	String
Usage	The namespace value is added as a prefix to the business object name to keep the business object schemas separated.
Example	`http://www.ibm.com/xmlns/prod/wbi/j2ca/siebel/IBMSiebelAccountInsertAccount`
Globalized	No
Bidi supported	No

Code page (CodePage)

Specifies the code page that the adapter uses to connect to the Siebel application.

Table 171. Code page details

Required	No
Default	No default value
Property type	String
Usage	Use this property to specify the code page details that the adapter uses to connect to the Siebel server for metadata discovery. The Siebel Java Data Bean supports a limited list of code page settings. If the current code page setting of your tooling is not supported, the adapter might not be able to connect to the Siebel server.
Globalized	No
Bidi supported	No

Connection URL (ConnectionURL)

This property specifies the connection URL that is needed to connect to the Siebel application.

Table 172. Connection URL details

Required	Yes
Default	No default value

Table 172. Connection URL details (continued)

Property type	String
Usage	The connection URLs for all versions of Siebel follow this format: `Protocol:// machinename:port/enterprisename/object manager/server name` The default port number is 2320. For Siebel version 7.5x and earlier versions, the port number (2320) and server name are specified. For Siebel version 7.8, the port and server name are not specified. If you do not select the default port, you can specify another port number (for example, 2321).
Examples	The following sample connection URLs are for different versions of Siebel: • For Siebel 7.5: `siebel://<IP_address>:2320/siebel/SSEObjMgr_ENU/sebldevl`. • For Siebel 7.8: `siebel://<IP_address>/Sieb78/SSEObjMgr_enu`. • For Siebel 8: `siebel://<IP_address>:2321/SBA_80/SSEObjMgr_enu`.
Globalized	Yes
Bidi supported	Yes

Delimiter for keys in the event store (DelimiterForKeysInTheEventStore)

Table 173. Delimiter for keys in the event store details

Required	Yes
Default	;
Property type	String
Usage	This is the delimiter that is used between two name value pairs that contain the object key name and value.
Examples	You can change the default value for this property. However, if you remove the default value and do not set it again, the default value (;) is used. If the event table key field has values, such as `AccountId=1-314:Id=1-325`, the event delimiter is the colon (:) . The object key names are `AccountId` and `Id`. The values are `1-314` and `1-325`.
Globalized	Yes
Bidi supported	Yes

Folder (Folder)

This property specifies the location of the generated business objects.

Table 174. Folder details

Required	No
Default	No default value
Property type	String
Usage	The generated business objects are copied into this folder.
Example	`inboundartifacts` and `outboundartifacts`
Globalized	No
Bidi supported	No

Generate business objects with shorter names (GenerateBusinessObjectsWithShorterNames)

This property ensures that the adapter generates shorter business object names, which are based on the Siebel integration components, business services, and business components rather than the concatenation of several words (which is the default).

Table 175. Generate business objects with shorter names details

Required	No
Default	No default value
Property type	Boolean
Usage	This property ensures that the adapter generates shorter business object names. The shorter business object names are based on the Siebel integration components, business services, and business components. The prefix is also attached to the shorter names. The adapter removes special characters from the shorter business object names. Alphanumeric characters (a-z, A-Z, and 1-9) are supported, and a counter (1-9) is added to the end of business object names to avoid duplication of names.
Example	If 'Account' is the name of the Siebel business component, and 'Siebel' is the prefix, the shorter name is 'Siebel_Account'.
Globalized	No
Bidi supported	No

Language code (LanguageCode)

This property specifies the language code that is used to log on to the Siebel server.

Table 176. Language code details

Required	Yes
Default	`ENU`
Property type	String
Usage	If the system locale is English, the value for this property is `ENU` (English). This is used to log on to the Siebel server.
Globalized	No
Bidi supported	No

Method name (MethodName)

This property specifies the name of the business service method to be implemented.

Table 177. Method name details

Required	Yes
Default	`Query`
Property type	String
Usage	The default is `Query`.
Example	`Query`, `QueryByExample`, `QueryById`, and so on.
Globalized	Yes

Table 177. Method name details (continued)

Bidi supported	Yes

Password (Password)

This property specifies the password for the corresponding user name.

Table 178. Password details

Required	Yes
Default	No default value
Property type	String
Usage	If a J2C Authentication Alias is used, a password is not required.
Example	`1-XYZ`
Globalized	Yes
Bidi supported	Yes

Prefix for business object names (PrefixForBusinessObjectNames)

This property specifies the prefix for the business object name.

Table 179. Prefix details

Required	No
Default	No default value
Property type	String
Usage	The prefix string is attached to the front of the business object name that was generated.
Example	For example, you use the prefix, `IBM`, and generate a business object for the EAI Siebel Adapter and the Insert method. Then you choose the Account Interface and Business Address Interface integration object against an Input and InputOutput method argument. The corresponding business object that is generated is: `IBMEAISiebelAdapterInsertAccountInterfacBusinessAddressInterface`
Globalized	Yes
Bidi supported	Yes

Siebel business object name for event store (SiebelBusinessObjectNameForEventStore)

This property specifies the name of the business object in the event store where events are stored for inbound processing.

Table 180. Siebel business object name for event store details

Required	Yes
Default	`IBM_EVENT`
Property type	String
Usage	To set this property, click **Advanced** on the connection properties pane on the Adapter Connection wizard, then go to the **Event configuration** tab. The two values listed are `IBM_EVENT` and `IBM2`. If you create a custom event component name, you can specify the value for it in the list box.
Globalized	Yes

Table 180. Siebel business object name for event store details (continued)

Bidi supported	No

Siebel repository name (SiebelRepositoryName)

This property specifies the name of the Siebel repository from which the objects are discovered.

Table 181. Siebel repository name details

Required	Yes
Default	`Siebel repository`
Property type	String
Usage	This default value is `Siebel Repository`. Although this is a required field, it is optional on the Adapter Connection wizard. You can edit this value to point to other repositories if needed.
Globalized	No
Bidi supported	No

Siebel server view mode (SiebelServerViewMode)

This property specifies the Siebel server view mode and controls the data that can be retrieved and what actions can be performed on it.

Table 182. Siebel server view mode details

Required	Yes
Default	`3`
Property type	Integer
Usage	This property displays when you click **Advanced** on the connection properties pane of the Adapter Connection wizard. This mode, when set to "Type of Siebel objects to discover" applies only to `Siebel business objects`, not to `Siebel business services`. The values that are supported by Siebel are `1` to `9`.
Globalized	No

Type of Siebel objects to discover (TypeOfSiebelObjectsToDiscover)

This property specifies the type of Siebel object that needs to be discovered and listed.

Table 183. Type of Siebel objects to discover details

Required	Yes
Possible values	`Siebel business objects` and `Siebel business services`
Default	`Siebel business objects`
Property type	String
Usage	Although the default is `Siebel business objects`, you can select `Siebel business services`. Based on your selection, the Adapter Connection wizard retrieves either the business objects or the business services.
Globalized	No
Bidi supported	No

Use resonate support for load balancing on Siebel server (UseResonateSupportForLoadBalancingOnSiebelServer)

This property indicates whether the Siebel server uses resonate support.

Table 184. Use resonate support for load balancing on Siebel server details

Required	No
Possible values	`True` `False`
Default	`True`
Property type	Boolean
Usage	This property displays after you click **Advanced** on the connection properties pane of the Adapter Connection wizard. If you select the check box, the property is set to `True` and the adapter takes advantage of the load balancing feature to connect to the Siebel server more efficiently. If you clear the check box, the property is set to `False`.
Globalized	No

User name (UserName)

This property specifies the user name that is used to log into the Siebel application.

Table 185. User name details

Required	Yes
Default	No default value
Property type	String
Usage	If a J2C Authentication Alias is used, a user name is not required.
Globalized	Yes
Bidi supported	Yes

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for Siebel Business Applications" on page 2002
With WebSphere Adapter for Siebel Business Applications, you can create integrated processes that exchange information with a Siebel application, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for Siebel properties" on page 4092
Reference information to refer to when you connect to a Siebel application.

"SiebelInput node" on page 4740
Use the SiebelInput node to interact with a Siebel application.

"SiebelRequest node" on page 4745
Use the SiebelRequest node to interact with a Siebel application.

Inbound adapter properties for Siebel:

Inbound adapter properties hold the configuration information for inbound event processing of a message endpoint. These properties also control the general operation of the adapter. Use the Adapter Connection wizard to set these properties.

These properties are used during endpoint activation to notify the adapter of eligible event listeners. During inbound processing, the adapter uses these event listeners to receive events before it forwards them to the endpoint.

The following table lists the properties for inbound processing that you set by using the Adapter Connection wizard. A more detailed description of each property is provided in the sections that follow the table.

Table 186. Inbound adapter properties

Property name	Description
Adapter ID property	The name of the adapter instance in the log and trace files
Connection URL	The connection URL that is needed to connect to the Siebel application
Delivery type	Determines the order in which events are delivered by the adapter to the export component.
Do not process events that have a timestamp in the future	Specifies whether the adapter filters out future events by comparing the timestamp on each event with the system time
Ensure once-only event delivery	Specifies whether the adapter provides assured once-only delivery of events.
Event component name	The name of the Siebel component for the event table
Event delimiter	Specifies whether the delimiter that is used between two name value pairs contains the object key name and value
Event types to process	A delimited list of event types that indicates to the adapter which events it should deliver
Interval between polling periods	The length of time that the adapter waits between polling periods.

Table 186. Inbound adapter properties (continued)

Property name	Description
Language code	The language code that is used to log on to the Siebel server
Maximum connections	The maximum number of connections that the adapter can use for inbound event delivery.
Maximum events in polling period	The number of events that the adapter delivers to the export component during each poll period.
Minimum connections	The minimum number of connections that the adapter can use for inbound event delivery.
Password	The password for the corresponding user name
Resonate support	Specifies that if resonate support is installed on the Siebel server, and the value is set to `True`, the adapter takes advantage of the load balancing feature to connect to the Siebel server more efficiently
Siebel server view mode	Specifies the Siebel view mode and controls the kind of data to be retrieved and what actions can be performed
Stop the adapter when an error is encountered while polling	Specifies whether the adapter stops polling for events when it encounters an error during polling.
"Retry interval if connection fails (RetryInterval)" on page 4115	The length of time that the adapter waits between attempts to establish a new connection after an error occurs during inbound operations.
User name	The user name that is used to log on to the Siebel application

Adapter ID to use for logging and tracing (AdapterID)

This property identifies a specific deployment, or instance, of the adapter.

Table 187. Adapter ID to use for logging and tracing details

Required	Yes
Default	`001`
Property type	String

Table 187. Adapter ID to use for logging and tracing details (continued)

Usage	This property identifies the adapter instance in the log and trace files, and also helps identify the adapter instance while monitoring adapters. The adapter ID is used with an adapter-specific identifier, SEBLRA, to form the component name used by the Log and Trace Analyzer tool. For example, if the adapter ID property is set to 001, the component ID is SEBLRA001. If you run multiple instances of the same adapter, ensure that the first eight characters of the adapter ID property are unique for each instance so that you can correlate the log and trace information to a particular adapter instance. By making the first seven characters of an adapter ID property unique, the component ID for multiple instances of that adapter is also unique, allowing you to correlate the log and trace information to a particular instance of an adapter. For example, when you set the adapter ID property of two instances of WebSphere Adapter for JD Edwards EnterpriseOne to 001 and 002. The component IDs for those instances, SEBLRA001 and SEBLRA002, are short enough to remain unique, enabling you to distinguish them as separate adapter instances. However, instances with longer adapter ID properties cannot be distinguished from each other. If you set the adapter ID properties of two instances to Instance01 and Instance02, you will not be able to examine the log and trace information for each adapter instance because the component ID for both instances is truncated to SEBLRAInstance. For inbound processing, this property is retrieved from the resource adapter properties. For outbound processing, it is retrieved from the managed connection factory properties.
Globalized	Yes
Bidi supported	No

Connection URL (ConnectionURL)

This property specifies the connection URL that is needed to connect to the Siebel application.

Table 188. Connection URL details

Required	Yes
Default	No default value
Property type	String
Usage	`Protocol://machinename:port/enterprisename/object manager/server name` For Siebel 7.0.5 to 7.5x : `siebel://<IP ADDRESS>/siebel/SSEObjMgr_ENU/sebldev1` For Siebel 7.8 : `siebel://<IP ADDRESS>:2321/Sieb78/SSEObjMgr_enu` For Siebel 8 : `siebel://<IP ADDRESS>:2321/SBA_80/SSEObjMgr_enu` The default port number is 2320. In the examples above (for Siebel versions 7.8 and 8), another port (2321) has been specified.
Globalized	Yes
Bidi supported	Yes

Delivery type (DeliveryType)

This property specifies the order in which events are delivered by the adapter to the export component.

Table 189. Delivery type details

Required	No

Table 189. Delivery type details (continued)

Possible values	`ORDERED` `UNORDERED`
Default	`ORDERED`
Property type	String
Usage	The following values are supported: • `ORDERED`: The adapter delivers events to the export component one at a time. • `UNORDERED`: The adapter delivers all events to the export component at once.
Globalized	No
Bidi supported	No

Do not process events that have a timestamp in the future (FilterFutureEvents)

This property specifies whether the adapter filters out future events by comparing the timestamp on each event with the system time.

Table 190. Do not process events that have a timestamp in the future details

Required	Yes
Possible values	`True` `False`
Default	`False`
Property type	Boolean
Usage	If set to `True`, the adapter compares the time of each event to the system time. If the event time is later than the system time, the event is not delivered. If set to `False`, the adapter delivers all events.
Globalized	No
Bidi supported	No

Ensure once-only event delivery (AssuredOnceDelivery)

This property specifies whether to provide ensured once-only event delivery for inbound events.

Table 191. Ensure once-only event delivery details

Required	Yes
Possible values	`True` `False`
Default	`True`
Property type	Boolean
Usage	When this property is set to `True`, the adapter provides assured once-only event delivery so that each event is delivered only once. A value of `False` does not provide assured once event delivery, but provides better performance. When this property is set to `True`, the adapter attempts to store transaction (XID) information in the event store. If it is set to `False`, the adapter does not attempt to store the information. This property is used only if the export component is transactional. If it is not, no transaction can be used, regardless of the value of this property.
Globalized	No

Table 191. Ensure once-only event delivery details (continued)

Bidi supported	No

Event component name (EventComponentName)

This property specifies the name of the event store where events are stored for inbound processing.

Table 192. Event component name details

Required	Yes
Default	`IBM2` (for Siebel version 7.x) and `IBM Event` (for Siebel version 8)
Property type	String
Usage	The default value is `IBM2` for Siebel version 7.x, and `IBM Event` for Siebel version 8. If you select one of these default values to configure the event business component on the Siebel server, it is the name of the Siebel event business component that was created. You can also select a value from the list of values provided by the adapter. You can edit the list of values. If you create your own Siebel event business component, you can edit the list to include the name of that event business component.
Globalized	Yes
Bidi supported	Yes

Event delimiter (EventDelimiter)

This property indicates that the delimiter that is used between two name value pairs contains the object key name and value.

Table 193. Event delimiter details

Required	Yes
Default	;
Property type	String
Usage	If multiple value pairs are set against the object key in the event component, they are used for the delimiter.
Globalized	No

Event types to process (EventTypeFilter)

This property contains a delimited list of event types that indicates to the adapter which events it should deliver.

Table 194. Event types to process details

Required	No
Possible values	A comma-delimited (,) list of business object types
Default	`null`
Property type	String
Usage	Events are filtered by business object type. If the property is set, the adapter delivers only those events that are in the list. A value of `null` indicates that no filter will be applied and that all events will be delivered to the export component.

Table 194. Event types to process details (continued)

Example	To receive only events that relate to the Customer and Order business objects, specify this value: `Customer,Order`
Globalized	No
Bidi supported	No

Interval between polling periods (PollPeriod)

This property specifies the length of time that the adapter waits between polling periods.

Table 195. Interval between polling periods details

Required	Yes
Possible values	Integers greater than or equal to `0`.
Default	`2000`
Unit of measure	Milliseconds
Property type	Integer
Usage	The poll period is established at a fixed rate, which means that if running the poll cycle is delayed for any reason (for example, if a prior poll cycle takes longer than expected to complete) the next poll cycle occurs immediately to make up for the lost time that was caused by the delay.
Globalized	No
Bidi supported	No

Language code (LanguageCode)

This property specifies the language code that is used to log on to the Siebel server.

Table 196. Language code details

Required	Yes
Default	`ENU`
Property type	String
Usage	If the system locale is English, the value for this property is `ENU` (English), which is used to log on to the Siebel server.
Globalized	No
Bidi supported	No

Maximum connections (MaximumConnections)

This property specifies the maximum number of connections that the adapter can use for inbound event delivery.

Table 197. Maximum connections details

Required	No
Default	1
Property type	Integer

Table 197. Maximum connections details (continued)

Usage	Only positive values are valid. The adapter considers any positive entry less than 1 to be equal to 1. Typing a negative value or 0 for this property might result in runtime errors.
Globalized	No
Bidi supported	No

Maximum events in polling period (PollQuantity)

This property specifies the number of events that the adapter delivers to the export component during each poll period.

Table 198. Maximum events in polling period details

Required	Yes
Default	`10`
Property type	Integer
Usage	The value must be greater than 0. If this value is increased, more events are processed per polling period, and the adapter might perform less efficiently. If this value is decreased, fewer events are processed per polling period, and the adapter's performance might improve slightly.
Globalized	No
Bidi supported	No

Minimum connections (MinimumConnections)

This property specifies the minimum number of connections that the adapter can use for inbound event delivery.

Table 199. Minimum connections details

Required	No
Default	`1`
Property type	Integer
Usage	Only positive values are valid. Any value less than 1 is treated as 1 by the adapter. Typing a negative value or 0 for this property might result in runtime errors.
Globalized	No
Bidi supported	No

Password (Password)

This property specifies the password for the corresponding user name.

Table 200. Password details

Required	Yes
Default	No default value
Property type	String
Usage	This property displays after you click **Advanced** on the connection properties pane of the Adapter Connection wizard. The password is saved in `.import` and `.export` files so that the adapter can connect to the Siebel application after it has been deployed. If a J2C Authentication Alias is used, a password is not required.
Example	`sadmin`

Table 200. Password details (continued)

Globalized	Yes
Bidi supported	Yes

Resonate support (ResonateSupport)

This property indicates whether the Siebel server uses resonate support.

Table 201. Resonate support details

Required	No
Possible values	`True` `False`
Default	`True`
Property type	Boolean
Usage	If you set this property to `True`, the adapter takes advantage of the load balancing feature to connect to the Siebel server more efficiently.
Globalized	No

Siebel server view mode (SiebelServerViewMode)

This property specifies the Siebel view mode and controls the data that can be retrieved and what actions can be performed on it.

Table 202. View mode details

Required	Yes
Default	`3`
Property type	Integer
Usage	The View mode property applies only to Siebel business objects and not to Siebel business services.
Globalized	No

Stop the adapter when an error is encountered while polling (StopPollingOnError)

This property specifies whether the adapter stops polling for events when it encounters an error during polling.

Table 203. Stop the adapter when an error is encountered while polling details

Required	No
Possible values	`True` `False`
Default	`False`
Property type	Boolean
Usage	If this property is set to `True`, the adapter stops polling when it encounters an error. If this property is set to `False`, the adapter logs an exception when it encounters an error during polling and continues polling.
Globalized	No

Table 203. Stop the adapter when an error is encountered while polling details (continued)

Bidi supported	No

Retry interval if connection fails (RetryInterval)

When the adapter encounters an error that is related to the inbound connection, this property specifies the length of time that the adapter waits before trying to establish a new connection.

Table 204. Retry interval details

Required	Yes
Default	`2000`
Unit of measure	Milliseconds
Property type	Integer
Usage	Only positive values are valid. When the adapter encounters an error that is related to the inbound connection, this property specifies the length of time that the adapter waits before trying to establish a new connection.
Globalized	No
Bidi supported	No

User name (UserName)

This property specifies the user name that is used to log into the Siebel application.

Table 205. User name details

Required	Yes
Default	No default value
Property type	String
Usage	This property displays after you click **Advanced** on the connection properties pane of the Adapter Connection wizard. The user name is saved in `.import` and `.export` files so that the adapter can connect to the Siebel application after it has been deployed. If a J2C Authentication Alias is used, a password is not required.
Globalized	Yes
Bidi supported	Yes

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for Siebel Business Applications" on page 2002
With WebSphere Adapter for Siebel Business Applications, you can create integrated processes that exchange information with a Siebel application, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for Siebel properties" on page 4092
Reference information to refer to when you connect to a Siebel application.

"SiebelInput node" on page 4740
Use the SiebelInput node to interact with a Siebel application.

"SiebelRequest node" on page 4745
Use the SiebelRequest node to interact with a Siebel application.

Outbound adapter properties for Siebel:

Outbound adapter properties define how the adapter creates an outbound connection instance with the Siebel application, and how operations are run on the server. Use the Adapter Connection wizard to set these properties.

The following table describes the outbound adapter properties. A more detailed description of each property is provided in the sections that follow the table.

Table 206. Managed connection factory properties

Property name	Description
Adapter ID	The name of the adapter instance in the log and trace files
Code page	Specifies the code page that the adapter uses to connect to Siebel server
Connection URL	The connection URL that is needed to connect to the Siebel application
Language code	The language code that is used to log on to the Siebel server
"Maximum records (maxRecords)" on page 4119	Maximum number of records to return during a RetrieveAll operation
Password	The password for the corresponding user name
Prefix	The prefix for the business object name
Resonate support	Specifies that if resonate support is installed on the Siebel server, and the value is set to `True`, the adapter takes advantage of the load balancing feature to connect to the Siebel server more efficiently

Table 206. Managed connection factory properties (continued)

Property name	Description
Show error on empty result set (errorOnEmptyResultSet)	For a business object, this property specifies if the adapter returns an empty business object or RecordNotFoundException when the result of RetreiveAll operation does not return any records. For a business service, the property specifies whether the adapter generates an exception when a returned Siebel message is null.
User name	The user name that is used to log into the Siebel application
View mode	Specifies the Siebel view mode and controls the data that can be retrieved and what actions can be performed on it

Adapter ID (AdapterID)

This property identifies a specific deployment or instance of the adapter.

Table 207. Connection URL details

Required	Yes
Default	`001`
Property type	String
Usage	This property identifies the adapter instance in the log and trace files, and also helps identify the adapter instance while monitoring adapters. The adapter ID is used with an adapter-specific identifier, SEBLRA, to form the component name used by the Log and Trace Analyzer tool. For example, if the adapter ID property is set to 001, the component ID is SEBLRA001. If you run multiple instances of the same adapter, ensure that the first seven characters of the adapter ID property are unique for each instance so that you can correlate the log and trace information to a particular adapter instance. By making the first seven characters of an adapter ID property unique, the component ID for multiple instances of that adapter is also unique, allowing you to correlate the log and trace information to a particular instance of an adapter. For example, when you set the adapter ID property of two instances of WebSphere Adapter for Siebel Business Applications to 001 and 002. The component IDs for those instances, SEBLRA001 and SEBLRA002, are short enough to remain unique, enabling you to distinguish them as separate adapter instances. However, instances with longer adapter ID properties cannot be distinguished from each other. If you set the adapter ID properties of two instances to Instance01 and Instance02, you will not be able to examine the log and trace information for each adapter instance because the component ID for both instances is truncated to SEBLRAInstanc.
Globalized	Yes
Bidi supported	No

Code page (codePage)

Specifies the code page that the adapter uses to connect to Siebel server.

Table 208. Code page details

Required	No
Default	No default value
Property type	String

Table 208. Code page details (continued)

Usage	Use this property to specify the code page details that the adapter uses to connect to the Siebel server. The Siebel Java Data Bean supports a limited list of code page settings. If the current code page setting of your run time is not supported, the adapter might not be able to connect to the Siebel server. The adapter uses the code page property to change the `file.encoding` property of the current Java Virtual Machine (JVM), when it establishes its first connection to the Siebel server. This change might affect other applications that run on the same JVM. Ensure that the code page has a valid value for all applications.
Globalized	No
Bidi supported	No

Connection URL (ConnectionURL)

This property specifies the connection URL that is needed to connect to the Siebel application.

Table 209. Connection URL details

Required	Yes
Default	No default value
Property type	String
Usage	`Protocol://machinename:port/enterprisename/object manager/server name` For Siebel 7.0.5 to 7.5x : `siebel://<IP ADDRESS>/siebel/SSEObjMgr_ENU/sebldev1` For Siebel 7.8 : `siebel://<IP ADDRESS>:2321/Sieb78/SSEObjMgr_enu` For Siebel 8 : `siebel://<IP ADDRESS>:2321/SBA_80/SSEObjMgr_enu` The default port number is 2320. In the examples above (for Siebel versions 7.8 and 8) another port (2321) has been specified.
Globalized	Yes
Bidi supported	Yes

Language code (LanguageCode)

This property specifies the language code that is used to log on to the Siebel server.

Table 210. Language code details

Required	Yes
Possible values	None
Default	`ENU`
Property type	String
Usage	If the system locale is English, the value for this property is `ENU` (English). This value is used to log on to the Siebel server.
Globalized	No
Bidi supported	No

Maximum records (maxRecords)

This property specifies the maximum number of records to return during a RetrieveAll operation.

Table 211. Maximum records details

Required	Yes
Default	100
Usage	If the number of hits in the database exceeds the value of the MaximumRecords property, the adapter returns the MatchesExceededLimitException error. The adapter uses this property to help avoid out-of-memory issues.
Property type	Integer
Globalized	No
Bidi supported	No

Password (Password)

This property specifies the password for the corresponding user name.

Table 212. Password details

Required	Yes
Default	No default value
Property type	String
Example	`sadmin`
Usage	This property displays after you click **Advanced** on the connection properties pane of the Adapter Connection wizard. The password is saved in `.import` and `.export` files so that the adapter can connect to the Siebel application after it has been deployed. If a J2C Authentication Alias is used, a password is not required.
Globalized	Yes
Bidi supported	Yes

Prefix (Prefix)

This property specifies the prefix for the business object name.

Table 213. Prefix details

Required	No
Default	No default value
Property type	String
Usage	The prefix string is attached to the front of the business object name.
Example	If you use the prefix `IBM`, generate a business object for the EAI Siebel Adapter and the Insert method, and choose the integration object, Account (PRM ANI), the corresponding business object generated is: `IBMEAISiebelAdapterInsertAccountU40PRMANIU41` where U40 and U41 are the unicode value replacements of `(and)`.
Globalized	Yes
Bidi supported	Yes

Resonate support (ResonateSupport)

This property indicates whether the Siebel server uses resonate support.

Table 214. Resonate support details

Required	No
Possible values	`True` `False`
Default	`True`
Property type	Boolean
Usage	If you select the check box, the property is set to `True`, and the adapter takes advantage of the load balancing feature to connect to the Siebel server more efficiently. If you clear the check box, the property is set to `False`.
Globalized	No

Show error on empty result set (errorOnEmptyResultSet)

For a business object, this property specifies if the adapter returns an empty business object or RecordNotFoundException when the result of `RetreiveAll` operation does not return any records. For a business service, this property specifies whether the adapter generates an exception when a returned Siebel message is null.

Table 215. Show error on empty result set details

Required	No
Default	For a business object: `True` For a business service: `False`
Property type	Boolean
Usage	For a business object: • When this property is set to `True`, a "record not found" exception is issued if no records are found in RetreiveAll operation. • When this property is set to `False`, an empty business object is returned if no records are found in RetreieveAll operation. For a business service: • When this property is set to `True`, a "record not found" exception is issued if no records are returned. • When this property is set to `False`, a null Siebel message is returned if no records are returned.
Globalized	No
Bidi supported	No

User name (UserName)

This property specifies the user name that is used to log into the Siebel application.

Table 216. User name details

Required	Yes
Possible values	None
Default	No default value

Table 216. User name details (continued)

Property type	String
Usage	This property displays after you click **Advanced** on the connection properties pane of the Adapter Connection wizard. The user name is saved in `.import` and `.export` files so that the adapter can connect to the Siebel application after it has been deployed. If a J2C Authentication Alias is used, a password is not required.
Globalized	Yes
Bidi supported	Yes

View mode (ViewMode)

This property specifies the Siebel view mode and controls the data that can be retrieved and what actions can be performed on it.

Table 217. View mode details

Required	Yes
Possible values	`1 - 9`
Default	`3`.
Property type	Integer
Usage	The View mode property applies only to Siebel business objects and not to Siebel business services. When this property is used for Siebel business objects, the default value is `3`.
Example	The adapter supports values `1` to `9`. For example, `1` is Manager View, `2` is Personal View, and `3` is All View.
Globalized	No

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for Siebel Business Applications" on page 2002
With WebSphere Adapter for Siebel Business Applications, you can create integrated processes that exchange information with a Siebel application, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection

wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for Siebel properties" on page 4092
Reference information to refer to when you connect to a Siebel application.

"SiebelInput node" on page 4740
Use the SiebelInput node to interact with a Siebel application.

"SiebelRequest node" on page 4745
Use the SiebelRequest node to interact with a Siebel application.

WebSphere Adapter for PeopleSoft properties:

Reference information to refer to when you connect to a PeopleSoft application.

- "Business-object information (PeopleSoft)" on page 4123
- "Supported operations (PeopleSoft)" on page 4124
- "PeopleCode for a custom event project" on page 4125
- "Configuration properties for the WebSphere Adapter for PeopleSoft Enterprise" on page 4131
- "PeopleSoft connection properties for the Adapter Connection wizard" on page 4132
- "Inbound adapter properties for PeopleSoft" on page 4135
- "Outbound adapter properties for PeopleSoft" on page 4143

Related concepts:

"Overview of WebSphere Adapter for PeopleSoft Enterprise" on page 2013
With the adapter for PeopleSoft Enterprise you can create integrated processes that exchange information with PeopleSoft Enterprise through a standard interface. This interface shields the message flow developer from having to understand the lower level details about implementation of the application or data structures used on the PeopleSoft Enterprise server.

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

Related tasks:

"Creating a custom event project in PeopleTools" on page 2083
The WebSphere Adapter requires an event project in PeopleSoft to perform asynchronous inbound event processing. Use PeopleTools to create the custom event project.

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

Related reference:

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

Business-object information (PeopleSoft):

Information about the content of a business object is located primarily inside the business-object definition file, a file that is generated by the Adapter Connection wizard when it creates business objects.

The business-object definition file contains application-specific information (ASI), which the adapter uses to perform operations, such as creating or updating. You can also find information about the content of a business object in the name of the business object. Although business-object names have no semantic value, they often contain clues about what the business object contains and what operation the adapter can perform on the PeopleSoft Enterprise server. For example, a business object named UpdateAddress suggests that the adapter can update an address in the PeopleSoft Enterprise server.

For more information, see "Supported operations (PeopleSoft)" on page 4124.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for PeopleSoft Enterprise" on page 2013
With the adapter for PeopleSoft Enterprise you can create integrated processes that exchange information with PeopleSoft Enterprise through a standard interface. This interface shields the message flow developer from having to understand the lower level details about implementation of the application or data structures used on the PeopleSoft Enterprise server.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Creating a custom event project in PeopleTools" on page 2083
The WebSphere Adapter requires an event project in PeopleSoft to perform asynchronous inbound event processing. Use PeopleTools to create the custom event project.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for PeopleSoft properties" on page 4122
Reference information to refer to when you connect to a PeopleSoft application.
"PeopleSoftInput node" on page 4630
Use the PeopleSoftInput node to interact with a PeopleSoft application.
"PeopleSoftRequest node" on page 4635
Use the PeopleSoftRequest node to interact with a PeopleSoft application.

Supported operations (PeopleSoft):

An operation is the action that an adapter can perform on the PeopleSoft Enterprise server during outbound processing. The name of the operation typically indicates the type of action that the adapter takes, such as create or update.

The following table defines the operations that the adapter supports.

Table 218. Supported operations of business objects

Operation	Definition
Create	The adapter accesses the PeopleSoft component and retrieves values from the attributes that have the primary key application-specific information set. The adapter then opens the corresponding component interface by using the value that is provided for the ObjectName application-specific information. It sets the attribute values on the corresponding Create Keys in the component interface. An empty Component Interface is created, and the adapter maps all the business-object data to the created component interface. When mapping the data, the adapter sends all data for simple attributes in the hierarchy, and it creates items that match each of the child objects in the hierarchy, including effective-dated and effective-sequenced child records.
Retrieve	The adapter accesses the PeopleSoft component and retrieves values from the attributes that have the primary key application-specific information set. The adapter then instantiates the corresponding component interface by using the value that is provided for the ObjectName application-specific information. It sets the attribute values on the corresponding Get Keys in the component interface. The adapter then maps the component data onto the business-object hierarchy. Child objects are included in the data mapping.
RetrieveAll	This operation performs in the same way as the Retrieve operation, except that it allows retrieval of multiple instances of the same PeopleSoft component.
Update	The adapter retrieves an object from PeopleSoft and compares it to the target business object. When the comparison reveals extra child objects in PeopleSoft, the adapter deletes children. When the comparison reveals missing children in PeopleSoft, the adapter creates children. When the comparison reveals child objects that have been updated in PeopleSoft, the adapter updates them.
Exists	The adapter processes an exist operation in the same way that it processes a retrieve operation, except that it does not populate the business object with retrieved data; it checks for the existence of an object in PeopleSoft.
Delete	Based on the values that are set for the application-specific metadata elements StatusColumnName and StatusValue, the adapter updates a business object to inactive. A delete operation can be performed only on a top level object. PeopleSoft does not allow an object to be physically deleted, so the inactive object remains in the PeopleSoft database.
Apply Changes	This operation updates the PeopleSoft component based on the operation that was performed on it. The supported operations are create, update, and delete.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and

PeopleSoft.

"Overview of WebSphere Adapter for PeopleSoft Enterprise" on page 2013
With the adapter for PeopleSoft Enterprise you can create integrated processes that exchange information with PeopleSoft Enterprise through a standard interface. This interface shields the message flow developer from having to understand the lower level details about implementation of the application or data structures used on the PeopleSoft Enterprise server.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Creating a custom event project in PeopleTools" on page 2083
The WebSphere Adapter requires an event project in PeopleSoft to perform asynchronous inbound event processing. Use PeopleTools to create the custom event project.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for PeopleSoft properties" on page 4122
Reference information to refer to when you connect to a PeopleSoft application.

"PeopleSoftInput node" on page 4630
Use the PeopleSoftInput node to interact with a PeopleSoft application.

"PeopleSoftRequest node" on page 4635
Use the PeopleSoftRequest node to interact with a PeopleSoft application.

PeopleCode for a custom event project:

Two PeopleCode functions are required to support inbound processing. If you create a custom event project in PeopleTools for inbound support, add the PeopleCode functions to the project.

The following PeopleCode contains the IBMPublishEvent and IBMPublishFutureDatedEvent functions that are used to publish events to the event table. Calls to these functions are made from the SavePostChange PeopleCode function in the PeopleSoft component of interest.

```
/* IBM event notification */
Component string &KEYSTRING;
Component string &KEYNAME;
```

```
Component array of string &KEYARRAY;
Component string &KEYDELIM;
Component string &IBMVERB;
Local Record &IBMREC;

Function IBMPublishFutureDatedEvent(&BO, &KEYS, &EFFDATE)
; /* == create a new record object for cw_event_tbl == */
   &IBMREC = CreateRecord(Record.IBM_EVENT_TBL);
 /* ============== KEYS =============== */
/* composing keys and values in name value format */
   &KEYSTRING = "";
   &KEYDELIM = ":";
   &KEYARRAY = Split(&KEYS, &KEYDELIM);
   &LEN = &KEYARRAY.Len;
   For &I = 1 To &LEN;
/* get keys and values */
/* get rid of record name */
      &POS1 = Find(".", &KEYARRAY [&I]);
      &L1 = Len(&KEYARRAY [&I]);
      &POS2 = &L1 - &POS1;
      &KEYNAME = Right(&KEYARRAY [&I], &POS2);
/****The code below will remove special characters and****/
/****adjust the characters' case to ensure it is same as the****/
/****attribute name in the business object definition***/
/****Start****/
      &lLen = Len(&KEYNAME);
      &sOrigString = &KEYNAME;
      &sNewString = "";
      &lCtr2 = 1;
      &isSpecialChar = "true";
      For &lCtr = 1 To &lLen;
         &sChar = Substring(&sOrigString, &lCtr, 1);
         If (&sChar = "A" Or
               &sChar = "a" Or
               &sChar = "B" Or
               &sChar = "b" Or
               &sChar = "C" Or
               &sChar = "c" Or
               &sChar = "D" Or
               &sChar = "d" Or
               &sChar = "E" Or
               &sChar = "e" Or
               &sChar = "F" Or
               &sChar = "f" Or
               &sChar = "G" Or
               &sChar = "g" Or
               &sChar = "H" Or
               &sChar = "h" Or
               &sChar = "I" Or
               &sChar = "i" Or
               &sChar = "J" Or
               &sChar = "j" Or
               &sChar = "K" Or
               &sChar = "k" Or
               &sChar = "L" Or
               &sChar = "l" Or
               &sChar = "M" Or
               &sChar = "m" Or
               &sChar = "N" Or
               &sChar = "n" Or
               &sChar = "O" Or
               &sChar = "o" Or
               &sChar = "P" Or
               &sChar = "p" Or
               &sChar = "Q" Or
               &sChar = "q" Or
               &sChar = "R" Or
               &sChar = "r" Or
               &sChar = "S" Or
               &sChar = "s" Or
               &sChar = "T" Or
               &sChar = "t" Or
               &sChar = "U" Or
               &sChar = "u" Or
               &sChar = "V" Or
               &sChar = "v" Or
               &sChar = "W" Or
               &sChar = "w" Or
               &sChar = "X" Or
               &sChar = "x" Or
               &sChar = "Y" Or
               &sChar = "y" Or
               &sChar = "Z" Or
               &sChar = "z" Or
               &sChar = "1" Or
               &sChar = "2" Or
               &sChar = "3" Or
               &sChar = "4" Or
               &sChar = "5" Or
               &sChar = "6" Or
               &sChar = "7" Or
               &sChar = "8" Or
```

```
               &sChar = "9" Or
               &sChar = "0") Then
            If (&isSpecialChar = "true") Then
               &sNewString = &sNewString | Upper(&sChar);
               &isSpecialChar = "false";
            Else
               &sNewString = &sNewString | Lower(&sChar);
            End-If;
         Else
            &isSpecialChar = "true";
         End-If;
      End-For;
      &KEYNAME = &sNewString;
/*********End*********/
      &KEYSTRING = &KEYSTRING | &KEYNAME | "=" | @&KEYARRAY [&I] | &KEYDELIM
   End-For;
   &KEYSTRING = RTrim(&KEYSTRING, ":");
   &IBMREC.IBM_OBJECT_KEYS.Value = &KEYSTRING;
/*============== VERB ========================*/
/*  verb determination uses variable &IBMVERB */
   Evaluate %Mode
   When = "A"
      &IBMVERB = "Create";
      Break;
   When = "U"
      &IBMVERB = "Update";
      Break;
   When = "L"
      &IBMVERB = "Update";
      Break;
   When = "C"
      &IBMVERB = "Update";
      Break;
   When-Other
      &IBMVERB = "Retrieve";
   End-Evaluate;
   &IBMREC.IBM_OBJECT_VERB.Value = &IBMVERB;
/*  =====================  EVENT_ID GEN   ================================== */
/*  create event_id */
   &NEWNUM = GetNextNumber(IBM_FETCH_ID.IBM_NEXT_EVENT_ID, 99999);
/*  only use newnum if no error generating next number  */
   If &NEWNUM > 0 Then
      &IBMREC.IBM_EVENT_ID.Value = &NEWNUM;
   Else
      &IBMREC.IBM_EVENT_ID.Value = %Datetime;
   End-If; /*Support for Future Effective Date - The adapter will poll such events when the date arrives*/
   If &EFFDATE > %Datetime Then
      &IBMREC.IBM_EVENT_DTTM.Value = &EFFDATE;
      &IBMREC.IBM_EVENT_STATUS.Value = "99";
   Else
      &IBMREC.IBM_EVENT_DTTM.Value = %Datetime;
      &IBMREC.IBM_EVENT_STATUS.Value = "0";
   End-If; /*================  INSERT EVENT INTO IBM_EVENT_TBL ============*/
/*  insert row into table using record object*/
   &IBMREC.IBM_OBJECT_NAME.Value = &BO;
   &IBMREC.Insert();
End-Function;
```

```
Function IBMPublishEvent(&BO, &KEYS);
   /* == create a new record object for cw_event_tbl == */
   &IBMREC = CreateRecord(Record.IBM_EVENT_TBL);


   /* =============== KEYS ================ */
   /* composing keys and values in name value format */
   &KEYSTRING = "";
   &KEYDELIM = ":";
   &KEYARRAY = Split(&KEYS, &KEYDELIM);
   &LEN = &KEYARRAY.Len;

   For &I = 1 To &LEN;
      /* get keys and values */
      /* get rid of record name */
      &POS1 = Find(".", &KEYARRAY [&I]);
      &L1 = Len(&KEYARRAY [&I]);
      &POS2 = &L1 - &POS1;
      &KEYNAME = Right(&KEYARRAY [&I], &POS2);

      /****The code below will remove special characters and
      /****adjust the characters' case to ensure it is same as the
      /****attribute name in the business object definition***/
      /****Start****/
```

```
&lLen = Len(&KEYNAME);
&sOrigString = &KEYNAME;
&sNewString = "";
&lCtr2 = 1;
&isSpecialChar = "true";
For &lCtr = 1 To &lLen;
   &sChar = Substring(&sOrigString, &lCtr, 1);
   If (&sChar = "A" Or
         &sChar = "a" Or
         &sChar = "B" Or
         &sChar = "b" Or
         &sChar = "C" Or
         &sChar = "c" Or
         &sChar = "D" Or
         &sChar = "d" Or
         &sChar = "E" Or
         &sChar = "e" Or
         &sChar = "F" Or
         &sChar = "f" Or
         &sChar = "G" Or
         &sChar = "g" Or
         &sChar = "H" Or
         &sChar = "h" Or
         &sChar = "I" Or
         &sChar = "i" Or
         &sChar = "J" Or
         &sChar = "j" Or
         &sChar = "K" Or
         &sChar = "k" Or
         &sChar = "L" Or
         &sChar = "l" Or
         &sChar = "M" Or
         &sChar = "m" Or
         &sChar = "N" Or
         &sChar = "n" Or
         &sChar = "O" Or
         &sChar = "o" Or
         &sChar = "P" Or
         &sChar = "p" Or
         &sChar = "Q" Or
         &sChar = "q" Or
         &sChar = "R" Or
         &sChar = "r" Or
         &sChar = "S" Or
         &sChar = "s" Or
         &sChar = "T" Or
         &sChar = "t" Or
         &sChar = "U" Or
         &sChar = "u" Or
         &sChar = "V" Or
         &sChar = "v" Or
         &sChar = "W" Or
         &sChar = "w" Or
         &sChar = "X" Or
         &sChar = "x" Or
         &sChar = "Y" Or
         &sChar = "y" Or
         &sChar = "Z" Or
         &sChar = "z" Or
         &sChar = "1" Or
         &sChar = "2" Or
         &sChar = "3" Or
         &sChar = "4" Or
         &sChar = "5" Or
```

```
            &sChar = "6" Or
            &sChar = "7" Or
            &sChar = "8" Or
            &sChar = "9" Or
            &sChar = "0") Then
         If (&isSpecialChar = "true") Then
            &sNewString = &sNewString | Upper(&sChar);
            &isSpecialChar = "false";
         Else
            &sNewString = &sNewString | Lower(&sChar);
         End-If;
      Else
         &isSpecialChar = "true";
      End-If;
   End-For;
   &KEYNAME = &sNewString;

   /*********End*********/
   &KEYSTRING = &KEYSTRING | &KEYNAME | "=" | @&KEYARRAY [&I] | &KEYDELIM
End-For;
&KEYSTRING = RTrim(&KEYSTRING, ":");

&IBMREC.IBM_OBJECT_KEYS.Value = &KEYSTRING;

/*============== VERB ========================*/
/*  verb determination uses variable &IBMVERB */
Evaluate %Mode
When = "A"
   &IBMVERB = "Create";
   Break;
When = "U"
   &IBMVERB = "Update";
   Break;
When = "L"
   &IBMVERB = "Update";
   Break;
When = "C"
   &IBMVERB = "Update";
   Break;
When-Other
   &IBMVERB = "Retrieve";
End-Evaluate;

&IBMREC.IBM_OBJECT_VERB.Value = &IBMVERB;

/*  =====================  EVENT_ID GEN   ============================== */
/*  create event_id */

&NEWNUM = GetNextNumber(IBM_FETCH_ID.IBM_NEXT_EVENT_ID, 99999);

/*  only use newnum if no error generating next number  */

If &NEWNUM > 0 Then
   &IBMREC.IBM_EVENT_ID.Value = &NEWNUM;
Else
   &IBMREC.IBM_EVENT_ID.Value = %Datetime;
End-If;

&IBMREC.IBM_EVENT_DTTM.Value = %Datetime;

/* ==============  EVENT_STATUS  =================*/
/* Validate and set event status &IBMSTATUS - list values if date is ok*/
&IBMREC.IBM_EVENT_STATUS.Value = "0";
```

```
   /*================  INSERT EVENT INTO IBM_EVENT_TBL ============*/
   /*  insert row into table using record object*/


   &IBMREC.IBM_OBJECT_NAME.Value = &BO;

   &IBMREC.Insert();

End-Function;
```

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for PeopleSoft Enterprise" on page 2013
With the adapter for PeopleSoft Enterprise you can create integrated processes that exchange information with PeopleSoft Enterprise through a standard interface. This interface shields the message flow developer from having to understand the lower level details about implementation of the application or data structures used on the PeopleSoft Enterprise server.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Creating a custom event project in PeopleTools" on page 2083
The WebSphere Adapter requires an event project in PeopleSoft to perform asynchronous inbound event processing. Use PeopleTools to create the custom event project.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for PeopleSoft properties" on page 4122
Reference information to refer to when you connect to a PeopleSoft application.

"PeopleSoftInput node" on page 4630
Use the PeopleSoftInput node to interact with a PeopleSoft application.

"PeopleSoftRequest node" on page 4635
Use the PeopleSoftRequest node to interact with a PeopleSoft application.

Configuration properties for the WebSphere Adapter for PeopleSoft Enterprise:

WebSphere Adapter for PeopleSoft Enterprise has several categories of configuration properties, which you set with the Adapter Connection wizard when it generates or creates objects and services.

You can change the connection properties for the Adapter Connection wizard, and the inbound and outbound adapter properties. For more information, see the following topics.

- "PeopleSoft connection properties for the Adapter Connection wizard" on page 4132
- "Inbound adapter properties for PeopleSoft" on page 4135
- "Outbound adapter properties for PeopleSoft" on page 4143

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for PeopleSoft Enterprise" on page 2013
With the adapter for PeopleSoft Enterprise you can create integrated processes that exchange information with PeopleSoft Enterprise through a standard interface. This interface shields the message flow developer from having to understand the lower level details about implementation of the application or data structures used on the PeopleSoft Enterprise server.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Creating a custom event project in PeopleTools" on page 2083
The WebSphere Adapter requires an event project in PeopleSoft to perform asynchronous inbound event processing. Use PeopleTools to create the custom event project.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for PeopleSoft properties" on page 4122
Reference information to refer to when you connect to a PeopleSoft application.

"PeopleSoftInput node" on page 4630
Use the PeopleSoftInput node to interact with a PeopleSoft application.
"PeopleSoftRequest node" on page 4635
Use the PeopleSoftRequest node to interact with a PeopleSoft application.

PeopleSoft connection properties for the Adapter Connection wizard:

Connection properties for the Adapter Connection wizard are used to establish a connection between the Adapter Connection wizard and the application from which the wizard obtains metadata. These properties specify such things as connection configuration, and logging options.

If you set any of these connection properties using bidirectional script, you must set values that identify the format of the bidirectional script entered for that property.

The connection properties and their purpose are described in the following table. A complete description of each property is provided in the sections that follow the table.

Table 219. Connection properties

Property name	Description
"Component interface jar file"	The PeopleSoft Enterprise component interface that the adapter uses to establish a connection to the PeopleSoft components that are targets of integration transactions
"Host name " on page 4133	The name or address of the server that hosts PeopleSoft Enterprise
"Password " on page 4133	The password of the user account of the adapter on the PeopleSoft Enterprise server
"Port number" on page 4133	The port number at which PeopleSoft Enterprise is configured to listen for client requests.
"Prefix for business object names" on page 4133	A prefix to be added to generated business objects
"User name" on page 4134	The name of the user account that the adapter uses on the PeopleSoft Enterprise server

Component interface jar file

This property specifies the PeopleSoft Enterprise component interface that the adapter uses to establish a connection to the PeopleSoft components that are targets of integration transactions.

Table 220. Component interface jar file details

Required	Yes
Default	No default
Property type	String
Usage	The name of the JAR file that the adapter uses to connect to the PeopleSoft Enterprise components of interest
Example	`CWYES_PeopleSoft\connectorModule\WbiEvent.jar`
Globalized	No
Bidi supported	No

Host name

This property specifies the name or address of the server that hosts PeopleSoft Enterprise.

Table 221. Host name details

Required	Yes
Default	No default value
Property type	String
Usage	Identifies the server, either by name or IP address, that hosts PeopleSoft Enterprise
Example	`9.26.248.202`
Globalized	No
Bidi supported	No

Password

This property specifies the password of the user account of the adapter on the PeopleSoft Enterprise server.

Table 222. Password details

Required	Yes
Default	No default value
Property type	String
Usage	The restrictions (case, length, and character) are determined by the PeopleSoft Enterprise version.
Globalized	Yes
Bidi supported	Yes

Port number

This property specifies the port number at which PeopleSoft Enterprise is configured to listen for client requests.

Table 223. Port number details

Required	Yes
Default	The port number that is entered when you run the Adapter Connection wizard
Property type	Integer
Example	9000
Globalized	No
Bidi supported	No

Prefix for business object names

This property specifies a prefix to be added to generated business objects.

Table 224. Prefix details

Required	No
Default	No default

Table 224. Prefix details (continued)

Property type	String
Usage	Use this property to distinguish between different business objects that are generated against the same PeopleSoft component interface.
Example	If you use `IB` as a prefix, all business objects that are generated by this service are named using this prefix.
Globalized	Yes
Bidi supported	No

User name

This property specifies the name of the user account that the adapter uses on the PeopleSoft Enterprise server.

Table 225. User name details

Required	Yes
Default	No default value
Property type	String
Usage	The restrictions (case, length, and character) are determined by the PeopleSoft Enterprise version.
Example	`DV1`
Globalized	Yes
Bidi supported	Yes

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for PeopleSoft Enterprise" on page 2013
With the adapter for PeopleSoft Enterprise you can create integrated processes that exchange information with PeopleSoft Enterprise through a standard interface. This interface shields the message flow developer from having to understand the lower level details about implementation of the application or data structures used on the PeopleSoft Enterprise server.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Creating a custom event project in PeopleTools" on page 2083
The WebSphere Adapter requires an event project in PeopleSoft to perform asynchronous inbound event processing. Use PeopleTools to create the custom event project.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for PeopleSoft properties" on page 4122
Reference information to refer to when you connect to a PeopleSoft application.

"PeopleSoftInput node" on page 4630
Use the PeopleSoftInput node to interact with a PeopleSoft application.

"PeopleSoftRequest node" on page 4635
Use the PeopleSoftRequest node to interact with a PeopleSoft application.

Inbound adapter properties for PeopleSoft:

Inbound adapter properties hold the configuration information for inbound event processing of an export component. These properties also control the general operation of the adapter, such as specifying the namespace for business objects. Use the Adapter Connection wizard to set these properties.

The following table lists the properties for inbound processing that you set by using the Adapter Connection wizard. A more detailed description of each property is provided in the sections that follow the table.

Table 226. Inbound adapter properties

Property name	Purpose
Adapter ID to use for logging and tracing (AdapterID)	The name of the adapter instance in the log and trace files.
Ensure once-only event delivery (AssuredOnceDelivery)	Specifies whether the adapter provides assured once-only delivery of events.
Component interface for testing failed connections (PingCompIntfc)	The component interface that the adapter uses to validate a connection to the PeopleSoft Enterprise server
Component interface name for event notification (EventCIName)	The component interface that the adapter uses for event notification
Delivery type (DeliveryType)	Determines the order in which events are delivered by the adapter to the export component.
Delimiter for keys in the event store (EventKeyDelimiter)	The name and value for an object key in the event table
Event types to process (EventTypeFilter)	A delimited list of event types that indicates to the adapter which events it should deliver
Java date format for event timestamp (DateFormat)	The format that is used to create the event timestamp
Maximum connections (MaximumConnections)	The maximum number of connections that the adapter can use for inbound event delivery.
Minimum connections (MinimumConnections)	The minimum number of connections that the adapter can use for inbound event delivery.

Table 226. Inbound adapter properties (continued)

Property name	Purpose
Interval between polling periods (PollPeriod)	The length of time that the adapter waits between polling periods.
Maximum number of events collected during each poll (PollQuantity)	The number of events that the adapter delivers to the export component during each poll period.
Stop polling on error (StopPollingOnError)	Specifies whether the adapter stops polling for events when it encounters an error during polling.
Retry interval if connection fails (RetryInterval)	The length of time that the adapter waits between attempts to establish a new connection after an error occurs during inbound operations.

Adapter ID to use for logging and tracing (AdapterID)

Required	Yes
Default	`001`
Property type	String
Usage	This property identifies the adapter instance in the log and trace files, and also helps identify the adapter instance while monitoring adapters. The adapter ID is used with an adapter-specific identifier, PSOFTRA, to form the component name used by the Log and Trace Analyzer tool. For example, if the adapter ID property is set to 001, the component ID is PSOFTRA001. If you run multiple instances of the same adapter, ensure that the first eight characters of the adapter ID property are unique for each instance so that you can correlate the log and trace information to a particular adapter instance. By making the first seven characters of an adapter ID property unique, the component ID for multiple instances of that adapter is also unique, allowing you to correlate the log and trace information to a particular instance of an adapter. For example, when you set the adapter ID property of two instances of WebSphere Adapter for PeopleSoft Enterprise to 001 and 002. The component IDs for those instances, PSOFTRA001 and PSOFTRA002, are short enough to remain unique, enabling you to distinguish them as separate adapter instances. However, instances with longer adapter ID properties cannot be distinguished from each other. If you set the adapter ID properties of two instances to Instance01 and Instance02, you will not be able to examine the log and trace information for each adapter instance because the component ID for both instances is truncated to PSOFTRAInstance. For inbound processing, this property is retrieved from the resource adapter properties. For outbound processing, it is retrieved from the managed connection factory properties.
Globalized	Yes
Bidi supported	No

Ensure once-only event delivery (AssuredOnceDelivery)

This property specifies whether to provide ensured once-only event delivery for inbound events.

Table 227. Ensure once-only event delivery details

Required	Yes
Possible values	`True` `False`
Default	`True`
Property type	Boolean
Usage	When this property is set to `True`, the adapter provides assured once-only event delivery so that each event is delivered only once. A value of `False` does not provide assured once event delivery, but provides better performance. When this property is set to `True`, the adapter attempts to store transaction (XID) information in the event store. If it is set to `False`, the adapter does not attempt to store the information. This property is used only if the export component is transactional. If it is not, no transaction can be used, regardless of the value of this property.
Globalized	No
Bidi supported	No

Component interface for testing failed connection (PingCompInterface)

This property specifies the name of the PeopleSoft Enterprise component interface that the adapter uses to validate a connection to the PeopleSoft Enterprise server.

Table 228. Component interface for testing failed connection details

Row	**Explanation**
Required	Yes
Default	The name of the first component interface in the list
Property type	String
Usage	The name of the component interface that the adapter uses to test connectivity to the PeopleSoft Enterprise server. Specify a component interface name that already exists in your PeopleSoft Enterprise applications.
Globalized	No
Bidi supported	No

Component interface name for event notification (EventCIName)

This property specifies the name of the PeopleSoft Enterprise component interface that the adapter uses to for inbound processing.

Table 229. Component interface name for event notification details

Row	**Explanation**
Required	Yes
Default	`IBM_EVENT_CI`
Property type	String
Usage	The name of the component interface that the adapter uses for inbound processing. To use inbound processing, create a component interface specifically for event notification in PeopleSoft Enterprise.
Globalized	No
Bidi supported	No

Delivery type (DeliveryType)

This property specifies the order in which events are delivered by the adapter to the export component.

Table 230. Delivery type details

Required	No
Possible values	`ORDERED` `UNORDERED`
Default	`ORDERED`
Property type	String
Usage	The following values are supported: • `ORDERED`: The adapter delivers events to the export component one at a time. • `UNORDERED`: The adapter delivers all events to the export component at once.
Globalized	No
Bidi supported	No

Delimiter for keys in the event store (EventKeyDelimiter)

This property specifies the delimiter for the object key name-value pair in the event table.

Table 231. Delimiter for keys in the event store details

Row	**Explanation**
Required	No
Default	`=:`
Property type	String
Usage	Use this property to specify an object name and value to be used as an object key in the event store.
Example	`CustomerID=2001`
Globalized	No
Bidi supported	No

Do not process events that have a timestamp in the future (FilterFutureEvents)

This property specifies whether the adapter filters out future events by comparing the timestamp on each event with the system time.

Table 232. Do not process events that have a timestamp in the future details

Required	Yes
Possible values	`True` `False`
Default	`False`
Property type	Boolean

Table 232. Do not process events that have a timestamp in the future details (continued)

Usage	If set to `True`, the adapter compares the time of each event to the system time. If the event time is later than the system time, the event is not delivered. If set to `False`, the adapter delivers all events.
Globalized	No
Bidi supported	No

Event types to process (EventTypeFilter)

This property contains a delimited list of event types that indicates to the adapter which events it should deliver.

Table 233. Event types to process details

Required	No
Possible values	A comma-delimited (,) list of business object types
Default	`null`
Property type	String
Usage	Events are filtered by business object type. If the property is set, the adapter delivers only those events that are in the list. A value of `null` or `*` indicates that no filter will be applied and that all events will be delivered to the export component.
Example	To receive only events that relate to the Customer and Order business objects, specify this value: `Customer,Order`
Globalized	No
Bidi supported	No

Java date format for event timestamp (DateFormat)

This property specifies the format that is used for the event timestamp.

Table 234. Java date format for event timestamp details

Row	**Explanation**
Required	Yes
Default	`MM/dd/yy`
Property type	String
Usage	This property is used to format the date values from the PeopleSoft Enterprise server.
Globalized	No
Bidi supported	No

Maximum connections (MaximumConnections)

This property specifies the maximum number of connections that the adapter can use for inbound event delivery.

Table 235. Maximum connections details

Required	No
Default	1

Table 235. Maximum connections details (continued)

Property type	Integer
Usage	Only positive values are valid. The adapter considers any positive entry less than 1 to be equal to 1. Typing a negative value or 0 for this property might result in runtime errors.
Globalized	No
Bidi supported	No

Minimum connections (MinimumConnections)

This property specifies the minimum number of connections that the adapter can use for inbound event delivery.

Table 236. Minimum connections details

Required	No
Default	1
Property type	Integer
Usage	Only positive values are valid. Any value less than 1 is treated as 1 by the adapter. Typing a negative value or 0 for this property might result in runtime errors.
Globalized	No
Bidi supported	No

Interval between polling periods (PollPeriod)

This property specifies the length of time that the adapter waits between polling periods.

Table 237. Interval between polling periods details

Required	Yes
Possible values	Integers greater than or equal to `0`.
Default	`2000`
Unit of measure	Milliseconds
Property type	Integer
Usage	The poll period is established at a fixed rate, which means that if running the poll cycle is delayed for any reason (for example, if a prior poll cycle takes longer than expected to complete) the next poll cycle occurs immediately to make up for the lost time that was caused by the delay.
Globalized	No
Bidi supported	No

Maximum events in polling period (PollQuantity)

This property specifies the number of events that the adapter delivers to the export component during each poll period.

Table 238. Maximum events in polling period details

Required	Yes
Default	`10`
Property type	Integer

Table 238. Maximum events in polling period details (continued)

Usage	The value must be greater than 0. If this value is increased, more events are processed per polling period, and the adapter might perform less efficiently. If this value is decreased, fewer events are processed per polling period, and the adapter's performance might improve slightly.
Globalized	No
Bidi supported	No

Number of times to retry the system connection (RetryLimit)

This property specifies the number of times that the adapter tries to reestablish an inbound connection.

Table 239. Number of times to retry the system connection details

Required	No
Possible values	Positive integers
Default	`0`
Property type	Integer
Usage	Only positive values are valid. When the adapter encounters an error that is related to the inbound connection, this property specifies the number of times that the adapter tries to restart the connection. A value of `0` indicates an infinite number of retries.
Globalized	Yes
Bidi supported	No

Stop the adapter when an error is encountered while polling (StopPollingOnError)

This property specifies whether the adapter stops polling for events when it encounters an error during polling.

Table 240. Stop the adapter when an error is encountered while polling details

Required	No
Possible values	`True` `False`
Default	`False`
Property type	Boolean
Usage	If this property is set to `True`, the adapter stops polling when it encounters an error. If this property is set to `False`, the adapter logs an exception when it encounters an error during polling and continues polling.
Globalized	No
Bidi supported	No

Retry interval if connection fails (RetryInterval)

When the adapter encounters an error that is related to the inbound connection, this property specifies the length of time that the adapter waits before trying to establish a new connection.

Table 241. Retry interval details

Required	Yes
Default	2000
Unit of measure	Milliseconds
Property type	Integer
Usage	Only positive values are valid. When the adapter encounters an error that is related to the inbound connection, this property specifies the length of time that the adapter waits before trying to establish a new connection.
Globalized	No
Bidi supported	No

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for PeopleSoft Enterprise" on page 2013
With the adapter for PeopleSoft Enterprise you can create integrated processes that exchange information with PeopleSoft Enterprise through a standard interface. This interface shields the message flow developer from having to understand the lower level details about implementation of the application or data structures used on the PeopleSoft Enterprise server.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Creating a custom event project in PeopleTools" on page 2083
The WebSphere Adapter requires an event project in PeopleSoft to perform asynchronous inbound event processing. Use PeopleTools to create the custom event project.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for PeopleSoft properties" on page 4122
Reference information to refer to when you connect to a PeopleSoft application.

"PeopleSoftInput node" on page 4630
Use the PeopleSoftInput node to interact with a PeopleSoft application.
"PeopleSoftRequest node" on page 4635
Use the PeopleSoftRequest node to interact with a PeopleSoft application.

Outbound adapter properties for PeopleSoft:

Outbound adapter properties define how the adapter creates an outbound connection instance with the PeopleSoft Enterprise server, and how operations are run on the server. Use the Adapter Connection wizard to set these properties.

The following table describes the outbound adapter properties. A more detailed description of each property is provided in the sections that follow the table.

Table 242. Managed connection factory properties

Property	Description
Component interface for testing failed connection	The component interface that the adapter uses to validate a connection to the PeopleSoft Enterprise server
Host name	The name or address of the server that hosts PeopleSoft Enterprise
"Language (Language)" on page 4144	The language code that the adapter uses to log on to the PeopleSoft Enterprise server
Maximum number of records for RetrieveAll operation	The maximum number of records to return during a RetrieveAll operation
"Password (Password)" on page 4144	The password of the user account of the adapter on the PeopleSoft Enterprise server
"Port number (Port)" on page 4145	The port number that the adapter uses to access the PeopleSoft Enterprise server
"User name (UserName)" on page 4145	The name of the user account that the adapter uses on the PeopleSoft Enterprise server

Component interface for testing failed connection (PingCompInterface)

This property specifies the name of the PeopleSoft Enterprise component interface that the adapter uses to validate a connection to the PeopleSoft Enterprise server.

Table 243. Component interface for testing failed connection details

Required	Yes
Default	The name of the first component interface in the list
Property type	String
Usage	Specify a component interface name that already exists within your PeopleSoft Enterprise applications.
Example	`WBI_CUSTOMER_CI`
Globalized	No
Bidi supported	No

Host name (HostName)

This property specifies the name or address of the server that hosts PeopleSoft Enterprise.

Table 244. Host name details

Required	Yes
Default	No default value
Property type	String
Usage	Identifies, either by name or IP address, the server that hosts PeopleSoft Enterprise
Example	`9.26.248.202`
Globalized	No
Bidi supported	No

Language (Language)

This property specifies the language code that the adapter uses to log on to the PeopleSoft Enterprise server.

Table 245. Language details

Required	Yes
Default	The default value for the Language property is based on the system locale.
Property type	String
Usage	Each of the supported languages is preceded by a three-character language code. The language itself is presented in parentheses.
Example	If the system locale is English, the value for this property is `ENG (English)`.
Globalized	No
Bidi supported	No

Maximum number of records for RetrieveAll operation (MaxRecords)

This property specifies the maximum number of records to return during a RetrieveAll operation.

Table 246. Maximum number of records for RetrieveAll operation details

Required	Yes
Default	100
Usage	If the number of hits in PeopleSoft Enterprise exceeds the value of the Maximum number of records for the RetrieveAll operation property, the adapter returns an error. The adapter uses this property to help avoid out-of-memory issues.
Property type	Integer
Globalized	No
Bidi supported	No

Password (Password)

This property specifies the password of the user account of the adapter on the PeopleSoft Enterprise server.

Table 247. Password details

Required	Yes
Default	No default value

Table 247. Password details (continued)

Property type	String
Usage	The restrictions (case, length, and character) are determined by the PeopleSoft Enterprise version.
Globalized	No
Bidi supported	No

Port number (Port)

This property specifies the port number that the adapter uses to access the PeopleSoft Enterprise server.

Table 248. Port number details

Required	Yes
Default	The port number that is entered when you use the Adapter Connection wizard to discover objects and services
Property type	Integer
Example	`9000`
Globalized	No
Bidi supported	No

User name (UserName)

This property specifies the name of the user account that the adapter uses on the PeopleSoft Enterprise server.

Table 249. User name details

Required	Yes
Default	No default value
Property type	String
Usage	The restrictions (case, length, and character) are determined by the PeopleSoft Enterprise version.
Example	`DV1`
Globalized	No
Bidi supported	No

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for PeopleSoft Enterprise" on page 2013
With the adapter for PeopleSoft Enterprise you can create integrated processes that exchange information with PeopleSoft Enterprise through a standard interface. This interface shields the message flow developer from having to understand the lower level details about implementation of the application or data structures used on the PeopleSoft Enterprise server.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Creating a custom event project in PeopleTools" on page 2083
The WebSphere Adapter requires an event project in PeopleSoft to perform asynchronous inbound event processing. Use PeopleTools to create the custom event project.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for PeopleSoft properties" on page 4122
Reference information to refer to when you connect to a PeopleSoft application.

"PeopleSoftInput node" on page 4630
Use the PeopleSoftInput node to interact with a PeopleSoft application.

"PeopleSoftRequest node" on page 4635
Use the PeopleSoftRequest node to interact with a PeopleSoft application.

WebSphere Adapter for JD Edwards EnterpriseOne properties:

Reference information to refer to when you connect to a JD Edwards EnterpriseOne application.

- "Business object information (JD Edwards)" on page 4147
- "Adapter configuration properties for JD Edwards" on page 4153

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for JD Edwards EnterpriseOne" on page 2023
With the WebSphere Adapter for JD Edwards EnterpriseOne, you can create integrated processes that include the exchange of information with a JD Edwards EnterpriseOne server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

Business object information (JD Edwards):

You can determine the purpose of a business object by examining both the application-specific information in the business object definition file and the name of the business object.

The application-specific information dictates what operations can be performed on the JD Edwards EnterpriseOne server. The name typically reflects the operation to be performed and the structure of the business object.

For more information about business objects, see the following topics:

- "Application-specific information (JD Edwards)" on page 4148
- "Supported operations (JD Edwards)" on page 4151
- "Naming conventions for JD Edwards EnterpriseOne" on page 4152

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for JD Edwards EnterpriseOne" on page 2023
With the WebSphere Adapter for JD Edwards EnterpriseOne, you can create integrated processes that include the exchange of information with a JD Edwards EnterpriseOne server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for JD Edwards EnterpriseOne properties" on page 4146
Reference information to refer to when you connect to a JD Edwards EnterpriseOne application.

Application-specific information (JD Edwards):

Application-specific information (ASI) is metadata that specifies adapter-dependent information about how to process business objects for the WebSphere Adapter for JD Edwards EnterpriseOne.

When the Adapter Connection wizard generates a business object, it automatically generates a business object definition, which is saved as an XSD (XML Schema Definition) file. The business object definition contains the ASI for that business object.

The WebSphere Adapter for JD Edwards EnterpriseOne uses ASI to create queries for Create, Retrieve, Update, and Delete operations. ASI is generated by the Adapter Connection wizard at three levels: the business-object level, the property level, and the operation level.

Application-specific information at the business-object-level

ASI at the business-object level is typically used to specify the name of the corresponding database table and to provide information necessary to perform a physical or logical delete operation. The following table describes the ASI at the business-object level.

Application-specific information	Description
Name	The name of the operation.
BSFN	The list of business functions associated with the operation.
AlwaysReturnResponse	Used to designate if the adapter returns a response business object for every request. If the value is set to true, the adapter always returns a response business object. If the value is set to false, an exception is generated after a JD Edwards EnterpriseOne business function is completed. This exception is generated against the user's component. The default value is false. For runtime exceptions, for example, if the adapter cannot establish a connection with the JD Edwards EnterpriseOne server, the exception is still generated against the user's component.

Application-specific information at the property level

ASI at the property level is typically used to specify the metadata for a property, and represents either child objects or an array of child objects. The following table describes the ASI of a complex property (a child) or a structure or table property (an array of child objects).

Application-specific information	Description	Possible values
Name	The business function parameter name as represented in JD Edwards EnterpriseOne.	BSFNName
Type	The type of the business function parameter as it exists in JD Edwards EnterpriseOne.	BSFN
IOType	The type of the business function parameter as it exists in JD Edwards EnterpriseOne.	• `IN`: the parameter is mapped from the business object to the business function. • `OUT`: the parameter is mapped from the business function to the business object. • `INOUT`: the parameter is mapped both ways. • `DEFAULT`: the parameter is mapped by using the default JD Edwards EnterpriseOne value. For adapter purposes, it is processed as INOUT.
RequiredType	Identifies if the parameter is required.	• `YES`: the parameter is required. • `NO`: the parameter is not required. • `DEFAULT`: the parameter is using the JD Edwards EnterpriseOne value. For adapter purposes, it is processed as `NO`.
Length	The maximum length for the parameter value.	None

Reference	The XPath of the business object property that is used to obtain the value of this attribute. The XPath expression starts at the business function level	`BusinessFunctionContainer`
		` BusinessFunction1`
		`  Prop1`
		` BusinessFunction2`
		`  Prop2`
		
		If the `BusinessFunction2/Prop2` property must be set with the value of `BusinessFunction1/Prop1`, the value of **Reference** for `BusinessFunction2/Prop2` must be set to `BusinessFunction1/Prop1`.

Application-specific information at the operation level

ASI at the operation level is used by the adapter to complete operations, such as to retrieve or update information in the JD Edwards EnterpriseOne server. The following table describes the ASI at the operation level.

Application-specific information	Description	Value
Name	The name of the business object operation.	• Create • Retrieve • Update • Delete • RetrieveAll
BSFN.Name	The name of the business functions to process.	• Name • RollbackOnWarnings
BSFN.RollbackOnWarnings	Indicates if the adapter must roll back the current transaction when the business function returns with warnings.	False (default setting)
RunOnError	Used to designate if the adapter should continue to process the sequential JD Edwards EnterpriseOne server business functions when a business function encounters an error while completing the business function. If the value is set to Yes, the adapter continues to process the subsequent business functions, even if the business function fails. The error message is stored to the attribute, BSFNExecutionErrors. If the value is set to false, the adapter stops the execution process and a rollback operation is performed. If the ASI `RunOnError` property is set to true for all business functions, the rollback operation is not performed. An error message for each business function that has failed is stored in the attribute BSFNExecutionErrors, against each business function in the response business object.	False (default setting)

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for JD Edwards EnterpriseOne" on page 2023
With the WebSphere Adapter for JD Edwards EnterpriseOne, you can create integrated processes that include the exchange of information with a JD Edwards EnterpriseOne server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere

Adapters, see the following topics.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for JD Edwards EnterpriseOne properties" on page 4146
Reference information to refer to when you connect to a JD Edwards EnterpriseOne application.

Supported operations (JD Edwards):

An operation is the action that an adapter can perform on the JD Edwards EnterpriseOne server during outbound processing. The name of the operation typically indicates the type of action that the adapter takes, such as create or update.

The following tables define the operations that the WebSphere Adapter for JD Edwards EnterpriseOne supports during outbound processing for business functions and XML Lists.

Table 250. Supported operations of business functions

Operation	Definition
Create	The top-level business object and all contained children are created.
Update	The top-level business object is modified. This operation can include adding and deleting child objects.
Delete	The top-level business object and any contained children are deleted.
Retrieve	The top-level business object and any contained children are retrieved.

Table 251. Supported operations of XML Lists

Operation	Definition
RetrieveAll	Retrieves all records from the JD Edwards EnterpriseOne server that correspond to the query values specified in the XML List. Returns a result set in the form of a container of JD Edwards EnterpriseOne query business objects, which represent the data for each row retrieved from the table.

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for JD Edwards EnterpriseOne" on page 2023
With the WebSphere Adapter for JD Edwards EnterpriseOne, you can create integrated processes that include the exchange of information with a JD Edwards

EnterpriseOne server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for JD Edwards EnterpriseOne properties" on page 4146
Reference information to refer to when you connect to a JD Edwards EnterpriseOne application.

Naming conventions for JD Edwards EnterpriseOne:

When the Adapter Connection wizard generates a business object, it provides a name for the business object based on the name of the object in the JD Edwards EnterpriseOne server that it uses to build the business object.

When the Adapter Connection wizard provides a name for the business object, it converts the name of the object to mixed case, removing separators, such as spaces or underscores, then capitalizes the first letter of each word. For example, if the Adapter Connection wizard uses a JD Edwards EnterpriseOne server object called CUSTOMER_ADDRESS to generate a business object, it generates a business object called CustomerAddress.

The generated business object name can indicate the structure of the business object. However, business objects names have no semantic value to the adapter. Therefore, if you change the business object name, the behavior of the business object remains the same.

Element	Naming convention	Example
Name of the business object container	<*name_of_business_object*>Container	GetEffectiveAddressContainer
Name of the business function	Name of the business function discovered by the Adapter Connection wizard	GetEffectiveAddress

Element	Naming convention	Example
Name of the XML List	Name of the XML List table discovered by the Adapter Connection wizard	F0116

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for JD Edwards EnterpriseOne" on page 2023
With the WebSphere Adapter for JD Edwards EnterpriseOne, you can create integrated processes that include the exchange of information with a JD Edwards EnterpriseOne server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for JD Edwards EnterpriseOne properties" on page 4146
Reference information to refer to when you connect to a JD Edwards EnterpriseOne application.

Adapter configuration properties for JD Edwards:

The WebSphere Adapter for JD Edwards EnterpriseOne has several categories of connection configuration properties, which you set with the Adapter Connection wizard while generating or creating objects and services.

You can change the connection properties for the Adapter Connection wizard, and the inbound and outbound adapter properties. For more information, see the following topics:

- "Connection properties for the Adapter Connection wizard (JD Edwards)" on page 4154
- "Inbound adapter properties for JD Edwards" on page 4158
- "Outbound adapter properties for JD Edwards" on page 4165

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for JD Edwards EnterpriseOne" on page 2023
With the WebSphere Adapter for JD Edwards EnterpriseOne, you can create integrated processes that include the exchange of information with a JD Edwards EnterpriseOne server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for JD Edwards EnterpriseOne properties" on page 4146
Reference information to refer to when you connect to a JD Edwards EnterpriseOne application.

Connection properties for the Adapter Connection wizard (JD Edwards):

Adapter configuration properties establish a connection between the Adapter Connection wizard and the JD Edwards EnterpriseOne server. The properties that you configure in the Adapter Connection wizard specify such things as connection configuration, bidirectional properties, and logging and tracing options.

After a connection between the Adapter Connection wizard and the JD Edwards EnterpriseOne server is established, the Adapter Connection wizard is able to access the metadata it needs from the JD Edwards EnterpriseOne server to create business objects.

The adapter connection properties and their purposes are described in the following table. A complete description of each property is provided in the sections that follow the table.

Important: If you set any of these connection properties using bidirectional script, you must set values that identify the format of the bidirectional script entered for that property.

Property	Description
“Environment”	Specifies the JD Edwards EnterpriseOne environment name.
“Log file output location”	Specifies the location of the log file for the discovery process.
“Logging level”	Specifies the type error for which logging occurs during the discovery process.
“Password” on page 4156	Password of the adapter user account on the JD Edwards EnterpriseOne environment.
“Role” on page 4156	Name of the role that is associated with the user name used to access the JD Edwards EnterpriseOne environment.
“User name” on page 4157	Name of the adapter user account on the JD Edwards EnterpriseOne environment.

Environment

This property specifies the JD Edwards EnterpriseOne environment name.

Required	Yes
Default	No default value
Property type	String
Usage	A JD Edwards EnterpriseOne environment is a user-defined pointer that indicates the location of data and objects on a JD Edwards EnterpriseOne server. Users can be authorized to use multiple JD Edwards EnterpriseOne environments on a single JD Edwards EnterpriseOne server.
Globalized	Yes
Bidi supported	Yes

Log file output location

This property specifies the location of the log file for the discovery process.

Required	Yes
Default	The `.metadata` directory of the workspace
Property type	String
Usage	Use this directory to hold the log file that lists the errors that occur during the discovery process. The type of discovery errors for which logging occurs is controlled by the `Logging level` property.
Example	`C:\IBM\wmbt70\workspace\.metadata\JDEMetadataDiscovery.log`
Globalized	Yes
Bidi supported	No

Logging level

This property specifies the type of error for which logging occurs during the discovery process.

Required	No

Possible values	ALL OFF FINE FINER FINEST CONFIG INFO SEVERE WARNING
Default	SEVERE
Property type	String
Usage	Use this property to tailor tracing capabilities. By specifying an error type, you are indicating that trace operations occur only for errors of the type specified.
Example	Accepting the default value of SEVERE provides trace information about errors that fall into the SEVERE category. Severe errors indicate that an operation cannot continue, although the adapter can still function. Severe errors also include error conditions that indicate an impending unrecoverable error, that is, reporting on situations that strongly suggest that resources are on the verge of being depleted. Other error descriptions are described in the following list: • Fatal – The adapter cannot continue. The adapter cannot function. • Warning – A potential error or impending error, which includes conditions that indicate a progressive failure; for example, the potential leaking of resources. • Audit – A significant event affecting adapter state or resources. • Info – General information outlining overall operation progress. • Config – Configuration change or status. • Detail – General information detailing operation progress.
Globalized	Yes
Bidi supported	No

Password

This property specifies the password of the adapter user account on the JD Edwards EnterpriseOne environment.

Required	No
Default	No default value
Property type	String
Usage	Passwords are created and named by the JD Edwards EnterpriseOne administrator. No restrictions exist on the type of characters used, the number of characters used, or the case of the characters used in passwords.
Globalized	No
Bidi supported	Yes

Role

This property specifies the name of the role that is associated with the user name used to access the JD Edwards EnterpriseOne environment.

Required	Yes
Default	No default value

Property type	String
Usage	Roles define what authority users have. Users can have multiple roles. A user's access to applications, forms, table columns, data sources, and so on, is based on one or more roles to which the user is assigned. Roles are created and named by the JD Edwards EnterpriseOne administrator.
Examples	• System administrator • Human resources • Accounting
Globalized	No
Bidi supported	Yes

User name

This property specifies the name of the adapter user account on the JD Edwards EnterpriseOne environment.

Required	No
Default	No default value
Property type	String
Usage	User names are created by the JD Edwards EnterpriseOne administrator. No restrictions exist on the type of characters used, the number of characters used, or the case of the characters used in user names.
Globalized	Yes
Bidi supported	Yes

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for JD Edwards EnterpriseOne" on page 2023
With the WebSphere Adapter for JD Edwards EnterpriseOne, you can create integrated processes that include the exchange of information with a JD Edwards EnterpriseOne server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection

wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for JD Edwards EnterpriseOne properties" on page 4146
Reference information to refer to when you connect to a JD Edwards EnterpriseOne application.

Inbound adapter properties for JD Edwards:

Inbound adapter properties hold the inbound event processing configuration information for a message endpoint. These properties also control the general operation of the adapter. Use the Adapter Connection wizard to set these properties.

The following table lists the properties for inbound processing that you set by using the Adapter Connection wizard. A more detailed description of each property is provided in the sections that follow the table.

Property	Description
"Adapter ID to use for logging and tracing (AdapterID)" on page 4159	The name of the adapter instance in the log and trace files.
"Auto acknowledge (AutoAcknowledge)" on page 4159	Specifies the event acknowledge mode that is used.
"Delivery type (DeliveryType)" on page 4160	Determines the order in which events are delivered by the adapter to the export component.
"Ensure once-only event delivery (AssuredOnceDelivery)" on page 4160	Specifies whether the adapter provides assured once-only delivery of events.
"Failed events folder" on page 4160	The absolute path to the file folder on the local system where unsuccessfully processed events are archived in file format.
Retry limit for failed events	The number of times the adapter attempts to deliver an event before marking the event as failed.
"Interval between polling periods (PollPeriod)" on page 4161	The length of time that the adapter waits between polling periods.
"Maximum connections (MaximumConnections)" on page 4162	The maximum number of connections that the adapter can use for inbound event delivery.
"Maximum events in polling period (PollQuantity)" on page 4162	The number of events that the adapter delivers to the export component during each poll period.
"Minimum connections (MinimumConnections)" on page 4162	The minimum number of connections that the adapter can use for inbound event delivery.
"No wait (NoWait)" on page 4162	Specifies whether the adapter waits for a time interval to get an event from the JD Edwards EnterpriseOne transaction server by calling the Dynamic Java Connector API.
Retry EIS connection on startup	Controls whether the adapter retries the connection to the JD Edwards EnterpriseOne transaction server if it cannot connect at startup.
"Retry interval if connection fails (RetryInterval)" on page 4164	The length of time that the adapter waits between attempts to establish a new connection after an error occurs during inbound operations.
"Stop the adapter when an error is encountered while polling (StopPollingOnError)" on page 4164	Specifies whether the adapter stops polling for events when it encounters an error during polling.

Property	Description
"Wait time (WaitTime)" on page 4164	Specifies the waiting time if the No Wait property is set to false.

Adapter ID to use for logging and tracing (AdapterID)

This property identifies a specific deployment, or instance, of the adapter.

Required	Yes
Default	001
Property type	String
Usage	This property identifies the adapter instance in the log and trace files, and also helps identify the adapter instance while monitoring adapters. The adapter ID is used with an adapter-specific identifier, JDERA, to form the component name used by the Log and Trace Analyzer tool. For example, if the adapter ID property is set to 001, the component ID is JDERA001. If you run multiple instances of the same adapter, ensure that the first eight characters of the adapter ID property are unique for each instance so that you can correlate the log and trace information to a particular adapter instance. By making the first seven characters of an adapter ID property unique, the component ID for multiple instances of that adapter is also unique, allowing you to correlate the log and trace information to a particular instance of an adapter. For example, when you set the adapter ID property of two instances of WebSphere Adapter for JD Edwards EnterpriseOne to 001 and 002. The component IDs for those instances, JDERA001 and JDERA002, are short enough to remain unique, enabling you to distinguish them as separate adapter instances. However, instances with longer adapter ID properties cannot be distinguished from each other. If you set the adapter ID properties of two instances to Instance01 and Instance02, you will not be able to examine the log and trace information for each adapter instance because the component ID for both instances is truncated to JDERAInstance. For inbound processing, this property is retrieved from the resource adapter properties. For outbound processing, it is retrieved from the managed connection factory properties.
Globalized	Yes
bidi supported	No

Auto acknowledge (AutoAcknowledge)

This property specifies the event acknowledge mode that is used. You can specify either the auto acknowledge mode or the client acknowledge mode.

Required	No
Possible values	`True` `False`
Default	`False`
Property type	Boolean
Usage	Specifies the event acknowledge mode, which is either the auto acknowledge mode or the client acknowledge mode.
Example	False
Globalized	No
bidi supported	Yes

Delivery type (DeliveryType)

This property specifies the order in which events are delivered by the adapter to the export component.

Required	No
Possible values	`ORDERED` `UNORDERED`
Default	`ORDERED`
Property type	String
Usage	The following values are supported: • `ORDERED`: The adapter delivers events to the export component one at a time. • `UNORDERED`: The adapter delivers all events to the export component at once.
Globalized	No
bidi supported	No

Ensure once-only event delivery (AssuredOnceDelivery)

This property specifies whether to provide ensured once-only event delivery for inbound events.

Required	Yes
Possible values	`True` `False`
Default	`True`
Property type	Boolean
Usage	When this property is set to `True`, the adapter provides assured once-only event delivery so that each event is delivered only once. A value of `False` does not provide assured once event delivery, but provides better performance. When this property is set to `True`, the adapter attempts to store transaction (XID) information in the event store. If it is set to `False`, the adapter does not attempt to store the information. This property is used only if the export component is transactional. If it is not, no transaction can be used, regardless of the value of this property.
Globalized	No
bidi supported	No

Failed events folder

This property specifies the file folder on the local system where unsuccessfully processed events are archived in file format.

Required	No
Possible values	No default value
Default	Null
Property type	String

Usage	Create this folder manually on the same system before the adapter is run. If no failed events folder is specified, the adapter does not archive unsuccessfully processed events.
Globalized	Yes
bidi supported	No

Retry limit for failed events (FailedEventRetryLimit)

This property specifies the number of times that the adapter attempts to deliver an event before marking the event as failed.

Required	No
Possible values	Integers
Default	5
Property type	Integer
Usage	Use this property to control how many times the adapter tries to send an event before marking it as failed. It accepts the following values: **Default** If this property is not set, the adapter tries five additional times before marking the event as failed. **0** The adapter tries to deliver the event an infinite number of times. When the property is set to 0, the event remains in the event store and the event is never marked as failed. **> 0** For integers greater than zero, the adapter retries the specified number of times before marking the event as failed. **< 0** For negative integers, the adapter does not retry failed events.
Globalized	No
bidi supported	No

Interval between polling periods (PollPeriod)

This property specifies the length of time for which the adapter waits between polling periods.

Required	Yes
Possible values	Integers greater than or equal to 0.
Default	2000
Unit of measure	Milliseconds
Property type	Integer
Usage	The poll period is established at a fixed rate, so that if running the poll cycle is delayed for any reason (for example, if a prior poll cycle takes longer than expected to complete), the next poll cycle occurs immediately to make up for the lost time that was caused by the delay.
Globalized	No
bidi supported	No

Maximum connections (MaximumConnections)

This property specifies the maximum number of connections that the adapter can use for inbound event delivery.

Required	No
Default	1
Property type	Integer
Usage	Only positive values are valid. The adapter considers any positive entry less than 1 to be equal to 1. Typing a negative value or 0 for this property might result in runtime errors.
Globalized	No
bidi supported	No

Maximum events in polling period (PollQuantity)

This property specifies the number of events that the adapter delivers to the export component during each poll period.

Required	Yes
Default	10
Property type	Integer
Usage	The value must be greater than 0. If this value is increased, more events are processed per polling period, and the adapter might perform less efficiently. If this value is decreased, fewer events are processed per polling period, and the performance of the adapter might improve slightly.
Globalized	No
bidi supported	No

Minimum connections (MinimumConnections)

This property specifies the minimum number of connections that the adapter can use for inbound event delivery.

Required	No
Default	1
Property type	Integer
Usage	Only positive values are valid. Any value less than 1 is treated as 1 by the adapter. Typing a negative value or 0 for this property might result in runtime errors.
Globalized	No
bidi supported	No

No wait (NoWait)

This property specifies whether the adapter waits for a time interval to get an event from the JD Edwards EnterpriseOne transaction server by calling the Dynamic Java Connector API.

Required	No

Possible values	`True` `False`
Default	`True`
Property type	Boolean
Usage	Specifies whether the adapter waits for a time interval to get an event from the JD Edwards EnterpriseOne transaction server by calling the Dynamic Java Connector API.
Example	True
Globalized	No
bidi supported	Yes

Number of times to retry the system connection (RetryLimit)

This property specifies the number of times that the adapter tries to reestablish an inbound connection.

Required	No
Possible values	Positive integers
Default	`0`
Property type	Integer
Usage	Only positive values are valid. When the adapter encounters an error that is related to the inbound connection, this property specifies the number of times that the adapter tries to restart the connection. A value of `0` indicates an infinite number of retries.
Globalized	Yes
bidi supported	No

Retry EIS connection on startup (RetryConnectionOnStartup)

This property controls whether the adapter attempts to connect again to the JD Edwards EnterpriseOne server if it cannot connect at startup.

Required	No
Possible values	`True` `False`
Default	`False`
Property type	Boolean
Usage	This property indicates whether the adapter retries the connection to the JD Edwards EnterpriseOne server if the connection cannot be made when the adapter is started. • Set the property to `False` when you want immediate feedback about whether the adapter can establish a connection to the JD Edwards EnterpriseOne server; for example, when you are building and testing the application that receives events from the adapter. If the adapter cannot connect, the adapter writes log and trace information, and stops. The application status is shown as `Stopped`. After you resolve the connection problem, start the adapter manually. • Set the property to `True` if you do not need immediate feedback about the connection. If the adapter cannot connect during startup, it writes log and trace information, and then attempts to reconnect, by using the `RetryInterval` property to determine how frequently to retry and the value of the `RetryLimit` property to retry multiple times until that value is reached. The application status is shown as `Started`.

Globalized	No
bidi supported	No

Retry interval if connection fails (RetryInterval)

When the adapter encounters an error that is related to the inbound connection, this property specifies the length of time that the adapter waits before trying to establish a new connection.

Required	Yes
Default	`2000`
Unit of measure	Milliseconds
Property type	Integer
Usage	Only positive values are valid. When the adapter encounters an error that is related to the inbound connection, this property specifies the length of time that the adapter waits before trying to establish a new connection.
Globalized	Yes
bidi supported	No

Stop the adapter when an error is encountered while polling (StopPollingOnError)

This property specifies whether the adapter stops polling for events when it encounters an error during polling.

Required	No
Possible values	`True` `False`
Default	`False`
Property type	Boolean
Usage	If this property is set to `True`, the adapter stops polling when it encounters an error. If this property is set to `False`, the adapter logs an exception when it encounters an error during polling, and continues polling.
Globalized	No
bidi supported	No

Wait time (WaitTime)

This property specifies the waiting time if the No Wait property is set to false.

Required	No
Possible values	Any positive integer. Any negative integer is treated as the default value (`3000 milliseconds`)
Default	`3000`
Unit of measure	Milliseconds
Property type	Integer
Usage	This property specifies the waiting time if the No Wait property is set to false.
Example	5000

Globalized	No
bidi supported	Yes

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for JD Edwards EnterpriseOne" on page 2023
With the WebSphere Adapter for JD Edwards EnterpriseOne, you can create integrated processes that include the exchange of information with a JD Edwards EnterpriseOne server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for JD Edwards EnterpriseOne properties" on page 4146
Reference information to refer to when you connect to a JD Edwards EnterpriseOne application.

Outbound adapter properties for JD Edwards:

Outbound adapter properties define how the adapter creates an outbound connection instance with the JD Edwards EnterpriseOne server, and how operations are run on the server. Use the Adapter Connection wizard to set these properties.

The following table describes the outbound adapter properties. A more detailed description of each property is provided in the sections that follow the table.

Property	Description
Adapter ID	The name of the adapter instance in the log and trace files.
"Environment" on page 4166	The JD Edwards EnterpriseOne environment name.

Property	Description
Maximum records for RetrieveAll operation	The maximum number of records to return during a RetrieveAll operation
"Password" on page 4167	The password of the adapter user account on the JD Edwards EnterpriseOne environment.
"Role" on page 4167	The name of the role that is associated with the user name used to access the JD Edwards EnterpriseOne environment.
"User name" on page 4168	The name of the adapter user account on the JD Edwards EnterpriseOne environment.

Adapter ID to use for logging and tracing (AdapterID)

This property identifies a specific deployment, or instance, of the adapter.

Required	Yes
Default	CWYMY_Adapter
Property type	String
Usage	This property identifies the adapter instance in the log and trace files, and also helps identify the adapter instance while monitoring adapters. The adapter ID is used with an adapter-specific identifier, JDERA, to form the component name used by the Log and Trace Analyzer tool. For example, if the adapter ID property is set to 001, the component ID is JDERA001. If you run multiple instances of the same adapter, ensure that the first eight characters of the adapter ID property are unique for each instance so that you can correlate the log and trace information to a particular adapter instance. By making the first seven characters of an adapter ID property unique, the component ID for multiple instances of that adapter is also unique, allowing you to correlate the log and trace information to a particular instance of an adapter. For example, when you set the adapter ID property of two instances of WebSphere Adapter for JD Edwards EnterpriseOne to 001 and 002. The component IDs for those instances, JDERA001 and JDERA002, are short enough to remain unique, enabling you to distinguish them as separate adapter instances. However, instances with longer adapter ID properties cannot be distinguished from each other. If you set the adapter ID properties of two instances to Instance01 and Instance02, you will not be able to examine the log and trace information for each adapter instance because the component ID for both instances is truncated to JDERAInstance. For inbound processing, this property is retrieved from the resource adapter properties. For outbound processing, it is retrieved from the managed connection factory properties.
Globalized	Yes
bidi supported	No

Environment

This property specifies the JD Edwards EnterpriseOne environment name.

Required	Yes
Default	No default value
Property type	String

Usage	A JD Edwards EnterpriseOne environment is a user-defined pointer that indicates the location of data and objects on a JD Edwards EnterpriseOne server. Users can be authorized to use multiple JD Edwards EnterpriseOne environments on a single JD Edwards EnterpriseOne server.
Example	
Globalized	Yes
bidi supported	Yes

Maximum number of records for RetrieveAll operation

This property specifies the maximum number of records to return for a RetrieveAll operation.

Required	Yes
Default	100
Usage	If the number of hits in the database exceeds the value of the `Maximum number of records` property, the adapter returns the errors MatchesExceededLimitException and MatchesExceededLimitFault. Use this property to avoid out-of-memory issues.
Property type	Integer
Globalized	No
bidi supported	No

Password

This property specifies the password of the adapter user account on the JD Edwards EnterpriseOne environment.

Required	Yes
Default	No default value
Property type	String
Usage	Passwords are created and named by the JD Edwards EnterpriseOne administrator. No restrictions exist for the type of characters used, the number of characters used, or the case of the characters used in passwords.
Example	
Globalized	No
bidi supported	Yes

Role

This property specifies the name of the role that is associated with the user name used to access the JD Edwards EnterpriseOne environment.

Required	Yes
Default	No default value
Property type	String

Usage	Roles define what authority users have. Users can have multiple roles. A user's access to applications, forms, table columns, data sources, and so on, is based on one or more roles to which the user is assigned. Roles are created and named by the JD Edwards EnterpriseOne administrator.
Examples	• System administrator • Human resources • Accounting
Globalized	No
bidi supported	Yes

User name

This property specifies the name of the adapter user account on the JD Edwards EnterpriseOne environment.

Required	Yes
Default	No default value
Property type	String
Usage	User names are created by the JD Edwards EnterpriseOne administrator. No restrictions exist for the type of characters used, the number of characters used, or the case of the characters used in user names.
Example	
Globalized	Yes
bidi supported	Yes

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"Overview of WebSphere Adapter for JD Edwards EnterpriseOne" on page 2023
With the WebSphere Adapter for JD Edwards EnterpriseOne, you can create integrated processes that include the exchange of information with a JD Edwards EnterpriseOne server, without special coding.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Connecting to an EIS by using the Adapter Connection wizard" on page 2037
Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection

wizard by adding them to a broker archive (BAR) file.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"WebSphere Adapter for JD Edwards EnterpriseOne properties" on page 4146
Reference information to refer to when you connect to a JD Edwards EnterpriseOne application.

Validation properties

You can control validation by setting properties on the Validate and Parser Options tabs for the nodes that are listed in the following table.

Validation options are available on the following nodes:

Node type	Nodes with validation options
Input node	FileInput, FTEInput, HTTPInput, JMSInput, MQInput, SCAInput, , SOAPInput,TimeoutNotification,
Output node	FileOutput, FTEOutput, HTTPReply, JMSOutput, JMSReply, MQOutput, MQReply, SCAReply, SOAPReply
Other nodes	Compute, CICSRequest, DatabaseRetrieve, HTTPRequest, JavaCompute, Mapping, MQGet, ResetContentDescriptor, SCAAsyncResponse, SCARequest, SOAPRequest, SOAPAsyncResponse, Validate, XSLTransform

For an overview of message validation in the broker, refer to "Validating messages" on page 1478.

You can set the properties that are shown in the following table.

Tab	Properties that affect validation
Validation	Validate, Failure Action
Parser Options	Parse Timing

Validation tab properties:

Validate

Sets whether validation is required. All nodes provide the following options:

None The default value. No validation is performed.

Content
Indicates that you want to perform content checks, such as Content validation and Composition.

Content and Value
Indicates that you want to perform content checks, such as Content validation and Composition, and value checks, such as whether the value conforms to data type, length, range, and enumeration.

Note: Even if **`Content`** is selected, the SOAP and XMLNSC domains always perform **`Content and Value`** validation.

Some nodes also provide the following option:

Inherit

Instructs the node to use all the validation options that are provided with the input message tree in preference to any supplied on the node. Inherit therefore resolves to `None`, `Content`, or `Content And Value`. If Inherit is selected, the other validation properties on the tab are not available.

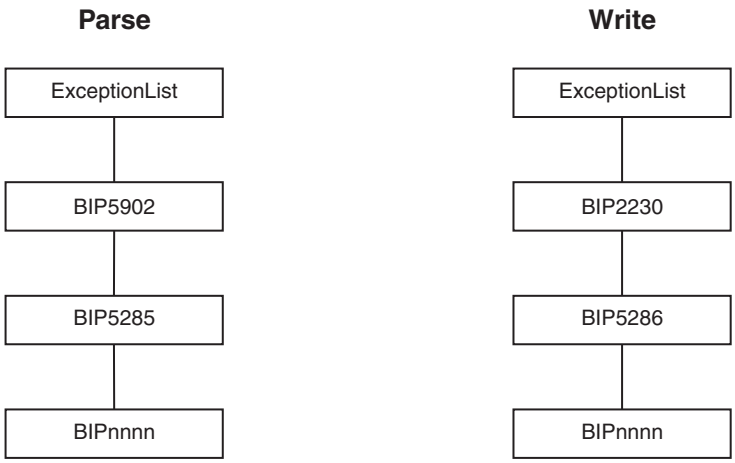
Failure Action

The action that you want to be taken when a validation failure occurs. You can set it to the following values:

Exception

The default value. An exception is thrown on the first validation failure encountered. The resulting exception list is shown below. The failure is also logged in the user trace if you have asked for user tracing of the message flow, and validation stops. Use this setting if you want processing of the message to halt as soon as a failure is encountered.

MRM and IDOC



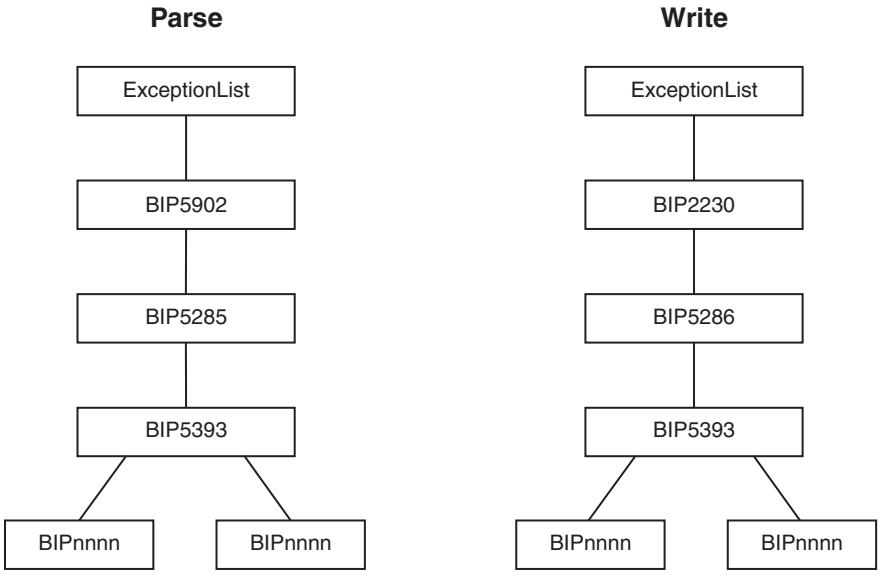
XMLNSC and SOAP

Exception List

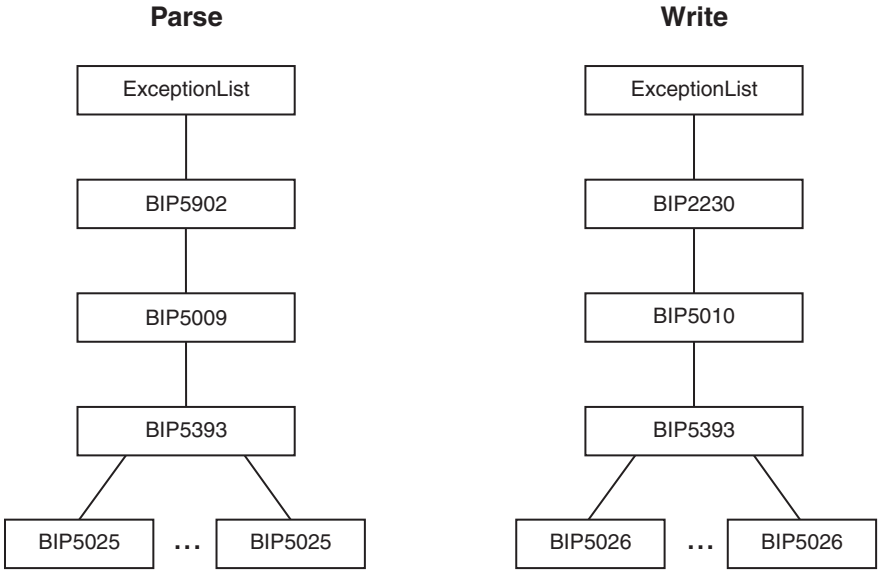
Throws an exception if validation failures are encountered, but only when the current parsing or writing operation has completed.

The resulting exception list is shown below. Each failure is also logged in the user trace if you have asked for user tracing of the message flow, and validation stops. Use this setting if you want processing of the message to halt if a validation failure occurs, but you want to see the full list of failures encountered. This property is affected by the Parse Timing property; when partial parsing is selected the current parsing operation parses only a portion of an input message, so only the validation failures in that portion of the message are reported.

MRM and IDOC



XMLNSC and SOAP



User Trace

Logs all validation failures to the user trace, even if you have not

asked for user tracing of the message flow. Use this setting if you want processing of the message to continue regardless of validation failures.

Local Error Log
Logs all validation failures to the error log (for example, the Event Log on Windows). Use this setting if you want processing of the message to continue regardless of validation failures.

Parser Options tab properties:

Parse Timing

The `Parse Timing` property determines whether on-demand parsing is to be used when parsing a message. It also gives you control over the timing of input message validation:

- If you select a Parse Timing value of `On Demand`, validation of a field in the message is delayed until it is parsed by on-demand parsing.
- If you select a Parse Timing value of `Immediate`, on-demand parsing is overridden, and everything in the message is parsed and validated except, if the message domain is MRM, those complex types with a Composition of Choice or Message that cannot be resolved at the time
- If you select a Parse Timing value of `Complete`, on-demand parsing is overridden, and everything is parsed and validated. If the message domain is MRM, complex types with a Composition of Choice or Message that cannot be resolved at the time cause a validation failure.

If you enable message validation, and you select `On Demand` or `Immediate` for Parse Timing, validation errors might not be detected until later in the processing of a message by a message flow, or might never be detected if a portion of the message is never parsed. To make sure that all fields in a message are validated, either select `Complete` or, if the message domain is MRM, select `Immediate` and make sure that you resolve all unresolved types with a Composition of Choice or Message at the start of your message flow.

The `Parse Timing` property does not affect the validation of output messages.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Parsers" on page 1072
A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from the internal message tree representation.

"MRM parser and domain" on page 1111
You can use the MRM domain to parse and write a wide range of message formats.

"SOAP parser and domain" on page 1082
You can use the SOAP parser to create a common WSDL-based logical tree format for working with Web services, independent of the physical bitstream format.

"XML parsers and domains" on page 1084
You can use XML domains to parse and write messages that conform to the W3C XML standard.

"XMLNSC parser" on page 1090
The XMLNSC parser is a flexible, general-purpose XML parser that offers high performance XML parsing and optional XML Schema validation.

Related tasks:

"Validating messages" on page 1478
The broker provides validation based on the message model for predefined messages.

"Designing a message flow" on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Parsing on demand

On-demand parsing, referred to as partial parsing, is used to parse an input message bit stream only as far as is necessary to satisfy the current reference. The parsers that can perform partial parsing of input messages are the JSON, MRM, XML, XMLNS, and XMLNSC parsers.

An input message can be of any length. To improve performance of message flows, a message is parsed only when necessary to resolve the reference to a particular part of its content. If none of the message content is referenced within the message flow (for example, the entire message is stored in a database by the DataUpdate node, but no manipulation of the message content takes place), the message body is not parsed.

If a parser can parse an input bit stream on demand, instead of immediately parsing the entire bit stream, the Parse Timing property of a message flow node controls the on-demand behavior of the parser.

You can set the Parse Timing property to On Demand (the default), Immediate, or Complete.

On Demand causes partial parsing to occur. When fields in the message are referenced, as much of the message is parsed as is necessary to completely resolve the reference. Therefore, fields might not be parsed until late in the message flow, or never. This restriction applies to both the message body and the message headers.

Immediate and Complete both override partial parsing and parse the entire message, including any message headers, except when the MRM parser encounters an element with a complex type with Composition set to Choice or Message that cannot be resolved at the time; for example, the content must be resolved by the user in ESQL. If Composition is set to Choice, the data is added to the message tree as an unresolved item, and parsing continues with the next element. If

Composition is set to Message, parsing terminates at that point. The only difference in behavior between Immediate and Complete occurs when MRM validation is enabled.

The Parse Timing property also gives you control over how MRM message validation interacts with partial parsing. Refer to “Validation properties” on page 4169 for a full description.

The Parse Timing property has no effect on the serialization of output messages.

Related concepts:

“Parsers” on page 1072
A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from the internal message tree representation.

“Which body parser should you use?” on page 1077
The characteristics of the messages that your applications exchange indicate which body parser you must use.

Related reference:

“Validation properties” on page 4169
You can control validation by setting properties on the Validate and Parser Options tabs for the nodes that are listed in the following table.

Impact analysis: reference

Some artifacts are excluded from secondary analysis.

Artifacts that are excluded from secondary analysis

The following artifacts are excluded from secondary analysis:

- Message set projects
- Message set folders
- Namespaces
- Message set file files
- Message set physical formats
- Message category files
- Adapter files (`.inadapter` and `.outadapter`)
- WebSphere Message Broker SCA definitions files (`.insca` and `.outsca`)
- Message flow projects
- WebSphere Message Broker schemas
- Message flow test files
- Data Design Projects
- DBM files
- Java projects
- Java artifacts
- BAR files

If you analyze the effect of renaming a local element or attribute, and that local element or attribute is used in an ESQL or mapping path, the path is not reported as a secondary impact.

ESQL paths

Only ESQL paths that start with one of the following identifiers are found as secondary impacts:

InputRoot

OutputRoot

Root

InputBody

Body

In other paths, such as paths that start with locally defined reference paths, the root element might not be identifiable (the reference is loosely typed). For example, you might have a complex type `PurchaseOrder` with attribute `address`. The reference path might be `ref.address`. If this path occurs in a routine, it cannot be identified that the attribute `address` is from `PurchaseOrder`, and the path cannot be indexed correctly.

One important result of this restriction is if there are any ESQL references to a message set artifact, and those references are within an ESQL routine that is not an ESQL Module, these references are not shown as secondary impacts. This is because message set artifact references satisfying this criteria cannot begin with one of the previously mentioned identifiers. Schema-scope routines have the advantage of being reusable, but there is a loss of impact analysis awareness associated with using message set artifact references within them.

For performance reasons, ESQL paths are indexed only as far as the first wildcard. For example, consider the following paths:

```
InputRoot.XMLNSC.a.b.*.d
InputRoot.XMLNSC.a.b.*
InputRoot.XMLNSC.a.b.*.e.*.f
```

If impact analysis is performed on artifacts `d`, `e`, or `f`, none of the preceding paths are reported as an impact. If artifacts `a` or `b` are analyzed, all three paths are reported as an impact.

Related concepts:

"Impact analysis: analyzing the effects of planned changes to your applications" on page 1150
When developing an application, you might want to change the names of some artifacts, or move them. Impact analysis reports the artifacts that are likely to be affected by a particular change.

Related tasks:

"Enabling and disabling indexing" on page 1454
Enable indexing to support impact analysis.

"Analyzing planned changes to message model objects" on page 2897
Use impact analysis to analyze the effect of renaming message model objects.

"Analyzing planned changes to ESQL objects" on page 2403
Use impact analysis to analyze the effect of renaming or moving ESQL objects, or moving or changing the broker schema of an ESQL file.

"Analyzing the impact of changes to message maps" on page 2234
Use impact analysis to analyze the effect of renaming or moving message maps.

"Analyzing planned changes to message flows" on page 1436
Use impact analysis to analyze the effect of renaming or moving message flows, including subflows.

"Analyzing planned changes to message set resources" on page 1165
Use impact analysis to analyze the effect of renaming message definition or deployable WSDL files.

Related reference:

"Impact Analysis view" on page 6801
The Impact Analysis view shows information about selected resources, which changes have been marked as complete, and previous results. You can also use this view to copy results to the clipboard.

Supported code pages

Application messages must conform to supported code pages.

The message flows that you create, configure, and deploy to a broker can process and construct application messages in any code page that is listed in the following tables. You can also generate a new code page converter.

This behavior might be affected by the use of other products with WebSphere Message Broker. Check the documentation for other products, including any databases that you use, for further code page support information.

If you experience code page translation problems on HP-UX, check the WebSphere MQ queue manager attribute *CodedCharSetID* (CCSID). The default value for this attribute is 1051. Change this attribute value to 819 for queue managers that host WebSphere Message Broker components.

When you handle UTF-16 data, CCSIDs 1200, 13488 and 17584 are treated differently to others. Traditionally, in ICU usage, the endian encoding of these CSSIDs was platform-specific, and WebSphere Message Broker uses an encoding parameter with these CSSIDs. You can specify the encoding parameter as MQENC_INTEGER_REVERSED to use these CCSIDs to explicitly produce little endian data.

For detailed information about Chinese code page GB18030 support, see "Chinese code page GB18030" on page 4221.

By default, WebSphere Message Broker supports the code pages that are given in the following tables. To find a code page for a specific CCSID, search for an internal converter name in the form `ibm-ccsid`, where *ccsid* is the CCSID for which you are looking.

- Unicode converters
- European and American language converters
- Asian language converters
- Windows US and European converters
- MAC-related converters
- Hebrew, Cyrillic and ECMA language converters
- Indian language converters
- EBCDIC converters

Unicode converters

Internal converter name	Aliases
UTF-8	UTF-8 ibm-1208 ibm-1209 ibm-5304 ibm-5305 ibm-13496 ibm-13497 ibm-17592 ibm-17593 windows-65001 cp1208 Java: *UTF-8
UTF-16	UTF-16 ISO-10646-UCS-2 ibm-1204 ibm-1205 unicode csUnicode ucs-2 Java: *UTF-16
UTF-16BE	UTF-16BE x-utf-16be ibm-1200 ibm-1201 ibm-13488 ibm-13489 ibm-17584 ibm-17585 ibm-21680 ibm-21681 ibm-25776 ibm-25777 ibm-61955 ibm-61956 windows-1201 cp1200 cp1201 UTF16_BigEndian Java: *UTF-16BE

Internal converter name	Aliases
UTF-16LE	UTF-16LE x-utf-16le ibm-1202 ibm-1203 ibm-13490 ibm-13491 ibm-17586 ibm-17587 ibm-21682 ibm-21683 ibm-25778 ibm-25779 UTF16_LittleEndian windows-1200 Java: *UTF-16LE
UTF-32	UTF-32 ISO-10646-UCS-4 ibm-1236 ibm-1237 csUCS4 ucs-4 (No JAVA alias)
UTF-32BE	UTF-32BE UTF32_BigEndian ibm-1232 ibm-1233 (No JAVA alias)
UTF-32LE	UTF-32LE UTF32_LittleEndian ibm-1234 ibm-1235 (No JAVA alias)
UTF16_PlatformEndian	UTF16_PlatformEndian (No JAVA alias)
UTF16_OppositeEndian	UTF16_OppositeEndian (No JAVA alias)
UTF32_PlatformEndian	UTF32_PlatformEndian (No JAVA alias)
UTF32_OppositeEndian	UTF32_OppositeEndian (No JAVA alias)
UTF-7	UTF-7 windows-65000 (No JAVA alias)

Internal converter name	Aliases
IMAP-mailbox-name	IMAP-mailbox-name (No JAVA alias)
SCSU	SCSU ibm-1212 ibm-1213 (No JAVA alias)
BOCU-1	BOCU-1 csBOCU-1 ibm-1214 ibm-1215 (No JAVA alias)
CESU-8	CESU-8 ibm-9400 (No JAVA alias)

European and American language converters

Internal converter name	Aliases
ISO-8859-1	ISO-8859-1 ibm-819 IBM819 cp819 latin1 8859_1 csISOLatin1 iso-ir-100 ISO_8859-1:1987 l1 819 Java: *ISO-8859-1

Internal converter name	Aliases
US-ASCII	US-ASCII ASCII ANSI_X3.4-1968 ANSI_X3.4-1986 ISO_646.irv:1991 iso_646.irv:1983 ISO646-US us csASCII iso-ir-6 cp367 ascii7 646 windows-20127 ibm-367 IBM367 Java: *ASCII
gb18030	gb18030 ibm-1392 windows-54936 (No JAVA alias)
ibm-912_P100-1995	ibm-912_P100-1995 ibm-912 ISO-8859-2 ISO_8859-2:1987 latin2 csISOLatin2 iso-ir-101 l2 8859_2 cp912 912 windows-28592 Java: *ISO-8859-2

Internal converter name	Aliases
ibm-913_P100-2000	ibm-913_P100-2000 ibm-913 ISO-8859-3 ISO_8859-3:1988 latin3 csISOLatin3 iso-ir-109 l3 8859_3 cp913 913 windows-28593 Java: *ISO-8859-3
ibm-914_P100-1995	ibm-914_P100-1995 ibm-914 ISO-8859-4 latin4 csISOLatin4 iso-ir-110 ISO_8859-4:1988 l4 8859_4 cp914 914 windows-28594 Java: *ISO-8859-4
ibm-915_P100-1995	ibm-915_P100-1995 ibm-915 ISO-8859-5 cyrillic csISOLatinCyrillic iso-ir-144 ISO_8859-5:1988 8859_5 cp915 915 windows-28595 Java: *ISO-8859-5

Internal converter name	Aliases
ibm-1089_P100-1995	ibm-1089_P100-1995 ibm-1089 ISO-8859-6 arabic csISOLatinArabic iso-ir-127 ISO_8859-6:1987 ECMA-114 ASMO-708 8859_6 cp1089 1089 windows-28596 ISO-8859-6-I ISO-8859-6-E Java: *ISO-8859-6
ibm-9005_X110-2007	ibm-9005_X110-2007 ibm-9005 ISO-8859-7 greek greek8 ELOT_928 ECMA-118 csISOLatinGreek iso-ir-126 ISO_8859-7:1987 windows-28597 sun_eu_greek (No JAVA alias)
ibm-813_P100-1995	ibm-813_P100-1995 ibm-813 ISO-8859-7 greek greek8 ELOT_928 ECMA-118 csISOLatinGreek iso-ir-126 ISO_8859-7:1987 8859_7 cp813 813 Java: *ISO-8859-7

Internal converter name	Aliases
ibm-5012_P100-1999	ibm-5012_P100-1999 ibm-5012 ISO-8859-8 hebrew csISOLatinHebrew iso-ir-138 ISO_8859-8:1988 ISO-8859-8-I ISO-8859-8-E 8859_8 windows-28598 Java: *ISO-8859-8
ibm-916_P100-1995	ibm-916_P100-1995 ibm-916 cp916 916 Java: *ibm-916
ibm-920_P100-1995	ibm-920_P100-1995 ibm-920 ISO-8859-9 latin5 csISOLatin5 iso-ir-148 ISO_8859-9:1989 l5 8859_9 cp920 920 windows-28599 ECMA-128 Java: *ISO-8859-9
iso-8859_10-1998	iso-8859_10-1998 ISO-8859-10 iso-ir-157 l6 ISO_8859-10:1992 csISOLatin6 latin6 (No JAVA alias)
iso-8859_11-2001	iso-8859_11-2001 ISO-8859-11 thai8 (No JAVA alias)

Internal converter name	Aliases
ibm-921_P100-1995	ibm-921_P100-1995 ibm-921 ISO-8859-13 8859_13 windows-28603 cp921 921 Java: *ISO-8859-13
iso-8859_14-1998	iso-8859_14-1998 ISO-8859-14 iso-ir-199 ISO_8859-14:1998 latin8 iso-celtic l8 (No JAVA alias)
ibm-923_P100-1998	ibm-923_P100-1998 ibm-923 ISO-8859-15 Latin-9 l9 8859_15 latin0 csisolatin0 csisolatin9 iso8859_15_fdis cp923 923 windows-28605 Java: *ISO-8859-15

Asian language converters

Internal converter name	Aliases
ibm-942_P12A-1999	ibm-942_P12A-1999 ibm-942 ibm-932 cp932 shift_jis78 sjis78 ibm-942_VSUB_VPUA ibm-932_VSUB_VPUA (No JAVA alias)

Internal converter name	Aliases
ibm-943_P15A-2003	ibm-943_P15A-2003 ibm-943 Shift_JIS MS_Kanji csShiftJIS windows-31j csWindows31J x-sjis x-ms-cp932 cp932 windows-932 cp943c IBM-943C ms932 pck sjis ibm-943_VSUB_VPUA Java: *cp943c
ibm-943_P130-1999	ibm-943_P130-1999 ibm-943 Shift_JIS cp943 943 ibm-943_VASCII_VSUB_VPUA Java: *cp943
ibm-33722_P12A_P12A-2004_U2	ibm-33722_P12A_P12A-2004_U2 ibm-33722 ibm-5050 EUC-JP Extended_UNIX_Code_Packed_Format_for_Japanese csEUCPkdFmtJapanese X-EUC-JP windows-51932 ibm-33722_VPUA IBM-eucJP (No JAVA alias)
ibm-33722_P120-1999	ibm-33722_P120-1999 ibm-33722 ibm-5050 cp33722 33722 ibm-33722_VASCII_VPUA Java: *cp33722

Internal converter name	Aliases
ibm-954_P101-2007	ibm-954_P101-2007 ibm-954 EUC-JP Extended_UNIX_Code_Packed_Format_for_Japanese csEUCPkdFmtJapanese X-EUC-JP eucjis ujis Java: *EUC-JP
ibm-1373_P100-2002	ibm-1373_P100-2002 ibm-1373 windows-950 (No JAVA alias)
windows-950-2000	windows-950-2000 Big5 csBig5 windows-950 x-big5 Java: *Big5
ibm-950_P110-1999	ibm-950_P110-1999 ibm-950 cp950 950 Java: *cp950
ibm-1375_P100-2007	ibm-1375_P100-2007 ibm-1375 Big5-HKSCS big5hk HKSCS-BIG5 Java: *Big5-HKSCS
ibm-5471_P100-2006	ibm-5471_P100-2006 ibm-5471 Big5-HKSCS MS950_HKSCS hkbig5 big5-hkscs:unicode3.0 Java: *MS950_HKSCS

Internal converter name	Aliases
ibm-1386_P100-2001	ibm-1386_P100-2001 ibm-1386 cp1386 windows-936 ibm-1386_VSUB_VPUA ibm-1381 cp1381 1381 Java: *cp1381
windows-936-2000	windows-936-2000 GBK CP936 MS936 windows-936 Java: *GBK
ibm-1383_P110-1999	ibm-1383_P110-1999 ibm-1383 GB2312 csGB2312 cp1383 1383 EUC-CN ibm-eucCN hp15CN ibm-1383_VPUA Java: *cp1383
ibm-5478_P100-1995	ibm-5478_P100-1995 ibm-5478 GB_2312-80 chinese iso-ir-58 csISO58GB231280 gb2312-1980 GB2312.1980-0 (No JAVA alias)
ibm-964_P110-1999	ibm-964_P110-1999 ibm-964 EUC-TW ibm-eucTW cns11643 cp964 964 ibm-964_VPUA Java: *cp964

Internal converter name	Aliases
ibm-949_P110-1999	ibm-949_P110-1999 ibm-949 cp949 949 ibm-949_VASCII_VSUB_VPUA Java: *cp949
ibm-949_P11A-1999	ibm-949_P11A-1999 ibm-949 cp949c ibm-949_VSUB_VPUA Java: *cp949c
ibm-970_P110_P110-2006_U2	ibm-970_P110_P110-2006_U2 ibm-970 EUC-KR KS_C_5601-1987 windows-51949 csEUCKR ibm-eucKR KSC_5601 5601 cp970 970 ibm-970_VPUA Java: *cp970
ibm-971_P100-1995	ibm-971_P100-1995 ibm-971 ibm-971_VPUA (No JAVA alias)
ibm-1363_P11B-1998	ibm-1363_P11B-1998 ibm-1363 KS_C_5601-1987 KS_C_5601-1989 KSC_5601 csKSC56011987 korean iso-ir-149 5601 cp1363 ksc windows-949 ibm-1363_VSUB_VPUA (No JAVA alias)

Internal converter name	Aliases
ibm-1363_P110-1997	ibm-1363_P110-1997 ibm-1363 ibm-1363_VASCII_VSUB_VPUA (No JAVA alias)
windows-949-2000	windows-949-2000 windows-949 KS_C_5601-1987 KS_C_5601-1989 KSC_5601 csKSC56011987 korean iso-ir-149 ms949 Java: *windows-949
windows-874-2000	windows-874-2000 TIS-620 windows-874 MS874 Java: *windows-874
ibm-874_P100-1995	ibm-874_P100-1995 ibm-874 ibm-9066 cp874 TIS-620 tis620.2533 eucTH Java: *cp874
ibm-1162_P100-1999	ibm-1162_P100-1999 ibm-1162 (No JAVA alias)

Windows US and European converters

Internal converter name	Aliases
ibm-437_P100-1995	ibm-437_P100-1995 ibm-437 IBM437 cp437 437 csPC8CodePage437 windows-437 Java: *cp437

Internal converter name	Aliases
ibm-720_P100-1997	ibm-720_P100-1997 ibm-720 windows-720 DOS-720 (No JAVA alias)
ibm-737_P100-1997	ibm-737_P100-1997 ibm-737 IBM737 cp737 windows-737 737 Java: *cp737
ibm-775_P100-1996	ibm-775_P100-1996 ibm-775 IBM775 cp775 csPC775Baltic windows-775 775 Java: *cp775
ibm-850_P100-1995	ibm-850_P100-1995 ibm-850 IBM850 cp850 850 csPC850Multilingual windows-850 Java: *cp850
ibm-851_P100-1995	ibm-851_P100-1995 ibm-851 IBM851 cp851 851 csPC851 (No JAVA alias)
ibm-852_P100-1995	ibm-852_P100-1995 ibm-852 IBM852 cp852 852 csPCp852 windows-852 Java: *cp852

Internal converter name	Aliases
ibm-855_P100-1995	ibm-855_P100-1995 ibm-855 IBM855 cp855 855 csIBM855 csPCp855 windows-855 Java: *cp855
ibm-856_P100-1995	ibm-856_P100-1995 ibm-856 IBM856 cp856 856 Java: *cp856
ibm-857_P100-1995	ibm-857_P100-1995 ibm-857 IBM857 cp857 857 csIBM857 windows-857 Java: *cp857
ibm-858_P100-1997	ibm-858_P100-1997 ibm-858 IBM00858 CCSID00858 CP00858 PC-Multilingual-850+euro cp858 windows-858 Java: *cp858
ibm-860_P100-1995	ibm-860_P100-1995 ibm-860 IBM860 cp860 860 csIBM860 Java: *cp860

Internal converter name	Aliases
ibm-861_P100-1995	ibm-861_P100-1995 ibm-861 IBM861 cp861 861 cp-is csIBM861 windows-861 Java: *cp861
ibm-862_P100-1995	ibm-862_P100-1995 ibm-862 IBM862 cp862 862 csPC862LatinHebrew DOS-862 windows-862 Java: *cp862
ibm-863_P100-1995	ibm-863_P100-1995 ibm-863 IBM863 cp863 863 csIBM863 Java: *cp863
ibm-864_X110-1999	ibm-864_X110-1999 ibm-864 IBM864 cp864 csIBM864 Java: *cp864
ibm-865_P100-1995	ibm-865_P100-1995 ibm-865 IBM865 cp865 865 csIBM865 Java: *cp865

Internal converter name	Aliases
ibm-866_P100-1995	ibm-866_P100-1995 ibm-866 IBM866 cp866 866 csIBM866 windows-866 Java: *cp866
ibm-867_P100-1998	ibm-867_P100-1998 ibm-867 (No JAVA alias)
ibm-868_P100-1995	ibm-868_P100-1995 ibm-868 IBM868 CP868 868 csIBM868 cp-ar Java: *CP868
ibm-869_P100-1995	ibm-869_P100-1995 ibm-869 IBM869 cp869 869 cp-gr csIBM869 windows-869 Java: *cp869
ibm-878_P100-1996	ibm-878_P100-1996 ibm-878 KOI8-R koi8 csKOI8R windows-20866 cp878 Java: *KOI8-R
ibm-901_P100-1999	ibm-901_P100-1999 ibm-901 (No JAVA alias)
ibm-902_P100-1999	ibm-902_P100-1999 ibm-902 (No JAVA alias)

Internal converter name	Aliases
ibm-922_P100-1999	ibm-922_P100-1999 ibm-922 IBM922 cp922 922 Java: *cp922
ibm-1168_P100-2002	ibm-1168_P100-2002 ibm-1168 KOI8-U windows-21866 (No JAVA alias)
ibm-4909_P100-1999	ibm-4909_P100-1999 ibm-4909 (No JAVA alias)
ibm-5346_P100-1998	ibm-5346_P100-1998 ibm-5346 windows-1250 cp1250 Java: *windows-1250
ibm-5347_P100-1998	ibm-5347_P100-1998 ibm-5347 windows-1251 cp1251 ANSI1251 Java: *windows-1251
ibm-5348_P100-1997	ibm-5348_P100-1997 ibm-5348 windows-1252 cp1252 Java: *windows-1252
ibm-5349_P100-1998	ibm-5349_P100-1998 ibm-5349 windows-1253 cp1253 Java: *windows-1253
ibm-5350_P100-1998	ibm-5350_P100-1998 ibm-5350 windows-1254 cp1254 Java: *windows-1254

Internal converter name	Aliases
ibm-9447_P100-2002	ibm-9447_P100-2002 ibm-9447 windows-1255 cp1255 Java: *windows-1255
ibm-9448_X100-2005	ibm-9448_X100-2005 ibm-9448 windows-1256 cp1256 Java: *windows-1256
ibm-9449_P100-2002	ibm-9449_P100-2002 ibm-9449 windows-1257 cp1257 Java: *windows-1257
ibm-5354_P100-1998	ibm-5354_P100-1998 ibm-5354 windows-1258 cp1258 Java: *windows-1258
ibm-1250_P100-1995	ibm-1250_P100-1995 ibm-1250 windows-1250 (No JAVA alias)
ibm-1251_P100-1995	ibm-1251_P100-1995 ibm-1251 windows-1251 (No JAVA alias)
ibm-1252_P100-2000	ibm-1252_P100-2000 ibm-1252 windows-1252 (No JAVA alias)
ibm-1253_P100-1995	ibm-1253_P100-1995 ibm-1253 windows-1253 (No JAVA alias)
ibm-1254_P100-1995	ibm-1254_P100-1995 ibm-1254 windows-1254 (No JAVA alias)
ibm-1255_P100-1995	ibm-1255_P100-1995 ibm-1255 (No JAVA alias)

Internal converter name	Aliases
ibm-5351_P100-1998	ibm-5351_P100-1998 ibm-5351 windows-1255 (No JAVA alias)
ibm-1256_P110-1997	ibm-1256_P110-1997 ibm-1256 (No JAVA alias)
ibm-5352_P100-1998	ibm-5352_P100-1998 ibm-5352 windows-1256 (No JAVA alias)
ibm-1257_P100-1995	ibm-1257_P100-1995 ibm-1257 (No JAVA alias)
ibm-5353_P100-1998	ibm-5353_P100-1998 ibm-5353 windows-1257 (No JAVA alias)
ibm-1258_P100-1997	ibm-1258_P100-1997 ibm-1258 windows-1258 (No JAVA alias)

MAC-related converters

Internal converter name	Aliases
macos-0_2-10.2	macos-0_2-10.2 macintosh mac csMacintosh windows-10000 (No JAVA alias)
macos-6_2-10.4	macos-6_2-10.4 x-mac-greek windows-10006 macgr (No JAVA alias)
macos-7_3-10.2	macos-7_3-10.2 x-mac-cyrillic windows-10007 mac-cyrillic maccy (No JAVA alias)

Internal converter name	Aliases
macos-29-10.2	macos-29-10.2 x-mac-centraleurroman windows-10029 x-mac-ce macce (No JAVA alias)
macos-35-10.2	macos-35-10.2 x-mac-turkish windows-10081 mactr (No JAVA alias)
ibm-1051_P100-1995	ibm-1051_P100-1995 ibm-1051 hp-roman8 roman8 r8 csHPRoman8 (No JAVA alias)
ibm-1276_P100-1995	ibm-1276_P100-1995 ibm-1276 Adobe-Standard-Encoding csAdobeStandardEncoding (No JAVA alias)

Hebrew, Cyrillic, and ECMA language converters

Internal converter name	Aliases
ibm-1006_P100-1995	ibm-1006_P100-1995 ibm-1006 IBM1006 cp1006 1006 Java: *cp1006
ibm-1098_P100-1995	ibm-1098_P100-1995 ibm-1098 IBM1098 cp1098 1098 Java: *cp1098
ibm-1124_P100-1996	ibm-1124_P100-1996 ibm-1124 cp1124 1124 Java: *cp1124

Internal converter name	Aliases
ibm-1125_P100-1997	ibm-1125_P100-1997 ibm-1125 cp1125 (No JAVA alias)
ibm-1129_P100-1997	ibm-1129_P100-1997 ibm-1129 (No JAVA alias)
ibm-1131_P100-1997	ibm-1131_P100-1997 ibm-1131 cp1131 (No JAVA alias)
ibm-1133_P100-1997	ibm-1133_P100-1997 ibm-1133 (No JAVA alias)
ISO_2022,locale=ja,version=0	ISO_2022,locale=ja,version=0 ISO-2022-JP csISO2022JP Java: *ISO-2022-JP
ISO_2022,locale=ja,version=1	ISO_2022,locale=ja,version=1 ISO-2022-JP-1 JIS_Encoding csJISEncoding ibm-5054 JIS (No JAVA alias)
ISO_2022,locale=ja,version=2	ISO_2022,locale=ja,version=2 ISO-2022-JP-2 csISO2022JP2 (No JAVA alias)
ISO_2022,locale=ja,version=3	ISO_2022,locale=ja,version=3 JIS7 (No JAVA alias)
ISO_2022,locale=ja,version=4	ISO_2022,locale=ja,version=4 JIS8 (No JAVA alias)
ISO_2022,locale=ko,version=0	ISO_2022,locale=ko,version=0 ISO-2022-KR csISO2022KR Java: *ISO-2022-KR
ISO_2022,locale=ko,version=1	ISO_2022,locale=ko,version=1 ibm-25546 (No JAVA alias)

Internal converter name	Aliases
ISO_2022,locale=zh,version=0	ISO_2022,locale=zh,version=0 ISO-2022-CN csISO2022CN Java: *ISO-2022-CN
ISO_2022,locale=zh,version=1	ISO_2022,locale=zh,version=1 ISO-2022-CN-EXT (No JAVA alias)
HZ	HZ HZ-GB-2312 (No JAVA alias)
ibm-897_P100-1995	ibm-897_P100-1995 ibm-897 JIS_X0201 X0201 csHalfWidthKatakana (No JAVA alias)

Indian language converters

Internal converter name	Aliases
ISCII,version=0	ISCII,version=0 x-iscii-de windows-57002 iscii-dev ibm-4902 (No JAVA alias)
ISCII,version=1	ISCII,version=1 x-iscii-be windows-57003 iscii-bng windows-57006 x-iscii-as (No JAVA alias)
ISCII,version=2	ISCII,version=2 x-iscii-pa windows-57011 iscii-gur (No JAVA alias)
ISCII,version=3	ISCII,version=3 x-iscii-gu windows-57010 iscii-guj (No JAVA alias)

Internal converter name	Aliases
ISCII,version=4	ISCII,version=4 x-iscii-or windows-57007 iscii-ori (No JAVA alias)
ISCII,version=5	ISCII,version=5 x-iscii-ta windows-57004 iscii-tml (No JAVA alias)
ISCII,version=6	ISCII,version=6 x-iscii-te windows-57005 iscii-tlg (No JAVA alias)
ISCII,version=7	ISCII,version=7 x-iscii-ka windows-57008 iscii-knd (No JAVA alias)
ISCII,version=8	ISCII,version=8 x-iscii-ma windows-57009 iscii-mlm (No JAVA alias)

EBCDIC converters

Internal converter name	Aliases
LMBCS-1	LMBCS-1 lmbcs ibm-65025 (No JAVA alias)

Internal converter name	Aliases
ibm-37_P100-1995	ibm-37_P100-1995 ibm-37 IBM037 ibm-037 ebcdic-cp-us ebcdic-cp-ca ebcdic-cp-wt ebcdic-cp-nl csIBM037 cp037 037 cpibm37 cp37 Java: *cp037
ibm-273_P100-1995	ibm-273_P100-1995 ibm-273 IBM273 CP273 csIBM273 ebcdic-de 273 Java: *CP273
ibm-277_P100-1995	ibm-277_P100-1995 ibm-277 IBM277 cp277 EBCDIC-CP-DK EBCDIC-CP-NO csIBM277 ebcdic-dk 277 Java: *cp277
ibm-278_P100-1995	ibm-278_P100-1995 ibm-278 IBM278 cp278 ebcdic-cp-fi ebcdic-cp-se csIBM278 ebcdic-sv 278 Java: *cp278

Internal converter name	Aliases
ibm-280_P100-1995	ibm-280_P100-1995 ibm-280 IBM280 CP280 ebcdic-cp-it csIBM280 280 Java: *CP280
ibm-284_P100-1995	ibm-284_P100-1995 ibm-284 IBM284 CP284 ebcdic-cp-es csIBM284 cpibm284 284 Java: *CP284
ibm-285_P100-1995	ibm-285_P100-1995 ibm-285 IBM285 CP285 ebcdic-cp-gb csIBM285 cpibm285 ebcdic-gb 285 Java: *CP285
ibm-290_P100-1995	ibm-290_P100-1995 ibm-290 IBM290 cp290 EBCDIC-JP-kana csIBM290 (No JAVA alias)
ibm-297_P100-1995	ibm-297_P100-1995 ibm-297 IBM297 cp297 ebcdic-cp-fr csIBM297 cpibm297 297 Java: *cp297

Internal converter name	Aliases
ibm-420_X120-1999	ibm-420_X120-1999 ibm-420 IBM420 cp420 ebcdic-cp-ar1 csIBM420 420 Java: *cp420
ibm-424_P100-1995	ibm-424_P100-1995 ibm-424 IBM424 cp424 ebcdic-cp-he csIBM424 424 Java: *cp424
ibm-500_P100-1995	ibm-500_P100-1995 ibm-500 IBM500 CP500 ebcdic-cp-be csIBM500 ebcdic-cp-ch 500 Java: *CP500
ibm-803_P100-1999	ibm-803_P100-1999 ibm-803 cp803 (No JAVA alias)
ibm-838_P100-1995	ibm-838_P100-1995 ibm-838 IBM838 IBM-Thai csIBMThai cp838 838 ibm-9030 Java: *cp838

Internal converter name	Aliases
ibm-870_P100-1995	ibm-870_P100-1995 ibm-870 IBM870 CP870 ebcdic-cp-roece ebcdic-cp-yu csIBM870 Java: *CP870
ibm-871_P100-1995	ibm-871_P100-1995 ibm-871 IBM871 ebcdic-cp-is csIBM871 CP871 ebcdic-is 871 Java: *CP871
ibm-875_P100-1995	ibm-875_P100-1995 ibm-875 IBM875 cp875 875 Java: *cp875
ibm-918_P100-1995	ibm-918_P100-1995 ibm-918 IBM918 CP918 ebcdic-cp-ar2 csIBM918 Java: *CP918
ibm-930_P120-1999	ibm-930_P120-1999 ibm-930 ibm-5026 IBM930 cp930 930 Java: *cp930
ibm-933_P110-1995	ibm-933_P110-1995 ibm-933 cp933 933 Java: *cp933

Internal converter name	Aliases
ibm-935_P110-1999	ibm-935_P110-1999 ibm-935 cp935 935 Java: *cp935
ibm-937_P110-1999	ibm-937_P110-1999 ibm-937 cp937 937 Java: *cp937
ibm-939_P120-1999	ibm-939_P120-1999 ibm-939 ibm-931 ibm-5035 IBM939 cp939 939 Java: *cp939
ibm-1025_P100-1995	ibm-1025_P100-1995 ibm-1025 cp1025 1025 Java: *cp1025
ibm-1026_P100-1995	ibm-1026_P100-1995 ibm-1026 IBM1026 CP1026 csIBM1026 1026 Java: *CP1026
ibm-1047_P100-1995	ibm-1047_P100-1995 ibm-1047 IBM1047 cp1047 1047 Java: *cp1047
ibm-1097_P100-1995	ibm-1097_P100-1995 ibm-1097 cp1097 1097 Java: *cp1097

Internal converter name	Aliases
ibm-1112_P100-1995	ibm-1112_P100-1995 ibm-1112 cp1112 1112 Java: *cp1112
ibm-1122_P100-1999	ibm-1122_P100-1999 ibm-1122 cp1122 1122 Java: *cp1122
ibm-1123_P100-1995	ibm-1123_P100-1995 ibm-1123 cp1123 1123 Java: *cp1123
ibm-1130_P100-1997	ibm-1130_P100-1997 ibm-1130 (No JAVA alias)
ibm-1132_P100-1998	ibm-1132_P100-1998 ibm-1132 (No JAVA alias)
ibm-1137_P100-1999	ibm-1137_P100-1999 ibm-1137 (No JAVA alias)
ibm-4517_P100-2005	ibm-4517_P100-2005 ibm-4517 (No JAVA alias)
ibm-1140_P100-1997	ibm-1140_P100-1997 ibm-1140 IBM01140 CCSID01140 CP01140 cp1140 ebcdic-us-37+euro Java: *cp1140
ibm-1141_P100-1997	ibm-1141_P100-1997 ibm-1141 IBM01141 CCSID01141 CP01141 cp1141 ebcdic-de-273+euro Java: *cp1141

Internal converter name	Aliases
ibm-1142_P100-1997	ibm-1142_P100-1997 ibm-1142 IBM01142 CCSID01142 CP01142 cp1142 ebcdic-dk-277+euro ebcdic-no-277+euro Java: *cp1142
ibm-1143_P100-1997	ibm-1143_P100-1997 ibm-1143 IBM01143 CCSID01143 CP01143 cp1143 ebcdic-fi-278+euro ebcdic-se-278+euro Java: *cp1143
ibm-1144_P100-1997	ibm-1144_P100-1997 ibm-1144 IBM01144 CCSID01144 CP01144 cp1144 ebcdic-it-280+euro Java: *cp1144
ibm-1145_P100-1997	ibm-1145_P100-1997 ibm-1145 IBM01145 CCSID01145 CP01145 cp1145 ebcdic-es-284+euro Java: *cp1145
ibm-1146_P100-1997	ibm-1146_P100-1997 ibm-1146 IBM01146 CCSID01146 CP01146 cp1146 ebcdic-gb-285+euro Java: *cp1146

Internal converter name	Aliases
ibm-1147_P100-1997	ibm-1147_P100-1997 ibm-1147 IBM01147 CCSID01147 CP01147 cp1147 ebcdic-fr-297+euro Java: *cp1147
ibm-1148_P100-1997	ibm-1148_P100-1997 ibm-1148 IBM01148 CCSID01148 CP01148 cp1148 ebcdic-international-500+euro Java: *cp1148
ibm-1149_P100-1997	ibm-1149_P100-1997 ibm-1149 IBM01149 CCSID01149 CP01149 cp1149 ebcdic-is-871+euro Java: *cp1149
ibm-1153_P100-1999	ibm-1153_P100-1999 ibm-1153 (No JAVA alias)
ibm-1154_P100-1999	ibm-1154_P100-1999 ibm-1154 (No JAVA alias)
ibm-1155_P100-1999	ibm-1155_P100-1999 ibm-1155 (No JAVA alias)
ibm-1156_P100-1999	ibm-1156_P100-1999 ibm-1156 (No JAVA alias)
ibm-1157_P100-1999	ibm-1157_P100-1999 ibm-1157 (No JAVA alias)
ibm-1158_P100-1999	ibm-1158_P100-1999 ibm-1158 (No JAVA alias)

Internal converter name	Aliases
ibm-1160_P100-1999	ibm-1160_P100-1999 ibm-1160 (No JAVA alias)
ibm-1164_P100-1999	ibm-1164_P100-1999 ibm-1164 (No JAVA alias)
ibm-1364_P110-2007	ibm-1364_P110-2007 ibm-1364 (No JAVA alias)
ibm-1371_P100-1999	ibm-1371_P100-1999 ibm-1371 (No JAVA alias)
ibm-1388_P103-2001	ibm-1388_P103-2001 ibm-1388 ibm-9580 (No JAVA alias)
ibm-1390_P110-2003	ibm-1390_P110-2003 ibm-1390 (No JAVA alias)
ibm-1399_P110-2003	ibm-1399_P110-2003 ibm-1399 (No JAVA alias)
ibm-5123_P100-1999	ibm-5123_P100-1999 ibm-5123 (No JAVA alias)
ibm-8482_P100-1999	ibm-8482_P100-1999 ibm-8482 (No JAVA alias)
ibm-16684_P110-2003	ibm-16684_P110-2003 ibm-16684 ibm-20780 (No JAVA alias)
ibm-4899_P100-1998	ibm-4899_P100-1998 ibm-4899 (No JAVA alias)
ibm-4971_P100-1999	ibm-4971_P100-1999 ibm-4971 (No JAVA alias)
ibm-9067_X100-2005	ibm-9067_X100-2005 ibm-9067 (No JAVA alias)

Internal converter name	Aliases
ibm-12712_P100-1998	ibm-12712_P100-1998 ibm-12712 ebcdic-he (No JAVA alias)
ibm-16804_X110-1999	ibm-16804_X110-1999 ibm-16804 ebcdic-ar (No JAVA alias)
ibm-37_P100-1995,swaplfnl	ibm-37_P100-1995,swaplfnl ibm-37-s390 (No JAVA alias)
ibm-1047_P100-1995,swaplfnl	ibm-1047_P100-1995,swaplfnl ibm-1047-s390 (No JAVA alias)
ibm-1140_P100-1997,swaplfnl	ibm-1140_P100-1997,swaplfnl ibm-1140-s390 (No JAVA alias)
ibm-1142_P100-1997,swaplfnl	ibm-1142_P100-1997,swaplfnl ibm-1142-s390 (No JAVA alias)
ibm-1143_P100-1997,swaplfnl	ibm-1143_P100-1997,swaplfnl ibm-1143-s390 (No JAVA alias)
ibm-1144_P100-1997,swaplfnl	ibm-1144_P100-1997,swaplfnl ibm-1144-s390 (No JAVA alias)
ibm-1145_P100-1997,swaplfnl	ibm-1145_P100-1997,swaplfnl ibm-1145-s390 (No JAVA alias)
ibm-1146_P100-1997,swaplfnl	ibm-1146_P100-1997,swaplfnl ibm-1146-s390 (No JAVA alias)
ibm-1147_P100-1997,swaplfnl	ibm-1147_P100-1997,swaplfnl ibm-1147-s390 (No JAVA alias)
ibm-1148_P100-1997,swaplfnl	ibm-1148_P100-1997,swaplfnl ibm-1148-s390 (No JAVA alias)
ibm-1149_P100-1997,swaplfnl	ibm-1149_P100-1997,swaplfnl ibm-1149-s390 (No JAVA alias)

Internal converter name	Aliases
ibm-1153_P100-1999,swaplfnl	ibm-1153_P100-1999,swaplfnl ibm-1153-s390 (No JAVA alias)
ibm-12712_P100-1998,swaplfnl	ibm-12712_P100-1998,swaplfnl ibm-12712-s390 (No JAVA alias)
ibm-16804_X110-1999,swaplfnl	ibm-16804_X110-1999,swaplfnl ibm-16804-s390 (No JAVA alias)
ebcdic-xml-us	ebcdic-xml-us (No JAVA alias)
gsm-03.38-2000	gsm-03.38-2000 GSM0338 (No JAVA alias)
ibm-1004_P100-1995	ibm-1004_P100-1995 ibm-1004 (No JAVA alias)
ibm-1008_P100-1995	ibm-1008_P100-1995 ibm-1008 (No JAVA alias)
ibm-1009_P100-1995	ibm-1009_P100-1995 ibm-1009 (No JAVA alias)
ibm-1010_P100-1995	ibm-1010_P100-1995 ibm-1010 NF_Z_62-010 iso-ir-69 ISO646-FR fr csISO69French (No JAVA alias)
ibm-1011_P100-1995	ibm-1011_P100-1995 ibm-1011 DIN_66003 iso-ir-21 de ISO646-DE csISO21German (No JAVA alias)

Internal converter name	Aliases
ibm-1012_P100-1995	ibm-1012_P100-1995 ibm-1012 IT iso-ir-15 ISO646-IT csISO15Italian (No JAVA alias)
ibm-1013_P100-1995	ibm-1013_P100-1995 ibm-1013 BS_4730 iso-ir-4 ISO646-GB gb uk csISO4UnitedKingdom (No JAVA alias)
ibm-1014_P100-1995	ibm-1014_P100-1995 ibm-1014 ES2 iso-ir-85 ISO646-ES2 csISO85Spanish2 (No JAVA alias)
ibm-1015_P100-1995	ibm-1015_P100-1995 ibm-1015 PT2 iso-ir-84 ISO646-PT2 csISO84Portuguese2 (No JAVA alias)
ibm-1016_P100-1995	ibm-1016_P100-1995 ibm-1016 NS_4551-1 iso-ir-60 ISO646-NO no csISO60DanishNorwegian csISO60Norwegian1 (No JAVA alias)
ibm-1017_P100-1995	ibm-1017_P100-1995 ibm-1017 (No JAVA alias)

Internal converter name	Aliases
ibm-1018_P100-1995	ibm-1018_P100-1995 ibm-1018 SEN_850200_B iso-ir-10 FI ISO646-FI ISO646-SE se csISO10Swedish (No JAVA alias)
ibm-1019_P100-1995	ibm-1019_P100-1995 ibm-1019 (No JAVA alias)
ibm-1020_P100-2003	ibm-1020_P100-2003 ibm-1020 CSA_Z243.4-1985-1 iso-ir-121 ISO646-CA csa7-1 ca csISO121Canadian1 (No JAVA alias)
ibm-1021_P100-2003	ibm-1021_P100-2003 ibm-1021 (No JAVA alias)
ibm-1023_P100-2003	ibm-1023_P100-2003 ibm-1023 ES iso-ir-17 ISO646-ES csISO17Spanish (No JAVA alias)
ibm-1046_X110-1999	ibm-1046_X110-1999 ibm-1046 (No JAVA alias)
ibm-1100_P100-2003	ibm-1100_P100-2003 ibm-1100 DEC-MCS dec csDECMCS (No JAVA alias)
ibm-1101_P100-2003	ibm-1101_P100-2003 ibm-1101 (No JAVA alias)

Internal converter name	Aliases
ibm-1102_P100-2003	ibm-1102_P100-2003 ibm-1102 (No JAVA alias)
ibm-1103_P100-2003	ibm-1103_P100-2003 ibm-1103 (No JAVA alias)
ibm-1104_P100-2003	ibm-1104_P100-2003 ibm-1104 NF_Z_62-010_1973 iso-ir-25 ISO646-FR1 csISO25French (No JAVA alias)
ibm-1105_P100-2003	ibm-1105_P100-2003 ibm-1105 (No JAVA alias)
ibm-1106_P100-2003	ibm-1106_P100-2003 ibm-1106 (No JAVA alias)
ibm-1107_P100-2003	ibm-1107_P100-2003 ibm-1107 DS_2089 ISO646-DK dk csISO646Danish (No JAVA alias)
ibm-1127_P100-2004	ibm-1127_P100-2004 ibm-1127 (No JAVA alias)
ibm-1161_P100-1999	ibm-1161_P100-1999 ibm-1161 (No JAVA alias)
ibm-1163_P100-1999	ibm-1163_P100-1999 ibm-1163 (No JAVA alias)
ibm-1165_P101-2000	ibm-1165_P101-2000 ibm-1165 (No JAVA alias)
ibm-1166_P100-2002	ibm-1166_P100-2002 ibm-1166 (No JAVA alias)

Internal converter name	Aliases
ibm-1167_P100-2002	ibm-1167_P100-2002 ibm-1167 KOI8-RU (No JAVA alias)
ibm-1174_X100-2007	ibm-1174_X100-2007 ibm-1174 KZ-1048 STRK1048-2002 RK1048 csKZ1048 (No JAVA alias)
ibm-1277_P100-1995	ibm-1277_P100-1995 ibm-1277 (No JAVA alias)
ibm-13125_P100-1997	ibm-13125_P100-1997 ibm-13125 (No JAVA alias)
ibm-13140_P101-2000	ibm-13140_P101-2000 ibm-13140 (No JAVA alias)
ibm-13218_P100-1996	ibm-13218_P100-1996 ibm-13218 (No JAVA alias)
ibm-1350_P110-1997	ibm-1350_P110-1997 ibm-1350 (No JAVA alias)
ibm-1351_P110-1997	ibm-1351_P110-1997 ibm-1351 (No JAVA alias)
ibm-1362_P110-1999	ibm-1362_P110-1999 ibm-1362 (No JAVA alias)
ibm-13676_P102-2001	ibm-13676_P102-2001 ibm-13676 (No JAVA alias)
ibm-1380_P100-1995	ibm-1380_P100-1995 ibm-1380 (No JAVA alias)
ibm-1382_P100-1995	ibm-1382_P100-1995 ibm-1382 (No JAVA alias)

Internal converter name	Aliases
ibm-17221_P100-2001	ibm-17221_P100-2001 ibm-17221 (No JAVA alias)
ibm-17248_X110-1999	ibm-17248_X110-1999 ibm-17248 (No JAVA alias)
ibm-21344_P101-2000	ibm-21344_P101-2000 ibm-21344 (No JAVA alias)
ibm-21427_P100-1999	ibm-21427_P100-1999 ibm-21427 (No JAVA alias)
ibm-256_P100-1995	ibm-256_P100-1995 ibm-256 (No JAVA alias)
ibm-259_P100-1995	ibm-259_P100-1995 ibm-259 IBM-Symbols csIBMSymbols (No JAVA alias)
ibm-274_P100-2000	ibm-274_P100-2000 ibm-274 IBM274 EBCDIC-BE CP274 csIBM274 (No JAVA alias)
ibm-275_P100-1995	ibm-275_P100-1995 ibm-275 IBM275 EBCDIC-BR cp275 csIBM275 (No JAVA alias)
ibm-286_P100-2003	ibm-286_P100-2003 ibm-286 EBCDIC-AT-DE-A csEBCDICATDEA (No JAVA alias)
ibm-293_P100-1995	ibm-293_P100-1995 ibm-293 (No JAVA alias)

Internal converter name	Aliases
ibm-300_P120-2006	ibm-300_P120-2006 ibm-300 (No JAVA alias)
ibm-301_P110-1997	ibm-301_P110-1997 ibm-301 (No JAVA alias)
ibm-33058_P100-2000	ibm-33058_P100-2000 ibm-33058 (No JAVA alias)
ibm-425_P101-2000	ibm-425_P101-2000 ibm-425 (No JAVA alias)
ibm-4930_P110-1999	ibm-4930_P110-1999 ibm-4930 (No JAVA alias)
ibm-4933_P100-2002	ibm-4933_P100-2002 ibm-4933 (No JAVA alias)
ibm-4948_P100-1995	ibm-4948_P100-1995 ibm-4948 (No JAVA alias)
ibm-4951_P100-1995	ibm-4951_P100-1995 ibm-4951 (No JAVA alias)
ibm-4952_P100-1995	ibm-4952_P100-1995 ibm-4952 (No JAVA alias)
ibm-4960_P100-1995	ibm-4960_P100-1995 ibm-4960 (No JAVA alias)
ibm-5039_P11A-1998	ibm-5039_P11A-1998 ibm-5039 (No JAVA alias)
ibm-5048_P100-1995	ibm-5048_P100-1995 ibm-5048 (No JAVA alias)
ibm-5049_P100-1995	ibm-5049_P100-1995 ibm-5049 (No JAVA alias)

Internal converter name	Aliases
ibm-5067_P100-1995	ibm-5067_P100-1995 ibm-5067 (No JAVA alias)
ibm-5104_X110-1999	ibm-5104_X110-1999 ibm-5104 (No JAVA alias)
ibm-806_P100-1998	ibm-806_P100-1998 ibm-806 (No JAVA alias)
ibm-808_P100-1999	ibm-808_P100-1999 ibm-808 (No JAVA alias)
ibm-834_P100-1995	ibm-834_P100-1995 ibm-834 (No JAVA alias)
ibm-835_P100-1995	ibm-835_P100-1995 ibm-835 (No JAVA alias)
ibm-837_P100-1995	ibm-837_P100-1995 ibm-837 (No JAVA alias)
ibm-848_P100-1999	ibm-848_P100-1999 ibm-848 (No JAVA alias)
ibm-849_P100-1999	ibm-849_P100-1999 ibm-849 (No JAVA alias)
ibm-859_P100-1999	ibm-859_P100-1999 ibm-859 (No JAVA alias)
ibm-8612_P100-1995	ibm-8612_P100-1995 ibm-8612 (No JAVA alias)
ibm-872_P100-1999	ibm-872_P100-1999 ibm-872 (No JAVA alias)

Internal converter name	Aliases
ibm-880_P100-1995	ibm-880_P100-1995 ibm-880 IBM880 cp880 EBCDIC-Cyrillic csIBM880 windows-20880 (No JAVA alias)
ibm-896_P100-1995	ibm-896_P100-1995 ibm-896 (No JAVA alias)
ibm-9027_P100-1999	ibm-9027_P100-1999 ibm-9027 (No JAVA alias)
ibm-9048_P100-1998	ibm-9048_P100-1998 ibm-9048 (No JAVA alias)
ibm-905_P100-1995	ibm-905_P100-1995 ibm-905 IBM905 CP905 ebcdic-cp-tr csIBM905 windows-20905 (No JAVA alias)
ibm-9056_P100-1995	ibm-9056_P100-1995 ibm-9056 (No JAVA alias)
ibm-9061_P100-1999	ibm-9061_P100-1999 ibm-9061 (No JAVA alias)
ibm-9145_P110-1997	ibm-9145_P110-1997 ibm-9145 (No JAVA alias)
ibm-9238_X110-1999	ibm-9238_X110-1999 ibm-9238 (No JAVA alias)

Internal converter name	Aliases
ibm-924_P100-1998	ibm-924_P100-1998 ibm-924 IBM00924 CCSID00924 CP00924 ebcdic-Latin9--euro (No JAVA alias)
ibm-926_P100-2000	ibm-926_P100-2000 ibm-926 (No JAVA alias)
ibm-927_P100-1995	ibm-927_P100-1995 ibm-927 (No JAVA alias)
ibm-928_P100-1995	ibm-928_P100-1995 ibm-928 (No JAVA alias)
ibm-941_P13A-2001	ibm-941_P13A-2001 ibm-941 (No JAVA alias)
ibm-944_P100-1995	ibm-944_P100-1995 ibm-944 (No JAVA alias)
ibm-946_P100-1995	ibm-946_P100-1995 ibm-946 (No JAVA alias)
ibm-947_P100-1995	ibm-947_P100-1995 ibm-947 (No JAVA alias)
ibm-948_P110-1999	ibm-948_P110-1999 ibm-948 (No JAVA alias)
ibm-951_P100-1995	ibm-951_P100-1995 ibm-951 (No JAVA alias)
ibm-952_P110-1997	ibm-952_P110-1997 ibm-952 (No JAVA alias)
ibm-953_P100-2000	ibm-953_P100-2000 ibm-953 (No JAVA alias)

Internal converter name	Aliases
ibm-955_P110-1997	ibm-955_P110-1997 ibm-955 (No JAVA alias)
ibm-9577_P100-2001	ibm-9577_P100-2001 ibm-9577 ibm-1385 (No JAVA alias)
iso-8859_16-2001	iso-8859_16-2001 ISO-8859-16 iso-ir-226 ISO_8859-16:2001 latin10 l10 (No JAVA alias)

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Code page converters" on page 823
Brokers complete string operations in Universal Character Set coded in 2 octets (UCS-2). If incoming strings are not encoded in UCS-2, they are converted to UCS-2 on arrival.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Generating a new code page converter" on page 824
Generate a code page converter to handle conversions of data that belongs to a code page that is not in the default set of code pages provided by WebSphere Message Broker.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"Chinese code page GB18030"
If you are working with messages in Chinese code page GB18030, your messages might be subject to some restrictions.

Related information:

WebSphere MQ Version 7 Information Center online

Chinese code page GB18030:

If you are working with messages in Chinese code page GB18030, your messages might be subject to some restrictions.

The broker can input, manipulate, and output application messages that are encoded in code page IBM-5488 (GB18030 support) with the following restrictions:

- If you configure a message flow to store GB18030 data in character form in a user database, ensure that the database manager that you are using supports GB18030.
- To enable support for GB18030 in the WebSphere Message Broker Toolkit:
  - If you run a WebSphere Message Broker Toolkit that requires GB18030 support on a computer that is running Windows 2003, apply the GB18030 patch supplied by Microsoft. This support is included in Windows XP.
  - Change the text font preference in the WebSphere Message Broker Toolkit to use GB18030:
    - Select **Window** > **Preferences**.
    - Expand the **Workbench** item in the left pane of the Preferences dialog (click the plus sign), and select **Fonts**.
    - In the Fonts window, select **Text Font**. Click **Change**, and select the correct values in the Fonts selection dialog box.
    - Click **OK** to confirm the selection and close the dialog box.
    - Click **Apply** to apply the change, then click **OK** to close the Preferences dialog box.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Creating the user databases" on page 661
If your message flows create, update, retrieve, or delete application and business data in one or more user databases, create the databases before you deploy the message flows to a broker.

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

WebSphere MQ connections

The number of WebSphere MQ connections a broker requires to its queue manager depends on the actions of the message flows that access the WebSphere MQ resource.

For each broker flow that accesses a queue, one connection is required for every message flow thread. If a different node on the same thread uses the same queue manager, the same connection is used.

The number of queue handles required also depends on the behavior of the flow. For each flow that accesses queues, one queue handle is required for each unique queue name for every message flow thread. Nodes that access the same queue name in the same flow use the same queue handle.

When you start a broker, and while it is running, it opens WebSphere MQ queue handles. The broker caches these queue handles. For example, when a message flow node initiates access to the first WebSphere MQ resource it uses, it opens a connection for the queue manager and opens the queue. This connection is opened

the first time that a message is processed by that message flow node. For MQInput nodes, the connection is opened when the flow is started. This queue handle remains open until:

The message flow becomes idle, and has not been used for 1 minute

The execution group is stopped

The broker is stopped

The queue handle for the input node is not released when the flow is idle. The queue handle is released only when you stop the message flow.

A thread performing WebSphere MQ work becomes idle when it has not received any messages on its input queue for 1 minute. The allowed idle time starts from when the input queue being read becomes empty. If a message flow gets a message from the input queue, the timer is reset.

When a message flow is idle, the execution group periodically releases WebSphere MQ queue handles. Therefore, connections held by the broker reflect its current use of these resources.

Related concepts:

"User database connections" on page 2110
User databases contain business data that is written and accessed by deployed message flows. You must create connections from the broker to the user database by using ODBC or JDBC.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Data integrity in message flows

Code pages in which data is manipulated must be compatible between brokers and databases that are accessed by those brokers.

Subscription data that is retrieved from client applications (for example, topics from publishers and subscribers, and content filters from subscribers), and the character data entered through the WebSphere Message Broker Toolkit (for example, message flow names), are stored by the broker. This data is translated from its originating code page to the code page of the process in which the broker is running, then by the database manager to the code page in which the database or databases were created.

To preserve data consistency and integrity, ensure that all this subscription data and WebSphere Message Broker Toolkit character data is originated in a compatible code page to the two code pages to which it is translated. If you do not do so, you might get unpredictable results and lose data.

The restrictions described do not apply to user data in messages. Ensure that any data in messages generated by your applications is compatible with the code page of all databases that you access from your message flows.

SQL statements that are generated as a result of explicit reference to databases in message processing nodes can contain character data that has a variety of sources. For example, the data might have been entered through the WebSphere Message

Broker Toolkit, derived from message content, or read from another database. All this data is translated from its originating code page to the code page in which the broker was created, then by the database manager to the code page in which the database was created. Ensure that these three code pages are compatible to avoid data conversion problems.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message flow transactions" on page 1281
A transaction describes a set of updates that are made by an application program, which must be managed together. The updates might be made to one or more systems. The updates made by the program are controlled by the environment in which the program executes, and either all are completed, or none. This property of a transaction is known as *consistency*: transactions might have other properties of *atomicity*, *isolation*, and *durability*.

Related tasks:

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Configuring transactionality for message flows" on page 1290
A message flow runs in a single transaction, which is started when data is received by an input node, and can be committed or rolled back when all processing has completed.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"Database connections for coordinated message flows" on page 4235
When you configure a message flow to access a database, the broker establishes a connection to that database based on the ODBC DSN.

"Database support for coordinated message flows" on page 4236
If the message flow processing includes interaction with an external database, you can coordinate the transaction by using XA technology.

Exception list structure

An exception list contains information about exceptions, such as error numbers, the name of the node that generated the exception, and the reason for the exception.

The following figure shows one way in which to construct an exception list.

```
ExceptionList {
    RecoverableException = {                 1
        File     = 'f:/build/argo/src/DataFlowEngine/ImbDataFlowNode.cpp'
        Line     = 538
        Function = 'ImbDataFlowNode::createExceptionList'
        Type     = 'ComIbmComputeNode'
        Name     = '0e416632-de00-0000-0080-bdb4d59524d5'
        Label    = 'mf1.Compute1'
        Text     = 'Node throwing exception'
        Catalog  = 'WebSphere Message
Broker2'
        Severity = 3
```

```
        Number   = 2230
        RecoverableException = {               2
            File     = 'f:/build/argo/src/DataFlowEngine/ImbRdlBinaryExpression.cpp'
            Line     = 231
            Function = 'ImbRdlBinaryExpression::scalarEvaluate'
            Type     = 'ComIbmComputeNode'
            Name     = '0e416632-de00-0000-0080-bdb4d59524d5'
            Label    = 'mf1.Compute1'
            Text     = 'error evaluating expression'
            Catalog  = 'WebSphere Message
Broker2'
            Severity = 2
            Number   = 2439
            Insert   = {
               Type = 2
               Text = '2'
            }
            Insert   = {
               Type = 2
               Text = '30'
            }
            RecoverableException = {             3
                File     = 'f:/build/argo/src/DataFlowEngine/ImbRdlValueOperations.cpp'
                Line     = 257
                Function = 'intDivideInt'
                Type     = 'ComIbmComputeNode'
                Name     = '0e416632-de00-0000-0080-bdb4d59524d5'
                Label    = 'mf1.Compute1'
                Text     = 'Divide by zero calculating '%1 / %2''
                Catalog  = 'WebSphere Message
Broker2'
                Severity = 2
                Number   = 2450
                Insert   = }
                   Type = 5
                   Text = '100 / 0'
                }
            }
        }
    }
}
```

Notes:

1. The first exception description 1 is a child of the root. This identifies error number 2230, indicating that an exception has been thrown. The node that has thrown the exception is also identified (*mf1.Compute1*).
2. Exception description 2 is a child of the first exception description 1. This identifies error number 2439.
3. Exception description 3 is a child of the second exception description 2. This identifies error number 2450, which indicates that the node has attempted to divide by zero.

The following topics provide examples of exception lists that have been written to the trace output destination (by the Trace node):

- "Database exception trace output" on page 4226
- "Conversion exception trace output" on page 4228
- "Parser exception trace output" on page 4230
- "User exception trace output" on page 4232

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an

input message is received.

"Exception list tree structure" on page 1066
The exception list tree is a part of the logical message tree in which the message flow writes information about exceptions that occur when a message is processed.

Related tasks:

"Accessing the ExceptionList tree using ESQL" on page 2471
The ExceptionList tree has its own correlation name, ExceptionList, and you must use this in all ESQL statements that refer to or set the content of this tree.

"Throwing an exception" on page 2511
If you detect an error or other situation in your message flow in which you want message processing to be ended, you can throw an exception in a message flow in two ways.

Related reference:

"Throw node" on page 4929
Use the Throw node to throw an exception in a message flow.

"Trace node" on page 4942
Use the Trace node to generate trace records that you can use to monitor the behavior of a message flow.

"THROW statement" on page 5161
Use the THROW statement to generate a user exception.

Database exception trace output:

Use a Trace node to produce details of exceptions that occur when a database exception is detected.

The following figure shows an extract of the output that might be generated by a Trace node that has `Pattern` property set to a value that represents a structure that includes the exception list tree.

The exception shown occurred when a database exception was detected

```
ExceptionList = (
  (0x1000000)RecoverableException = (
    (0x3000000)File                = 'F:\build\S000_D\src\DataFlowEngine\ImbComputeNode.cpp'
    (0x3000000)Line                = 402
    (0x3000000)Function            = 'ImbComputeNode::evaluate'
    (0x3000000)Type                = 'ComIbmComputeNode'
    (0x3000000)Name                = 'acd8f35d-e700-0000-0080-b78796c5e70d'
    (0x3000000)Label               = 'esql_13485_check_defect.Compute1'
    (0x3000000)Text                = 'Caught exception and rethrowing'
    (0x3000000)Catalog             = 'WMQIv210'
    (0x3000000)Severity            = 3
    (0x3000000)Number              = 2230
    (0x1000000)RecoverableException = (
      (0x3000000)File              = 'F:\build\S000_D\src\DataFlowEngine\ImbRdl\ImbRdlExternalDb.cpp'
      (0x3000000)Line              = 278
      (0x3000000)Function          = 'SqlExternalDbStmt::executeStmt'
      (0x3000000)Type              = 'ComIbmComputeNode'
      (0x3000000)Name              = 'acd8f35d-e700-0000-0080-b78796c5e70d'
      (0x3000000)Label             = 'esql_13485_check_defect.Compute1'
      (0x3000000)Text              = 'The following error occurred execution SQL statement &3. inserts where &4'
      (0x3000000)Catalog           = 'WMQIv210'
      (0x3000000)Severity          = 3
      (0x3000000)Number            = 2519
      (0x1000000)Insert            = (
        (0x3000000)Type = 2
        (0x3000000)Text = '1'
      )
      (0x1000000)Insert            = (
        (0x3000000)Type = 2
        (0x3000000)Text = '1'
      )
      (0x1000000)Insert            = (
        (0x3000000)Type = 5
        (0x3000000)Text = 'USERDB'
      )
      (0x1000000)Insert            = (
        (0x3000000)Type = 5
        (0x3000000)Text = 'DELETE FROM DB2ADMIN.STOCK WHERE (STOCK_ID)=(?)'
      )
      (0x1000000)Insert            = (
        (0x3000000)Type = 5
        (0x3000000)Text = '500027,  '
      )
      (0x1000000)DatabaseException = (
        (0x3000000)File     = 'F:\build\S000_D\src\DataFlowEngine\ImbOdbc.cpp'
        (0x3000000)Line     = 153
        (0x3000000)Function = 'ImbOdbcHandle::checkRcInner'
        (0x3000000)Type     = ''
        (0x3000000)Name     = ''
        (0x3000000)Label    = ''
        (0x3000000)Text     = 'Root SQL exception'
        (0x3000000)Catalog  = 'WMQIv210'
        (0x3000000)Severity = 3
        (0x3000000)Number   = 2321
        (0x1000000)Insert   = (
          (0x3000000)Type = 2
          (0x3000000)Text = '100'
        )
      )
    )
  )
)
```

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an

input message is received.

"Exception list tree structure" on page 1066
The exception list tree is a part of the logical message tree in which the message flow writes information about exceptions that occur when a message is processed.

Related tasks:

"Accessing the ExceptionList tree using ESQL" on page 2471
The ExceptionList tree has its own correlation name, ExceptionList, and you must use this in all ESQL statements that refer to or set the content of this tree.

"Throwing an exception" on page 2511
If you detect an error or other situation in your message flow in which you want message processing to be ended, you can throw an exception in a message flow in two ways.

Related reference:

"Throw node" on page 4929
Use the Throw node to throw an exception in a message flow.

"Trace node" on page 4942
Use the Trace node to generate trace records that you can use to monitor the behavior of a message flow.

"THROW statement" on page 5161
Use the THROW statement to generate a user exception.

Conversion exception trace output:

Use a Trace node to produce details of exceptions that occur when a conversion (CAST) exception is detected.

The following figure shows an extract of the output that might be generated by a Trace node that has the `Pattern` property set to a value that represents a structure that includes the exception list tree.

The exception shown occurred when a conversion (CAST) exception was detected.

```
ExceptionList = (
  (0x1000000)RecoverableException = (
    (0x3000000)File                = 'F:\build\S000_D\src\DataFlowEngine\ImbComputeNode.cpp'
    (0x3000000)Line                = 402
    (0x3000000)Function            = 'ImbComputeNode::evaluate'
    (0x3000000)Type                = 'ComIbmComputeNode'
    (0x3000000)Name                = 'acd8f35d-e700-0000-0080-b78796c5e70d'
    (0x3000000)Label               = 'esql_13485_check_defect.Compute1'
    (0x3000000)Text                = 'Caught exception and rethrowing'
    (0x3000000)Catalog             = 'WMQIv210'
    (0x3000000)Severity            = 3
    (0x3000000)Number              = 2230
    (0x1000000)RecoverableException = (
      (0x3000000)File               = 'F:\build\S000_D\src\DataFlowEngine\ImbRdl\ImbRdlTypeCast.cpp'
      (0x3000000)Line               = 163
      (0x3000000)Function           = 'SqlTypeCast::evaluate'
      (0x3000000)Type               = ''
      (0x3000000)Name               = ''
      (0x3000000)Label              = ''
      (0x3000000)Text               = 'Error casting from %3 to %4'
      (0x3000000)Catalog            = 'WMQIv210'
      (0x3000000)Severity           = 3
      (0x3000000)Number             = 2521
      (0x1000000)Insert             = (
        (0x3000000)Type = 2
        (0x3000000)Text = '12'
      )
      (0x1000000)Insert             = (
        (0x3000000)Type = 2
        (0x3000000)Text = '28'
      )
      (0x1000000)Insert             = (
        (0x3000000)Type = 5
        (0x3000000)Text = 'CHARACTER'
      )
      (0x1000000)Insert             = (
        (0x3000000)Type = 5
        (0x3000000)Text = 'INTEGER'
      )
      (0x1000000)ConversionException = (
        (0x3000000)File     = 'F:\build\S000_D\src\CommonServices\ImbUtility.cpp'
        (0x3000000)Line     = 195
        (0x3000000)Function = 'imbWcsToInt64'
        (0x3000000)Type     = ''
        (0x3000000)Name     = ''
        (0x3000000)Label    = ''
        (0x3000000)Text     = 'Invalid characters'
        (0x3000000)Catalog  = 'WMQIv210'
        (0x3000000)Severity = 3
        (0x3000000)Number   = 2595
        (0x1000000)Insert   = (
          (0x3000000)Type = 5
          (0x3000000)Text = 'fred'
        )
      )
    )
  )
)
```

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Exception list tree structure" on page 1066
The exception list tree is a part of the logical message tree in which the message

flow writes information about exceptions that occur when a message is processed.

Related tasks:

"Accessing the ExceptionList tree using ESQL" on page 2471
The ExceptionList tree has its own correlation name, ExceptionList, and you must use this in all ESQL statements that refer to or set the content of this tree.

"Throwing an exception" on page 2511
If you detect an error or other situation in your message flow in which you want message processing to be ended, you can throw an exception in a message flow in two ways.

Related reference:

"Throw node" on page 4929
Use the Throw node to throw an exception in a message flow.

"Trace node" on page 4942
Use the Trace node to generate trace records that you can use to monitor the behavior of a message flow.

"THROW statement" on page 5161
Use the THROW statement to generate a user exception.

"CAST function" on page 5245

Parser exception trace output:

Use a Trace node to produce details of exceptions that occur when the message parser is started.

The following figure shows an extract of the output that might be generated by a Trace node that has the `Pattern` property set to a value that represents a structure that includes the exception list tree.

The exception shown occurred when the message parser was invoked.

```
ExceptionList = (
  (0x1000000)RecoverableException = (
    (0x3000000)File            = 'F:\build\S000_D\src\DataFlowEngine\ImbMqOutputNode.cpp'
    (0x3000000)Line            = 1444
    (0x3000000)Function        = 'ImbMqOutputNode::evaluate'
    (0x3000000)Type            = 'ComIbmMQOutputNode'
    (0x3000000)Name            = 'c76eb6cd-e600-0000-0080-b78796c5e70d'
    (0x3000000)Label           = 'esql_13485_check_defect.OUT'
    (0x3000000)Text            = 'Caught exception and rethrowing'
    (0x3000000)Catalog         = 'WMQIv210'
    (0x3000000)Severity        = 3
    (0x3000000)Number          = 2230
    (0x1000000)ParserException = (
      (0x3000000)File            = 'F:\build\S000_D\src\MTI\MTIforBroker\GenXmlParser2\XmlImbParser.cpp'
      (0x3000000)Line            = 210
      (0x3000000)Function        = 'XmlImbParser::refreshBitStreamFromElements'
      (0x3000000)Type            = 'ComIbmMQInputNode'
      (0x3000000)Name            = 'ce64b6cd-e600-0000-0080-b78796c5e70d'
      (0x3000000)Label           = 'esql_13485_check_defect.IN'
      (0x3000000)Text            = 'XML Writing Errors have occurred'
      (0x3000000)Catalog         = 'WMQIv210'
      (0x3000000)Severity        = 3
      (0x3000000)Number          = 5010
      (0x1000000)ParserException = (
        (0x3000000)File     = 'F:\build\S000_D\src\MTI\MTIforBroker\GenXmlParser2\XmlImbParser.cpp'
        (0x3000000)Line     = 551
        (0x3000000)Function = 'XmlImbParser::checkForBodyElement'
        (0x3000000)Type     = ''
        (0x3000000)Name     = ''
        (0x3000000)Label    = ''
        (0x3000000)Text     = 'No valid body of the document could be found.'
        (0x3000000)Catalog  = 'WMQIv210'
        (0x3000000)Severity = 3
        (0x3000000)Number   = 5005
      )
    )
  )
)
```

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Exception list tree structure" on page 1066
The exception list tree is a part of the logical message tree in which the message flow writes information about exceptions that occur when a message is processed.

Related tasks:

"Accessing the ExceptionList tree using ESQL" on page 2471
The ExceptionList tree has its own correlation name, ExceptionList, and you must use this in all ESQL statements that refer to or set the content of this tree.

"Throwing an exception" on page 2511
If you detect an error or other situation in your message flow in which you want message processing to be ended, you can throw an exception in a message flow in two ways.

Related reference:

"Throw node" on page 4929
Use the Throw node to throw an exception in a message flow.

"Trace node" on page 4942
Use the Trace node to generate trace records that you can use to monitor the behavior of a message flow.

"THROW statement" on page 5161
Use the THROW statement to generate a user exception.

User exception trace output:

Use a Trace node to produce details of exceptions that occur when user exceptions are generated.

The following example is an extract of the output that might be generated by a Trace node that has the `Pattern` property set to a value that represents a structure that includes the exception list tree.

The exception shown occurred when a user exception was generated (with the ESQL THROW statement).

```
ExceptionList = (
  (0x1000000)RecoverableException = (
    (0x3000000)File          = 'F:\build\S000_D\src\DataFlowEngine\ImbComputeNode.cpp'
    (0x3000000)Line          = 402
    (0x3000000)Function      = 'ImbComputeNode::evaluate'
    (0x3000000)Type          = 'ComIbmComputeNode'
    (0x3000000)Name          = 'acd8f35d-e700-0000-0080-b78796c5e70d'
    (0x3000000)Label         = 'esql_13485_check_defect.Compute1'
    (0x3000000)Text          = 'Caught exception and rethrowing'
    (0x3000000)Catalog       = 'WMQIv210'
    (0x3000000)Severity      = 3
    (0x3000000)Number        = 2230
    (0x1000000)UserException = (
      (0x3000000)File     = 'F:\build\S000_D\src\DataFlowEngine\ImbRdl\ImbRdlThrowExceptionStatements.cpp'
      (0x3000000)Line     = 148
      (0x3000000)Function = 'SqlThrowExceptionStatement::execute'
      (0x3000000)Type     = 'ComIbmComputeNode'
      (0x3000000)Name     = 'acd8f35d-e700-0000-0080-b78796c5e70d'
      (0x3000000)Label    = 'esql_13485_check_defect.Compute1'
      (0x3000000)Text     = 'User Generated SQL 'USER' exception'
      (0x3000000)Catalog  = 'WMQIv210'
      (0x3000000)Severity = 1
      (0x3000000)Number   = 2949
      (0x1000000)Insert   = (
        (0x3000000)Type = 5
        (0x3000000)Text = 'USER'
      )
      (0x1000000)Insert   = (
        (0x3000000)Type = 5
        (0x3000000)Text = 'Insert1'
      )
      (0x1000000)Insert   = (
        (0x3000000)Type = 5
        (0x3000000)Text = 'Insert2'
      )
      (0x1000000)Insert   = (
        (0x3000000)Type = 5
        (0x3000000)Text = 'etc'
      )
      (0x1000000)Insert   = (
        (0x3000000)Type = 5
        (0x3000000)Text = ''
      )
      (0x1000000)Insert   = (
        (0x3000000)Type = 5
        (0x3000000)Text = ''
      )
      (0x1000000)Insert   = (
        (0x3000000)Type = 5
        (0x3000000)Text = ''
```

```
      )
      (0x1000000)Insert   = (
        (0x3000000)Type = 5
        (0x3000000)Text = ''
      )
      (0x1000000)Insert   = (
        (0x3000000)Type = 5
        (0x3000000)Text = ''
      )
    )
  )
)
```

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Exception list tree structure" on page 1066
The exception list tree is a part of the logical message tree in which the message flow writes information about exceptions that occur when a message is processed.

Related tasks:

"Accessing the ExceptionList tree using ESQL" on page 2471
The ExceptionList tree has its own correlation name, ExceptionList, and you must use this in all ESQL statements that refer to or set the content of this tree.

"Throwing an exception" on page 2511
If you detect an error or other situation in your message flow in which you want message processing to be ended, you can throw an exception in a message flow in two ways.

Related reference:

"Throw node" on page 4929
Use the Throw node to throw an exception in a message flow.

"Trace node" on page 4942
Use the Trace node to generate trace records that you can use to monitor the behavior of a message flow.

"THROW statement" on page 5161
Use the THROW statement to generate a user exception.

Message flow porting

If you have configured a message flow that runs on a broker on a distributed system, and you now want to deploy it to a broker that runs on z/OS, you might have to take additional actions to port the flow successfully.

Consider the following resources and attributes:

WebSphere MQ queue manager and queue names

WebSphere MQ imposes some restrictions for resource names on z/OS:

- The queue manager name cannot be greater than four characters.
- All queue names must be in uppercase. Although using quotation marks preserves the case, certain WebSphere MQ activities on z/OS cannot find the queue names being referenced.

For more information about configuring on z/OS, refer to the *Concepts and Planning Guide* section of the WebSphere MQ Version 7 Information Center online

File system references

File system references must reflect a UNIX file path. If you deploy a

message flow to z/OS that you have previously run on Windows, you might have to make changes. If you have previously deployed the message flow to a UNIX system (AIX, Linux, Solaris, or HP-UX), you do not have to make any changes.

Databases
If the message flow accesses one or more databases, it might be subject to some restrictions based on the system on which the database is defined. These restrictions are described in "Database locations" on page 3595.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"User database connections" on page 2110
User databases contain business data that is written and accessed by deployed message flows. You must create connections from the broker to the user database by using ODBC or JDBC.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Accessing databases from message flows" on page 2112
Create and configure message flows to access user databases.

"Configuring transactionality for message flows" on page 1290
A message flow runs in a single transaction, which is started when data is received by an input node, and can be committed or rolled back when all processing has completed.

Related reference:

"Supported databases" on page 3591
You can optionally configure databases to contain data that is accessed by your message flows. Databases from IBM and other suppliers are supported at specific versions on supported operating systems.

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"Database locations" on page 3595
The broker can access databases set up on the local computer or on a remote server, subject to restrictions.

Related information:

WebSphere MQ Version 7 Information Center online

Coordinated message flows

A coordinated message flow runs in a single transaction, which is started when a message is received by an input node, and can be committed or rolled back when all processing has completed.

The following topics provide reference information for database use in coordinated message flows:

- "Database connections for coordinated message flows" on page 4235
- "Database support for coordinated message flows" on page 4236

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an

input message is received.

"Message flow transactions" on page 1281
A transaction describes a set of updates that are made by an application program, which must be managed together. The updates might be made to one or more systems. The updates made by the program are controlled by the environment in which the program executes, and either all are completed, or none. This property of a transaction is known as *consistency*: transactions might have other properties of *atomicity*, *isolation*, and *durability*.

Related tasks:

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Configuring transactionality for message flows" on page 1290
A message flow runs in a single transaction, which is started when data is received by an input node, and can be committed or rolled back when all processing has completed.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Database connections for coordinated message flows:

When you configure a message flow to access a database, the broker establishes a connection to that database based on the ODBC DSN.

When you configure a message flow to access a database, the broker establishes a connection to that database based on the ODBC or JDBC DSN. To coordinate the database updates with other updates (determined by the configuration you have set for each node that accesses a database), the broker makes a connection for each transaction mode for each DSN accessed on each message flow thread.

Therefore, if you set the `Transaction Mode` property for one node in the message flow to Automatic, and for another node to Commit, the broker establishes two separate connections to this DSN from the same thread. Take this action into account when you calculate the number of connections required between a broker and a specific DSN.

For further information about connections made by the broker to user databases, see "User database connections" on page 2110.

For details of these connections, refer to "Enabling ODBC connections to the databases" on page 668 and "Enabling JDBC connections to the databases" on page 683.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"User database connections" on page 2110
User databases contain business data that is written and accessed by deployed message flows. You must create connections from the broker to the user database

by using ODBC or JDBC.

Related tasks:

"Enabling ODBC connections to the databases" on page 668
Set up the resources and environment that the broker requires for Open Database Connectivity (ODBC) connections to user databases on distributed systems.

"Enabling JDBC connections to the databases" on page 683
Configure connections to a user database through a JDBCProvider service.

"Configuring transactionality for message flows" on page 1290
A message flow runs in a single transaction, which is started when data is received by an input node, and can be committed or rolled back when all processing has completed.

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Database support for coordinated message flows:

If the message flow processing includes interaction with an external database, you can coordinate the transaction by using XA technology.

XA coordination ensures that all participants update or return to a consistent state. This external coordination support is provided by the underlying WebSphere MQ facilities on distributed systems, and by Resource Recovery Services (RRS) on z/OS.

The following databases provide the correct level of XA support for coordinating message flows on distributed systems:

- DB2
- Oracle
- Sybase

On z/OS, database support for coordinated message flows is provided by DB2 only.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Configuring transactionality for message flows" on page 1290
A message flow runs in a single transaction, which is started when data is received by an input node, and can be committed or rolled back when all processing has completed.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

Element definitions for message parsers

The topics in this section discuss data types for the WebSphere MQ headers, and define the element names, types, and attributes for each of the supported headers:

For each parser, the following terms are defined:

- Root element name: the name of the syntax element created by the parser at the root of its own part of the tree.
- Class name: the name by which the parser defines itself to WebSphere Message Broker.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Developing Java" on page 2628
When you use the JavaCompute node, you customize it to determine the exact processing that it provides.

"Accessing headers" on page 2453
If the input message received by an input node includes message headers that are recognized by the input node, the node invokes the correct parser for each header. You can access these headers using ESQL.

Data types of fields and elements:

The fields within WebSphere MQ headers and other subtrees built from the message are of a particular data type. When you manipulate the messages and their headers using ESQL in the message flow nodes, be aware of type information in field references:

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message tree structure" on page 1045
The message tree is a part of the logical message tree in which the broker stores its internal representation of the message body.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"Element definitions for message parsers" on page 4237

Data types of the fields in the WebSphere MQ headers:

The fields in the WebSphere MQ headers have specific data types. Parsers are supplied for the headers included in WebSphere MQ messages.

The parsers determine the data type of each field in the header:

- "The MQCFH parser" on page 4245
- "The MQCIH parser" on page 4246
- "The MQDLH parser" on page 4248
- "The MQIIH parser" on page 4248
- "The MQMD parser" on page 4249
- "The MQMDE parser" on page 4251
- "The MQRFH parser" on page 4252
- "The MQRFH2 and MQRFH2C parsers" on page 4253
- "The MQRMH parser" on page 4253
- "The MQSAPH parser" on page 4254
- "The MQWIH parser" on page 4255
- "The SMQ_BMH parser" on page 4256

The mapping of the WebSphere MQ data types to the data types used in the broker is shown in the following table.

Data type of the field	Represented as
MQLONG	INTEGER
MQCHAR, MQCHAR4	CHARACTER
MQBYTE, MQBYTEn	BLOB

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Developing Java" on page 2628
When you use the JavaCompute node, you customize it to determine the exact processing that it provides.

"Accessing headers" on page 2453
If the input message received by an input node includes message headers that are recognized by the input node, the node invokes the correct parser for each header. You can access these headers using ESQL.

Related reference:

"Element definitions for message parsers" on page 4237

"Data types of fields and elements" on page 4237

"Data types for elements in the Properties subtree"
A parser is supplied for the Properties subtree; it associates each field with a specific data type.

"Data types for elements in the MQ DestinationData subtree" on page 4240
The DestinationData subtree is part of the Destination subtree in the local environment. Local environment trees are created by input nodes when they receive a message and, optionally, by Compute nodes. When created, the trees are empty but you can create data in them by using ESQL statements coded in any of the SQL nodes.

"Data types for elements in an MRM message" on page 6254
A parser is supplied for the body of a message in the MRM domain; it associates each field with a specific data type.

"Data types for an unstructured (BLOB) message" on page 4244
A parser is supplied for the body of a message in the BLOB domain; it associates each field with a specific data type.

Data types for elements in the Properties subtree:

A parser is supplied for the Properties subtree; it associates each field with a specific data type.

The fields and data type of each field are shown in the following table.

Data type of the element	**Represented as**
CodedCharSetId	INTEGER
CreationTime	TIMESTAMP
ContentType	CHARACTER
Encoding	INTEGER
ExpirationTime	TIMESTAMP
IdentityMappedIssuedBy	CHARACTER
IdentityMappedPassword	CHARACTER
IdentityMappedToken	CHARACTER
IdentityMappedType	CHARACTER
IdentitySourceIssuedBy	CHARACTER
IdentitySourcePassword	CHARACTER
IdentitySourceToken	CHARACTER
IdentitySourceType	CHARACTER
MessageFormat	CHARACTER
MessageSet	CHARACTER
MessageType	CHARACTER
Persistence	BOOLEAN

Data type of the element	Represented as
Priority	INTEGER
ReplyIdentifier	BLOB
ReplyProtocol	CHARACTER
Topic (this field contains a list)	CHARACTER
Transactional	BOOLEAN

Related concepts:

"Message tree structure" on page 1045
The message tree is a part of the logical message tree in which the broker stores its internal representation of the message body.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Accessing the Properties tree" on page 2460
The Properties tree has its own correlation name, Properties, and you must use this in all ESQL statements that refer to or set the content of this tree.

Related reference:

"Element definitions for message parsers" on page 4237

"Data types of fields and elements" on page 4237

"Data types of the fields in the WebSphere MQ headers" on page 4238
The fields in the WebSphere MQ headers have specific data types. Parsers are supplied for the headers included in WebSphere MQ messages.

"Data types for elements in the MQ DestinationData subtree"
The DestinationData subtree is part of the Destination subtree in the local environment. Local environment trees are created by input nodes when they receive a message and, optionally, by Compute nodes. When created, the trees are empty but you can create data in them by using ESQL statements coded in any of the SQL nodes.

"Data types for elements in an MRM message" on page 6254
A parser is supplied for the body of a message in the MRM domain; it associates each field with a specific data type.

"Data types for an unstructured (BLOB) message" on page 4244
A parser is supplied for the body of a message in the BLOB domain; it associates each field with a specific data type.

Data types for elements in the MQ DestinationData subtree:

The DestinationData subtree is part of the Destination subtree in the local environment. Local environment trees are created by input nodes when they receive a message and, optionally, by Compute nodes. When created, the trees are empty but you can create data in them by using ESQL statements coded in any of the SQL nodes.

The Destination subtree consists of subtrees for zero or more protocols; for example, WebSphere MQ and WebSphere MQ Everyplace, or a subtree for routing destinations (RouterList), or both.

The protocol tree has two children:

- Defaults is the first element; there can be only one.

- DestinationData is the following element, and can be repeated any number of times, to represent each destination to which a message is sent.

"Local environment tree structure" on page 1056 includes a picture of a typical tree, showing a Destination tree that has both protocol and RouterList subtrees.

The structure of data within the DestinationData folder is the same as that in Defaults for the same protocol, and can be used to override the default values in Defaults. You can therefore set up Defaults to contain values that are common to all destinations, and set only the unique values in each DestinationData subtree. If a value is set neither in DestinationData, nor in Defaults, the value that you have set for the corresponding node property is used.

The fields, data type, and valid values for each element of Defaults and DestinationData subtrees for WebSphere MQ are shown in the following table. "MQOutput node" on page 4612 describes the corresponding node properties.

Refer to "Accessing the local environment tree" on page 2463 for information about using DestinationData.

Data type of the element	Represented as	Corresponding node property	Valid values
queueManagerName	CHARACTER	Queue Manager Name	
queueName	CHARACTER	Queue Name	
transactionMode	CHARACTER	Transaction Mode	no, yes, automatic
persistenceMode	CHARACTER	Persistence Mode	no, yes, automatic, asQdef
newMsgId	CHARACTER	New Message ID	no, yes
newCorrelId	CHARACTER	New Correlation ID	no, yes
segmentationAllowed	CHARACTER	Segmentation Allowed	no, yes
alternateUserAuthority	CHARACTER	Alternate User Authority	no, yes
replyToQMgr	CHARACTER	Reply-to queue manager	
replyToQ	CHARACTER	Reply-to queue	

Case-sensitivity for data types and values

When you create these fields in the DestinationData folder, enter the data type and value exactly as shown in the table. If any variations in spelling or case are used, these fields or values are ignored in the DestinationData records and the next available value is used.

For example, the following ESQL samples could result in unexpected output:

```
SET OutputLocalEnvironment.Destination.MQ.DestinationData[1].persistenceMode = 'YES';
SET OutputLocalEnvironment.Destination.MQ.DestinationData[2].PersistenceMode = 'yes';
```

In each case, the DestinationData folder might not write a persistent message for these destinations. In the first example, the persistenceMode field has been given a value of 'YES', which is not one of the valid values listed in the table, and this value is ignored. In the second example, the field named 'PersistenceMode' is specified incorrectly and is ignored. Either the persistenceMode value of the Defaults folder, or the value of the associated attribute on the MQOutput node are used. If this behavior causes a value of 'no' or 'automatic' to be used, a persistent message is not written.

If a DestinationData folder is producing unexpected output, check that you have used the correct case and spelling in the fields and values.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Local environment tree structure" on page 1056
The local environment tree is a part of the logical message tree in which you can store information while the message flow processes the message.

Related tasks:

"Creating destination lists" on page 1477
Create a list of destinations to indicate where a message is sent.

"Accessing the local environment tree" on page 2463
The local environment tree has its own correlation name, LocalEnvironment, and you must use this name in all ESQL statements that refer to or set the content of this tree.

Related reference:

"Element definitions for message parsers" on page 4237

"Data types of fields and elements" on page 4237

"Data types of the fields in the WebSphere MQ headers" on page 4238
The fields in the WebSphere MQ headers have specific data types. Parsers are supplied for the headers included in WebSphere MQ messages.

"Data types for elements in the Properties subtree" on page 4239
A parser is supplied for the Properties subtree; it associates each field with a specific data type.

"Data types for elements in an MRM message" on page 6254
A parser is supplied for the body of a message in the MRM domain; it associates each field with a specific data type.

"Data types for an unstructured (BLOB) message" on page 4244
A parser is supplied for the body of a message in the BLOB domain; it associates each field with a specific data type.

Using LocalEnvironment variables with JMSOutput and JMSReply nodes:

The LocalEnvironment data elements related to the processing of JMS Messages in the JMSOutput and JMSReply nodes.

LocalEnvironment.WrittenDestination.JMS.DestinationData fields

The DestinationData element is a data structure created by the JMSOutput and JMSReply nodes for each message that is sent to a JMS Queue or published to a JMS Topic when:

- The Out terminal of the node is connected to another node in the message flow
- An output message callback routine has been enabled for the message flow. See "cciOutputMessageCallback" on page 6626.

The fields in the DestinationData structure are described in the following table, and can be used by a receiving application, or an output message callback routine, to link request messages with reply messages:

Element name	Element Data Type	Description
destinationName	CHARACTER	The name of the JMS Queue to which the node sends an outgoing message or the JMS Topic to which the node publishes. This value can specified in these formats: • a JNDI administered object (predefined in the JNDI bindings specified in the `Location JNDI bindings` node property). In which case the format of the value is `jndi://<JNDI Object name>` • a character string that represents the JMS Destination name in an internal format that recognized by that particular JMS provider. For example, when using WebSphere MQ as the JMS provider a JMS Queue Destination would be represented by the character string `queue://<queue manager>/<queue name>`
initialContext	CHARACTER	The Java Class name of the Initial Context Factory for the JMS provider that the JMSOutput or JMSReply node connects to.
JMSMessageID	CHARACTER	A JMS message ID is the value assigned by a JMS Provider when a message is sent to a JMS Queue or published to a JMS Topic. This value is retrieved from the JMS Message object after the message is sent or published.
JMSCorrelationID	CHARACTER	The JMS Message header property called JMSCorrelationID can be used to hold a value referencing some external information to be used for linking request with reply messages. When creating the DestinationData element in the LocalEnvironment this correlation ID value is obtained from the JMSCorrelationID message tree field in the folder OutputRoot.JMSTransport.Transport_Folders. Header_Values.

LocalEnvironment.Destination.JMSDestinationList fields

Transformation nodes can write data elements called DestinationData[n] in the LocalEnvironment folder called Destination.JMSDestinationList. The DestinationData elements are written with an array subscript format where the subscript is an integer that identifies an individual element in the destination list.

A JMSOutput node searches for DestinationData[n] entries in the LocalEnvironment if it has been configured to send to a destination list. The node sends an output message to each entry found in the destination list. The following table describes the format of a DestinationData element.

Element name	Element Data Type	Description
DestinationData	CHARACTER	The name of the JMS Queue to which the node sends an outgoing message or the JMS Topic to which the node publishes. This value can specified in the following formats: • A JNDI administered object (predefined in the JNDI bindings specified in the `Location JNDI bindings` node property). In this case, the format of the value is `jndi://<JNDI Object name>` • A character string that represents the JMS Destination name in an internal format that is recognized by that particular JMS provider. For example, when using WebSphere MQ as the JMS provider, a JMS Queue Destination is represented by the character string `queue://<queue manager>/<queue name>` • A complex element with DestinationName and DestinationType child elements that specify the name and type of the JMS Destination.
DestinationData.DestinationName	CHARACTER	A character string that represents the name of the JMS destination in an external format that is recognized by the JMS provider. If this field is specified, you must set the DestinationType field.
DestinationData.DestinationType	CHARACTER	The type of the JMS destination referred to in the DestinationName element. If the DestinationType field is set, you must set the DestinationName field. Set the DestinationType field to `Queue` or `Topic`.

Related concepts:

"Local environment tree structure" on page 1056
The local environment tree is a part of the logical message tree in which you can store information while the message flow processes the message.

Related tasks:

"Creating destination lists" on page 1477
Create a list of destinations to indicate where a message is sent.

"Accessing the local environment tree" on page 2463
The local environment tree has its own correlation name, LocalEnvironment, and you must use this name in all ESQL statements that refer to or set the content of this tree.

"Populating Destination in the local environment tree" on page 2467
Use the Destination subtree to set up the target destinations that are used by output nodes, the HTTPRequest node, the SOAPRequest node, the SOAPAsyncRequest node, and the RouteToLabel node. The following examples show how you can create and use an ESQL procedure to perform the task of setting up values for each of these uses.

Related reference:

"JMSOutput node" on page 4549
Use the JMSOutput node to send messages to JMS destinations.

"JMSReply node" on page 4562
Use the JMSReply node to send messages to JMS destinations.

Data types for an unstructured (BLOB) message:

A parser is supplied for the body of a message in the BLOB domain; it associates each field with a specific data type.

An unstructured (BLOB) message has the data types shown in the following table.

Data type of the element	Represented as
BLOB	BLOB
UnknownParserName	CHARACTER

If the broker cannot find a parser that corresponds to the domain that is requested by the user, the message is assigned to the BLOB parser, and the requested domain is preserved in the *UnknownParserName* field. If a BLOB parser is explicitly created, the UnknownParserName field is still present. It can contain the values of "BLOB" or "none" or can be the zero length string ("").

This information is used by the header integrity routine (described in “Parsers” on page 1072) to ensure that the semantic meaning of the message is preserved.

Related concepts:

“Parsers” on page 1072
A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from the internal message tree representation.

“Message flows overview” on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

“Manipulating messages in the BLOB domain” on page 2615
How to deal with messages that belong to the BLOB domain, and that are parsed by the BLOB parser.

“Developing ESQL” on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

“Developing Java” on page 2628
When you use the JavaCompute node, you customize it to determine the exact processing that it provides.

Related reference:

“Data types of fields and elements” on page 4237

The MQCFH parser:

The elements of the MQCFH parser are listed in this topic.

The root name for this parser is MQPCF. The class name is MQPCF.

The following table lists the elements native to the MQCFH header.

Element Name	Element Data Type	Element Attributes
Type	INTEGER	Name Value
StrucLength	INTEGER	Name Value
Version	INTEGER	Name Value
Command	INTEGER	Name Value
MsgSeqNumber	INTEGER	Name Value
Control	INTEGER	Name Value

Element Name	Element Data Type	Element Attributes
CompCode	INTEGER	Name Value
Reason	INTEGER	Name Value
ParameterCount	INTEGER	Name Value

For further information about this header and its contents, see the *Programmable Command Formats and Administration Interface* section of the WebSphere MQ Version 7 Information Center online.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Developing Java" on page 2628
When you use the JavaCompute node, you customize it to determine the exact processing that it provides.

Related reference:

"Data types of fields and elements" on page 4237

Related information:

WebSphere MQ Version 7 Information Center online

The MQCIH parser:

The elements of the MQCIH parser are listed in this topic.

The root name for this parser is MQCIH. The class name is MQCICS.

The following table lists the elements native to the MQCIH header.

Element Name	Element Data Type	Element Attributes
Format	CHARACTER	Name Value
Version	INTEGER	Name Value
Encoding	INTEGER	Name Value
CodedCharSetId	INTEGER	Name Value
Flags	INTEGER	Name Value
ReturnCode	INTEGER	Name Value
CompCode	INTEGER	Name Value
Reason	INTEGER	Name Value
UOWControl	INTEGER	Name Value
GetWaitInterval	INTEGER	Name Value
LinkType	INTEGER	Name Value
OutputDataLength	INTEGER	Name Value
FacilityKeepTime	INTEGER	Name Value

Element Name	Element Data Type	Element Attributes
ADSDescriptor	INTEGER	Name Value
ConversationalTask	INTEGER	Name Value
TaskEndStatus	INTEGER	Name Value
Facility	BLOB	Name Value
Function	CHARACTER	Name Value
AbendCode	CHARACTER	Name Value
Authenticator	CHARACTER	Name Value
Reserved1	CHARACTER	Name Value
ReplyToFormat	CHARACTER	Name Value
RemoteSysId	CHARACTER	Name Value
RemoteTransId	CHARACTER	Name Value
TransactionId	CHARACTER	Name Value
FacilityLike	CHARACTER	Name Value
AttentionId	CHARACTER	Name Value
StartCode	CHARACTER	Name Value
CancelCode	CHARACTER	Name Value
NextTransactionId	CHARACTER	Name Value
Reserved2	CHARACTER	Name Value
Reserved3	CHARACTER	Name Value
CursorPosition	INTEGER	Name Value
ErrorOffset	INTEGER	Name Value
InputItem	INTEGER	Name Value
Reserved4	INTEGER	Name Value

For further information about this header and its contents, see the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Developing Java" on page 2628
When you use the JavaCompute node, you customize it to determine the exact processing that it provides.

Related reference:

"Data types of fields and elements" on page 4237

Related information:

WebSphere MQ Version 7 Information Center online

The MQDLH parser:

The elements of the MQDLH parser are listed in this topic.

The root name for this parser is MQDLH. The class name is MQDEAD.

The following table lists the elements native to the MQDLH header.

Element Name	Element Data Type	Element Attributes
Format	CHARACTER	Name Value
Version	INTEGER	Name Value
Encoding	INTEGER	Name Value
CodedCharSetId	INTEGER	Name Value
Reason	INTEGER	Name Value
DestQName	CHARACTER	Name Value
DestQMgrName	CHARACTER	Name Value
PutApplType	INTEGER	Name Value
PutApplName	CHARACTER	Name Value
PutDate	TIMESTAMP/CHARACTER	Name Value
PutTime	TIMESTAMP/CHARACTER	Name Value

For further information about this header and its contents, see the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Developing Java" on page 2628
When you use the JavaCompute node, you customize it to determine the exact processing that it provides.

Related reference:

"Data types of fields and elements" on page 4237

Related information:

WebSphere MQ Version 7 Information Center online

The MQIIH parser:

The elements of the MQIIH parser are listed in this topic.

The root name for this parser is MQIIH. The class name is MQIMS.

The following table lists the elements native to the MQIIH header.

Element Name	Element Data Type	Element Attributes
Format	CHARACTER	Name Value
Version	INTEGER	Name Value
Encoding	INTEGER	Name Value
CodedCharSetId	INTEGER	Name Value
Flags	INTEGER	Name Value
LTermOverride	CHARACTER	Name Value
MFSMapName	CHARACTER	Name Value
ReplyToFormat	CHARACTER	Name Value
Authenticator	CHARACTER	Name Value
TranInstanceId	BLOB	Name Value
TranState	CHARACTER	Name Value
CommitMode	CHARACTER	Name Value
SecurityScope	CHARACTER	Name Value
Reserved	CHARACTER	Name Value

For further information about this header and its contents, see the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Developing Java" on page 2628
When you use the JavaCompute node, you customize it to determine the exact processing that it provides.

Related reference:

"Data types of fields and elements" on page 4237

Related information:

WebSphere MQ Version 7 Information Center online

The MQMD parser:

The elements of the MQMD parser are listed in this topic.

The root name for this parser is MQMD. The class name is MQHMD.

The following table lists the orphan elements adopted by the MQMD header.

Element Name	Element Data Type	Element Attributes
SourceQueue	CHARACTER	Name Value
Transactional	BOOLEAN	Name Value

The following table lists the elements native to the MQMD header.

Element Name	Element Data Type	Element Attributes
Format	CHARACTER	Name Value
Version	INTEGER	Name Value
Report	INTEGER	Name Value
MsgType	INTEGER	Name Value
Expiry[1]	INTEGER/GMTTIMESTAMP	Name Value
Feedback	INTEGER	Name Value
Encoding	INTEGER	Name Value
CodedCharSetId	INTEGER	Name Value
Priority	INTEGER	Name Value
Persistence	INTEGER	Name Value
MsgId	BLOB	Name Value
CorrelId	BLOB	Name Value
BackoutCount	INTEGER	Name Value
ReplyToQ	CHARACTER	Name Value
ReplyToQMgr	CHARACTER	Name Value
UserIdentifier	CHARACTER	Name Value
AccountingToken	BLOB	Name Value
ApplIdentityData	CHARACTER	Name Value
PutApplType	INTEGER	Name Value
PutApplName	CHARACTER	Name Value
PutDate	TIMESTAMP/CHARACTER	Name Value
PutTime	TIMESTAMP/CHARACTER	Name Value
ApplOriginData	CHARACTER	Name Value
GroupId	BLOB	Name Value
MsgSeqNumber	INTEGER	Name Value
Offset	INTEGER	Name Value
MsgFlags	INTEGER	Name Value
OriginalLength	INTEGER	Name Value

Note:

1. The Expiry field in the MQMD is a special case:
   - An INTEGER value represents an expiry interval in tenths of a second. If the Expiry field is set to -1, it represents an unlimited expiry interval (that is, the message never expires) If the Expiry field is a positive INTEGER, it represents an expiry interval of that number of tenths of a second (for example, if it is set to 4, it represents 4 tenths of a second, if it is set to 15, it represents one and a half seconds).
   - A GMTTIMESTAMP value represents a specific expiration time.

   If Expiry contains a GMTTIMESTAMP in the past, or an INTEGER of less than 1 (excluding -1), it is set to the value 1 (one tenth of a second, the minimum value).

For further information about this header and its contents, see the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Developing Java" on page 2628
When you use the JavaCompute node, you customize it to determine the exact processing that it provides.

"Accessing the MQMD header" on page 2455
Code ESQL statements to access the fields of the MQMD header.

Related reference:

"Data types of fields and elements" on page 4237

Related information:

WebSphere MQ Version 7 Information Center online

The MQMDE parser:

The elements of the MQMDE parser are listed in this topic.

The root name for this parser is MQMDE. The class name is MQHMDE.

The following table lists the elements native to the MQMDE header.

Element Name	Element Data Type	Element Attributes
Format	CHARACTER	Name Value
Version	INTEGER	Name Value
Encoding	INTEGER	Name Value
CodedCharSetId	INTEGER	Name Value
Flags	INTEGER	Name Value
GroupId	BLOB	Name Value
MsgSeqNumber	INTEGER	Name Value
Offset	INTEGER	Name Value
MsgFlags	INTEGER	Name Value
OriginalLength	INTEGER	Name Value

For further information about this header and its contents, see the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Developing Java" on page 2628
When you use the JavaCompute node, you customize it to determine the exact processing that it provides.

Related reference:

"Data types of fields and elements" on page 4237

Related information:

WebSphere MQ Version 7 Information Center online

The MQRFH parser:

The elements of the MQRFH parser are listed in this topic.

The root name for this parser is MQRFH. The class name is MQHRF.

The following table lists the elements native to the MQRFH header.

Element Name	Element Data Type	Element Attributes
Format	CHARACTER	Name Value
Version	INTEGER	Name Value
Encoding	INTEGER	Name Value
CodedCharSetId	INTEGER	Name Value
Flags	INTEGER	Name Value

Other name value elements might be present that contain information as parsed from or destined for the option buffer.

For further information about this header and its contents, see the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Developing Java" on page 2628
When you use the JavaCompute node, you customize it to determine the exact processing that it provides.

Related reference:

"Data types of fields and elements" on page 4237

Related information:

WebSphere MQ Version 7 Information Center online

The MQRFH2 and MQRFH2C parsers:

The MQRFH2 header can be parsed using either the MQRFH2 parser or the MQRFH2C compact parser.

The root names for these parsers are MQRFH2 and MQRFH2C. The class names are MQHRF2 and MQHRF2C.

The following table lists the elements that are required for the MQRFH2 header.

Element Name	Element Data Type	Element Attributes
Format	CHARACTER	Name Value
Version	INTEGER	Name Value
Encoding	INTEGER	Name Value
CodedCharSetId	INTEGER	Name Value
Flags	INTEGER	Name Value
NameValueCCSID	INTEGER	Name Value

Other name and child name value elements might be present that contain information that is parsed from, or destined for, the option buffer. For further information about this header and its contents, see "MQRFH2 header" on page 6397 and the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Accessing the MQRFH2 header" on page 2456
Code ESQL statements to access the fields of the MQRFH2 header.

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Developing Java" on page 2628
When you use the JavaCompute node, you customize it to determine the exact processing that it provides.

Related reference:

"Data types of fields and elements" on page 4237

"MQRFH2 header" on page 6397
The MQRFH2 header is used to pass messages to and from a message broker that belongs to WebSphere Message Broker.

Related information:

WebSphere MQ Version 7 Information Center online

The MQRMH parser:

The elements of the MQRMH parser are listed in this topic.

The root name for this parser is MQRMH. The class name is MQHREF.

The following table lists the elements native to the MQRMH header.

Element Name	Element Data Type	Element Attributes
Format	CHARACTER	Name Value
Version	INTEGER	Name Value
Encoding	INTEGER	Name Value
CodedCharSetId	INTEGER	Name Value
Flags	INTEGER	Name Value
ObjectType	CHARACTER	Name Value
ObjectInstanceId	BLOB	Name Value
SrcEnv	CHARACTER[1]	Name Value
SrcName	CHARACTER[2]	Name Value
DestEnv	CHARACTER[3]	Name Value
DestName	CHARACTER[4]	Name Value
DataLogicalLength	INTEGER	Name Value
DataLogicalOffset	INTEGER	Name Value
DataLogicalOffset2	INTEGER	Name Value

Notes:
1. This field represents both SrcEnvLength and Offset
2. This field represents both SrcNameLength and Offset
3. This field represents both DestEnvLength and Offset
4. This field represents both DestNameLength and Offset

For further information about this header and its contents, see the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Developing Java" on page 2628
When you use the JavaCompute node, you customize it to determine the exact processing that it provides.

Related reference:

"Data types of fields and elements" on page 4237

Related information:

WebSphere MQ Version 7 Information Center online

The MQSAPH parser:

The elements of the MQSAPH parser are listed in this topic.

The root name for this parser is MQSAPH. The class name is MQHSAP.

The following table lists the elements native to the MQSAPH header.

Element Name	Element Data Type	Element Attributes
Format	CHARACTER	Name Value
Version	INTEGER	Name Value
Encoding	INTEGER	Name Value
CodedCharSetId	INTEGER	Name Value
Flags	INTEGER	Name Value
Client	CHARACTER	Name Value
Language	CHARACTER	Name Value
HostName	CHARACTER	Name Value
UserId	CHARACTER	Name Value
Password	CHARACTER	Name Value
SystemNumber	CHARACTER	Name Value
Reserved	BLOB	Name Value

For further information about this header and its contents, see the WebSphere MQ Version 7 Information Center online.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Developing Java" on page 2628
When you use the JavaCompute node, you customize it to determine the exact processing that it provides.

Related reference:

"Data types of fields and elements" on page 4237

Related information:

WebSphere MQ Version 7 Information Center online

The MQWIH parser:

The elements of the MQWIH parser are listed in this topic.

The root name for this parser is MQWIH. The class name is MQHWIH.

The following table lists the elements native to the MQWIH header.

Element Name	Element Data Type	Element Attributes
Format	CHARACTER	Name Value
Version	INTEGER	Name Value
Encoding	INTEGER	Name Value
CodedCharSetId	INTEGER	Name Value

Element Name	Element Data Type	Element Attributes
Flags	INTEGER	Name Value
ServiceName	CHARACTER	Name Value
ServiceStep	CHARACTER	Name Value
MsgToken	BLOB	Name Value
Reserved	CHARACTER	Name Value

For further information about this header and its contents, see the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Developing Java" on page 2628
When you use the JavaCompute node, you customize it to determine the exact processing that it provides.

Related reference:

"Data types of fields and elements" on page 4237

Related information:

WebSphere MQ Version 7 Information Center online

The SMQ_BMH parser:

The elements of the SMQ_BMH parser are listed in this topic.

The root name for this parser is SMQ_BMH. The class name is SMQBAD.

The following table lists the elements native to the SMQ_BMH header.

Element Name	Element Data Type	Element Attributes
Format	CHARACTER	Name Value
Version	INTEGER	Name Value
Encoding	INTEGER	Name Value
CodedCharSetId	INTEGER	Name Value
ErrorType	INTEGER	Name Value
Reason	INTEGER	Name Value
PutApplType	INTEGER	Name Value
PutApplName	CHARACTER	Name Value
PutDate	TIMESTAMP/CHARACTER	Name Value
PutTime	TIMESTAMP/CHARACTER	Name Value

Related concepts:

“Message flows overview” on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

“Developing ESQL” on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

“Developing Java” on page 2628
When you use the JavaCompute node, you customize it to determine the exact processing that it provides.

Related reference:

“Data types of fields and elements” on page 4237

Related information:

WebSphere MQ Version 7 Information Center online

XML constructs

A self-defining XML message carries the information about its content and structure within the message in the form of a document that adheres to the XML specification. Its definition is not held anywhere else.

When the broker receives an XML message, it interprets the message using the generic XML parser, and created an internal message tree structure according to the XML definitions contained within that message.

A self-defining message is also known as a generic XML message. It does not have a recorded format.

The information provided with WebSphere Message Broker does not provide a full definition or description of XML terminology, concepts, and message constructs: it is a summary that highlights aspects that are important when you use XML messages with brokers and message flows.

For further information about XML, see the developerWorks Web site.

Example XML message:
The name elements used in this description (for example, XmlDecl) are provided by WebSphere Message Broker, and are referred to as field type constants. They are available for symbolic use within the ESQL that defines the processing of message content performed by the nodes, such as a Filter node, within a message flow. They are not part of the XML specification.

A simple XML message might take the form:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE s1 PUBLIC "http://www.ibm.com/example.dtd" "example.dtd">
<s1>.........</s1>
```

The corresponding syntax element tree (top level elements only) is shown in the following diagram:

The WhiteSpace elements within the tree are there because of the line breaks in the original XML document, and have no business meaning. White space is used in XML for readability; if you process XML messages that contain line breaks (as shown above), blanks lines, or spaces between tags, these all appear as elements in the message tree.

WhiteSpace within an XML element (between start and end tags) has business meaning and is represented using the Content syntax element. See "XML WhiteSpace and DocTypeWhiteSpace" on page 4287 for more information.

The field type constants for XML name elements (for example, Element and XmlDecl) equate to a constant value of the form `0x01000000`. You can see these constants in the output created by the Trace node when a message, or a portion of the message, is traced.

Related concepts:

"Correlation names" on page 1069
A correlation name is a field reference that identifies a well-defined starting point in the logical message tree and is used in field references to describe a standard part of the tree format.

Related tasks:

"Manipulating messages in the XML domains" on page 2535
You can manipulate messages in the XML, XMLNS, and XMLNSC domains.

"Writing ESQL" on page 2413
How you can use ESQL to customize nodes.

Related reference:

"The XML declaration"
The beginning of an XML message can contain an XML declaration.

"The XML message body" on page 4262
Every XML message must have a body. The body comprises a top-level XML element that contains all the message data.

"XML document type declaration" on page 4271
The document type declaration (DTD) of an XML message is represented by a syntax element of type DocTypeDecl and its children and descendants. These comprise the DOCTYPE construct.

"XML DocTypeDecl" on page 4272
The DocTypeDecl represents the DOCTYPE declaration.

The XML declaration:

The beginning of an XML message can contain an XML declaration.

The following is an example of a declaration:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE s1 PUBLIC "http://www.ibm.com/example.dtd" "example.dtd">
<s1>.........</s1>
```

The XML declaration includes the following field type constants:

- "XML encoding" on page 4259
- "XML standalone" on page 4259
- "XML version" on page 4260
- "XMLDecl" on page 4261

"XML declaration example" on page 4261 includes another example of an XML declaration and the tree structure it forms.

Related concepts:

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Manipulating messages in the XML domains" on page 2535
You can manipulate messages in the XML, XMLNS, and XMLNSC domains.

"Adding an XML wire format" on page 2853
You can add an XML wire format physical format layer to a message set by using the Message Set editor.

Related reference:

"XML declaration example" on page 4261
The following example shows an XML declaration in an XML document.

"XML document type declaration" on page 4271
The document type declaration (DTD) of an XML message is represented by a syntax element of type DocTypeDecl and its children and descendants. These comprise the DOCTYPE construct.

XML encoding:

The encoding element is a value element and is always a child of the XmlDecl element.

The value of the encoding element is a string that corresponds to the value of the encoding string in the declaration. In the following example, the encoding element has a value of `UTF-8`.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE s1 PUBLIC "http://www.ibm.com/example.dtd" "example.dtd">
<s1>.........</s1>
```

You cannot specify WebSphere MQ encodings in this element.

Related concepts:

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Manipulating messages in the XML domains" on page 2535
You can manipulate messages in the XML, XMLNS, and XMLNSC domains.

"Adding an XML wire format" on page 2853
You can add an XML wire format physical format layer to a message set by using the Message Set editor.

XML standalone:

The XML standalone element defines the existence of an externally-defined DTD.

It is a value element and stores the data corresponding to the value of the standalone string in the declaration. It is always a child of the XmlDecl element. Valid values for the standalone element are `yes` and `no`. The following is an example of this:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE s1 PUBLIC "http://www.ibm.com/example.dtd" "example.dtd">
<s1>.........</s1>
```

A value of `no` indicates that this XML document is not standalone and depends on an externally-defined DTD. A value of `yes` indicates that the XML document is self-contained. However, the current release of WebSphere Message Broker does not resolve externally-defined DTDs, so the setting of standalone is irrelevant and is ignored.

Related concepts:

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Adding an XML wire format" on page 2853
You can add an XML wire format physical format layer to a message set by using the Message Set editor.

"Manipulating messages in the XML domains" on page 2535
You can manipulate messages in the XML, XMLNS, and XMLNSC domains.

XML version:

The XML version element is a value element and stores the data corresponding to the version string in the declaration.

It is always a child of the XmlDecl element. In the following example, the version element contains the string value `1.0`:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE s1 PUBLIC "http://www.ibm.com/example.dtd" "example.dtd">
<s1>.........</s1>
```

Related concepts:

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Manipulating messages in the XML domains" on page 2535
You can manipulate messages in the XML, XMLNS, and XMLNSC domains.

"Adding an XML wire format" on page 2853
You can add an XML wire format physical format layer to a message set by using the Message Set editor.

XMLDecl:

XMLDecl is a name element that corresponds to the XML declaration itself.

The XmlDecl element is a child of the XML parser and is written first to a bit stream. Although the XMLDecl element is a named element, its name has no relevance. The following shows an example:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE s1 PUBLIC "http://www.ibm.com/example.dtd" "example.dtd">
<s1>.........</s1>
```

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

Related tasks:

"Manipulating messages in the XML domains" on page 2535
You can manipulate messages in the XML, XMLNS, and XMLNSC domains.

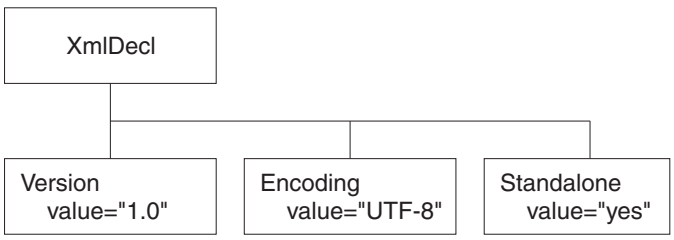
"Adding an XML wire format" on page 2853
You can add an XML wire format physical format layer to a message set by using the Message Set editor.

XML declaration example:

The following example shows an XML declaration in an XML document.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

The following figure shows the tree structure that is created from the declaration:



Related concepts:

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Manipulating messages in the XML domains" on page 2535
You can manipulate messages in the XML, XMLNS, and XMLNSC domains.

"Adding an XML wire format" on page 2853
You can add an XML wire format physical format layer to a message set by using the Message Set editor.

Related reference:

"The XML declaration" on page 4258
The beginning of an XML message can contain an XML declaration.

The XML message body:

Every XML message must have a body. The body comprises a top-level XML element that contains all the message data.

The body contains complex XML markup, which translates to many syntax element types in the parsed tree. Each syntax element type is introduced here, with a series of example XML fragments.

The following common element types are discussed:
- "XML element" on page 4267
- "XML attribute" on page 4264
- "XML content" on page 4267

"XML message body example: elements, attributes, and content" on page 4269 provides an example of an XML message body and the tree structure that is created from it using the syntax elements types listed above.

More complex XML messages might use some of the following syntax element types:
- "XML CDataSection" on page 4265
- "XML EntityReferenceStart and EntityReferenceEnd" on page 4268
- "XML comment" on page 4266
- "XML ProcessingInstruction" on page 4270
- "XML AsisElementContent" on page 4263
- "XML BitStream" on page 4264

Related concepts:

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Manipulating messages in the XML domains" on page 2535
You can manipulate messages in the XML, XMLNS, and XMLNSC domains.

"Adding an XML wire format" on page 2853
You can add an XML wire format physical format layer to a message set by using the Message Set editor.

Related reference:

"XML declaration example" on page 4261
The following example shows an XML declaration in an XML document.

"XML document type declaration" on page 4271
The document type declaration (DTD) of an XML message is represented by a syntax element of type DocTypeDecl and its children and descendants. These comprise the DOCTYPE construct.

XML AsisElementContent:

AsisElementContent is a special value syntax element that is used to precisely control generated XML.

The AsisElementContent syntax element is a special value element. The element is used to precisely control the XML generated in an output message without the safeguards of the Element, Attribute, and Content syntax elements. If you use AsisElementContent, you must ensure that the output message is well-formed XML.

You might choose to use this syntax element if, for example, you want to suppress the usual behavior in which occurrences of ampersand (&), less than (<), greater than (>), quotation mark ("), and apostrophe (') are replaced by the predefined XML entities `&`, `<`, `>`, `&quot;`, and `&apos;`.

The following example illustrates the use of AsisElementContent. The statement:

```
Set OutputRoot.XMLNS.(XML.Element)Message.(XML.Content) = '<rawMarkup>';
```

generates the following XML in an output message:

```
<Message>&lt;rawMarkup&gt;</Message>
```

However, the following statement:

```
Set OutputRoot.XMLNS.(XML.Element)Message.(XML.AsisElementContent) = '<rawMarkup>';
```

generates the following output message:

```
<Message><rawMarkup></Message>
```

These examples show that the value of an AsisElementContent syntax element is not modified before it is written to the output message.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"The XML message body" on page 4262
Every XML message must have a body. The body comprises a top-level XML element that contains all the message data.

"XML element" on page 4267

"XML attribute" on page 4264

"XML content" on page 4267
The content element represents character data that occurs in the XML message.

"XML message body example: elements, attributes, and content" on page 4269
The XML document contains elements, attributes, and content.

"XML CDataSection" on page 4265
The CDataSection value element represents CData sections in the XML message.

"XML EntityReferenceStart and EntityReferenceEnd" on page 4268
The EntityReferenceStart and EntityReferenceEnd elements are used to store entity references that occur in the XML message.

"XML comment" on page 4266
An XML comment encountered outside the document type declaration is represented by the Comment value syntax element. It contains the comment text from the XML message.

"XML ProcessingInstruction" on page 4270
ProcessingInstruction is a syntax element used with XML processing instructions.

"XML BitStream"
BitStream is a specialized name-value element designed to aid the processing of very large messages.

XML attribute:
This syntax element is the default name-value element supported by the XML parser. Use it to represent the attributes that are associated with its parent element. The name and value of the syntax element correspond to the name and value of the attribute that is being represented. Attribute elements have no children, and must always be children of an element.

When attributes are written to a message, occurrences of ampersand (&), less than (<), greater than (>), double quotation mark ("), and apostrophe (') within the attribute value are replaced by the predefined XML entities `&`, `<`, `>`, `&quot;`, and `&apos;`.

The `attr` element is also supported for compatibility with earlier versions of the product.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"The XML message body" on page 4262
Every XML message must have a body. The body comprises a top-level XML element that contains all the message data.

"XML element" on page 4267

"XML message body example: elements, attributes, and content" on page 4269
The XML document contains elements, attributes, and content.

XML BitStream:

BitStream is a specialized name-value element designed to aid the processing of very large messages.

The XML BitStream syntax element is a name-value element. When writing an XML message, the value of the BitStream element is written directly into the message, and the name is not important. The BitStream element might be the only element in the message tree.

The value of the element must be of type BLOB; any other data type generates an error while writing the element. Ensure that the content of the element is appropriate for use in the output message.

Use of the BitStream element is similar to the AsisElementContent element, except that the AsisElementContent type converts its value into a string, whereas the BitStream element uses its BLOB value directly. This is a specialized element designed to aid processing of very large messages.

The following ESQL excerpts demonstrate a typical use for this element. First, declare the element:

```
DECLARE StatementBitStream BLOB
```

Initialize the contents of StatementBitStream from an appropriate source, such as an input message. If the source field is not of type BLOB, use the CAST statement to convert the contents to BLOB. Then create the new field in the output message, for example:

```
CREATE LASTCHILD OF resultCursor
 Type XML.BitStream
 NAME 'StatementBitStream'
 VALUE StatementBitstream;
```

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"The XML message body" on page 4262
Every XML message must have a body. The body comprises a top-level XML element that contains all the message data.

"XML AsisElementContent" on page 4263
AsisElementContent is a special value syntax element that is used to precisely control generated XML.

"XML element" on page 4267

"XML attribute" on page 4264

"XML content" on page 4267
The content element represents character data that occurs in the XML message.

"XML comment" on page 4266
An XML comment encountered outside the document type declaration is represented by the Comment value syntax element. It contains the comment text from the XML message.

"CAST function" on page 5245

"XML message body example: elements, attributes, and content" on page 4269
The XML document contains elements, attributes, and content.

XML CDataSection:

The CDataSection value element represents CData sections in the XML message.

The content of the CDataSection element is the value of the CDataSection element without the `<![CDATA[` that marks its beginning and the]]> that marks its end.

For example, the following Cdata section:

```
<![CDATA[<greeting>Hello, world!</greeting>]]>
```

is represented by a CDataSection element with a string value of:

```
"<greeting>Hello, world!</greeting>"
```

Unlike Content, occurrences of <,>, &, ", and ' are not translated to their escape sequences when the CDataSection is written out to a serialized message (bit stream).

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"The XML message body" on page 4262
Every XML message must have a body. The body comprises a top-level XML element that contains all the message data.

"XML element" on page 4267

"XML message body example: elements, attributes, and content" on page 4269
The XML document contains elements, attributes, and content.

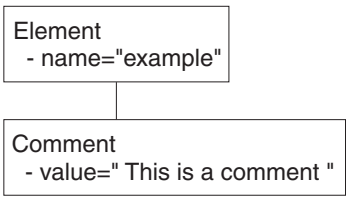
XML comment:

An XML comment encountered outside the document type declaration is represented by the Comment value syntax element. It contains the comment text from the XML message.

If the value of the element contains the character sequence `-->`, the sequence is replaced with the text `-->`. This ensures that the contents of the comment cannot prematurely terminate the comment. Occurrences of <, >, &, ", and ' are not translated to their escape sequences.

The following are examples of the XML comment in an XML document and in tree structure form:

```
<example><!-- This is a comment --></example>
```



Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"The XML message body" on page 4262
Every XML message must have a body. The body comprises a top-level XML element that contains all the message data.

"XML message body example: elements, attributes, and content" on page 4269
The XML document contains elements, attributes, and content.

"XML document type declaration" on page 4271
The document type declaration (DTD) of an XML message is represented by a syntax element of type DocTypeDecl and its children and descendants. These comprise the DOCTYPE construct.

XML content:

The content element represents character data that occurs in the XML message.

This syntax element is the default value element supported by the XML parser. Use content to represent character data (including white space characters) that is part of the element content. There might be many content elements as children of a single element, in which case they are separated by other syntax element types such as nested elements or attributes.

When content is written to a message, occurrences of ampersand (&), less than (<), greater than (>), double quotation mark ("), and apostrophe (') are replaced by the predefined XML entities `&`, `<`, `>`, `&quot;`, and `&apos;`.

The pcdata element is also supported for compatibility with earlier versions of the product.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"The XML message body" on page 4262
Every XML message must have a body. The body comprises a top-level XML element that contains all the message data.

"XML element"

"XML attribute" on page 4264

"XML message body example: elements, attributes, and content" on page 4269
The XML document contains elements, attributes, and content.

XML element:
This syntax element is the default name element supported by the XML parser, and is one of the most common elements. The name of the syntax element corresponds to the name of the XML element in the message. This element can have many children, including attributes, elements, and content.

The tag element is also supported for backward compatibility.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

Chapter 9, “Developing message flow applications,” on page 1019
Develop message flows to process your business messages and data.

Related reference:

“The XML message body” on page 4262
Every XML message must have a body. The body comprises a top-level XML element that contains all the message data.

“XML attribute” on page 4264

“XML content” on page 4267
The content element represents character data that occurs in the XML message.

“XML message body example: elements, attributes, and content” on page 4269
The XML document contains elements, attributes, and content.

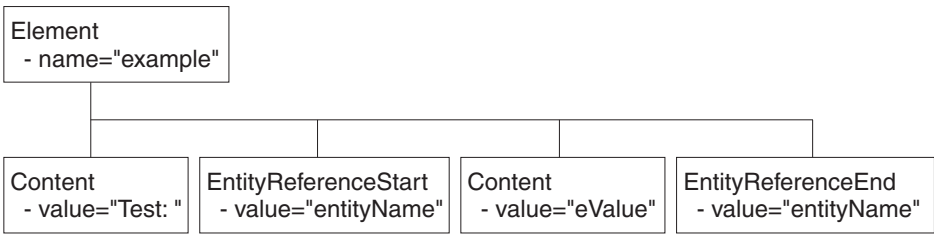
XML EntityReferenceStart and EntityReferenceEnd:

The EntityReferenceStart and EntityReferenceEnd elements are used to store entity references that occur in the XML message.

When an entity reference is encountered in the XML message, both the expanded form and the original entity name are stored in the syntax element tree. The name of the entity is stored as the value of the EntityReferenceStart and EntityReferenceEnd syntax elements, and any syntax elements between contain the entity expansion.

The following examples show the XML entity references in an XML document and in tree structure form:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE example [ <!ENTITY entityName "eValue"> ]>
<example>Test: &entityName;</example>
```



The XML declaration and the document type declaration are not shown here. Refer to “The XML declaration” on page 4258 and “XML document type declaration” on page 4271 for details of those sections of the syntax element tree.

Related concepts:

“Message flows overview” on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

Chapter 9, “Developing message flow applications,” on page 1019
Develop message flows to process your business messages and data.

Related reference:

“The XML message body” on page 4262
Every XML message must have a body. The body comprises a top-level XML element that contains all the message data.

"XML element" on page 4267

"XML attribute" on page 4264

"XML content" on page 4267
The content element represents character data that occurs in the XML message.

"XML message body example: elements, attributes, and content"
The XML document contains elements, attributes, and content.

"The XML declaration" on page 4258
The beginning of an XML message can contain an XML declaration.

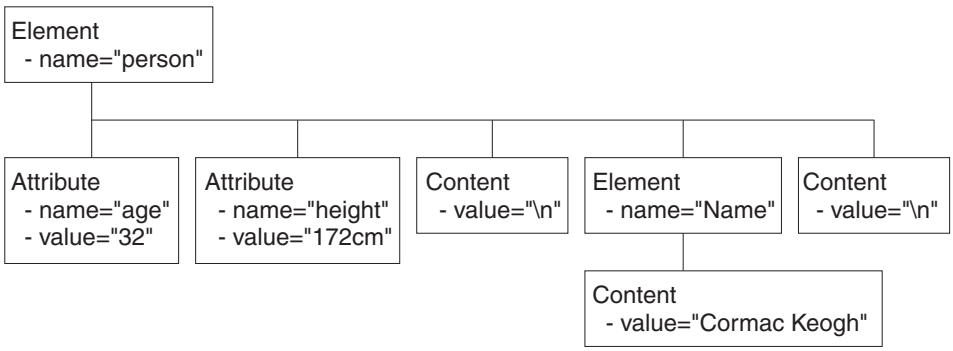
"XML document type declaration" on page 4271
The document type declaration (DTD) of an XML message is represented by a syntax element of type DocTypeDecl and its children and descendants. These comprise the DOCTYPE construct.

XML message body example: elements, attributes, and content:

The XML document contains elements, attributes, and content.

The following are examples of an XML message body in an XML document and in tree structure form.

```
<Person age="32" height="172cm">
<Name>Cormac Keogh</Name>
</Person>
```



Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"The XML message body" on page 4262
Every XML message must have a body. The body comprises a top-level XML element that contains all the message data.

"XML element" on page 4267

"XML attribute" on page 4264

"XML content" on page 4267
The content element represents character data that occurs in the XML message.

XML ProcessingInstruction:

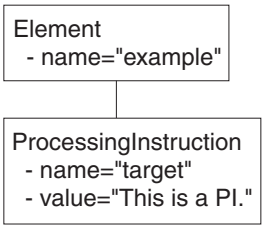
ProcessingInstruction is a syntax element used with XML processing instructions.

An XML processing instruction, encountered outside the document type declaration, is represented by the ProcessingInstruction syntax element. This is a name-value element; the name of the syntax element is the processing instruction target name, and the value of the syntax element is the character data of the processing instruction. The value of the syntax element must not be empty. The name cannot be XML in either uppercase or lowercase.

If the value of the element contains the character sequence ?>, the sequence is replaced with the text ?>. This ensures that the content of the processing instruction cannot prematurely end the processing instruction. Occurrences of <,>, &, ", and ' are not translated to their escape sequences.

Examples of the XML ProcessingInstruction in an XML document and in tree structure form are shown in the following:

```
<example><?target This is a PI.?></example>
```



Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"The XML message body" on page 4262
Every XML message must have a body. The body comprises a top-level XML element that contains all the message data.

"XML element" on page 4267

"XML attribute" on page 4264

"XML content" on page 4267
The content element represents character data that occurs in the XML message.

"XML message body example: elements, attributes, and content" on page 4269
The XML document contains elements, attributes, and content.

"XML CDataSection" on page 4265
The CDataSection value element represents CData sections in the XML message.

"XML EntityReferenceStart and EntityReferenceEnd" on page 4268
The EntityReferenceStart and EntityReferenceEnd elements are used to store entity references that occur in the XML message.

"XML comment" on page 4266
An XML comment encountered outside the document type declaration is represented by the Comment value syntax element. It contains the comment text from the XML message.

"XML AsisElementContent" on page 4263
AsisElementContent is a special value syntax element that is used to precisely control generated XML.

"XML BitStream" on page 4264
BitStream is a specialized name-value element designed to aid the processing of very large messages.

XML document type declaration:

The document type declaration (DTD) of an XML message is represented by a syntax element of type DocTypeDecl and its children and descendants. These comprise the DOCTYPE construct.

Only internal (inline) DTD subsets are represented in the syntax element tree. An inline DTD is a DTD that is declared within the XML document itself. It can be a complete DTD definition, or can extend the definition in an external DTD.

External DTD subsets (identified by the SystemID or PublicId elements described later in this section) can be referenced in the message, but those referenced are not resolved by the broker.

Field type constants are defined by WebSphere Message Broker:
- DocTypeDecl
- NotationDecl
- Entities
- ElementDef
- AttributeList
- AttributeDef
- DocTypePI
- WhiteSpace and DocTypeWhiteSpace
- DocTypeComment

DTD example is an example of an XML DTD.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"The XML declaration" on page 4258
The beginning of an XML message can contain an XML declaration.

"The XML message body" on page 4262
Every XML message must have a body. The body comprises a top-level XML element that contains all the message data.

"XML DocTypeDecl" on page 4272
The DocTypeDecl represents the DOCTYPE declaration.

"XML NotationDecl" on page 4275
The NotationDecl element represents a notation declaration in an XML message.

"XML entities" on page 4275

"XML ElementDef" on page 4281
The ElementDef element represents the <!ELEMENT construct in a DTD.

"XML AttributeList" on page 4281
The AttributeList name element represents the <!ATTLIST construct in a DTD.

"XML AttributeDef" on page 4282
The AttributeDef name element describes the definition of an attribute within an <!ATTLIST construct.

"XML DocTypePI" on page 4286
The DocTypePI element represents a processing instruction found within the DTD.

"XML WhiteSpace and DocTypeWhiteSpace" on page 4287
The WhiteSpace and DocTypeWhiteSpace XML elements represent white space characters in the message.

"XML DocTypeComment" on page 4286
Comments in the XML DTD are represented by the DocTypeComment element.

"XML DTD example" on page 4288

XML DocTypeDecl:

The DocTypeDecl represents the DOCTYPE declaration.

DocTypeDecl is a named element and is a child of the XML parser. DocTypeDecl is written to the bit stream before the element that represents the body of the document during serialization. The following attributes can be specified within this element:

- IntSubset
- PublicId
- SystemId

The following example is included in DTD example:

```
<!DOCTYPE test PUBLIC "//this/is/a/URI/test" "test.dtd" [
...
...
]>
```

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"XML document type declaration" on page 4271
The document type declaration (DTD) of an XML message is represented by a syntax element of type DocTypeDecl and its children and descendants. These comprise the DOCTYPE construct.

"XML IntSubset" on page 4273
The IntSubset element groups all elements that represent the DTD constructs contained in the internal subset of the message.

"XML PublicId"
PublicId is an element that represents a public identifier in an XML message.
"XML SystemId" on page 4274
SystemId is a value element that represents a system identifier in an XML message.

"XML DTD example" on page 4288

XML IntSubset:

The IntSubset element groups all elements that represent the DTD constructs contained in the internal subset of the message.

Although the IntSubset element is a named element, its name is not relevant.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"XML document type declaration" on page 4271
The document type declaration (DTD) of an XML message is represented by a syntax element of type DocTypeDecl and its children and descendants. These comprise the DOCTYPE construct.
"XML DocTypeDecl" on page 4272
The DocTypeDecl represents the DOCTYPE declaration.
"XML PublicId"
PublicId is an element that represents a public identifier in an XML message.
"XML SystemId" on page 4274
SystemId is a value element that represents a system identifier in an XML message.

"XML DTD example" on page 4288

XML PublicId:

PublicId is an element that represents a public identifier in an XML message.

The PublicId element can be part of a DocTypeDecl, NotationDecl, or UnparsedEntityDecl element. The value of the PublicId element is typically a URL. A public identifier of the form `PUBLIC "//this/is/a/URI/test"` has a string value of `//this/is/a/URI/test`.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"XML document type declaration" on page 4271
The document type declaration (DTD) of an XML message is represented by a syntax element of type DocTypeDecl and its children and descendants. These comprise the DOCTYPE construct.

"XML DocTypeDecl" on page 4272
The DocTypeDecl represents the DOCTYPE declaration.

"XML IntSubset" on page 4273
The IntSubset element groups all elements that represent the DTD constructs contained in the internal subset of the message.

"XML SystemId"
SystemId is a value element that represents a system identifier in an XML message.

"XML NotationDecl" on page 4275
The NotationDecl element represents a notation declaration in an XML message.

"XML UnparsedEntityDecl" on page 4280
An UnparsedEntityDecl is used to include data in an XML document that is not well-formed XML, and is not parsed by an XML processor.

"XML DTD example" on page 4288

XML SystemId:

SystemId is a value element that represents a system identifier in an XML message.

SystemId can be part of a DocTypeDecl, NotationDecl, or UnparsedEntityDecl element. The value of the SystemId is a URI, and is typically a URL or the name of a file on the current system. A system identifier of the form `SYSTEM "Note.dtd"` has a string value of `Note.dtd`.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"XML document type declaration" on page 4271
The document type declaration (DTD) of an XML message is represented by a syntax element of type DocTypeDecl and its children and descendants. These comprise the DOCTYPE construct.

"XML DocTypeDecl" on page 4272
The DocTypeDecl represents the DOCTYPE declaration.

"XML IntSubset" on page 4273
The IntSubset element groups all elements that represent the DTD constructs contained in the internal subset of the message.

"XML PublicId" on page 4273
PublicId is an element that represents a public identifier in an XML message.

"XML NotationDecl" on page 4275
The NotationDecl element represents a notation declaration in an XML message.

"XML UnparsedEntityDecl" on page 4280
An UnparsedEntityDecl is used to include data in an XML document that is not well-formed XML, and is not parsed by an XML processor.

"XML DTD example" on page 4288

XML NotationDecl:

The NotationDecl element represents a notation declaration in an XML message.

NotationDecl is a name element whose name corresponds to the name given with the notation declaration. It must have a SystemId as a child and it can optionally have a child element of type PublicId. For example:

```
<!NOTATION gif SYSTEM "image.gif">
```

The name of the NotationDecl is `gif`.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"XML document type declaration" on page 4271
The document type declaration (DTD) of an XML message is represented by a syntax element of type DocTypeDecl and its children and descendants. These comprise the DOCTYPE construct.

"XML DocTypeDecl" on page 4272
The DocTypeDecl represents the DOCTYPE declaration.

"XML PublicId" on page 4273
PublicId is an element that represents a public identifier in an XML message.

"XML SystemId" on page 4274
SystemId is a value element that represents a system identifier in an XML message.

"XML DTD example" on page 4288

XML entities:

Entities in the DTD are represented by one of six named element types described in the following topics:

- EntityDecl
- EntityDeclValue
- ExternalParameterEntityDecl
- ExternalEntityDecl
- ParameterEntityDecl
- UnparsedEntityDecl

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"XML document type declaration" on page 4271
The document type declaration (DTD) of an XML message is represented by a syntax element of type DocTypeDecl and its children and descendants. These comprise the DOCTYPE construct.

"XML DocTypeDecl" on page 4272
The DocTypeDecl represents the DOCTYPE declaration.

"XML EntityDecl"
The EntityDecl element represents a general entity and is declared in the internal subset of the DTD.

"XML ParameterEntityDecl" on page 4279
The ParameterEntityDecl represents a parameter entity definition in the internal subset of the DTD.

"XML ExternalParameterEntityDecl" on page 4278
The ExternalParameterEntityDecl element represents a parameter entity definition where the entity definition is contained externally to the current message.

"XML ExternalEntityDecl" on page 4279
The ExternalEntityDecl element represents a general entity where the entity definition is contained externally to the current message.

"XML UnparsedEntityDecl" on page 4280
An UnparsedEntityDecl is used to include data in an XML document that is not well-formed XML, and is not parsed by an XML processor.

"XML EntityDeclValue" on page 4277
The EntityDeclValue element represents the value of an EntityDecl or ParameterEntityDecl defined internally in the DOCTYPE construct.

"XML DTD example" on page 4288

XML EntityDecl:

The EntityDecl element represents a general entity and is declared in the internal subset of the DTD.

The EntityDecl is a named element and has a single child element, which is of type EntityDeclValue.

An entity declaration of the form:

```
<!ENTITY bookTitle "User Guide">
```

has an EntityDecl element of name bookTitle and a child element of type EntityDeclValue with a string value of `User Guide`.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"XML document type declaration" on page 4271
The document type declaration (DTD) of an XML message is represented by a syntax element of type DocTypeDecl and its children and descendants. These comprise the DOCTYPE construct.

"XML DocTypeDecl" on page 4272
The DocTypeDecl represents the DOCTYPE declaration.
"XML NotationDecl" on page 4275
The NotationDecl element represents a notation declaration in an XML message.
"XML entities" on page 4275
"XML ParameterEntityDecl" on page 4279
The ParameterEntityDecl represents a parameter entity definition in the internal subset of the DTD.
"XML ExternalParameterEntityDecl" on page 4278
The ExternalParameterEntityDecl element represents a parameter entity definition where the entity definition is contained externally to the current message.
"XML ExternalEntityDecl" on page 4279
The ExternalEntityDecl element represents a general entity where the entity definition is contained externally to the current message.
"XML UnparsedEntityDecl" on page 4280
An UnparsedEntityDecl is used to include data in an XML document that is not well-formed XML, and is not parsed by an XML processor.
"XML EntityDeclValue"
The EntityDeclValue element represents the value of an EntityDecl or ParameterEntityDecl defined internally in the DOCTYPE construct.
"XML ElementDef" on page 4281
The ElementDef element represents the `<!ELEMENT` construct in a DTD.
"XML AttributeList" on page 4281
The AttributeList name element represents the `<!ATTLIST` construct in a DTD.
"XML AttributeDef" on page 4282
The AttributeDef name element describes the definition of an attribute within an `<!ATTLIST` construct.
"XML DocTypePI" on page 4286
The DocTypePI element represents a processing instruction found within the DTD.

"XML WhiteSpace and DocTypeWhiteSpace" on page 4287
The WhiteSpace and DocTypeWhiteSpace XML elements represent white space characters in the message.
"XML DocTypeComment" on page 4286
Comments in the XML DTD are represented by the DocTypeComment element.
"XML DTD example" on page 4288

XML EntityDeclValue:

The EntityDeclValue element represents the value of an EntityDecl or ParameterEntityDecl defined internally in the DOCTYPE construct.

The EntityDeclValue is always a child of either an EntityDecl element or a ParameterEntityDecl element, and is a value element. For the following entity:

```
<!ENTITY  bookTitle "User Guide">
```

the EntityDeclValue element has the string value `User Guide`.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

Chapter 9, “Developing message flow applications,” on page 1019
Develop message flows to process your business messages and data.

Related reference:

“XML document type declaration” on page 4271
The document type declaration (DTD) of an XML message is represented by a syntax element of type DocTypeDecl and its children and descendants. These comprise the DOCTYPE construct.

“XML DocTypeDecl” on page 4272
The DocTypeDecl represents the DOCTYPE declaration.

“XML entities” on page 4275

“XML EntityDecl” on page 4276
The EntityDecl element represents a general entity and is declared in the internal subset of the DTD.

“XML ParameterEntityDecl” on page 4279
The ParameterEntityDecl represents a parameter entity definition in the internal subset of the DTD.

“XML DTD example” on page 4288

XML ExternalParameterEntityDecl:

The ExternalParameterEntityDecl element represents a parameter entity definition where the entity definition is contained externally to the current message.

The ExternalParameterEntityDecl is a named element and has a child of type SystemId. It can also have a child of type PublicId. The name of the entity does not include the percent sign %. In XML an external parameter entity declaration takes the form:

```
<!ENTITY % bookDef SYSTEM "BOOKDEF.DTD">
```

This represents an ExternalParameterEntityDecl element of name bookDef with a single child of type SystemId with a string value of `BOOKDEF.DTD`.

Related concepts:

“Message flows overview” on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

Chapter 9, “Developing message flow applications,” on page 1019
Develop message flows to process your business messages and data.

Related reference:

“XML document type declaration” on page 4271
The document type declaration (DTD) of an XML message is represented by a syntax element of type DocTypeDecl and its children and descendants. These comprise the DOCTYPE construct.

“XML DocTypeDecl” on page 4272
The DocTypeDecl represents the DOCTYPE declaration.

“XML PublicId” on page 4273
PublicId is an element that represents a public identifier in an XML message.

“XML SystemId” on page 4274
SystemId is a value element that represents a system identifier in an XML message.

"XML NotationDecl" on page 4275
The NotationDecl element represents a notation declaration in an XML message.
"XML entities" on page 4275

"XML DTD example" on page 4288

XML ExternalEntityDecl:

The ExternalEntityDecl element represents a general entity where the entity definition is contained externally to the current message.

The ExternalEntityDecl is a named element and has a child of type SystemId. It can also have a child of type PublicId.

An external entity declaration of the form:

```
<!ENTITY  bookAppendix SYSTEM "appendix.txt">
```

has an EntityDecl element of name bookAppendix and a child element of type SystemId with a string value of `appendix.txt`.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"XML document type declaration" on page 4271
The document type declaration (DTD) of an XML message is represented by a syntax element of type DocTypeDecl and its children and descendants. These comprise the DOCTYPE construct.

"XML DocTypeDecl" on page 4272
The DocTypeDecl represents the DOCTYPE declaration.

"XML PublicId" on page 4273
PublicId is an element that represents a public identifier in an XML message.

"XML SystemId" on page 4274
SystemId is a value element that represents a system identifier in an XML message.

"XML entities" on page 4275

"XML DTD example" on page 4288

XML ParameterEntityDecl:

The ParameterEntityDecl represents a parameter entity definition in the internal subset of the DTD.

The ParameterEntityDecl is a named element and has a single child element that is of type EntityDeclValue. For parameter entities, the name of the entity does not include the percent sign %. In XML a parameter entity declaration takes the form:

```
<!ENTITY % inline "#PCDATA | emphasis | link">
```

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"XML document type declaration" on page 4271
The document type declaration (DTD) of an XML message is represented by a syntax element of type DocTypeDecl and its children and descendants. These comprise the DOCTYPE construct.

"XML DocTypeDecl" on page 4272
The DocTypeDecl represents the DOCTYPE declaration.

"XML EntityDeclValue" on page 4277
The EntityDeclValue element represents the value of an EntityDecl or ParameterEntityDecl defined internally in the DOCTYPE construct.

"XML DTD example" on page 4288

XML UnparsedEntityDecl:

An UnparsedEntityDecl is used to include data in an XML document that is not well-formed XML, and is not parsed by an XML processor.

An unparsed entity is an external entity whose external reference is not parsed by an XML processor. This means that you can include data in an XML document that is not well-formed XML, such as a graphic file. The UnparsedEntityDecl is named element and a child of type SystemId that identifies the URI for the entity (a URL or a local file location). UnparsedEntityDecl can optionally have a child of type PublicId.

UnparsedEntityDecl can also have a child of type NotationReference, a value element that represents a reference to a notation declaration elsewhere in the XML document. It defines the type of data of the unparsed entity.

An unparsed entity declaration takes the form:

```
<!ENTITY pic SYSTEM "scheme.gif" NDATA gif>
```

In this example, the SystemId has a string value of `scheme.gif`. The value of NotationReference is `gif`. It refers to a NOTATION defined within the XML document:

```
<!NOTATION gif SYSTEM "image/gif">
```

The next entity is included in the DTD example:

```
<!ENTITY unpsd PUBLIC "//this/is/a/URI/me.gif" "me.gif" NDATA TeX>
```

This shows the optional PublicId element, which has the string value of `//this/is/a/URI/me.gif`.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

Chapter 9, “Developing message flow applications,” on page 1019
Develop message flows to process your business messages and data.

Related reference:

“XML document type declaration” on page 4271
The document type declaration (DTD) of an XML message is represented by a syntax element of type DocTypeDecl and its children and descendants. These comprise the DOCTYPE construct.

“XML DocTypeDecl” on page 4272
The DocTypeDecl represents the DOCTYPE declaration.

“XML PublicId” on page 4273
PublicId is an element that represents a public identifier in an XML message.

“XML SystemId” on page 4274
SystemId is a value element that represents a system identifier in an XML message.

“XML NotationDecl” on page 4275
The NotationDecl element represents a notation declaration in an XML message.

“XML DTD example” on page 4288

XML ElementDef:

The ElementDef element represents the `<!ELEMENT` construct in a DTD.

The ElementDef element is a child of the DOCTYPE element. The name of the element that is defined corresponds to the name of the syntax element. The value corresponds to the element definition.

This example is included in the DTD example:

```
<!ELEMENT subel2 (#PCDATA)>
```

The name of the element is `subel2` and the value is `(#PCDATA)`.

Related concepts:

“Message flows overview” on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

Chapter 9, “Developing message flow applications,” on page 1019
Develop message flows to process your business messages and data.

Related reference:

“XML document type declaration” on page 4271
The document type declaration (DTD) of an XML message is represented by a syntax element of type DocTypeDecl and its children and descendants. These comprise the DOCTYPE construct.

“XML DocTypeDecl” on page 4272
The DocTypeDecl represents the DOCTYPE declaration.

“XML DTD example” on page 4288

XML AttributeList:

The AttributeList name element represents the `<!ATTLIST` construct in a DTD.

The name of the AttributeList element corresponds to the name of the element for which the list of attributes is being defined. Its content represents one or more AttributeDef elements.

This example is included in the DTD example:

```
<!ATTLIST el5  el5satt CDATA #IMPLIED>
```

This example shows an AttributeList that defines one AttributeDef, and its content is explained in AttributeDef.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"XML document type declaration" on page 4271
The document type declaration (DTD) of an XML message is represented by a syntax element of type DocTypeDecl and its children and descendants. These comprise the DOCTYPE construct.

"XML DocTypeDecl" on page 4272
The DocTypeDecl represents the DOCTYPE declaration.

"XML AttributeDef"
The AttributeDef name element describes the definition of an attribute within an `<!ATTLIST` construct.

"XML DTD example" on page 4288

XML AttributeDef:

The AttributeDef name element describes the definition of an attribute within an `<!ATTLIST` construct.

The AttributeDef element is always a child of the AttributeList element. The name of the syntax element is the name of the attribute being defined. It can have three children:

- AttributeDefValue
- AttributeDefType
- AttributeDefDefaultType

This example is included in the DTD example:

```
<!ATTLIST el5  el5satt CDATA #IMPLIED>
```

The name of the AttributeDef is `el5satt` and it is a child of AttributeList `el5`. The name of the AttributeDefType is `CDATA`, and the value of the AttributeDefDefaultType is `IMPLIED`.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

Chapter 9, “Developing message flow applications,” on page 1019
Develop message flows to process your business messages and data.

Related reference:

“XML document type declaration” on page 4271
The document type declaration (DTD) of an XML message is represented by a syntax element of type DocTypeDecl and its children and descendants. These comprise the DOCTYPE construct.

“XML DocTypeDecl” on page 4272
The DocTypeDecl represents the DOCTYPE declaration.

“XML NotationDecl” on page 4275
The NotationDecl element represents a notation declaration in an XML message.

“XML entities” on page 4275

“XML ElementDef” on page 4281
The ElementDef element represents the `<!ELEMENT` construct in a DTD.

“XML AttributeList” on page 4281
The AttributeList name element represents the `<!ATTLIST` construct in a DTD.

“XML AttributeDefValue”
the AttributeDefValue gives the default value of attributes of type CDATA or attributes defined by an enumerated list.

“XML AttributeDefDefaultType” on page 4285
The AttributeDefDefaultType syntax element is a value element that represents the attribute default as defined under the attribute definition.

“XML AttributeDefType” on page 4284
The AttributeDefType syntax element is a name-value element whose name corresponds to the attribute type found in the attribute definition.

“XML DocTypePI” on page 4286
The DocTypePI element represents a processing instruction found within the DTD.

“XML WhiteSpace and DocTypeWhiteSpace” on page 4287
The WhiteSpace and DocTypeWhiteSpace XML elements represent white space characters in the message.

“XML DocTypeComment” on page 4286
Comments in the XML DTD are represented by the DocTypeComment element.

“XML DTD example” on page 4288

XML AttributeDefValue:

the AttributeDefValue gives the default value of attributes of type CDATA or attributes defined by an enumerated list.

For attributes of type CDATA, see “XML AttributeDefType” on page 4284.

For an example of AtrtibuteDefValue, see DTD example.

Related concepts:

“Message flows overview” on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

Chapter 9, “Developing message flow applications,” on page 1019
Develop message flows to process your business messages and data.

Related reference:

"XML document type declaration" on page 4271
The document type declaration (DTD) of an XML message is represented by a syntax element of type DocTypeDecl and its children and descendants. These comprise the DOCTYPE construct.

"XML DocTypeDecl" on page 4272
The DocTypeDecl represents the DOCTYPE declaration.

"XML AttributeList" on page 4281
The AttributeList name element represents the `<!ATTLIST` construct in a DTD.

"XML AttributeDef" on page 4282
The AttributeDef name element describes the definition of an attribute within an `<!ATTLIST` construct.

"XML AttributeDefDefaultType" on page 4285
The AttributeDefDefaultType syntax element is a value element that represents the attribute default as defined under the attribute definition.

"XML AttributeDefType"
The AttributeDefType syntax element is a name-value element whose name corresponds to the attribute type found in the attribute definition.

"XML DTD example" on page 4288

XML AttributeDefType:

The AttributeDefType syntax element is a name-value element whose name corresponds to the attribute type found in the attribute definition.

Possible values for the name are:
- CDATA
- ID
- IDREF
- IDREFS
- ENTITY
- ENTITIES
- NMTOKEN
- NMTOKENS
- NOTATION

If there is an enumeration present for the attribute definition, the entire enumeration string is held as a string in the value member of the name-value syntax element. In this case, the name member of the name-value syntax element is empty. The value string starts with an open parenthesis (and ends with a close parenthesis). Each entry in the enumeration string is separated by a vertical bar | character. If the Attribute value is not defined by an enumerated list, the value member of the syntax element is empty.

An example is included in AttributeDef.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"XML document type declaration" on page 4271
The document type declaration (DTD) of an XML message is represented by a syntax element of type DocTypeDecl and its children and descendants. These comprise the DOCTYPE construct.

"XML DocTypeDecl" on page 4272
The DocTypeDecl represents the DOCTYPE declaration.

"XML NotationDecl" on page 4275
The NotationDecl element represents a notation declaration in an XML message.

"XML entities" on page 4275

"XML ElementDef" on page 4281
The ElementDef element represents the `<!ELEMENT` construct in a DTD.

"XML AttributeList" on page 4281
The AttributeList name element represents the `<!ATTLIST` construct in a DTD.

"XML AttributeDef" on page 4282
The AttributeDef name element describes the definition of an attribute within an `<!ATTLIST` construct.

"XML AttributeDefValue" on page 4283
the AttributeDefValue gives the default value of attributes of type CDATA or attributes defined by an enumerated list.

"XML AttributeDefDefaultType"
The AttributeDefDefaultType syntax element is a value element that represents the attribute default as defined under the attribute definition.

"XML DocTypePI" on page 4286
The DocTypePI element represents a processing instruction found within the DTD.

"XML WhiteSpace and DocTypeWhiteSpace" on page 4287
The WhiteSpace and DocTypeWhiteSpace XML elements represent white space characters in the message.

"XML DocTypeComment" on page 4286
Comments in the XML DTD are represented by the DocTypeComment element.

"XML DTD example" on page 4288

XML AttributeDefDefaultType:

The AttributeDefDefaultType syntax element is a value element that represents the attribute default as defined under the attribute definition.

The value can be one of the following strings:
- #REQUIRED
- #IMPLIED
- #FIXED

An example is included in AttributeDef.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"XML document type declaration" on page 4271
The document type declaration (DTD) of an XML message is represented by a syntax element of type DocTypeDecl and its children and descendants. These comprise the DOCTYPE construct.

"XML DocTypeDecl" on page 4272
The DocTypeDecl represents the DOCTYPE declaration.

"XML AttributeList" on page 4281
The AttributeList name element represents the `<!ATTLIST` construct in a DTD.

"XML AttributeDef" on page 4282
The AttributeDef name element describes the definition of an attribute within an `<!ATTLIST` construct.

"XML AttributeDefValue" on page 4283
the AttributeDefValue gives the default value of attributes of type CDATA or attributes defined by an enumerated list.

"XML AttributeDefType" on page 4284
The AttributeDefType syntax element is a name-value element whose name corresponds to the attribute type found in the attribute definition.

"XML DTD example" on page 4288

XML DocTypeComment:

Comments in the XML DTD are represented by the DocTypeComment element.

It is a value element for which the value string contains the comment text. This element follows the same processing rules as the Comment element. See "XML comment" on page 4266.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"XML comment" on page 4266
An XML comment encountered outside the document type declaration is represented by the Comment value syntax element. It contains the comment text from the XML message.

"XML document type declaration" on page 4271
The document type declaration (DTD) of an XML message is represented by a syntax element of type DocTypeDecl and its children and descendants. These comprise the DOCTYPE construct.

"XML DocTypeDecl" on page 4272
The DocTypeDecl represents the DOCTYPE declaration.

"XML DTD example" on page 4288

XML DocTypePI:

The DocTypePI element represents a processing instruction found within the DTD.

This element is a name-value element. The name of the element is used to store the processing instruction target name, and the value contains the character data of the processing instruction. The value of the element can be empty. The name cannot be the string XML in either uppercase or lowercase form. This element follows the same processing rules as the ProcessingInstruction element. See "XML ProcessingInstruction" on page 4270.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"XML document type declaration" on page 4271
The document type declaration (DTD) of an XML message is represented by a syntax element of type DocTypeDecl and its children and descendants. These comprise the DOCTYPE construct.

"XML DocTypeDecl" on page 4272
The DocTypeDecl represents the DOCTYPE declaration.

"XML NotationDecl" on page 4275
The NotationDecl element represents a notation declaration in an XML message.

"XML entities" on page 4275

"XML ElementDef" on page 4281
The ElementDef element represents the `<!ELEMENT` construct in a DTD.

"XML AttributeList" on page 4281
The AttributeList name element represents the `<!ATTLIST` construct in a DTD.

"XML AttributeDef" on page 4282
The AttributeDef name element describes the definition of an attribute within an `<!ATTLIST` construct.

"XML WhiteSpace and DocTypeWhiteSpace"
The WhiteSpace and DocTypeWhiteSpace XML elements represent white space characters in the message.

"XML DocTypeComment" on page 4286
Comments in the XML DTD are represented by the DocTypeComment element.

"XML DTD example" on page 4288

XML WhiteSpace and DocTypeWhiteSpace:

The WhiteSpace and DocTypeWhiteSpace XML elements represent white space characters in the message.

The WhiteSpace element represents any white space characters outside the message body and DTD that are not represented by any other element. For example, white space within the body of the message (within elements) is reported as element content using the Content element type, but white space characters between the XML declaration and the beginning of the message body are represented by the WhiteSpace element.

```
<?xml version="1.0"?>       <BODY>....</BODY>
```

The characters between "1.0"?> and <BODY> are represented by the WhiteSpace element.

White space is used in XML for readability and has no business meaning. Input XML messages can include line breaks, blanks lines, and spaces between tags (all shown in the following example). If you process XML messages that contain any of these spaces, they are represented as elements in the message tree. Therefore they appear when you view the message in the debugger, and in any trace output.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE s1 PUBLIC "http://www.ibm.com/example.dtd" "example.dtd">
<s1>.........
<s2>abc</s2>   <s2>def</s2>

<s3>123</s3>
</s1>
```

If you do not want white space elements in your message trees, you must present the input message as a single line, or use the XMLNSC compact parser in its default mode

The DocTypeWhiteSpace element represents white space that is found inside the DTD that is not represented by any other element. White space characters found within a DocType between two definitions are represented by the DocTypeWhiteSpace element.

```
    <!ENTITY % bookDef SYSTEM "BOOKDEF.DTD">  <!ENTITY bookTitle "User Guide">
```

The characters between DTD"> and <!ENTITY are represented by the DocTypeWhiteSpace element.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"XML document type declaration" on page 4271
The document type declaration (DTD) of an XML message is represented by a syntax element of type DocTypeDecl and its children and descendants. These comprise the DOCTYPE construct.

"XML DocTypeDecl" on page 4272
The DocTypeDecl represents the DOCTYPE declaration.

"XML DocTypePI" on page 4286
The DocTypePI element represents a processing instruction found within the DTD.

"XML DocTypeComment" on page 4286
Comments in the XML DTD are represented by the DocTypeComment element.

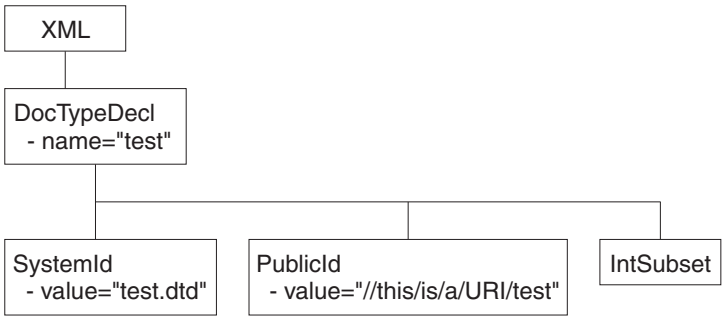
"XML DTD example"

XML DTD example:

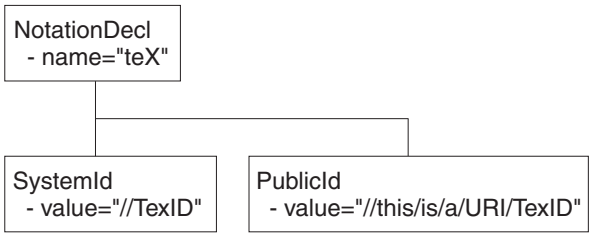
This example shows an XML DTD in an XML document and the tree structure form of that document:

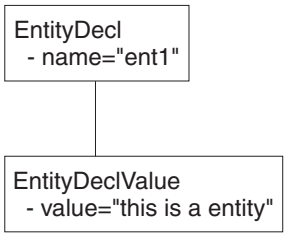
```
<!DOCTYPE test PUBLIC "//this/is/a/URI/test" "test.dtd" [
<!NOTATION TeX  PUBLIC "//this/is/a/URI/TexID" "//TexID">
<!ENTITY ent1 "this is an entity">
<!ENTITY % ent2 "#PCDATA | subel2">
<!ENTITY % extent1 PUBLIC "//this/is/a/URI/extent1" "more.txt">
<!ENTITY extent2 PUBLIC "//this/is/a/URI/extent2" "more.txt">
<!ENTITY unpsd PUBLIC "//this/is/a/URI/me.gif" "me.gif" NDATA TeX>
<?test Do this?>
<!--this is a comment-->
<!ELEMENT subel2 (#PCDATA)>
<!ELEMENT subel1 (subel2 | el4)+>
<!ELEMENT el1 (#PCDATA)>
<!ELEMENT el2 (#PCDATA | subel2)*>
<!ELEMENT el3 (#PCDATA | subel2)*>
<!ELEMENT el4 (#PCDATA)>
<!ELEMENT el5 (#PCDATA | subel1)*>
<!ELEMENT el6 (#PCDATA)>
<!ATTLIST subel1
 size (big | small) "big"
 shape (round | square) #REQUIRED>
<!ATTLIST el5
 el5satt CDATA #IMPLIED>
]>
```

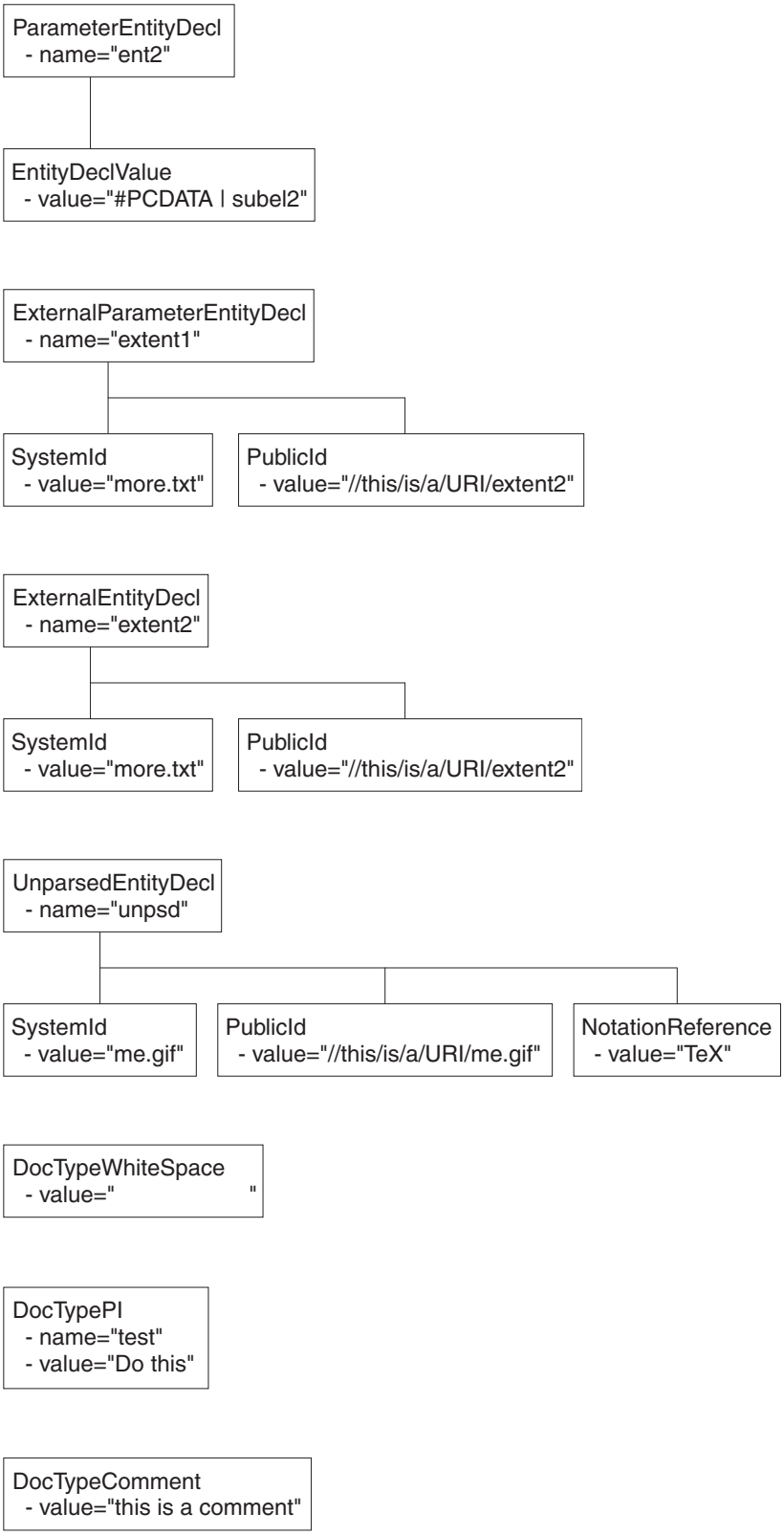
When a message is parsed by the generic XML parser, the relevant part of the message tree looks like this (assuming that there are no carriage returns or white space between tags):

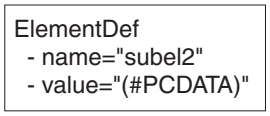


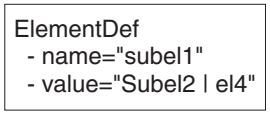
The IntSubset structure contains the following structures at the next level of nesting: the tree structure for each of these is shown in the following tree structures.

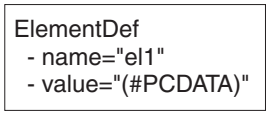


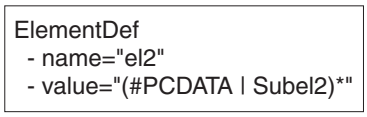


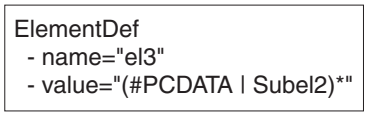

ParameterEntityDecl
- name="ent2"
EntityDeclValue
- value="#PCDATA | subel2"
ExternalParameterEntityDecl
- name="extent1"
SystemId
- value="more.txt"
PublicId
- value="//this/is/a/URI/extent2"
ExternalEntityDecl
- name="extent2"
SystemId
- value="more.txt"
PublicId
- value="//this/is/a/URI/extent2"
UnparsedEntityDecl
- name="unpsd"
SystemId
- value="me.gif"
PublicId
- value="//this/is/a/URI/me.gif"
NotationReference
- value="TeX"
DocTypeWhiteSpace
- value="
"
DocTypePI
- name="test"
- value="Do this"
DocTypeComment
- value="this is a comment"

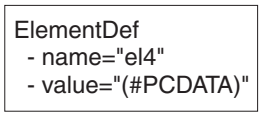

ElementDef
- name="subel2"
- value="(#PCDATA)"

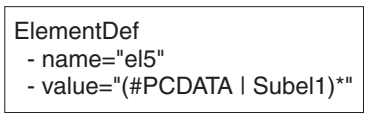

ElementDef
- name="subel1"
- value="Subel2 | el4"

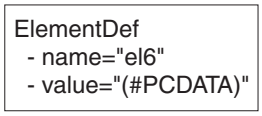

ElementDef
- name="el1"
- value="(#PCDATA)"

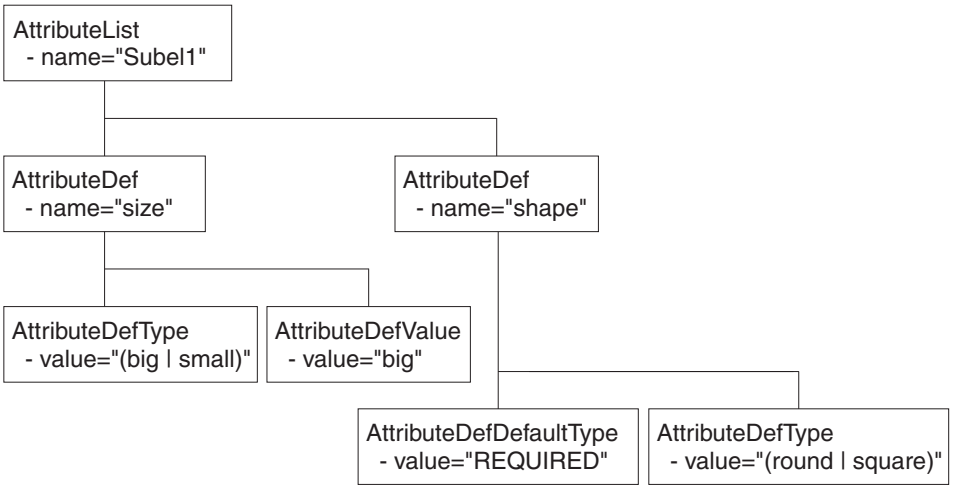

ElementDef
- name="el2"
- value="(#PCDATA | Subel2)*"

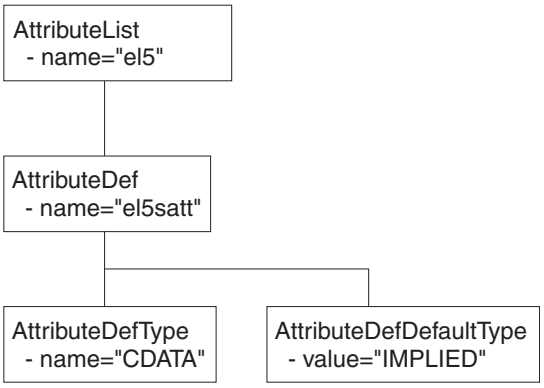

ElementDef
- name="el3"
- value="(#PCDATA | Subel2)*"


ElementDef
- name="el4"
- value="(#PCDATA)"


ElementDef
- name="el5"
- value="(#PCDATA | Subel1)*"


ElementDef
- name="el6"
- value="(#PCDATA)"





Related reference:

"XML document type declaration" on page 4271
The document type declaration (DTD) of an XML message is represented by a syntax element of type DocTypeDecl and its children and descendants. These comprise the DOCTYPE construct.

"XML DocTypeDecl" on page 4272
The DocTypeDecl represents the DOCTYPE declaration.

"XML NotationDecl" on page 4275
The NotationDecl element represents a notation declaration in an XML message.

"XML entities" on page 4275

"XML ElementDef" on page 4281
The ElementDef element represents the `<!ELEMENT` construct in a DTD.

"XML AttributeList" on page 4281
The AttributeList name element represents the `<!ATTLIST` construct in a DTD.

"XML AttributeDef" on page 4282
The AttributeDef name element describes the definition of an attribute within an `<!ATTLIST` construct.

"XML DocTypePI" on page 4286
The DocTypePI element represents a processing instruction found within the DTD.

"XML WhiteSpace and DocTypeWhiteSpace" on page 4287
The WhiteSpace and DocTypeWhiteSpace XML elements represent white space characters in the message.

"XML DocTypeComment" on page 4286
Comments in the XML DTD are represented by the DocTypeComment element.

Built-in nodes

WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

The mode that your broker is working in can affect the types of node that you can use; see "Restrictions that apply in each operation mode" on page 3657.

For information about each of these built-in nodes, use the following links. The nodes are listed in the categories under which they are grouped in the node palette, see "Message flow node palette" on page 1027.

WebSphere MQ
- "MQInput node" on page 4594
- "MQOutput node" on page 4612
- "MQReply node" on page 4621
- "MQGet node" on page 4578
- "MQHeader node" on page 4590
- "MQOptimizedFlow node" on page 4612

JMS
- "JMSInput node" on page 4532
- "JMSOutput node" on page 4549
- "JMSReply node" on page 4562
- "JMSHeader node" on page 4529
- "JMSMQTransform node" on page 4547
- "MQJMSTransform node" on page 4610

HTTP
- "HTTPInput node" on page 4474
- "HTTPReply node" on page 4484
- "HTTPRequest node" on page 4488
- "HTTPHeader node" on page 4470

Web Services
- "SOAPInput node" on page 4795
- "SOAPReply node" on page 4819
- "SOAPRequest node" on page 4828
- "SOAPAsyncRequest node" on page 4750
- "SOAPAsyncResponse node" on page 4777
- "SOAPEnvelope node" on page 4786
- "SOAPExtract node" on page 4790
- "RegistryLookup node" on page 4646
- "EndpointLookup node" on page 4407

SCA
- "SCAInput node" on page 4707
- "SCAReply node" on page 4726
- "SCAAsyncRequest node" on page 4690
- "SCAAsyncResponse node" on page 4698
- "SCARequest node" on page 4719

WebSphere Adapters

- "JDEdwardsInput node" on page 4519
- "JDEdwardsRequest node" on page 4524
- "PeopleSoftInput node" on page 4630
- "PeopleSoftRequest node" on page 4635
- "SAPInput node" on page 4676
- "SAPReply node" on page 4682
- "SAPRequest node" on page 4685
- "SiebelInput node" on page 4740
- "SiebelRequest node" on page 4745
- "TwineballInput node" on page 4951
- "TwineballRequest node" on page 4955

Routing
- "Filter node" on page 4452
- "Label node" on page 4569
- "Publication node" on page 4643
- "RouteToLabel node" on page 4673
- "Route node" on page 4669
- "AggregateControl node" on page 4296
- "AggregateReply node" on page 4299
- "AggregateRequest node" on page 4303
- "Collector node" on page 4333
- "Resequence node" on page 4651
- "Sequence node" on page 4736

Transformation
- "Mapping node" on page 4571
- "XSLTransform node" on page 4968
- "Compute node" on page 4340
- "JavaCompute node" on page 4514
- "PHPCompute node" on page 4639

Construction
- "Input node" on page 4511
- "Output node" on page 4626
- "Throw node" on page 4929
- "Trace node" on page 4942
- "TryCatch node" on page 4949
- "FlowOrder node" on page 4458
- "Passthrough node" on page 4628
- "ResetContentDescriptor node" on page 4663

Database
- "Database node" on page 4354
- "DatabaseInput node" on page 4360
- "DataDelete node" on page 4382
- "DataInsert node" on page 4386
- "DataUpdate node" on page 4390
- "Warehouse node" on page 4963
- "DatabaseRetrieve node" on page 4363
- "DatabaseRoute node" on page 4373
- "Extract node" on page 4412

File
- "FileInput node" on page 4415
- "FileOutput node" on page 4430

- “FTEInput node” on page 4461
- “FTEOutput node” on page 4466
- “FileRead node” on page 4444
- “CDInput node” on page 4305
- “CDOutput node” on page 4312

Email
- “EmailInput node” on page 4394
- “EmailOutput node” on page 4400

TCPIP
- “TCPIPClientInput node” on page 4854
- “TCPIPClientOutput node” on page 4867
- “TCPIPClientReceive node” on page 4877
- “TCPIPServerInput node” on page 4890
- “TCPIPServerOutput node” on page 4903
- “TCPIPServerReceive node” on page 4913

CORBA
- “CORBARequest node” on page 4349

CICS
- “CICSRequest node” on page 4321

IMS
- “IMSRequest node” on page 4504

Validation
- “Validate node” on page 4959
- “Check node” on page 4318

Security
- “SecurityPEP node” on page 4729

Timer
- “TimeoutControl node” on page 4932
- “TimeoutNotification node” on page 4936

Related concepts:

“Message flows overview” on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

“Designing a message flow” on page 1455
A message flow can perform a wide range of operations, depending on your business and operational requirements. For best performance and capability, you must design it to include the most appropriate nodes.

“Creating a message flow” on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

“Defining message flow content” on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

AggregateControl node

Use the AggregateControl node to mark the beginning of a fan-out of requests that are part of an aggregation.

This topic contains the following sections:
- "Purpose"
- "Using this node in a message flow"
- "Terminals and properties"

Purpose:
Aggregation is an extension of the request/reply application model. It combines the generation and fan-out of a number of related requests with the fan-in of the corresponding replies, and compiles those replies into a single aggregated reply message.

The aggregation function is provided by the following three nodes:

- The AggregateControl node marks the beginning of a fan-out of requests that are part of an aggregation. It sends a control message that is used by the AggregateReply node to match the different requests that have been made. The information that is propagated from the Control terminal includes the broker identifier, the aggregate name property, and the timeout property. You must not change the aggregation information that is added to the message Environment by the AggregateControl node.
- The AggregateRequest node records the fact that the request messages have been sent. It also collects information that helps the AggregateReply node to construct the aggregated reply message. You must preserve the information that is added to the message Environment by the AggregateControl node, otherwise the aggregation fails.
- The AggregateReply node marks the end of an aggregation fan-in. It collects replies and combines them into a single aggregated reply message.

This node creates the `LocalEnvironment.ComIbmAggregateControlNode` folder. This folder and the fields in it are for internal use by WebSphere Message Broker and you should not rely on their existence or values when developing your message flows.

The AggregateControl node is contained in the **Routing** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:
Look at the following samples to see how to use this node:

- Aggregation
- Airline Reservations

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Terminals and properties:

When you have put an instance of the AggregateControl node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The AggregateControl node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Out	The output terminal to which the original message is routed when processing completes successfully.
Control	The output terminal to which a control message is routed. The control message is sent to a corresponding AggregateReply node. The Control terminal is deprecated in Version 6.0; to use connections from the Control terminal, see "Using control messages in aggregation flows" on page 2745.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The AggregateControl node Description properties are described in the following table:

Property	M	C	Default	Description
Node name	No	No	The node type (AggregateControl)	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The AggregateControl node Basic properties are described in the following table:

Property	M	C	Default	Description	`mqsiapplybaroverride` command property
Aggregate Name	Yes	Yes		A name that is used to associate the fan-out message flow with the fan-in message flow. This value must be contextually unique in a broker. This name is also used to identify an Aggregation configurable service (if one exists) to be used by the node.	aggregateName

Property	M	C	Default	Description	mqsiapplybaroverride command property
Timeout (sec)	Yes	No	0	The amount of time, in seconds, that it waits for replies to arrive at the fan-in. The default value is zero; if you accept this default value, the timeout is disabled for fan-outs from this node (that is, it waits for replies indefinitely). If not all responses are received, the message flow continues to wait, and does not complete. Set a value greater than zero to ensure that the message flow can complete, even if not all responses are received. For further information about timeouts, see "AggregateReply node" on page 4299. z/OS On z/OS, if the Timeout property is not set to zero, set the queue manager parameter **EXPRYINT** to 5. The value specified by the `Timeout (sec)` property is overridden by the value set in the **timeoutSeconds** property of the Aggregation configurable service, if it is set. Timeout values specified by the node and the configurable service are overridden by a timeout value defined in the message, at the location specified by the `Timeout location` property of the AggregateControl node.	

The AggregateControl node Advanced properties are described in the following table:

Property	M	C	Default	Description
Timeout location	No	No	`'$LocalEnvironment/Aggregation/Timeout'`	The location in the message tree where the aggregation timeout value is defined. The value specified in the message tree overrides the `Timeout (sec)` property of the AggregateControl node and the **timeoutSeconds** property of the Aggregation configurable service.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message flow aggregation" on page 2718
Aggregation is the generation and fan-out of related requests that are derived from a single input message, and the fan-in of the corresponding replies to produce a

single aggregated reply message.

Related tasks:

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Configuring aggregation flows" on page 2721
Use aggregation message flows to generate and fan-out a number of related requests, fan-in the corresponding replies, and compile those replies into a single aggregated reply message, by using the AggregateControl, AggregateRequest, and AggregateReply nodes.

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

Related reference:

"AggregateReply node"
Use the AggregateReply node to mark the end of an aggregation fan-in. This node collects replies and combines them into a single compound message.

"AggregateRequest node" on page 4303
Use the AggregateRequest node to record the fact that request messages have been sent. This node also collects information that helps the AggregateReply node to construct the compound response message.

AggregateReply node

Use the AggregateReply node to mark the end of an aggregation fan-in. This node collects replies and combines them into a single compound message.

This topic contains the following sections:

Purpose:

Aggregation is an extension of the request/reply application model. It combines the generation and fan-out of a number of related requests with the fan-in of the corresponding replies, and compiles those replies into a single aggregated reply message.

The aggregation function is provided by the following three nodes:

- The AggregateControl node marks the beginning of a fan-out of requests that are part of an aggregation. It sends a control message that is used by the AggregateReply node to match the different requests that have been made. The information that is propagated from the Control terminal includes the broker identifier, the aggregate name property, and the timeout property. The aggregation information that is added to the message Environment by the AggregateControl node must not be changed.
- The AggregateRequest node records the fact that the request messages have been sent. It also collects information that helps the AggregateReply node to construct

the aggregated reply message. The information that is added to the message Environment by the AggregateRequest must be preserved, otherwise the aggregation fails.

- The AggregateReply node marks the end of an aggregation fan-in. It collects replies and combines them into a single aggregated reply message.

The AggregateReply node is contained in the **Routing** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

When incoming messages are stored by the AggregateReply node before all responses for the aggregation are received, the persistence of the message determines whether the message is retained across a restart.

If, during an aggregation, one or more of the response messages are not received by the AggregateReply node, the normal timeout or unknown message processing deals with the responses that have been received already.

The MQMD.Expiry value of each AggregateReply message is set to -1 in the compound output message. This value is set because the MQMD.Expiry value has no meaning once the reply message is no longer on the WebSphere MQ Transport and has been stored by the broker during the aggregation process.

Using this node in a message flow:
Look at the following samples to see how to use this node:

- Aggregation
- Airline Reservations

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Terminals and properties:
When you have put an instance of the AggregateReply node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The AggregateReply node terminals are described in the following table.

Terminal	Description
Control	The input terminal that accepts control messages that are sent by a corresponding AggregateControl node. The Control terminal is deprecated in Version 6.0; to use connections to the Control terminal, see “Using control messages in aggregation flows” on page 2745.
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the message is routed if a failure is detected during processing.
Unknown	The output terminal to which messages are routed when they cannot be identified as valid reply messages.

Terminal	Description
Out	The output terminal to which the compound message is routed when processing completes successfully.
Timeout	The output terminal to which the incomplete compound message is routed when the timeout interval that is specified in the corresponding AggregateControl node has expired.
Catch	The output terminal to which the message is routed if an exception is thrown downstream and then caught by this node.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Description properties of the AggregateReply node are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type (AggregateReply)	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The AggregateReply node Basic properties are described in the following table.

Property	M	C	Default	Description	**mqsiapplybaroverride command property**
Aggregate Name	Yes	Yes		A name that is used to associate the fan-in message flow with the fan-out message flow. This value must be contextually unique within a broker. This name is also used to identify an Aggregation configurable service (if one exists) to be used by the node.	aggregateName
Unknown Message Timeout	No	No	`0`	The amount of time, in seconds, for which messages that cannot be identified as replies are held before they are propagated to the Unknown terminal. The default value is zero; if you accept this default value, the timeout is disabled, and unknown messages are propagated to the Unknown terminal upon receipt. z/OS On z/OS, if the Unknown Message Timeout property is not set to zero, set the queue manager parameter **EXPRYINT** to 5.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Transaction Mode	Yes	No	Selected	This property defines the transactional characteristics of this message: • If you select the check box (the default), the subsequent message flow is under transaction control. This setting remains true for messages that derive from the output message and are produced by an MQOutput node, unless the MQOutput node explicitly overrides the transaction status. No other node can change the transactional characteristics of the output message. • If you clear the check box, the subsequent message flow is not under transaction control. This setting remains true for messages that derive from the output message and are produced by an MQOutput node, unless the MQOutput node has specified that the message must be put under sync point.	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message flow aggregation" on page 2718
Aggregation is the generation and fan-out of related requests that are derived from a single input message, and the fan-in of the corresponding replies to produce a single aggregated reply message.

Related tasks:

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

"Configuring transactionality for message flows" on page 1290
A message flow runs in a single transaction, which is started when data is received by an input node, and can be committed or rolled back when all processing has completed.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Configuring aggregation flows" on page 2721
Use aggregation message flows to generate and fan-out a number of related requests, fan-in the corresponding replies, and compile those replies into a single aggregated reply message, by using the AggregateControl, AggregateRequest, and AggregateReply nodes.

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

"Using control messages in aggregation flows" on page 2745
The default behavior is that connections between AggregateControl and AggregateReply nodes for sending control messages are ignored. This configuration optimizes performance and removes the possibility that response messages will be received by the AggregateReply node before the control message.

Related reference:

"AggregateControl node" on page 4296
Use the AggregateControl node to mark the beginning of a fan-out of requests that are part of an aggregation.

"AggregateRequest node"
Use the AggregateRequest node to record the fact that request messages have been sent. This node also collects information that helps the AggregateReply node to construct the compound response message.

AggregateRequest node

Use the AggregateRequest node to record the fact that request messages have been sent. This node also collects information that helps the AggregateReply node to construct the compound response message.

This topic contains the following sections:
- "Purpose"
- "Using this node in a message flow" on page 4304
- "Terminals and properties" on page 4304

Purpose:
Aggregation is an extension of the request/reply application model. It combines the generation and fan-out of a number of related requests with the fan-in of the corresponding replies, and compiles those replies into a single aggregated reply message.

The aggregation function is provided by the following three nodes:

- The AggregateControl node marks the beginning of a fan-out of requests that are part of an aggregation. It sends a control message that is used by the AggregateReply node to match the different requests that have been made. The information that is propagated from the Control terminal includes the broker identifier, the aggregate name property, and the timeout property. The aggregation information that is added to the message Environment by the AggregateControl node must not be changed.
- The AggregateRequest node records the fact that the request messages have been sent. It also collects information that helps the AggregateReply node to construct the aggregated reply message. The information that is added to the message Environment by the AggregateRequest node must be preserved, otherwise the aggregation fails.
- The AggregateReply node marks the end of an aggregation fan-in. It collects replies and combines them into a single aggregated reply message.

The AggregateRequest node is contained in the **Routing** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:
Look at the following samples to see how to use this node:

- Aggregation
- Airline Reservations

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Terminals and properties:
When you have put an instance of the AggregateRequest node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The AggregateRequest node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts messages sent as part of an aggregate request.
Out	The output terminal to which the input message is routed when processing completes successfully.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The AggregateRequest node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type (AggregateRequest)	The name of the node
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The AggregateRequest node Basic property is described in the following table.

Property	M	C	Default	Description
Folder Name	Yes	No		The name that is used as a folder in the AggregateReply node's compound message to store the reply to this request. You must enter a value for this property, but the value does not need to be unique.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message flow aggregation" on page 2718
Aggregation is the generation and fan-out of related requests that are derived from a single input message, and the fan-in of the corresponding replies to produce a single aggregated reply message.

Related tasks:

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Configuring aggregation flows" on page 2721
Use aggregation message flows to generate and fan-out a number of related requests, fan-in the corresponding replies, and compile those replies into a single aggregated reply message, by using the AggregateControl, AggregateRequest, and AggregateReply nodes.

Related reference:

"AggregateControl node" on page 4296
Use the AggregateControl node to mark the beginning of a fan-out of requests that are part of an aggregation.

"AggregateReply node" on page 4299
Use the AggregateReply node to mark the end of an aggregation fan-in. This node collects replies and combines them into a single compound message.

CDInput node

Use the CDInput node to preview files when using IBM Sterling Connect:Direct in conjunction with WebSphere Message Broker.

This topic contains the following sections:

Purpose:

You can use the CDInput node to extend WebSphere Message Broker support for file processing through its integration with IBM Sterling Connect:Direct.

On z/OS, when the CDInput node receives notification of the arrival of a dataset that it needs to process, the node copies that dataset into Unix System Services temporarily, before processing.

The CDInput node is contained in the **File** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:
You can use the CDInput node in any flow that is designed to accept files from an IBM Sterling Connect:Direct network.

Terminals and properties:
The CDInput node terminals are described in the following table.

Terminal	Description
Failure	The output terminal to which a message is routed if an error occurs before a message is propagated to the Out terminal. Messages propagated to this terminal are not validated, even if you have specified, using the `Validate` property, that validation is to take place.
Out	The output terminal to which a message is routed if it has been successfully extracted from the input file. If no errors occur within the input node, a message received from an external resource is always sent to the Out terminal first.
End of Data	The output terminal to which the End of Data message is routed after all the messages in a file have been processed. The End of Data message flow transaction is initiated only if this terminal is attached. The end of data structure consists of an empty message body, and the Local Environment information propagated from the out terminal.
Catch	The output terminal to which the message is routed if an exception is thrown downstream and caught by this node. Exceptions are caught only if this terminal is attached.

The following tables describe the node properties that you can set on a specified tab. The column headed M indicates whether the property is *mandatory* (marked in the toolkit with an asterisk if you must enter a value when no default is defined). The column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

When the CDInput node propagates a message, it stores information about it in the LocalEnvironment.CD and LocalEnvironment.CD.Transfer message trees. If the input file is empty, an empty message is propagated (assuming that it is valid). If you specify a file name pattern that contains wildcard characters in the `File name filter` property, the CDInput node copies the characters in the file name matched by wildcard characters, together with any intermediate characters, to the LocalEnvironment.Wildcard.WildcardMatch message tree. See “Using local environment variables with file nodes” on page 1820 for more information.

Description properties

Property	M	C	Default	Description
Node name	No	No	`CDInput`	The name of the node.
Short Description	No	No	None	A brief description of the node.

Property	M	C	Default	Description
Long Description	No	No	None	Text that describes the purpose of the node in the message flow.

Basic properties

Property	M	C	Default	Description
Directory filter	No	Yes	None	Specify the directory from which the node can process files. If this property is left blank, the node can process files from all directories. If multiple CDInput nodes are deployed to the same execution group, files are distributed randomly between the nodes unless filtering is defined. The directory must exist. If the WebSphere Message Broker and IBM Sterling Connect:Direct server are on different machines, this is the path to the directory on the broker machine. For information on various configurations when using IBM Sterling Connect:Direct, see "Advanced configuration properties when using IBM Sterling Connect:Direct nodes" on page 727 and refer to the Input sections. If the `brokerPathToInputDir` field in the configurable service is set, the directory filter can be a path relative to that value; otherwise it must be an absolute path. On z/OS, if the file is a sequential dataset, partitioned dataset, or partitioned dataset member, leave the Directory filter field blank.
File name filter	Yes	Yes	None	Specify either a file name, or a character sequence that matches a file name. Either the file name or the character sequence can contain at least one of the following wildcard characters: • * (any sequence of zero or more characters) • ? (any single character) By default the node processes all files. If multiple CDInput nodes are deployed to the same execution group, files are distributed randomly between the nodes unless filtering is defined. The CDInput node can process z/OS sequential datasets, entire partitioned datasets or partitioned dataset members. The syntax to address a dataset is based on the full name for the dataset, for example, `MBUSER.TEST1`. Wildcard characters can be used anywhere within a dataset file name filter, this works in the same way as for normal file name filter patterns. For a member within a partitioned dataset, use brackets to specify the member name, for example, `MBUSER.TEST(MEME01)`. When receiving an entire partitioned dataset, each member in the received dataset is processed as an individual message.

Property	M	C	Default	Description
Connect:Direct server configurable service	No	Yes	Default	The name of the configurable service being used to connect to the Connect:Direct server, in order to collect transfer information. If this value is not set, the default configurable service (named "Default") is used. The default configurable service connects to a Connect:Direct server located on the same machine as the broker, and using default port configurations. The default configurable service also uses the "default" security identity, which must be created using the mqsisetdbparms command; for example: `mqsisetdbparms MB7BROKER -n cd::default -u mqbroker -p xxxxxxx` For information on various configurations when using IBM Sterling Connect:Direct, see "Advanced configuration properties when using IBM Sterling Connect:Direct nodes" on page 727 and refer to the Input sections.
Action on successful processing	Yes	No	No Action	Select the action to take once the node has successfully processed the file. You can choose to leave the file in the input directory if other processes also need access to the file. In this case the notification from IBM Sterling Connect:Direct is deleted, and the file is left in place. Deleting files prevents the build up of processed files. Use the timestamp option if an archive of the file is required to record all transfers made. Note that on z/OS the Add Time Stamp option is not supported when using datasets.

Input Message Parsing properties

Property	M	C	Default	Description	**mqsiapplybaroverride command property**
Message Domain	No	No	None	The domain that is used to parse the incoming message.	
Message Set	No	No	None	The name or identifier of the message set in which the incoming message is defined. If you set this property, and then update the project dependencies to remove this message set reference, a warning is issued. Either update the `Message Set` property, or restore the reference to this message set project.	
Message Type	No	No	None	The name of the incoming message.	
Message Format	No	No	None	The name of the physical format of the incoming message.	
Message coded character set ID	Yes	Yes	`Broker System Default`	The ID of the coded character set used to interpret bytes of the file being read.	messageCodedCharSetIdProperty

Property	M	C	Default	Description	mqsiapplybaroverride command property
Message encoding	Yes	Yes	Broker System Determined	The encoding scheme for numbers used to interpret bytes of the file being read. Valid values are Broker System Determined or a numeric encoding value. For more information about encoding, see "Data conversion" on page 1151.	messageEncodingProperty

Parser Options properties

Property	M	C	Default	Description
Parse timing	No	No	On Demand	Specifies when an input message is parsed. Valid values are: • On Demand • Immediate • Complete For a full description of this property, see "Parsing on demand" on page 4173.
Build tree using XML schema data types	No	No	Cleared	Specifies whether the syntax elements in the message tree have data types taken from the XML schema.
Use XMLNSC compact parser for XMLNS domain	No	No	Cleared	Specifies whether the XMLNSC Compact Parser is used for messages in the XMLNS Domain. If you set this property, the message data appears under XMLNSC in nodes that are connected to the output terminal when either of the following items is XMLNS: • The input MQRFH2 header. • The Input Message Parsing property, Message Domain.
Retain mixed content	No	No	Cleared	Specifies whether the XMLNSC parser creates elements in the message tree for mixed text in an input message. If you select the check box, elements are created for mixed text. If you clear the check box, mixed text is ignored and no elements are created.
Retain comments	No	No	Cleared	Specifies whether the XMLNSC parser creates elements in the message tree for comments in an input message. If you select the check box, elements are created for comments. If you clear the check box, comments are ignored and no elements are created.
Retain processing instructions	No	No	Cleared	Specifies whether the XMLNSC parser creates elements in the message tree for processing instructions in an input message. If you select the check box, elements are created for processing instructions. If you clear the check box, processing instructions are ignored and no elements are created.
Opaque elements	No	No	Blank	Specifies a list of elements in the input message that are to be opaquely parsed by the XMLNSC parser. Opaque parsing is performed only if validation is not enabled (that is, if Validate is None); entries that are specified in Opaque Elements are ignored if validation is enabled.

Retry properties

Property	M	C	Default	Description	mqsiapplybaroverride command property
Retry mechanism	Yes	No	Failure	How the node handles a flow failure. Valid options are: • Failure • Short retry • Short and long retry	
Retry threshold	Yes	Yes	0	The number of times to try the flow transaction again when the Retry mechanism property value is Short retry.	retryThreshold
Short retry interval (seconds)	No	Yes	0	The interval, in seconds, between each retry if the Retry threshold property is not zero.	shortRetryInterval
Long retry interval (seconds)	No	Yes	300	The interval between retries, if the Retry mechanism property is Short and long retry and the retry threshold has been exhausted.	longRetryInterval
Action on failing file	Yes	Yes	No Action	The action that the node takes with the input file if all attempts to process the contents of the input file fail. Valid options are: • No Action • Delete • Add Time Stamp This option is not available on z/OS	

Records and Elements properties

Property	M	C	Default	Description
Record detection	Yes	No	Whole File	The mechanism used to identify records in the input file. Valid options are: • Whole File • Fixed Length • Delimited • Parsed Record Sequence
Length	Yes	No	80	The length of each record, in bytes, when Fixed Length record detection is selected.
Delimiter	Yes	Yes	DOS or UNIX Line End	The type of delimiter bytes that separate, or end, each record when Delimited record detection is selected. Valid options are: • DOS or UNIX Line End • Custom Delimiter
Custom delimiter	No	Yes	None	The delimiter bytes, expressed in hexadecimal, when Delimited record detection and Custom Delimiter are selected. This property is mandatory only if the Delimiter property is set to Custom Delimiter.
Delimiter type	Yes	No	Postfix	The position of the delimiter when Delimited record detection is selected. Valid options are: • Postfix • Infix This property is ignored unless the Delimiter property is set to Custom Delimiter.

Validation properties

For a full description of these properties, see "Validation properties" on page 4169.

Property	M	C	Default	Description	**mqsiapplybaroverride command property**
Validate	No	Yes	None	This property controls whether validation takes place. Valid values are: • None • Content and Value • Content	validateMaster
Failure action	No	No	Exception	This property controls what happens if validation fails. Valid values are: • User Trace • Local Error Log • Exception • Exception List	

Transactions properties

Property	M	C	Default	Description
Transaction mode	No	No	No	The transaction mode on this input node determines whether the rest of the nodes in the flow are processed under sync point. Valid options are: • Yes • No

Instances properties.

For a full description of these properties, see "Configurable message flow properties" on page 4020.

Property	M	C	Default	Description	**mqsiapplybaroverride command property**
Additional instances pool	No	Yes	Use Pool Associated with Message Flow	The pool from which additional instances are obtained. • If you select Use Pool Associated with Message Flow, additional instances are obtained from the message flow pool. • If you select Use Pool Associated with Node, additional instances are allocated from the additional instances of the node based on the number specified in the Additional instances property.	componentLevel
Additional instances	No	Yes	0	The number of additional instances that the node can start if the Additional instances pool property is set to Use Pool Associated with Node.	additionalInstances

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"IBM Sterling Connect:Direct overview and concepts" on page 1810
An overview of IBM Sterling Connect:Direct and its terminology when used with WebSphere Message Broker.

"Using local environment variables with file nodes" on page 1820
You can use fields in the local environment to dynamically alter the behavior of the FileInput, FileOutput, FTEInput, and FTEOutput nodes. You can also find what values the output nodes used to process the file.

"File name patterns" on page 1830
You can specify a file name pattern, using wildcard characters, to identify a file to be read by the FileInput and FTEInput nodes. You can also specify a file name pattern, using a single wildcard character, to name the file to be created by the FileOutput and FTEOutput nodes.

Related tasks:

"Initiating a managed file transfer using IBM Sterling Connect:Direct" on page 1873
Use a CDOutput node to send a file from a specified directory on your primary Connect:Direct server (PNODE) to a filename and directory on a secondary Connect:Direct server (SNODE).

"Advanced configuration properties when using IBM Sterling Connect:Direct nodes" on page 727
CDInput and CDOutput nodes can get connection details and staging directories in conjunction with a configurable service. To pick up new values when a configurable service is created or modified, you must reload the broker or execution group, by using the **mqsistop** and **mqsistart** commands, or the **mqsireload** command.

Related reference:

"CDOutput node"
Use the CDOutput node when using IBM Sterling Connect:Direct with WebSphere Message Broker.

CDOutput node

Use the CDOutput node when using IBM Sterling Connect:Direct with WebSphere Message Broker.

This topic contains the following sections:
- "Purpose"
- "Using this node in a message flow" on page 4313
- "Terminals and properties" on page 4313

Purpose:
Use the CDOutput node to serialize the message tree to a file and then transfer it between two Connect:Direct servers. A directory under the work path within the

execution group is used as the staging area, until the file is ready to be transferred. Once the file is transferred, it is deleted from the staging area.

The CDOutput node is contained in the **File** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:
You can use the CDOutput node in conjunction with another transport input node, for example, MQInput. Any data received from WebSphere MQ is sent to the CDOutput node which writes the file to an internal directory using the file name given on the node. The file is then transferred across to the destination Connect:Direct server, where it is written to the directory and file name given on the node.

Terminals and properties:
The CDOutput node terminals are described in the following table:

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Finish File	The input terminal that accepts a message that triggers the final processing of a file.
Out	The message received on the In terminal is propagated to this terminal if the record is written successfully. The message is unchanged except for status information in the Local Environment.
End of Data	The message received on the Finish File terminal is propagated to this terminal if the file is processed successfully.
Failure	The output terminal to which the message is routed if a failure is detected when a message is propagated.

The following tables describe the node properties that you can set on a specified tab. The column headed M indicates whether the property is *mandatory* (marked in the toolkit with an asterisk if you must enter a value when no default is defined). The column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

When the CDOutput node propagates a message, either to the Out terminal or to the End of Data terminal, it stores information about it in the `LocalEnvironment.WrittenDestination.CD` message tree.

You can override the:

- `Secondary Connect:Direct server (SNODE)name`
- `Process name`
- `Accounting data`
- `Destination file directory`
- `Destination file name`
- `Copy from options including SYSOPTS`
- `Copy to options including SYSOPTS`

by making changes to the `LocalEnvironment.Destination.CD` message tree; see "Using local environment variables with file nodes" on page 1820 for more information.

Description properties

Property	M	C	Default	Description
Node name	No	No	`CDOutput`	The name of the node.
Short Description	No	No	None	A brief description of the node.
Long Description	No	No	None	Text that describes the purpose of the node in the message flow.

Basic properties

Property	M	C	Default	Description	**`mqsiapplybaroverride` command property**
SNODE	No	Yes	The same as the PNODE being used	The secondary Connect:Direct server (SNODE) to which the file is being transferred. If this is not set, the file is transferred back to the primary Connect:Direct server server (PNODE). You must set up a Netmap on the PNODE to ensure that the transfer reaches the SNODE. The Netmap entry for this SNODE must contain the operating system type in the appropriate field. If the PNODE is on a UNIX operating system, you must use the description field in the Netmap to store this information. The description must contain the string remoteos=xxx, where the value is Windows, UNIX or OS390. The CDOutput node cannot be used to transfer to a SNODE on operating systems other than these.	snode
Destination file directory	No	Yes	Empty string	The directory to which the file is being transferred on the secondary Connect:Direct server (SNODE). Note that the directory must exist. If this field is left blank, the default directory is used by the SNODE. On z/OS, if the File name is a sequential dataset, or partitioned dataset member, leave the directory field blank.	destinationDirectory

Property	M	C	Default	Description	mqsiapplybaroverride command property
Destination file name	Yes	Yes	None	The specific file name or a pattern containing a single wildcard that defines the name of the file to be created by the secondary Connect:Direct server (SNODE). The File name can be a pattern that contains a single wildcard. The wildcard value is taken from the element in the local environment folder called `LocalEnvironment.Wildcard.WildcardMatch` . This value is normally set by CDInput or FileInput nodes. For example, if the CDInput node sets its file pattern to `*.in`, it matches the file `test.in` as "`test`". If the CDOutput node's file pattern is set to `*.out`, the "`test`" is substituted to make a file name of `test.out`. The file name must be set, but can be overridden using the local environment field `LocalEnvironment.Destination.CD.Name` You can specify z/OS sequential datasets or partitioned data set members. For whole, sequential data sets, specify the full name for the dataset, for example, `MBUSER.TEST1` For a member within a partitioned dataset, use brackets to specify the member name, for example, `MBUSER.TEST(MEME01)`. One wildcard character can be used anywhere within a dataset name, and works in the same way as for normal file name patterns.	destinationFileName
`CDServer` configurable service	No	Yes	Default	The name of the configurable service being used to connect to the primary Connect:Direct server (PNODE), in order to initiate the transfer. If this value is not set, the default configurable service (named "Default") is used. The default configurable service connects to a PNODE located on the same machine as the broker, and using default port configurations. The default configurable service also uses the "default" security identity, which must be created using the mqsisetdbparms command; for example: `mqsisetdbparms MB7BROKER -n cd::default -u mqbroker -p xxxxxxx` For information on various configurations when using IBM Sterling Connect:Direct, see “Advanced configuration properties when using IBM Sterling Connect:Direct nodes” on page 727 and refer to the Output sections.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Process name	No	Yes	None	The name used for the process script generated to send the file from the primary Connect:Direct server (PNODE) to the secondary Connect:Direct server (SNODE). Use this option if you want to identify this transfer uniquely. If this value is not set, the process name WMBPROC is used. You can use any name you want in the process script. Note, however, that the name must be a maximum of eight characters and cannot contain any spaces.	processName
Disposition	Yes	Yes	`RPL - replace file`	How to create the file on the secondary system: RPL - Replace or create a new file MOD - Append to existing file. This option is not available on z/OS NEW - Create a new file On z/OS, for partitioned datasets, only RPL and NEW options are supported. NEW attempts to allocate a new partitioned dataset, with the following IBM Sterling Connect:Direct SPACE options: `'(23036,(2,1,1))'` You can override these values in the local environment by using: `LocalEnvironment.Destination.CD.Copy.To.Option.SPACE =`	disposition
Transfer Mode	yes	Yes	Binary transfer (no conversion)	The mode in which to transfer the file. binary - The file is transferred as binary, with no conversion text - The file is transferred with conversion between local codepages, as required. When using transfer mode "text" on z/OS, IBM Sterling Connect:Direct requires that the file contains at least one newline character. You must ensure that the file is created in the correct form for IBM Sterling Connect:Direct to successfully complete the text transfer.	transferMode

Request properties

Property	M	C	Default	Description
Data location	No	No	`$Body`	The location in the input message tree that contains the record to be written to the output file. The default value, `$Body`, means the entire message.

Records and Elements properties

Property	M	C	Default	Description
Record definition	Yes	No	`Record is Whole File`	Specifies how records are placed in the output file. Valid options are: • `Record is Whole File` • `Record is Unmodified Data` • `Record is Fixed Length Data` • `Record is Delimited Data`
Length (bytes)	Yes	No	`80`	The required length of the output record. The property is available only when `Record is Fixed Length Data` is specified in `Record definition`.
Padding byte (hexadecimal)	Yes	No	`20`	The 2-digit hexadecimal byte to be used to pad short messages. The property is available only when `Record is Fixed Length Data` is specified in `Record definition`.
Delimiter	Yes	No	`Broker System Line End`	The delimiter to be used. The property is available only when `Record is Delimited Data` is specified in `Record definition`. Valid options are: • `Broker System Line End` • `Custom Delimiter (hexadecimal)`
Custom delimiter (hexadecimal)	Yes	No	None	The delimiter byte sequence to be used. The property is available only when `Record is Delimited Data` is specified in the `Record definition` property, and `Custom Delimiter (hexadecimal)` is specified in the `Delimiter` property.
Delimiter type	Yes	No	`Postfix`	This property specifies how delimiters are to be inserted between records. The property is available only when `Record is Delimited Data` is specified in `Record definition`. Valid options are: • `Postfix` • `Infix`

Validation properties

For a full description of these properties, see "Validation properties" on page 4169.

Property	M	C	Default	Description	**mqsiapplybaroverride command property**
Validate	Yes	Yes	`Inherit`	This property controls whether validation takes place. Valid values are: • `None` • `Content and Value` • `Content` • `Inherit`	validateMaster
Failure action	Yes	No	`Exception`	This property controls what happens if validation fails. The property is available only if you set `Validate` to `Content` or `Content and Value`. Valid values are: • `User Trace` • `Local Error Log` • `Exception` • `Exception List`	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"IBM Sterling Connect:Direct overview and concepts" on page 1810
An overview of IBM Sterling Connect:Direct and its terminology when used with WebSphere Message Broker.

"Local environment tree structure" on page 1056
The local environment tree is a part of the logical message tree in which you can store information while the message flow processes the message.

"Using local environment variables with file nodes" on page 1820
You can use fields in the local environment to dynamically alter the behavior of the FileInput, FileOutput, FTEInput, and FTEOutput nodes. You can also find what values the output nodes used to process the file.

Related tasks:

"Initiating a managed file transfer using IBM Sterling Connect:Direct" on page 1873
Use a CDOutput node to send a file from a specified directory on your primary Connect:Direct server (PNODE) to a filename and directory on a secondary Connect:Direct server (SNODE).

Related reference:

"CDInput node" on page 4305
Use the CDInput node to preview files when using IBM Sterling Connect:Direct in conjunction with WebSphere Message Broker.

"*CDServer* configurable service properties" on page 3798
Select the objects and properties that you want to change for the *CDServer* configurable service.

Check node

Use the Check node to compare the template of a message that is arriving on its input terminal with a message template that you supply when you configure the Check node.

Attention: The Check node is deprecated in WebSphere Message Broker Version 6.0 and subsequent versions. Although message flows that contain a Check node remain valid, redesign your message flows where possible to replace Check nodes with Validate nodes.

This topic contains the following sections:

Purpose:
The message domain, message set, and message type of the message are collectively called the *message template*. The domain defines the parser that is used for the message. The set is the message set to which the message belongs. The type

is the structure of the message itself. You can check the incoming message against one or more of these properties. The message property is checked only if you select its corresponding `Check` property, which means that a message property that contains a null string can be compared.

If the message properties match the specification, the message is propagated to the Match terminal of the node. If the message properties do not match the specification, the message is propagated to the Failure terminal. If the Failure terminal is not connected to some failure handling processing, an exception is generated.

The Check node is contained in the **Validation** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:
Use the Check node to ensure that the message is routed appropriately through the message flow. For example, you can configure the node to direct a message that requests stock purchases through a different route from that required for a message that requests stock sales.

Another example of this node's use is for the receipt of electronic messages from staff at your head office. These messages are used for multiple purposes; for example, to request technical support or stationery, or to advise you about new customer leads. These messages can be processed automatically because your staff complete a standard form. If you want these messages to be processed separately from other messages received, use the Check node to ensure that only staff messages with a specific message type are processed by this message flow.

Terminals and properties:
When you have put an instance of the Check node into a message flow, node into a message flow, you can configure it. For more information, see . The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The Check node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the message is routed if the incoming message does not match the specified properties.
Match	The output terminal to which the message is routed if the incoming message matches the specified properties.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Check node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	Check	The name of the node
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The Check node Basic properties are described in the following table.

Property	M	C	Default	Description
Domain	No	No		The name of the domain.
Check domain	Yes	No	Cleared	This property checks that a message belongs to a particular domain. To check the parser that is to be used for the incoming message, select this check box and select one of the values from the `Domain` list.
Set	No	No		The message set to which the incoming message belongs. If you are using the MRM, IDOC, or XMLNSC parser, check that the incoming message belongs to a particular message set by selecting `Check set` and entering the name of the message set in `Set`. Leave `Set` clear for other parsers. If you set this property, then subsequently update the project dependencies to remove this message set reference, a warning is issued. Either update the `Message set` property, or restore the reference to this message set project.
Check set	Yes	No	Cleared	If you select this check box, the incoming message is checked against the `Set` property.
Type	No	No		The message name. If you are using the MRM parser, check that the incoming message is a particular message type by selecting `Check type` and entering the name of the message in `Type`. Leave `Type` clear for other parsers.
Check type	Yes	No	Cleared	If you select this check box, the incoming message is checked against the `Type` property.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Validating messages" on page 1478
The broker provides validation based on the message model for predefined messages.

Related reference:

"Validate node" on page 4959
Use the Validate node to check that the message that arrives on its input terminal is as expected. You can use this node to check that the message has the expected message template properties, and to check that the content of the message is correct by selecting message validation.

CICSRequest node

Use the CICSRequest node to call CICS Transaction Server for z/OS programs over TCP/IP-based IP InterCommunications (IPIC) protocol.

This topic contains the following sections:

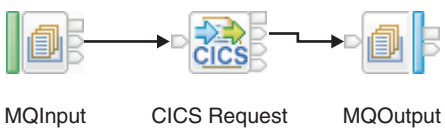
- "Purpose"
- "Using the CICSRequest node in a message flow"
- "Configuring the CICSRequest node" on page 4323
- "Local environment overrides" on page 4323
- "Terminals and properties" on page 4323

Purpose:

You can use the CICSRequest node to connect WebSphere Message Broker to CICS applications. The CICSRequest node communicates with CICS by sending Distributed Program Link (DPL) requests over TCP/IP-based IP InterCommunications protocol (IPIC). You can create a message flow that contains a CICSRequest node, which calls a CICS application on a targeted region. The CICSRequest node can be used by a message flow that is deployed to any broker platform. The CICSRequest node supports CICS Transaction Server for z/OS Version 3.2 and later.

Using the CICSRequest node in a message flow:

One possible use of a CICSRequest node is to connect to a CICS application by using a synchronous style of message flow. You can achieve this connection by creating the following message flow:



The CICSRequest node support in WebSphere Message Broker provides direct communication with CICS (two-tier connection) by sending Distributed Program Link (DPL) requests over TCP/IP-based IPIC, or communication with CICS through CICS Transaction Gateway for Multiplatforms (three-tier connection). For more information about the two-tier and three-tier connection models, see "CICS Transaction Server for z/OS overview" on page 2173 for a high-level overview, or "CICS Transaction Server for z/OS two-tier connectivity" on page 2177 and "CICS Transaction Server for z/OS three-tier connectivity" on page 2181 for detailed conceptual information.

You can specify either a COMMAREA data structure or a channel data structure on the CICSRequest node to use as input for linking to CICS programs. The data structure that is specified as input returns the same data structure as output. Channels are an alternative for COMMAREAs, providing relief from the COMMAREA maximum size of `32767` bytes, and allowing greater flexibility in input/output data structures. For more information about using a COMMAREA or channel data structure, see "COMMAREA or channel data structures" on page 2183.

CICS channels hold a number of structures called containers. In WebSphere Message Broker, a CICS channel is represented as a message collection structure. A message collection can hold child messages, each treated as a container by the CICSRequest node. For information about using ESQL to create a message collection, see "Creating a message collection by using ESQL" on page 2758.

If a single container is required for input only, a message collection does not need to be constructed. Instead a regular message can be used, provided the 16-character maximum alphanumeric channel name and the single 16-character maximum alphanumeric container name are specified in the local environment. For more information about using single message mode, see "COMMAREA or channel data structures" on page 2183.

Because it is not possible to know how many containers are in the response, a message collection is always produced as output. However, the CICSRequest node `Result data location` property can be used to reduce the result tree down to a single message folder, or down to a single field or subtree for output.

You can add name-value attributes to a message collection to create CICS containers. Name-value attributes in the message collection, apart from *`CollectionName`*, can be used in lieu of full message-folders for simple data. For example, a name-value string attribute can be set in the message collection and used directly by the CICSRequest node without needing to create a message set for the element. For information about using attributes, see "COMMAREA or channel data structures" on page 2183.

Name-value attributes can be produced from containers on output, as well as accepted for input. To create an attribute instead of a message folder from a container, select the check box that is available on the Response Message Parsing tab.

The CICSRequest node is contained in the **CICS** drawer of the message flow node palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

View the following sample to see how to use the CICSRequest node:

- CICS Transaction Server for z/OS Connectivity

View the following sample to see how to call a channel-based CICS program by creating and populating a message collection for the CICSRequest node, and how to process the collection after the call:

- CICS Transaction Server for z/OS Channel Connectivity

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

The CICSRequest node can use an identity that is present on an input message, and propagate it to CICS, by using the `Propagate` property on the security profile that is defined for the node. For more information, see "Propagating security credentials to CICS Transaction Server for z/OS" on page 2208 and "Identity and security token propagation" on page 426.

You can specify a mirror transaction name on the CICSRequest node for CICS tasks and programs to run under. This grouping greatly assists stat collection, accounting, and aids decision making about task priority. For more information about mirror transactions, see "CICS Transaction Server for z/OS mirror transactions" on page 2189.

Using configurable services for the CICSRequest node

You can configure the CICSRequest node to get connection details from a configurable service. For details about creating, changing, reporting, and deleting the configurable services, see "Changing connection information for the CICSRequest node" on page 738.

Configuring the CICSRequest node:
When you have put an instance of the CICSRequest node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view.

All mandatory properties for which you must enter a value (properties that do not have a default value defined) are marked with an asterisk.

Local environment overrides:
You can dynamically override values, on a per message basis, in the local environment. You can override the CICS program that you are calling, the COMMAREA length, mirror transactions, and the processing of the response message. For more information, see "Local environment overrides for the CICSRequest node" on page 2191.

Terminals and properties:
The CICSRequest node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node. The CICSRequest node is driven by a message arriving on the In terminal. Data is taken from the message tree and is sent to CICS.
Out	The output terminal from which the message tree is propagated, including the data returned from CICS.
Failure	The output terminal to which a message is routed if a CICSRequest node exception is detected, or a CICSRequest node to CICS connection failure occurs.
Error	The output terminal to which a message is propagated if a CICS error (abend) occurs. The input message is propagated with a CICS\AbendCode field in the LocalEnvironment. If the Error terminal is not connected and CICS abend occurs, the abend is lost.
Timeout	The output terminal to which the message is propagated if a per-request timeout occurs when an individual request is sent to CICS, but the request takes too long. The output message contains the input message body and a timeout exception in the ExceptionList. If the Timeout terminal is not connected and a timeout occurs, the request timeout exception is routed to the Failure terminal. If the Failure terminal is not connected, the broker throws an exception and returns control to the closest upstream node that can process it. The default behavior is that the message is returned to the input node.

The following tables describe the CICSRequest node properties.

The table column headed M indicates whether the property is *mandatory*. For example, the property is marked with an asterisk meaning that you must enter a value if no default is defined.

The column headed C indicates whether the property is *configurable*. For example, you can change the value when you add the message flow to the BAR file to deploy it.

The CICSRequest node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	Yes	No	CICS Request	The name of the node.
Short description	No	No	None	A brief description of the node.
Long description	No	No	None	Text that describes the purpose of the node in the message flow.

The CICSRequest node Basic properties are described in the following table.

Property	M	C	Default	Description	**mqsiapplybaroverride command property**
CICS server	Yes	Yes	None	The `CICS server` property is defined either as a configurable service name, for example: *myCICSConnection*, or as a URL. A URL allows you to specify a protocol, host name, and port number, which is the minimum information you need to connect to the CICS target region. The URL must be made up of the following structure: *protocol*`://`*hostname*`:`*port* Where: • *protocol* can be `tcp` or `ssl`. • *hostname* is the Internet Protocol version 4 (IPv4) TCP/IP address or DNS-resolvable hostname of the CICS host. • *port* is the port number of the TCPIPSERVICE listener in CICS that is listening for IPIC requests over TCP/IP or Secure Sockets Layer (SSL) protocol. You can enter an integer in the range 1 - 65535. For example: `tcp://mycicsregion.com:12345` or `ssl://mycicsregion.com:56789`. You can obtain the *hostname* and *port* values from the IPIC TCPIPSERVICE definition in the target CICS region.	cicsServer
Program name	Yes	No	None	The Program name property is the name of the application that is on the target CICS region that you are calling. Programs that use COMMAREAs or channels are supported. You can override this property in the local environment by specifying a value in the following location: `$LocalEnvironment/Destination/CICS/CICSProgramName`	
Data structure	Yes	No	Commarea	Whether to use a COMMAREA or a channel data structure. The default for this property is `Commarea`. The decision depends on the targeted CICS program, for example; whether the target program is channel-based or not.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Commarea length	Yes	No	0	The Commarea length property is the size, in bytes, of the COMMAREA that is used by the CICS program. The byte size value is sent to CICS, and before the program is started, an area of memory is created to match that number. For example, if you send a Commarea length value of 100, 100 bytes are allocated. The program accesses this area as the DFHCOMMAREA. Ensure that the Commarea length property value is large enough to contain the input request data, or the output response data, but that it does not exceed the maximum value of 32767 bytes. If the Commarea length value is not large enough to be used for the response data, or the request data, a memory leak occurs in CICS. The size of the COMMAREA cannot be changed by the CICS program. If the serialized request data is larger than the Commarea length, the data is truncated to the Commarea length. You can obtain the Commarea length value from the CICS administrator or developer. You can override this property in the local environment by specifying a value in the following location: $LocalEnvironment/Destination/CICS/CICSCommareaLen This property is not configurable if a value of Channel is selected for the Data structure property.	
Security identity	No	Yes	None	The name of the security identity object that is created and configured by the **mqsisetdbparms** command, which contains the user ID and password to be used by the broker to authenticate the connection to CICS. Use the **mqsisetdbparms** command to set the security identity user ID and password to be accessed by the broker. The default value for this property is None, which signifies that the user ID and password are not passed to CICS. For more information about CICS security identity support, see “**mqsisetdbparms** command” on page 3954.	securityIdentity

Property	M	C	Default	Description	mqsiapplybaroverride command property
Request timeout (in seconds)	Yes	Yes	120	If individual requests sent to CICS take more time than specified here, the request attempt is abandoned. Because the CICSRequest node is non-transactional, the state of the work in CICS cannot be guaranteed and might need to be manually purged. A request timeout causes an exception to be propagated from the Timeout terminal. If the Timeout terminal is not connected, the request timeout exception is routed to the Failure terminal. If the Failure terminal is not connected, the broker throws an exception and returns control to the closest upstream node that can process it. The default behavior is that the message is returned to the input node. A value of 0 indicates that no timeout occurs.	requestTimeoutSecs
Mirror transaction ID	No	Yes	None	You can specify a mirror transaction name by configuring this property, however the `Mirror transaction ID` property value that you specify must correspond to a defined TRANSACTION resource in CICS. For example, if you have a defined TRANSACTION resource of `ATRN` in CICS, and you want tasks and programs to run under that transaction, you must configure `ATRN` as the `Mirror transaction ID` property value. When the message flow containing the configured CICSRequest node is deployed, any CICS programs that are started thereafter appear in CICS as running under the specified mirror transaction. If the value of the `Mirror transaction ID` property is not set, the mirror transaction name defaults to `CPMI` if called by a distributed platform, or `CSMI` if called by a z/OS system. For more information about mirror transactions, see “CICS Transaction Server for z/OS mirror transactions” on page 2189.	mirrorTran

Property	M	C	Default	Description	mqsiapplybaroverride command property
Set EIBTRNID only	No	Yes	Cleared	You can use a weaker form of mirror transaction that does not change the TRANSACTION resource, but instead sets a variable called `EIBTRNID`, which is available to the called program. You can configure the `EIBTRNID` variable to tell the program what TRANSACTION resource it is running under, without the TRANSACTION resource being defined in CICS. For example, you can specify this weaker form of mirror transaction by configuring the `Mirror transaction ID` property with the name of the required TRANSACTION resource; for example `ATRN`, and by selecting this property. When the message flow containing the configured CICSRequest node is deployed, any CICS programs that are started thereafter appear in CICS as running under the specified mirror transaction. If the value of the `Mirror transaction ID` property is not set, the mirror transaction name defaults to `CPMI` if called by a distributed platform, or `CSMI` if called by a z/OS system. For more information about mirror transactions, see "CICS Transaction Server for z/OS mirror transactions" on page 2189.	eibtrnidOnly

The CICSRequest node Request properties are described in the following table.

Property	M	C	Default	Description
Data location	Yes	No	`$Body`	The location in the incoming message tree from which data is retrieved to form the request that is sent from the CICSRequest node to CICS. The default value, `$Body`, represents the incoming message body. You can enter any XPath or ESQL expression that defines the location of the message tree to serialize and send to CICS.
Program name override location	No	No	`$LocalEnvironment/Destination/CICS/CICSProgramName`	The element of the message assembly that contains the name of the program to run in CICS.

The CICSRequest node Result properties are described in the following table.

Property	M	C	Default	Description
Result data location	No	No	`$ResultRoot`	This property specifies which subtree of the result to place in the message. If a value is not specified for this property, `$ResultRoot` is used as the default and the whole response is placed in the output message at the location that is specified in `Output data location`.
Output data location	No	No	`$OutputRoot`	The message tree location to which the CICSRequest node sends output. The default value, `$OutputRoot`, replaces the incoming message with the response.

Property	M	C	Default	Description
Copy local environment	No	No	Selected	This property controls whether to copy the incoming local environment or propagate the incoming local environment. By default, this check box is selected, which signifies that the local environment is copied so that the incoming local environment is preserved. The additions to the local environment are visible only to nodes downstream of this node. If this check box is cleared, the incoming local environment is used for the outgoing message. Any modifications that are made to the local environment by this node are visible to both downstream and upstream nodes after this node has completed.

The CICSRequest node default Response Message Parsing properties are described in the following table.

Property	M	C	Default	Description
Message domain	No	No	None	**If using a COMMAREA data structure:** The domain to use to parse the response COMMAREA bit stream. **If using a channel data structure:** The domain to use to parse the response container bit stream. You can override this property in the local environment by specifying a value in the following location: `$LocalEnvironment/Destination/CICS/Response/messageDomain`
Message set	No	No	None	**If using a COMMAREA or channel data structure:** The name of the message set in which the response message is defined. If you set this property, and then update the project dependencies to remove this message set reference, a warning is issued. Either update the `Message set` property, or restore the reference to this message set project. You can override this property in the local environment by specifying a value in the following location: `$LocalEnvironment/Destination/CICS/Response/messageSet`
Message type	No	No	None	**If using a COMMAREA or channel data structure:** The type of the response message. You can override this property in the local environment by specifying a value in the following location: `$LocalEnvironment/Destination/CICS/Response/messageType`
Message format	No	No	None	**If using a COMMAREA or channel data structure:** The name of the physical format of the response message. You can override this property in the local environment by specifying a value in the following location: `$LocalEnvironment/Destination/CICS/Response/messageFormat`

Property	M	C	Default	Description
Message coded character set ID	No	No	`EBCDIC (037)`	**If using a COMMAREA data structure:** The ID of the coded character set (CCSID) that is used to interpret bytes of the response COMMAREA. This property defines the CCSID for the message that is returned from CICS. **If using a channel data structure:** The ID of the coded character set (CCSID) that is used to interpret bytes of the response container, however this is ignored for character containers. You can override this property in the local environment by specifying a value in the following location: `$LocalEnvironment/Destination/CICS/Response/messageCCSID` For a list of valid values, see "Supported code pages" on page 4176.
Message encoding	No	No	`Big Endian, with S390 Floating Point (785)`	**If using a COMMAREA data structure:** The encoding scheme for numbers and large characters that is used to interpret bytes of the response COMMAREA. This property defines the message encoding for the message that is returned from CICS. **If using a channel data structure:** The encoding scheme for numbers and large characters that is used to interpret bytes of the response container, however this is ignored for character containers. Valid values are: • Little Endian, with IEEE Floating Point (546) • Big Endian, with IEEE Floating Point (273) • Big Endian, with S390 Floating Point (785) (the default value) • Broker System Determined For more information about encoding, see "Data conversion" on page 1151. You can override this property in the local environment by specifying a value in the following location: `$LocalEnvironment/Destination/CICS/Response/messageEncoding`

Channel Options

If a value of `Channel` is selected for the `Data structure` property on the Basic tab, an additional table is available on the Response Message Parsing tab named Channel Options. The Channel Options table allows you to override the default message template properties by specifying parsing options for individual response containers, on a per-folder basis, when output from the CICSRequest node. You can also create a name-value attribute in the message collection to hold the container by selecting the check box provided.

If the container details are specified in the Channel Options table, those container details are used in preference to the default message template properties. If the container details are not specified in the Channel Options table and the attribute check box is selected, a name-value pair is created.

If the container details are not specified in the Channel Options table and the name-value attribute check box is not selected, the default message template properties are used.

The CICSRequest node Parser Options properties are described in the following table.

Property	M	C	Default	Description
Parse timing	Yes	No	`On Demand`	This property controls when a response message is parsed. Valid values are `On Demand`, `Immediate`, and `Complete`. For a full description of this property, see "Parsing on demand" on page 4173.
Build tree using XML schema data types	Yes	No	Cleared	This property controls whether the XMLNSC parser creates syntax elements in the message tree with data types taken from the XML Schema. You can select this property only if you set the `Validate` property on the **Validation** tab to `Content` or `Content and Value`.
Use XMLNSC compact parser for XMLNS domain	Yes	No	Cleared	This property controls whether the XMLNSC Compact Parser is used for messages in the XMLNS Domain. If you set this property, the response message data appears under XMLNSC in nodes that are connected to the output terminal when the input MQRFH2 header or Response Message Parsing properties Domain is XMLNS.
Retain mixed content	Yes	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters mixed text in a response message. If you select the check box, elements are created for mixed text. If you clear the check box, mixed text is ignored and no elements are created.
Retain comments	Yes	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters comments in a response message. If you select the check box, elements are created for comments. If you clear the check box, comments are ignored and no elements are created.
Retain processing instructions	Yes	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters processing instructions in a response message. If you select the check box, elements are created for processing instructions. If you clear the check box, processing instructions are ignored and no elements are created.
Opaque elements	No	No	Blank	This property is used to specify a list of elements in the response message that are to be opaquely parsed by the XMLNSC parser. Opaque parsing is performed only if validation is not enabled (that is, if `Validate` is `None`); entries that are specified in `Opaque Elements` are ignored if validation is enabled.

The CICSRequest node Validation properties are described in the following table.

For a full description of these properties see "Validation properties" on page 4169.

Property	M	C	Default	Description	**mqsiapplybaroverride command property**
Validate	Yes	Yes	`None`	This property controls whether validation takes place. Valid values are `None`, `Content and Value`, `Content`, and `Inherit`.	validateMaster

Property	M	C	Default	Description	mqsiapplybaroverride command property
Failure action	Yes	No	Exception	This property controls what happens if validation fails. You can set this property only if you set Validate to Content or Content and Value. Valid values are User Trace, Local Error Log, Exception, and Exception List.	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"CICS Transaction Server for z/OS overview" on page 2173
CICS Transaction Server for z/OS provides general-purpose transaction processing software for z/OS. CICS is a powerful application server that meets the transaction-processing needs of both large and small enterprises.

"CICS Transaction Server for z/OS connectivity" on page 2174
Use the CICSRequest node to connect WebSphere Message Broker with CICS Transaction Server for z/OS applications.

"CICS Transaction Server for z/OS mirror transactions" on page 2189
You can use a mirror transaction to group CICS Transaction Server for z/OS tasks and programs together. This grouping greatly assists stat collection, accounting, and aids decision making about task priority.

"Configurable services" on page 1296
Configurable services are typically runtime properties. You can use them to define properties that are related to external services on which the broker relies; for example, an SMTP server or a JMS provider.

"Identity and security token propagation" on page 426
Identity and security token propagation enables the identity and security tokens (associated with each message) to be propagated throughout a message flow, and on to target applications through output or request nodes.

"Resource statistics" on page 3306
Resource statistics are collected by a broker to record performance and operating details of resources that are used by execution groups.

"Samples" on page 98
The WebSphere Message Broker Toolkit provides samples that show the features that are available in WebSphere Message Broker, and how to use them. This topic provides links to the information about the individual samples.

Related tasks:

"Connecting to a CICS Transaction Server for z/OS application" on page 2192
Connecting to a CICS Transaction Server for z/OS application involves creating a message flow, building a message, and processing the response from the CICSRequest node.

"Changing connection information for the CICSRequest node" on page 738
You can create a configurable service that the CICSRequest node or message flow refers to at run time for connection information, instead of defining the connection properties on the node or the message flow. The advantage being that you can change the host name and performance values without needing to redeploy your message flow.

"Securing the connection to CICS Transaction Server for z/OS by using SSL" on page 547
Configure the CICSRequest node to communicate with CICS Transaction Server for z/OS over the Secure Sockets Layer (SSL) protocol by updating a CICSConnection configurable service or the CICSRequest node to use SSL.

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

"Validating messages" on page 1478
The broker provides validation based on the message model for predefined messages.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

Collector node

Use the Collector node to create message collections based on rules that you configure in the node.

This topic contains the following sections:

Purpose:

Use the Collector node to create a message collection from one or more sources based on configurable criteria. For example, you might need to extract, combine, and transform information from three different sources. The messages from these different sources might arrive at the input terminals at different times and in an unknown order. A collection is defined by configuring an event handler for each input terminal. Each event handler controls the acceptance of a message into a collection according to the following properties:

- Number of messages
- Collect messages for a set period
- Match the contents of a correlation path
- Match the contents against a correlation pattern

The correlation properties allow collections to be made according to the content of the messages. The content is specified by using an XPath expression. The Collector node ensures that each collection contains an identical correlation string across all its inputs. For more information about XPath 1.0 query syntax, see W3C XPath 1.0 Specification.

A message collection is created when the first message arrives at any of the dynamic input terminals on the Collector node. Message collections are stored on a WebSphere MQ queue.

When the conditions set by the event handlers for a message collection have been met, the message collection is complete and ready to be propagated. For example, if you set the event handlers on the Collector node to wait for two messages from each input terminal, the message collection is complete when two messages have been received by every terminal. When the next message arrives on an input terminal, it is added to a new message collection. You can select from a number of options to determine how the propagation of the message collection is coordinated. You can specify that the message collection is propagated automatically for processing, or alternatively that the message collection is propagated when a control message is received.

You can also set an expiry timeout for message collections that fail to be completed in a satisfactory time by using a property on the Collector node. The timeout starts when the first message is added to a message collection. If the timeout expires before the message collection is complete, the incomplete message collection is propagated to the **Expire** terminal. Set a value for the collection expiry to ensure that incomplete message collections do not remain stored on a queue indefinitely. Add appropriate processing to your message flow to handle incomplete message collections.

The Collector node is contained in the **Routing** drawer of the message flow node palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:
Look at the following sample to see how to use this node:

- Collector Node

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Use the Collector node to group messages from different input sources for further processing. A message collection can be processed by the following nodes only:

- Compute
- JavaCompute
- PHPCompute

Errors might be generated if you try to process a message collection with a node that does not support message collections.

The Collector node has one static input terminal, **Control**, and four static output terminals: **Out**, **Expire**, **Failure** and **Catch**. These static terminals are always present on the node. In addition to the static input and output terminals, you can add dynamic input terminals for each input source that you want to use with the Collector node.

You can add and configure as many input terminals as required to the Collector node. You can configure the properties of each input terminal separately to control how the messages received on each input terminal are added to the appropriate message collection.

You can use the **Control** terminal to trigger the output of completed message collections from the Collector node. Configure the `Event coordination` property to set the behavior of the Collector node when messages are received on the **Control** terminal.

When a message collection is successfully completed, it is ready to be propagated to the **Out** terminal. If a value greater than zero is set on the `Collection expiry` property, any incomplete message collections are propagated to the **Expire** terminal.

A new transaction is created when a message collection is complete, and is propagated to the next node. Any exceptions that are caught from downstream nodes cause the message collection to be propagated to the **Catch** terminal on the Collector node, together with the exception list. If the **Catch** terminal is not connected to any other nodes, the transaction is caused to roll back. Messages in the message collection are backed out onto the queue of the Collector node. The exception list is written to the system log. This step is repeated until the message collection is successfully processed. To avoid an exception that causes the message collection to fail to be propagated successfully, ensure that you connect the **Catch** terminal to a flow to handle any exceptions. Also, ensure that you set an expiry timeout to propagate incomplete message collections.

Note: Any exceptions that occur downstream of the Collector node are routed to the **Catch** terminal. The exception is not processed any further upstream because the completion of the message collection in the Collector node is the start of the transaction. This behavior is like the AggregateReply node. Do not connect a Throw node to the **Catch** terminal of the Collector node, because control is returned to the same **Catch** terminal.

If you use additional instances of a message flow or multiple inputs to the Collector node, you can use the `Correlation path` and `Correlation pattern` properties to ensure that related messages are added to the same message collection. If you use additional instances, or multiple inputs to the Collector node the order of messages in the message collection can be unpredictable. The order of messages is also unpredictable if you use WebSphere MQ cluster queues as inputs to the Collector node.

Configuring the Collector node:
When you have put an instance of the Collector node into a message flow, you can configure it; see “Configuring the Collector node” on page 2767. The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those properties that do not have a default value defined) are marked with an asterisk.

Terminals and properties:
The Collector node terminals are described in the following table.

Terminal	Description
Control	The static input terminal that accepts control messages. Any message received by the Control terminal is treated as a control message.

Terminal	Description
Out	The output terminal to which the complete message collection is routed if the received messages satisfy the configured conditions for the message collection.
Expire	The output terminal to which the incomplete message collection is routed if the received messages do not satisfy the configured conditions within the time specified on the Collection expiry property. If you have not set a value for the Collection expiry property this terminal is not used.
Failure	The output terminal to which the message collection is routed if a failure is detected during processing.
Catch	The output terminal to which the message collection is routed if an exception is thrown downstream and caught by this node.

The Collector node can have further dynamic input terminals. You can create numeric terminal labels for the Collector node; however, the Compute node does not support numeric labels. Therefore, when you are defining a custom terminal for the Collector node, ensure that the name begins with an alphabetic character.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Collector node Description properties are described in the following table:

Property	M	C	Default	Description
Node name	No	No	The node type, Collector	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The Collector node has two types of Basic properties. You can set properties for each dynamic input terminal that you add to the Collector node in the Collection Definition table in the Basic properties. The properties in the Collection Definition table define the event handlers for the messages arriving on the individual input terminals. The properties that you can set for each of the dynamic input terminals are described in the following table:

Property	M	C	Default	Description
Terminal	Yes	No	The terminal name	`Terminal` is not a property of the node, but a label to show the name of the dynamic input terminal. Enter values for the event handler properties for each dynamic input terminal that you have added to the Collector node in the Collection Definition table.

Property	M	C	Default	Description
Quantity	Yes	No	1	This property specifies the number of messages that the input terminal accepts to add to a message collection. The default value is 1; if you accept this default value, only one message is added to a collection. If a second message is received on the terminal, a new collection instance is created for it. If you select `0` (zero) or do not specify a value, there is no limit to the number of messages accepted. In this case, the value that is set on the `Timeout` property must be greater than zero. If you set both the `Timeout` and `Quantity` properties to values that are greater than zero, the terminal stops accepting messages when the first of the two thresholds is reached. When this threshold is reached, and other criteria have also been satisfied, the collection is complete and it is propagated to the Out terminal.
Timeout	Yes	No	0	This property specifies the maximum time in seconds for which the input terminal accepts messages. If you select `0` (zero), the timeout is disabled and there is no limit on the time to wait for messages. In this case, the value that is set on the `Quantity` property must be greater than zero. If you set both the `Timeout` and `Quantity` properties to values that are greater than zero, the terminal stops accepting messages when the first of the two thresholds is reached. When this threshold is reached, and other criteria have also been satisfied, the collection is complete and it is propagated to the Out terminal.
Correlation path	No	No		Messages are only accepted into a message collection if they have the same correlation string. If the message has a different correlation string, it is offered to the next collection in the queue. If none of the collections accept the message, a new collection is created with correlation string set to the value of the correlation string in the message. Messages are grouped by the value from the correlation path. The correlation path is defined by using XPath. You can define your own correlation path by using XPath, or select from the following predefined paths: • `$LocalEnvironment/Wildcard/WildcardMatch` • `$Root/MQMD/CorrelId` • `$LocalEnvironment/FileInput/Name` • `$Root/JMSTransport/Transport_Folders/Header_Values/JMSCorrelationID` If you define a value for `Correlation path`, you can optionally configure a `Correlation pattern`.
Correlation pattern	No	No		This property specifies a pattern to match the contents of a correlation path value against. You must set the `Correlation path` property before you set the value for the `Correlation pattern` property. If you set the correlation pattern, you must use one `*` character, optionally surrounded by other text. For example, `*.dat`. If the correlation pattern is blank, the entire text from the correlation path must be matched by the incoming message.

The remaining Basic properties for the Collector node are shown in the following table:

Property	M	C	Default	Description	mqsiapplybaroverride command property
Collection name	No	No		This property specifies the name of the message collection. • If you set this property to contain the wildcard *, the wildcard is replaced by the correlation string from the relevant event handler. • If you leave this property blank or use * and the correlation string is empty, then the collection name is set to an empty string.	
Collection expiry	No	Yes		The amount of time, in seconds, that the Collector node waits for messages to arrive. After this time an incomplete message collection is expired and propagated to the Expire output terminal. If you set this property to zero, the collection expiry is disabled and the Collector node waits for messages indefinitely. Set a value greater than zero to ensure that the message collection is processed, even if not all messages are received. A warning is issued if this property is not set. This property overrides the `Timeout` properties that are set on the input terminals. The `Timeout` property specifies the maximum time in seconds for which the input terminal accepts messages. After this time, the collection is complete and is propagated to the Out terminal. If one input terminal does not have a timeout set, or if its timer has not started, a message collection might wait for an input message indefinitely. In this situation, the `Collection expiry` property is used to ensure that the incomplete message collection is not left in an unresolved state, but is propagated to the Expiry terminal. This property is overridden by the **`Collection expiry`** property, if set, in the Collector configurable service.	collectionExpiry

The Collector node Advanced properties are described in the following table:

Property	M	C	Default	Description
Persistence mode	No	No	The node type, Collector	This property specifies whether messages are stored on the queues of the Collector node persistently.

Property	M	C	Default	Description
Event coordination	Yes	No	`Disabled`	This property specifies how messages received on the Control terminal for event coordination processing are handled in the Collector node. • If you accept the default value (`Disabled`), messages to the Control terminal are ignored and collections are propagated when they are complete. • If you select `All complete collections`, complete message collections are held on a WebSphere MQ queue. When a message is received on the control terminal, all complete message collections on the WebSphere MQ queue are propagated to the Out terminal. • If you select `First complete collection`, complete message collections are held on a WebSphere MQ queue. When a message is received on the control terminal, the first complete message collection on the WebSphere MQ queue is propagated to the Out terminal. The collections are propagated in the same order that they become complete. If the WebSphere MQ queue is empty when the message is received on the Control terminal, the next complete message collection is immediately propagated to the Out terminal.
Configurable service	No	Yes	None set	This property specifies the name of the Collector configurable service to be used by the Collector node. The properties set by the Collector configurable service override the equivalent properties set on the Collector node. For more information about the properties than you can set with this configurable service, see "Configurable services properties" on page 3766.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

You cannot use monitoring properties to configure transaction events on the following nodes:

"Collector node" on page 4333

"Resequence node" on page 4651

Use a monitoring profile instead; see "Configuring monitoring event sources using a monitoring profile" on page 762.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message collections" on page 2755
A message collection is a single message that contains multiple messages derived from one or more sources.

Related tasks:

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

"Creating a flow that uses message collections" on page 2764
Use a Collector node in your message flow to group messages from one or more sources into a message collection. You can add dynamic input terminals to your Collector node for each message source that you want to configure for your message flow.

"Configuring the Collector node" on page 2767
You can configure the Collector node to determine how messages are added to message collections. You can also use properties on the Collector node to control when message collections are propagated.

"Adding an input terminal to a Collector node for each input source" on page 2768
Add new dynamic input terminals to the Collector node for all of the sources of messages for your message collections.

"Using message collections" on page 2755
A message collection is a single message that contains multiple messages derived from one or more sources. You can use a Collector node to group together messages from one or more sources into a message collection, so that they can be processed together by downstream nodes.

Compute node

Use the Compute node to construct one or more new output messages.

This topic contains the following sections:

Purpose:

The output messages that you create in the Compute node might be created by modifying the information that is provided in the input message, or by using only new information which can be taken from a database or from other sources. Elements of the input message (for example, headers, header fields, and body data), its associated environment, and its exception list can be used to create the new output message.

Specify how the new messages are created by coding ESQL in the message flow ESQL resource file. For more information, see “Specifying ESQL” on page 4342.

Use the Compute node to:
- Build a new message using a set of assignment statements
- Copy messages between parsers
- Convert messages from one code set to another
- Transform messages from one format to another

The Compute node is contained in the **Transformation** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:
Look at the following samples to see how to use this node:
- Airline Reservations
- Aggregation
- JMS Nodes
- Large Messaging
- Message Routing
- Scribble
- Timeout Processing
- Video Rental

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Consider a message flow in which you want to give each order that you receive a unique identifier for audit purposes. The Compute node does not modify its input message; it creates a new, modified copy of the message as an output message. You can use the Compute node to insert a unique identifier for your order into the output message, which can be used by subsequent nodes in the message flow.

Configuring the Compute node:
When you have put an instance of the Compute node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view.

All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

Configure the Compute node by:
1. “Defining database interaction”
2. “Specifying ESQL” on page 4342
3. “Setting the mode” on page 4344
4. “Validating messages” on page 4346

Defining database interaction:
To access a database from this node:
- On the **Basic** tab, specify in `Data Source` the name by which the appropriate database is known on the system on which this message flow is to run. The

broker connects to this database with user ID and password information that you have specified on the **mqsicreatebroker**, **mqsichangebroker**, or **mqsisetdbparms** command.

z/OS On z/OS systems, the broker uses the broker started task ID, or the user ID and password that were specified on the **mqsisetdbparms** command JCL, BIPSDBP in the customization data set <hlq>.SBIPPROC.

- Select the `Transaction` setting from the list. The values are:
  - `Automatic` (the default). The message flow, of which the Compute node is a part, is committed if it is successful. That is, the actions that you define in the ESQL module are performed on the message and it continues through the message flow. If the message flow fails, it is rolled back. If you choose `Automatic`, the ability to commit or roll back the action of the Compute node on the database depends on the success or failure of the entire message flow.
  - `Commit`. To commit the action of the Compute node on the database, irrespective of the success or failure of the message flow as a whole, select `Commit`. The database update is committed even if the message flow itself fails.

  The value that you choose is implemented for the one or more database tables that you have added; you cannot select a different value for each table.
- To treat database warning messages as errors and have the node propagate the output message to the Failure terminal, select `Treat warnings as errors`. The check box is cleared initially.

  When you select the check box, the node handles all positive return codes from the database as errors and generates exceptions in the same way as it does for the negative, or more serious, errors.

  If you do not select the check box, the node treats warnings as typical return codes, and does not raise any exceptions. The most significant warning raised is `not found`, which can be handled as a typical return code safely in most circumstances.
- To force the broker to generate an exception when a database error is detected, select `Throw exception on database error`. The check box is selected initially.

  If you clear the check box, you must include ESQL to check for any database error that might be returned after each database call that you make (you can use SQLCODE and SQLSTATE to do this). If an error occurs, you must handle the error in the message flow to ensure the integrity of the broker and the database; the error is ignored if you do not handle it through your own processing because you have chosen not to invoke the default error handling by the broker. For example, you can include the ESQL THROW statement to throw an exception in this node, or you can use the Throw node to generate your own exception at a later point in the message flow.

Specifying ESQL:
Code ESQL statements to customize the behavior of the Compute node. For example, you can customize the node to create a new output message or messages, using input message or database content (unchanged or modified), or new data. For example, you might want to modify a value in the input message by adding a value from a database, and storing the result in a field in the output message.

Code the ESQL statements that you want in an ESQL file that is associated with the message flow in which you have included this instance of the Compute node. The ESQL file, which by default has the name `<message_flow_name>.esql`, contains ESQL for every node in the message flow that requires it. Each portion of code that is related to a specific node is known as a module.

If an ESQL file does not already exist for this message flow, double-click the Compute node, or right-click the node and click **Open ESQL**. This action creates and opens a new ESQL file in the ESQL editor view. If you prefer, you can open the appropriate ESQL file in the Broker Development view and select this node in the Outline view.

If the file exists already, click **Browse** beside the `ESQL Module` property to display the Module Selection dialog box, which lists the available Compute node modules defined in the ESQL files that are accessible by this message flow (ESQL files can be defined in other, dependent, projects). Select the appropriate module and click **OK**. If no suitable modules are available, the list is empty.

If the module that you have specified does not exist, it is created for you and the editor displays it. If the file and the module exist already, the editor highlights the correct module.

If a module skeleton is created for this node in a new or existing ESQL file, it consists of the following ESQL. The default module name is shown in this example:

```
CREATE COMPUTE MODULE <flow_name>_Compute
       CREATE FUNCTION Main() RETURNS BOOLEAN
       BEGIN
              -- CALL CopyMessageHeaders();
              -- CALL CopyEntireMessage();
              RETURN TRUE;
       END;

       CREATE PROCEDURE CopyMessageHeaders() BEGIN
              DECLARE I INTEGER 1;
              DECLARE J INTEGER CARDINALITY(InputRoot.*[]);
              WHILE I < J DO
                     SET OutputRoot.*[I] = InputRoot.*[I];
                     SET I = I + 1;
              END WHILE;
       END;

       CREATE PROCEDURE CopyEntireMessage() BEGIN
              SET OutputRoot = InputRoot;
       END;
END MODULE;
```

If you create your own ESQL module, you must create this skeleton exactly as shown except for the procedure calls and definitions (described later in this section). You can change the default name, but ensure that the name you specify matches the name of the corresponding node property `ESQL Module`. If you want the module name to include one or more spaces, enclose the name in double quotation marks in the `ESQL Module` property.

Add your own ESQL to customize this node after the BEGIN statement that follows CREATE FUNCTION, and before RETURN TRUE. You can use the two calls included in the skeleton, to procedures CopyEntireMessage and CopyMessageHeaders.

These procedures, defined following function Main, provide common functions that you might want when you manipulate messages. The calls in the skeleton are commented out; remove the comment markers if you want to use the procedure. If you do not want to use a procedure, remove both the call and the procedure definition from the module.

Setting the mode:
The `Compute Mode` property controls which components are used by default in the output message. Select the property to specify whether the `Message`, `LocalEnvironment` (previously specified as DestinationList), and `Exception List` components that are either generated in the node or contained in the incoming message are used.

This default value is used when the transformed message is routed to the Out terminal when processing in the node is completed. The default value is also used whenever a PROPAGATE statement does not specify the composition of its output message.

Those components that are not included in your selection are passed on unchanged; even if you modify those components, the updates are local to this node.

Conversely, those components that are included in the selection are not passed on and the updates that are made in the node persist.

The seven possible values that the `Compute Mode` property can take are listed in the following table.

Mode	**Description**
Message (the default)	The message is generated or passed through by the Compute node as modified in the node. The message contains the modified OutputRoot tree, the original InputLocalEnvironment tree, and the original InputExceptionList tree.
LocalEnvironment	The LocalEnvironment tree structure is generated or passed through by the Compute node as modified in the node. The message contains the original InputRoot tree, the modified OutputLocalEnvironment tree, and the original InputExceptionList tree.
LocalEnvironment And Message	The LocalEnvironment tree structure and message are generated or passed through by the Compute node as modified by the node. The message contains the modified OutputRoot tree, the modified OutputLocalEnvironment tree, and the original InputExceptionList tree.
Exception	The Exception List is generated or passed through by the Compute node as modified by the node. The message contains the original InputRoot tree, the original InputLocalEnvironment tree, and the modified OutputExceptionList tree.
Exception And Message	The Exception List and message are generated or passed through by the Compute node as modified by the node. The message contains the modified OutputRoot tree, the original InputLocalEnvironment tree, and the modified OutputExceptionList tree.

Mode	Description
Exception and LocalEnvironment	The Exception List and LocalEnvironment tree structure are generated or passed through by the Compute node as modified by the node. The message contains the original InputRoot tree, the modified OutputLocalEnvironment tree, and the modified OutputExceptionList tree.
All	The message, Exception List, and LocalEnvironment are generated or passed through by the Compute node as modified by the node. The message contains the modified OutputRoot tree, the modified OutputLocalEnvironment tree, and the modified OutputExceptionList tree.

The value of the `Compute Mode` property specifies which new message trees are propagated from the Compute node. Therefore, for those message trees that are selected, the input messages are discarded unless they are explicitly copied into the new equivalent output message tree.

If `All` is selected, the Compute node is expecting to generate all three new message trees for the Root, LocalEnvironment, and ExceptionList by populating the OutputRoot, OutputLocalEnvironment, and OutputExceptionList. The input message trees are not passed to the output unless they are copied explicitly from the Input to the Output.

Therefore, if the `Compute Mode` property is set to `All`, you must code the following ESQL to allow the input trees to be propagated to the output terminal:

```
SET OutputRoot = InputRoot;
SET OutputLocalEnvironment = InputLocalEnvironment;
SET OutputExceptionList = InputExceptionList;
```

If the ESQL was `CopyEntireMessage()`, the LocalEnvironment and ExceptionList are not copied across and are not propagated to the output terminal; they are lost at that node in the message flow.

To produce a new or changed output message, and propagate the same LocalEnvironment and ExceptionList, set the `Compute Mode` property to `Message` so that the LocalEnvironment and ExceptionList that are passed to the Compute or Mapping node, are passed on from the Compute node.

The `Compute Mode` applies only to propagation from the node. You can create all three output trees in a Compute or Mapping node and these can be manipulated and exist in the node. However, the `Compute Mode` determines whether such output trees are used on propagation from the node.

On propagation from the node, the following trees are propagated from the Compute or Mapping node for the following settings.

Compute Mode	**Trees propagated**
All	OutputRoot, OutputLocalEnvironment, OutputExceptionList
Message	OutputRoot, InputLocalEnvironment, InputExceptionList
LocalEnvironment	InputRoot, OutputLocalEnvironment, InputExceptionList

Compute Mode	Trees propagated
LocalEnvironment and Message	OutputRoot, OutputLocalEnvironment, InputExceptionList
Exception	InputRoot, InputLocalEnvironment, OutputExceptionList
Exception and Message	OutputRoot, InputLocalEnvironment, OutputExceptionList
Exception and LocalEnvironment	InputRoot, OutputLocalEnvironment, OutputExceptionList

Where an output tree is named, ESQL creates this message tree before propagation. If your ESQL does not create the tree, no tree is propagated for that correlation name, and the input tree is not used in its place because the `Compute Mode` property did not indicate this option. Therefore, dependent on `Compute Mode` property settings and your ESQL, you might delete a tree that was input to the node, because you did not transfer it to the output tree, or propagate a changed tree as you intended.

The converse is also true. If your ESQL interrogates the input trees and does not need to propagate these trees, the `Compute Mode` property value might mean that the message tree propagates when you do not intend it to. For example, you might not want to propagate the LocalEnvironment and ExceptionList from a Compute node, but because you selected `Message`, the input versions of the trees are propagated. Even if the ESQL explicitly deletes the OutputLocalEnvironment and OutputExceptionList, these changes are local to that node because the `Compute Mode` property setting causes the input trees to be propagated.

The Environment component of the message tree is not affected by the `Compute Mode` property setting. Its contents, if any, are passed on from this node in the output message.

Set this property to reflect the output message format that you require. If you select an option (or accept the default value) that does not include a particular part of the message, that part of the message is not included in any output message that is constructed.

The Compute node has both an input and output message, so that you can use ESQL to refer to fields in either message. You can also work with both InputLocalEnvironment and OutputLocalEnvironment, and InputExceptionList and OutputExceptionList, as well as the input and output message bodies.

Validating messages:
Set the validation properties to define how the message that is produced by the Compute node is to be validated. These properties do not cause the input message to be validated. It is expected that, if such validation is required, the validation has already been performed by the input node or a preceding validation node.

For more details, see “Validating messages” on page 1478 and “Validation properties” on page 4169.

Terminals and properties:
The Compute node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the input message is routed if an unhandled exception occurs during the computation.
Out	The output terminal to which the transformed message is routed when processing in the node is completed. The transformed message might also be routed to this terminal by a PROPAGATE statement.
Out1	The first alternative output terminal to which the transformed message might be routed by a PROPAGATE statement.
Out2	The second alternative output terminal to which the transformed message might be routed by a PROPAGATE statement.
Out3	The third alternative output terminal to which the transformed message might be routed by a PROPAGATE statement.
Out4	The fourth alternative output terminal to which the transformed message might be routed by a PROPAGATE statement.

For the syntax of the PROPAGATE statement, see "PROPAGATE statement" on page 5150.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file for deployment).

The Compute node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The Compute node Basic properties are described in the following table.

Property	M	C	Default	Description	**mqsiapplybaroverride command property**
Data Source	No	Yes		The ODBC data source name for the database that contains the tables to which you refer in the ESQL file that is associated with this message flow (identified in the `ESQL Module` property). You can specify only one data source for the node. If the ESQL that is associated with this node includes a PASSTHRU statement or SELECT function and a database reference, you must specify a value for the `Data Source` property.	dataSource
Transaction	Yes	No	Automatic	The transaction mode for the node. Valid options are `Automatic` and `Commit`. The property is valid only if you have selected a database table for input.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
ESQL Module	No	No	Compute	The name of the module in the ESQL file that contains the statements to run against the database and input and output messages.	
Compute Mode	Yes	No	`Message`	Choose from: • `Message` • `LocalEnvironment` • `LocalEnvironment And Message` • `Exception` • `Exception And Message` • `Exception And LocalEnvironment` • `All` For more information about setting the mode options, see "Setting the mode" on page 4344.	
Treat warnings as errors	Yes	No	Cleared	If you select the check box, database SQL warnings are treated as errors.	
Throw exception on database error	Yes	No	Selected	If you select this check box, database errors cause the broker to throw an exception.	

The Validation properties of the Compute node are described in the following table. For a full description of these properties, see "Validation properties" on page 4169.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Validate	No	Yes	`None`	This property controls whether validation takes place. Valid values are `None`, `Content and Value`, `Content`, and `Inherit`.	validateMaster
Failure Action	No	No	`Exception`	This property controls what happens if a validation failure occurs. You can set this property only if `Validate` is set to `Content` or `Content and Value`. Valid values are `User Trace`, `Local Error Log`, `Exception`, and `Exception List`.	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Correlation names" on page 1069
A correlation name is a field reference that identifies a well-defined starting point in the logical message tree and is used in field references to describe a standard part of the tree format.

Related tasks:

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

"Validating messages" on page 1478
The broker provides validation based on the message model for predefined messages.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Configuring transactionality for message flows" on page 1290
A message flow runs in a single transaction, which is started when data is received by an input node, and can be committed or rolled back when all processing has completed.

"Writing ESQL" on page 2413
How you can use ESQL to customize nodes.

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

Related reference:

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"PROPAGATE statement" on page 5150
The PROPAGATE statement propagates a message to the downstream nodes.

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

"Data sources on z/OS" on page 4014

"Validation properties" on page 4169
You can control validation by setting properties on the Validate and Parser Options tabs for the nodes that are listed in the following table.

CORBARequest node

Use the CORBARequest node to call an external CORBA application over Internet Inter-Orb Protocol (IIOP).

This topic contains the following sections:

- "Purpose" on page 4350

- "Using this node in a message flow"
- "Configuring the CORBARequest node"
- "Terminals and properties" on page 4351

Purpose:
You can use the CORBARequest to connect WebSphere Message Broker to CORBA applications. CORBA is a standard for distributing objects across networks so that operations on those objects can be called remotely. CORBA objects are described in Interface Definition Language (IDL) files. You can create a message flow that contains a CORBARequest node, which calls a CORBA server. The message flow uses the IDL file to configure which operation is called on which interface. By using a message flow that includes a CORBARequest node, you can give existing CORBA applications a new external interface; for example, a SOAP interface. The IDL file is stored in a message set project inside a folder called `CORBA IDLs`, and is used to configure the CORBARequest node in the message flow.

Using this node in a message flow:
One possible use of a CORBARequest node is to connect a SOAP-based Web service application to an existing CORBA IIOP application by using a synchronous style of message flow. You can achieve this connection by creating the following message flow:

In this example, the SOAPInput node receives a Web service request, the Mapping node transforms the data in the SOAP message to a CORBA request, and a request is made to the CORBA server. The second Mapping node transforms the response message back into a SOAP reply, which is propagated by the SOAPReply node.

The CORBARequest node is not transactional. After the node has made a request, it cannot roll back the request. The CORBA nodes use the DataObject domain.

The CORBARequest node is contained in the **CORBA** drawer of the message flow node palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Look at the following sample to see how to use this node:

- CORBA nodes

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Configuring the CORBARequest node:
When you have put an instance of the CORBARequest node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view.

All mandatory properties for which you must enter a value (properties that do not have a default value defined) are marked with an asterisk.

Terminals and properties:
The CORBARequest node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node. Data is taken from the message tree and sent to the CORBA server.
Out	The output terminal from which return parameters are propagated.
Failure	The output terminal to which a message is routed if a failure is detected.
Error	CORBA exception messages that are received in response to the sent request are sent to the Error terminal. If the Error terminal is not connected, and an error is received, no further processing occurs in the message flow. The error is logged as a warning in user trace.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined; the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The CORBARequest node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node name	The name of the node.
Short description	No	No	None	A brief description of the node.
Long description	No	No	None	Text that describes the purpose of the node in the message flow.

The CORBARequest node Basic properties are described in the following table.

Property	M	C	Default	Description	`mqsiapplybaroverride` command property
Naming service	Yes	Yes		The location of the naming service that contains the object that you are calling. For this property, you can specify either the host name of the naming service or a CORBA configurable service. • Specify the host name of the naming service from which to get the object reference in the format *host:port*, where *port* is optional. For example, localhost:2809. If you do not specify a port number, the default value is 2809. You can obtain this value from the administrator of the CORBA application that you are calling. • Specify the name of a configurable service of type CORBA. If you specify a configurable service name, you can use the configurable service to override the `Naming service` and `Object reference name` properties. For more information about using a configurable service to specify a naming service, see "Defining where the CORBARequest node gets the object reference" on page 734.	namingService

Property	M	C	Default	Description	mqsiapplybaroverride command property
Object reference name	Yes	Yes		The name of the reference to the object in the naming service. You can obtain this value from the CORBA server that you are calling. For more information about how to specify the object reference name, see "CORBA naming service" on page 2154. You can also use a configurable service to specify an object reference name. For more information, see "Defining where the CORBARequest node gets the object reference" on page 734.	referenceName
IDL file	Yes	No		The IDL file in a message set project that is used to configure the CORBARequest node. Click **Browse** to select an IDL file in a referenced message set. If you have dragged an imported IDL file onto the canvas or onto the CORBARequest node, the `IDL file` property is set to the name of the IDL file. When you specify an IDL file, the `Interface name` property is populated with a list of available interfaces.	
Interface name	Yes	No		The interface from the IDL file that the node calls. Either type a valid interface name, or click **Select interface** and select an interface from the list. If the list contains a large number of interfaces, you can filter the results. The values that are listed for this property are the interfaces in the selected IDL file that have one or more operations. This list is populated only when an IDL file is specified. If you have dragged an imported IDL file onto the canvas or onto the CORBARequest node, the `Interface name` property is set according to the IDL file. If the interface name is contained in a module in the IDL file, the interface name is qualified with the name of the module. If the module is nested in another module, the interface name is qualified with all module names, starting from the root module; for example: *ModuleNameA.ModuleNameB.InterfaceName* When you specify an interface name, the `Operation name` property is populated with a list of available operations.	
Operation name	Yes	No		The operation to call from the interface. The values that are listed for this property are the supported operations that are available in the selected interface. This list is populated only when an interface name is specified. If you have dragged an imported IDL file onto the canvas or onto the CORBARequest node, the `Operation name` property is set according to the IDL file. You can override this property in the local environment by specifying a value in the following location: `$LocalEnvironment/Destination/CORBA/Request/OperationName`	

The CORBARequest node Response Message Parsing properties are described in the following table.

Property	M	C	Default	Description
Message domain	No	No	`DataObject`	The domain that is used to parse the response message. By default, the message that is propagated from the CORBARequest node is in the DataObject domain. You cannot specify a different domain.
Message set	No	No	Set automatically	The name of the message set in which the incoming message is defined. The node detects the message set automatically.
Message type	No	No		The name of the response message. The node detects the message type automatically. You cannot set this property.
Message format	No	No		The name of the physical format of the response message. You cannot set this property.

The CORBARequest node Request properties specify how to make a request, and are described in the following table.

Property	M	C	Default	Description
Data location	Yes	No	$Body	The location in the incoming message tree from which data is retrieved to form the request that is sent by the CORBARequest node.

The CORBARequest node Result properties specify where to store the reply, and are described in the following table.

Property	M	C	Default	Description
Output data location	No	No	$OutputRoot	The message tree location to which the CORBARequest node sends output.
Copy local environment	No	No	Selected	This property controls how the local environment is copied to the output message. If you select this check box, a new copy of the local environment is created in the tree (at each node in the message flow), and it is populated with the contents of the local environment from the preceding node. Therefore, if a node changes the local environment, the previous nodes in the flow do not see those changes because they have their own copies. This behavior might be an issue if you are using a FlowOrder node, or if you use the propagate command on a Compute node. If you clear the check box, each node does not generate its own copy of the local environment, but it uses the local environment that is passed to it by the previous node. Therefore, if a node changes the local environment, those changes are seen by the upstream nodes.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see “Configuring monitoring event sources using monitoring properties” on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Common Object Request Broker Architecture (CORBA)" on page 2145
The Common Object Request Broker Architecture (CORBA) is a standard defined by the Object Management Group (OMG) that enables software components written in multiple computer languages and running on multiple computers to work together.

"CORBA nodes" on page 2147
Use CORBA nodes to connect WebSphere Message Broker with CORBA Internet Inter-Orb Protocol (IIOP) applications.

"IDL data types" on page 2150
When you use the DataObject domain with CORBA, you need to know how XML schema and ESQL types correspond to the types in the IDL file.

"CORBA naming service" on page 2154
A CORBA naming service holds CORBA object references.

"CORBA support" on page 2149
The CORBA nodes in WebSphere Message Broker support a set of types and operations in imported IDL files.

"DataObject parser and domain" on page 1114
Use the DataObject domain to parse and write messages for WebSphere Adapters and CORBA applications.

Related tasks:

"Importing an IDL file" on page 2952
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to create a message definition from an IDL file.

"Connecting to an external CORBA application" on page 2159
Connecting to an external CORBA application involves importing an IDL file, creating a message flow, building a message, and processing the response from the CORBARequest node.

"Developing a message flow with a CORBARequest node" on page 2161
To connect to an external CORBA application, create a message flow that contains a CORBARequest node.

"Building a message for the CORBARequest node" on page 2164
You can use an XML message for the CORBARequest node, or you can build a message by using another message flow node.

"Processing responses from a CORBARequest node" on page 2167
Configure the CORBARequest node to define the location to which responses are sent.

"Resolving problems when you use CORBA nodes" on page 3396
Advice for dealing with common problems that can arise when you develop message flows that contain CORBA nodes.

Database node

Use the Database node to interact with a database in the specified ODBC data source.

This topic contains the following sections:
- "Purpose"
- "Using this node in a message flow" on page 4355
- "Terminals and properties" on page 4355

Purpose:
You define the nature of the interaction by coding ESQL statements that specify the data from the input message, and perhaps transform it in some way (for example, to perform a calculation), and assign the result to a database table.

You can set a property to control whether the update to the database is committed immediately, or deferred until the message flow completes, at which time the update is committed or rolled back, according to the overall completion status of the message flow.

You can use specialized forms of this node to:
- Update values within a database table (the DataUpdate node)
- Insert rows into a database table (the DataInsert node)
- Delete rows from a database table (the DataDelete node)
- Store the message, or parts of the message, in a warehouse (the Warehouse node)

The Database node is contained in the **Database** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:
Look at the following samples to see how to use this node:
- Airline Reservations
- Error Handler

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Consider a situation in which you receive an order for 20 monitors. If you have sufficient numbers of monitors in your warehouse, you want to reduce the stock level on your stock database. You can use the Database node to check that you have enough monitors available, and reduce the value of the quantity field in your database.

Terminals and properties:
When you have put an instance of the Database node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those properties that do not have a default value defined) are marked with an asterisk.

The terminals of the Database node are described in the following table. For more information about the PROPAGATE statement, including its syntax, see "PROPAGATE statement" on page 5150

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the input message is propagated if a failure is detected during the computation. If you have selected `Treat warnings as errors`, the node propagates the message to this terminal even if the processing completes successfully.
Out	The output terminal to which the transformed message is routed when processing in the node is completed. The transformed message might also be routed to this terminal by a PROPAGATE statement.
Out1	The first alternative output terminal to which the transformed message might be routed by a PROPAGATE statement.

Terminal	Description
Out2	The second alternative output terminal to which the transformed message might be routed by a PROPAGATE statement.
Out3	The third alternative output terminal to which the transformed message might be routed by a PROPAGATE statement.
Out4	The fourth alternative output terminal to which the transformed message might be routed by a PROPAGATE statement.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Description properties of the Database node are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, Database	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The Database node Basic properties are described in the following table.

Property	M	C	Default	Description	**mqsiapplybaroverride command property**
Data Source	No	Yes		The ODBC data source name of the database that contains the tables to which you refer in the ESQL that is associated with this node (identified by the `Statement` property). This name identifies the appropriate database as it is known on the system on which this message flow is to run. The broker connects to this database with user ID and password information that you have specified on the **mqsisetdbparms** command. If the ESQL that is associated with this node includes a PASSTHRU statement or SELECT function and a database reference, you must specify a value for the `Data Source` property.	dataSource

Property	M	C	Default	Description	mqsiapplybaroverride command property

Statement	No	No	Database	The name of the module in the ESQL file that contains the statements to use against the database. If you want the module name to include one or more spaces, enclose the name in double quotation marks.
				
				The ESQL file, which by default has the name <message_flow_name>.esql, contains ESQL for every node in the message flow that requires it. Each portion of code that is related to a specific node is known as a module. When you code ESQL statements that interact with tables, those tables are assumed to exist within this database. If they do not exist, a database error is generated by the broker at run time.
				
				Code ESQL statements to customize the behavior of the Database node in an ESQL file that is associated with the message flow in which you have included this instance of the Database node. If an ESQL file does not exist for this message flow, double-click the Database node, or right-click the node and click **Open ESQL** to create and open a new ESQL file in the ESQL editor view.
				
				If an ESQL file exists, click **Browse** beside the Statement property to display the Module Selection dialog box, which lists the available Database node modules that are defined in the ESQL files that are accessible by this message flow (ESQL files can be defined in other, dependent, projects). Select the appropriate module and click **OK**. If no suitable modules are available, the list is empty.
				
				If the module that you have specified does not exist, it is created for you and the editor displays it. If the file and the module exist, the editor highlights the correct module. If a module skeleton is created for this node in a new or existing ESQL file, it consists of the following ESQL. The default module name is shown in this example:
				`CREATE DATABASE MODULE <flow_name>_Database`
				`  CREATE FUNCTION Main() RETURNS BOOLEAN`
				`  BEGIN`
				`  RETURN TRUE;`
				`  END;`
				`END MODULE;`
				
				If you create your own ESQL module, create exactly this skeleton. You can update the default name, but ensure that the name that you specify matches the name of the corresponding node property Statement.
				
				To customize this node, add your own ESQL after the BEGIN statement and before RETURN TRUE. You can use all the ESQL statements including SET, WHILE, DECLARE, and IF in this module, but (unlike the Compute node) the Database node propagates, unchanged, the message that it receives at its input terminal to its output terminal. Therefore, like the Filter node, you have only one message to refer to in a Database node.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Transaction	Yes	No	`Automatic`	The transaction mode for the node. The values are: • `Automatic` (the default). The message flow, of which the Database node is a part, is committed if it is successful; that is, the actions that you define in the ESQL module are performed and the message continues through the message flow. If the message flow fails, it is rolled back. If you select `Automatic`, the ability to commit or roll back the action of the Database node on the database depends on the success or failure of the entire message flow. • `Commit`. To commit all uncommitted actions that are performed in this message flow on the database that is connected to this node, irrespective of the success or failure of the message flow as a whole, select `Commit`. The changes to the database are committed even if the message flow itself fails.	
Treat Warnings as Errors	Yes	No	Cleared	For database warning messages to be treated as errors, and for the node to propagate the output message to the Failure terminal, select `Treat Warnings as Errors`. The check box is cleared initially. When you select the check box, the node handles all positive return codes from the database as errors, and generates exceptions in the same way that it does for the negative, or more serious, errors. If you do not select the check box, the node treats warnings as normal return codes, and does not raise an exception. The most significant warning raised is `not found`, which can be handled safely as a typical return code in most circumstances.	
Throw Exception on Database Error	Yes	No	Selected	For the broker to generate an exception when a database error is detected, select `Throw Exception on Database Error`. The check box is selected initially. If you clear the check box, include ESQL to check for database errors that might be returned after each database call that you make (you can use SQLCODE and SQLSTATE to get this information). If an error has occurred, you must handle the error in the message flow to ensure the integrity of the broker and the database; the error is ignored if you do not handle it through your own processing because you have chosen not to use the default error handling by the broker. For example, you can include the ESQL THROW statement to throw an exception in this node, or you can use the Throw node to generate your own exception at a later point.	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related tasks:

"Authorizing access to user databases" on page 662
When you have created a user database, you must authorize the broker and its execution groups to access it.

"Accessing databases from message flows" on page 2112
Create and configure message flows to access user databases.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Writing ESQL" on page 2413
How you can use ESQL to customize nodes.

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

Related reference:

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"DataDelete node" on page 4382
Use the DataDelete node to interact with a database in the specified ODBC data source.

"DataInsert node" on page 4386
Use the DataInsert node to interact with a database in the specified ODBC data source.

"DataUpdate node" on page 4390
Use the DataUpdate node to interact with a database in the specified ODBC data source.

"Warehouse node" on page 4963
Use the Warehouse node to interact with a database in the specified ODBC data source.

"PROPAGATE statement" on page 5150
The PROPAGATE statement propagates a message to the downstream nodes.

"Data sources on z/OS" on page 4014

DatabaseInput node

Use the DatabaseInput node to detect events recorded in a database, and to retrieve the data affected by those events.

This topic contains the following sections:
- "Purpose"
- "Terminals and properties"

Purpose:
The DatabaseInput node is contained in the **Database** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon: 

Terminals and properties:
The DatabaseInput node terminals are described in the following table.

Terminal	Description
Failure	The output terminal to which a message is routed if an error occurs before a message is propagated to the Out terminal.
Out	If no errors occur within the input node, a message received from an external resource is always sent to the Out terminal first.
Catch	The output terminal to which the message is routed if an exception is thrown downstream and caught by this node. Exceptions are caught only if this terminal is attached.

The following tables describe the node properties that you can set on a specified tab. The column headed M indicates whether the property is *mandatory* (marked in the toolkit with an asterisk if you must enter a value when no default is defined). The column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

Description properties:

Property	M	C	Default Value	Description
Node Name	Yes	No	`Database Input`	The name of the node.
Short Description	No	No	None	A brief description of the node.
Long Description	No	No	None	Text that describes the purpose of the node in the message flow.

Basic properties:

Property	M	C	Default	Description	mqsiapplybaroverride command property
Data Source	Yes	Yes	None	The ODBC data source name of the database that contains the tables to which you refer in the ESQL file that is associated with this message flow (identified by the `ESQL Module` property). You can specify only one data source for the node.	dataSource
ESQL Module	Yes	No	None	The name of the DATABASEEVENT module in the ESQL file that contains the statements to run against the database and output messages.	

Polling properties:

Property	M	C	Default	Description	mqsiapplybaroverride command property
Polling interval (seconds)	Yes	Yes	`5`	The polling interval in seconds.	waitInterval

Retry properties:

Property	M	C	Default	Description	mqsiapplybaroverride command property
Retry mechanism	Yes	No	`Failure`	How the node handles a flow failure. Valid options are: • `Failure` • `Short retry` • `Short and long retry`	
Retry threshold	Yes	Yes	`0`	The number of times to try the flow transaction again when the `Retry mechanism` property value is `Short retry` or `Short and long retry`.	retryThreshold
Short retry interval	No	Yes	`0`	The interval, in seconds, between each retry if `Retry threshold` property is not zero.	shortRetryInterval
Long retry interval	No	Yes	`300`	The interval between retries, if `Retry mechanism` property is `Short and long retry` and the retry threshold has been exhausted.	longRetryInterval

Transactions properties:

Property	M	Read Only	C	Default	Description	mqsiapplybaroverride command property
Transaction mode	No	Yes	No	`Yes`	The transaction mode of `Yes` on this input node determines that the interactions between this node and the database are under transactional control. It also determines that the rest of the nodes in the flow are executed under sync point if those downstream nodes are configured with automatic transaction (that is, if `Transaction mode` property is `Automatic`). It is not possible to make the database interactions non-transactional.	

Instances properties:

Property	M	C	Default	Description	mqsiapplybaroverride command property
Additional instances pool	No	Yes	`Use Pool Associated with Message Flow`	The pool from which additional instances are obtained. • If you select `Use Pool Associated with Message Flow`, additional instances are obtained from the message flow pool. • If you select `Use Pool Associated with Node`, additional instances are allocated from the additional instances of the node based on the number specified in the `Additional instances` property.	componentLevel
Additional instances	No	Yes	`0`	The number of additional instances that the node can start if the `Additional instances pool` property is set to `Use Pool Associated with Node`.	additionalInstances

For a full description of the Instances properties, see “Configurable message flow properties” on page 4020.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Event-based database integration" on page 2118
Use a DatabaseInput node to respond to events in a database. For example, the broker can keep an external system synchronized with a database by sending updates to the target system whenever data is changed in the database.

"Event tables" on page 2126
An event table stores information about changes to application tables.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Configuring a DatabaseInput node" on page 2120
Create and configure message flows that respond to events in a database.

"Responding to database updates" on page 2123
Implement a message flow that responds to database updates, and presents the data to another application.

"Working with databases" on page 2109
Create and configure databases to use with your message flow applications.

"Interaction with databases using ESQL" on page 2487
Use ESQL statements and functions to read from, write to, and modify databases from your message flows.

"Configuring transactionality for message flows" on page 1290
A message flow runs in a single transaction, which is started when data is received by an input node, and can be committed or rolled back when all processing has completed.

Related reference:

"Supported databases" on page 3591
You can optionally configure databases to contain data that is accessed by your message flows. Databases from IBM and other suppliers are supported at specific versions on supported operating systems.

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"Data sources on z/OS" on page 4014

DatabaseRetrieve node

Use the DatabaseRetrieve node to ensure that information in a message is up to date.

This topic contains the following sections:

Purpose:
Use the DatabaseRetrieve node to modify a message using information from a database. For example, you can add information to a message using a key that is contained in a message; the key can be an account number.

The DatabaseRetrieve node is contained in the **Database** drawer of the message flow node palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:
Look at the following sample to see how to use this node:

- Simplified Database Routing

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Input parameter values that are acquired from message elements in the incoming message are supported for insertion into prepared statements that are used by this node. These values are acquired from name, value, and name-value elements in the incoming parsed input message. Elements are acquired initially in the form of a com.ibm.broker.plugin.MbElement object, therefore the range of supported Java object types that values can take is governed by this object's interface. When values are in the form of Java primitive types or Objects they are converted to their equivalent JDBC data type, as shown in the following table.

Java	**JDBC**
Integer	INTEGER
Long	BIGINT
Double	DOUBLE
BigDecimal	NUMERIC
Boolean	BIT
byte[]	VARBINARY or LONGVARBINARY
BitSet	VARBINARY or LONGVARBINARY
String	VARCHAR or LONGVARCHAR
MbTime	java.sql.Time
MbTimestamp	java.sql.Timestamp
MbDate	java.sql.Date

Values are used from an element only if the element is of a known type, and its

value state is valid, otherwise an exception is issued. Output column values that are acquired in the result set from SQL queries that are carried out by this node are converted first into matching Java types, then into internal message element value types, as shown in the following table.

JDBC	Java	ESQL Type
SMALLINT	Integer	INTEGER
INTEGER	Integer	INTEGER
BIGINT	Long	DECIMAL
DOUBLE	Double	FLOAT
REAL	Double	FLOAT
FLOAT	Double	FLOAT
NUMERIC	BigDecimal	DECIMAL
DECIMAL	BigDecimal	DECIMAL
BIT	Boolean	BOOLEAN
BOOLEAN	Boolean	BOOLEAN
BINARY	byte[]	BLOB
VARBINARY	byte[]	BLOB
LONGVARBINARY	byte[]	BLOB
CHAR	String	CHARACTER
VARCHAR	String	CHARACTER
LONGVARCHAR	String	CHARACTER
TINYINT	byte[1]	BLOB
TIME	java.util.Date	TIME
TIMESTAMP	java.util.Date	TIMESTAMP
DATE	java.util.Date	DATE

You can route a message to the same location, whether a query is successful against a specified database, by wiring both of the non-failure output terminals to the same output location.

If an error is found in the XPath expression of a pattern, it is reported during validation in the WebSphere Message Broker Toolkit. The reported error message might include the incorrect expression string and its associated unique dynamic or static terminal name, or the string might be marked as broker in the table.

The DatabaseRetrieve node looks up values from a database and stores them as elements in the outgoing message assembly trees. The type of information that is obtained from the database in the form of output column values, which is acquired and passed back in the result set from SQL queries, is converted first into a matching Java type, then into an internal message element value type when it is finally stored in a location in am outgoing message assembly tree. If a message element already exists in the outgoing message tree, the new value overwrites the old value. If the target element does not exist, it is created, and the value is stored.

The node needs query information that is used to form an SQL select query, which can access multiple tables in a database using multiple test conditions. Sometimes, not all the information that you want to retrieve in a result set is in a single database table. To get the column values that you want, you might need to retrieve them from two or more tables. This node supports the use of SELECT statements

that facilitate getting columns from one or more tables in a single result set. The normal join syntax that is supported is also referred to as *inner join*.

Inner join information that is collected to form a query includes a list of table qualified column values to retrieve and a list of test conditions, which form the WHERE clause of the SELECT statement. Table qualified column values can form the left hand operand in a test condition. Choose a comparison operator to apply to this operand and, optionally, specify an operand on the right to complete the test condition. The operator could be a null comparison test, in which case an operand on the right is not needed. The value of the operand on the right can be a database type (such as Integer, Boolean, or Long), another table qualified column, or a value that is acquired from an element in the incoming message, as expressed through an XPath 1.0 general expression.

The application of the expression must result in a single element, double, Boolean, or string being returned, otherwise an exception occurs. If the query returns multiple rows, the first row is chosen and the rest are ignored, unless the `Multiple rows` option is selected. In this case, all rows are processed, and values in those rows are used to update the outgoing message assembly trees.

It can be useful to combine a DatabaseRetrieve node with other message flow nodes. For example, you can use an XSLTransform node to manipulate data before or after the DatabaseRetrieve node is invoked.

The DatabaseRetrieve node has one input terminal (In) and three output terminals (Out, KeyNotFound, and Failure). If the message is modified successfully, it is routed to the Out terminal. If the message is not modified successfully or a failure is detected during processing, the message is routed to the Failure terminal. If no rows are returned in the result set following execution of a specified SQL select query, the original message is routed to the KeyNotFound terminal.

Making the JDBC provider service available to the DatabaseRetrieve node:
The DatabaseRetrieve node constructs its JDBC connections using connection details that are stored in the broker's registry as a configurable service. JDBCProvider configurable services are supplied for all supported databases.

Use the WebSphere Message Broker Explorer to modify or create the supplied service; see "Using the WebSphere Message Broker Explorer to work with configurable services" on page 644 for more information.

You can also use the **mqsichangeproperties** command to modify the settings of the supplied service for your chosen database, or create a new service using the **mqsicreateconfigurableservice** command. See "Setting up a JDBC provider for type 4 connections" on page 684 for further information and assistance on working with JDBCProvider services. You must set up a different JDBCProvider service for each database to which you want to connect.

Note: The `maxConnectionPoolSize` property does not apply to the JDBC connections used by the DatabaseRetrieve or DatabaseRoute nodes.

When you have defined the service, set the `Data source name` property of this node to the name of the JDBCProvider service; the attributes of the service are used to establish connections for the DatabaseRetrieve node.

You must stop and restart the broker for your changes to take effect, unless you intend to create a new execution group on the broker to which you will deploy the message flow that contains this node.

Using the Data Source Explorer view to query data sources:
Use the Data Source Explorer view to discover the names of tables in a target database, and the names of any columns in those tables. You must import database definitions for your databases into the WebSphere Message Broker Toolkit before you can view them in the Data Source Explorer view. See "Adding database definitions to the WebSphere Message Broker Toolkit" on page 2278 for more information.

1. Switch to the Broker Application Development perspective.
2. In the Data Source Explorer view, expand **Connections**. The database connections are listed.
3. Expand a database connection to list the databases, then expand the appropriate database.
4. Expand **Schemas** to list the schemas, then expand the appropriate schema.
5. Expand **Tables** to list all the tables.
6. Click a table to show its properties in the Properties view.
7. In the Properties view, click the **Columns** tab to view the column names.

Configuring the DatabaseRetrieve node:
When you have put an instance of the DatabaseRetrieve node into a message flow, you can configure it. For more information, see "Configuring a message flow node" on page 1503. The properties of the node are displayed in the Properties view.

All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

Example:
The following example adds new elements (family name and wage) to the incoming message structure. This example uses a database table called Employee.

EmployeeNumber	FamilyName	FirstName	Salary
00001	Smith	John	20000
00002	Jones	Harry	26000
00003	Doe	Jane	31000

To make a copy of the incoming message, select `Copy message`. When this property is selected, the node always creates an outgoing message assembly that is based on the incoming message assembly, and these trees in the new outgoing message assembly are modified and propagated to the node's Out terminal. This behavior enables modification of the outgoing message tree itself ($OutputRoot), in addition to the other logical trees in the message assembly: $OutputLocalEnvironment, $OutputDestinationList, $OutputExceptionList and $Environment. If the logical trees only in the message assembly are to be modified by this node, for performance reasons do not select `Copy message`. When this property is not selected, the node always works against the original incoming message assembly, and it expects that no updates are attempted against the message tree. If an XPath expression in the Data elements table tries to update this message tree through a reference to $OutputRoot, an MbReadOnlyMessageException occurs. The incoming message is:

```
<EmployeeRecord>
 <EmployeeNumber>00001</EmployeeNumber>
</EmployeeRecord>
```

Here is an example of the Query elements table.

Table name	Column name	Operator	Value Type	Value
Employee	FamilyName			
Employee	Salary			
Employee	EmployeeNumber	=	Element	$InputRoot/XMLNSC/ EmployeeRecord/ EmployeeNumber

Here is an example of the Data elements table.

Column name	Message element
Employee.FamilyName	$OutputRoot/XMLNSC/EmployeeRecord/FamilyName
Employee.Salary	$OutputRoot/XMLNSC/EmployeeRecord/Wage

The DatabaseRetrieve node connects to the Employee database table and extracts the value to compare from each incoming message. The XPath expression that is used to navigate to the message body is `$InputBody/EmployeeRecord/ EmployeeNumber`. The SQL query is:

```
SELECT Employee.FamilyName, Employee.Salary
FROM Employee
WHERE EmployeeNumber=?
ORDER BY Employee.FamilyName ASC, Employee.Salary ASC
```

where *?* is the value that is retrieved from the incoming message, which is located through the `Value` property in the third row of the Query elements table, which has a `Value Type` of `Element`.

- If the value at this location is *00001*, information for John Smith is retrieved. The first data element row says get the value of the FamilyName column that is returned from the query, and insert it into a new element named "FamilyName" under EmployeeRecord. The second data element row says get the value of the Salary column that is returned from the query, and insert it into a new element named "Wage" under EmployeeRecord. The resulting outgoing message is:

  ```
  <EmployeeRecord>
   <EmployeeNumber>00001</EmployeeNumber>
   <FamilyName>Smith</FamilyName>
   <Wage>20000</Wage>
  </EmployeeRecord>
  ```

- If the value at this location is *00002*, information for Harry Jones is retrieved. The resulting outgoing message is:

  ```
  <EmployeeRecord>
   <EmployeeNumber>00002</EmployeeNumber>
   <FamilyName>Jones</FamilyName>
   <Wage>26000</Wage>
  </EmployeeRecord>
  ```

If you select the `Multiple rows` property, and details of both of the employees are returned from a query in the form of two rows in the result set, the resulting outgoing message is:

```
<EmployeeRecord>
 <EmployeeNumber>00001</EmployeeNumber>
 <FamilyName>Smith</FamilyName>
 <Wage>20000</Wage>
 <EmployeeNumber>00002</EmployeeNumber>
 <FamilyName>Jones</FamilyName>
 <Wage>26000</Wage>
</EmployeeRecord>
```

Validating messages:
Set the Validation properties to define how the message that is produced by the DatabaseRetrieve node is to be validated. These properties do not cause the input message to be validated. It is expected that, if such validation is required, the validation has already been performed by the input node or a preceding validation node.

For more details, see "Validating messages" on page 1478 and "Validation properties" on page 4169.

Terminals and properties:
The DatabaseRetrieve node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Out	The output terminal to which the outgoing message is routed when it has been modified successfully.
KeyNotFound	The output terminal to which the original message is routed, unchanged, when the result set is empty.
Failure	The output terminal to which the message is routed if a failure is detected during processing.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The DatabaseRetrieve node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	DatabaseRetrieve	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The DatabaseRetrieve node Basic properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Data source name	Yes	Yes	DB2	The alias that is used to locate JDBCProvider service definition that is stored in the broker registry. The alias is used to locate and build the JDBC connection URL that is used to connect to a DBMS. The connection URL is driver specific, but it includes the database name to which to connect. If connection to the database is by a login account and password, the node also uses this property as a lookup key, through which these values can be acquired from an expected matching broker registry DSN entry. If the DBMS is password protected, define the **-n** parameter on the **mqsisetdbparms** command for the JDBC unique security key before you deploy the message flow that contains this DatabaseRetrieve node.	dataSource
Copy message	No	Yes	Cleared	This property indicates if a copy of the original incoming message is required because the message tree is to be updated, possibly in addition to logical trees in the message assembly. By default, this check box is cleared. For performance reasons, select this property only if the input message will be augmented.	
Multiple rows	No	Yes	Cleared	This property indicates if all rows are processed when a query returns multiple rows. If you select `Multiple rows`, all rows are processed, and values in those rows are used to update the outgoing message assembly trees. If you do not select this property, the first row is chosen and the rest are ignored.	
Query elements	Yes	No		A table of query elements that are used to compose a single SQL select statement. The table consists of five columns and one or more rows. The columns are `Table name`, `Column name`, `Operator`, `Value Type`, and `Value`. These five properties describe a query element, indicating a table qualified column value to be retrieved from a database. In this case, the element forms part of the SELECT and ORDER BY clauses in the generated query. Otherwise, the query element acts as a test condition that forms a predicate in the WHERE clause in the generated query.	
Table name	Yes	No		The name of a database table that forms part of the SQL select statement, including the schema name; for example, *myschema.mytable*.	
Column name	Yes	No		The name of the column in the database table to be retrieved in the results set, as qualified by the value of the `Table name` property. This SELECT clause can refer to this name as a column value to return from a query or to be referenced in a test condition in the WHERE clause.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Operator	Yes	No		A comparison operator to apply to an operand on the left (the table column that is specified in the row's first two columns) and optionally a value to apply to an operand on the right. If you specify an Ascending 'ASC' or Descending 'DESC' operator value for this property, this row signifies the declaration of a table qualified column that forms part of the SELECT and ORDER BY clauses in the generated query and optionally can be referenced in future rows as a value to an operand on the right.	
Value Type	Yes	No		A value that is either set to `None`, or that indicates the type of value that is expressed in the last column of this row. If this property is not set to `None`, it refers to a row that describes a test condition in the WHERE clause of the SQL select statement.	
Value	Yes	No		A value that is either set to `None`, or that specifies one of a specified set of property types as expressed by the `Value Type` property. For example, if the `Value Type` property is set to `Element`, the `Value` property collects an XPath 1.0 general expression. The value that is returned from the expression when it is applied to the node's incoming message is used as value of the operand on the right to be compared through this predicate. The compared value of the operand on the right must match the type that is retrieved for the table column that is compared against as the operand on the left. Complex expressions are possible, where zero or more values can be acquired from the incoming message, and manipulated to formed a single value for comparison. For example, the sum of multiple field values in the incoming message can be calculated by a general expression that is presented for a value type of `Element`.	

The DatabaseRetrieve node Data elements table properties are described in the following table.

Property	M	C	Default	Description
Data elements	Yes	No		A list of data elements. A data element is described by the `Column name` and `Message element` properties.
Column name	Yes	No		The name of the database column from which to obtain the element value. The list of names is updated dynamically based on the column entries that are entered in the Query elements table.
Message element	Yes	No		An XPath 1.0 read-write path expression that describes the path location of a message element. The message element is where the database value is stored. The XPath expression must evaluate to a single element in the message.

The DatabaseRetrieve node Validation properties are described in the following table.

For a full description of these properties, see “Validation properties” on page 4169.

Property	M	C	Default	Description
Validate	No	Yes	`None`	This property controls whether validation takes place. Valid values are `None`, `Content and Value`, `Content`, and `Inherit`.
Failure action	No	No	`Exception`	This property controls what happens if a validation failure occurs. You can set this property only if `Validate` is set to `Content` or `Content and Value`. Valid values are `User Trace`, `Local Error Log`, `Exception`, and `Exception List`.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Using nodes for decision making" on page 2209
You can use several built-in nodes in different ways to control the path that a message takes through the message flow.

"Validating messages" on page 1478
The broker provides validation based on the message model for predefined messages.

Related reference:

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"Validation properties" on page 4169
You can control validation by setting properties on the Validate and Parser Options tabs for the nodes that are listed in the following table.

DatabaseRoute node

Use the DatabaseRoute node to route messages using information from a database in conjunction with XPath expressions.

This topic contains the following sections:

Purpose:
The DatabaseRoute node uses a collection of named column values from a selected database row and synchronously applies one or more XPath expressions to these acquired values to make routing decisions.

For more information about XPath 1.0 query syntax, see W3C XPath 1.0 Specification.

The DatabaseRoute node is contained in the **Database** drawer of the message flow node palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:
Look at the following sample to see how to use this node:

- Simplified Database Routing

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Input parameter values that are acquired from message elements in the incoming message are supported for insertion into prepared statements that are used by this node. These values are acquired from name, value, and name-value elements in the incoming parsed input message. Elements are acquired initially in the form of a com.ibm.broker.plugin.MbElement object, therefore the range of supported Java

object types that values can take is governed by this object's interface. When values are in the form of Java primitive types or Objects they are converted into their equivalent JDBC data type, as shown in the following table.

Java	JDBC
Integer	INTEGER
Long	BIGINT
Double	DOUBLE
BigDecimal	NUMERIC
Boolean	BIT
byte[]	VARBINARY or LONGVARBINARY
BitSet	VARBINARY or LONGVARBINARY
String	VARCHAR or LONGVARCHAR
MbTime	java.sql.Time
MbTimestamp	java.sql.Timestamp
MbDate	java.sql.Date

Values are used from an element only if the element is of a known type, and its value state is valid, otherwise an exception is issued. Output column values that are acquired in the result set from SQL queries that are carried out by this node are converted first into matching Java types, then into internal message element value types, as shown in the following table.

JDBC	Java	ESQL Type
SMALLINT	Integer	INTEGER
INTEGER	Integer	INTEGER
BIGINT	Long	DECIMAL
DOUBLE	Double	FLOAT
REAL	Double	FLOAT
FLOAT	Double	FLOAT
NUMERIC	BigDecimal	DECIMAL
DECIMAL	BigDecimal	DECIMAL
BIT	Boolean	BOOLEAN
BOOLEAN	Boolean	BOOLEAN
BINARY	byte[]	BLOB
VARBINARY	byte[]	BLOB
LONGVARBINARY	byte[]	BLOB
CHAR	String	CHARACTER
VARCHAR	String	CHARACTER
LONGVARCHAR	String	CHARACTER
TINYINT	byte[1]	BLOB
TIME	java.util.Date	TIME
TIMESTAMP	java.util.Date	TIMESTAMP
DATE	java.util.Date	DATE

You can route a message to the same location, whether a query is successful

against a specified database, by wiring both of the non-failure output terminals to the same output location.

If an error is found in the XPath expression of a pattern, it is reported during validation in the WebSphere Message Broker Toolkit. The reported error message might include the incorrect expression string and its associated unique dynamic or static terminal name, or the string might be marked as broken within the table.

The node needs query information that is used to form an SQL select query, which can access multiple tables in a database using multiple test conditions. Sometimes, not all the information that you want to retrieve in a result set is in a single database table. To get the column values that you want, you might need to retrieve them from two or more tables. This node supports the use of SELECT statements that facilitate getting columns from one or more tables in a single result set. The typical join syntax that is supported is also referred to as *inner join*.

Inner join information that is collected to form a query includes a list of table qualified column values to retrieve and a list of test conditions, which form the WHERE clause of the SELECT statement. Table qualified column values can form the left operand in a test condition. Choose a comparison operator to apply to this operand and, optionally, specify a right operand to complete the test condition. The operator can be a null comparison test, in which no right operand is needed. The value of the right operand can be a database type (such as Integer, Boolean, or Long), another table qualified column, or a value that is acquired from an element in the incoming message, as expressed through an XPath 1.0 general expression.

When you deploy a DatabaseRoute node in a message flow, you can select a value that is associated with the `Data Source Name` property. The list of values contains references to existing IBM predefined JDBC provider entries that are defined when a broker is first created. These entries are incomplete, therefore you must modify them to access the data source definition with which you want to work. If an existing default IBM predefined JDBC provider is already referenced and in use by another JDBC database node that requires different settings, use the WebSphere Message Broker Explorer; see “Using the WebSphere Message Broker Explorer to work with configurable services” on page 644 for more information, or the **mqsicreateconfigurableservice** command to specify a new JDBC provider entry.

Use the WebSphere Message Broker Explorer, or the **mqsideleteconfigurableservice** command to delete any unwanted JDBC provider entries.

You can delete only custom-named configurable services; you cannot delete IBM-defined configurable services.

The DatabaseRoute node has one input terminal and a minimum of four output terminals: Match, keyNotFound, Default, and Failure. The keyNotFound, Default, and Failure output terminals are static, therefore they are always present on the node. The dynamic Match terminal is created automatically each time a new DatabaseRoute node is selected and used in the Message Flow editor. This behavior means that you do not always need to create this node's first dynamic output terminal, which is the minimum number of terminals needed for this node to operate. You can rename this dynamic terminal if "Match" is not an appropriate name.

A message is copied to the Default terminal if none of the filter expressions are true. If an exception occurs during filtering, the message is propagated to the

Failure terminal. If the database query that is applied to the node's data source produces an empty result set (that is, no database rows are matched), a message is copied to the keyNotFound terminal. The DatabaseRoute node can define one or more dynamic output terminals. For all terminals, the associated filter expression is applied to the input message and, if the result is true, a copy of the message is routed to the specified terminal.

Each filter expression in the Filter table can be applied to:

- The input message
- The collection of named column values that are selected from a matched database row
- Both the input message and the returned column values
- Neither

because expressions can be any valid general XPath 1.0 expression.

As with the Route node, expressions are applied in the order that they are given in the filter table. If the result is true, a copy of the message is routed to its associated dynamic output terminal. If you set the `Distribution Mode` property to First, the application of all filter expressions might not occur.

The filter expression can fail if you compare a returned column value to a string literal. How a column entry is stored (for example, a fixed-length character field) determines what is returned for a specified column from the database. White space padding occurs for fixed-length character fields that are retrieved from a database, where the value that is stored is less than the specified column character storage length. In this case, padding occurs to the right of the character string that is returned, forming part of the string. You should remember this when comparing such a column value to a string literal, because an equality comparison expression might fail if the literal does not contain the exact same string, including padding characters.

For example, in a table called Employee, a database column called LastName that is defined as `char(10)` with the value 'Smith', is returned as 'Smith ', therefore the filter expression must be:

```
$Employee_LastName = 'Smith     '
```

which resolves to true. The expression:

```
$Employee_LastName = 'Smith'
```

resolves to false.

Alternatively, the XPath expression can use the following function:

```
Function: string normalize-space(string?)
```

The normalize-space function returns the argument string with white space normalized by stripping leading and trailing white space and replacing sequences of white space characters with a single space. Therefore the expression is:

```
normalize-space($Employee_LastName) = 'Smith'
```

Making the JDBC provider service available to the DatabaseRoute node:
The DatabaseRoute node constructs its JDBC connections using connection details that are stored in the broker's registry as a configurable service. JDBCProvider configurable services are supplied for all supported databases. Use the **mqsichangeproperties** command to modify the settings of the supplied service for

your chosen database, or create a new service using the **mqsicreateconfigurableservice** command. See "Setting up a JDBC provider for type 4 connections" on page 684 for further information and assistance on working with JDBCProvider services. You must set up a different JDBCProvider service for each database to which you want to connect.

Note: The `maxConnectionPoolSize` property does not apply to the JDBC connections used by the DatabaseRetrieve or DatabaseRoute nodes.

When you have defined the service, set the `Data Source Name` property of this node to the name of the JDBCProvider service; the attributes of the service are used to establish connections for the DatabaseRoute node.

You must stop and restart the broker for your changes to take effect, unless you intend to create a new execution group on the broker to which you will deploy the message flow that contains this node.

Using the Data Source Explorer view to query data sources:

Use the Data Source Explorer view to discover the names of tables within a target database, and the names of any columns in those tables. You must import database definitions for your databases into the WebSphere Message Broker Toolkit before you can view them in the Data Source Explorer view. See "Adding database definitions to the WebSphere Message Broker Toolkit" on page 2278 for more information.

1. Switch to the Broker Application Development perspective.
2. In the Data Source Explorer view, expand **Connections**. The database connections are listed.
3. Expand a database connection to list the databases, then expand the appropriate database.
4. Expand **Schemas** to list the schemas, then expand the appropriate schema.
5. Expand **Tables** to list all the tables.
6. Click a table to show its properties in the Properties view.
7. In the Properties view, click the **Columns** tab to view the column names.

Configuring the DatabaseRoute node:
When you have put an instance of the DatabaseRoute node into a message flow, you can configure it. For more information, see "Configuring a message flow node" on page 1503. The properties of the node are displayed in the Properties view.

All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

Example:
Consider the following example, which uses a database table called Employee.

EmployeeNumber	FamilyName	FirstName	Salary
00001	Smith	John	20000
00002	Jones	Harry	26000
00003	Doe	Jane	31000

The following DatabaseRoute node properties are set as specified:

Table Name	Column Name	Operator	Value Type	Value
Employee	FamilyName	ASC	None	None
Employee	Salary	ASC	None	None
Employee	EmployeeNumber	=	Element	$Body/EmployeeRecord/EmployeeNumber

The DatabaseRoute node connects to the Employee database table and extracts the value to compare from each incoming message. The XPath expression that is used to navigate to the message body is `$Body/EmployeeRecord/EmployeeNumber`. The SQL query is:

```
SELECT Employee.FamilyName, Employee.Salary
FROM Employee
WHERE EmployeeNumber=?
ORDER BY Employee.FamilyName ASC, Employee.Salary ASC
```

where *?* is the value that is retrieved from the incoming message. This value is located through the `Value` property in the third row of the query elements table, which has a value type of `Element`.

- If the value at this location is *00001*, information for John Smith is retrieved. The first XPath expression, which is associated with the out_expr1 dynamic terminal, is not met, therefore it does not meet its condition, and no message is propagated to the Out terminal. The second XPath expression is met, so that a copy of the input message is routed to the out_expr2 dynamic terminal.
- If the value at this location is *00002*, information for Harry Jones is retrieved. The first XPath expression, which is associated with the out_expr1 dynamic terminal, is met, so that a copy of the input message is routed to the out_expr1 terminal. The second XPath expression is not processed because the `Distribution Mode` property is set to `First`.

Terminals:
The DatabaseRoute node terminals are described in the following table.

Terminal	Description
In	The static input terminal that accepts a message for processing by the node.
Match	A dynamic output terminal to which the original message can be routed when processing completes successfully. You can create additional dynamic terminals; see "Dynamic terminals."
Default	The static output terminal to which the message is routed if no filter expression resolves to true.
keyNotFound	The static output terminal to which the message is copied if no database rows are matched.
Failure	The static output terminal to which the message is routed if a failure is detected during processing.

Dynamic terminals

The DatabaseRoute node can have further dynamic output terminals. Not all dynamic output terminals that are created on a DatabaseRoute node need to be mapped to an expression in the filter table. For unmapped dynamic output terminals, messages are never propagated to them. Several expressions can map to the same single dynamic output terminal. For more information about using dynamic terminals, see "Using dynamic terminals" on page 1518.

Properties:
The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The DatabaseRoute node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, DatabaseRoute	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The DatabaseRoute node Basic properties are described in the following table.

Property	M	C	Default	Description	**mqsiapplybaroverride command property**
Data Source Name	Yes	Yes	`DB2`	The alias that is used to locate JDBCProvider service definition that is stored in the broker registry. The alias is used to locate and build the JDBC connection URL for connection to a DBMS. The connection URL is driver specific, but it includes the database name to which to connect. If connection to the database is by a login account and password, the node also uses this property as a lookup key, through which these values can be acquired from an expected matching broker registry DSN entry. If the DBMS is password protected, define the **-n** parameter on the **mqsisetdbparms** command for the JDBC unique security key before you deploy the message flow that contains this DatabaseRoute node.	dataSource
Query Elements	Yes	No		A table of query elements that are used to compose a single SQL select statement. The table consists of five columns and one or more rows. The columns are `Table Name`, `Column Name`, `Operator`, `Value Type`, and `Value`. These five properties describe a query element, indicating a table qualified column value to be retrieved from a database. In this case, the element forms part of the SELECT and ORDER BY clauses in the generated query. Otherwise, the query element acts as a test condition that forms a predicate within the WHERE clause in the generated query.	
Table Name	Yes	No		The name of a database table that forms part of the SQL select statement, including the schema name; for example, *myschema.mytable*.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Column Name	Yes	No		The name of the column in the database table to be retrieved in the results set, as qualified by the value of the `Table Name` property. This SELECT clause can refer to this name as a column value to return from a query, or to be referenced in a test condition in the WHERE clause.	
Operator	Yes	No		A comparison operator to apply to a left operand (the table column that is specified in the row's first two columns) and optionally a right operand value. If you specify an Ascending 'ASC' or Descending 'DESC' operator value for this property, this row signifies the declaration of a table qualified column that forms part of the SELECT and ORDER BY clauses in the generated query and optionally can be referenced in future rows as a right operand value.	
Value Type	Yes	No		A value that is either set to `None`, or that indicates the type of value that is expressed in the last column of this row. If this property is not set to `None`, it refers to a row that describes a test condition in the WHERE clause of the SQL select statement.	
Value	Yes	No		A value that is either set to `None`, or that specifies one of a specified set of property types as expressed by the `Value Type` property. For example, if the `Value Type` property is set to `Element`, the `Value` property collects an XPath 1.0 general expression. The value that is returned from the expression when it is applied to the node's incoming message is used as the right operand value to be compared through this predicate. The compared value of the right operand must match the type that is retrieved for the table column that is compared against as the left operand. Complex expressions are possible, where zero or more values can be acquired from the incoming message, and manipulated to form a single value for comparison. For example, the sum of multiple field values in the incoming message can be calculated by a general expression that is presented for a value type of `Element`.	
Distribution mode	No	Yes	`All`	This property specifies the routing behavior of this node when an inbound message matches multiple expressions. If the `Distribution Mode` property is set to `First`, the message is propagated to the first matching output terminal. If the `Distribution Mode` property is set to `All`, the message is propagated to all matching output terminals. If there is no matching output terminal, the message is propagated to the Default terminal.	

The DatabaseRoute node Filter Expression Table properties are described in the following table.

Property	M	C	Default	Description
Filter table	Yes	No		A table of filters (XPath 1.0 general expressions) and associated terminal names that define any extra filtering that is performed by this node. The table consists of two columns and one or more rows. You must have at least one row in this table so that the node can perform routing logic. As with the Route node, expressions are evaluated in the order in which they are displayed in the table. To improve performance, put the XPath expressions that are satisfied most frequently at the top of the filter table.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Message flow node terminals" on page 1034
A terminal is the point at which one node in a message flow is connected to another node.

"XPath overview" on page 1507
The XML Path Language (XPath) is used to uniquely identify or address parts of an XML document. An XPath expression can be used to search through an XML document, and extract information from any part of the document, such as an element or attribute (referred to as a *node* in XML) in it. XPath can be used alone or in conjunction with XSLT.

Related tasks:

"Using dynamic terminals" on page 1518
You can add, rename, and remove dynamic terminals on a node in the Message Flow editor.

"Using XPath" on page 1506
XPath provides an alternative method to ESQL for entering expressions in the property fields of specific built-in nodes.

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Using nodes for decision making" on page 2209
You can use several built-in nodes in different ways to control the path that a message takes through the message flow.

"Setting up a JDBC provider for type 4 connections" on page 684
Use the **mqsicreateconfigurableservice** or the **mqsichangeproperties** command to configure a JDBC provider service.

"Securing a JDBC type 4 connection" on page 689
Set up security for the JDBC connection if required by the database provider.

Related reference:

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsideleteconfigurableservice** command" on page 3866
Use the **mqsideleteconfigurableservice** command to delete a configurable service, such as a JMS provider, JDBC provider, or FTP server, that you have created by using the **mqsicreateconfigurableservice** command.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"Route node" on page 4669
Use the Route node to direct messages that meet certain criteria down different paths of a message flow.

W3C XPath 1.0 Specification

DataDelete node

Use the DataDelete node to interact with a database in the specified ODBC data source.

This topic contains the following sections:
- "Purpose"
- "Using this node in a message flow"
- "Terminals and properties" on page 4383

Purpose:
The DataDelete node is a specialized form of the Database node, and the interaction is restricted to deleting one or more rows from a table within the database. You specify what is deleted by defining mapping statements that use the data from the input message to identify the action required.

You can set a property to control whether the update to the database is committed immediately, or deferred until the message flow completes, at which time the update is committed or rolled back, according to the overall completion status of the message flow.

The DataDelete node is contained in the **Database** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:
Consider a situation in which you are running a limited promotion. The goods are available only for the period of the promotion, and each customer can order only

one item. When stocks of the sale goods run out, you want to remove their details from the stock database. When a message containing an order for the last item comes in, the DataDelete node is triggered to remove all the details about that item from the database.

Terminals and properties:
When you have put an instance of the DataDelete node into a message flow, you can configure it. For more information, see "Configuring a message flow node" on page 1503. The properties of the node are displayed in the Properties view. (If you double-click the DataDelete node, you open the New Message Map dialog box.) All mandatory properties for which you must enter a value (those properties that do not have a default value defined) are marked with an asterisk.

The terminals of the DataDelete node are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the input message is propagated if a failure is detected during the computation. If you have selected `Treat warnings as errors`, the node propagates the message to this terminal even if the processing completes successfully.
Out	The output terminal that outputs the message following the execution of the database statement.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The DataDelete node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	DataDelete	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The DataDelete node Basic properties are described in the following table.

Property	M	C	Default	Description	**`mqsiapplybaroverride` command property**
Data source	No	Yes		The ODBC data source name of the database that contains the tables to which you refer in the mappings that are associated with this node (identified by the `Statement` property). This name identifies the appropriate database as it is known on the system on which this message flow is to run. The broker connects to this database with user ID and password information that you have specified on the **`mqsisetdbparms`** command.	dataSource

Property	M	C	Default	Description	mqsiapplybaroverride command property
Statement	Yes	No	DataDelete	The name of the mapping routine that contains the statements that are to be executed against the database or the message tree. The routine is unique to this type of node. By default, the name that is assigned to the mapping routine is identical to the name of the mappings file in which the routine is defined. The default name for the file is the name of the message flow concatenated with the name of the node when you include it in the message flow (for example, `MFlow1_DataDelete.msgmap` for the first DataDelete node in message flow MFlow1). You cannot specify a value that includes spaces. If you click **Browse** next to this entry field, a dialog box is displayed that lists all the available mapping routines that can be accessed by this node. Select the routine that you want and click **OK**; the routine name is set in `Statement`. To work with the mapping routine that is associated with this node, double-click the node, or right-click the node and click **Open Mappings**. If the mapping routine does not exist, it is created for you with the default name in the default file. If the file exists, you can also open file *`flow_name_node_name`*`.msgmap` in the Broker Development view. A mapping routine is specific to the type of node with which it is associated; you cannot use a mapping routine that you have developed for a DataDelete node with a different node that uses mappings (for example, a DataInsert node). If you create a mapping routine, you cannot call it from another mapping routine, although you can call it from an ESQL routine. For more information about working with mapping files, and defining their content, see "Using message mappings" on page 2228.	
Transaction	Yes	No	Automatic	The transaction mode for the node. The values are: • Automatic (the default). The message flow, of which the DataDelete node is a part, is committed if it is successful; that is, the actions that you define in the mappings are performed and the message continues through the message flow. If the message flow fails, it is rolled back. Therefore, if you select Automatic, the ability to commit or roll back the action of the DataDelete node on the database depends on the success or failure of the entire message flow. • Commit. To commit any uncommitted actions performed in this message flow on the database connected to this node, irrespective of the success or failure of the message flow as a whole, select Commit. The changes to the database are committed even if the message flow itself fails.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Treat warnings as errors	Yes	No	Cleared	For database warning messages to be treated as errors, and the node to propagate the output message to the failure terminal, select `Treat warnings as errors`. The check box is cleared by default. When you select the check box, the node handles all positive return codes from the database as errors, and generates exceptions in the same way as it does for the negative, or more serious, errors. If you do not select the check box, the node treats warnings as typical return codes, and does not raise an exception. The most significant warning raised is `not found`, which can be handled as a typical return code safely in most circumstances.	
Throw exception on database error	Yes	No	Selected	For the broker to generate an exception when a database error is detected, select `Throw exception on database error`. The check box is selected by default. If you clear the check box, you must handle the error in the message flow to ensure the integrity of the broker and the database: the error is ignored if you do not handle it through your own processing because you have chosen not to run the default error handling by the broker. For example, you can connect the Failure terminal to an error processing subroutine.	

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Client application programming interfaces" on page 1038
You can configure the nodes in your message flows to customize the behavior of those nodes by using one or more of the supported programming interfaces.

Related tasks:

"Authorizing access to user databases" on page 662
When you have created a user database, you must authorize the broker and its execution groups to access it.

"Accessing databases from message flows" on page 2112
Create and configure message flows to access user databases.

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

"Configuring transactionality for message flows" on page 1290
A message flow runs in a single transaction, which is started when data is received by an input node, and can be committed or rolled back when all processing has completed.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Using message mappings" on page 2228
Message mappings define the blueprint for creating a message.

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

Related reference:

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"DataInsert node"
Use the DataInsert node to interact with a database in the specified ODBC data source.

DataInsert node

Use the DataInsert node to interact with a database in the specified ODBC data source.

This topic contains the following sections:

Purpose:

The DataInsert node is a specialized form of the Database node, and the interaction is restricted to inserting one or more rows into a table within the database. You specify what is inserted by defining mapping statements that use the data from the input message to define the action required.

You can set a property to control whether the update to the database is committed immediately, or deferred until the message flow completes, at which time the update is committed, or rolled back according to the overall completion status of the message flow.

The DataInsert node is contained in the **Database** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:

Consider a situation in which your company has developed a new product. The details about the product have been sent from your engineering department. Your message flow must extract details from the message and add them as a new row in your stock database.

Terminals and properties:
When you have put an instance of the DataInsert node into a message flow, you can configure it. For more information, see "Configuring a message flow node" on page 1503. The properties of the node are displayed in the Properties view. (If you double-click the DataInsert node, you open the New Message Map dialog box.) All mandatory properties for which you must enter a value (those properties that do not have a default value defined) are marked with an asterisk.

The terminals of the DataInsert node are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the input message is propagated if a failure is detected during the computation. If you have selected `Treat warnings as errors`, the node propagates the message to this terminal even if the processing completes successfully.
Out	The output terminal that outputs the message following the execution of the database statement.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The DataInsert node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	DataInsert	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The DataInsert node Basic properties are described in the following table.

Property	M	C	Default	Description	**`mqsiapplybaroverride` command property**
Data source	No	Yes		The ODBC data source name of the database that contains the tables to which you refer in the mappings that are associated with this node (identified by the `Statement` property). This name identifies the appropriate database as it is known on the system on which this message flow is to run. The broker connects to this database with user ID and password information that you have specified on the **`mqsisetdbparms`** command.	dataSource

Property	M	C	Default	Description	mqsiapplybaroverride command property
Statement	Yes	No	DataInsert	The name of the mapping routine that contains the statements that are to be executed against the database or the message tree. The routine is unique to this type of node. By default, the name that is assigned to the mapping routine is identical to the name of the mappings file in which the routine is defined. The default name for the file is the name of the message flow concatenated with the name of the node when you include it in the message flow (for example, `MFlow1_DataInsert.msgmap` for the first DataInsert node in message flow MFlow1). You cannot specify a value that includes spaces. If you click **Browse** next to this entry field, a dialog box is displayed that lists all the available mapping routines that can be accessed by this node. Select the routine that you want and click **OK**; the routine name is set in `Statement`. To work with the mapping routine that is associated with this node, double-click the node, or right-click the node and select **Open Mappings**. If the mapping routine does not exist, it is created for you with the default name in the default file. If the file exists, you can also open file *`flow_name_node_name`*`.msgmap` in the Broker Development view. A mapping routine is specific to the type of node with which it is associated; you cannot use a mapping routine that you have developed for a DataInsert node with a different node that uses mappings (for example, a DataDelete node). If you create a mapping routine, you cannot call it from another mapping routine, although you can call it from an ESQL routine. For more information about working with mapping files, and defining their content, see "Using message mappings" on page 2228.	
Transaction	Yes	No	Automatic	The transaction mode for the node. The values are: • Automatic (the default). The message flow, of which the DataInsert node is a part, is committed if it is successful. That is, the actions that you define in the mappings are taken and the message continues through the message flow. If the message flow fails, it is rolled back. Therefore if you select Automatic, the ability to commit or roll back the action of the DataInsert node on the database depends on the success or failure of the entire message flow. • Commit. To commit all uncommitted actions that are taken in this message flow on the database that is connected to this node, irrespective of the success or failure of the message flow as a whole, select Commit. The changes to the database are committed even if the message flow itself fails.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Treat warnings as errors	Yes	No	Cleared	For database warning messages to be treated as errors, and the node to propagate the output message to the failure terminal, select `Treat warnings as errors`. The check box is cleared by default. When you select the check box, the node handles all positive return codes from the database as errors and generates exceptions in the same way as it does for the negative, or more serious, errors. If you do not select the check box, the node treats warnings as typical return codes, and does not raise an exception. The most significant warning raised is `not found`, which can be handled as a typical return code safely in most circumstances.	
Throw exception on database error	Yes	No	Selected	For the broker to generate an exception when a database error is detected, select `Throw exception on database error`. The check box is selected by default. If you clear the check box, you must handle the error in the message flow to ensure the integrity of the broker and the database. The error is ignored if you do not handle it through your own processing because you have chosen not to run the default error handling by the broker. For example, you can connect the failure terminal to an error processing subroutine.	

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Client application programming interfaces" on page 1038
You can configure the nodes in your message flows to customize the behavior of those nodes by using one or more of the supported programming interfaces.

Related tasks:

"Authorizing access to user databases" on page 662
When you have created a user database, you must authorize the broker and its execution groups to access it.

"Accessing databases from message flows" on page 2112
Create and configure message flows to access user databases.

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

"Configuring transactionality for message flows" on page 1290
A message flow runs in a single transaction, which is started when data is received by an input node, and can be committed or rolled back when all processing has completed.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Using message mappings" on page 2228
Message mappings define the blueprint for creating a message.

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

Related reference:

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"DataDelete node" on page 4382
Use the DataDelete node to interact with a database in the specified ODBC data source.

DataUpdate node

Use the DataUpdate node to interact with a database in the specified ODBC data source.

This topic contains the following sections:
- "Purpose"
- "Using this node in a message flow"
- "Terminals and properties" on page 4391

Purpose:
The DataUpdate node is a specialized form of the Database node, and the interaction is restricted to updating one or more rows in a table within the database. You define what is updated by defining mapping statements that use the data from the input message to identify the action required.

You can set a property to control whether the update to the database is committed immediately, or deferred until the message flow completes, at which time the update is committed or rolled back according to the overall completion status of the message flow.

The DataUpdate node is contained in the **Database** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:
Consider a scenario in which you have added the details of a new product, a keyboard, to your stock database. Now you have received a message from the Goods In department that indicates that 500 keyboards have been delivered to

your premises. You can use the DataUpdate node to change the quantity of keyboards in your database from zero to 500.

Terminals and properties:
When you have put an instance of the DataUpdate node into a message flow, you can configure it. For more information, see "Configuring a message flow node" on page 1503. The properties of the node are displayed in the Properties view. (If you double-click the DataUpdate node, you open the New Message Map dialog box.) All mandatory properties for which you must enter a value (those properties that do not have a default value defined) are marked with an asterisk.

The terminals of the DataUpdate node are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the input message is propagated if a failure is detected during the computation. If you have selected `Treat warnings as errors`, the node propagates the message to this terminal even if the processing completes successfully.
Out	The output terminal that outputs the message following the execution of the database statement.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The DataUpdate node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	DataUpdate	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The DataUpdate node Basic properties are described in the following table.

Property	M	C	Default	Description	**`mqsiapplybaroverride` command property**
Data source	No	Yes		The ODBC data source name of the database that contains the tables to which you refer in the mappings that are associated with this node (identified by the `Statement` property). This name identifies the appropriate database as it is known on the system on which this message flow is to run. The broker connects to this database with user ID and password information that you have specified on the **`mqsisetdbparms`** command.	dataSource

Property	M	C	Default	Description	mqsiapplybaroverride command property
Statement	Yes	No	DataUpdate	The name of the mapping routine that contains the statements that are to be executed against the database or the message tree. The routine is unique to this type of node. By default, the name that is assigned to the mapping routine is identical to the name of the mappings file in which the routine is defined. The default name for the file is the name of the message flow concatenated with the name of the node when you include it in the message flow (for example, MFlow1_DataUpdate.msgmap for the first DataUpdate node in message flow MFlow1). You cannot specify a value that includes spaces. If you click **Browse** next to this entry field, a dialog box is displayed that lists all available mapping routines that can be accessed by this node. Select the routine that you want and click **OK**; the routine name is set in Statement. To work with the mapping routine that is associated with this node, double-click the node, or right-click the node and click **Open Mappings**. If the mapping routine does not exist, it is created for you with the default name in the default file. If the file exists, you can also open file *flow_name_node_name*.msgmap in the Broker Development view. A mapping routine is specific to the type of node with which it is associated; you cannot use a mapping routine that you have developed for a DataUpdate node with a different node that uses mappings (for example, a DataInsert node). If you create a mapping routine, you cannot call it from another mapping routine, although you can call it from an ESQL routine. For more information about working with mapping files, and defining their content, see "Using message mappings" on page 2228.	
Transaction	Yes	No	Automatic	The transaction mode for the node. The values are: • Automatic (the default). The message flow, of which the DataUpdate node is a part, is committed if it is successful. That is, the actions that you define in the mappings are performed and the message continues through the message flow. If the message flow fails, it is rolled back. Therefore, if you choose Automatic, the ability to commit or roll back the action of the DataUpdate node on the database depends on the success or failure of the entire message flow. • Commit. To commit all uncommitted actions that are performed in this message flow on the database that is connected to this node, irrespective of the success or failure of the message flow as a whole, select Commit. The changes to the database are committed even if the message flow itself fails.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Treat warnings as errors	Yes	No	Cleared	For database warning messages to be treated as errors, and the node to propagate the output message to the Failure terminal, select `Treat warnings as errors`. The check box is cleared by default. When you select the check box, the node handles all positive return codes from the database as errors, and generates exceptions in the same way as it does for the negative, or more serious, errors. If you do not select the check box, the node treats warnings as normal return codes, and does not raise an exception. The most significant warning raised is `not found`, which can be handled safely as a normal return code in most circumstances.	
Throw exception on database error	Yes	No	Selected	For the broker to generate an exception when a database error is detected, select `Throw exception on database error`. The check box is selected by default. If you clear the check box, you must handle the error in the message flow to ensure the integrity of the broker and the database: the error is ignored if you do not handle it through your own processing, because you have chosen not to run the default error handling by the broker. For example, you can connect the Failure terminal to an error processing subroutine.	

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Client application programming interfaces" on page 1038
You can configure the nodes in your message flows to customize the behavior of those nodes by using one or more of the supported programming interfaces.

Related tasks:

"Authorizing access to user databases" on page 662
When you have created a user database, you must authorize the broker and its execution groups to access it.

"Accessing databases from message flows" on page 2112
Create and configure message flows to access user databases.

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

"Configuring transactionality for message flows" on page 1290
A message flow runs in a single transaction, which is started when data is received by an input node, and can be committed or rolled back when all processing has completed.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Using message mappings" on page 2228
Message mappings define the blueprint for creating a message.

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

Related reference:

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"DataInsert node" on page 4386
Use the DataInsert node to interact with a database in the specified ODBC data source.

EmailInput node

Use the EmailInput node to retrieve an email, with or without attachments, from an email server that supports Post Office Protocol 3 (POP3) or Internet Message Access Protocol (IMAP).

This topic contains the following sections:
- "Purpose"
- "Using the EmailInput node in a message flow" on page 4395
- "Configuring the EmailInput node" on page 4396
- "Terminals and properties" on page 4396

Purpose:
The EmailInput node receives an email, with or without attachments, from an email server that supports Post Office Protocol 3 (POP3) or Internet Message Access Protocol (IMAP). The body of the email message, and any attachments, are propagated in the Multipurpose Internet Mail Extensions (MIME) domain. All other information relating to the email is stored in the `Root.EmailInputHeader` MIME logical tree. The fields in this structure are described in the following table:

Element Location	Element Data Type	Description
Root.EmailInputHeader.To	CHARACTER	A comma-separated list of email addresses.
Root.EmailInputHeader.Cc	CHARACTER	A comma-separated list of email addresses.
Root.EmailInputHeader.From	CHARACTER	A comma-separated list of email addresses.
Root.EmailInputHeader.ReplyTo	CHARACTER	A comma-separated list of email addresses.
Root.EmailInputHeader.Subject	CHARACTER	The summary of the email content or the subject of the email.
Root.EmailInputHeader.Size	INTEGER	The size of the email, including any attachments.
Root.EmailInputHeader.SentDate	CHARACTER	The sent delivery date of the email.

This structure is propagated with each message written to the Out terminal of the EmailInput node. For more information about the MIME logical tree, see "MIME parser and domain" on page 1117.

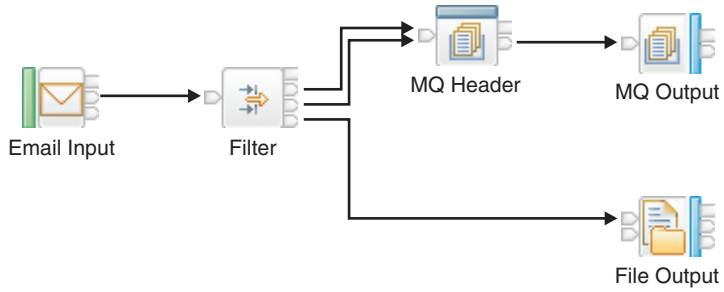
Using the EmailInput node in a message flow:
To enable function that becomes available in WebSphere Message Broker fix packs, use the **-f** parameter on the **mqsichangebroker** command. For more information, see "**mqsichangebroker** command" on page 3723.

You can configure the EmailInput node by using the node properties in the WebSphere Message Broker Toolkit. One possible use of the EmailInput node is to poll an email server for email messages at regular intervals, and to retrieve an email message when an email is available. The message flow receives the email from the email server as a MIME message.

In the following example message flow, the EmailInput node then passes the email message, which is associated with the MIME parser, onto a Filter node. The Filter node processes the email and directs it either to an MQHeader node or to a FileOutput node based on whether the email has an attachment or not.

If the email message does not contain an attachment, the Filter node sends the email to an MQHeader node for MQ Message Descriptor (MQMD) customization. The MQHeader node adds a WebSphere MQ header to the message tree and passes the message onto the MQOutput node. The MQOutput node serializes the message body, for example; the text contents of the email, and places the body of the email on a WebSphere MQ queue for further processing.

If the email message does contain an attachment, the Filter node sends the email onto a FileOutput node. The FileOutput node then saves the attachment as a file named `email_attachment.dat` to a hard disk drive that you specify and, in this example, the text content of the email is discarded. You can achieve this example by creating the following message flow:



The EmailInput node is contained in the **Email** drawer of the message flow node palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

View the following sample to see how to use the EmailInput node:

- Email

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Using configurable services for the EmailInput node

You can configure the EmailInput node to get the email server URL and security identity information from a configurable service. For details about creating, changing, reporting, and deleting the configurable services, see "Changing connection information for the EmailInput node" on page 1805.

Using a security identity to authenticate with an email server

You can use the **mqsisetdbparms** command to set a user ID and password security identity object for the EmailInput node or EmailServer configurable service to use for accessing the email server. For detailed information about how to configure email server security identity support, see "**mqsisetdbparms** command" on page 3954.

Configuring the EmailInput node:
When you have put an instance of the EmailInput node into a message flow, you can configure it. For more information, see "Configuring a message flow node" on page 1503. The properties of the node are displayed in the Properties view.

All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

To configure the EmailInput node by using the node properties in the WebSphere Message Broker Toolkit to retrieve an email, with or without attachments, see "Receiving an email" on page 1801.

Terminals and properties:
The EmailInput node terminals are described in the following table.

Terminal	Description
Failure	The output terminal to which the message is routed if an EmailInput node failure is detected when a message is propagated, or an EmailInput node fails to access the email server. Connect the Failure terminal of this node to another node in the message flow to process errors.
Out	The output terminal to which the message is routed if it has been propagated successfully. Connect the Out terminal of this node to another node in the message flow to process the message further, or send the message to an additional destination.
Catch	The output terminal to which the message is routed if an exception is thrown downstream and caught by this node. Exceptions are caught only if this terminal is attached.

The following tables describe the EmailInput node properties.

The table column headed M indicates whether the property is *mandatory*. For example, the property is marked with an asterisk meaning that you must enter a value if no default is defined.

The column headed C indicates whether the property is *configurable*. For example, you can change the value when you add the message flow to the BAR file to deploy it.

The EmailInput node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	Email Input	The name of the node.

Property	M	C	Default	Description
Short description	No	No	None	A brief description of the node.
Long description	No	No	None	Text that describes the purpose of the node in the message flow.

The EmailInput node Basic properties are described in the following table.

Property	M	C	Default	Description	**mqsiapplybaroverride command property**
Email server	Yes	Yes	None	The Email server property is defined either as a configurable service name, for example: *myEmailConfigurableServiceName*, or as a URL. A URL allows you to specify a protocol, host name, and port number, which is the minimum information you need to access the email server. The URL must be made up of the following structure: *protocol*`://`*hostname*`:`*port* Where: • *protocol* can be `pop3` or `imap`. • *hostname* is the Internet Protocol version 4 (IPv4) TCP/IP address or DNS-resolvable host name of the email host. • *port* is the port number that the email server is listening on for connections over POP3 or IMAP. You can enter an integer in the range 1- 65535. For example: `pop3://myemailserver.com:12345` or `imap://myemailserver.com:56789`. There is no default value for this property, however this property is mandatory, and therefore must be configured with a configurable service name or a URL. You can obtain the *hostname* and *port* values from the email server or email server administrator.	emailServer

The EmailInput node Polling properties are described in the following table.

Property	M	C	Default	Description	**mqsiapplybaroverride command property**
Polling interval (in seconds)	Yes	Yes	5	The interval at which the EmailInput node polls the email server for new emails.	waitInterval

The EmailInput node Security properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Security identity	No	Yes	None	The name of the security identity object that is created and configured by the **mqsisetdbparms** command, which contains the user ID and password to be used by the broker to authenticate with the email server. Use the **mqsisetdbparms** command to set the security identity user ID and password to be accessed by the broker. The default value for this property is None, which signifies that the user ID and password are not passed to the email server. For more information about email server security identity support, see "**mqsisetdbparms** command" on page 3954.	securityIdentity

The EmailInput node Retry properties are described in the following table. For more information about configuring the EmailInput node Retry properties, see "Receiving an email" on page 1801.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Retry mechanism	Yes	No	Short and Long Retry	How the EmailInput node handles a message flow failure. Valid values are `Failure`, `Short Retry`, or `Short and Long Retry`.	
Retry threshold	Yes	Yes	0	The number of times to try the message flow transaction again when the `Retry mechanism` property value is set to `Short Retry`.	retryThreshold
Short retry interval (in seconds)	No	Yes	0	The interval, in seconds, between each retry if the `Retry threshold` property value is not set to zero.	shortRetryInterval
Long retry interval (in seconds)	No	Yes	300	The interval, in seconds, between each retry, if the `Retry mechanism` property value is `Short and Long Retry` and the retry threshold has been exhausted.	longRetryInterval
Action on failing email	Yes	No	Delete Email	The action that the EmailInput node takes with the input data source after all attempts to process the contents fail.	emailFailureAction

The EmailInput node Transactions properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Transaction mode	No	Yes	No	The transaction mode on the EmailInput node determines if the rest of the nodes in the message flow are run under sync point. Valid values are `Yes` or `No`.	

The EmailInput node Instances properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Additional instances pool	No	Yes	Use Pool Associated with Message Flow	The pool from which additional instances are obtained. If you select the property value `Use Pool Associated with Message Flow`, additional instances are obtained from the message flow pool. If you select the property value `Use Pool Associated with Node`, additional instances are allocated from the additional instances of the node, based on the number specified in the `Additional instances` property.	componentLevel
Additional instances	No	Yes	0	The number of additional instances that the EmailInput node can start if the `Additional instances pool` property is set to the value `Use Pool Associated with Node`.	additionalInstances

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Configurable services" on page 1296
Configurable services are typically runtime properties. You can use them to define properties that are related to external services on which the broker relies; for example, an SMTP server or a JMS provider.

"MIME parser and domain" on page 1117
Use the MIME domain if your messages use the MIME standard for multipart messages.

"Manipulating messages in the MIME domain" on page 2612
A MIME message does not need to be received over a particular transport. For example, a message can be received over HTTP by using an HTTPInput node, or over WebSphere MQ by using an MQInput node. The MIME parser is used to process a message if the message domain is set to MIME in the input node properties, or if you are using WebSphere MQ, and the MQRFH2 header has a message domain of MIME.

"Samples" on page 98
The WebSphere Message Broker Toolkit provides samples that show the features that are available in WebSphere Message Broker, and how to use them. This topic provides links to the information about the individual samples.

Related tasks:

"Processing email messages" on page 1786
You can configure the EmailOutput node to deliver an email from a message flow to an email server that supports Simple Mail Transfer Protocol (SMTP). You can also configure the EmailInput node to retrieve an email from an email server that supports Post Office Protocol 3 (POP3) or Internet Message Access Protocol (IMAP).

"Receiving emails" on page 1799
You can configure the EmailInput node to receive an email, with or without an attachment, from an email server that supports Post Office Protocol 3 (POP3) or Internet Message Access Protocol (IMAP).

"Receiving an email" on page 1801
You can receive an email, with or without attachments, from an email server that supports Post Office Protocol 3 (POP3) or Internet Message Access Protocol (IMAP).

"Processing responses from an EmailInput node" on page 1804
The EmailInput node can return different response messages that indicate the success or failure of receiving an email, with or without attachments, from an email server that supports Post Office Protocol 3 (POP3) or Internet Message Access Protocol (IMAP).

"Changing connection information for the EmailInput node" on page 1805
You can create a configurable service that the EmailInput node or message flow refers to at run time for email server connection information, instead of defining the connection properties on the node or the message flow. The advantage being that you can change the host name and security identity values without needing to redeploy your message flow.

"Resolving problems when you use Email nodes" on page 3398
Advice for dealing with common problems that can arise when you develop message flows that contain Email nodes.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

EmailOutput node

Use the EmailOutput node to send email messages to one or more recipients.

This topic contains the following sections:
- "Purpose"
- "Configuring the EmailOutput node" on page 4401
- "Terminals and properties" on page 4404

Purpose:
The EmailOutput node delivers an email message from a message flow to an SMTP server that you specify.

You can configure the EmailOutput node by using the node properties in the WebSphere Message Broker Toolkit, or dynamically from the local environment and email output header (EmailOutputHeader) that are associated with the message (for more information, see "Producing dynamic email messages" on page 1791). You can configure the EmailOutput node to produce an e-mail with a single attachment. When you produce email messages dynamically, you can specify multiple attachments.

The EmailOutput node is contained in the **Email** drawer of the message flow node palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Look at the following sample to see how to use this node:

- Email

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Configuring the EmailOutput node:
When you have put an instance of the EmailOutput node into a message flow, you can configure it. For more information, see "Configuring a message flow node" on page 1503. The properties of the node are displayed in the Properties view.

All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The following is a description of the four levels of configuration of the EmailOutput node:

- Option 1: Configure the EmailOutput node by using the node properties in the WebSphere Message Broker Toolkit to send an email with a statically-defined subject and text to a statically-defined list of recipients. The same email is sent to the same recipients and it has no attachments. This method is useful when you want to test the EmailOutput node, or when notification alone is sufficient. For more details, see "Sending an email" on page 1789.
- Option 2: This option is the same as Option 1 but with the inclusion of an attachment. This option causes the email message to be constructed as a MIME message. The subject, text, and list of recipients remains static, but the content of the attachment is sought dynamically from the message that is passed to the EmailOutput node at run time. The location of the attachment in the message is defined statically. For more details, see "Sending an email with an attachment" on page 1790.
- Option 3: This option allows for those properties in Options 1 and 2 to be optional, and to be overridden at run time by values that are specified in the local environment, the email output header (EmailOutputHeader), or the body of the message. This option allows a dynamic email message to be produced where the SMTP server, list of recipients, subject, text, and multiple attachments are all determined at run time. This option requires previous nodes in the message flow to construct these overrides. Where a text value is not specified in the node properties for the main body of the email, the body of the message that is passed to the EmailOutput node is used. However, content that you set in the `Email message text` property overrides dynamically-generated text in the message body. For more details, see "Producing dynamic email messages" on page 1791.
- Option 4: This option passes a MIME message to the EmailOutput node. The EmailOutput node uses the MIME parser to write the MIME message to a bit stream. This message is then sent to the list of recipients in the SMTP header.

Local environment overrides are not taken into consideration when a MIME message is passed. For more details, see “Sending a MIME message” on page 1795.

Email output header

The email output header (EmailOutputHeader) is a child of root. Values that you specify in this header override equivalent properties that you set on the node. Use the SMTP output header to specify any of the email attributes, such as its recipients.

Location	Description
Root.EmailOutputHeader.To	A comma separated list of email addresses.
Root.EmailOutputHeader.Cc	A comma separated list of email addresses.
Root.EmailOutputHeader.Bcc	A comma separated list of email addresses.
Root.EmailOutputHeader.From	A comma separated list of email addresses.
Root.EmailOutputHeader.Reply-To	A comma separated list of email addresses.
Root.EmailOutputHeader.Subject	The subject of the email.

Local environment

Use the local environment to specify overrides to the SMTP server connection information and attachments.

Local environment	Description
Destination.Email.SMTPServer	The Server:Port of the SMTP server. Port is optional; if you do not specify it, the default value is 25.
Destination.Email.SecurityIdentity	The security identity for authentication with the SMTP server, which can be the name of the userid and password pair that is defined using the **mqsisetdbparms** command, or it can reference an external resource that has a securityIdentity attribute that references a userid and password that are defined using the **mqsisetdbparms** command. In both cases, the value is appended after the string “smtp::”. For example, if you use the **mqsisetdbparms** command to create a userid and password of *smtp::myUseridPassword*, the securityIdentity that is specified on the node, or indirectly in an external resource, is *myUseridPassword*.

Destination.Email.BodyContentType	Identifies that the body of the email message contains HTML rather than plain text. You can set this property to text/plain, text/html, or text/xml; text/plain is the default value.
	
	To set the content type for the body of the message, use the following notation.
	`SET OutputLocalEnvironment.Destination.Email.BodyContentType = "text/html"`
	
	To additionally set the character set (charset) in which the message body is sent, use the following notation.
	`SET OutputLocalEnvironment.Destination.Email.BodyContentType = "text/html; charset=utf-8"`
	
	This example sends a text/HTML email with a charset of UTF-8.

Destination.Email.MultiPartContentType	The type of multipart, including related, mixed, and alternative. You can set any value here.

Local environment	Description
Destination.Email.Attachment.Content	Either the attachment (BLOB/text), or an XPath or ESQL expression that references an element; for example, an element in the message tree or local environment. The value of the referenced element is taken as the content of the attachment. • If the element is a BLOB, it is an attachment. • If the element is text, check to see if it can be resolved to another element in the message tree or local environment. If it can be resolved, use that element. If it cannot be resolved, add this element as the attachment.
Destination.Email.Attachment.ContentType	The type of attachment (also known as Internet Media Type), including text/plain, text/html, and text/xml. You can set any value here.
Destination.Email.Attachment.ContentName	The name of the attachment.
Destination.Email.Attachment.ContentEncoding	The encoding of the attachment: 7bit, base64, or quoted-printable. • 7bit is the default value that is used for ASCII text. • Base64 is used for non ASCII, whether non English or binary data. This format can be difficult to read. • Quoted-printable is an alternative to Base64, and is appropriate when the majority of the data is ASCII with some non-ASCII parts. This format is more readable; it provides a more compact encoding because the ASCII parts are not encoded.

Broker properties

You can also configure the SMTP server, port number, and security identity as a broker external resource property. To do this, use an alias that is specified in the `SMTP Server and Port` property on the EmailOutput node. The security identity references a user ID and password pair that is defined on the broker using the **mqsisetdbparms** command. Use the **mqsicreateconfigurableservice** command to create an SMTP broker external resource for the alias that is specified on the node. Then use the **mqsichangeproperties** command to create an SMTPServer property with the value in the form of *server:port*. The port value is optional; if you do not specify it, the default value is 25. You can also use the **mqsichangeproperties** command to create an SMTPSecurityIdentity property with a value that is the name of a security identity that can be resolved at run time to a user ID and password for authentication with the SMTP server. For example:

```
mqsicreateconfigurableservice MY_BROKER –c SMTP –o SMTP_MyAlias
```

followed by:

```
mqsichangeproperties MY_BROKER –c SMTP –o SMTP_MyAlias –n serverName –v smtp.hursley.ibm.com:25
```

These commands override the SMTP server and port values that are specified on any nodes that also specify an alias of SMTP_MyAlias. If the local environment contains any overrides, they take preference over the broker external resource properties. See also the following example:

```
mqsichangeproperties MY_BROKER –c SMTP –o SMTP_MyAlias –n securityIdentity –v mySecurityIdentity
```

You must also use the **mqsisetdbparms** command to define the security identity at the broker run time.

Connecting the terminals:

Connect the In terminal to the node from which outbound messages bound are routed.

Connect the Out or Failure terminal of this node to another node in this message flow to process the message further, process errors, or send the message to an additional destination.

If you connect one of these output terminals to another node in the message flow, the local environment that is associated with the message is enhanced with the following information for each destination to which the message has been put by this node:

Location	Description
WrittenDestination.Email.smtpServer	The Server:Port of the SMTP server.
WrittenDestination.Email.messageId	The ID of the email sent message.

These values are written in WrittenDestination within the local environment tree structure.

If you do not connect either terminal, the local environment tree is unchanged.

Terminals and properties:
The EmailOutput node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the message is routed if a failure is detected when a message is propagated. Connect the Failure terminal of this node to another node in the message flow to process errors.
Out	The output terminal to which the message is routed if it has been propagated successfully. Connect the Out terminal of this node to another node in the message flow to process the message further or send the message to an additional destination.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file at deployment).

The EmailOutput node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, EmailOutput	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

Use the EmailOutput node Basic properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
SMTP Server and Port	No	Yes		This property defines the SMTP server and port to which emails are sent from this node, and is in the format *server:port*; for example: *my.smtp.server:25*. The port value is optional, but if you do not specify a port value, the default value is 25. You can specify an alias value for this property. If the alias exists at run time, the specified values are used. If the alias does not exist at run time, the broker assumes the value to be a valid SMTP host.	smtpServer

The EmailOutput node Email properties are described in the following table.

Property	M	C	Default	Description
To Addresses	No	No		The main recipient or recipients of the email. This property can include a single email address or a comma-separated list of email addresses.
Cc Addresses	No	No		The carbon copy recipient or recipients of the email. This property can include a single email address or a comma-separated list of email addresses.
Bcc Addresses	No	No		The blind carbon copy recipient or recipients of the email. This property can include a single email address or a comma-separated list of email addresses.
From Address	No	No		The email address of the sender of the email.
Reply-To Address	No	No		The email address to which recipients of the email reply.
Subject of email	No	No		The subject of the email.
Email message text	No	No		The main text of the email. Use this property to provide a static main body of an email. If you use this property, it overrides the content that is provided in the body of the message tree that is passed to the input node. If you do not specify a value for this property, the text of the email is the body of the message tree that is passed to the EmailOutput node.
Body Content Type	No	No	`text/plain`	You can use this property to force the content type for the body of the email message. Valid values are: • `None` • `text/plain` • `text/html` • `text/xml`

The EmailOutput node Security properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Security Identity	No	Yes		A security identifier to retrieve a user ID and password that are configured at the broker run time.	securityIdentity

The EmailOutput node Attachment properties are described in the following table.

Property	M	C	Default	Description
Attachment Content	No	No		An XPath or ESQL expression that references an element; for example, an element in the message tree, or local environment. The content of the attachment is the value of the element that is referenced.
Attachment Content Name	No	No		The name of the attachment that is seen by the recipient of the email. This property is optional. If you do not specify a name, a default name is assigned.
Attachment Content Type	No	No	`text/plain`	The type of the attachment. This property is optional, even if you have specified an attachment. Valid values are: • `text/plain` is simple text. • `text/html` is HTML. • `text/xml` is XML. • `application/octet-stream` is the default type for non-text and HTML (binary data).
Attachment Content Encoding	No	No	`7bit`	The encoding of the attachment. This property is optional. If you do not specify a value, a default encoding is assigned. Valid values are: • `7bit` is the default value for ASCII text. • `base64` is used for non-ASCII data, whether it is non-English or binary data. • `quoted-printable` is a more readable-alternative to base64. Use quoted-printable when the majority of the data is ASCII text with some non-ASCII parts. This option provides a more compact encoding because the ASCII parts are not encoded.
Multipart Content Type	No	No	`Mixed`	The type of multipart. Valid values are: • `Mixed`: each MIME body part is independent of the others. • `Alternative`: Each MIME body part is an alternative to the others. • `Related`: All MIME body parts should be considered in the aggregate only.

The Validation properties of the EmailOutput node are described in the following table.

See “Validation properties” on page 4169 for a full description of these properties.

Property	M	C	Default	Description	**mqsiapplybaroverride command property**
Validate	Yes	Yes	`Inherit`	This property controls whether validation takes place. Valid values are `None`, `Content and Value`, `Content`, and `Inherit`.	validateMaster
Failure action	Yes	No	`Exception`	This property controls what happens if validation fails. You can set this property only if you set `Validate` to `Content` or `Content and Value`. Valid values are `User Trace`, `Local Error Log`, `Exception`, and `Exception List`.	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Local environment tree structure" on page 1056
The local environment tree is a part of the logical message tree in which you can store information while the message flow processes the message.

"MIME parser and domain" on page 1117
Use the MIME domain if your messages use the MIME standard for multipart messages.

Related tasks:

"Sending an email" on page 1789
You can send an email with a static subject and static text to a static list of recipients.

"Sending an email with an attachment" on page 1790
You can send an email with a fixed subject and fixed text, and an attachment, to a static list of recipients.

"Producing dynamic email messages" on page 1791
You can produce an email where the SMTP server, list of recipients, subject, text, and multiple attachments are all determined at run time.

"Sending a MIME message" on page 1795
You can send an email that is constructed from a MIME message.

"Changing connection information for the EmailOutput node" on page 1798
You can configure the SMTP server, port number, and security identity for the EmailOutput node as a broker external resource.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Validating messages" on page 1478
The broker provides validation based on the message model for predefined messages.

"Accessing the local environment tree" on page 2463
The local environment tree has its own correlation name, LocalEnvironment, and you must use this name in all ESQL statements that refer to or set the content of this tree.

Related reference:

"Validation properties" on page 4169
You can control validation by setting properties on the Validate and Parser Options tabs for the nodes that are listed in the following table.

EndpointLookup node

Use the EndpointLookup node to retrieve service endpoint information held in the WebSphere Service Registry and Repository (WSRR). The entities that match the specified search criteria are stored in the local environment. Depending on the

node configuration, the Web service URL destination used by the SOAP and HTTP request nodes can also be set. The input message is not modified.

This topic contains the following sections:

Purpose:
The EndpointLookup node retrieves service endpoint information related to a WSRR service described by WSDL. A WSDL definition defines a service in terms of an interface (referred to as a portType) made available at a specified port. The WSDL port defines the endpoint information required to access the service. Endpoints are retrieved according to search criteria defined by node properties, optionally supplemented or overridden by local environment definitions at run time. See "LocalEnvironment overrides" on page 4409 for more details.

The retrieved data is placed in the local environment tree, making it available to subsequent nodes. The input message received by the node is propagated to the output terminal unchanged. In addition, the EndpointLookup node can automatically set up the destination URL to be used by a subsequent SOAPRequest, SOAPAsyncRequest or HTTPRequest node, depending on the value of the `Match Policy` property, see "EndpointLookup node processing." This is done by the node setting the local environment overrides that are used by those nodes.

EndpointLookup node processing:
The EndpointLookup node is contained in the **Web services** drawer of the message flow node palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

When the EndpointLookup node receives a message the following steps occur in sequence.

1. The EndpointLookup node retrieves the service data from the WSRR by using the specified search criteria.
2. If one or more matches are found, the EndpointLookup node adds a representation of those endpoints to the local environment tree.
   - If `Match Policy` is set to `One`, a single entity is returned by WSRR and added to the local environment tree. A different entity might be returned each time the query is issued. In addition, the retrieved endpoint value is set as the local environment override for the destination URL used by SOAPRequest, SOAPAsyncRequest, or HTTPRequest nodes. If the registry contains more than one entity that matches the specified search criteria it is not possible to determine which one is returned by WSRR.
   - If `Match Policy` is set to `All`, all matching entities are added to the local environment tree. The order of the entities is determined by WSRR and might vary between queries. The destination URL used by SOAPRequest, SOAPAsyncRequest, or HTTPRequest nodes is not set. Instead, you must add

a compute node to your message flow to select the required address and to set up the local environment settings required by those request nodes.

The input message is propagated unchanged to the Out terminal. The local environment tree is propagated to the Out terminal, where it is available for further processing by transformation nodes. See "EndpointLookup node output" on page 1894 for details of the local environment output tree.

3. If no matches are found, the EndpointLookup node propagates the input message to the NoMatch terminal.
4. If a processing error occurs, for example if the WSRR server configured on the `DefaultWSRR` configurable service object cannot be connected to, or the connection times out, the EndpointLookup node propagates the input message unchanged to the Failure terminal. The ExceptionList is populated with details of the error.

Sample:
Look at the following sample to see how to use this node:

- WebSphere Service Registry and Repository Connectivity

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

LocalEnvironment overrides:
You can override the RegistryLookup node properties by using local environment settings. See "Dynamically defining the search criteria" on page 1891.

Terminals and properties:
When you have put an instance of the EndpointLookup node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view.

The EndpointLookup node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the message is routed if an error occurs within the node's processing.
Out	The output terminal to which the unmodified input message and updated local environment containing the matched registry data is sent.
NoMatch	The terminal to which the input message is sent if no matching entity is found based on the specified search criteria.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The EndpointLookup node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type: EndpointLookup	The name of the node.
Short description	No	No	None	A brief description of the node.
Long description	No	No	None	Text that describes the purpose of the node in the message flow.

The EndpointLookup node Basic properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
PortType Name	No	Yes	None	Name tuple that uniquely identifies a WebSphere Service Registry and Repository defined WSDL service portType. At least one of the properties is required. If you leave all three property values blank, an error message is shown when you try to save.	name
PortType Namespace	No	Yes	None		namespace
PortType Version	No	Yes	None		portVersion
User Properties	No	No	None	Allows a query to specify user-defined properties. Add `User Properties` by clicking **Add**. `User Properties` refer to the Additional Properties that are used to catalog the entities in WSRR. Enter values for `Property Name`, which is the case-sensitive match of the additional property in WSRR, `Property Type`, and `Property Value`. The `Property Type` can be: • a String (the default), in which case the `Property Value` is a character string to be matched with the additional property value present in WSRR • XPATH, or ESQL, in which case the `Property Value` is a XPath or ESQL expression which locates a field in the message tree that contains the character string to be matched with the additional property value present in WSRR. These `User Properties` and `Classification` properties are used in the query to uniquely identify the WSDL service port.	
Classification	No	No	None	The Web Ontology Language (OWL) classification system property. Each classifier is a class in OWL, and has a Uniform Resource Identifier (URI). Using classifications in the registry can help to make objects easier to find and can also add meaning to custom objects that are unique to a particular system. Add a `Classification` by clicking **Add** and typing the complete fully-qualified OWL URI for the OWL classification. For example, it can define a particular service endpoint's lifecycle state. These `User Properties` and `Classification` properties are used in the query to uniquely identify the WSDL service port.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Match Policy	Yes	No	`One`	WSRR can contain multiple entities that match the search criteria specified by the previous properties. If `Match Policy` is set to `One`, at most one matching entity is returned. If `Match Policy` is set to `All`, all matching entities are returned. See "EndpointLookup node output" on page 1894. If you request a single matching entity by setting `Match Policy` to `One`, the retrieved endpoint value is set as the local environment override for the destination URL used by SOAPRequest, SOAPAsyncRequest, or HTTPRequest nodes.	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message flow aggregation" on page 2718
Aggregation is the generation and fan-out of related requests that are derived from a single input message, and the fan-in of the corresponding replies to produce a single aggregated reply message.

"Dynamically defining the search criteria" on page 1891
You can use the RegistryLookup and EndpointLookup nodes to issue WebSphere Service Registry and Repository (WSRR) queries specified in the local environment.

"EndpointLookup node output" on page 1894
Use an EndpointLookup node to retrieve the endpoint addresses for WSDL service definitions held in WebSphere Service Registry and Repository (WSRR).

"WebSphere Service Registry and Repository" on page 1875
The WebSphere Service Registry and Repository (WSRR) is a central repository of entities. A wide range of entities can be stored and retrieved, including user-defined concepts and definitions related specifically to Web services, such as WSDL services, service interfaces, and associated policies.

"Configurable services" on page 1296
Configurable services are typically runtime properties. You can use them to define properties that are related to external services on which the broker relies; for example, an SMTP server or a JMS provider.

Related tasks:

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Configuring aggregation flows" on page 2721
Use aggregation message flows to generate and fan-out a number of related requests, fan-in the corresponding replies, and compile those replies into a single aggregated reply message, by using the AggregateControl, AggregateRequest, and AggregateReply nodes.

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

"Populating Destination in the local environment tree" on page 2467
Use the Destination subtree to set up the target destinations that are used by output nodes, the HTTPRequest node, the SOAPRequest node, the SOAPAsyncRequest node, and the RouteToLabel node. The following examples show how you can create and use an ESQL procedure to perform the task of setting up values for each of these uses.

Related reference:

"RegistryLookup node" on page 4646
Use the RegistryLookup node to retrieve any type of entity held in the WebSphere Service Registry and Repository (WSRR). The entities that match the specified search criteria are stored in the local environment. The input message is not modified.

"SOAPRequest node" on page 4828
Use the SOAPRequest node to send a SOAP request to the remote Web service.

"HTTPRequest node" on page 4488
Use the HTTPRequest node to interact with a web service.

Extract node

Use the Extract node to extract the contents of the input message that you want to be processed by later nodes in the message flow.

Attention: The Extract node is deprecated in WebSphere Message Broker Version 6.0 and later releases. Although message flows that contain an Extract node remain valid, redesign your message flows where possible to replace Extract nodes with Mapping nodes.

This topic contains the following sections:
- "Purpose"
- "Using this node in a message flow" on page 4413
- "Terminals and properties" on page 4413

Purpose:
Using the Extract node, you can create a new output message that contains only a subset of the contents of the input message. The output message comprises only those elements of the input message that you specify for inclusion when configuring the Extract node, by defining mapping statements.

The Extract node is contained in the **Database** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:
You might find this node useful if you require only a subset of the message after initial processing of the whole message. For example, you might want to store the whole message for audit purposes (in the Warehouse node), but propagate only a small part of the message (order information, perhaps) for further processing.

For example, you receive orders from new clients and you want to collect their names and addresses for future promotions. Use the Extract node to get this information from each order, and send it as a new message to head office. These messages are processed at head office so that the customer details can be included in the next marketing campaign.

Terminals and properties:
When you have put an instance of the Extract node into a message flow, you can configure it. For more information, see "Configuring a message flow node" on page 1503. The properties of the node are displayed in the Properties view. (If you double-click the Extract node, you open the New Message Map dialog box.) All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The Extract node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the input message is routed if a failure is detected during extraction.
Out	The output terminal to which the transformed message is routed if the input message is processed successfully.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined), the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Extract node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	Extract	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The Extract node Basic properties are described in the following table.

Property	M	C	Default	Description
Mapping module	Yes	No	Extract	The name of the mapping routine that contains the statements to run against the message tree. By default, the name that is assigned to the mapping routine is identical to the name of the mappings file in which the routine is defined. The default name for the file is the name of the message flow concatenated with the name of the node when you include it in the message flow (for example, `MFlow1_Extract.msgmap` for the first Extract node in message flow MFlow1). You cannot specify a value that includes spaces. To work with the mapping routine that is associated with this node, right-click the node and click **Open Mappings**. If the mapping routine does not exist, it is created for you with the default name in the default file. If the file exists already, you can also open file *`flow_name_node_name`*`.msgmap` in the Broker Development view. A mapping routine is specific to the type of node with which it is associated; you cannot use a mapping routine that you have developed for an Extract node with any other node that uses mappings (for example, a DataInsert node). If you create a mapping routine, you cannot call it from any other mapping routine, although you can call it from an ESQL routine. For more information about working with mapping files, and defining their content, see “Using message mappings” on page 2228.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see “Configuring monitoring event sources using monitoring properties” on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

“Message flows overview” on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

“Client application programming interfaces” on page 1038
You can configure the nodes in your message flows to customize the behavior of those nodes by using one or more of the supported programming interfaces.

Related tasks:

“Deciding which nodes to use” on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

“Configuring transactionality for message flows” on page 1290
A message flow runs in a single transaction, which is started when data is received by an input node, and can be committed or rolled back when all processing has completed.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Using message mappings" on page 2228
Message mappings define the blueprint for creating a message.

Related reference:

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"DataInsert node" on page 4386
Use the DataInsert node to interact with a database in the specified ODBC data source.

FileInput node

Use the FileInput node to process messages that are read from files.

This topic contains the following sections:

Purpose:
One or more messages can be read from a single file, and each message is propagated as a separate flow transaction. The part of a file that generates one message flow transaction is called a record. A file can be a single record, or a series of records. Properties on the node specify how the FileInput node determines the records in a file.

The FileInput node is contained in the **File** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Message structure

The FileInput node handles messages in the following message domains:
- XMLNSC
- DataObject
- JSON
- BLOB
- MIME
- MRM
- JMSMap
- JMSStream
- XMLNS

When the FileInput node propagates a message, it stores information about it in the LocalEnvironment.File message tree. If the input file is empty, an empty

message is propagated (assuming that it is valid). The following table lists the LocalEnvironment.File message tree structure. The elements contain data about the current record.

Element Name	Element Data Type	Description
Directory	CHARACTER	Absolute directory path of the input directory in the form used by the file system of the broker. For example, on Windows systems, this starts with the drive letter prefix (such as `C:`).
Name	CHARACTER	File name and extension.
LastModified	TIMESTAMP	Date and time the file was last modified.
TimeStamp	CHARACTER	Date and time, in the Coordinated Universal Time (UTC) zone, the input node started processing the file as a character string. This data is the string used to create archive and backout file names if a timestamp is included.
The following elements contain data about the current record:		
Offset	INTEGER	The start of the record within the file. The first record starts at offset 0. When Offset is part of the End of Data message tree, this value is the length of the input file.
Record	INTEGER	The number of the record within the file. The first record is record number 1. When Record is part of the End of Data message tree, this value is the number of records.
Delimiter	CHARACTER	The characters used to separate this record from the preceding record, if `Delimited` is specified in `Record detection`. The first record has a null delimiter. When Delimiter is part of the End of Data message tree, this value is the delimiter that follows the last record, if any.
IsEmpty	BOOLEAN	Whether the record that is propagated by the message flow is empty. IsEmpty is set to TRUE if the current record is empty. When IsEmpty is part of the End of Data message tree, this property is always set to TRUE.

Using this node in a message flow:
The FileInput node can be used in any message flow that must accept messages in files. You can also look at the following samples to see how to use this node:

- Batch Processing
- WildcardMatch

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

If you configure a File node to use FTP, your network might need it to connect to an FTP proxy server, instead of directly to the remote FTP server. How you configure the File nodes to use an FTP proxy depends on how that proxy handles requests. For some FTP proxies, you must encode the target FTP server information in the logon credentials that you create with the **mqsisetdbparms** command. For example, some FTP proxies support the following values:

```
Username: FtpTargetHostUsername@ProxyUserName@TargetFtpHostname
Password: TargetFtpUserPassword@ProxyUserPassword
```

Other proxies might require different encodings, or might require external configuration, or you might not be able to use them with the File nodes.

Configuring the FileInput node:

When you have put an instance of the FileInput node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (the properties that do not have a default value defined) are marked with an asterisk.

Configure the FileInput node:

1. Optional: On the **Description** tab, enter a `Short description`, a `Long description`, or both. You can also rename the node on this tab.
2. On the **Basic** tab, enter the directories and files to be processed by the FileInput node, together with what to do with any duplicate files encountered.
   - In `Input directory`, specify the directory from which the FileInput node obtains files. Specify the directory as either an absolute or a relative directory path. If the directory path is relative, it is based on the directory specified in the environment variable MQSI_FILENODES_ROOT_DIRECTORY. An example on Windows systems is `C:\fileinput`. An example on UNIX systems is `/var/fileinput`.

     The FileInput node creates an `mqsitransitin` subdirectory in the specified input directory to hold and lock input files while they are being processed. If an execution group that processes files in this input directory is removed, check the `mqsitransitin` subdirectory for partially processed or unprocessed files. Move any such files back into the input directory (and remove the execution group UUID prefix from the file names) so that they can be processed by a different execution group. For more information about the `mqsitransitin` subdirectory, see "How multiple file nodes share access to files in the same directory" on page 1818.
   - In `File name or pattern`, specify a pattern for the file name. It is either a file name or a character sequence (a pattern) that matches a file name. A pattern is a sequence containing at least one of the following wildcard characters:

Wildcard character	Description	Example
*	Any sequence of zero or more characters	`*.xml` matches all file names with an `xml` extension
?	Any single character	`f??????.csv` matches all file names consisting of the letter f followed by six characters and then the sequence `.csv`.

     For a file to be processed, its name must match the pattern.

     If you specify a file name pattern that contains wildcard characters, the FileInput node copies the characters in the file name matched by wildcards, together with any intermediate characters, to the LocalEnvironment.Wildcard.WildcardMatch element. See "File name patterns" on page 1830 for more information.
   - Select `Action on successful processing` to specify the action that the FileInput node takes after successfully processing the file. The action can be to move the file to the archive subdirectory, to augment the file name with a time stamp and move the source file to the archive subdirectory, or to delete the file.
     – If you select `Move to Archive Subdirectory`, the source file is moved to the archive subdirectory of the input directory. The subdirectory name is `mqsiarchive`. For example, if the input directory is `/var/fileinput`, the

absolute path of the archive subdirectory is /var/fileinput/mqsiarchive. If this directory does not exist, the broker creates it when it first tries to move a file there.

– If you select Add Timestamp and Move to Archive Subdirectory, the current date and time are added to the file name, and the file is then moved to mqsiarchive.

– If you select Delete, the file is deleted after successful processing.

The FileInput node writes a message to the user trace, if user tracing is in operation, whenever it processes a file.

- Select Replace duplicate archive files if you want to replace a file in the archive subdirectory with a successfully processed file of the same name. If you do not set this option, and a file with the same name exists in the archive subdirectory, the node throws an exception when it tries to move the successfully processed file.

3. On the **Input Message Parsing** tab, set values for the properties that the node uses to determine how to parse the incoming message.
   - In Message domain, select the name of the parser that you are using from the supplied list. The default is BLOB. You can choose from the following options:
     – XMLNSC
     – DataObject
     – JSON
     – BLOB
     – MIME
     – MRM
     – JMSMap
     – JMSStream
     – XMLNS

     You can also specify a user-defined parser, if appropriate.
   - If you are using the MRM or IDOC parser, or the XMLNSC parser in validating mode, select the Message set that you want to use. This list is populated with available message sets when you select MRM, XMLNSC, or IDOC as the domain.
   - If you are using the MRM parser, select the correct message type from the list in Message type. This list is populated with available message types when you select the MRM parser.
   - If you are using the MRM or IDOC parser, select the correct message format from the list in Message format. This list is populated with available message formats when you select the MRM or IDOC parser.
   - Specify the message coded character set ID in Message coded character set ID.
   - Select the message encoding from the list in Message encoding or specify a numeric encoding value. For more information about encoding, see "Data conversion" on page 1151.
4. On the **Parser Options** subtab:
   - Parse timing is, by default, set to On Demand, which causes parsing of the message to be delayed. To cause the entire message to be parsed immediately, set this property to Immediate or Complete. See "Parsing on demand" on page 4173 for more details.

- If you are using the XMLNSC parser, set values for the properties that determine how the XMLNSC parser operates. For more information, see "Manipulating messages in the XMLNSC domain" on page 2546.

5. On the **Polling** tab, enter a value for the `Polling interval`. This property controls the frequency with which the FileInput node accesses the file system looking for files to process.

   After the initial scan of the directory when the flow is started, whenever the directory is found to contain no files that match the input pattern, the FileInput node waits for the period defined by this property. This process avoids the need for the FileInput node to be continually accessing the file system, and consuming large amounts of system resource.

   The smaller the value set in this property, the more quickly the FileInput node discovers files that are in the input directory. However, a smaller value increases the use of system resources. A larger value reduces the use of system resource but at the cost of the FileInput node discovering files to process less quickly.

   Do not use this property as a means to regulate work, or to schedule processing. If you want the FileInput node to monitor the input directory for selected periods only, start and stop the message flow at appropriate times.

   If you select the `Remote Transfer` property and set the `Scan delay` property on the FTP tab, the value that you set for `Scan delay` overrides the value set for `Polling interval`.

6. Use the **Retry** tab to define how retry processing is performed when a message flow fails:
   - `Retry mechanism` determines the action that occurs if the flow fails:
     - Select `Failure` for the node to report a failure without any retry attempts.
     - Select `Short retry` for the node to try again before reporting a failure if the condition persists. The number of times that it tries again is specified in `Retry threshold`.
     - Select `Short retry and long retry` for the node to try again, first using the value in `Retry threshold` as the number of attempts it is to make. If the condition persists after the `Retry threshold` value has been reached, the node then uses the value of `Long retry interval` between attempts.
   - Specify a value for the `Retry threshold` property. The number of times the node tries the flow transaction again if the `Retry mechanism` property is set to either `Short retry` or `Short retry and long retry`.
   - Specify a value for the `Short retry interval` property. The length of time, in seconds, to wait between short retry attempts.
   - Specify a value for the `Long retry interval` property. The length of time to wait between long retry attempts until a message is successful, the message flow is stopped, or the message flow is redeployed. The `MinLongRetryInterval` broker property defines the minimum value that the `Long retry interval` can take. If the value is lower than the minimum, the broker value is used.
   - Specify a value for the `Action on failing file` property to determine what the node is to do with the input file after all attempts to process its contents fail:
     - `Move to Backout Subdirectory`. The file is moved to the backout subdirectory of the input directory. The name of this subdirectory is `mqsibackout`. If the input directory is `/var/fileinput`, the absolute path of the backout subdirectory is `/var/fileinput/mqsibackout`. If this subdirectory does not exist, the broker creates it when it first tries to

move a file there. If the file cannot be moved to this subdirectory, perhaps because a file of the same name exists there, the node adds the current date and time to the file name and makes a second attempt to move the file. If this second attempt fails, the node stops processing. Messages BIP3331 and BIP3325 are issued. Resolve the problem with the subdirectory or file before attempting to restart the message flow.

- `Delete`. The file is deleted after processing fails.
- `Add Time Stamp and Move to Backout Subdirectory`. The current date and time are added to the file name, and then the file is moved to the backout subdirectory.

7. Use the **Records and Elements** tab to specify how each file is interpreted as records:
   - Use the `Record detection` property to determine how the file is split into records, each of which generates a single message. Choose from the following options:
     - `Whole File` specifies that the whole file is a single record. A limit of 100 MB applies to the size of the files.
     - `Fixed Length` specifies that each record is a fixed number of bytes in length. Each record contains the number of bytes specified in the `Length` property, except possibly a shorter final record in the file. The value specified in `Length` must be in the range 1 byte through 100 MB. The default is 80 bytes.
     - Select `Delimited` if the records that you are processing are separated, or terminated, by a DOS or UNIX line end or by a sequence of user-defined delimiter bytes. Specify the delimiter and delimiter type in the `Delimiter` and `Delimiter type` properties. A limit of 100 MB applies to the length of the records.
     - Select `Parsed Record Sequence` if the file contains a sequence of one or more records that are serially recognized by the parser specified in `Message domain`. The node propagates each recognized record as a separate message. If you select the `Record detection` option, the parser specified in `Message domain` must be either XMLNSC or MRM (either CWF or TDS physical format).
   -

     If you specify `Parsed Record Sequence` in `Record detection`, the FileInput node does not determine or limit the length of a record. Nodes that are downstream in the message flow might try to determine the record length or process a long record. If you intend to process large records in this way, ensure that your broker has sufficient memory. You might have to apply flow techniques described in the Large Messaging sample to make best use of the available memory.
   - If you specified `Delimited` in `Record detection`, use `Delimiter` to specify the delimiter to be used. Choose from:
     - `DOS or UNIX Line End`, which, on UNIX systems, specifies the line feed character (<LF>, X'0A'), and, on Windows systems, specifies a carriage return character followed by a line feed character (<CR><LF>, X'0D0A'). The node treats both of these strings as delimiters, irrespective of the system on which the broker is running. If they are both in the same file, the node recognizes both as delimiters. The node does not recognize X'15' which, on z/OS systems, is the 'newline' byte; specify a value of `Custom Delimiter` in this property and a value of `15` in the `Custom delimiter` property if your input file is coded using EBCDIC new lines, such as EBCDIC files from a z/OS system.

- Custom Delimiter, which permits a sequence of bytes to be specified in Custom delimiter

- In Custom delimiter, specify the delimiter byte or bytes to be used when Custom delimiter is set in the Delimiter property. Specify this value as an even-numbered string of hexadecimal digits. The default is X'0A' and the maximum length of the string is 16 bytes (represented by 32 hexadecimal digits).
- If you specified Delimited in Record detection, use Delimiter type to specify the type of delimiter. Permitted values are:
  - Infix. If you select this value, each delimiter separates two records. If the file ends with a delimiter, the zero length file content following the final delimiter is still propagated as a message although it contains no data.
  - Postfix. If you specify this value, each delimiter terminates a record. If the file ends with a delimiter, no empty record is propagated after the delimiter. If the file does not end with a delimiter, the file is processed as if a delimiter follows the final bytes of the file. Postfix is the default value.
- The FileInput node considers each occurrence of the delimiter in the input file as either separating (Infix) or terminating (Postfix) each record. If the file begins with a delimiter, the node treats the (zero length) file contents preceding that delimiter as a record and propagates an empty record to the flow. The delimiter is never included in the propagated message.

8. Use the **Validation** tab to provide validation based on the message set for predefined messages. For more information about validation, see "Validating messages" on page 1478. For information about how to complete this tab, see "Validation tab properties" on page 4169.
9. On the **FTP** tab, select the Remote Transfer property if you want the node to read files from an FTP or SFTP server using the following properties:
   - In Transfer Protocol, specify the protocol that is to be used for remote file transfer. Possible values are FTP and SFTP.
   - In Remote server and port, supply the IP address and port number of the FTP or SFTP server to be used. Use one of the following syntaxes:
     - *IP_address_or_URL*
     - *IP_address_or_URL*:*port_number*

     If you specify the IP address in IPv6 format, ensure that you enclose it in square brackets, as in the following examples:
     - [12a::13bd:24cd]
     - [12a::13bd:24cd]:123 where 123 is the port number

     If you are using FTP and you do not specify a port number, 21 is assumed. If you are using SFTP and you do not specify a port number, a port number of 22 is assumed. However, if an FtpServer configurable service is defined, you can enter the name of the configurable service in this field. For information about how an FtpServer configurable service definition and the properties on this tab interact, see "FtpServer configurable service properties" on page 3794.
   - In Security identity, specify the name of a security identity that has been defined using the **mqsisetdbparms** command. The user identifier and password that are to be used to log on to the FTP or SFTP server are obtained from this definition, the name of which must have an ftp:: prefix. The value of this property is overridden by the value in the securityIdentity property of the FtpServer configurable service, if it is set.

- In `Server directory`, specify the directory in the FTP or SFTP server from which to transfer files. The default is a period (`.`) which means the default directory after logon. If you specify a relative path, the directory is based on the default directory after FTP or SFTP logon. Ensure that the syntax of the path conforms to the file system standards in the FTP or SFTP server. The value in this property is overridden by the value in the remoteDirectory property of the FtpServer configurable service, if it is set.
- In `Transfer mode`, specify how files are transferred. Select `Binary` if the file contents are not to be transformed. Select `ASCII` if the file is to be transmitted as ASCII. The value of this property is overridden by the value in the transferMode property of the FtpServer configurable service, if it is set.

  This property is valid only when FTP is selected as the protocol for remote transfer. If you have specified SFTP as the protocol, the `Transfer mode` mode property is ignored and binary encoding is used.
- In `Scan delay`, specify the delay, in seconds, between directory scans. The default is 60 seconds. The value set in this property overrides the value set for the polling interval on the **Polling** tab when the `Remote Transfer` property is selected. The value of this property is overridden by the value in the scanDelay property of the FtpServer configurable service, if it is set.

10. On the **Transactions** tab, set the transaction mode. Although all file operations are non-transactional, the transaction mode on this input node determines whether the rest of the nodes in the flow are to be run under sync point. Select `Yes` if you want the flow updates to be treated transactionally, if possible, or `No` if you do not. The default for this property is `No`.
11. Optional: On the **Instances** tab, set values for the properties that control the additional instances (threads) that are available for a node. For more details, see "Configurable message flow properties" on page 4020.

Terminals and properties:
The FileInput node terminals are described in the following table.

Terminal	Description
Failure	The output terminal to which a message is routed if an error occurs before a message is propagated to the Out terminal. Messages propagated to this terminal are not validated, even if you have specified, using the `Validate` property, that validation is to take place.
Out	The output terminal to which a message is routed if it has been successfully extracted from the input file. If no errors occur within the input node, a message received from an external resource is always sent to the Out terminal first.
End of Data	The output terminal to which the End of Data message is routed after all the messages in a file have been processed. The End of Data message flow transaction is initiated only if this terminal is attached.
Catch	The output terminal to which the message is routed if an exception is thrown downstream and caught by this node. Exceptions are caught only if this terminal is attached.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

Description properties:

Property	M	C	Default	Description
Node name	No	No	FileInput	The name of the node.
Short description	No	No	None	A brief description of the node.
Long description	No	No	None	Text that describes the purpose of the node in the message flow.

Basic properties:

Property	M	C	Default	Description	**mqsiapplybaroverride command property**
Input directory	Yes	Yes	None	The path of the directory from which input files are processed. The directory must be in a file system to which the broker has access. If the input directory does not exist, no files are processed. The FileInput node checks that the input directory exists at intervals that are defined by the `Scan delay` property. The input directory must exist, even if you are processing files over FTP or SFTP. The FileInput node creates an `mqsitransitin` subdirectory in the specified input directory. The `mqsitransitin` subdirectory holds and locks input files while they are being processed. If an execution group that processes files in this input directory is removed, check the `mqsitransitin` subdirectory for partially processed or unprocessed files. Move any such files back into the input directory (and remove the execution group UUID prefix from the file names) so that they can be processed by a different execution group.	inputDirectory
File name or pattern	Yes	Yes	*	A file name or string containing optional wildcard characters (* or ?) identifying the file or files to process from the input directory.	filenamePattern
Action on successful processing	Yes	No	`Delete`	The action the node takes on the file after successfully processing the contents. Valid options are: • `Move to Archive Subdirectory` • `Add Time Stamp and Move to Archive Subdirectory` • `Delete`	
Replace duplicate archive files	Yes	No	Cleared	This property controls whether the node replaces existing archive files with the same name as the input file. It applies only when `Action on successful processing` is not `Delete`.	

The FileInput node Input Message Parsing properties are described in the following table:

Property	M	C	Default	Description	**mqsiapplybaroverride command property**
Message Domain	No	No		The domain that is used to parse the incoming message.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Message Set	No	No		The name or identifier of the message set in which the incoming message is defined. If you set this property, and then update the project dependencies to remove this message set reference, a warning is issued. Either update the `Message Set` property, or restore the reference to this message set project.	
Message Type	No	No		The name of the incoming message.	
Message Format	No	No		The name of the physical format of the incoming message.	
Message coded character set ID	Yes	No	Broker System Default	The ID of the coded character set used to interpret bytes of the file being read.	messageCodedCharSetIdProperty
Message encoding	Yes	No	Broker System Determined	The encoding scheme for numbers and large characters used to interpret bytes of the file being read. Valid values are `Broker System Determined` or a numeric encoding value. For more information about encoding, see "Data conversion" on page 1151.	messageEncodingProperty

Parser Options properties:

Property	M	C	Default	Description
Parse timing	No	No	`On Demand`	This property controls when an input message is parsed. Valid values are: • `On Demand` • `Immediate` • `Complete` For a full description of this property, see "Parsing on demand" on page 4173.
Build tree using XML schema data types	No	No	Cleared	This property controls whether the syntax elements in the message tree have data types taken from the XML schema.
Use XMLNSC compact parser for XMLNS domain	No	No	Cleared	Specifies whether the XMLNSC Compact Parser is used for messages in the XMLNS Domain. If you set this property, the message data is displayed under XMLNSC in nodes that are connected to the output terminal when either of the following items is XMLNS: • The input MQRFH2 header. • The Input Message Parsing property, Message Domain.
Retain mixed content	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree for mixed text in an input message. If you select the check box, elements are created for mixed text. If you clear the check box, mixed text is ignored and no elements are created.

Property	M	C	Default	Description
Retain comments	No	No	Cleared	Specifies whether the XMLNSC parser creates elements in the message tree for comments in an input message. If you select the check box, elements are created for comments. If you clear the check box, comments are ignored and no elements are created.
Retain processing instructions	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree for processing instructions in an input message. If you select the check box, elements are created for processing instructions. If you clear the check box, processing instructions are ignored and no elements are created.
Opaque elements	No	No	Blank	This property is used to specify a list of elements in the input message that are to be opaquely parsed by the XMLNSC parser. Opaque parsing is performed only if validation is not enabled (that is, if `Validate` is `None`); entries that are specified in `Opaque Elements` are ignored if validation is enabled.

Polling property:

Property	M	C	Default	Description	**mqsiapplybaroverride command property**
Polling interval (seconds)	Yes	Yes	`5`	The polling interval in seconds.	waitInterval

Retry properties:

Property	M	C	Default	Description	**mqsiapplybaroverride command property**
Retry mechanism	Yes	No	`Failure`	How the node handles a flow failure. Valid options are: • `Failure` • `Short retry` • `Short and long retry`	
Retry threshold	Yes	Yes	`0`	The number of times to try the flow transaction again when the `Retry mechanism` property value is `Short retry`.	retryThreshold
Short retry interval	No	Yes	`0`	The interval, in seconds, between each retry if `Retry threshold` property is not zero.	shortRetryInterval
Long retry interval	No	Yes	`300`	The interval between retries, if the `Retry mechanism` property is `Short and long retry` and the retry threshold has been exhausted.	longRetryInterval
Action on failing file	Yes	Yes	`Move to Backout Subdirectory`	The action that the node takes with the input file if all attempts to process the contents of the input file fail. Valid options are: • `Move to Backout Subdirectory` • `Delete` • `Add Time Stamp and Move to Backout Subdirectory`	

Records and Elements properties:

Property	M	C	Default	Description
Record detection	Yes	No	`Whole File`	The mechanism used to identify records in the input file. Valid options are: • `Whole File` When you set this property to `Whole File`, files larger than 2 Gb are not supported. • `Fixed Length` • `Delimited` • `Parsed Record Sequence`
Length	Yes	No	`80`	The length of each record, in bytes, when `Fixed Length` record detection is selected.
Delimiter	Yes	No	`DOS or UNIX Line End`	The type of delimiter bytes that separate, or end, each record when `Delimited` record detection is selected. Valid options are: • `DOS or UNIX Line End` • `Custom Delimiter`
Custom delimiter	No	No		The delimiter bytes, expressed in hexadecimal, when `Delimited` record detection and `Custom Delimiter` are selected. This property is mandatory only if the `Delimiter` property is set to `Custom Delimiter`.
Delimiter type	Yes	No	`Postfix`	The position of the delimiter when `Delimited` record detection is selected. Valid options are: • `Postfix` • `Infix` This property is ignored unless the `Delimiter` property is set to `Custom Delimiter`.
Skip first record	Yes	No	FALSE	Skip the first record in the file. The FileInput node will read the first record in the file but not propagate the record to the Out terminal. Records will be propagated as normal, from the second record onwards. Use this option when the first record is a header that does not need to be processed. It is not valid to use this option when using the whole file.

Validation properties:

For a full description of these properties, see "Validation properties" on page 4169.

Property	M	C	Default	Description	**`mqsiapplybaroverride` command property**
Validate	No	Yes	`None`	This property controls whether validation takes place. Valid values are: • `None` • `Content and Value` • `Content`	validateMaster
Failure action	No	No	`Exception`	This property controls what happens if validation fails. Valid values are: • `User Trace` • `Local Error Log` • `Exception` • `Exception List`	

FTP properties:

Property	M	C	Default	Description	mqsiapplybaroverride command property
Remote Transfer	No	Yes	Cleared	This property defines whether the node uses the remote file transfer properties listed on the **FTP** tab and reads files from either an FTP or SFTP server.	fileFtp
Transfer protocol	No	Yes	FTP	This property specifies the protocol to be used for remote transfer. Valid values are: • `FTP` • `SFTP`	remoteTransferType
Remote server and port	No	Yes	None	This property can have either of the following values: • The IP address or name (and, optionally, the port number) of a remote FTP or SFTP server; for example `ftp.server.com:21` or `sftp.server.com:22` • The name of a configurable service of type FtpServer If a configurable service name is specified, any or all the other remote transfer properties on the FTP tab can be overridden by the configurable service.	fileFtpServer
Security identity	No	Yes		The name of the user identification used to access the FTP or SFTP server. This property is overridden by the securityIdentity property, if set, in the FtpServer configurable service.	fileFtpUser
Server directory	No	Yes	"."	The directory on the FTP or SFTP server from which to transfer files. If you specify this property as a relative path, it is relative to the home directory after logon. This property is overridden by the remoteDirectory property, if set, in the FtpServer configurable service.	fileFtpDirectory
Transfer mode	No	No	`Binary`	The FTP transfer mode for transfer of file data. This property is valid only when FTP is selected as the protocol for remote transfer. Valid values are: • `Binary` • `ASCII` This property is overridden by the transferMode property, if set, in the FtpServer configurable service. If you have specified SFTP as the protocol for remote transfer, the `Transfer mode` property is ignored, and binary encoding is used.	
Scan delay	No	Yes	`60`	The delay, in seconds, between remote directory scans. This property overrides the value set for `Polling interval` when the `Remote Transfer` property is selected. This property is overridden by the scanDelay property, if set, in the FtpServer configurable service.	

Transactions properties:

Property	M	C	Default	Description
Transaction mode	No	Yes	`No`	The transaction mode on this input node determines whether the rest of the nodes in the flow are run under sync point. Valid options are: • `Yes` • `No`

Instances properties. For a full description of these properties, see "Configurable message flow properties" on page 4020.

Property	M	C	Default	Description	**`mqsiapplybaroverride` command property**
Additional instances pool	No	Yes	`Use Pool Associated with Message Flow`	The pool from which additional instances are obtained. • If you select `Use Pool Associated with Message Flow`, additional instances are obtained from the message flow pool. • If you select `Use Pool Associated with Node`, additional instances are allocated from the additional instances of the node based on the number specified in the `Additional instances` property.	componentLevel
Additional instances	No	Yes	0	The number of additional instances that the node can start if the `Additional instances pool` property is set to `Use Pool Associated with Node`.	additionalInstances

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"File name patterns" on page 1830
You can specify a file name pattern, using wildcard characters, to identify a file to be read by the FileInput and FTEInput nodes. You can also specify a file name pattern, using a single wildcard character, to name the file to be created by the FileOutput and FTEOutput nodes.

"How the broker processes files" on page 1814
The broker reads files with the FileInput, FTEInput, and FileRead nodes, and writes files with the FileOutput and FTEOutput nodes.

"How multiple file nodes share access to files in the same directory" on page 1818
WebSphere Message Broker controls access to files so that only one file node at a time can read or write to a file.

"Archiving" on page 1833
Files that are successfully processed by the FileInput node or FileOutput node can optionally be moved to the `mqsiarchive` subdirectory of the input or output directory.

"Using local environment variables with file nodes" on page 1820
You can use fields in the local environment to dynamically alter the behavior of the FileInput, FileOutput, FTEInput, and FTEOutput nodes. You can also find what values the output nodes used to process the file.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Parsers" on page 1072
A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from the internal message tree representation.

"Samples" on page 98
The WebSphere Message Broker Toolkit provides samples that show the features that are available in WebSphere Message Broker, and how to use them. This topic provides links to the information about the individual samples.

Related tasks:

"Reading files" on page 1834
Use the FileInput, CDInput, FTEInput, and FileRead nodes to read files.

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

"Validating messages" on page 1478
The broker provides validation based on the message model for predefined messages.

"Manipulating messages in the XMLNSC domain" on page 2546
If you are writing ESQL to process messages in the XMLNSC domain, it is helpful to learn about the structure of the message tree that the XMLNSC parser builds.

"Problems when developing message flows with file nodes" on page 3402
Use the advice given here to help you to resolve some common problems that can arise when you develop message flows that contain file nodes.

Related reference:

"Recognizing file records as messages to be parsed" on page 1817
Use the FileInput, FTEInput and FileRead nodes to segment your input file into messages that are to be parsed.

"Configurable message flow properties" on page 4020
When you add a message flow to a broker archive (BAR) file in preparation for deploying it to a broker, you can set additional properties that influence its run time operation. These properties are available for review and update when you select the **Manage and Configure** tab for the broker archive file.

"FileOutput node" on page 4430
Use the FileOutput node to write messages to files.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a

broker external resource.

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

"FtpServer configurable service properties" on page 3794
Select the properties and values that you want to change for an existing FtpServer configurable service or to create a new service.

"Validation properties" on page 4169
You can control validation by setting properties on the Validate and Parser Options tabs for the nodes that are listed in the following table.

"Parsing on demand" on page 4173
On-demand parsing, referred to as partial parsing, is used to parse an input message bit stream only as far as is necessary to satisfy the current reference. The parsers that can perform partial parsing of input messages are the JSON, MRM, XML, XMLNS, and XMLNSC parsers.

"Supported code pages" on page 4176
Application messages must conform to supported code pages.

"FTEOutput node" on page 4466
Use the FTEOutput node to write messages to files by using the WebSphere MQ File Transfer Edition.

"FTEInput node" on page 4461
Use the FTEInput node to receive files using the WebSphere MQ File Transfer Edition.

FileOutput node

Use the FileOutput node to write messages to files.

This topic contains the following sections:
- "Purpose"
- "Using this node in a message flow" on page 4433
- "Configuring the FileOutput node" on page 4433
- "Terminals and properties" on page 4433

Purpose:
You can write one or more messages from message flow transactions to a file in the file system of the broker. Each message, as it is written to a file, is converted to a sequence of bytes called a *record*. Records are accumulated until a process is triggered that completes the file and places it either in the specified output directory or a remote FTP or SFTP server directory. Properties on the node specify how records are accumulated into files and where the files are placed when they are finished.

The FileOutput node is contained in the **File** drawer of the palette and is represented in the workbench by the following icon:

Record processing

The FileOutput node writes files as a sequence of one or more records. Each record is generated from a single message received on the In terminal of the node.

By default, each file comprises a single record but properties on the FileOutput node can specify that the file comprises multiple records and how these records are accumulated in a file. If multiple records are written, the FileOutput node starts with an empty file and writes records to it until a message is received by the Finish File terminal. The node does not append new records to a file that exists. Multiple records can be accumulated in a file in the following ways:

- Concatenated: The record created from each message is added, unmodified, to the file.
- Padded: Each record is adjusted to be a specific length and padded with a padding byte, if necessary, before being added to the file.
- Delimited: A delimiter is used to separate or terminate the records as they are added to the file.

For each message received, whether on the In terminal or the Finish File terminal, you can modify the output directory and the name of the file to be written (or finished) by using elements of the message. On the node you can specify these elements, which, by default, identify elements in the local environment, on the **Request** properties tab.

File processing

The FileOutput node writes accumulated messages to a file, and places it in a specified directory (the output directory) at either of the following times:

- After each record, if the file is to contain a single record. (Specify this behavior by setting the `Record definition` property to `Record is Whole File` on the **Records and Elements** tab.)
- When the Finish File terminal receives a message.

The name of the output directory and the names of the output files are determined by the node properties that you specify and by elements of the message that is being processed.

The FileOutput node uses subdirectories of the output directory to store files during and after processing. All these subdirectories begin with the prefix `mqsi`, and include subdirectories called `mqsitransit` (the transit directory) and `mqsiarchive` (the archive directory). Records are not accumulated directly into a file in the output directory but are accumulated in a file in the transit directory. Files are moved from the transit directory to the output directory when the file is complete. If a file that is to be moved to the output directory has the same name as a file that is already there, you can choose whether the file in the output directory is deleted, moved to the archive directory (`mqsiarchive`), or renamed before being moved to the archive directory.

You can specify that the FileOutput node transfers files to a remote FTP or SFTP server as part of file processing. If the file is successfully transferred, it can be deleted from the local file system, or, optionally, retained for the rest of the file processing to occur as usual. The server is identified by the `Remote server and port` property on the node. Alternatively, you can override the node property by setting a value in the local environment. You can also use the local environment to specify commands to run before or after an FTP or SFTP transfer finishes. For more information, see “Local environment overrides for the FileOutput node” on page 4443.

During the file transfer operation, the FileOutput creates the destination file. However, the destination file is readable before the file transfer is complete. Therefore, ensure that remote applications do not read the file until the file transfer is complete.

When multiple records are written, no file processing occurs until a message is received on the Finish File terminal of the node. Any message received on the Finish File terminal causes the file to be moved from the transit directory to either the specified output directory or to a remote FTP or SFTP directory.

It is not an error if file processing is initiated when there is no file in the transit directory.

If you set the `Record definition` property to `Record is Whole File` on the **Records and Elements** tab, messages received on the Finish File processing are ignored because the file has already been processed.

The UMASK value for WebSphere Message Broker on UNIX and z/OS systems can be set using the MQSI_SET_DFE_UMASK environment variable as follows:

```
MQSI_SET_DFE_UMASK=nnnn
```

where *nnnn* is the UMASK value that you need to specify. The minimum permissions level required by WebSphere Message Broker is `rwrw_(660)`. If neither of the UMASK environment variable are set, then files are created with a UMASK of `6`.

Message propagation

For every message received on the In terminal and successfully processed by the node, a copy is propagated to the Out terminal for further processing if the terminal is attached.

For every message received on the Finish File terminal and successfully processed by the node, a copy is propagated to the End of Data terminal for further processing if the terminal is attached.

When the FileOutput node propagates a message, either to the Out terminal or to the End of Data terminal, it stores information in the LocalEnvironment.WrittenDestination.File message tree. This table describes the LocalEnvironment.WrittenDestination.File elements:

Element Name	Element Data Type	Description
Directory	CHARACTER	Absolute path of the output directory in the form used by the file system of the broker. For example, on Windows systems, the directory path starts with the drive letter prefix (such as `C:`).
Name	CHARACTER	Name of the output file.
Action	CHARACTER	Possible values are: • `Replace` if an output file of the same name is replaced. • `Create` if a new output file is created. • `Append` if this message is associated with a record that is added to an output file. • `Finish` if a Finish File message is received and no file is found to finish (for example, if `Record is Whole File` is specified and a message is sent to the Finish File terminal). • `Transmit` if the file was transferred by FTP or SFTP and the file was not retained.

Element Name	Element Data Type	Description
Timestamp	CHARACTER	The date and time, in character string form, when the node started to process this file. This value prefixes the names of files that are archived if you set the `Output file action` property to `Time Stamp, Archive and Replace Existing File` on the **Basic** tab.

Multiple instances

Several message flows might write to the same file, which can happen where there are additional instances of the flow, or where multiple flows contain FileOutput nodes. The FileOutput node permits only a single instance, within an execution group and between execution groups, to write to a file at the same time. While a record is being written, all other instances in the execution group must wait. The order in which instances gain access is not defined.

When the file is complete, the first instance to gain access processes it, and other instances do not find the file. The Action element of the LocalEnvironment.WrittenDestination.File message tree is set to `Finish` for all instances that fail to discover the file in the transit directory.

Using this node in a message flow:
The FileOutput node can be used in any message flow that sends messages to files. See "Working with files" on page 1807. You can also look at the following samples to see how to use this node:
- File Output
- Batch Processing
- WildcardMatch

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

If you configure a File node to use FTP, your network might need it to connect to an FTP proxy server, instead of directly to the remote FTP server. How you configure the File nodes to use an FTP proxy depends on how that proxy handles requests. For some FTP proxies, you must encode the target FTP server information in the logon credentials that you create with the **mqsisetdbparms** command. For example, some FTP proxies support the following values:

```
Username: FtpTargetHostUsername@ProxyUserName@TargetFtpHostname
Password: TargetFtpUserPassword@ProxyUserPassword
```

Other proxies might require different encodings, or might require external configuration, or you might not be able to use them with the File nodes.

Configuring the FileOutput node:
When you have put an instance of the FileOutput node into a message flow, you must configure it (for more information, see "Configuring a message flow node" on page 1503). The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those properties that do not have a default value defined) are marked with an asterisk in that view.

Terminals and properties:
The FileOutput node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Finish File	The input terminal that accepts a message that triggers the final processing of a file.
Out	The message received on the In terminal is propagated to this terminal if the record is written successfully. The message is unchanged except for status information in the Local Environment.
End of Data	The message received on the Finish File terminal is propagated to this terminal if the file is processed successfully.
Failure	The output terminal to which the message is routed if a failure is detected when a message is propagated.

The following tables describe the node properties that you can set on a specified tab. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The FileOutput node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	FileOutput	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The FileOutput node Basic properties are described in the following table.

Property	M	C	Default	Description	`mqsiapplybaroverride` command property
Directory	No	Yes	None	Specify the output directory in which the FileOutput node places its files. Specify the directory as an absolute or relative directory path. If the directory path is relative, it is based on the directory specified in the environment variable MQSI_FILENODES_ROOT_DIRECTORY. For example: • On Windows: `C:\fileoutput` • On UNIX: `/var/fileoutput` To write files in the directory that is identified by MQSI_FILENODES_ROOT_DIRECTORY, ensure that you specify a value of `.` (a period) in this property. You can override the output directory path to be used by setting values in the current message. For more information, see the Request tab properties.	outputDirectory

Property	M	C	Default	Description	mqsiapplybaroverride command property
File name or pattern	No	Yes	None	Specify a file name pattern. This property defines the name of the file that is created by the FileOutput node. The value is either a specific file name or a character sequence (pattern) that matches a file name. Only patterns with a single wildcard character (the asterisk, *) are allowed in this property field. The file name to be used is determined in the following way: • If the file name contains no wildcard, the value of this property is the name of the file created. This value must be a valid file name on the file system or the FTP file system that hosts the broker to which the message flow is deployed. • If the file name contains a single wildcard character, the value of the element `LocalEnvironment.Wildcard.WildcardMatch` in the current message replaces the wildcard character, and the resulting value is the name of the file created. This value must be a valid file name on the file system or the FTP file system that hosts the broker to which the message flow is deployed. If the WildcardMatch value is not found, the wildcard character is replaced by an empty string. You can override the name of the file by setting values in the current message. For more information, see the Request tab properties. If the `File name or pattern` property is empty, you must override the name by using the current message. Wildcard substitution occurs only if this property is not overridden in this way. File names are passed to the file system to which the broker has access and must adhere to the conventions of these file systems. For example, file names on Windows systems are not case-sensitive; while on UNIX systems, they are.	outputFilename
Mode for writing to file	Yes	No	Stage in transit directory	Specify if the file must be staged or written to directly. Select one of the following options: • `Stage in transit and move to output directory on Finish File` • `Write directly to the output file`	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Action if file exists	Yes	No	Replace Existing File	Specify how the file is to be processed when it is complete. Select one of the following options: • Replace Existing File (the default value) specifies that if a file of the same name exists in the output directory, the new file replaces it. • Append to Existing File moves the output file to the transit directory and appends the file contents to the file. The file is moved back to the output directory when the append is finished. • Fail if File Exists specifies that a new file is created, and that if a file of the same name exists in the output directory, the new file remains in the transit directory and exception BIP3307 is produced. • Archive and Replace Existing File specifies that if any file of the same name exists in the output directory, it is moved to the archive directory before the new file is placed in the output directory. If any file of the same name exists in the archive directory, an exception is produced. • Time Stamp, Archive and Replace Existing File specifies that if a file of the same name exists in the output directory, its name is augmented with a time stamp (a character-based version of the date and time) before being moved to the archive directory. The format of the time stamp is *yyyyMMdd_HHmmss_SSSUUU*, in UTC time, where *UUU* is an additional ID to ensure that the time stamp is unique.	
Replace duplicate archive files	Yes	No	Cleared	Select the Replace duplicate archive files check box to specify that, in cases where Archive and Replace Existing File or Time Stamp, Archive and Replace Existing File is specified in Output file action, files moved to the archive directory replace files that exist there already with the same name. By default, this check box is cleared. If this check box is not selected, and there is already a file in the archive directory with the same name as a file that is to be moved there, an exception is produced, and the new file remains in the transit directory.	

The FileOutput node Request properties are described in the following table.

These properties specify the location of the data to be written, and control information that overrides the Directory and File name or pattern properties on the **Basic** tab. You can specify the properties on this tab as XPath or ESQL expressions. Content assist is available in the properties pane and also in the XPath Expression Builder, which you can open by using the **Edit** button to the right of each property.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Data location	Yes	No	`$Body`	Specify the input data location, which is the location in the input message tree that contains the record to be written to the output file. The default value is `$Body`, meaning the entire message body (`$InputRoot.Body`). When you specify this property, and the data in the message tree that it identifies is owned by a model-driven parser, such as the MRM parser or XMLNSC parser, consider the following factors. • If you are using MRM CWF format, ensure that the identified message tree exists as a message definition. If this element is defined as a global element only, exceptions BIP5180 and BIP5167 are produced. • If you are using MRM TDS format, serialization of the identified message is successful if the element is defined as a global element or message. However, if the identified field is not found as a global element or message, the following conditions apply: – If the field is a leaf field in the message tree, the field is written as self-defining. No validation occurs even if validation is enabled. – If the field is a complex element, an internal exception is generated (BIP5522), indicating that the logical type cannot be converted to a string. • If you are using MRM XML, the events are similar as for the MRM TDS format except that, if the field is a complex element, it is written as self-defining. • If you use the XMLNSC parser, no validation occurs even if validation is enabled.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Request directory property location	Yes	Yes	`$LocalEnvironment/Destination/File/Directory`	Specify the location of the value to override the `Directory` property on the **Basic** tab. If you do not specify a location, the default value is `$LocalEnvironment/Destination/File/Directory`. If you specify a location but the element is empty or missing, the `Directory` property is used. The element has a data type of CHARACTER and is an absolute or relative directory path. Use the path separator character ('/' or '\') according to the file system on which the broker runs. Trailing path separator characters are ignored. Relative directory paths are based on the value of the MQSI_FILENODES_ROOT_DIRECTORY environment variable.	requestDirectoryLocation
Request file name property location	Yes	Yes	`$LocalEnvironment/Destination/File/Name`	Specify the location of the value to override the `File name or pattern` property on the **Basic** tab. If you do not specify a location, the default value is `$LocalEnvironment/Destination/File/Name`. If you specify a location but the element is empty or missing, the `File name or pattern` property is used. The element has a data type of CHARACTER and is an explicit file name. No wildcard substitution occurs for this value.	requestNameLocation

The FileOutput node Records and Elements properties are described in the following table.

These properties specify how the FileOutput node writes the record derived from the message.

Property	M	C	Default	Description
Record definition	Yes	No	`Record is Whole File`	Specify how the records are placed in the output file. Select one of the following options: • `Record is Whole File` specifies that the file is to contain a single record. The file is finished immediately after the record is written; the FileOutput node does not wait for a message on the Finish File terminal. This value is the default. • `Record is Unmodified Data` specifies that records are accumulated in a file with neither padding or delimiters applied. The file is finished only when a message is received on the Finish File terminal. • `Record is Fixed Length Data` specifies that records are padded to a given length if necessary and accumulated in a file by concatenation. You specify this length in the `Length` property. If the record is longer than the value specified in `Length`, the node produces an exception. Use the `Padding byte` property to specify the byte to be used for padding the message to the required length. Records are added to this file until a message is received on the Finish File terminal. • `Record is Delimited Data` to specify that records are separated by a delimiter and accumulated by concatenation. The delimiter is specified by the `Delimiter`, `Custom delimiter`, and `Delimiter type` properties. Records are added to this file until a message is received on the Finish File terminal.
Length	Yes	No	`80`	Specify the length (in bytes) of records when `Record is Fixed Length Data` is specified in `Record definition`. Records that are longer than this value cause an exception to be produced. This value must be in the range 1 byte through 104857600 bytes (100 MB). The default value is 80 bytes.
Padding byte	Yes	No	`X'20'`	When `Record is Fixed Length Data` is specified in `Record definition`, use the `Padding byte` property to specify the byte to be used when padding records to the specified length if they are shorter than this length. Specify this value as two hexadecimal digits. The default value is `X'20'`.
Delimiter	Yes	No	`Broker System Line End`	Specify the delimiter to be used if you specify `Record is Delimited Data` in `Record definition`. Select one of the following options: • `Broker System Line End` specifies that a line end sequence of bytes is used as the delimiter, as appropriate for the file system on which the broker is to run. For example, on Windows systems, the delimiter is a 'carriage-return, line-feed' pair (X'0D0A'); on UNIX systems, it is a single 'line-feed' byte (X'0A'); on z/OS systems, it is a 'newline' byte (X'15'). This value is the default. • `Custom Delimiter` specifies that the explicit delimiter sequence defined in the `Custom delimiter` property is to be used to delimit records.
Custom delimiter	No	No	None	Specify the delimiter sequence of bytes to be used to delimit records when `Custom Delimiter` is specified in the `Delimiter` property. Specify this value as an even-numbered string of hexadecimal digits. The default value is `X'0A'` and the maximum length of the string is 16 bytes.

Property	M	C	Default	Description
Delimiter type	Yes	No	Postfix	If you set the Record definition property to Record is Delimited Data, use Delimiter type to specify how the delimiter is to separate records. Select one of the following options: • Postfix specifies that the delimiter is added after each record that is written. This value is the default. • Infix specifies that the delimiter is inserted only between any two adjacent records.

The FileOutput node Validation properties are described in the following table.

For a full description of these properties, see "Validation properties" on page 4169.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Validate	No	Yes	Inherit	Specify whether validation takes place. Valid values are: • None • Content and Value • Content • Inherit	validateMaster
Failure action	No	No	Exception	Specifies what happens if validation fails. You can set this property only if you set Validate to Content or Content and Value. Valid values are: • User Trace • Local Error Log • Exception • Exception List	

The FileOutput node FTP properties are described in the following table.

To transfer files to an FTP or SFTP server, select the Remote Transfer property, then set the properties described in this table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Remote transfer	No	Yes	Cleared	To transfer files to an FTP or SFTP server, select Remote Transfer, then set the other properties in this table.	fileFtp
Transfer protocol	No	Yes	FTP	This property specifies the protocol to be used for remote transfer. Valid values are: • FTP • SFTP	remoteTransferType

Property	M	C	Default	Description	mqsiapplybaroverride command property
Remote server and port	No	Yes	None	This property can have either of the following values: • The IP address or name (and, optionally, the port number) of a remote FTP or SFTP server • The name of a configurable service of type FtpServer Specify the IP address and port number of the FTP or SFTP server to be used, by using the following syntax: • *IP_address_or_URL* or • *IP_address_or_URL:port_number* If you specify the IP address in IPv6 format, ensure that you enclose it in square brackets, for example: • `[12a::13bd:24cd]` or • `[12a::13bd:24cd]:123` where `123` is the port number If you are using FTP and you do not specify a port number, 21 is assumed. If you are using SFTP and you do not specify a port number, a port number of 22 is assumed. However, if an FtpServer configurable service is defined, you can enter the name of the configurable service in this field. If a configurable service name is specified, any or all of the other remote transfer properties on the FTP tab can be overridden by the configurable service. For information about how an FtpServer configurable service definition and the properties on this tab interact, see "FtpServer configurable service properties" on page 3794. You can override this property by setting the location of the server in the local environment. For more details, see "Local environment overrides for the FileOutput node" on page 4443.	fileFtpServer
Security identity	No	Yes	None	Specify the name of a security identity that has been defined by using the **mqsisetdbparms** command. The user identifier and password that are to be used to log on to the FTP or SFTP server are obtained from this definition. The name of the definition must have the prefix `ftp::`. The value of this property is overridden by the value in the FtpServer configurable service property securityIdentity, if it is set.	fileFtpUser
Server directory	No	Yes	"."	Specify the directory on the FTP or SFTP server to which to transfer files. The default value is `.` (a period), which indicates the default directory after logon. If you specify a relative path, the directory is based on the default directory after FTP or SFTP logon. Ensure that the syntax of the path conforms to the file system standards in the FTP or SFTP server. The value of this property is overridden by the value in the remoteDirectory property of the FtpServer configurable service, if it is set.	fileFtpDirectory

Property	M	C	Default	Description	mqsiapplybaroverride command property
Transfer mode	No	Yes	Binary	Specify how files are transferred. If the file contents are not transformed, select Binary. If the file is transmitted as ASCII, select ASCII. The value of this property is overridden by the value in the FtpServer configurable service property transferMode, if it is set. This property is valid only when FTP is selected as the protocol for remote transfer. If you have specified SFTP as the protocol, the Transfer mode property is ignored and Binary encoding is used.	
Action if remote file exists	No	No	Replace File	Specify whether to create the file or append to an existing file. Select one of the following options: • Replace Existing File (the default value) specifies that if a file of the same name exists in the output directory, the new file replaces it. The file is replaced by using the FTP put verb. • Append to Existing File moves the output file to the transit directory and appends the file contents to the file. The file is transferred to the remote machine by using the FTP append verb and moved back to the output directory when the append is finished.	
Retain local file after transfer	No	No	Cleared	To retain a local copy of the file after the file transfer process has completed, select the Retain local file after transfer check box. If this check box is selected, the local copies are processed after the transfer is complete, as are other output files, as specified on the **Basic** tab. If the check box is cleared, successfully transferred files are not retained locally.	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"File name patterns" on page 1830
You can specify a file name pattern, using wildcard characters, to identify a file to be read by the FileInput and FTEInput nodes. You can also specify a file name pattern, using a single wildcard character, to name the file to be created by the FileOutput and FTEOutput nodes.

"How the broker processes files" on page 1814
The broker reads files with the FileInput, FTEInput, and FileRead nodes, and writes files with the FileOutput and FTEOutput nodes.

"How multiple file nodes share access to files in the same directory" on page 1818
WebSphere Message Broker controls access to files so that only one file node at a time can read or write to a file.

"Archiving" on page 1833
Files that are successfully processed by the FileInput node or FileOutput node can optionally be moved to the `mqsiarchive` subdirectory of the input or output directory.

"Using local environment variables with file nodes" on page 1820
You can use fields in the local environment to dynamically alter the behavior of the FileInput, FileOutput, FTEInput, and FTEOutput nodes. You can also find what values the output nodes used to process the file.

"Message flow aggregation" on page 2718
Aggregation is the generation and fan-out of related requests that are derived from a single input message, and the fan-in of the corresponding replies to produce a single aggregated reply message.

Related tasks:

"Writing a file" on page 1852
Use the FileOutput, CDOutput, and FTEOutput nodes to write files.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Configuring aggregation flows" on page 2721
Use aggregation message flows to generate and fan-out a number of related requests, fan-in the corresponding replies, and compile those replies into a single aggregated reply message, by using the AggregateControl, AggregateRequest, and AggregateReply nodes.

"Validating messages" on page 1478
The broker provides validation based on the message model for predefined messages.

Related reference:

"Recognizing file records as messages to be parsed" on page 1817
Use the FileInput, FTEInput and FileRead nodes to segment your input file into messages that are to be parsed.

"Local environment overrides for the FileOutput node"
Set values in the local environment to override the location of the remote server specified on the FileOutput node.

"FileInput node" on page 4415
Use the FileInput node to process messages that are read from files.

"FtpServer configurable service properties" on page 3794
Select the properties and values that you want to change for an existing FtpServer configurable service or to create a new service.

"Validation properties" on page 4169
You can control validation by setting properties on the Validate and Parser Options tabs for the nodes that are listed in the following table.

"FTEOutput node" on page 4466
Use the FTEOutput node to write messages to files by using the WebSphere MQ File Transfer Edition.

Local environment overrides for the FileOutput node:

Set values in the local environment to override the location of the remote server specified on the FileOutput node.

Remote server location

You can set the location of the FTP or SFTP server for the FileOutput node to use in `LocalEnvironment.Destination.File.Remote.Server`. If this field is present in the local environment, it overrides the `Remote server and port` property on the FileOutput node. Set this local environment field to the name of a configurable service, the name of a server, or a server name and port number.

The local environment override works in the same way as the node property. If a configurable service exists with the name specified, all the properties of the configurable service are used instead of the properties on the node. If a configurable service of that name does not exist, the value is used as a server name to which the node connects. The system checks for a value with the format *serverName*:*portNumber* first, then it searches for a server name on its own. When a server name is found, no other properties are overridden. This process is repeated for every message that passes through the node.

Related concepts:

"How the broker processes files" on page 1814
The broker reads files with the FileInput, FTEInput, and FileRead nodes, and writes files with the FileOutput and FTEOutput nodes.

"Local environment tree structure" on page 1056
The local environment tree is a part of the logical message tree in which you can store information while the message flow processes the message.

Related tasks:

"Writing a file to a remote FTP or SFTP server" on page 1855
Use a FileOutput node to write a file to a directory on a remote FTP or SFTP server.

"Accessing the local environment tree" on page 2463
The local environment tree has its own correlation name, LocalEnvironment, and you must use this name in all ESQL statements that refer to or set the content of this tree.

Related reference:

"FileOutput node" on page 4430
Use the FileOutput node to write messages to files.

FileRead node

Use the FileRead node to read one record, or the entire contents of a file, from within a message flow.

This topic contains the following sections:

Purpose:

You can use the FileRead node to read one record, or the entire contents of a file, from within the middle of a message flow, see "Routing or enriching a message based on the contents of a file" on page 1837. The FileRead node is like the FileInput node which reads a file from the start of a message flow, except it is driven to read the file by an incoming message.

The FileRead node is contained in the **File** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:
To enable function that becomes available in WebSphere Message Broker fix packs, use the **-f** parameter on the **mqsichangebroker** command. For more information, see "**mqsichangebroker** command" on page 3723.

By using any built-in parser, the FileRead node can parse the contents of the file and propagate the contents as a message tree. The node streams data to parsers which support this function, in an identical way to the FileInput node.

The main properties of the node specify the file and directory to read the file from. The file name is given as a pattern which can include wildcards. Both the directory and file pattern can be overridden by using fields in the local environment.

Terminals and properties:
The FileRead node input terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Finish file in	The input terminal that accepts a request to perform the finish file action, without reading any data. When a message is received by the **Finish file in** terminal, the FileRead node starts the processing that is specified by the `Action` property. When a message is received by the **Finish file in** terminal, no data is read before the action is taken.

The FileRead node output terminals are described in the following table.

Terminal	Description
Failure	The output terminal to which a message is routed if a failure is detected when the message is propagated.
Out	The output terminal to which a message is routed if it is successfully retrieved from an external resource. If no errors occur in the input node, a message received from an external resource is always sent to the **Out** terminal first.
No match	The message received on the **No match** terminal is propagated to this terminal if the file does not exist on the file system or it does exist but no record which matches the filter expression can be found. If the terminal is not connected, the message is not used.
Finish file out	A message arriving on the **Finish file in** terminal is propagated to the **Finish file out** terminal with the content unchanged, but the local environment is updated with details of the action the node has taken.

The following tables describe the node properties that you can set on a specified tab. The column headed M indicates whether the property is *mandatory* (marked in the toolkit with an asterisk if you must enter a value when no default is defined). The column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

When the contents of the file have been successfully propagated down the flow, the file is deleted from the file system. No other file node can access the file when the read node starts reading data from it. If a file does not exist which matches the

pattern, then the original message is propagated to the 'No match' terminal. If the terminal is not connected then an exception is thrown.

At the end of processing, it is possible to configure the node not to delete the file. In this mode, any other file read node can also access the file if running in the same browse only mode.

Description properties

Property	M	C	Default	Description
Node name	No	No	`FileRead`	The name of the node.
Long Description	No	No	None	Text that describes the purpose of the node in the message flow.
Short Description	No	No	None	A brief description of the node.

Basic properties

Property	M	C	Default	Description	**mqsiapplybaroverride command property**
Input directory	No	Yes	None	Absolute path of the input directory in the form used by the file system of the broker. For example, on Windows systems, the directory path starts with the drive letter prefix (such as `C:`). Alternatively, the path can be relative to the file nodes root directory (which can be overridden with the same environment variable as used for the File input and output nodes).	inputDirectory
File name or pattern	No	Yes	*	A file name, or a character sequence (pattern) that matches a file name. A pattern contains at least one of the following wildcard characters: • Asterisk (*), representing any sequence of zero or more characters • Question mark (?), representing any single character If more than one file matches the pattern, and the **`Action`** property is set to `No action - do nothing to the file`, an exception is thrown. Select the `Use environment wildcard` option if you want to use part of the file name from the input node in this field.	filenamePattern
Substitute wildcard match	No	No	False	If you select this option, the file pattern is no longer a regular expression. Instead, it must contain only one *. The * is replaced by the **$LocalEnvironment/Wildcard/WildcardMatch** field. This location is normally populated by a previous node, such as the FileInput node.	substituteWildcardMatch

Property	M	C	Default	Description	mqsiapplybaroverride command property
Action	Yes	No	No action	The action performed to the file after either the end of the file is reached, or a message is received by the **Finish file in** terminal. This action occurs in any of the following circumstances: 1. When a file is read, if `Record detection` is set to `Whole File`. 2. When the record read is the last record in the file, if `Record detection` is set to anything other than `Whole File`. 3. When a message is received by **Finish file in**. In this case no data is read before the action is performed. Valid actions are: • `No action` - do nothing to the file. • `Delete` - delete the file. • `Archive` - move the file to an archive directory. • `Archive with timestamp` - move the file to the archive directory and add a timestamp. By default if move to `Archive` is selected the file is moved to the `mqsiarchive` sub directory, but this default can be overridden by setting a local environment variable. When the local environment override for the archive file name is set and the finish file action on the node is set to `Archive with timestamp`, the file name is `timestamp_nameYouSpecified`. Where `timestamp` is the date and time the file is archived and `nameYouSpecified` is the name you give to the file. When the local environment override for the archive name is set and the finish file action on the node is set to `Delete`, the file is deleted. When the local environment override for the archive name is set and the finish file action on the node is set to `No action`, no action is applied to the file.	
Replace duplicate archives	Yes	No	False	By default, an error occurs if a file is archived and a file exists with the same name. Set this property to true to ignore the error and replace the archive file.	

Request properties

Property	M	C	Default	Description	mqsiapplybaroverride command property
Request directory property location	No	No	`$LocalEnvironment/Destination/File/Directory`	The message element location containing the name of the input directory.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Request file name property location	No	No	`$LocalEnvironment/Destination/File/Name`	The message element location containing the name of the input file pattern.	
Offset property location	No	No	`$LocalEnvironment/Destination/File/Offset`	The message element location containing the offset to start searching for records from.	
Length property location	No	No	`$LocalEnvironment/Destination/File/Length`	The message element location containing the length of the record to read if using fixed-length record detection.	

Result properties

Property	M	C	Default	Description	mqsiapplybaroverride command property
Result data location	Yes	No	`$ResultRoot`	The location in the message retrieved from the file to copy to the Output data location field in the outgoing message.	
Output data location	Yes	No	`$OutputRoot`	The location in the outgoing message where the record read from the file is to be copied to. The part of the record that is copied to the output data location is defined in the result data location. By default, the entire record is copied.	
Copy local environment	No	No	Selected	This property specifies whether the local environment is copied to the output message. • If `Copy local environment` is selected, a new copy of the local environment is created in the tree, and it is populated with the contents of the local environment from the preceding node. Therefore, if a node changes the local environment, the upstream nodes are not affected by those changes because they have their own copies. This value is the default. • If `Copy local environment` is cleared, the node does not generate its own copy of the local environment, but uses the local environment that is passed to it by the preceding node. Therefore, if a node changes the local environment, the changes are reflected by the upstream nodes.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Record selection expression	No	No	true()	The expression used to select the correct record from the file. The expression is evaluated for each record in the file until one is found that evaluates to true. That record is then propagated to the **Out** terminal. The expression can be set to any valid XPath expression that returns a Boolean value. The expression is not used when `Whole File` is selected as the `Record Detection` option. The following correlation names are in scope to use in the expression: • `InputRoot` • `InputLocalEnvironment` • `OutputRoot` • `OutputLocalEnvironment` • `ResultRoot`	

Input Message Parsing properties

Property	M	C	Default	Description	mqsiapplybaroverride command property
Message Domain	No	No		The domain that is used to parse the incoming message.	
Message Set	No	No		The name or identifier of the message set in which the incoming message is defined. If you set this property, and then update the project dependencies to remove this message set reference, a warning is issued. Either update the `Message Set` property, or restore the reference to this message set project.	
Message Type	No	No		The name of the incoming message.	
Message Format	No	No		The name of the physical format of the incoming message.	
Message coded character set ID	No	Yes	`Broker System Default`	The ID of the coded character set used to interpret bytes of the file being read.	messageCodedCharSetIdProperty
Message encoding	No	Yes	`Broker System Determined`	The encoding scheme for numbers and large characters used to interpret bytes of the file being read. Valid values are `Broker System Determined` or a numeric encoding value. For more information about encoding, see "Data conversion" on page 1151.	messageEncodingProperty

Parser Options properties

Property	M	C	Default	Description
Parse timing	No	No	`On Demand`	This property controls when an input message is parsed. Valid values are: • `On Demand` • `Immediate` • `Complete` For a full description of this property, see "Parsing on demand" on page 4173.
Build tree using XML schema data types	No	No	Cleared	This property controls whether the syntax elements in the message tree have data types taken from the XML schema.
Use XMLNSC compact parser for XMLNS domain	No	No	Cleared	This property controls whether the XMLNSC Compact Parser is used for messages in the XMLNS Domain. If you set this property, the message data is shown under XMLNSC in nodes that are connected to the output terminal when the input MQRFH2 header or Input Message Parsing property, `Message Domain` is XMLNS.
Retain mixed content	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters mixed text in an input message. If you select the check box, elements are created for mixed text. If you clear the check box, mixed text is ignored and no elements are created.
Retain comments	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters comments in an input message. If you select the check box, elements are created for comments. If you clear the check box, comments are ignored and no elements are created.
Retain processing instructions	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters processing instructions in an input message. If you select the check box, elements are created for processing instructions. If you clear the check box, processing instructions are ignored and no elements are created.
Opaque elements	No	No	Blank	This property is used to specify a list of elements in the input message that are to be opaquely parsed by the XMLNSC parser. Opaque parsing is performed only if validation is not enabled (that is, if `Validate` is `None`); entries that are specified in `Opaque Elements` are ignored if validation is enabled.

Records and Elements properties

Property	M	C	Default	Description
Record detection	Yes	No	`Whole File`	The mechanism used to identify records in the input file. Valid options are: • `Whole File` • `Fixed Length` • `Delimited` • `Parsed Record Sequence`
Length	Yes	No	`80`	The length of each record, in bytes, when `Fixed Length` record detection is selected.

Property	M	C	Default	Description
Delimiter	Yes	No	DOS or UNIX Line End	The type of delimiter bytes that separate, or end, each record when Delimited record detection is selected. Valid options are: • DOS or UNIX Line End • Custom Delimiter You can set the delimiter only when the Record detection property is set to Delimited.
Custom delimiter	No	Yes		The delimiter bytes, expressed in hexadecimal, when Delimited record detection and Custom Delimiter are selected. This property is mandatory only if the Delimiter property is set to Custom Delimiter.
Delimiter type	Yes	No	Postfix	The position of the delimiter when Delimited record detection is selected. Valid options are: • Postfix • Infix This property is ignored unless the Delimiter property is set to Custom Delimiter.

Validation properties

For a full description of these properties, see "Validation properties" on page 4169.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Validate	No	Yes	None	This property controls whether validation takes place. Valid values are: • None • Content and Value • Content	validateMaster
Failure action	No	No	Exception	This property controls what happens if validation fails. Valid values are: • User Trace • Local Error Log • Exception • Exception List	

Related concepts:

"Routing or enriching a message based on the contents of a file" on page 1837
The FileRead node can route or enrich messages based on the contents of the file.

"Using local environment variables with file nodes" on page 1820
You can use fields in the local environment to dynamically alter the behavior of the FileInput, FileOutput, FTEInput, and FTEOutput nodes. You can also find what values the output nodes used to process the file.

"File name patterns" on page 1830
You can specify a file name pattern, using wildcard characters, to identify a file to be read by the FileInput and FTEInput nodes. You can also specify a file name pattern, using a single wildcard character, to name the file to be created by the FileOutput and FTEOutput nodes.

Related tasks:

"Controlling the functional level of WebSphere Message Broker" on page 51
You can control the functional level of your broker to enable new functionality added in WebSphere Message Broker fix packs.

Filter node

Use the Filter node to route a message according to message content.

This topic contains the following sections:

Purpose:
Create a filter expression in ESQL to define the route that the message is to take. You can include elements of the input message or message properties in the filter expression, and you can use data that is held in an external database to complete the expression. The output terminal to which the message is routed depends on whether the expression evaluates to true, false, or unknown.

Connect the terminals that cover all situations that could result from the filter; if the node propagates the message to a terminal that is not connected, the message is discarded even if it is transactional.

The Filter node accepts ESQL statements in the same way as the Compute and Database nodes. The last statement that is executed must be a `RETURN <expression>` statement, whose expression evaluates to a Boolean value. This Boolean value determines the terminal to which the message is routed. In many cases, the routing algorithm is a simple comparison of message field values. The comparison is described by the expression and the RETURN statement is the only statement. If you code RETURN without an expression (`RETURN;`) or with a null expression, the node propagates the message to the Unknown terminal.

If your message flow requires more complex routing options, use the RouteToLabel and Label nodes.

The Filter node is contained in the **Routing** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:
Look at the following samples for examples of how to use this node:
- Airline Reservations
- Scribble
- Error Handler
- Large Messaging

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Consider a situation in which you have produced an online test with ten multiple choice questions. Each message coming in has a candidate name and address

followed by a series of answers. Each answer is checked, and if it is correct, the field SCORE is incremented by one. When all the answers have been checked, the field SCORE is tested to see if it is greater than five. If it is, the Filter node propagates the message to the flow that handles successful candidate input; otherwise, the message is filtered into the rejection process, and a rejection message is created.

Terminals and properties:
When you have put an instance of the Filter node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The Filter node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node
Failure	The output terminal to which the message is routed if a failure is detected during the computation
Unknown	The output terminal to which the message is routed if the specified filter expression evaluates to `unknown` or a null value
False	The output terminal to which the message is routed if the specified filter expression evaluates to `false`
True	The output terminal to which the message is routed if the specified filter expression evaluates to `true`

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default value is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Filter node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type	The name of the node.
Short Description	No	No		A brief description of the node
Long Description	No	No		Text that describes the purpose of the node in the message flow

The Filter node Basic properties are described in the following table.

Property	M	C	Default	Description	**mqsiapplybaroverride command property**
Data Source	No	Yes		The ODBC data source name of the database that contains the tables to which you refer in the ESQL that is associated with this node (identified by the `Filter Expression` property). This name identifies the appropriate database on the system on which this message flow is to execute. The broker connects to this database with user ID and password information that you have specified on the **mqsicreatebroker**, **mqsichangebroker**, or **mqsisetdbparms** command. z/OS On z/OS systems, the broker uses the broker started task ID, or the user ID and password that are specified on the **mqsisetdbparms** command JCL, BIPSDBP, in the customization data set <hlq>.SBIPPROC. If the ESQL that is associated with this node includes a PASSTHRU statement or SELECT function and a database reference, you must specify a value for the `Data Source` property.	dataSource
Transaction	Yes	No	Automatic	The transaction mode for the node. The values are: • Automatic (the default). The message flow, of which the Filter node is a part, is committed if it is successful. That is, the actions that you define in the ESQL module are performed and the message continues through the message flow. If the message flow fails, it is rolled back. Therefore, if you choose Automatic, the ability to commit or roll back the action of the Filter node on the database depends on the success or failure of the entire message flow. • Commit. To commit any uncommitted actions that are performed in this message flow on the database that is connected to this node, irrespective of the success or failure of the message flow as a whole, select Commit. The changes to the database are committed even if the message flow itself fails.	

Property	M	C	Default	Description	mqsiapplybaroverride command property

Filter Expression	No	No	Filter	The name of the module within the ESQL resource (file) that contains the statements to execute against the message that is received in the node The ESQL file, which by default has the name `<message_flow_name>.esql`, contains ESQL for every node in the message flow that requires it. Each portion of code that is related to a specific node is known as a module. If you want the module name to include one or more spaces, enclose it in double quotation marks in the `Filter Expression` property.
				
				Code ESQL statements to customize the behavior of the Filter node in an ESQL file that is associated with the message flow in which you have included this instance of the Filter node.
				
				If an ESQL file does not already exist for this message flow, double-click the Filter node, or right-click the node and click **Open ESQL** to create and open a new ESQL file in the ESQL editor view.
				
				If the file exists already, click **Browse** beside the `Filter Expression` property to display the Module Selection dialog box, which lists the available Filter node modules defined in the ESQL files that can be accessed by this message flow (ESQL files can be defined in other, dependent, projects). Select the appropriate module and click **OK**; if no suitable modules are available, the list is empty.
				
				If the module that you specify does not exist, that module is created for you, and the editor displays it. If the file and the module exist already, the editor highlights the correct module.
				
				If a module skeleton is created for this node in a new or existing ESQL file, it consists of the following ESQL. The default module name is shown in this example:
				`CREATE FILTER MODULE <flow_name>_Filter`
				`  CREATE FUNCTION Main() RETURNS BOOLEAN`
				`  BEGIN`
				`  RETURN TRUE;`
				`  END;`
				`END MODULE;`
				
				If you create your own ESQL module, you must create this skeleton exactly. You can update the default name, but ensure that the name that you specify matches the name of the corresponding node property `Filter Expression`.
				
				To customize this node, add your own ESQL after the BEGIN statement, and before the RETURN statement. If the expression on the RETURN statement is not TRUE or FALSE, its value is resolved to determine the terminal to which the message is propagated. If the expression resolves to a null value, or you code	

Property	M	C	Default	Description	mqsiapplybaroverride command property
				You can use all the ESQL statements including SET, WHILE, DECLARE, and IF in this module, but (unlike the Compute node) the Filter node propagates the message that it receives at its input terminal to its output terminal unchanged. Therefore, in the Filter node, like the Database node, you have only one message to which to refer. The ESQL correlation names that you use in a Filter node are different from those used for a Compute node. For more information about correlation names refer to the related links. You cannot modify any part of any message, so the assignment statement (the SET statement, not the SET clause of the INSERT statement) can assign values only to temporary variables. The scope of actions that you can take with an assignment statement is therefore limited.	
Treat warnings as errors	Yes	No	Cleared	For database warning messages to be treated as errors, and to propagate the output message from the node to the Failure terminal, select `Treat warnings as errors`. The check box is cleared initially. When you select the check box, the node handles all positive return codes from the database as errors and generates exceptions in the same way as it does for the negative, or more serious, errors. If you do not select the check box, the node treats warnings as normal return codes and does not raise any exceptions. The most significant warning raised is `not found`, which can be handled safely as a normal return code in most circumstances.	
Throw exception on database error	Yes	No	Selected	For the broker to generate an exception when a database error is detected, select `Throw exception on database error`. The check box is selected initially. If you clear the check box, you must include ESQL to check for any database error that might be returned after each database call that you make (you can use SQLCODE and SQLSTATE to do this). If an error has occurred, you must handle the error in the message flow to ensure the integrity of the broker and the database; the error is ignored if you do not handle it through your own processing because you have chosen not to invoke the default error handling by the broker. For example, you can include the ESQL THROW statement to throw an exception in this node, or you can use the Throw node to generate your own exception at a later point.	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Client application programming interfaces" on page 1038
You can configure the nodes in your message flows to customize the behavior of those nodes by using one or more of the supported programming interfaces.

"Correlation names" on page 1069
A correlation name is a field reference that identifies a well-defined starting point in the logical message tree and is used in field references to describe a standard part of the tree format.

Related tasks:

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

"Configuring transactionality for message flows" on page 1290
A message flow runs in a single transaction, which is started when data is received by an input node, and can be committed or rolled back when all processing has completed.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Writing ESQL" on page 2413
How you can use ESQL to customize nodes.

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

Related reference:

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Database node" on page 4354
Use the Database node to interact with a database in the specified ODBC data source.

"Label node" on page 4569
Use the Label node to process a message that is propagated by a RouteToLabel node to dynamically determine the route that the message takes through the message flow.

"Route node" on page 4669
Use the Route node to direct messages that meet certain criteria down different paths of a message flow.

"RouteToLabel node" on page 4673
Use the RouteToLabel node in combination with one or more Label nodes to dynamically determine the route that a message takes through the message flow, based on its content.

"RETURN statement" on page 5155
The RETURN statement ends processing. What happens next depends on the programming context in which the RETURN statement is issued.

FlowOrder node

Use the FlowOrder node to control the order in which a message is processed by a message flow.

This topic contains the following sections:

Purpose:
The FlowOrder node propagates the input message to the first output terminal, and the sequence of nodes that is connected to this terminal processes the message. When that message processing is complete, control returns to the FlowOrder node. If the message processing completes successfully, the FlowOrder node propagates the input message to the second output terminal, and the sequence of nodes that is connected to this terminal processes the message.

The message that is propagated through the second output terminal is the input message. It is not modified in any way by the FlowOrder node. If a compute node that is connected to the first terminal modifies the InputRoot, for example by using references, these modifications are made visible in the message that is propagated to the Second terminal.

You can include this node in a message flow at any point where the order of execution of subsequent nodes is important.

If you connect multiple nodes to the first output terminal, the second output terminal, or both, the order in which the multiple connections on each terminal are processed is random and unpredictable. However, the message is propagated to all target nodes that are connected to the first output terminal, which must all complete successfully, before the message is propagated to any node that is connected to the second output terminal.

Your message flow performance can benefit from including the FlowOrder node in a situation where one sequence of processing that is required for a message is significantly shorter than another sequence of processing. If you connect the shorter sequence to the first terminal, any failure is identified quickly and prevents execution of the second longer sequence of processing.

The FlowOrder node is contained in the **Construction** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:
For an example of using this node, assume that your company receives orders from customers using the Internet. When the order is received, it is processed by nodes that are connected to the first terminal of a FlowOrder node to debit the stock level in your database and raise an invoice. A check is made to see whether the customer has indicated that his details can be sent to other suppliers. If the customer has indicated that he does not want this information to be divulged, this check fails and no further processing occurs. If the customer is happy for you to share his details with other companies (that is, the test is successful), the input message is propagated to the second terminal so that the customer's details can be added to the mailing list.

Connecting the terminals:
The FlowOrder node has no configurable properties that affects its operation. You determine how it operates by connecting the first and second output terminals to subsequent nodes in your message flow.

1. Connect the First terminal to the first node in the sequence of nodes that provide the first phase of processing this message. This sequence can contain one or more nodes that perform any valid processing. The sequence of nodes can optionally conclude with an output node.
2. Connect the Second terminal to the first node in the sequence of nodes that provide the second phase of processing this message. This sequence can contain one or more nodes that perform any valid processing. The sequence of nodes can optionally conclude with an output node.

   The message that is propagated through the Second terminal is identical to that propagated through the First terminal. Any changes that you have introduced as a result of the first phase of processing are ignored by this node.

   If the first phase of processing fails, the FlowOrder node does not regain control and does not propagate the message through the Second terminal.

Terminals and properties:
When you have put an instance of the FlowOrder node into a message flow, you can configure it. For more information, see “Configuring a message flow node” on page 1503. The properties of the node are displayed in the Properties view.

The FlowOrder node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the message is routed if a failure is detected during the computation.

Terminal	Description
First	The output terminal to which the input message is routed in the first instance.
Second	The output terminal to which the input message is routed in the second instance. The message is routed to this terminal only if routing to First is successful.

The following table describes the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The FlowOrder node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	FlowOrder	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

Related reference:

"Label node" on page 4569
Use the Label node to process a message that is propagated by a RouteToLabel node to dynamically determine the route that the message takes through the message flow.

"RouteToLabel node" on page 4673
Use the RouteToLabel node in combination with one or more Label nodes to dynamically determine the route that a message takes through the message flow, based on its content.

FTEInput node

Use the FTEInput node to receive files using the WebSphere MQ File Transfer Edition.

This topic contains the following sections:
- "Purpose"
- "Using this node in a message flow"
- "Terminals and properties"

Purpose:
You can use the FTEInput node to extend WebSphere Message Broker Version 7.0 support for file processing through its integration with WebSphere MQ File Transfer Edition. WebSphere MQ File Transfer Edition is a managed file transfer product that uses WebSphere MQ as the transport.

The FTEInput node is contained in the **File** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:
You can use the FTEInput node in any flow that is designed to accept files from a WebSphere MQ File Transfer Edition network. Look at the following sample to see how to use this node:
- Managed File Transfer

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Terminals and properties:
The FTEInput node terminals are described in the following table.

Terminal	Description
Failure	The output terminal to which a message is routed if an error occurs before a message is propagated to the Out terminal. Messages propagated to this terminal are not validated, even if you have specified, using the `Validate` property, that validation is to take place.
Out	The output terminal to which a message is routed if it has been successfully extracted from the input file. If no errors occur within the input node, a message received from an external resource is always sent to the Out terminal first.
End of Data	The output terminal to which the End of Data message is routed after all the messages in a file have been processed. The End of Data message flow transaction is initiated only if this terminal is attached.
Catch	The output terminal to which the message is routed if an exception is thrown downstream and caught by this node. Exceptions are caught only if this terminal is attached.

The following tables describe the node properties that you can set on a specified tab. The column headed M indicates whether the property is *mandatory* (marked in

the toolkit with an asterisk if you must enter a value when no default is defined). The column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

When the FTEInput node propagates a message, it stores information about it in the LocalEnvironment.FTE and LocalEnvironment.FTE.Transfer message trees. If the input file is empty, an empty message is propagated (assuming that it is valid). If you specify a file name pattern that contains wildcard characters in the `File name filter` property, the FTEInput node copies the characters in the file name matched by wildcards, together with any intermediate characters, to the LocalEnvironment.Wildcard.WildcardMatch message tree. See "Using local environment variables with file nodes" on page 1820 for more information.

Description properties

Property	M	C	Default	Description
Node name	No	No	`FTE Input`	The name of the node.
Short Description	No	No	None	A brief description of the node.
Long Description	No	No	None	Text that describes the purpose of the node in the message flow.

Basic properties

Property	M	C	Default	Description	**mqsiapplybaroverride command property**
Directory filter	No	Yes	None	The WebSphere MQ File Transfer Edition destination directory from which input files are processed.	inputDirectory
File name filter	Yes	Yes	*	A file name or string containing optional wildcard characters (* or ?) identifying the file or files to process from the Input directory.	filenamePattern
Action on successful processing	Yes	No	`No Action`	The action the node takes on the file after successfully processing the contents. Valid options are: • `No Action` • `Delete` • `Add Time Stamp`	

Input Message Parsing properties

Property	M	C	Default	Description	**mqsiapplybaroverride command property**
Message Domain	No	No	None	The domain that is used to parse the incoming message.	
Message Set	No	No	None	The name or identifier of the message set in which the incoming message is defined. If you set this property, and then update the project dependencies to remove this message set reference, a warning is issued. Either update the `Message Set` property, or restore the reference to this message set project.	
Message Type	No	No	None	The name of the incoming message.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Message Format	No	No	None	The name of the physical format of the incoming message.	
Message coded character set ID	Yes	Yes	`Broker System Default`	The ID of the coded character set used to interpret bytes of the file being read.	messageCodedCharSetIdProperty
Message encoding	Yes	Yes	`Broker System Determined`	The encoding scheme for numbers used to interpret bytes of the file being read. Valid values are `Broker System Determined` or a numeric encoding value. For more information about encoding, see "Data conversion" on page 1151.	messageEncodingProperty

Parser Options properties

Property	M	C	Default	Description
Parse timing	No	No	`On Demand`	Specifies when an input message is parsed. Valid values are: • `On Demand` • `Immediate` • `Complete` For a full description of this property, see "Parsing on demand" on page 4173.
Build tree using XML schema data types	No	No	Cleared	Specifies whether the syntax elements in the message tree have data types taken from the XML schema.
Use XMLNSC compact parser for XMLNS domain	No	No	Cleared	Specifies whether the XMLNSC Compact Parser is used for messages in the XMLNS Domain. If you set this property, the message data appears under XMLNSC in nodes that are connected to the output terminal when either of the following items is XMLNS: • The input MQRFH2 header. • The Input Message Parsing property, Message Domain.
Retain mixed content	No	No	Cleared	Specifies whether the XMLNSC parser creates elements in the message tree for mixed text in an input message. If you select the check box, elements are created for mixed text. If you clear the check box, mixed text is ignored and no elements are created.
Retain comments	No	No	Cleared	Specifies whether the XMLNSC parser creates elements in the message tree for comments in an input message. If you select the check box, elements are created for comments. If you clear the check box, comments are ignored and no elements are created.
Retain processing instructions	No	No	Cleared	Specifies whether the XMLNSC parser creates elements in the message tree for processing instructions in an input message. If you select the check box, elements are created for processing instructions. If you clear the check box, processing instructions are ignored and no elements are created.
Opaque elements	No	No	Blank	Specifies a list of elements in the input message that are to be opaquely parsed by the XMLNSC parser. Opaque parsing is performed only if validation is not enabled (that is, if `Validate` is `None`); entries that are specified in `Opaque Elements` are ignored if validation is enabled.

Retry properties

Property	M	C	Default	Description	mqsiapplybaroverride command property
Retry mechanism	Yes	No	Failure	How the node handles a flow failure. Valid options are: • Failure • Short retry • Short and long retry	
Retry threshold	Yes	Yes	0	The number of times to try the flow transaction again when the Retry mechanism property value is Short retry.	retryThreshold
Short retry interval (seconds)	No	Yes	0	The interval, in seconds, between each retry if the Retry threshold property is not zero.	shortRetryInterval
Long retry interval (seconds)	No	Yes	300	The interval between retries, if the Retry mechanism property is Short and long retry and the retry threshold has been exhausted.	longRetryInterval
Action on failing file	Yes	Yes	No Action	The action that the node takes with the input file if all attempts to process the contents of the input file fail. Valid options are: • No Action • Delete • Add Time Stamp	

Records and Elements properties

Property	M	C	Default	Description
Record detection	Yes	No	Whole File	The mechanism used to identify records in the input file. Valid options are: • Whole File • Fixed Length • Delimited • Parsed Record Sequence
Length	Yes	No	80	The length of each record, in bytes, when Fixed Length record detection is selected.
Delimiter	Yes	No	DOS or UNIX Line End	The type of delimiter bytes that separate, or end, each record when Delimited record detection is selected. Valid options are: • DOS or UNIX Line End • Custom Delimiter
Custom delimiter	No	No	None	The delimiter bytes, expressed in hexadecimal, when Delimited record detection and Custom Delimiter are selected. This property is mandatory only if the Delimiter property is set to Custom Delimiter.
Delimiter type	Yes	No	Postfix	The position of the delimiter when Delimited record detection is selected. Valid options are: • Postfix • Infix This property is ignored unless the Delimiter property is set to Custom Delimiter.

Validation properties

For a full description of these properties, see "Validation properties" on page 4169.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Validate	No	Yes	None	This property controls whether validation takes place. Valid values are: • None • Content and Value • Content	validateMaster
Failure action	No	No	Exception	This property controls what happens if validation fails. Valid values are: • User Trace • Local Error Log • Exception • Exception List	

Transactions properties

Property	M	C	Default	Description
Transaction mode	No	No	No	The transaction mode on this input node determines whether the rest of the nodes in the flow are executed under sync point. Valid options are: • Yes • No

Instances properties.

For a full description of these properties, see "Configurable message flow properties" on page 4020.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Additional instances pool	No	Yes	Use Pool Associated with Message Flow	The pool from which additional instances are obtained. • If you select Use Pool Associated with Message Flow, additional instances are obtained from the message flow pool. • If you select Use Pool Associated with Node, additional instances are allocated from the additional instances of the node based on the number specified in the Additional instances property.	componentLevel
Additional instances	No	Yes	0	The number of additional instances that the node can start if the Additional instances pool property is set to Use Pool Associated with Node.	additionalInstances

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Managed file transfers using WebSphere MQ File Transfer Edition" on page 1869
Transfer files, with file transfer metadata, in a timely and reliable manner.

"Using local environment variables with file nodes" on page 1820
You can use fields in the local environment to dynamically alter the behavior of the FileInput, FileOutput, FTEInput, and FTEOutput nodes. You can also find what values the output nodes used to process the file.

"File name patterns" on page 1830
You can specify a file name pattern, using wildcard characters, to identify a file to be read by the FileInput and FTEInput nodes. You can also specify a file name pattern, using a single wildcard character, to name the file to be created by the FileOutput and FTEOutput nodes.

Related tasks:

"Sending a file by WebSphere MQ File Transfer Edition" on page 1859
Send files to an existing WebSphere MQ File Transfer Edition network.

"Receiving a file by WebSphere MQ File Transfer Edition" on page 1845
Use the FTEInput node to receive files from an existing WebSphere MQ File Transfer Edition network.

Related reference:

"FTEOutput node"
Use the FTEOutput node to write messages to files by using the WebSphere MQ File Transfer Edition.

FTEOutput node

Use the FTEOutput node to write messages to files by using the WebSphere MQ File Transfer Edition.

This topic contains the following sections:

Purpose:

You can use the FTEOutput node to extend WebSphere Message Broker Version 7.0 support for file processing through its integration with WebSphere MQ File Transfer Edition. WebSphere MQ File Transfer Edition is a managed file transfer product that uses WebSphere MQ as the transport.

The FTEOutput node is contained in the **File** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:
You can use the FTEOutput node in any flow that is designed to send a file across a WebSphere MQ File Transfer Edition network. Look at the following sample to see how to use this node:
- Managed File Transfer

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Terminals and properties:
The FTEOutput node terminals are described in the following table:

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Finish File	The input terminal that accepts a message that triggers the final processing of a file.
Out	The message received on the In terminal is propagated to this terminal if the record is written successfully. The message is unchanged except for status information in the Local Environment.
End of Data	The message received on the Finish File terminal is propagated to this terminal if the file is processed successfully.
Failure	The output terminal to which the message is routed if a failure is detected when a message is propagated.

The following tables describe the node properties that you can set on a specified tab. The column headed M indicates whether the property is *mandatory* (marked in the toolkit with an asterisk if you must enter a value when no default is defined). The column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

When the FTEOutput node propagates a message, either to the Out terminal or to the End of Data terminal, it stores information about it in the LocalEnvironment.WrittenDestination.FTE message tree. You can override the Destination agent, Destination queue manager, Job name, Destination file directory, Destination file name, and Overwrite files on destination system properties using additions to the LocalEnvironment.Destination.FTE message tree. See "Using local environment variables with file nodes" on page 1820 for more information.

Description properties

Property	M	C	Default	Description
Node name	No	No	`FTE Output`	The name of the node.
Short Description	No	No	None	A brief description of the node.
Long Description	No	No	None	Text that describes the purpose of the node in the message flow.

Basic properties

Property	M	C	Default	Description	mqsiapplybaroverride command property
Job name	No	Yes	None	The name for the transfer that appears in the transfer logs and the metadata.	jobName
Destination agent	No	Yes	None	The name of the destination agent to send the file to. If Destination agent is not specified, and not overridden using the LocalEnvironment tree, it will default to the agent embedded in the Execution Group.	destinationAgent
Destination queue manager	No	Yes	None	The name of the destination queue manager to send the file to.	destinationQMgr
Destination file directory	No	Yes	None	The remote directory that the destination agent writes the file to.	destinationDirectory
Destination file name	Yes	Yes	None	The specific file name or a pattern containing a single wildcard that defines the name of the file to be created by the destination agent.	destinationFileName
Mode	No	Yes	`Binary transfer (no conversion)`	The mode to transfer the file in. Valid values are: • `Binary transfer (no conversion)` • `Text transfer (ASCII/EBCDIC and CR/LF automated)`	transferMode
Disable computation of MD5 check sum	Yes	Yes	`FALSE`	Specifies whether the computation of the MD5 check sum on the transferred file is disabled.	checkSumDisabled
Overwrite files on destination system	Yes	Yes	`FALSE`	Specifies whether files on the destination system can be overwritten when the destination agent moves files of the same name there. If the destination agent fails to overwrite the file, the transfer fails and the transfer logs report the failure. The FTEOutput node does not throw or log any errors.	overwriteDestination

Request properties

Property	M	C	Default	Description
Data location	No	No	`$Body`	The location in the input message tree that contains the record to be written to the output file. The default value, `$Body`, means the entire message.

Records and Elements properties

Property	M	C	Default	Description
Record definition	Yes	No	`Record is Whole File`	Specifies how records are placed in the output file. Valid options are: • `Record is Whole File` • `Record is Unmodified Data` • `Record is Fixed Length Data` • `Record is Delimited Data`
Length (bytes)	Yes	No	`80`	The required length of the output record. The property is available only when `Record is Fixed Length Data` is specified in `Record definition`.
Padding byte (hexadecimal)	Yes	No	`20`	The 2-digit hexadecimal byte to be used to pad short messages. The property is available only when `Record is Fixed Length Data` is specified in `Record definition`.

Property	M	C	Default	Description
Delimiter	Yes	No	`Broker System Line End`	The delimiter to be used. The property is available only when `Record is Delimited Data` is specified in `Record definition`. Valid options are: • `Broker System Line End` • `Custom Delimiter (hexadecimal)`
Custom delimiter (hexadecimal)	Yes	No	None	The delimiter byte sequence to be used. The property is available only when `Record is Delimited Data` is specified in the `Record definition` property, and `Custom Delimiter (hexadecimal)` is specified in the `Delimiter` property.
Delimiter type	Yes	No	`Postfix`	This property specifies how delimiters are to be inserted between records. The property is available only when `Record is Delimited Data` is specified in `Record definition`. Valid options are: • `Postfix` • `Infix`

Validation properties

For a full description of these properties, see "Validation properties" on page 4169.

Property	M	C	Default	Description	**`mqsiapplybaroverride` command property**
Validate	Yes	Yes	`Inherit`	This property controls whether validation takes place. Valid values are: • `None` • `Content and Value` • `Content` • `Inherit`	validateMaster
Failure action	Yes	No	`Exception`	This property controls what happens if validation fails. The property is available only if you set `Validate` to `Content` or `Content and Value`. Valid values are: • `User Trace` • `Local Error Log` • `Exception` • `Exception List`	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Managed file transfers using WebSphere MQ File Transfer Edition" on page 1869
Transfer files, with file transfer metadata, in a timely and reliable manner.

"Local environment tree structure" on page 1056
The local environment tree is a part of the logical message tree in which you can

store information while the message flow processes the message.

"Using local environment variables with file nodes" on page 1820
You can use fields in the local environment to dynamically alter the behavior of the FileInput, FileOutput, FTEInput, and FTEOutput nodes. You can also find what values the output nodes used to process the file.

Related tasks:

"Sending a file by WebSphere MQ File Transfer Edition" on page 1859
Send files to an existing WebSphere MQ File Transfer Edition network.

"Receiving a file by WebSphere MQ File Transfer Edition" on page 1845
Use the FTEInput node to receive files from an existing WebSphere MQ File Transfer Edition network.

Related reference:

"FTEInput node" on page 4461
Use the FTEInput node to receive files using the WebSphere MQ File Transfer Edition.

HTTPHeader node

Use the HTTPHeader node to add, modify, or delete HTTP headers such as HTTPInput, HTTPResponse, HTTPRequest and HTTPReply.

This topic contains the following sections:
- "Purpose"
- "Using this node in a message flow"
- "Terminals and properties"

Purpose:
The HTTPHeader node provides a toolkit interface to manipulate HTTP headers without any need for coding; it does not modify the message body. You can add or remove a whole header, or selected header properties. You can set the properties to a fixed value, or to a value specified by an XPath expression that accesses a value in one of the message trees. XPath is used to provide a valid location from which a value for a property can be copied. For example, the location can be the body of the message, the local environment tree or exception list.

HTTPInput and HTTPResponse headers can only be deleted or carried forward from the incoming message; their header properties cannot be modified or added to.

The HTTPHeader node is contained in the **HTTP** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:
Look at the following sample for more details about how to use the node:
- HTTPHeader node

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Terminals and properties:

When you have put an instance of the node into a message flow, you can configure it; see . This node has no mandatory properties.

HTTPHeader node terminals are described in the following table:

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the input message is routed if a failure is detected during extraction.
Out	The output terminal to which the transformed message is routed if the input message is processed successfully.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The HTTPHeader node Description properties are described in the following table:

Property	M	C	Default	Description
Node name	No	No	HTTPHeader	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

HTTPInput Header properties are described in the following table:

Property	M	C	Default	Description
HTTPInput Header Options	No	Yes	Carry forward header	Options to control the HTTPInputHeader as a whole. Select **Carry forward the header** to carry forward values from incoming message if present. Select **Delete header** to delete the header if present.

HTTPResponse Header properties are described in the following table:

Property	M	C	Default	Description
HTTPResponse Header Options	No	Yes	Carry forward header	Options to control the HTTPResponseHeader as a whole. Select **Carry forward the header** to carry forward values from incoming message if present. Select **Delete header** to delete the header if present.

HTTPRequest Header properties are described in the following table:

Property	M	C	Default	Description
HTTPRequest Header Options	No	Yes	Carry forward header	Configure the HTTPRequest header. These options are available. **Carry forward header** Select this option to carry forward values from an incoming message. **Add header** Select this option to add new properties to the header, or to modify or delete existing properties. **Modify header** Select this option to add properties, or modify and delete existing properties. **Delete header** Select this option to remove the HTTPRequest header and all associated properties from the incoming message.
Clear incoming values	No	Yes	Cleared	This option, which is enabled only if you choose **Modify header**, removes all property names and associated values from the incoming message if present.
HTTPRequest Header	No	Yes	No default value	This field is enabled only if you chose **Add header** or **Modify header** for the **HTTPRequest Header Options**. The screen has no predefined properties; you use it to create custom properties and values. Use the property table to add new properties, or modify or delete existing properties, for the header. There is no limit to the number of properties. Each property must have a name and a type qualifier. The type qualifier can be `Value`, `XPath`, or `Delete`. **Value** Enter a new valid value for the selected property. A null value or empty string is also considered as a valid value. **XPath** Specify a valid XPath expression. WebSphere Message Broker supports XPath definitions that start with an XPath variable such as `$Root` or `$LocalEnvironment`. Only the first occurrence is returned if there are multiple values for the XPath expression. (Examples of valid XPath expressions are: $LocalEnvironment/Host, and $Root/HTTPRequest/Content-Type). **Delete** Specify the property to be deleted from the incoming message. The value associated with the selected property is also deleted.

HTTPReply Header properties are described in the following table:

Property	M	C	Default	Description
HTTPReply Header Options	No	Yes	Carry forward header	Configure the HTTPReply header. These options are available. **Carry forward header** Select this option to carry forward values from an incoming message. **Add header** Select this option to add new properties to the header, or to modify or delete existing properties. **Modify header** Select this option to add properties, or modify and delete existing properties. **Delete header** Select this option to remove the HTTPReply header and all associated properties from the incoming message.
Clear incoming values	No	Yes	Cleared	This option, which is is enabled only if you choose **Modify header**, removes all property names and associated values from the incoming message if present.
HTTPReply Header	No	Yes	No default value	This field is enabled only if you chose **Add header** or **Modify header** for **HTTPRequest Header Options**. The screen has no predefined properties; you use it to create custom properties and values. Use the property table to add new properties, or modify or delete existing properties, for the header. There is no limit to the number of properties. Each property must have a name and a type qualifier. The type qualifier can be `Value`, `XPath`, or `Delete`. **Value** Enter a new valid value for the selected property. A null value or empty string is also considered as a valid value. **XPath** Specify a valid XPath expression. WebSphere Message Broker supports XPath definitions that start with an XPath variable such as `$Root` or `$LocalEnvironment`. Only the first occurrence is returned if there are multiple values for the XPath expression. (Examples of valid XPath expressions are: $LocalEnvironment/Host, and $Root/HTTPRequest/Content-Type). **Delete** Specify the property to be deleted from the incoming message. The value associated with the selected property is also deleted.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

HTTPInput node

Use the HTTPInput node to receive an HTTP message from an HTTP client for processing by a message flow.

This topic contains the following sections:

Purpose:
If you use the HTTPInput node with the HTTPReply and HTTPRequest nodes, the broker can act as an intermediary for web services, and web service requests can be transformed and routed in the same way as other message formats that are supported by WebSphere Message Broker.

Web service requests can be received either in standard HTTP (1.0 or 1.1) format or in HTTP over SSL (HTTPS) format. For more information about web services, see "Processing Web service messages" on page 1601.

The HTTPInput node supports HTTP POST and HTTP GET. For more information about enabling HTTP GET, see "HTTPRequest node" on page 4488.

If your message flows are processing SOAP messages, use the SOAP nodes in preference to the HTTPInput node to take advantage of enhanced features, including WS-Addressing and WS-Security.

The HTTPInput node handles messages in the following message domains:
- MRM
- XMLNSC
- XMLNS
- MIME
- BLOB
- XML (this domain is deprecated; use XMLNSC)
- JSON

HTTP messages are always non-persistent, and have no associated order.

HTTP messages are non-transactional. However, if the message flow interacts with a database or another external resource, such as a WebSphere MQ queue, these interactions are performed in a transaction. The HTTPInput node provides commit or rollback, depending on how the message flow has ended, and how it is configured for error handling (how failure terminals are connected, for example). If

the message flow is rolled back by this node, a fault message is generated and returned to the client. The format of the fault is defined by the `Fault format` property.

If an exception occurs downstream in this message flow, and it is not caught but is returned to this node, the node constructs an error reply to the client. This error is derived from the exception, and the format of the error is defined by the `Fault format` property.

If you include an output node in a message flow that starts with an HTTPInput node, the output node can be any of the supported output nodes (including user-defined output nodes). You can create a message flow that receives messages from web service clients, and generates messages for clients that use all the supported transports to connect to the broker. You can configure the message flow to request the broker to provide any conversion that is necessary.

If you create a message flow to use as a subflow, you cannot use a standard input node; you must use an Input node as the first node to create an In terminal for the subflow.

The HTTPInput node is contained in the **HTTP** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:
If you include an HTTPInput node in a message flow, you must either include an HTTPReply node in the same flow, or pass the message to another flow that includes an HTTPReply node (for example, through an MQOutput node to a second flow that starts with an MQInput node). In the latter case, the request from, and reply to, the client are coordinated by the request identifier stored in the local environment by the HTTPInput node.

You cannot use a SOAPReply node to respond to a web service request that is received by an HTTPInput node; the broker generates an exception when the reply is attempted.

When the HTTPInput node receives a message from a web service client, the node starts the appropriate parsers to interpret the headers and the body of the message, and to create the message tree that is used internally by the message flow. The node creates a unique identifier for the input message and stores it as a binary array of 24 bytes in the local environment tree at LocalEnvironment.Destination.HTTP.RequestIdentifier. This value is used by the HTTPReply node, therefore you must not modify it.

By default, HTTP and HTTPS messages are handled by the broker-wide listener, which is started when a message flow that includes an HTTP node is started. All inbound and outbound HTTP messages are routed through this listener, for all HTTP nodes deployed to all message flows in all execution groups on the broker.

You can configure an execution group to use its embedded listener to service the HTTP nodes in all message flows that are deployed to that execution group. The embedded listener, which is always used for the SOAP nodes, communicates directly with the client and the nodes.

If you configure the execution group to use its embedded listener for HTTP nodes, you must deploy the flow that includes the HTTPReply node to the same execution group. If your broker is configured to start the broker-wide listener to support HTTP nodes, you must deploy the reply flow to the same broker, but the execution group is not significant.

For further information about using the embedded listener, see "HTTP listeners" on page 1589.

If a client sends a `gzip` or `deflate` compressed request to the HTTPInput node, it can be extracted using the `Decompress input message` option. This decompression occurs only if the execution group has been configured so that HTTP nodes use the embedded execution group HTTP listener, and not the broker-wide HTTP listener.

Using the HTTPInput and HTTPReply nodes to act as a web server:
A broker can support multiple HTTPInput nodes. When you configure the HTTPInput node, specify the requests to which the node listens in the form of a URL path, excluding the host and port details.

For example, if the broker is listening on address `http://localhost:7080`, and receives the request `http://localhost:7080/Joe/Mary`, the listener removes the HTTP address, leaving the request `Joe/Mary`. The listener then matches this request with the information that is specified in the URL property of the HTTPInput node or nodes.

The match is done from the most specific to the most generic data; you can use a wildcard (an asterisk) to satisfy less specific matches. For example, if you have configured an HTTPInput node to accept requests that match `/Joe/Mary`, that node receives the message. However if the request is `http://localhost:7080/Joe/Sally`, the match is not made with this node. It can match with a node that has a more generic URL, such as one of the following values:

```
/Joe/*
/*
```

If the request does not match any URL property, and you do not have an input node with /* specified, the HTTPInput node returns a response to the originator.

You can use a URL of /* to catch all requests that failed to match the URLs in the HTTPInput nodes, so that you can send a reply message and take other actions as appropriate.

This example uses port `7080`, which is the default HTTP port for the broker-wide listener. The default port numbers for the embedded execution group listener are `7800` for HTTP and `7843` for HTTPS. You can change these port numbers, and port ranges used by the execution group listeners, by using the **mqsichangeproperties** command.

If you choose to handle HTTP messages by using the execution group listener, you must carefully check the URL specifications in your HTTPInput and SOAPInput nodes. If both URL specifications match an incoming message, the wrong type of node might get the message, and processing might fail or produce unexpected results. This situation occurs if you specify identical values for the Path suffix for URL properties of the HTTPInput node and the SOAPInput node. It can also occur if you use wildcards in either or both specifications, and an incoming message matches both properties.

If you want to use the broker listener for HTTP and HTTPS traffic, check that the broker properties for the listener ports for HTTP and HTTPS are suitable. The default port for HTTP is `7080`; the default port for HTTPS is `7083`.

If you want to use the execution group listener, you must configure the execution group by using the **mqsichangeproperties** command to activate the listener for HTTP and HTTPS messages. The default port for HTTP is `7800`; the default port for HTTPS is `7843`. You can change these port numbers, and the ranges from which the ports are allocated, by using the **mqsichangeproperties** command.

If you want to collect trace information about HTTP message processing, see "Resolving problems when you use HTTP and SOAP nodes" on page 3407.

Connecting the terminals:
The HTTPInput node routes each message that it retrieves successfully to the Out terminal. If message validation fails, the message is routed to the Failure terminal; you can connect nodes to this terminal to handle this condition. If you have not connected the Failure terminal, the message is discarded, the `Maximum client wait time` expires, and an error is returned to the client. No other situations exist in which the message is routed to the Failure terminal.

If the message is caught by this node after an exception has been thrown further on in the message flow, the message is routed to the Catch terminal. If you have not connected the Catch terminal, the message is discarded, the `Maximum client wait time` expires, and an error is returned to the client.

If the `Maximum client wait time` expires, by default, the listener returns a fault message to the client to indicate that the timeout has expired. If the HTTP Timeout terminal is connected and the execution group is configured such that the HTTP nodes use the embedded listener, this timeout fault message is propagated to the timeout terminal. In this scenario the listener waits again for the interval defined by the `Maximum client wait time` (sec) property, or for 10 seconds, whichever is the shorter interval:

- If a response is received before this second interval expires, the listener propagates the response to the client.
- If a response is not received before this second interval expires, the listener sends a fault message to the client, indicating that its timeout has expired.

Because the listener waits for only a brief interval after the message has been propagated through the HTTP Timeout terminal, you must ensure that the sequence of nodes that you connect to the HTTP Timeout terminal includes an HTTPReply node, which sends a response before this interval expires.

Terminals and properties:
When you have put an instance of the HTTPInput node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view.

The terminals of the HTTPInput node are described in the following table.

Terminal	Description
Failure	The output terminal to which the message is routed if an error occurs.
Out	The output terminal to which the message is routed if it is successfully retrieved.

Terminal	Description
HTTPTimeout	The output terminal to which a timeout fault message is routed if the HTTPReply node that is connected to the Out terminal does not respond within the time interval specified by the Maximum client wait time property. This terminal is used only if the execution group has been configured so that HTTP nodes use the embedded execution group listener.
Catch	The output terminal to which the message is routed if an exception is thrown downstream and caught by this node.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The HTTPInput node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, HTTPInput	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The HTTPInput node Basic properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Path suffix for URL	Yes	Yes		This property identifies the location from where web service requests are retrieved. Do not use the full URL. If the URL that you want is `http://`*`hostname`*`[:`*`port`*`]/[`*`path`*`]`, specify either */path* or */path fragment*`/*` where * is a wildcard that you can use to mean match any.	URLSpecifier
Use HTTPS	No	Yes	Cleared	This property identifies whether the node is to accept secure HTTP. If the node is to accept secure HTTP, select the check box.	useHTTPS

The HTTPInput node Advanced properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Set destination list	No	No	Selected	This property specifies whether to add the method binding name to the route to label destination list. If you select this check box, the method binding name is added so that you can use a RouteToLabel node in the message flow after the HTTPInput node.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Label prefix	No	No	None	The prefix to add to the method name when routing to label. Add a label prefix to avoid a clash of corresponding label nodes when you include multiple WebSphere Message Broker input nodes in the same message flow. By default, there is no label prefix, therefore the method name and label name are identical.	

Parse Query String	No	No	False	This property causes any query string that is present with an incoming message to be parsed and decoded (according to `http://tools.ietf.org/html/rfc3986`) into the following location in the local environment as a series of name-value elements that match the names and values present in the query string:
				
				`LocalEnvironment.HTTP.Input.QueryString`
				
				For example, for this query string:
				
				`?myParam1=my%22Value%221&myParam2=my%22Value%222`
				the following elements are placed into the local environment under the QueryString folder:
				
				`myParam1` with a value of `my"Value"1`
				`myParam2` with a value of `my"Value"2`
				
				If the `QueryString` uses a character set that is not UTF-8, you can use the *MQSI_HTTP_QUERY_STRING_CCSID* environment variable to specify the CCSID of the `QueryString`. For example, if your HTTPRequest node has a `QueryStringCCSID` of 943, you can set *MQSI_HTTP_QUERY_STRING_CCSID* to 943 to convert the query string parameters to the 943 code page.	

Decompress input message	No	Yes	Cleared	This property indicates whether an inbound HTTP request is decompressed or not. This property is used only if the execution group has been configured so that HTTP nodes use the embedded execution group HTTP listener. If this option is selected, and the HTTP header Content-Encoding field is "`gzip`" or "`deflate`", the input message is decompressed and propagated to the Out terminal, and the Content-Encoding field is removed.	decompressInputMessage

The HTTPInput node Input Message Parsing properties are described in the following table.

Property	M	C	Default	Description
Message domain	No	No	BLOB	The domain that is used to parse the incoming message. If you leave this field blank, the default value is BLOB. Select the name of the parser that you are using from the list: • MRM • XMLNSC • XMLNS • MIME • BLOB • XML (this domain is deprecated; use XMLNSC) • JSON You can also specify a user-defined parser, if appropriate.
Message set	No	No		The name or identifier of the message set in which the incoming message is defined. All available message sets are in the list. If you are using the MRM parser or the XMLNSC parser in validating mode, select the Message set that you want to use. This list is populated with available message sets when you select MRM or XMLNSC as the domain. If you set this property, then later update the project dependencies to remove this message set reference, a warning is issued. Either update the Message set property, or restore the reference to this message set project.
Message type	No	No		The name of the incoming message. If you are using the MRM parser, select the type of message from the list in Message type. This list is populated with messages that are defined in the Message set that you have selected.
Message format	No	No		The name of the physical format of the incoming message. If you are using the MRM parser, select the format of the message from the list in Message format. This list includes all the physical formats that you have defined for this Message set.

The properties of the Parser Options for the HTTPInput node are described in the following table.

Property	M	C	Default	Description
Parse timing	No	No	On Demand	This property controls when an input message is parsed. Valid values are On Demand, Immediate, and Complete. By default, this property is set to On Demand, which causes parsing of the message to be delayed. To cause the message to be parsed immediately, see “Parsing on demand” on page 4173.
Build tree using XML schema data types	No	No	Cleared	This property controls whether the XMLNSC parser creates syntax elements in the message tree with data types taken from the XML schema. You can select this property only if you set the Validate property on the **Validation** tab to Content or Content and Value.
Use XMLNSC compact parser for XMLNS domain	No	No	Cleared	This property controls whether the XMLNSC Compact Parser is used for messages in the XMLNS Domain. If you set this property, the message data is displayed under XMLNSC in nodes that are connected to the output terminal when the input MQRFH2 header or the input message Parsing property Message domain is XMLNS.

Property	M	C	Default	Description
Retain mixed content	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters mixed text in an input message. If you select the check box, elements are created for mixed text. If you clear the check box, mixed text is ignored and no elements are created.
Retain comments	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters comments in an input message. If you select the check box, elements are created for comments. If you clear the check box, comments are ignored and no elements are created.
Retain processing instructions	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters processing instructions in an input message. If you select the check box, elements are created for processing instructions. If you clear the check box, processing instructions are ignored and no elements are created.
Opaque elements	No	No	Blank	This property is used to specify a list of elements in the input message that are to be opaquely parsed by the XMLNSC parser. Opaque parsing is performed only if validation is not enabled (that is, if `Validate` is `None`); entries that are specified in `Opaque Elements` are ignored if validation is enabled.

The HTTPInput node Error handling properties are described in the following table.

Property	M	C	Default	Description	**mqsiapplybaroverride command property**
Maximum client wait time (sec)	Yes	Yes	`180`	The length of time, in seconds, for which the TCP/IP listener that received the input message from the web service client waits for a response from the HTTPReply node in the message flow. The valid range is zero (which means a short wait) through (2^{31})-1. If a response is received within this time, the listener propagates the response to the client. If a response is not received in this time, a fault message is generated indicating that the timeout has expired. This fault message is either sent by the listener or timeout terminal processing. For more details about the HTTP Timeout terminal, see "Connecting the terminals" on page 4477 and "Terminals and properties" on page 4477.	
Fault format	No	Yes	`SOAP 1.1`	The format of any HTTP errors that are returned to the client. Valid values are `SOAP 1.1`, `SOAP 1.2`, and `HTML`.	faultFormat

The Validation properties of the HTTPInput node are described in the following table. If a message is propagated to the Failure terminal of the node, it is not validated. For more details, see "Validating messages" on page 1478 and "Validation properties" on page 4169.

Property	M	C	Default	Description	**mqsiapplybaroverride command property**
Validate	No	Yes	`None`	This property controls whether validation takes place. Valid values are `None`, `Content and Value`, and `Content`.	validatemaster

Property	M	C	Default	Description	mqsiapplybaroverride command property
Failure action	No	No	Exception	This property controls what happens if validation fails. You can set this property only if you set Validate to Content or Content and Value. Valid values are User Trace, Local Error Log, Exception, and Exception List.	

The Security properties of the HTTPInput node are described in the following table. Set values for the properties that control the extraction of an identity from a message when a security profile is associated with the node. For more information about these properties, see "Identity" on page 390, "Configuring the extraction of an identity or security token" on page 447, "Message flow security overview" on page 383, and "Setting up message flow security" on page 431

Property	M	C	Default	Description
Identity token type	No	No	None	This property specifies the type of identity token that is present in the incoming message. Valid values are: • Transport Default • Username • Username + Password • SAML Assertion • X.509 Certificate If this property is not specified, the identity is retrieved from the Basic-Auth transport header and the type is set to Username + Password.
Identity token location	No	No	None	This property specifies where, in the message, the identity can be found. The location is specified as an ESQL field reference, an XPath expression, or a string literal. If you use a string literal, it must be enclosed in single quotation marks and must not contain a period (.), If this property is not specified, the identity is retrieved from the Authorization Transport headers.
Identity password location	No	No	None	This property specifies where, in the message, the password can be found. The location is specified as an ESQL field reference, an XPath expression, or a string literal. If you use a string literal, it must be enclosed in single quotation marks and must not contain a period (.), If you do not specify a value for this property, the password is retrieved from the Authorization Transport headers. You can set this property only if the Identity type is set to Username + Password.
Identity IssuedBy location	No	No	None	This property specifies a string or path expression that describes the issuer of the identity. The location is specified as an ESQL field reference, an XPath expression, or a string literal. If you use a string literal, it must be enclosed in single quotation marks and must not contain a period (.), The value specifies the Issuer that is passed to a WS-Trust v1.3 STS provider. If you do not specify a value for this property, the default value is the name of the User Agent, or, if this name is not set, the string HTTP.
Treat security exceptions as normal exceptions	No	No	False	This property specifies whether to treat security exceptions (such as Access Denied) as normal exceptions, and propagate them to the Failure terminal (if wired). This option is turned off by default, which ensures that security exceptions cause the message to be backed out even if the Failure terminal is wired.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Processing HTTP messages" on page 1579
Hypertext Transfer Protocol (HTTP) is an Internet protocol that is used to transfer and display hypertext and XML documents on the Web.

"Message flow security overview" on page 383
WebSphere Message Broker provides a security manager, which enables you to control access to individual messages in a message flow, using the identity of the message.

"ESQL field references" on page 2381
An ESQL field reference is a sequence of period-separated values that identify a specific field (which might be a structure) within a message tree or a database table.

"HTTP listeners" on page 1589
You can choose between broker-wide listeners and execution group (embedded) listeners to manage HTTP messages in your HTTP flows. Learn about the two types of listener, how ports are assigned to them, and how you can switch from one to the other for individual execution groups.

"HTTP headers" on page 1583
When an HTTPInput or HTTPRequest node receives a message, it parses the HTTP headers to create elements in the message tree. When an HTTPReply or HTTPRequest node sends a message, it parses the HTTP headers from the message tree into a bit stream.

"Using compression with HTTP and SOAP nodes" on page 1597
You can configure HTTP and SOAP nodes to use HTTP compression and decompression when sending and receiving messages.

Related tasks:

"Setting up message flow security" on page 431
Set up security on a message flow to control access based on the identity of a message passing through the message flow.

"Configuring HTTPInput and HTTPReply nodes to use SSL (HTTPS)" on page 535
Configure the HTTPInput and HTTPReply nodes to communicate with other applications that use HTTPS by creating a keystore file, configuring the broker or execution group to use SSL, and creating a message flow to process HTTPS requests.

"Resolving problems when you use HTTP and SOAP nodes" on page 3407
Use the advice given here to help you to resolve common problems that can arise when you develop Web Services message flows that contain HTTP and SOAP nodes.

"Validating messages" on page 1478
The broker provides validation based on the message model for predefined

messages.

"Using XPath" on page 1506
XPath provides an alternative method to ESQL for entering expressions in the property fields of specific built-in nodes.

"Using timeouts with HTTP and SOAP nodes" on page 1595
Connect the HTTP Timeout terminal of the HTTPInput or SOAPInput nodes to further nodes to process timeouts.

Related reference:

"Validation properties" on page 4169
You can control validation by setting properties on the Validate and Parser Options tabs for the nodes that are listed in the following table.

"Parsing on demand" on page 4173
On-demand parsing, referred to as partial parsing, is used to parse an input message bit stream only as far as is necessary to satisfy the current reference. The parsers that can perform partial parsing of input messages are the JSON, MRM, XML, XMLNS, and XMLNSC parsers.

"**mqsichangetrace** command" on page 3822
Use the **mqsichangetrace** command to set the tracing characteristics for a broker.

"**mqsireadlog** command" on page 3905
Use the **mqsireadlog** command to retrieve trace records for the specified component.

"HTTPReply node"
Use the HTTPReply node to return a response from the message flow to an HTTP client. This node generates the response to an HTTP client from which the input message was received by the HTTPInput node, and waits for confirmation that it has been sent.

"HTTPRequest node" on page 4488
Use the HTTPRequest node to interact with a web service.

"Input node" on page 4511
Use the Input node as an In terminal for an embedded message flow (a subflow).

"SecurityPEP node" on page 4729
Use the SecurityPEP node in a message flow to invoke the message flow security manager at any point in the message flow.

HTTPReply node

Use the HTTPReply node to return a response from the message flow to an HTTP client. This node generates the response to an HTTP client from which the input message was received by the HTTPInput node, and waits for confirmation that it has been sent.

This topic contains the following sections:
- "Purpose"
- "Connecting the output terminals to another node" on page 4485
- "Terminals and properties" on page 4485

Purpose:
The HTTPReply node can be used in a message flow that sends a response to an inbound HTTP or HTTPS messages. The most common example of this scenario is a message flow that implements a Web service.

For more information about Web services, see "Processing Web service messages" on page 1601.

By default, HTTP messages are handled by the broker-wide listener, which is started when a message flow that includes HTTP nodes is started. All inbound and outbound HTTP messages are routed through this listener, for all HTTP nodes deployed to all message flows in all execution groups on the broker.

You can configure the execution group to use its embedded listener to service the HTTP nodes in all message flows that are deployed to that execution group. The embedded listener, which is always used for the SOAP nodes, communicates directly with the client and the nodes.

For further information about using the embedded listener, see "HTTP listeners" on page 1589.

You cannot use an HTTPReply node to respond to a Web service request that is received by a SOAPInput node; the broker generates an exception when the reply is attempted.

If you have configured the execution group to use its embedded listener for HTTP nodes, you must deploy the flow that includes the HTTPReply node to the same execution group as the message flow that includes the HTTPInput node. If your broker is configured to start the broker-wide listener to support HTTP nodes, you must deploy the reply flow to the same broker, but the execution group is not significant, because the listener is shared.

The HTTPReply node constructs a reply message for the Web service client from the entire input message tree, and returns it to the requester. If the message was initially received by an HTTPInput node in another message flow, the response is associated with the reply by a request identifier that is stored in the local environment of the message by the HTTPInput node.

The HTTPReply node is contained in the **HTTP** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Connecting the output terminals to another node:
Connect the Out or Failure terminal of this node to another node in this message flow if you want to process the message further, process errors, or send the message to an additional destination.

Terminals and properties:
When you have put an instance of the HTTPReply node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The HTTPReply node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the message is routed if a failure is detected when the message is propagated.

Terminal	Description
Out	The output terminal to which the message is routed if it has been propagated successfully, and if further processing is required in this message flow.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file for deployment).

The HTTPReply node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	HTTPReply	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The HTTPReply node Basic properties are described in the following table.

Property	M	C	Default	Description
Ignore transport failures	Yes	No	Selected	Select `Ignore transport failures` if you want transport-related failures to be ignored (for example, if the client is disconnected). If you clear the check box, and a transport-related error occurs, the input message is propagated to the Failure terminal. If you clear the check box, you must supply a value for `Reply send timeout (sec)`.
Reply send timeout (sec)	Yes	No	`120`	Set the `Reply send timeout (sec)` value if you are not ignoring transport failures. This property specifies the length of time, in seconds, that the node waits for an acknowledgment that the client has received the reply. If the acknowledgment is received within this time, the input message is propagated through the Out terminal to the rest of the message flow, if it is connected. If an acknowledgment is not received within this time, the input message is propagated through the Failure terminal, if it is connected. If the Failure terminal is not connected, and an acknowledgment is not received in time, an exception is generated. The valid range is zero (which means an indefinite wait) to (2^{31})-1. This property is valid only if `Ignore transport failures` is cleared.
Generate default HTTP headers from reply or response	Yes	No	Selected	Select `Generate default HTTP headers from reply or response` if you want the default Web service headers to be created using values from the HTTPReplyHeader or the HTTPResponseHeader. If the appropriate header is not present in the input message, default values are used. The node always includes, in the HTTPReplyHeader, a Content-Length header, which is set to the correct calculated value, even if this header was not included in the original request.

The Validation properties of the HTTPReply node are described in the following table.

If a message is propagated to the Failure terminal of the node, it is not validated. For a full description of these properties, see "Validation properties" on page 4169.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Validate	No	Yes	Inherit	This property controls whether validation takes place. Valid values are None, Content and Value, Content, and Inherit.	validateMaster
Failure action	No	No	Exception	This property controls what happens if validation fails. You can set this property only if you set Validate to Content or Content and Value. Valid values are User Trace, Local Error Log, Exception, and Exception List.	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Processing HTTP messages" on page 1579
Hypertext Transfer Protocol (HTTP) is an Internet protocol that is used to transfer and display hypertext and XML documents on the Web.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"HTTP listeners" on page 1589
You can choose between broker-wide listeners and execution group (embedded) listeners to manage HTTP messages in your HTTP flows. Learn about the two types of listener, how ports are assigned to them, and how you can switch from one to the other for individual execution groups.

Related tasks:

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Validating messages" on page 1478
The broker provides validation based on the message model for predefined messages.

Related reference:

"Validation properties" on page 4169
You can control validation by setting properties on the Validate and Parser Options tabs for the nodes that are listed in the following table.

"HTTPInput node" on page 4474
Use the HTTPInput node to receive an HTTP message from an HTTP client for processing by a message flow.

"HTTPRequest node"
Use the HTTPRequest node to interact with a web service.

HTTPRequest node

Use the HTTPRequest node to interact with a web service.

This topic contains the following sections:

Purpose:
The HTTPRequest node interacts with a web service, using all or part of the input message as the request that is sent to that service. You can also configure the node to create an output message from the contents of the input message, augmented by the contents of the web service response, before you propagate the message to subsequent nodes in the message flow.

Depending on the configuration, this node constructs an HTTP or an HTTP over SSL (HTTPS) request from the specified contents of the input message, and sends this request to the web service. The node receives the response from the web service, and parses the response for inclusion in the output tree. The node generates HTTP headers if they are required by your configuration.

You can use this node in a message flow that does or does not contain an HTTPInput or HTTPReply node.

The HTTPRequest node handles messages in the following message domains:
- XMLNSC
- JSON
- BLOB
- MIME
- MRM
- XMLNS

The HTTPRequest node is contained in the **HTTP** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using the HTTPRequest node to issue a request to a web service:
An HTTP request has two parts:

1. The URL of a service.
2. A stream of data that the remote server processes, then sends back a response, which is often a SOAP or other web service message in XML.

The URL is of the format `http://<address>[:<port>]/<function>`; for example, `http://localhost:7080/request`. This URL can be specified statically in the HTTPRequest node parameters as a field in the message itself, or as a field in the local environment. The data to be sent to the web service can be the whole, or a portion of, the message tree, as specified in the HTTPRequest node properties.

The data must be in CCSID 1208 format for most requests. The reply can replace the input message, or be inserted into the message tree; the location is specified in the HTTPRequest node parameters. The domain for the reply is XMLNS. If the request is successful, the HTTPResponse is inserted into the front of the message tree, the reply placed in the specified location in the tree, and the request propagated to the Out terminal. If the HTTPRequest node is not able to issue the request, an ExceptionList is inserted into the message tree and the tree is propagated to the Failure terminal.

If the request is sent successfully by the HTTPRequest node, but the web service is not successful, the HTTPResponse is inserted into the message tree, and propagated to the Error terminal. The error message location parameter on the HTTPRequest node specifies where in the tree the response is placed, for example `OutputRoot.XMLNS.error`. You might have to use a Compute node to cast this response to an appropriate code page to be able to display the data, for example:

```
Set OutputRoot.XMLNS.error850 = CAST(InputRoot.XMLNS.error.BLOB as CHAR CCSID 850);
```

For information about HTTP, see Hypertext Transfer Protocol - HTTP/1.1. For more information about HTTP return codes, see HTTP Response codes.

You can specify a timeout interval, so that if the request takes longer than the specified duration, the request is propagated to the Failure terminal with an appropriate message. For each request that the HTTPRequest node processes, it opens a connection, and then closes it when the response is returned. If the timeout interval is specified, the socket is closed after the interval. This closure ensures that a request gets only the correct response, and any response data for a request that has timed out is discarded.

You can use the HTTP proxy to route a request through an intermediate site. You can run tools as a proxy to see the request and the response, and therefore debug your flows. The HTTP destination is as seen by the proxy; if you specify the HTTP destination of localhost, and the HTTP proxy is running on a different computer, the request is routed to the remote proxy computer, not the computer from which the original request was issued.

Using the HTTPRequest node in a message flow:
The HTTPRequest node can be used in any message flow that must send an HTTP request. The most common example is a message flow that calls a web service.

For more information about web services, see “Processing Web service messages” on page 1601.

Handling errors

The node interacts directly with an external service using TCP/IP; it can, therefore, experience the following types of error:

- Errors that are generated by TCP/IP, for example `no route to host` or `connection refused`.

  If the node detects these errors, it generates an exception, populates the exception list with the error information that is received, and routes the input message unchanged to the Failure terminal.
- Errors that are returned by the web server. These errors are represented by HTTP status codes that are outside the range 100 - 299. If the node detects these errors, it routes the reply to the Error terminal while following the properties specified on the **Error** tab.

The reply is produced as a BLOB message because the node cannot determine in what format the reply will be. If you have not configured this node to handle redirection, messages with a redirection status code (3xx) are also handled in the same way.

HTTP Response Codes

The HTTPRequest node treats the 100 series status codes as a 'continue' response, discards the current response, and waits for another response from the web server.

The 200 series status codes are treated as success, the settings on the various tabs on the node determine the format of the output message that is generated, and the response is routed to the Out terminal of the node.

The 300 series status codes are for redirection. If the Follow HTTP(s) Redirection property is selected, the node resends the request to the new destination that is specified in the response that is received. If the Follow HTTP(s) Redirection property is not selected, the codes are treated as an error, as described in "Using the HTTPRequest node to issue a request to a web service" on page 4488. For more information about HTTP return codes, see HTTP Response codes.

The 400 and 500 series status codes are errors, and are treated as described in "Using the HTTPRequest node to issue a request to a web service" on page 4488. For more information about HTTP return codes, see HTTP Response codes.

Manipulating headers

If you select Replace input message with web-service response or Replace input with error, the header for the input message (the header that belongs to the message when it arrives at the In terminal of the HTTPRequest node) is not propagated with the message that leaves the HTTPRequest node. However, if one of the properties that specify a location in the message tree is specified, the input message headers are propagated.

The HTTPResponse header, which contains the headers that are returned by the remote web service, is the first header in the message (after Properties) that is propagated from the node. This action is taken regardless of the options that are selected. Therefore, for the reply from the HTTPRequest node to be put to a WebSphere MQ queue, manipulate the headers so that an MQMD is the first header (after Properties).

If you are replacing the input message with a response, you can copy the input message MQMD to the Environment tree before the HTTPRequest node, and then copy it back into the message tree after the HTTPRequest node. If you are specifying a location for the response, in order to maintain existing input message headers, you must move or remove the HTTP Response header so that the MQMD is the first header.

The following example contains ESQL that removes the HTTPHeader:

```
SET OutputRoot = InputRoot;
SET OutputRoot.HTTPResponseHeader = NULL;
```

The following example contains ESQL for moving the HTTPHeader, and therefore preserving the information that it provides:

```
SET OutputRoot = InputRoot;
DECLARE HTTPHeaderRef REFERENCE TO OutputRoot.HTTPResponseHeader;
DETACH HTTPHeaderRef;
ATTACH HTTPHeaderRef TO OutputRoot.MQMD AS NEXTSIBLING;
```

Configuring the HTTPRequest node:
When you have put an instance of the HTTPRequest node into a message flow, you can configure the node; see . The properties of the node are displayed in the Properties view.

All mandatory properties, for which you must enter a value, are marked with an asterisk.

Configure the HTTPRequest node:

1. Optional: On the **Description** tab, enter a `Short description`, a `Long description`, or both. You can also rename the node on this tab.
2. On the **Basic** tab:
   a. The HTTPRequest node determines the URL for the web service to which it sends a request. Select one of the following three options; the node checks these in the order shown (that is, the first always overrides the second, the second overrides the third):
      1) X-Original-HTTP-URL in the HTTPRequest header in the input message
      2) LocalEnvironment.Destination.HTTP.RequestURL in the input message
      3) The `Web service URL` property

      The first two options provide dynamic methods to set a URL for each input message as it passes through the message flow. To use either of these options, include a Compute node in the message flow, before the HTTPRequest node, to create and initialize the required value.

      The third option provides a value that is fixed for every message that is received in this node. Set this property to contain a default setting that is used if the other fields have not been created, or contain a null value. If either field contains a value, the setting of this property is ignored. The `Web service URL` property must contain a valid URL or the deployment fails. Ensure that the value that you set in X-Original-HTTP-URL or the LocalEnvironment.Destination.HTTP.RequestURL is also a valid URL; if it is not, the node uses the default setting from the `Web service URL` property.

      If a URL begins `http://`, the request node makes an HTTP request to the specified URL. If the URL begins `https://`, the request node makes an HTTP over SSL (HTTPS) request to the specified URL, using the parameters that are specified on the **SSL** tab for the node.
   b. Set the value of the `Request timeout (sec)` property, which is the length of time, in seconds, that the node waits for a response from the web service. If a response is received within this time, the reply is propagated through the Out terminal to the rest of the message flow. If a response is not received within this time, the input message is propagated through the Failure terminal, if it is connected. If the Failure terminal is not connected, and a response is not received in this time, an exception is generated.
3. On the **HTTP Settings** tab:
   a. In `HTTP(S) proxy location`, set the location of the proxy server to which requests are sent.
   b. Select `Follow HTTP(S) redirection` to specify how the node handles web service responses that contain an HTTP status code of 300 to 399:

- If you select the check box, the node follows the redirection that is provided in the response, and reissues the web service request to the new URL (included in the message content).
- If you clear the check box, the node does not follow the redirection provided. The response message is propagated to the Error terminal.

c. Select one of the options for the `HTTP version` property. Valid values are: `1.0` or `1.1`.

   If you select the `HTTP version` property value `1.1`, you can also select `Enable HTTP/1.1 keep-alive`.

d. Select one of the options for the `HTTP method` property. Valid values are: `POST`, `GET`, `PUT`, `DELETE`, and `HEAD`.

e. Select one of the options for the `Use compression` property to specify the compression of the content of the HTTP request. You can select `gzip`, `zlib (deflate)`, `deflate` or `none`. The value `zlib (deflate)` represents RFC 1950 and RFC 1951 combined, and `deflate` represents RFC 1951 only. The default value is `none`, meaning that the content of the request is not compressed.

4. On the **SSL** tab, if you want to use HTTP over SSL (HTTPS) requests, set the values for HTTPS requests:

   a. Specify the `Protocol` property that you want to use to make the request. Both ends of an SSL connection must agree on the protocol to use. Therefore, the selected protocol must be one that the remote server can accept. The following options are available:
      - `SSL`. This option is the default. This option tries to connect using the SSLv3 protocol first, but enables the handshake to fall back to the SSLv2 protocol where the SSLv2 protocol is supported by the underlying JSSE provider.
      - `SSLv3`. This option tries to connect with the SSLv3 protocol only. Fallback to SSLv2 is not possible.
      - `TLS`. This option tries to connect with the TLS protocol only. Fallback to SSLv3 or SSLv2 is not possible.
      - `TLSv1.0`. This option attempts to connect with the TLS v1.0 protocol only. Fallback to SSLv3 or SSLv2 is not allowed.
      - `TLSv1.1`. This option attempts to connect with the TLS v1.1 protocol only. Fallback to SSLv3, SSLv2, or TLSv1.0 is not allowed.
      - `TLSv1.2`. This option attempts to connect with the TLS v1.2 protocol only. Fallback to SSLv3, SSLv2, TLSv1.0, or TLSv1.1 is not allowed.
      - `SSL_TLS`. This option enables all SSL v3.0 and TLS v1.0 protocols. Fallback to SSLv2 is not allowed.
      - `SSL_TLSv2`. This option enables all SSL v3.0 and TLS v1.0, v1.1, and v1.2 protocols. Fallback to SSLv2 is not allowed.

   b. Set the `Allowed SSL ciphers` property. Use this setting to specify a single cipher (such as SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA) or a list of ciphers that are the only ones used by the connection. This set of ciphers must include one or more that are accepted by the remote server. A comma is used as a separator between the ciphers. The default value is an empty string, which enables the node to use any, or all, of the available ciphers during the SSL connection handshake. This method gives the greatest scope for making a successful SSL connection.

5. On the **Response Message Parsing** tab, set values for the properties that describe the message domain, message set, message type, and message format that the node uses to determine how to parse the response message returned

by the web service. If an error message is returned by the web service, the values of these properties are ignored, and the message is parsed by the BLOB parser.

a. In `Message domain`, select the name of the parser that you are using from the list. If the field is blank, the default value is `BLOB`. Choose from the following options:
   - XMLNSC
   - JSON
   - BLOB
   - MIME
   - MRM
   - XMLNS

   You can also specify a user-defined parser, if appropriate.

b. If you are using the MRM or IDOC parser, or the XMLNSC parser in validating mode, select the `Message set` that you want to use. This list is populated with available message sets when you select MRM, XMLNSC, or IDOC as the domain.

c. If you are using the MRM parser, select the correct message type from the list in `Message type`. This list is populated with available message types when you select the MRM parser.

d. If you are using the MRM or IDOC parser, select the correct message format from the list in `Message format`. This list is populated with available message formats when you select the MRM or IDOC parser.

6. On the **Parser Options** subtab:

   a. `Parse timing` is, by default, set to `On Demand`, which causes parsing of the message to be delayed. To cause the message to be parsed immediately, see "Parsing on demand" on page 4173.

   b. If you are using the XMLNSC parser, set values for the properties that determine how the XMLNSC parser operates. For more information, see "Manipulating messages in the XMLNSC domain" on page 2546.

7. On the **Error Handling** tab, set values for the properties that determine how an error message returned by the web service is handled:

   a. For the whole web service error message to be propagated as the output message, leave `Replace input with error` selected (the default setting).

      For the web service error message to be included in the output message with part of the input message content, clear `Replace input with error` and set the `Error message location` property. If you clear this property, the node copies the input message to the output message and writes the web service error message over the output message content at the specified location (the input message itself is not modified).

   b. In the `Error message location` field, enter the start location (within the output message tree) at which the parsed elements from the web service error message bit stream are stored. This property is required only if you have cleared `Replace input with error`.

      You can enter any valid ESQL field reference, including expressions within the reference and new field references (to create a node in the message tree for the response). For example, enter:

      ```
      OutputRoot.XMLNSC.ABC.DEF
      ```

      or

      ```
      Environment.WSError
      ```

      If you select `Replace input with error`, this property is ignored.

8. On the **Advanced** tab, set values for the Advanced properties that describe the structure and content of the web service request and response:
   a. Specify the content of the request message that is sent to the web service:
      - For the request message to be the whole input message body, leave `Use whole input message as request` selected (the default setting).

        For the request message to contain a subset of the input message, clear `Use whole input message as request` and set the Request message location in tree property.
      - In the Request message location in tree field, enter the start location from which the content of the input message tree is copied to the request message. This property is required only if you have cleared Use whole input message as request. The node creates a request message and copies the specified parts of the input message (the input message itself is not modified).

        You can enter any valid ESQL field reference, including expressions within the reference. For example, enter:

        ```
        InputRoot.XMLNSC.ABC
        ```

        If you select Use whole input message as request, this property is ignored.

      When the appropriate message tree content is parsed to create a bit stream, the message properties (`Message domain`, `Message set`, `Message type`, and `Message format`) that are associated with the input message body and stored in the Properties folder are used.
   b. Specify the content of the output message that is propagated to the next node in the message flow:
      - For the whole web service response message to be propagated as the output message, leave `Replace input message with web-service response` selected (the default setting).

        For the web service response message to be included in the output message with part of the input message content, clear `Replace input message with web-service response` and set the `Response message location in tree` property. If you clear this property, the node copies the input message to the output message and writes the web service response message over the output message content at the specified location (the input message itself is not modified).
      - In the `Response message location in tree` field, enter the start location (within the output message tree) at which the parsed elements from the web service response message bit stream are stored. This property is required only if you have cleared `Replace input message with web-service response`.

        You can enter any valid ESQL field reference, including expressions within the reference, and including new field references (to create a node in the message tree for the response). For example, enter:

        ```
        OutputRoot.XMLNSC.ABC.DEF
        ```

        or

        ```
        Environment.WSReply
        ```

        If you select `Replace input message with web-service response`, this property is ignored.

When the response bit stream is parsed to create message tree contents, the message properties (`Message domain`, `Message set`, `Message type`, and `Message format`), that you have specified in the Response Message Parsing properties of the node, are used.

c. For the node to generate an HTTPRequestHeader for the request message, leave `Generate default HTTP headers from input` selected (the default setting).

   If you do not want the node to generate an HTTPRequestHeader for the request message, clear `Generate default HTTP headers from input`. To control the contents of the HTTPRequestHeader that is included in the request message, include a Compute node that adds an HTTPRequestHeader to the input message before this HTTPRequest node in the message flow, and clear this check box.

   - If you have selected `Generate default HTTP headers from input` and the input message includes an HTTPRequestHeader, the HTTPRequest node extracts web service headers from the input HTTPRequestHeader and adds any unique web service headers, except Host (see the following table), that are present in an HTTPInputHeader, if one exists in the input message. (An HTTPInputHeader might be present if the input message has been received from a web service by the HTTPInput node.)

     The HTTPRequest node also adds the web service headers shown in the following table, with default values, if these are not present in the HTTPRequestHeader or the HTTPInputHeader.

Header	**Default value**
SOAPAction	"" (empty string)
Content-Type	`text/xml; charset=`*`ccsid of the message body`* Unless the input message is in the JSON domain, where the default is: `application/json; charset=`*`ccsid of the message body`*
Host	The host name to which the request is to be sent.

     The HTTPRequest node also adds the optional header Content-Length with the correct calculated value, even if this value is not present in the HTTPRequestHeader or the HTTPInputHeader.

   - If you have selected `Generate default HTTP headers from input` and the input message does not include an HTTPRequestHeader, the HTTPRequest node extracts web service headers, except Host, from the HTTPInputHeader (if it is present in the input message). The HTTPRequest node adds the required web service headers with default values, if these values are not present in the HTTPInputHeader.
   - If you have cleared Generate default HTTP headers from input and the input message includes an HTTPRequestHeader, the node extracts all web service headers present in the input HTTPRequestHeader. The node does not check for the presence of an HTTPInputHeader in the input message, and it does not add the required web service headers if they are not supplied by the input HTTPRequestHeader.
   - If you have cleared Generate default HTTP headers from input and the input message does not include an HTTPRequestHeader, no web service headers are generated. The HTTPRequest node does not check for the presence of an HTTPInputHeader in the input message and does not add any required web service header. The request message is propagated to

the web service without an HTTPRequestHeader. This action typically causes an error to be generated by the web service, unless the web service is configured to handle the message contents.

If you have selected `Use compression` or `Accept compressed responses by default`, the Content-Encoding and Accept-Encoding HTTP header fields are populated regardless of whether you have selected `Generate default HTTP headers from input`:

- If the value of `Use compression` is not the default of `None`, the Content-Encoding HTTP header is populated with this value, and the bit stream is compressed. If the Content-Encoding header is already present in an existing HTTP header, this field is updated with the value of the `Use compression` property. If the existing Content-Encoding header already starts with the named compression function, then no further compression takes place. If the Content-Encoding header starts with `deflate`, then no compression takes place irrespective of whether `ZLIB (deflate)`or `deflate` is selected.
- If you have selected `Accept compressed responses`, the Accept-Encoding field is populated. If this field is already present in an existing HTTP header, the existing value overrides the property on the node. However, if a compressed response is received, it is not decompressed.

d. Select the `Accept compressed responses by default` property to indicate whether the request accepts compressed responses. If you select this option, the request can receive responses with a Content-Encoding of `gzip` or `deflate`. If such a response is received, the content is decoded and the Content-Encoding header is removed. If the Request Header does not contain an Accept-Encoding header then selecting this option sets the Accept-Encoding header to "gzip, deflate".

9. On the **Validation** tab, set Validation properties if you want the parser to validate the body of response messages against the `Message set`. (If a message is propagated to the Failure terminal of the node, it is not validated.) These properties do not cause the input message to be validated. It is expected that, if such validation is required, the validation has already been performed by the input node or a preceding validation node.

   For more details see “Validating messages” on page 1478 and “Validation properties” on page 4169.

Connecting the output terminals to another node

Connect the Out, Error, or Failure terminal of this node to another node in this message flow to process the message further, to process errors, or to send the message to an additional destination. If you do not connect the Error terminal, the message is discarded. If you do not connect the Failure terminal, the broker provides default error processing, see “Handling errors in message flows” on page 2823.

Terminals and properties:
The HTTPRequest node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the message is routed if a failure is detected during processing in the node.
Out	The output terminal to which the message is routed if it represents successful completion of the web service request, and if further processing is required within this message flow.

Terminal	Description
Error	The output terminal to which messages that include an HTTP status code that is not in the range 200 through 299, including redirection codes (3xx) if you have not set the property `Follow HTTP(s) redirection` property, is routed.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk on the panel if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the broker archive file to deploy it).

The HTTPRequest node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, HTTPRequest	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The HTTPRequest node Basic properties are described in the following table.

Property	M	C	Default	Description	**mqsiapplybaroverride command property**
Web service URL	Yes	Yes		The URL for the web service. You must provide this in the form `http://`*`hostname`*`[:`*`port`*`]/[`*`path`*`]` where • `http://`*`hostname`* must be specified. • *`port`* has a default of `80`. If you specify a value, you must include the : before the port number. • *`path`* has a default of `/`. If you specify a value, you must include the / before the path.	URLSpecifier
Request timeout (sec)	Yes	Yes	`120`	The time in seconds that the node waits for a response from the web service. The valid range is 1 through (2^{31})-1. You cannot enter a value that represents an unlimited wait. The timeout might take up to one second longer than the specified value.	timeoutForServer

The HTTPRequest node HTTP Settings properties are described in the following table.

Property	M	C	Default	Description	**mqsiapplybaroverride command property**
HTTP(S) proxy location	No	Yes		The proxy server to which requests are sent. This value must be in the form `hostname:port`.	httpProxyLocation
Follow HTTP(S) redirection	No	No	Cleared	If you select the check box, redirections are followed. If you clear this check box, redirections are not followed.	
HTTP version	No	Yes	`1.0`	The HTTP version to use for requests. Valid values are `1.0` and `1.1`.	httpVersion

Property	M	C	Default	Description	mqsiapplybaroverride command property
Enable HTTP/1.1 keep-alive	No	Yes	Selected (if HTTP version is 1.1)	Use HTTP/1.1 Keep-Alive.	enableKeepAlive
HTTP method	No	No	POST	The HTTP method. Valid values are POST, GET, PUT, DELETE, and HEAD. By default, the HTTPRequest node uses the HTTP POST method when it connects to the remote web server. HEAD is used to determine whether a service is available - for example, by a Network Dispatcher trying to work out which servers are available - and sends back the correct headers (including content-length) but no body data.	
Use compression	No	Yes	None	This property controls whether the content of the HTTP request is compressed. You can choose a value from none, gzip, zlib (deflate), and deflate. If the request is compressed, the Content-Encoding header is set to indicate that the content is compressed. zlib (deflate) represents RFC 1950 + RFC 1951 combined. deflate represents RFC 1951 only.	requestCompressionType

The HTTPRequest node SSL properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Protocol	No	Yes	SSL	The SSL protocol to use when making an HTTPS request.	protocol
Allowed SSL ciphers	No	Yes		A comma-separated list of ciphers to use when making an SSL request. The default value of an empty string means use all available ciphers.	allowedCiphers
Perform hostname checking	No	Yes	No	This property specifies whether the host name of the server that is receiving the request must match the host name in the SSL certificate.	hostnameChecking

The HTTPRequest node Response Message Parsing properties are described in the following table.

Property	M	C	Default	Description
Message domain	No	No	BLOB	The domain that is used to parse the message that is received from the web service. If the field is blank then the default is BLOB.
Message set	No	No		The name or identifier of the message set in which the response message is defined. If you set this property, and then update the project dependencies to remove this message set reference, a warning is issued. Either update the Message set property, or restore the reference to this message set project.
Message type	No	No		The name of the response message.

Property	M	C	Default	Description
Message format	No	No		The name of the physical format of the response message.

The HTTPRequest node Parser Options properties are described in the following table.

Property	M	C	Default	Description
Parse timing	No	No	`On Demand`	This property controls when a response message is parsed. Valid values are `On Demand`, `Immediate`, and `Complete`. For a full description of this property, see "Parsing on demand" on page 4173.
Build tree using XML schema data types	No	No	Cleared	This property controls whether the XMLNSC parser creates syntax elements in the message tree with data types taken from the XML schema. You can select this property only if you set the `Validate` property on the **Validation** tab to `Content` or `Content and Value`.
Use XMLNSC compact parser for XMLNS domain	No	No	Cleared	This property controls whether the XMLNSC Compact Parser is used for messages in the XMLNS Domain. If you set this property, the response message data is displayed under XMLNSC in nodes that are connected to the output terminal when the input MQRFH2 header or Response Message Parsing properties Domain is XMLNS.
Retain mixed content	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters mixed text in a response message. If you select the check box, elements are created for mixed text. If you clear the check box, mixed text is ignored and no elements are created.
Retain comments	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters comments in a response message. If you select the check box, elements are created for comments. If you clear the check box, comments are ignored and no elements are created.
Retain processing instructions	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters processing instructions in a response message. If you select the check box, elements are created for processing instructions. If you clear the check box, processing instructions are ignored and no elements are created.
Opaque elements	No	No	Blank	This property is used to specify a list of elements in the response message that are to be opaquely parsed by the XMLNSC parser. Opaque parsing is performed only if validation is not enabled (that is, if the `Validate` property is set to `None`); entries that are specified in `Opaque Elements` are ignored if validation is enabled.

The HTTPRequest node Error Handling properties are described in the following table.

Property	M	C	Default	Description
Replace input with error	No	No	Selected	If you select this check box, the input message content is replaced by the error message content. If you clear this check box, you must specify `Error message location`.

Property	M	C	Default	Description
Error message location	Yes	No	`OutputRoot`	The start location at which the parsed elements from the web service error bit stream are stored. This property takes the form of an ESQL field reference.

The HTTPRequest node Advanced properties are described in the following table.

Property	M	C	Default	Description	**`mqsiapplybaroverride` command property**
Use whole input message as request	No	No	Selected	If you select this check box, the whole input message body is to be passed to the web service. If you clear this check box, you must select `Request message location in tree`.	
Request message location in tree	Yes	No	`InputRoot`	The start location from which the bit stream is created for sending to the web service. This property takes the form of an ESQL field reference.	
Replace input message with web-service response	No	No	Selected	If you select this check box, the web service response message replaces the copy of the input message as the content of the output message that is created. If you clear this check box, you must select `Response message location in tree`.	
Response message location in tree	Yes	No	`OutputRoot`	The start location at which the parsed elements from the web service response bit stream are stored. This property takes the form of an ESQL field reference.	
Generate default HTTP headers from input	No	No	Selected	If you select this check box, an HTTPRequestHeader is generated. If you clear this check box, a valid HTTPRequestHeader must exist in the input message.	
Accept compressed responses by default	No	Yes	Cleared	This property indicates whether the request node handles compressed responses by default. If the request header does not contain an Accept-Encoding header and this option is selected, the node sets the Accept-Encoding header to "gzip, deflate", and any compressed response that is received is decompressed by the node. If the message propagated to the Request node includes an Accept-Encoding header, the message flow or client application should handle any compressed response. Therefore selecting this option has no effect in that case.	acceptCompressedResponses

The HTTPRequest node Validation properties are described in the following table.

For a full description of these properties see "Validation properties" on page 4169.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Validate	No	Yes	None	This property controls whether validation takes place. Valid values are None, Content and Value, Content, and Inherit.	validateMaster
Failure action	No	No	Exception	This property controls what happens if validation fails. You can set this property only if you set Validate to Content or Content and Value. Valid values are User Trace, Local Error Log, Exception, and Exception List.	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Local environment overrides:
You can dynamically override set values in the local environment in the same way as setting values in other elements of a message. The following values can be set under LocalEnvironment.Destination.HTTP.

Setting	Description
RequestURL	Overrides the Web service URL property on the node. For example: `SET OutputLocalEnvironment.Destination.HTTP.RequestURL = 'http://ibm.com/abc/';`
Timeout	Overrides the Request timeout (sec) property on the node. For example: `SET OutputLocalEnvironment.Destination.HTTP.Timeout = 42;`
TimeoutMillis	Overrides the Request timeout (sec) property on the node. For example: `SET OutputLocalEnvironment.Destination.HTTP.TimeoutMillis = 5000;` This property defines the timeout in milliseconds. The value of TimeoutMillis overrides the value for Timeout if both values are set.
ProxyURL	Overrides the HTTP(S) proxy location property on the node. For example: `SET OutputLocalEnvironment.Destination.HTTP.ProxyURL = 'my.proxy';`
RequestLine.RequestURI	Overrides the RequestURI, which is the path after the URL and port. For example: `SET OutputLocalEnvironment.Destination.HTTP.RequestLine.RequestURI = '/abc/def';`
RequestLine.HTTPVersion	Overrides the HTTP version property on the node. For example: `SET OutputLocalEnvironment.Destination.HTTP.RequestLine.HTTPVersion = 'HTTP/1.1';`
KeepAlive	Overrides the Enable HTTP/1.1 keep-alive property on the node. For example: `SET OutputLocalEnvironment.Destination.HTTP.KeepAlive = TRUE;`
RequestLine.Method	Overrides the HTTP method property on the node. For example: `SET OutputLocalEnvironment.Destination.HTTP.RequestLine.Method = 'GET';`

Setting	Description
SSLProtocol	Overrides the SSLProtocol. For example: `SET OutputLocalEnvironment.Destination.HTTP.SSLProtocol = 'TLS';` Valid values are: SSL, SSLv3, TLS, TLSv1, TLSv1.1, TLSv1.2, SSL_TLS, and SSL_TLSv2

SSLCiphers	Overrides the Allowed SSL Ciphers property on the node. For example:
	`SET OutputLocalEnvironment.Destination.HTTP.SSLCiphers =`
	`'SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA';`

ProxyConnectHeaders	Specifies additional headers that are used if the outbound request is an SSL connection through a proxy. These additional headers are sent with the initial CONNECT request to the proxy. For example, you can send proxy authentication information to a proxy server when you are using SSL. You can send multiple headers but each one must be separated by a carriage return and a line feed (ASCII 0x0D 0x0A), in accordance with RFC2616; for example:
	`DECLARE CRLF CHAR CAST(X'0D0A' AS CHAR CCSID 1208);`
	`SET OutputLocalEnvironment.Destination.HTTP.ProxyConnectHeaders =`
	`'Proxy-Authorization: Basic Zm5lcmJsZTpwYXNzd29yZA==' || CRLF ||`
	`'Proxy-Connection: Keep-Alive' || CRLF;`
	
	This setting is used only if the request is an SSL request through a proxy server. To send proxy authentication information for a non-SSL request, specify the individual headers in the HTTPRequestHeader folder, as shown in the following example:
	`SET OutputRoot.HTTPRequestHeader."Proxy-Authorization" =`
	`'Basic Zm5lcmJsZTpwYXNzd29yZA==';`
	`SET OutputRoot.HTTPRequestHeader."Proxy-Connection" = 'Keep-Alive';`

UseFolderMode	Sets the UseFolderMode. Use for bitstream generation; for certain parsers this changes the output bitstream. For example: `SET OutputLocalEnvironment.Destination.HTTP.UseFolderMode = TRUE;`

QueryString	Allows the setting of outbound query string parameters. Each parameter must be set individually. For example:
	`SET OutputLocalEnvironment.Destination.HTTP.QueryString.param1 = 'my"Value"1';`
	`SET OutputLocalEnvironment.Destination.HTTP.QueryString.param2 = 'my"Value"2';`
	
	The above ESQL results in the following query string being encoded (according to http://tools.ietf.org/html/rfc3986) and sent with the outbound request:
	`?param1=my%22Value%221&param2= my%22Value%222`
	
	If the destination URL already has one or more query parameters, additional parameters specified here are appended to the existing list.

QueryStringCCSID	Specifies that, before encoding, the query string parameters must be converted into a character set other than the default, which is UTF-8. Any query string parameters are first converted into the specified CCSID before the resulting string is encoded, according to RFC3986. For example: `SET OutputLocalEnvironment.Destination.HTTP.QueryStringCCSID = 943;` The above ESQL results in any QueryString parameters being converted to the 943 code page before they are encoded. Note: Any query string parameters must contain the data in unicode.

Compression	Overrides the Use compression property on the node. For example:
	`SET OutputLocalEnvironment.Destination.HTTP.Compression =`
	`'gzip';`

Working with WrittenDestination data

After the request has been made, the WrittenDestination folder in the local environment is updated with the URI to which the request was sent and compression details (if used). A WrittenDestination for an HTTPRequest node has the following format, with Compression present only if it is used:

```
WrittenDestination = (
   HTTP  = (
      RequestURL = 'http://127.0.0.1:7800/HTTPFLOW' (CHARACTER)
      Compression = (
        OriginalSize = 53 (INTEGER)
        CompressedSize = 71 (INTEGER)
      )
   )
)
```

Related concepts:

"Specifying namespaces in the Message Type property" on page 1208
When using the MRM domain, the *Message Type* property is used to specify the name of the message.

"Parsers" on page 1072
A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from the internal message tree representation.

"ESQL field references" on page 2381
An ESQL field reference is a sequence of period-separated values that identify a specific field (which might be a structure) within a message tree or a database table.

"EndpointLookup node output" on page 1894
Use an EndpointLookup node to retrieve the endpoint addresses for WSDL service definitions held in WebSphere Service Registry and Repository (WSRR).

"Using compression with HTTP and SOAP nodes" on page 1597
You can configure HTTP and SOAP nodes to use HTTP compression and decompression when sending and receiving messages.

Related tasks:

"Configuring authentication with HTTP basic authentication" on page 451
Use a security profile to configure HTTP basic authentication in the HTTPRequest or SOAPInput nodes.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

"Accessing the local environment tree" on page 2463
The local environment tree has its own correlation name, LocalEnvironment, and you must use this name in all ESQL statements that refer to or set the content of this tree.

"Validating messages" on page 1478
The broker provides validation based on the message model for predefined messages.

Related reference:

"Parsing on demand" on page 4173
On-demand parsing, referred to as partial parsing, is used to parse an input message bit stream only as far as is necessary to satisfy the current reference. The parsers that can perform partial parsing of input messages are the JSON, MRM, XML, XMLNS, and XMLNSC parsers.

"Validation properties" on page 4169
You can control validation by setting properties on the Validate and Parser Options tabs for the nodes that are listed in the following table.

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"HTTPInput node" on page 4474
Use the HTTPInput node to receive an HTTP message from an HTTP client for processing by a message flow.

"HTTPReply node" on page 4484
Use the HTTPReply node to return a response from the message flow to an HTTP client. This node generates the response to an HTTP client from which the input message was received by the HTTPInput node, and waits for confirmation that it has been sent.

"SOAPRequest node" on page 4828
Use the SOAPRequest node to send a SOAP request to the remote Web service.

"ASBITSTREAM function" on page 5224
The ASBITSTREAM field function generates a bit stream for the subtree of a given field according to the rules of the parser that owns the field.

"EndpointLookup node" on page 4407
Use the EndpointLookup node to retrieve service endpoint information held in the WebSphere Service Registry and Repository (WSRR). The entities that match the specified search criteria are stored in the local environment. Depending on the node configuration, the Web service URL destination used by the SOAP and HTTP request nodes can also be set. The input message is not modified.

IMSRequest node

Use the IMSRequest node to send a request to run a transaction on a local or remote IMS system, and wait for a response. IMS Connect must be configured and running on the IMS system.

This topic contains the following sections:
- "Purpose"
- "Using the IMSRequest node in a message flow" on page 4505
- "Terminals and properties" on page 4506

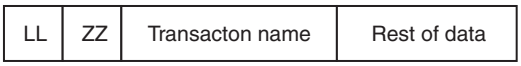
Purpose:
The following example illustrates a situation in which you would use an IMSRequest node.

WebSphere Message Broker can be used to expose an existing target IMS banking application as a Web Service. For example, the IMS application provides transactions that operate on a database that contains information about customers' bank accounts. In this example, the Web service consumer sends a SOAP message across HTTP to WebSphere Message Broker and synchronously waits for the response. The WebSphere Message Broker message flow transforms the SOAP message to IMS format (including the LLZZ and transaction code fields), then sends that bit stream to IMS. The message flow waits for a response. IMS schedules the destination program and queues the request data for that program. The target program accesses the customer account database, builds a response

message that consists of the account statement, and returns it to the WebSphere Message Broker message flow. The message flow transforms the IMS format to a SOAP format and sends that SOAP response back across HTTP to the Web service consumer.

The IMSRequest node is contained in the **IMS** drawer of the message flow node palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using the IMSRequest node in a message flow:
The IMSRequest node sends requests to IMS by using IMS Connect. The node takes the incoming message's bit stream and sends it to IMS. It then receives a bit stream, which it passes to the response parser. The bit stream that is sent to IMS must conform to the format that is shown in the following diagram:



The message flow that contains the IMSRequest node ensures that the message that is received by the IMSRequest node has this structure where:

- *LLZZ* is a four-byte field. The first two bytes indicate the length of the bit stream, and the other two bytes are reserved for use by IMS.
- For request segments, the transaction code must follow. The transaction code can contain up to eight characters; if it contains less than eight characters, the transaction code must be delimited by a space. The response segments do not need to have the transaction name, but an IMS program can add it.
- The rest of the data comprises anything else that the IMS program needs.

The IMS program produces many messages. You can receive all messages as a single transmission bit stream, or you can receive them separately. Each message can contain multiple segments; all segments for each message are returned at the same time.

The IMSRequest node has two modes of operation, which you specify by selecting or clearing the `Use connection properties defined on node` check box. If you select the check box, all properties are taken from the node by using the following properties in the node connection details section:

- `Hostname`
- `Port number`
- `Data store name`

If you clear the `Use connection properties defined on node` check box, all connection details are retrieved from the configurable services. However, if the `Security identity` property is set, the security identity on the configurable service is ignored and the value of the node property is used instead.

The IMSRequest node can also use an identity that is present on an input message, and propagate it to IMS, by using the `Propagate` property on the security profile that is defined for the node. For more information, see “Propagating security credentials to IMS” on page 2144.

View the following sample to see how to use this node:

- IMS Synchronous Request

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Using configurable services for IMS nodes

You can configure IMS nodes to get connection details from a configurable service. For details about creating, changing, reporting, and deleting the configurable services, see "Changing connection information for the IMSRequest node" on page 732.

Terminals and properties:
When you have put an instance of the IMSRequest node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The IMSRequest node terminals are described in the following table.

Terminal	Description
In	The input terminal that receives the message that triggers the node.
Out	The output terminal to which the node sends a message after it has been received from the external resource. The message is sent to the terminal unchanged, except for some added status information.
Failure	If an error occurs in the IMSRequest node, the message is sent to the Failure terminal.
Timeout	The output terminal to which the message is sent if a timeout occurs. The input message is propagated to this terminal with an exception list that describes the timeout. If the Timeout terminal is not connected and a timeout occurs, the message is routed to the Failure terminal. A timeout can occur in either of the following situations: • The IMS program has not responded by the time the execution timeout has expired. The execution timeout is configured by using the `Timeout waiting for a transaction to be executed` property on the IMSRequest node. • WebSphere Message Broker has not received the response across the TCP/IP network by the time the socket timeout expires. You can configure the socket timeout on the configurable service.

The following tables describe the node properties. The columns headed M indicate whether the property is *mandatory* (marked with an asterisk on the panel if you must enter a value when no default is defined); the columns headed C indicate whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The IMSRequest node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, IMSRequest	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The IMSRequest node Basic properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Use connection properties defined on node	No	Yes	Selected	If you select this check box, the connection properties that are defined on the node are used instead of a configurable service and security identity that are defined on the broker. If you clear this check box, you must set the `Configurable service` and `Security identity` properties.	
Hostname	Yes	Yes		The IP address or host name of the computer that is running the target IMS Connect system. This property is mandatory if the `Use connection properties defined on node` check box is selected and can be set only if the `Use connection properties defined on node` check box is selected.	hostname
Port number	Yes	Yes	`0`	The port number on which IMS Connect is listening for TCP/IP connections. You can obtain the port number from the IMS Connect job log on the IMS system. This property is mandatory only if the `Use connection properties defined on node` check box is selected and can be set only if the `Use connection properties defined on node` check box is selected.	portNumber
Data store name	Yes	Yes		The name of the data store that IMS Connect is using. This value must match the ID parameter of the Datastore statement that is specified in the IMS Connect configuration member. This name also serves as the XCF member name for IMS during internal XCF communications between IMS Connect and IMS OTMA. You can obtain the data store name from the IMS Connect job log on the IMS system. This property is mandatory only if the `Use connection properties defined on node` check box is selected and can be set only if the `Use connection properties defined on node` check box is selected.	dataStoreName
Timeout waiting for a transaction to be executed	No	No	`60`	The time (in seconds) that the node waits for IMS to process a transaction. If IMS fails to process a transaction in this time, the node issues an exception, but the connection is not closed. You can set this property only if the `Use connection properties defined on node` check box is selected. If the check box is cleared, the `ExecutionTimeoutSec` property on the configurable service is used.	
Configurable service	Yes	Yes		The configurable service from which to get the connection details. All connection details are obtained from the configurable service, except for security information, which is obtained from the `Security identity` property. This property is mandatory if the `Use connection properties defined on node` check box is cleared, and can be set only if the `Use connection properties defined on node` is cleared.	configurableService

Property	M	C	Default	Description	**mqsiapplybaroverride command property**
Security identity	No	Yes	An empty string	The security identity to look up in the broker to get the user name and password to use. You use the **mqsisetdbparms** command to set the security identity on the broker. The default value for this property is an empty string, which signifies that the user ID and password are not passed to IMS Connect.	securityIdentity

The IMSRequest node Advanced properties are described in the following table.

Property	M	C	Default	Description
Commit mode	Yes	No	`1: SEND_THEN_COMMIT`	The commit mode to use when processing IMS transactions. Available values are: • `1: SEND_THEN_COMMIT`: (the default value) The transaction is processed using commit mode 1 in IMS. The request transaction is run and data is sent back to the node. After the node acknowledges that it has received the response, the transaction is committed. You cannot process the response before it is committed. The node sends the response after the acknowledgment is sent to IMS. • `0: COMMIT_THEN_SEND`: The transaction is processed using commit mode 0 in IMS. The request transaction is run and committed before data is sent back to the node. The node waits for all response messages to be returned before it continues processing.
Sync level	Yes	No	`1: CONFIRM`	The synchronization level to use when processing IMS transactions. If `Commit mode` is set to 0, the `Sync level` is automatically set to `1: CONFIRM`. If `Commit mode` is set to 1, the `Sync level` property can have either of the following values: • `1: CONFIRM`: (the default value) The node sends an acknowledgment to IMS after it receives the replies. The node then sends the response after the acknowledgment is sent to IMS. If you set the `Sync level` to `1: CONFIRM`, the IMS program is blocked until the IMSRequest node acknowledges the transaction output, which might affect performance. • `0: NONE`: The node does not send any acknowledgments. This value is applicable only when `Commit mode` is set to 1. Typically, `Sync level` can be set to `0: NONE` for read-only types of interactions, such as queries, which do not need an acknowledgment. However, for critical transactions that involve updates and deletions, it is important to be able to acknowledge the output from IMS. If the acknowledgment is not received (for example, because of a connection failure between WebSphere Message Broker and IMS Connect), the transaction is backed out, avoiding the need for a compensation transaction.

The IMSRequest node Request properties are described in the following table.

Property	M	C	Default	Description
Data Location	Yes	No	`$Body`	The location in the incoming message tree from which data is retrieved to form the request that is sent from the IMSRequest node to IMS. The default value, `$Body`, represents the incoming message body. You can enter any XPath or ESQL expression that defines the location of the message tree to serialize and send to IMS.

The IMSRequest node Result properties are described in the following table.

Property	M	C	Default	Description
Output data location	No	No	`$OutputRoot`	The message tree location to which the IMSRequest node copies the response message tree in the outgoing message assembly. The default value, `$OutputRoot`, replaces the incoming message with the response.
Copy local environment	No	No	Selected	This property controls whether to copy the incoming local environment or propagate the incoming local environment. By default, this check box is selected, which signifies that the local environment is copied so that the incoming local environment is preserved. The additions to the local environment are visible only to nodes downstream of this node. If this check box is cleared, the incoming local environment is used for the outgoing message. Any modifications that are made to the local environment by this node are visible to both downstream and upstream nodes after this node has completed.

The IMSRequest node Response Message Parsing properties are described in the following table.

Property	M	C	Default	Description	**`mqsiapplybaroverride` command property**
Message domain	No	No		The domain to use to parse the message from the external resource's supplied bit stream.	
Message set	No	No	Set automatically	The name of the message set in which the incoming message is defined. If you set this property, then subsequently update the project dependencies to remove this message set reference, a warning is issued. Either update the `Message set` property, or restore the reference to this message set project.	
Message type	No	No		The name of the response message.	
Message format	No	No		The name of the physical format of the response message.	
Message coded character set ID	Yes	No	`EBCDIC (500)`	The ID of the coded character set that is used to interpret bytes of the data that is being read. Valid values are `EBCDIC (500)` and `Broker System Default`.	messageCodedCharSetIdProperty
Message encoding	Yes	No	`Big Endian, with S390 Floating Point (785)`	The encoding scheme for numbers and large characters that is used to interpret bytes of the data that is being read. Valid values are: • Little Endian, with IEEE Floating Point (546) • Big Endian, with IEEE Floating Point (273) • Big Endian, with S390 Floating Point (785) • Broker System Determined For more information about encoding, see “Data conversion” on page 1151.	messageEncodingProperty

The IMSRequest node Parser Options properties are described in the following table.

Property	M	C	Default	Description
Parse timing	Yes	No	On Demand	This property controls when a response message is parsed. Valid values are On Demand, Immediate, and Complete. For a full description of this property, see "Parsing on demand" on page 4173.
Build tree using XML schema data types	Yes	No	Cleared	This property controls whether the XMLNSC parser creates syntax elements in the message tree with data types taken from the XML Schema. You can select this property only if you set the Validate property on the **Validation** tab to Content or Content and Value.
Use XMLNSC compact parser for XMLNS domain	Yes	No	Cleared	This property controls whether the XMLNSC Compact Parser is used for messages in the XMLNS Domain. If you set this property, the response message data appears under XMLNSC in nodes that are connected to the output terminal when the input MQRFH2 header or Response Message Parsing properties Domain is XMLNS.
Retain mixed content	Yes	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters mixed text in a response message. If you select the check box, elements are created for mixed text. If you clear the check box, mixed text is ignored and no elements are created.
Retain comments	Yes	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters comments in a response message. If you select the check box, elements are created for comments. If you clear the check box, comments are ignored and no elements are created.
Retain processing instructions	Yes	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters processing instructions in a response message. If you select the check box, elements are created for processing instructions. If you clear the check box, processing instructions are ignored and no elements are created.
Opaque elements	No	No	Blank	This property is used to specify a list of elements in the response message that are to be opaquely parsed by the XMLNSC parser. Opaque parsing is performed only if validation is not enabled (that is, if Validate is None); entries that are specified in Opaque Elements are ignored if validation is enabled.

The IMSRequest node Validation properties are described in the following table.

For a full description of these properties see "Validation properties" on page 4169.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Validate	Yes	Yes	None	This property controls whether validation takes place. Valid values are None, Content and Value, Content, and Inherit.	validateMaster

Property	M	C	Default	Description	mqsiapplybaroverride command property
Failure action	Yes	No	Exception	This property controls what happens if validation fails. You can set this property only if you set Validate to Content or Content and Value. Valid values are User Trace, Local Error Log, Exception, and Exception List.	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"IBM Information Management System (IMS)" on page 2129
IMS is a message-based transaction manager and hierarchical-database manager for z/OS. External applications can use transactions to interact with applications that run inside IMS.

"IMS nodes" on page 2130
WebSphere Message Broker message flows use IMS nodes to call programs that are running in IMS.

Related tasks:

"Preparing the environment for IMS nodes" on page 731
Before you can use the IMS nodes, you must set up the broker runtime environment so that you can access the IMS system.

"Changing connection information for the IMSRequest node" on page 732
You can create a configurable service that the IMSRequest node or message flow refers to at run time for connection information, instead of defining the connection properties on the node or the message flow. The advantage being that you can change the host name, performance, and security values without needing to redeploy your message flow.

"Resolving problems when you use IMS nodes" on page 3419
Advice for dealing with common problems that can arise when you develop message flows that contain IMS nodes.

Related reference:

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

Input node

Use the Input node as an In terminal for an embedded message flow (a subflow).

This topic contains the following sections:

Purpose:
You can use a subflow for a common task that can be represented by a sequence of message flow nodes. For example, you can create a subflow to increment or decrement a loop counter, or to provide error processing that is common to a number of message flows.

You must use an Input node to provide the In terminal to a subflow; you cannot use a standard input node (a built-in input node such as MQInput, or a user-defined input node).

When you have started your subflow with an Input node, you can connect it to any In terminal on any message flow node, including an Output node.

You can include one or more Input nodes in a subflow. Each Input node that you include provides a terminal through which to introduce messages to the subflow. If you include more than one Input node, you cannot predict the order in which the messages are processed through the subflow.

The Input node is contained in the **Construction** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

When you select and include a subflow in a message flow, it is represented by the following icon:

When you include the subflow in a message flow, this icon shows a terminal for each Input node that you include in the subflow, and the name of the terminal (which you can see when you hold the mouse pointer over it) matches the name of that instance of the Input node. Give your Input nodes meaningful names that you can recognize easily when you use their corresponding terminal on the subflow node in your message flow.

Using this node in a message flow:
Look at the following sample to see how to use this node:
- Error Handler

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Terminals and properties:
When you have put an instance of the Input node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view.

The Input node terminals are described in the following table.

Terminal	Description
Out	The input terminal that delivers a message to the subflow.

The following table describes the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Input node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, Input.	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Subflows" on page 1030
You can include subflows in your message flows in the same way as you include built-in or user-defined nodes. You can also connect subflows to other nodes in the same way.

Related tasks:

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

Related reference:

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

"Output node" on page 4626
Use the Output node as an out terminal for an embedded message flow (a subflow).

JavaCompute node

Use the JavaCompute node to work with messages by using the Java language.

This topic contains the following sections:

Purpose:
By using this node, you can complete the following tasks:

- Use Java to examine an incoming message and, depending on its content, propagate it unchanged to one of the two output terminals of the node. The node behaves in a similar way to a Filter node, but uses Java instead of ESQL to determine which output terminal to use.
- Use Java to change part of an incoming message and propagate the changed message to one of the output terminals.
- Use Java to create and build a new output message that is totally independent of the input message.

The Java code that is used by the node is stored in an Eclipse Java project.

The JavaCompute node is contained in the **Transformation** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:
The JavaCompute node uses the same execution model as Java user-defined nodes and therefore the restrictions and assumptions associated with Java user-defined nodes also apply to Java code developed for JavaCompute nodes; see "Creating a message processing or output node in Java" on page 3062. Only one instance of the JavaCompute node is created regardless of the number of threads running against the flow (either as a result of additional instances or multiple input nodes). Therefore all of your user Java code must be threadsafe and reentrant. For more information, see "User-defined extensions execution model" on page 2981 and "Threading considerations for user-defined extensions" on page 2978.

Double-click the JavaCompute node to open the New Java Compute Node Class wizard. The wizard guides you through the creation of a new Java project and a Java class that contains some skeleton code. This skeleton code is displayed in a Java editor. For more information about creating Java code for a JavaCompute node, and for examples of the skeleton code or template that are provided, see "Creating Java code for a JavaCompute node" on page 2629. If it is not the first time that you have double-clicked the node, the Java code is displayed.

The MbJavaComputeNode class contains two methods that you can override: `onInitialize()` and `onDelete()`. If you want the node to perform cleanup operations, for example closing sockets, include an implementation of the onDelete method:

```
public void onDelete()
{
  // perform node cleanup if necessary
}
```

The `onInitialize()` method is called either during deployment or on broker startup. During deployment, the method is called before deployment has been committed. Other nodes can throw exceptions during their initialization; therefore, the flow will fail to deploy, even if the `onInitialize()` method for this node works. If your `onInitialize()` method throws an exception, the flow either fails to deploy, or fails to start. Therefore, complete tasks that will always work or always fail during the `onInitialize()` method. The broker does not try to start the flow again until the broker is restarted. If you need to initialize an external connection that might need to be retried, do so on the first message through the flow so that the flow can retry the transaction as necessary.

Look at the following sample to see how to use this node.

- JavaCompute Node

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

For instructions about the tasks that you can complete with a JavaCompute node, see the following topics:

- Manipulating message body data
- Manipulating other parts of the message tree
- Accessing broker properties
- Accessing user-defined properties
- "Adding keywords to JAR files" on page 2660
- Interacting with databases
- "Calling an Enterprise Java Bean" on page 2666
- Handling exceptions
- Logging errors

Specifying Java:
Code Java statements to customize the behavior of the JavaCompute node. For example, you can customize the node to create a new output message or messages, using input message or database content (unchanged or modified), or new data. For example, you might want to modify a value in the input message by adding a value from a database, and store the result in a field in the output message.

Code the Java statements that you want in a Java file that is associated with the JavaCompute node.

If a Java file does not already exist for this node, right-click the JavaCompute node and click **Open Java** to create and open a new Java file in the Editor view. If the file exists already, click **Browse** beside the `Java Class` property to display the JavaCompute Node Type Selection window. When you type at least one character in the **Select** field, matching Java classes are listed. You can use the asterisk (*) to represent any character as part of a search string; for example, *a*b*. Select the appropriate Java class and click **OK**.

Restriction: Do not try to create another instance of a JavaCompute node from Java code; this is not supported.

Terminals and properties:
When you have put an instance of the JavaCompute node into a message flow, you can configure it; see . To associate an instance of a JavaCompute node with a Java class, configure the node's properties. The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The JavaCompute node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the input message is routed if a failure is detected during the computation. (Even if the `Validate` property is set, messages that are propagated to the Failure terminal of the node are not validated.)
Out	The output terminal to which the transformed message is routed.
Alternate	An alternative output terminal to which the transformed message can be routed, instead of to the Out terminal.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Description properties of the JavaCompute node are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type: JavaCompute	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The Basic properties for the JavaCompute node are described in the following table.

Property	M	C	Default	Description
Java class	Yes	No	None	The name of the Java class that is used in this node. This name must be displayed in the list of Java classes that are available in the project references for the message flow project. To select a file that already exists, click **Browse**. When you type at least one character in the **Select** field, matching Java classes are listed. You can use the asterisk (*) to represent any character as part of a search string; for example, *a*b*. Select the appropriate Java class and click **OK**.

Property	M	C	Default	Description
Java classloader service	No	Yes	None	The name of the JavaClassLoader configurable service that is used in this node. If no name is specified, and a if JavaClassLoader configurable service with the same name as the execution group exists, it is used in this node. Otherwise, no JavaClassLoader configurable service is used. All the nodes in an execution group that specify the same JavaClassLoader configurable service use the same instance of the classloader. Consequently all the nodes use the same in-memory version of the classes, including access to the same static variables. For more information, see "JavaCompute node classloading" on page 2635.

The Parser Options properties for the JavaCompute node are described in the following table.

Property	M	C	Default	Description
Use XMLNSC Compact Parser for XMLNS Domain	No	No	Cleared	Setting this property causes the outgoing MQRFH2 to specify the XMLNS instead of XMLNSC parser, allowing an external application to remain unchanged. If outgoing messages do not contain MQRFH2 headers, this property has no effect.

The Validation properties of the JavaCompute node are described in the following table.

Set the validation properties to define how the message that is produced by the JavaCompute node is validated. These properties do not cause the input message to be validated. It is expected that, if such validation is required, the validation has already been performed by the input node or a preceding validation node. For more details, see "Validating messages" on page 1478 and "Validation properties" on page 4169.

Property	M	C	Default	Description	`mqsiapplybaroverride` command property
Validate	No	Yes	None	This property controls whether validation takes place, and what part of the message is validated. Valid values are None, Content and Value, Content, and Inherit.	validateMaster
Failure action	No	No	Exception	This property controls what happens if a validation failure occurs. You can set this property only if `Validate` is set to Content or Content and Value. Valid values are User Trace, Local Error Log, Exception, and Exception List.	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related tasks:

"Developing Java" on page 2628
When you use the JavaCompute node, you customize it to determine the exact processing that it provides.

"Creating Java code for a JavaCompute node" on page 2629
Use these instructions to associate Java code with your JavaCompute node.

"Writing Java" on page 2638
When you create a message flow, you include input nodes that receive messages and, optionally, output nodes that send out new or updated messages. If the message processing requires it, you can include other nodes after the input node that are customized in Java to complete the actions that your applications need.

"Accessing broker properties from the JavaCompute node" on page 2658
For each broker, WebSphere Message Broker maintains a set of properties. You can access some of these properties from your Java programs. It can be useful, during the run time of your code, to have real-time access to details of a specific node, flow, or broker.

"Validating messages" on page 1478
The broker provides validation based on the message model for predefined messages.

"Creating destination lists" on page 1477
Create a list of destinations to indicate where a message is sent.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Configuring transactionality for message flows" on page 1290
A message flow runs in a single transaction, which is started when data is received by an input node, and can be committed or rolled back when all processing has completed.

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

Related reference:

"Validation properties" on page 4169
You can control validation by setting properties on the Validate and Parser Options tabs for the nodes that are listed in the following table.

"Simple type logical value constraints" on page 5450
The properties, and their permissible values, vary according to the object type.

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

Related information:

Java user-defined extensions API

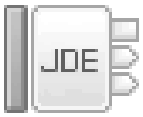
JDEdwardsInput node

Use the JDEdwardsInput node to interact with a JD Edwards EnterpriseOne server.

This topic contains the following sections:
- "Purpose"
- "Using this node in a message flow"
- "Terminals and properties" on page 4520

Purpose:
Use the JDEdwardsInput node to interact with JD Edwards EnterpriseOne applications. For example, a JDEdwardsInput node monitors a JD Edwards EnterpriseOne server for a specified event. When that event occurs, the JDEdwardsInput node generates a message tree that represents the business object with the new event details. The message tree is propagated to the Out terminal so that the rest of the message flow can use the data to update other systems, or audit the changes.

The JDEdwardsInput node is contained in the **WebSphere Adapters** drawer of the message flow node palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:



Using this node in a message flow:
To enable function that becomes available in WebSphere Message Broker fix packs, use the **-f** parameter on the **mqsichangebroker** command. For more information, see "**mqsichangebroker** command" on page 3723.

Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment (as described in "Adding external software dependencies for JD Edwards EnterpriseOne" on page 2090).

To function correctly, the JDEdwardsInput node needs an adapter component (see "Connecting to an EIS by using the Adapter Connection wizard" on page 2037), which you set by using the `Adapter component` node property, and business object definitions, which are stored in the message set that you reference from the node. For this reason, you must provide a message set. By default, the message that is propagated from the JDEdwardsInput node is in the DataObject domain, therefore the `Message domain` property is set to DataObject. You cannot specify a different domain. The message type is detected automatically by the node.

The JDEdwardsInput node populates the route to label destination list with the name of the method binding. If you add a RouteToLabel node to the message flow after the JDEdwardsInput node, the RouteToLabel node can use the name of the method binding to route the message to the correct part of the message flow for processing.

You can deploy only one input node that uses a particular adapter component to an execution group, but you can deploy many input nodes that use different adapter components to an execution group.

You can use the **mqsisetdbparms** command in the following format to configure an account name with a user name and password for the Adapter for JD Edwards EnterpriseOne.

```
mqsisetdbparms broker name -n adapter name -u user name -p password
```

For example:

```
mqsisetdbparms BRK1 -n eis::JDEdwardsCustomerInbound.inadapter -u jdedwardsuid -p ********
```

Using configurable services for JD Edwards nodes

JD Edwards nodes can get JD Edwards connection details from either the adapter component or a configurable service. By using a configurable service, you can change the connection details for an adapter without the need to redeploy the adapter. For more details about creating, changing, reporting, and deleting the configurable services for JD Edwards, see "Changing connection details for JD Edwards adapters" on page 724.

Terminals and properties:
When you have put an instance of the JDEdwardsInput node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view. If you double-click a JDEdwardsInput node, you open the Adapter Connection wizard. All mandatory properties for which you must enter a value (those properties that do not have a default value defined) are marked with an asterisk.

The SiebelInput node terminals are described in the following table.

Terminal	Description
Out	Business object events from the adapter are sent to the Out terminal.
Failure	If an error happens in the JDEdwardsInput node, the message is propagated to the Failure terminal. Information about the error, and business object events can also be propagated to the Failure terminal.
Catch	Business object events are sent to the Catch terminal if they cause an uncaught exception in the message flow. If the Catch terminal is not connected, the retry process is activated to handle the business object.

The following table describes the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The JDEdwardsInput node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, JDEdwardsInput.	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The JDEdwardsInput node Basic properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Primary adapter component	Yes	Yes		The name of the adapter component that contains configuration properties for the adapter. Either enter a name of an adapter file or click **Browse** to select an adapter file from the list of files that are available in referenced message set projects. When theJDEdwardsInput node receives data from the JD Edwards system, it associates that data with a method name, depending on the service operation name that is assigned to that type of data when you run the Adapter Connection wizard. The JDEdwardsInput node attempts to handle methods that are defined in the primary adapter. If the type of data that is received does not correspond to any of the methods that are defined in the primary adapter, the node can handle methods that are defined in matching secondary adapters that are deployed to the same execution group.	adapterComponent
Secondary adapter mode	No	Yes	None	Specifies whether the node can handle methods that are defined in secondary adapters. If you set the `Secondary adapter mode` property to `None`, the node handles only methods that are defined in the primary adapter. If the type of data that is received does not correspond to any of the methods that are defined in the primary adapter, a failure occurs. If you set this property to `All adapters in execution group`, the node can handle methods that are defined in any JD Edwards inbound adapters that are deployed to the same execution group.	secondaryAdapterMode

The JDEdwardsInput node Routing properties are described in the following table.

Property	M	C	Default	Description
Set destination list	No	No	Selected	This property specifies whether to add the method binding name to the route to label destination list. If you select this check box, the method binding name is added so that you can use a RouteToLabel node in the message flow after the JDEdwardsInput node.
Label prefix	No	No		The prefix to add to the method name when routing to label. Add a label prefix to avoid a clash of corresponding label nodes when you include multiple WebSphere Adapters input nodes in the same message flow. By default, there is no label prefix, so the method name and label name are identical.

The JDEdwardsInput node Input Message Parsing properties are described in the following table.

Property	M	C	Default	Description
Message domain	No	No	DataObject	The domain that is used to parse the incoming message. By default, the message that is propagated from the JDEdwardsInput node is in the DataObject domain. You cannot specify a different domain.

Property	M	C	Default	Description
Message set	Yes	No	Set automatically	The name of the message set in which the incoming message is defined. This field is set automatically from the Adapter component property. If you set this property, then subsequently update the project dependencies to remove this message set reference, a warning is issued. Either update the `Message set` property, or restore the reference to this message set project.
Message type	No	No		The name of the incoming message. The node detects the message type automatically. You cannot set this property.
Message format	No	No		The name of the physical format of the incoming message. You cannot set this property.

The JDEdwardsInput node Transactionality properties are described in the following table.

Property	M	C	Default	Description
Transaction mode	No	No	Yes	The transaction mode on this input node determines whether the rest of the nodes in the flow operate under sync point.

The Instances properties of the JDEdwardsInput node are described in the following table. For a full description of these properties, see "Configurable message flow properties" on page 4020.

Property	M	C	Default	Description	**mqsiapplybaroverride command property**
Additional instances pool	No	Yes	`Use Pool Associated with Message Flow`	The pool from which additional instances are obtained. • If you select `Use Pool Associated with Message Flow`, additional instances are obtained from the message flow value. • If you select `Use Pool Associated with Node`, additional instances are allocated from the additional instances for the node, based on the number specified in the `Additional instances` property.	componentLevel
Additional instances	No	Yes	`0`	The number of additional instances that the node can start if the `Additional instances pool` property is set to `Use Pool Associated with Node`. By default, no additional instances are given to the node.	additionalInstances

The JDEdwardsInput node Retry properties are described in the following table.

Property	M	C	Default	Description	**mqsiapplybaroverride command property**
Retry mechanism	No	No	Failure	This property specifies how retry processing is handled when a failure is rolled back to the JDEdwardsInput node. • If you select `Failure`, retry processing is not performed so you cannot set the remaining properties. • If you select `Short and long retry`, retry processing is performed first at the interval that is specified by the `Short retry interval` property, and if that is unsuccessful, it is then performed at the interval that is specified by the `Long retry interval` property.	

Property	M	C	Default	Description	**mqsiapplybaroverride command property**
Retry threshold	No	Yes	0	The maximum number of times that retry processing is performed for short retry.	retryThreshold
Short retry interval	No	Yes	0	The interval between short retry attempts.	shortRetryThreshold
Long retry interval	No	Yes	0	The interval between long retry attempts.	longRetryThreshold

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Overview of WebSphere Adapter for JD Edwards EnterpriseOne" on page 2023
With the WebSphere Adapter for JD Edwards EnterpriseOne, you can create integrated processes that include the exchange of information with a JD Edwards EnterpriseOne server, without special coding.

"DataObject parser and domain" on page 1114
Use the DataObject domain to parse and write messages for WebSphere Adapters and CORBA applications.

Related tasks:

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for JD Edwards EnterpriseOne" on page 2090
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Changing connection details for JD Edwards adapters" on page 724
JD Edwards nodes can get JD Edwards EnterpriseOne connection details from either the adapter component or a configurable service. By using configurable services, you can change the connection details for adapters without the need to redeploy the adapters. To pick up new values when a configurable service is created or modified, you must reload the broker or execution group to which the adapter was deployed, by using the **mqsistop** and **mqsistart** commands, or the **mqsireload** command.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Resolving problems when developing message flows with WebSphere Adapters nodes" on page 3428
Advice for dealing with common problems that can arise when you develop message flows that contain WebSphere Adapters nodes.

"Controlling the functional level of WebSphere Message Broker" on page 51
You can control the functional level of your broker to enable new functionality added in WebSphere Message Broker fix packs.

Related reference:

"WebSphere Adapter for JD Edwards EnterpriseOne properties" on page 4146
Reference information to refer to when you connect to a JD Edwards EnterpriseOne application.

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

"JDEdwardsRequest node"
Use the JDEdwardsRequest node to interact with a JD Edwards EnterpriseOne server.

JDEdwardsRequest node

Use the JDEdwardsRequest node to interact with a JD Edwards EnterpriseOne server.

This topic contains the following sections:

Purpose:

Use the JDEdwardsRequest node to discover JD Edwards EnterpriseOne business functions and XML lists. For example, a user might want to retrieve address details for a person where the record exists on a JD Edwards server.

The JDEdwardsRequest node can send and receive business data.

The JDEdwardsRequest node is contained in the **WebSphere Adapters** drawer of the message flow node palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:

To enable function that becomes available in WebSphere Message Broker fix packs, use the **-f** parameter on the **mqsichangebroker** command. For more information, see "**mqsichangebroker** command" on page 3723.

Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment (as described in "Adding external software dependencies for JD Edwards EnterpriseOne" on page 2090).

To function correctly, the JDEdwardsRequest node needs an adapter component (see "Connecting to an EIS by using the Adapter Connection wizard" on page 2037), which you set by using the `Adapter component` node property, and business

object definitions, which are stored in the message set that you reference from the node. For this reason, you must provide a message set. By default, the message that is propagated from the JDEdwardsRequest node is in the DataObject domain, therefore the `Message domain` property is set to DataObject. You cannot specify a different domain. The message type is detected automatically by the node.

The JDEdwardsRequest node supports local transactions by using the Local Transaction Manager of the broker, and global transactions by using the external syncpoint coordinator of the broker.

To effectively maintain the pool of connections to JD Edwards, you can set a connection timeout value on a configurable service. For more information, see "Configuring EIS connections to expire after a specified time" on page 726.

You can deploy several WebSphere Adapters request nodes that use the same adapter component to an execution group.

Use the **mqsisetdbparms** command in the following format to configure an account name with a user name and password for the Adapter for JD Edwards EnterpriseOne.

```
mqsisetdbparms broker name -n adapter name -u user name -p password
```

For example:

```
mqsisetdbparms BRK1 -n eis::JDEdwardsCustomerOutbound.outadapter -u jdedwardsuid -p ********
```

Look at the following sample to see how to use this node:

- JD Edwards Connectivity

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Using configurable services for JD Edwards nodes

JD Edwards nodes can get JD Edwards connection details from either the adapter component or a configurable service. By using a configurable service, you can change the connection details for an adapter without the need to redeploy the adapter. For more details about creating, changing, reporting, and deleting the configurable services for JD Edwards, see "Changing connection details for JD Edwards adapters" on page 724.

Terminals and properties:
When you have put an instance of the JDEdwardsRequest node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view. If you double-click a JDEdwardsRequest node, you open the Adapter Connection wizard. All mandatory properties for which you must enter a value (those properties that do not have a default value defined) are marked with an asterisk.

The JDEdwardsRequest node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the JDEdwardsRequest node. The JDEdwardsRequest node is driven by a message arriving on the In terminal.

Terminal	Description
Out	The output terminal from which the message tree is propagated.
Failure	If an error happens in the JDEdwardsRequest node, the message is propagated to the Failure terminal.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk on the panel if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The JDEdwardsRequest node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, for example, JDEdwardsRequest	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The JDEdwardsRequest node Basic properties are described in the following table.

Property	M	C	Default	Description	`mqsiapplybaroverride` command property
Primary adapter component	Yes	No		The name of the adapter component that contains configuration properties for the adapter. Either enter a name of an adapter file, or click **Browse** to select an adapter file from the list of files that are available in referenced message set projects. When theJDEdwardsRequest node receives data from the JD Edwards system, it associates that data with a method name. The JDEdwardsRequest node attempts to call methods that are defined in the primary adapter. If the method is not defined in the primary adapter, the node can call methods that are defined in matching secondary adapters that are deployed to the same execution group.	
Secondary adapter mode	No	Yes	None	Specifies whether the node can call methods that are defined in secondary adapters. If you set the `Secondary adapter mode` property to `None`, the JDEdwardsRequest node calls only methods that are defined in the primary adapter. If the method is not defined in the primary adapter, an error occurs. If you set this property to `All adapters in execution group`, the node can call methods that are defined in any JD Edwards outbound adapter that is deployed to the same execution group. For more information, see "Calling new services from a WebSphere Adapters request node without changing existing deployed resources" on page 2042.	secondaryAdapterMode

Property	M	C	Default	Description	mqsiapplybaroverride command property
Default method	Yes	Yes		The default method binding to use. This property lists the methods that are defined by the adapter. You can override this property by setting the method name in the LocalEnvironment.Adapter subtree. For more information, see "Local environment tree structure" on page 1056. The method names correspond to the Service Operation names, which are configured by the Adapter Connection wizard. In most cases, the names are based on the name of the service that is being discovered.	defaultMethod

The JDEdwardsRequest node Response Message Parsing properties are described in the following table.

Property	M	C	Default	Description
Message domain	No	No	DataObject	The domain that is used to parse the response message. By default, the response message that is propagated from the JDEdwardsRequest node is in the DataObject domain. You cannot specify a different domain.
Message set	No	No	Set automatically	The name of the message set in which the incoming message is defined. This field is set automatically from the Adapter component property. If you set this property, then subsequently update the project dependencies to remove this message set reference, a warning is issued. Either update the `Message set` property, or restore the reference to this message set project.
Message type	No	No		The name of the response message. The node detects the message type automatically. You cannot set this property.
Message format	No	No		The name of the physical format of the response message. You cannot set this property.

The JDEdwardsRequest node Transactionality properties are described in the following table.

Property	M	C	Default	Description
Transaction mode	No	No	No	This property specifies that updates are performed independently, not as part of a local transaction. You cannot change this property.

The JDEdwardsRequest node Request properties are described in the following table.

Property	M	C	Default	Description
Method Location	Yes	No	$LocalEnvironment/ Adapter/MethodName	The location of the business function or XML list method that is used to trigger the JDEdwardsRequest node to perform an action on the external system.
Data Location	Yes	No	$Body	The location in the incoming message tree from which data is retrieved to form the request that is sent from the JDEdwardsRequest node to the EIS. The default value, `$Body`, represents the incoming message body. You can enter any XPath or ESQL expression that defines the location of the message tree to serialize and send.

The JDEdwardsRequest node Result properties are described in the following table.

Property	M	C	Default	Description
Output data location	No	No	$OutputRoot	The message tree location to which the JDEdwardsRequest node sends output. The default value, `$OutputRoot`, replaces the incoming message with the response.
Copy local environment	No	No	Selected	This property controls whether to copy the incoming local environment or propagate the incoming local environment. By default, this check box is selected, which signifies that the local environment is copied so that the incoming local environment is preserved. The additions to the local environment are visible only to nodes downstream of this node. If this check box is cleared, the incoming local environment is used for the outgoing message. Any modifications that are made to the local environment by this node are visible to both downstream and upstream nodes after this node has completed.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Overview of WebSphere Adapter for JD Edwards EnterpriseOne" on page 2023
With the WebSphere Adapter for JD Edwards EnterpriseOne, you can create integrated processes that include the exchange of information with a JD Edwards EnterpriseOne server, without special coding.

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"WebSphere Adapters deployment" on page 3219
When you have run the Adapter Connection wizard and created a message flow, you must deploy the resources that are generated by adding them to a broker archive (BAR) file.

"DataObject parser and domain" on page 1114
Use the DataObject domain to parse and write messages for WebSphere Adapters and CORBA applications.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

"Calling new services from a WebSphere Adapters request node without changing existing deployed resources" on page 2042
If your message flow acts as a gateway to an Enterprise Information System (EIS), you can use it to call new services that did not exist when you developed the flow. Therefore, if a new service is provided by the EIS, you do not have to modify and retest the message flow.

"Controlling the functional level of WebSphere Message Broker" on page 51
You can control the functional level of your broker to enable new functionality added in WebSphere Message Broker fix packs.

Related reference:

"WebSphere Adapter for JD Edwards EnterpriseOne properties" on page 4146
Reference information to refer to when you connect to a JD Edwards EnterpriseOne application.

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

JMSHeader node

Use the JMSHeader node to modify contents of the JMS Header_Values and Application properties so that you can control the node's output without programming.

This topic contains the following sections:
- "Purpose"
- "Using this node in a message flow"
- "Terminals and properties" on page 4530

Purpose:
Use the node to control the output from a JMSOutput node. A subset of common values can be changed in the JMS Header, and user-chosen properties can be added, changed, or deleted for the Application properties.

For JMS Header_Values properties, the node provides a set of fields that you can modify using predefined values, user-defined values, or XPath expressions. XPath is used to provide a valid location from which a value for a property can be copied. For example, the location can be the body of the message, the local environment tree, or an exception list.

For JMS Application properties, the node provides a way to add, modify, and delete name-value pairs of application properties.

The JMSHeader node is contained in the **JMS** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:

Look at the following sample for more details about how to use the node:

- JMSHeader node

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Terminals and properties:
When you have put an instance of the node into a message flow, you can configure it; see . This node has no mandatory properties.

JMSHeader node terminals are described in the following table:

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the input message is routed if a failure is detected during extraction.
Out	The output terminal to which the transformed message is routed if the input message is processed successfully.

The following tables describes the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The JMSHeader node Description properties are described in the following table:

Property	M	C	Default	Description
Node name	No	No	JMSHeader	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The JMSHeader node JMS transport header options are described in the following table:

Property	M	C	Default	Description
JMS header options	No	Yes	Carry forward header	Options to control the JMSTransport header as a whole. Select **Carry forward header** to carry forward any values that are present in an incoming message. Select **Add header** to add a new header using the specified property values. If a header already exists, the header is modified using the specified property values. If *Inherit from header* is specified as a property value and the header does not exist, the default value for the property is used. Select **Modify header** to change an existing header using the specified property values. If a header does not exist, a new header is added first. If *Inherit from header* is specified as a property value and the header does not exist, the default value for the property is used. Select **Delete header** to delete the header, if it exists. **Note:** The **Add header** and **Modify header** options both add a header if it does not exist, and change a header if it does exist. However, the default values offered by each option differ, so use the appropriate option.

The JMSHeader node JMSHeader_Value properties are described in the following table:

Property	M	C	Default	Description	`mqsiapplybaroverride` command property
JMS Delivery Mode	No	Yes	Non _Persistent	Filter messages by message delivery mode: • Non_Persistent • Persistent	
JMS Message Expiration(ms)	No	Yes	0	Ask the JMS provider to keep the output JMS message for a specified time. Values are in milliseconds; the default value 0 means that the message should not expire.	
JMS Message Priority	No	Yes	4	Assign relative importance to the message. A receiving JMS client application or a JMSOutput node can use this value. JMS defines a ten-level priority value, with 0 as the lowest priority and 9 as the highest.	
JMS Correlation Identifier	No	Yes	No default value	A client can use the JMS Correlation Identifier header field to link one message with another. A typical use is to link a response message with its request message.	
JMS Reply To	No	Yes	No default value	The JMS Reply To header field contains a destination supplied by a client when a message is sent. It is the destination where a reply message should be sent to.	jmsReplyTo

The JMSHeader node Application properties are described in the following table:

Property	M	C	Default	Description
Application Properties	No	Yes		This screen is enabled only if you chose **Add header** or **Modify header** for JMS Transport header. The screen has no predefined properties; you use it to create custom properties and values. Use the property table to add new properties, or modify or delete existing properties, for the header. There is no limit to the number of properties. Each property must have a name, and a type qualifier. The type qualifier can be `Value`, `XPath`, or `Delete`. **Value** Enter a new valid value for the selected property. A null value or empty string is also considered as a valid value. **XPath** Specify a valid XPath expression. WebSphere Message Broker supports XPath definitions that start with an XPath variable such as `$Root` or `$LocalEnvironment`. Only the first occurrence is returned if there are multiple values for the XPath expression. (Examples of valid XPath expressions are: $LocalEnvironment/Host, and $Root/HTTPRequest/Content-Type). **Delete** Specify the property to be deleted from the incoming message. The value associated with the selected property is also deleted.
Clear incoming values	No	Yes	Cleared	This option, which is is enabled only if you choose **Modify header**, removes all property names and associated values from the incoming message if present.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related tasks:

"Accessing headers" on page 2453
If the input message received by an input node includes message headers that are recognized by the input node, the node invokes the correct parser for each header. You can access these headers using ESQL.

Related reference:

"JMSInput node"
Use the JMSInput node to receive messages from JMS destinations. JMS destinations are accessed through a connection to a JMS provider.

"JMSOutput node" on page 4549
Use the JMSOutput node to send messages to JMS destinations.

JMSInput node

Use the JMSInput node to receive messages from JMS destinations. JMS destinations are accessed through a connection to a JMS provider.

This topic contains the following sections:

Purpose:
The JMSInput node acts as a JMS message consumer and can receive all six message types that are defined in the Java Message Service Specification, version 1.1. Messages are received by using method calls, which are described in the JMS specification.

The JMSInput node is contained in the **JMS** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using the JMSInput node in a message flow:
The following sample contains a message flow in which the JMSInput node is used. This sample is an example of how to use the JMSInput node.

- JMS Nodes

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

The JMSInput node receives and propagates messages with a JMS message tree. You can set the properties of the JMSInput node to control the way in which the JMS messages are received.

The JMSInput node handles messages in the following message domains:
- XMLNSC
- DataObject
- JSON
- BLOB
- MIME
- MRM
- JMSMap
- JMSStream
- XMLNS

Message flows that handle messages that are received from connections to JMS providers must always start with a JMSInput node. If you include an output node in a message flow that starts with a JMSInput node, it can be any of the supported output nodes (including user-defined output nodes); you do not have to include a JMSOutput node. However, if you do not include a JMSOutput node, you must include the JMSMQTransform node to transform the message to the format that is expected by the output node.

If you are propagating JMS messages and creating a message flow to use as a subflow, you cannot use a standard input node; you must use an instance of the JMSInput node as the first node in order to create an In terminal for the subflow.

Restriction: When the JMSInput node receives publication topics, it internally restricts the message flow property Additional Instances to zero to prevent the receipt of duplicate publications.

Making the JMS provider client available to the JMS nodes:
Configurable services are defined for a number of JMS providers. You can choose one of the predefined services, or you can create a service for a new provider, or for one of the existing providers. The predefined services are listed in "Configurable services properties" on page 3766.

- If you want to use the WebSphere MQ JMS provider, and you have installed WebSphere MQ in the default location on the broker system, the properties are already set and you do not have to change them.
- If you want to use the WebSphere MQ JMS provider, and you have installed WebSphere MQ in a different (nondefault) location, or if you want to use one of the other defined services, you must set the `jarsURL` property to identify the location of the service JAR files on the broker system. On Windows, the file location cannot be a mapped network drive on a remote Windows computer; the directory must be local or on a storage area network (SAN) disk.

  Use the **mqsireportproperties** command to view the provider properties, and the **mqsichangeproperties** command to set or modify the properties.
- If no service is defined for your JMS provider, or if you want to create another service for an existing JMS provider, use the **mqsicreateconfigurableservice** command to identify the new service and to set its properties.
- When you configure the node, select the appropriate service from the list of predefined services shown for the JMS provider name property, or type in the name of your required service.
- Some JMS providers provide an alternative interface to the standard JMS specification for particular JMS API calls. In these cases, IBM supplies a Java class to interface with that proprietary API. For example, if the JMS nodes use BEA WebLogic as the JMS provider, and the nodes have to participate in a globally coordinated message flow, you must modify the configurable services properties that are associated with that vendor. For more information, see "Configuring the broker to enable a JMS provider's proprietary API" on page 748.
- Some JMS providers, such as the BEA WebLogic provider, do not update the optional JMSXDeliveryCount field in the JMS message header; therefore, JMSInput node backout processing is not possible. To cope with any failures in the message flow, connect the Failure terminal of the JMSInput node.
- To connect to different versions of a JMS provider, create a JMSProviders configurable service for each version of the JMS provider, then set the **jarsURL** property to a unique path.

Connecting the terminals:
For each message that is received successfully, the JMSInput node routes the message to the Out terminal. If this action fails, the message is retried. If the retry threshold is reached, where the threshold is defined by the `Backout threshold` property of the node, the message is routed to the Failure terminal. You can connect nodes to the Failure terminal to handle this condition.

If an exception occurs in the failure path, the path is tried again until the number of attempts is twice the `Backout threshold`. If that limit is exceeded, the message is put to the `Backout destination`.

If you have not connected nodes to the Failure terminal, the message is written to the `Backout destination`. If you have not specified a `Backout destination`, the node issues a BIP4669 error message and stops processing further input.

If processing is not resumed after you restart the broker or execution group, check the Deployment Log for a cause, such as an incorrect parser being specified in the node properties. Correct the problem and redeploy the message flow. If the message itself is not valid, remove the message from the input queue to resume processing.

If the message is caught by the JMSInput node after an exception has been generated elsewhere in the message flow, the message is routed to the Catch terminal. If you have not connected nodes to the Catch terminal, the node backs out messages for redelivery until the problem is resolved, or the `Backout threshold` is reached. If you do not define a `Backout destination`, the node issues a BIP4669 error message and stops processing further input.

Configuring for coordinated transactions:
When you include a JMSInput node in a message flow, the value that you set for `Transaction mode` defines whether messages are received under sync point. See "Configuring for coordinated JMS transactions" on page 4544.

Configuring for batch acknowledgment:
You can configure the JMSInput node to send a batch acknowledgment for receipt of non-transactional JMS messages. See "Configuring the JMSInput node for batch message processing" on page 751.

Terminals and properties:
When you have put an instance of the JMSInput node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view. All mandatory properties that do not have a default value defined are marked with an asterisk.

The terminals of the JMSInput node are described in the following table.

Terminal	Description
Failure	The output terminal to which the message is routed if an error occurs. Even if the Validation property is set, messages that are propagated to this terminal are not validated.
Out	The output terminal to which the message is routed if it is retrieved successfully.
Catch	The output terminal to which the message is routed if an exception is generated downstream and caught by this node.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Description properties of the JMSInput node are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, JMSInput	The name of the node.

Property	M	C	Default	Description
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The Basic properties of the JMSInput node are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Source queue	No	Yes	Selected	The name of the queue from which the node receives incoming messages. If the node is to read from a queue (point-to-point), select `Source queue` and enter the name of the source queue, which is the JMS queue that is listed in the bindings file. This property is mutually exclusive with `Subscription topic`.	sourceQueueName
Subscription topic	No	Yes	Cleared	The name of the topic to which the node is subscribed. If the node is to read from a `Subscription topic` (publish/subscribe), select `Subscription topic` and enter the name of the subscription topic. • If you select `Subscription topic`, the node operates in the publish/subscribe message domain only. • This property is mutually exclusive with `Source queue`. • The `Subscription topic` name must conform to the standards of the JMS provider that is being used by the node.	topic
Durable subscription ID	No	Yes		The identifier for a durable subscription topic. If the node is to receive publications from a durable subscription topic, enter a `Durable subscription ID`. • Removing a durable subscription is a separate administration task. For information about removing a durable subscription see the JMS provider documentation. • This property is valid only when a `Subscription topic` string has been specified.	durableSubscriptionID

The JMS Connection properties of the JMSInput node are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
JMS provider name	Yes	No	`WebSphere MQ`	Select a JMS vendor name from the list, or enter a name of your choice. When you select a name from the list, the `Initial context factory` property is updated automatically with the relevant Java class. If you enter your own JMS provider name, you must also enter a value for the `Initial context factory` The name must match the name of a configurable service that is defined for the broker to which you deploy the message flow.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Initial context factory	Yes	Yes	com.sun.jndi.fscontext.RefFSContextFactory	The starting point for a JNDI namespace. A JMS application uses the initial context to obtain and look up the connection factory and queue or topic objects for the JMS provider. If you select a JMS provider name from the list in JMS provider name, the Initial context factory property is updated automatically with the relevant Java class. If you enter your own JMS provider name, you must also enter a value for the Initial context factory. The default value is com.sun.jndi.fscontext.RefFSContextFactory, which defines the file-based Initial context factory for the WebSphere MQ JMS provider. If the node is set to use your own JMS Provider, and the corresponding Configurable services property of the **mqsichangeproperties** definition has the InitialContextFactory attribute set, this attribute overrides the setting on the node.	initialContextFactory
Location JNDI bindings	Yes	Yes		The system path or the LDAP location for the bindings file. The bindings file contains definitions for the JNDI administered objects that are used by the JMSInput node. When you enter a value for Location JNDI bindings, ensure that it complies with the following instructions: • Construct the bindings file before you deploy a message flow that contains a JMSInput node. • Do not include the file name of the bindings file in this field. • If you have specified an LDAP location that requires authentication, configure the LDAP principal (user ID) and LDAP credentials (password) separately. These values are configured at broker level. For information about configuring these values, see "**mqsicreatebroker** command" on page 3831 and "**mqsichangebroker** command" on page 3723. • The string value must include a supported URL prefix that has a URL handler that is available on the class path. For information about constructing the JNDI administered objects bindings file, see the JMS provider documentation. If the node is set to use your own JMS Provider, and the corresponding Configurable services property of the **mqsichangeproperties** definition has the jndiBindingsLocation attribute set, this attribute overrides the setting on the node.	locationJndiBindings

Property	M	C	Default	Description	mqsiapplybaroverride command property
Connection factory name	Yes	Yes		The name of the connection factory that is used by the JMSInput node to create a connection to the JMS provider. This name must exist in the bindings file. The `Connection factory name` can be a JMS QueueConnectionFactory or a JMS TopicConnectionFactory, but it must match the message model that is used by the node. You can also specify the generic JMS ConnectionFactory, which can be used for both JMS queue or JMS topic destinations.	connectionFactoryName
Backout destination	No	Yes		The JMSInput node sends input messages to this destination when errors prevent the message flow from processing the message, and the message must be removed from the input destination. The backout destination name must exist in the bindings file.	backoutDestination
Backout threshold	No	Yes	0	The value that controls when a redelivered message is put to the backout destination. For example, if the value is 3, the JMS provider attempts to deliver the message to the input destination three times. After the third attempted delivery, the message is removed from the input destination and is sent to the `Backout destination`. See "Configuring the backout threshold property" on page 4544.	

The JMSInput node Input Message Parsing properties are described in the following table.

Property	M	C	Default	Description
Message domain	No	No		The domain that is used to parse the incoming message. • XMLNSC • DataObject • JSON • BLOB • MIME • MRM • JMSMap • JMSStream • XMLNS You can also specify a user-defined parser, if appropriate. The `Message domain` that is set on the node takes precedence except when the `Message domain` is set to blank on the node property. If `Message domain` is left blank, the JMSInput node determines the message domain in one of two ways: • By checking for the presence of data in the JMSType header value of the JMS input message • Based upon the Java Class of the JMS message For more information, see "JMS message payload and appropriate parser" on page 1698.

Property	M	C	Default	Description
Message set	No	No		The name or identifier of the message set in which the incoming message is defined. If you are using the MRM parser or the XMLNSC parser in validating mode, select the `Message set` that you want to use. This list is populated with available message sets when you select MRM or XMLNSC as the `Message domain`. If you set this property, then later update the project dependencies to remove this message set reference, a warning is issued. Either update the `Message set` property, or restore the reference to this message set project.
Message type	No	No		The name of the incoming message. If you are using the MRM parser, select the message that you want from the list in `Message type`. This list is populated with messages that are defined in the `Message set` that you have selected.
Message format	No	No		The name of the physical format of the incoming message. If you are using the MRM parser, select the format of the message from the list in `Message format`. This list includes all of the physical formats that you have defined for this `Message set`.

The properties of the Parser Options for the JMSInput node are described in the following table.

Property	M	C	Default	Description
Parse timing	No	No	`On Demand`	This property controls when an input message is parsed. Valid values are: • `On Demand` • `Immediate` • `Complete` `Parse timing` is, by default, set to `On Demand`, which causes parsing of the message to be delayed. To cause the message to be parsed immediately, see "Parsing on demand" on page 4173.
Build tree using XML schema data types	No	No	Cleared	This property controls whether the XMLNSC parser creates syntax elements in the message tree with data types taken from the XML schema. For more information about how the XMLNSC parser operates, see "Manipulating messages in the XMLNSC domain" on page 2546.
Use XMLNSC compact parser for XMLNS domain	No	No	Cleared	This property controls whether the XMLNSC Compact Parser is used for messages in the XMLNS Domain. If you set this property, the message data is displayed under XMLNSC in nodes that are connected to the output terminal when the input MQRFH2 header or Input Message Parsing properties `Message domain` is XMLNS.
Retain mixed content	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters mixed text in an input message. If you select the check box, elements are created for mixed text. If you clear the check box, mixed text is ignored and no elements are created.
Retain comments	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters comments in an input message. If you select the check box, elements are created for comments. If you clear the check box, comments are ignored and no elements are created.
Retain processing instructions	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters processing instructions in an input message. If you select the check box, elements are created for processing instructions. If you clear the check box, processing instructions are ignored and no elements are created.

Property	M	C	Default	Description
Opaque elements	No	No	Blank	This property is used to specify a list of elements in the input message that are to be opaquely parsed by the XMLNSC parser. Opaque parsing is performed only if validation is not enabled (that is, if `Validate` is `None`); entries that are specified in `Opaque Elements` are ignored if validation is enabled.

The Message Selectors properties of the JMSInput node are described in the following table. Set these properties to filter messages. For a description of how to construct the JMS message selector, see JMS message selector.

Property	M	C	Default	Description
Application property	No	No		The message selector that filters messages according to the application property value. If the JMS provider is required to filter messages, based on message properties that are set by the originating JMS client application, enter a selector string for `Application property`, specifying both the property name and the selection conditions; for example, `OrderValue > 200`. Leave `Application property` blank if you do not want the input node to make a selection based on application property.
Timestamp	No	No		The message selector that filters messages according to the JMSTimestamp. If the JMS provider is required to filter messages that have been generated at specific times, enter a selector string for `Timestamp`, where the value is an unqualified Java millisecond time; for example, `105757642321`. Qualify the selector with operators, such as `=`, `BETWEEN` or `AND`. Leave `Timestamp` blank if you do not want the input node to make a selection based on the JMSTimeStamp.
Delivery mode	No	No	`All`	The message selector that filters messages according to the message delivery mode. If the JMS provider is required to filter messages based on the JMSDeliveryMode header value in the JMS messages, select an option for `Delivery mode` from the list: • Select `Non Persistent` to receive messages that are marked as nonpersistent by the originating JMS client application. • Select `Persistent` to receive messages that are marked as persistent by the originating JMS client application. • Select `All` to receive both persistent and nonpersistent messages. (This value is the default.)
Priority	No	No		The message selector that filters messages according to the message priority. If the JMS provider is required to filter messages based on the JMSPriority header value in the JMS message, enter a selector string for `Priority`. Valid values for `Priority` are from `0` (lowest) to `9` (highest). For example, enter `= 5` to receive messages of priority 5, `> 4` to receive messages with a priority greater than 4, or `BETWEEN 4 AND 8` to receive messages with a priority in the range 4 - 8. Leave `Priority` blank if you do not want the input node to make a selection based on the JMSPriority.

Property	M	C	Default	Description
Message ID	No	No		The message selector that filters messages according to the message ID. If the JMS provider is required to filter messages based on the JMSMessageID header, enter a selector string for `Message ID`. For example, enter `> WMBRK123456` to return messages where the `Message ID` is greater than `WMBRK123456`. Leave `Message ID` blank if you do not want the input node to make a selection based on JMSMessageID.
Redelivered	No	No		If the JMS provider is required to filter messages based on the JMSRedelivered header, enter a selector string for `Redelivered`: • Enter `= FALSE` if the input node accepts only messages that have not been redelivered by the JMS provider. • Enter `= TRUE` if the input node accepts only messages that have been redelivered by the JMS provider. • Leave `Redelivered` blank if you do not want the input node to make a selection based on JMSRedelivered.
Correlation ID	No	No		The message selector that filters messages according to the correlation ID. If the JMS provider is required to filter messages based on the JMSCorrelationID header, enter a selector string for `Correlation ID`. For example, `= WMBRKABCDEFG` returns messages with a `Correlation ID` that matches this value. Leave `Correlation ID` blank if you do not want the input node to make a selection based on JMSCorrelationID.

The Advanced properties of the JMSInput node are described in the following table.

Property	M	C	Default	Description
Transaction mode	Yes	No	`None`	This property controls whether the incoming message is received under external sync point, local sync point, or out of sync point. • Select `None` if the incoming message is to be treated as nonpersistent. If you select this value, the message is received using a non-transacted JMS session that is created using the `Session.AUTO_ACKNOWLEDGE` flag. • Select `Local` if the JMSInput node must coordinate the commit or roll back of JMS messages that are received by the node, along with any other resources such as DB2 or WebSphere MQ that perform work within the message flow. If you select this value, the node uses a transacted JMS session. • Select `Global` if the JMSInput node must participate in a global message flow transaction that is managed by the external sync point coordinator of the Broker. The sync point coordinator is the queue manager of the Broker on distributed systems and RRS (Resource Recovery Services) on z/OS. If you select this value, any messages that are received by the node are globally coordinated using an XA JMS session.

The Validation properties of the JMSInput node are described in the following table. For more details, see “Validating messages” on page 1478 and “Validation properties” on page 4169.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Validate	No	Yes	None	This property controls whether validation takes place. Valid values are: • None • Content and Value • Content If you select Content or Content and Value, select an option from the Failure action list.	validateMaster
Failure action	No	No	Exception	This property controls what happens if validation fails. You can set this property only if you set Validate to Content or Content and Value. Valid values are: • User Trace • Local Error Log • Exception (The default value) • Exception List	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"JMS parsers and domains" on page 1116
The JMSMap and JMSStream domains can be used for modeling messages that are produced by the implementations of the Java Messaging Service standard.

"JMS message transformation" on page 1684
The JMSInput and JMSOutput nodes expect JMS messages, and therefore expect a native JMS message tree representation.

Related tasks:

"Manipulating messages in the XMLNSC domain" on page 2546
If you are writing ESQL to process messages in the XMLNSC domain, it is helpful to learn about the structure of the message tree that the XMLNSC parser builds.

"Configuring transactionality for message flows" on page 1290
A message flow runs in a single transaction, which is started when data is received by an input node, and can be committed or rolled back when all processing has completed.

"Configuring JMS and SOAP nodes to support global transactions" on page 1716
To include nodes that use JMS transport, such as the JMS and SOAP nodes, in globally coordinated transactions, you must complete additional configuration.

"Configuring the broker to enable a JMS provider's proprietary API" on page 748
Some JMS providers provide an alternative interface to the standard JMS specification for particular JMS API calls. In these cases, IBM supplies a Java class to interface with that proprietary API.

"Connecting to different versions of the same JMS provider" on page 750
To use different versions of the same JMS provider, create a configurable service

for the JMS provider, and set the jarsURL property to a unique path.

"Configuring the broker to use SSL with JMS nodes" on page 530
Configure your broker to work with a JMS provider that supports JMS clients that can connect by using the Secure Sockets Layer (SSL) protocol.

Related reference:

"Configuring the backout threshold property" on page 4544
You can set the backout threshold property on nodes that use JMS transport to specify how many attempts are made to deliver the message to the input destination.

"Configuring for coordinated JMS transactions" on page 4544
Configure your message flow to receive or output messages under coordinated transactions.

"JMS message payload and appropriate parser" on page 1698
Configure the JMSInput node properties to specify the message domain that will be used to parse the JMS message payload.

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"JMS message selector" on page 1703
A message selector allows a JMS consumer to be more selective about the messages that it receives from a particular topic or queue.

"JMSOutput node" on page 4549
Use the JMSOutput node to send messages to JMS destinations.

"JMSReply node" on page 4562
Use the JMSReply node to send messages to JMS destinations.

"JMSMQTransform node" on page 4547
Use the JMSMQTransform node to transform a message with a JMS message tree into a message that has a message tree structure that is compatible with the format of messages that are produced by the WebSphere MQ JMS provider.

"MQJMSTransform node" on page 4610
Use the MQJMSTransform node to receive messages that have a WebSphere MQ JMS provider message tree format, and transform them into a format that is compatible with messages that are to be sent to JMS destinations.

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

Related information:

WebSphere MQ Version 7 Information Center online

Configuring the backout threshold property:

You can set the backout threshold property on nodes that use JMS transport to specify how many attempts are made to deliver the message to the input destination.

If the Backout threshold is set to `0` then redelivery is not attempted. If the Backout threshold is 1 or greater, the message will be redelivered the specified number of times.

Set the value of Backout threshold depending on the capabilities of the JMS provider.

If the JMS provider supports `JMSXDeliveryCount`, you can set the Backout threshold to any value.

If the JMS provider does not support `JMSXDeliveryCount`, the Backout threshold must only be set to 0 or 1. If the JMS provider does not support `JMSXDeliveryCount` and the value is set to greater than 1, a redelivered message is repeatedly backed out and reprocessed, and is never delivered to the backout destination.

Related reference:

"JMSInput node" on page 4532
Use the JMSInput node to receive messages from JMS destinations. JMS destinations are accessed through a connection to a JMS provider.

"SOAPInput node" on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

"SOAPAsyncRequest node" on page 4750
Use the SOAPAsyncRequest node with the SOAPAsyncResponse node to construct a pair of message flows that call a Web service asynchronously.

Configuring for coordinated JMS transactions:

Configure your message flow to receive or output messages under coordinated transactions.

When you include a node using JMS transport in a message flow, such as the JMSInput or SOAPInput node when using JMS transport, the value that you set for `Transaction mode` defines whether messages are received under sync point.

- If you set this property to `Global`, the message is received under external sync point coordination; that is, within a WebSphere MQ unit of work. Any messages that are sent later, by an output node in the same instance of the message flow, are put under sync point, unless the output node overrides this setting explicitly.
- If you set this property to `Local`, the message is received under the local sync point control of the node. Any messages that are sent later, by an output node in the flow, are not put under local sync point, unless an individual output node specifies that the message must be put under local sync point.
- If you set this property to `None`, the message is not received under sync point. Any messages that are sent later, by an output node in the flow, are not put under sync point, unless an individual output node specifies that the message must be put under sync point.

To receive messages under external sync point, you must take additional configuration steps, which need be applied only the first time that a specific node using JMS transport is deployed to the broker for a particular JMS provider.

- On distributed systems, the external sync point coordinator for the broker is WebSphere MQ. Before you deploy a message flow in which the `Transaction mode` property is set to `Global` or `Yes`, and is intended to use globally coordinated transactions, modify the queue manager `.ini` file to include extra definitions for each JMS provider resource manager that participates in globally coordinated transactions.
  - Windows On Windows:
    1. Start WebSphere MQ Explorer.
    2. Right-click the queue manager name in the left pane and click **Properties**.
    3. Click **XA resource managers** in the left pane.
    4. Click **Add...** .
    5. Set the options as follows:
       - Set `Name` to any value.
       - On Windows on x86 systems, set the `SwitchFile` property to *`install_dir`*`\bin\JMSSwitch.dll`. On Windows on x86-64 systems, set the `SwitchFile` property to `JMSSwitch.dll`.
       - Set the `XAOpenString` property to a string value as follows: *`Initial Context`*`,`*`location JNDI`*`,`*`Optional_parms`*.
       - Set the `ThreadOfControl` property to `Thread`.
    6. On Windows on x86-64 systems only, copy the switch file *JMSSwitch32.dll* to the `\exits` subdirectory in the WebSphere MQ installation directory, and rename it to `JMSSwitch.dll`. Copy the switch file `JMSSwitch.dll` to the `\exits64` subdirectory in the WebSphere MQ installation directory.

    For more information, see the *System Administration Guide* section of the WebSphere MQ Version 7 Information Center online.
  - Linux UNIX On Linux and UNIX systems, add a stanza to the queue manager `.ini` file for each JMS provider.

    For example:

    ```
    XAResourceManager:
    Name=Jms_Provider_Name
    SwitchFile=/install_dir/bin/ JMSSwitch.so
    XAOpenString=Initial Context,location JNDI,Optional_parms
    ThreadOfControl=THREAD
    ```

    Where:

    Name is an installation defined name that identifies a JMS provider resource manager.

    SwitchFile
    is the file system path to the JMSSwitch library that is supplied in the `bin` directory of the broker.

  XAOpenString can have the following values:
  - *Initial Context* is the value that is set in the JMSInput node property `Initial context factory`.
  - *location JNDI* is the value that is set in the JMSInput node property `Location JNDI bindings`. This value must include a supported URL prefix that has a URL handler that is available on the class path.

  The following parameters are optional:

– `LDAP Principal` matches the value that is set for the broker by using the **mqsicreatebroker** or **mqsichangebroker** commands.
– `LDAP Credentials` matches the value that is set for the broker by using the **mqsicreatebroker** or **mqsichangebroker** commands.
– `Recovery Connection Factory Name` is the JNDI administered connection factory that is defined in the bindings file. If a value is not specified, you must add a default value for `recoverXAQCF` to the bindings file. In either case, the Recovery Connection Factory must be defined as an XA Queue Connection Factory for the JMS provider that is associated with the `Initial context factory`.

The optional parameters are comma-separated and are positional. Therefore, any parameters that are missing must be represented by a comma. For example:

```
com.sun.jndi.fscontext.RefFSContextFactory,file:/C:/webservices/SOAP/JMS/JNDIXA,,,QCF
```

1. Update the Java CLASSPATH environment variable for the queue manager of the broker to include a reference to `xarecovery.jar`; for example:

   install_dir`/classes/xarecovery.jar`

2. Update the Java PATH environment variable for the queue manager of the broker to point to the bin directory in which the SwitchFile is located; for example:

   install_dir`/bin`

Finally, ensure that you have taken the following configuration steps:

– In the message flow, ensure that the coordinated property is enabled by using the WebSphere Message Broker Archive editor.
– Ensure that each node that must be part of the XA transaction is set to the global transaction mode.
– Ensure that the service ID that is used for the broker and the queue manager is the same user ID.
– Ensure that the JNDI connection factory objects that the JMS nodes use for a global transaction are configured to be of the type MQXAConnectionFactory, MQXAQueueConnectionFactory, or MQXATopicConnectionFactory.
  - If you create the bindings using WebSphere Message Broker Explorer, ensure the `Support XA Transactions` option is checked when you define your connection factory.
  - If you create the bindings using JMSAdmin, use the command **DEF XAQCF** or **DEF XATCF**, instead of **DEF QCF** or **DEF TCF**, when you define your connection factory.

For more information, see the *System Administration Guide* section of the WebSphere MQ Version 7 Information Center online.

- z/OS On z/OS, the external sync point manager is Resource Recovery Services (RRS). The only JMS provider that is supported on z/OS is WebSphere MQ JMS. The only transport option that is supported for WebSphere MQ JMS on z/OS is the bind option.

  Sync point control for the JMS provider is managed with RRS sync point coordination of the queue manager of the broker. You do not have to modify the `.ini` file.

Related reference:

"JMSInput node" on page 4532
Use the JMSInput node to receive messages from JMS destinations. JMS destinations are accessed through a connection to a JMS provider.

"SOAPInput node" on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker

operates as a SOAP Web Services provider.

"SOAPReply node" on page 4819
Use the SOAPReply node to send SOAP messages from the broker to the originating client in response to a message received by a SOAPInput node.

"SOAPRequest node" on page 4828
Use the SOAPRequest node to send a SOAP request to the remote Web service.

"SOAPAsyncRequest node" on page 4750
Use the SOAPAsyncRequest node with the SOAPAsyncResponse node to construct a pair of message flows that call a Web service asynchronously.

"SOAPAsyncResponse node" on page 4777
Use the SOAPAsyncResponse node in conjunction with the SOAPAsyncRequest node to construct a pair of message flows that call a Web service asynchronously.

JMSMQTransform node

Use the JMSMQTransform node to transform a message with a JMS message tree into a message that has a message tree structure that is compatible with the format of messages that are produced by the WebSphere MQ JMS provider.

This topic contains the following sections:
- "Purpose"
- "Using the JMSMQTransform node in a message flow"
- "Terminals and properties"

Purpose:
You can use the JMSMQTransform node to send messages to existing message flows and to work with WebSphere MQ JMS and WebSphere Message Broker publish/subscribe.

The JMSMQTransform node handles messages in all supported message domains.

The JMSMQTransform node is contained in the **JMS** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using the JMSMQTransform node in a message flow:
The following sample contains a message flow in which the JMSMQTransform node is used. Look at this sample for an example of how to use the JMSMQTransform node.
- JMS Nodes

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Terminals and properties:
When you have put an instance of the JMSMQTransform node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view.

The terminals of the JMSMQTransform node are described in the following table.

Terminal	Description
Failure	The output terminal to which the message is routed if an error occurs. Even if the Validation property is set, messages that are propagated to this terminal are not validated.
Out	The output terminal to which the message is routed if it is successfully retrieved from the JMS destination.
In	The input terminal that accepts a message for processing by the node.

The following table describes the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The JMSMQTransform node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, JMSMQTransform	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related reference:

"JMSInput node" on page 4532
Use the JMSInput node to receive messages from JMS destinations. JMS destinations are accessed through a connection to a JMS provider.

"JMSOutput node" on page 4549
Use the JMSOutput node to send messages to JMS destinations.

"MQJMSTransform node" on page 4610
Use the MQJMSTransform node to receive messages that have a WebSphere MQ JMS provider message tree format, and transform them into a format that is compatible with messages that are to be sent to JMS destinations.

JMSOutput node

Use the JMSOutput node to send messages to JMS destinations.

This topic contains the following sections:

Purpose:

The JMSOutput node acts as a JMS message producer, and can publish all six message types that are defined in the Java Message Service Specification, version 1.1. Messages are published by using method calls, which are described in the JMS specification.

The JMSOutput node is contained in the **JMS** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using the JMSOutput node in a message flow:

The following sample contains a message flow in which the JMSOutput node is used. Look at this sample for an example of how to use the JMSOutput node.

- JMS Nodes

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Message flows that handle messages that are received from connections to JMS providers must always start with a JMSInput node. If you include the JMSOutput node in a message flow, you do not need to include a JMSInput node; but if you do not include a JMSInput node, you must include the MQJMSTransform node to transform the message to the format that is expected by the JMSOutput node.

If you are propagating JMS messages and creating a message flow to use as a subflow, use an instance of the JMSOutput node as the last node to create an out terminal for the subflow.

Controlling the type of the JMS output message:

In the JMS message tree, the JMS message type is represented by the PayloadType field of the Message_MetaData subfolder. To control the type of JMS message that is created by the JMSOutput node, use ESQL code to set the Payload value, as shown in the following example:

```
SET OutputRoot.JMSTransport.Transport_Folders.Message_MetaData.PayloadType=Payload value
```

For more information about the JMS message tree and payload values, see "Representation of messages in the JMS Transport" on page 1691.

Sending a JMS message to a destination list:
To send a JMS message to a destination list, ensure that the following conditions are met.

- Select `Send to destination list in local environment` on the **Basic** properties tab of the JMSOutput node.
- Set up the list in the local environment, as shown in the following example.

```
CREATE PROCEDURE CreateJMSDestinationList() BEGIN
  SET OutputLocalEnvironment.Destination.JMSDestinationList.DestinationData[1]  = 'jndi://TestDestQueue1';
  SET OutputLocalEnvironment.Destination.JMSDestinationList.DestinationData[2]  = 'jndi://TestDestQueue2';
  SET OutputLocalEnvironment.Destination.JMSDestinationList.DestinationData[3]  = 'jndi://TestDestQueue3';
END;
```

- Ensure that the message model (point-to-point or publish/subscribe) matches the model that is used by the JMSOutput node. In this case, the model is point-to-point.
- If the destination name in the list is prefixed with the string "jndi://", it indicates to the JMSOutput node that the value represents the name of a JNDI administered object, which needs to be looked up. Alternatively, if the JMS provider-specific format for a destination is known, it can be used; for example, `queue://qmgrname/queuename` for WebSphere MQ. Otherwise, the value is used to create a temporary destination.
- The items to which the JMS destination list refers represent JMS destinations that can be either JMS queues or JMS topics. These destination types must be consistent with the connection factory type that is specified in the JMSOutput node that will process the destination list. For example, a JMS queue destination can be processed by a JMS queue connection factory or a generic JMS connection factory. Similarly, a JMS topic destination can be processed by a JMS topic connection factory or a generic JMS connection factory.

For further information about using local environment variables in a JMSOutput node, see "Using LocalEnvironment variables with JMSOutput and JMSReply nodes" on page 4242.

Making the JMS provider client available to the JMS nodes:
Configurable services are defined for a number of JMS providers. You can choose one of the predefined services, or you can create a new service for a new provider, or for one of the existing providers. The predefined services are listed in "Configurable services properties" on page 3766.

- If you want to use the WebSphere MQ JMS provider, and you have installed WebSphere MQ in the default location on the broker system, the properties are already set and you do not have to make any changes.
- If you want to use the WebSphere MQ JMS provider, and you have installed WebSphere MQ in a different (non-default) location, or if you want to use one of the other defined services, you must set the `jarsURL` property to identify the location of the service JAR files on the broker system. On Windows, the file location cannot be a mapped network drive on a remote Windows computer; the directory must be local or on a Storage Area Network (SAN) disk.

  Use the **mqsireportproperties** command to view the provider properties, and the **mqsichangeproperties** command to set or modify the properties.
- If no service is defined for your JMS provider, or if you want to create another service for an existing JMS provider, use the **mqsicreateconfigurableservice** command to identify the new service and set its properties.

- When you configure the node, select the appropriate service from the list of predefined services shown for the `JMS provider name` property, or type in the name of your new service.
- Some JMS providers provide an alternative interface from the standard JMS specification for particular JMS API calls. In these cases, IBM supplies a Java JAR file to interface with that proprietary API. For example, if the JMS nodes use BEA WebLogic as the JMS provider, and the nodes need to participate in a globally coordinated message flow, you must modify the configurable services properties that are associated with that vendor. For more information, see "Configuring the broker to enable a JMS provider's proprietary API" on page 748.
- To connect to different versions of a JMS provider, create a JMSProviders configurable service for each version of the JMS provider, then set the **`jarsURL`** property to a unique path.

Using the Message Destination Mode:
The JMSOutput node acts as a message producer and supports the following message scenarios:

- "Sending a datagram message"
- "Sending a reply message"
- "Sending a request message" on page 4552

For more information about how to build JMS destination lists, see "Populating Destination in the local environment tree" on page 2467.

Sending a datagram message

A *datagram* is a self-contained, independent entity of data that carries sufficient information to be routed from the source to the destination computer, without reliance on earlier exchanges between the source and destination computer and the transporting network. The following instructions describe how to send a datagram message:

1. On the **Basic** tab, set the message destination depending on the message model that is being used. Set one of the following properties to a valid JNDI administered object name:
   - `Publication Topic`
   - `Destination Queue`
2. Leave the `Reply To Destination` field blank.

The node resolves the name of the JNDI administered object, which is supplied in either the `Publication topic` or `Destination queue` property, and sends the message to that JMS destination.

Sending a reply message

The sender of a message might want the recipient to reply to the message. In this case, the JMSOutput message can treat the outgoing message as a reply, and route it according to the value that is obtained from the `JMSReplyTo` property from the request message. You can modify the value of the `JMSReplyTo` property in the MbMessage; for example, using a Compute node or a JavaCompute node. This action allows dynamic routing of messages from the JMSOutput node. The node sends the message to the JMS destination name that is set in the JMSReplyTo field of the MbMessage Tree.

The JMSReplyTo value in the MbMessage Tree represents the name of the JMS Destination that is resolved from JNDI. For example:

```
queue://QM_mn2/myJMSQueue4
```

In this case, the value is the JMS-provider specific representation of a JMS destination for the WebSphere MQ JMS provider.

If you do not want to specify a resolved JMS destination name, the JMSOutput node can also accept a JNDI administered object name in the `JMSReplyTo` field. However, the node must resolve an administered object name through JNDI before it can route the message to the underlying JMS destination. In this case, the value in the `JMSReplyTo` field must be prefixed with the string: `jndi://`. For example:

```
jndi://jmsQ4
```

where `jmsQ4` is the name of the JNDI-administered object.

Performance might be affected when you use this method because of the need to look up the administered object in JNDI.

Sending a request message

The JMSOutput node can send a message to a JMS destination with the expectation of a response from the message consumer that processes the request. The following instructions describe how to send a request message:

1. On the **Basic** tab, set the message destination depending on the message model that is being used. Set one of the following properties to a valid JNDI-administered object name:
   - `Publication Topic`
   - `Destination Queue`
2. The JMSReplyTo destination in the outgoing message can be derived from the JMSReplyTo field of the MbMessage Tree that is passed to the node. Alternatively, this value can be overridden by a JNDI-administered object name that is set in the `Reply To Destination` node property.

   To allow the JMSOutput node to set the JMSReplyTo property dynamically in the outgoing message, leave the `Reply To Destination` field blank on the **Basic** tab, and set the JMSReplyTo value in the MbMessage using a Compute node or a JavaCompute node.

The node looks first for a value in the JMSReplyTo field of the MbMessage. If the node finds the value, it passes this value into the JMSReplyTo field of the outgoing message. However, if the `Reply To Destination` field of the **Basic** tab has been specified, this value overrides anything that is set previously in the JMSReplyTo property of the outgoing message, after first resolving the name of the JNDI-administered object.

The node resolves the name of the JNDI-administered object that is supplied in either `Publication Topic` or `Destination Queue`, and sends the message to that JMS destination.

Invoking an output message callback function:
The cciOutputMessageCallback function can be registered as a callback and invoked whenever a message is sent by a JMSOutput node. See "cciOutputMessageCallback" on page 6626.

If the user exit state is active, the cciOutputMessageCallback function is invoked for every output message that is sent successfully from a JMSOutput node where the callback is registered.

If the node provides WrittenDestination information in the LocalEnvironment tree, the callback is invoked after this information is created. See "Using LocalEnvironment variables with JMSOutput and JMSReply nodes" on page 4242.

Working with the JMS message ID:
The JMS message ID is generated by the JMS provider when a message is sent by the JMSOutput node. You cannot set the message ID in the message flow, but you can use one of the following methods to obtain the generated ID after the message has been sent:

- Connect a Compute node to the Out terminal.

  Connect a Compute node to the Out terminal of a JMSOutput node and interrogate the WrittenDestination List. For more information, see "Viewing the logical message tree in trace output" on page 1481.

  An entry for a JMSOutput node has the following format:

```
WrittenDestination = (
   JMS = (
      DestinationData = (
        destinationName = 'queue://jmsQueue1'
        initialContext = 'com.sun.jndi.fscontext.RefFSContextFactory'
        JMSMessageID = ID:414d512054657374514d2020202020206ab98b4520017a02'
        JMSCorrelationID = 'ABCDEFGHIJKLMNOPQRSTUVW'
  )
 )
)
```

- Configure a user exit to process an output message callback event. For more information, see "Exploiting user exits" on page 2985.

Configuring for coordinated transactions:
When you include a JMSOutput node in a message flow, the value that you set for `Transaction Mode` defines whether messages are sent under syncpoint.

- If you set the `Transaction Mode` to `Global`, the message is sent under external syncpoint coordination; that is, within a WebSphere MQ unit of work. Any messages that are sent subsequently by an output node in the same instance of the message flow are put under syncpoint, unless the output node overrides this setting explicitly.
- If you set the `Transaction Mode` to `Local`, the message is sent under the local syncpoint control of the JMSOutput node. Any messages that are sent subsequently by an output node in the flow are not put under local syncpoint, unless an individual output node specifies that the message must be put under local syncpoint.
- If you set the `Transaction Mode` to `None`, the message is not sent under syncpoint. Any messages that are sent subsequently by an output node in the flow are not put under syncpoint, unless an individual output node specifies that the message must be put under syncpoint.

When you want to send messages under external syncpoint, you must perform additional configuration steps, which need to be applied only the first time that a JMSOutput or JMSInput is deployed to the broker for a particular JMS provider:

- On distributed systems, the external syncpoint coordinator for the broker is WebSphere MQ. Before you deploy a message flow in which the `Transaction`

Mode is set to Global, modify the queue manager .ini file to include extra definitions for each JMS provider resource manager that participates in globally coordinated transactions:

– Windows On Windows on x86 systems:

1. Start WebSphere MQ Explorer.
2. Right-click the queue manager name in the left pane and click **Properties**.
3. Click **XA resource managers** in the left pane.
4. Set the SwitchFile property to the following value:

```
install_dir/bin/ JMSSwitch.dll
XAOpenString=Initial Context,location JNDI,Optional_parms
ThreadOfControl=THREAD
```

For more information, see the *System Administration Guide* section of the WebSphere MQ Version 7 Information Center online.

– Windows On Windows on x86-64 systems:

1. Start WebSphere MQ Explorer.
2. Right-click the queue manager name in the left pane and click **Properties**.
3. Click **XA resource managers** in the left pane.
4. Set the SwitchFile property to the following value:

```
JMSSwitch
     XAOpenString=Initial Context,location JNDI,Optional_parms
     ThreadOfControl=THREAD
```

5. Copy the switch file *JMSSwitch32.dll* to the \exits subdirectory in the WebSphere MQ installation directory, and rename it to JMSSwitch.dll. Copy the switch file JMSSwitch.dll to the \exits64 subdirectory in the WebSphere MQ installation directory.

For more information, see the *System Administration Guide* section of the WebSphere MQ Version 7 Information Center online.

– Linux UNIX On Linux and UNIX systems, add a stanza to the queue manager's .ini file for each JMS provider.

For example:

```
XAResourceManager:
Name=Jms_Provider_Name
SwitchFile=/install_dir/bin/JMSSwitch.so
XAOpenString=Initial Context,location JNDI,Optional_parms
ThreadOfControl=THREAD
```

Where:

Name is an installation-defined name that identifies a JMS provider resource manager.

SwitchFile
is the file system path to the JMSSwitch library that is supplied in the bin directory of the broker.

XAOpenString can have the following values:

- *Initial Context* is the value that is set in the JMSInput node basic property Initial context factory.
- *location JNDI* is the value that is set in the JMSInput node basic property Location of JNDI bindings. This value must include the leading keyword, which is file://, iiop://, or ldap://

The following parameters are optional:

- `LDAP Principal` matches the value that is set for the broker by using the **mqsicreatebroker** or **mqsichangebroker** commands.
- `LDAP Credentials` matches the value that is set for the broker by using the **mqsicreatebroker** or **mqsichangebroker** commands.
- `Recovery Connection Factory Name` is the JNDI administered connection factory that is defined in the bindings file. If a value is not specified, a default value for `recoverXAQCF` must be added to the bindings file. In either case, the Recovery Connection Factory must be defined as an XA Queue Connection Factory for the JMS provider that is associated with the Initial Context Factory.

The optional parameters are comma-separated and are positional. Therefore, any parameters that are missing must be represented by a comma.

1. Update the Java CLASSPATH environment variable for the broker's queue manager to include a reference to `xarecovery.jar`; for example:

   install_dir`/classes/xarecovery.jar`
2. Update the Java PATH environment variable for the broker's queue manager to point to the bin directory, which is where the switch file is located; for example:

   install_dir`/bin`

For more information, see the *System Administration Guide* section of the WebSphere MQ Version 7 Information Center online.

– z/OS On z/OS, the external syncpoint manager is Resource Recovery Services (RRS). The only JMS provider that is supported on z/OS is WebSphere MQ JMS. The only transport option that is supported for WebSphere MQ JMS on z/OS is the bind option.

Syncpoint control for the JMS provider is managed with RRS syncpoint coordination of the queue manager of the broker. You do not need to modify the `.ini` file.

If the JMSOutput node uses BEA WebLogic as the JMS provider, and the nodes need to participate in a globallyan XA coordinated message flow, see “Making the JMS provider client available to the JMS nodes” on page 4550.

Connecting the terminals:
Connect the In terminal of the JMSOutput node to the node from which outbound messages are routed.

Connect the Out terminal of the JMSOutput node to another node in the message flow to process the message further, to process errors, or to send the message to an additional destination.

Terminals and properties:
When you have put an instance of the JMSOutput node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view. All mandatory properties that do not have a default value defined are marked with an asterisk.

The terminals of the JMSOutput node are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the message is routed if an error occurs. Even if the Validation property is set, messages that are propagated to this terminal are not validated.

Terminal	Description
Out	The output terminal to which the message is routed if it has been successfully put to the output destination (topic or queue).

The following tables describe the node properties. The column headed M indicates whether the property is mandatory (marked with an asterisk if you must enter a value when no default is defined), the column headed C indicates whether the property is configurable (you can change the value when you add the message flow to the BAR file to deploy it).

The Description properties of the JMSOutput node are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, JMSOutput	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The Basic properties of the JMSOutput node are described in the following table.

Property	M	C	Default	Description	**mqsiapplybaroverride command property**
Destination Queue	No	Yes		The name of the queue to which the node publishes outgoing messages. If the JMSOutput node is to be used to send point-to-point messages, enter the `Destination queue` name for the JMS queue name that is listed in the bindings file.	destinationQueueName
Publication Topic	No	Yes		The name of the topic to which the node publishes messages. • If this property is configured, the node operates only in the publish/subscribe message domain. • This property is mutually exclusive with the `Destination queue` property. • The `Publication Topic` name must conform to the standards of the JMS provider that is being used by the node.	topic
Reply to destination	No	Yes		The name of the JMS destination to which the receiving application must send a reply message. For a reply message to be returned to this JMS destination, the JMS destination name must be known to the domain of the JMS provider that is used by the receiving client. You can enter a JMS destination, which can be either a subscription queue or a destination topic. The default value is blank, in which case the JMS output message can be regarded as a datagram. If the field is blank, the JMSOutput node does not expect a reply from the receiving JMS client.	replyToDestination

Property	M	C	Default	Description	mqsiapplybaroverride command property
Send to destination list in local environment	No	Yes	Cleared	When you have built a list of JMS destinations in the local environment, select this check box to use the destination list. If you do not select this check box, the node uses the configured JMS destination. If you select this check box but you have not built a list of JMS destination in the local environment, the node uses the configured JMS destination.	useDistList

The JMS Connection properties of the JMSOutput node are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
JMS provider name	Yes	No	`WebSphere MQ`	Select a JMS vendor name from the list, or enter a name of your choice. When you select a name from the list, the `Initial Context Factory` property is updated automatically with the relevant Java class. If you enter your own JMS provider name, you must also enter a value for the `Initial Context Factory`. The name must match the name of a configurable service defined for the broker to which you deploy the message flow.	
Initial Context Factory	Yes	Yes	`com.sun.jndi.fscontext.RefFSContextFactory`	This property is the starting point for a JNDI namespace. A JMS application uses the initial context to obtain and look up the connection factory and queue or topic objects for the JMS provider. If you select a JMS provider name from the list in `JMS provider name`, the `Initial Context Factory` property is updated automatically with the relevant Java class. If you enter your own JMS provider name, you must also enter a value for the `Initial Context Factory`. The default value is `com.sun.jndi.fscontext.RefFSContextFactory`, which defines the file-based initial context factory for the WebSphere MQ JMS provider. If the node is set to use your own JMS provider, and the corresponding Configurable services property of the **`mqsichangeproperties`** definition has the `InitialContextFactory` attribute set, this overrides the setting on the node.	initialContextFactory

Property	M	C	Default	Description	**mqsiapplybaroverride command property**
Location JNDI Bindings	Yes	Yes		The system path or the LDAP location for the bindings file. The bindings file contains definitions for the JNDI-administered objects that are used by the JMSOutput node. When you enter a value for Location JNDI Bindings, ensure that it complies with the following instructions: • Construct the bindings file before you deploy a message flow that contains a JMSOutput node. • Do not include the file name of the bindings file in this field. • If you have specified an LDAP location that requires authentication, configure both the LDAP principal (userid) and LDAP credentials (password) separately. These values are configured at broker level. For information about configuring these values, see "**mqsicreatebroker** command" on page 3831 and "**mqsichangebroker** command" on page 3723. • The string value must include a supported URL prefix that has a URL handler that is available on the class path. For information about constructing the JNDI-administered objects bindings file, see the documentation that is supplied with the JMS provider. If the node is set to use your own JMS provider, and the corresponding Configurable services property of the **mqsichangeproperties** definition has the `jndiBindingsLocation` attribute set, this overrides the setting on the node.	locationJndiBindings
Connection Factory Name	Yes	Yes		The name of the connection factory that is used by the JMSOutput node to create a connection to the JMS provider. This name must already exist in the bindings file. The Connection factory can be a JMS QueueConnectionFactory or a JMS TopicConnectionFactory, but it must match the message model that is used by the node. Alternatively, you can specify the generic JMS ConnectionFactory, which can be used for both JMS queue or JMS topic destinations.	connectionFactoryName

The Advanced properties of the JMSOutput node are described in the following table.

Property	M	C	Default	Description
New Correlation ID	No	Yes		If the JMSOutput node is required to generate a new Correlation ID for the message, select `New Correlation ID`. If you leave the check box cleared, the Correlation ID of the output message is taken from the JMSCorrelationID field in the JMSTransport_Header_Values section of the message tree.
Transaction Mode	Yes	No	`None`	This property controls whether the incoming message is received under syncpoint. • Select `None` if the outgoing message is to be treated as nonpersistent. If you select this value, the message is sent using a non-transacted JMS session that is created using the `Session.AUTO_ACKNOWLEDGE` flag. • Select `Local` if the input node that received the message must coordinate the commit or roll-back of JMS messages that have been sent by the JMSOutput node, along with any other resources, such as DB2 or WebSphere MQ, that perform work within the message flow. If you select this value, the node uses a transacted JMS session. • Select `Global` if the JMSOutput node must participate in a global message flow transaction that is managed by the broker's external syncpoint coordinator. The syncpoint coordinator is the broker's queue manager on distributed systems, and RRS (Resource Recovery Services) on z/OS. If you select this value, any messages that are received by the node are globally coordinated using an XA JMS session.
Delivery Mode	No	Yes	`Non Persistent`	This property controls the persistence mode that a JMS provider uses for a message. Valid values are: • `Automatic`: the mode from the input message is inherited • `Persistent`: the message survives if the JMS provider has a system failure • `Non Persistent`: the message is lost if the JMS provider has a system failure
Message Expiration (ms)	No	Yes	`0`	This property controls the length of time, in milliseconds, for which the JMS provider keeps the output JMS message. The default value, `0`, is used to indicate that the message must not expire. Select `Inherit from header` or enter an integer that represents a number of milliseconds. If you select `Inherit from header`, the property inherits the value of the JMSExpiry field in the JMS message, which is found at the following location: `OutputRoot.JMSTransport.Transport_Folders.Header_Values.JMSExpiration`
Message Priority	No	Yes	`4`	This property assigns relative importance to the message and it can be used for message selection by a receiving JMS client application or a JMSOutput node. Select a value between `0` (lowest priority) and `9` (highest priority) or select `Inherit from header`. The default value is `4`, which indicates medium priority. Priorities in the range `0` to `4` relate to typical delivery. Priorities in the range `5` to `9` relate to graduations of expedited delivery. If you select `Inherit from header`, the property inherits the value of the JMSPriority field in the JMS message, which is found at the following location: `OutputRoot.JMSTransport.Transport_Folders.Header_Values.JMSPriority`

Property	M	C	Default	Description
Message Type	No	Yes	`Determine output message type from the JMS Message Tree`	Select a value from the list to configure the type of JMS message that is produced by the JMSOutput node. If you do not set a value for this property, the node assumes the output type from the metadata PayLoadType field in the JMS message tree, as indicated by the default value, `Determine output message type from the JMS Message Tree`. Valid values are: • Determine output message type from the JMS Message Tree • TextMessage • BytesMessage • MapMessage • StreamMessage • ObjectMessage • Base JMS message with no payload

The Validation properties of the JMSOutput node are described in the following table. For more information about Validation properties, see "Validating messages" on page 1478 and "Validation properties" on page 4169.

Property	M	C	Default	Description	**`mqsiapplybaroverride` command property**
Validate	No	Yes	`Inherit`	This property controls whether validation takes place. Valid values are `None`, `Content`, `Content And Value`, and `Inherit`.	validateMaster
Failure Action	No	No	`Exception`	This property controls what happens if validation fails. You can set this property only if you set `Validate` to `Content` or `Content and Value`. Valid values are `User Trace`, `Local Error Log`, `Exception`, and `Exception List`.	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"JMS parsers and domains" on page 1116
The JMSMap and JMSStream domains can be used for modeling messages that are produced by the implementations of the Java Messaging Service standard.

Related tasks:

"Configuring transactionality for message flows" on page 1290
A message flow runs in a single transaction, which is started when data is received by an input node, and can be committed or rolled back when all processing has completed.

"Configuring JMS and SOAP nodes to support global transactions" on page 1716
To include nodes that use JMS transport, such as the JMS and SOAP nodes, in globally coordinated transactions, you must complete additional configuration.

"Configuring the broker to enable a JMS provider's proprietary API" on page 748
Some JMS providers provide an alternative interface to the standard JMS specification for particular JMS API calls. In these cases, IBM supplies a Java class to interface with that proprietary API.

"Configuring the broker to use SSL with JMS nodes" on page 530
Configure your broker to work with a JMS provider that supports JMS clients that can connect by using the Secure Sockets Layer (SSL) protocol.

"Processing bytes messages with JMS nodes" on page 1729
The default behavior of WebSphere Message Broker when processing bytes messages can affect clients that are designed to use the readUTF() and writeUTF() methods. Construct an equivalent UTF bit stream by using a Compute node.

"Configuring for identity propagation" on page 492
To enable a message flow to perform identity propagation, the input nodes must extract the identity and the output node must propagate it.

"Connecting to different versions of the same JMS provider" on page 750
To use different versions of the same JMS provider, create a configurable service for the JMS provider, and set the jarsURL property to a unique path.

Related reference:

"Representation of messages in the JMS Transport" on page 1691
Messages that are sent in the JMS Transport are represented by the JMS Transport message tree.

"JMSInput node" on page 4532
Use the JMSInput node to receive messages from JMS destinations. JMS destinations are accessed through a connection to a JMS provider.

"JMSReply node" on page 4562
Use the JMSReply node to send messages to JMS destinations.

"JMSMQTransform node" on page 4547
Use the JMSMQTransform node to transform a message with a JMS message tree into a message that has a message tree structure that is compatible with the format of messages that are produced by the WebSphere MQ JMS provider.

"MQJMSTransform node" on page 4610
Use the MQJMSTransform node to receive messages that have a WebSphere MQ JMS provider message tree format, and transform them into a format that is compatible with messages that are to be sent to JMS destinations.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

"Using LocalEnvironment variables with JMSOutput and JMSReply nodes" on page 4242
The LocalEnvironment data elements related to the processing of JMS Messages in the JMSOutput and JMSReply nodes.

Related information:

JMSReply node

Use the JMSReply node to send messages to JMS destinations.

This topic contains the following sections:

Purpose:
The JMSReply node has a similar function to the JMSOutput node, but the JMSReply node sends JMS messages only to the reply destination that is supplied in the JMSReplyTo header field of the JMS message tree. Use the JMSReply node when you want to treat a JMS message that is produced from a message flow as a reply to a JMS input message, and where you have no other routing requirements.

The JMSReply node is contained in the **JMS** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using the JMSReply node in a message flow:
Consider a situation in which you create a message flow in which a JMSInput node message obtains point-to-point messages from a JMS destination called MyJMSInputQueue. The message flow updates a database using the contents of the message, then replies to a JMS destination called MyJMSReplyQueue, which is set by the generating application in the JMSReplyTo header of the input message.

In a similar scenario for the publish/subscribe message model, a JMSInput node subscribes to TopicA, and the JMSReply node publishes on the TopicB destination, which was retrieved from the JMSReplyTo header of the input message.

Calling an output message callback function:
The cciOutputMessageCallback function can be registered as a callback and called whenever a message is sent by a JMSReply node. See "cciOutputMessageCallback" on page 6626.

If the user exit state is active, the cciOutputMessageCallback function is called for every output message that is sent successfully from a JMSReply node where the callback is registered.

If the node provides WrittenDestination information in the LocalEnvironment tree, the callback is called after this information is created. See "Using LocalEnvironment variables with JMSOutput and JMSReply nodes" on page 4242.

Working with the JMS message ID:
The JMS message ID is generated by the JMS provider when a message is sent by the JMSReply node. You cannot set the message ID in the message flow, but you can use one of the following methods to obtain the generated ID after the message has been sent:

- Connect a Compute node to the Out terminal.

Connect a Compute node to the Out terminal of a JMSReply node and interrogate the WrittenDestination List. For more information, see “Viewing the logical message tree in trace output” on page 1481.

An entry for a JMSReply node has the following format:

```
WrittenDestination = (
   JMS = (
      DestinationData = (
        destinationName = 'queue://jmsQueue1'
        initialContext = 'com.sun.jndi.fscontext.RefFSContextFactory'
        JMSMessageID = ID:414d512054657374514d2020202020206ab98b4520017a02'
        JMSCorrelationID = 'ABCDEFGHIJKLMNOPQRSTUVW'
  )
 )
)
```

- Configure a user exit to process an output message callback event. For more information, see “Exploiting user exits” on page 2985.

Terminals and properties:
When you have put an instance of the JMSReply node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view. All mandatory properties that do not have a default value defined are marked with an asterisk.

The terminals of the JMSReply node are described in the following table.

Terminal	Description
Failure	The output terminal to which the message is routed if an error occurs. Even if the Validation property is set, messages that are propagated to this terminal are not validated.
Out	The output terminal to which the message is routed if it is successfully retrieved from the WebSphere MQ queue.

The following tables describe the node properties. The column headed M indicates whether the property is mandatory (marked with an asterisk if you must enter a value when no default is defined), the column headed C indicates whether the property is configurable (you can change the value when you add the message flow to the BAR file to deploy it).

The Description properties of the JMSReply node are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The Basic properties of the JMSReply node are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Send to destination list in local environment	No	Yes	Cleared	When you have built a list of JMS destinations in the local environment, select this check box to use the destination list. If you do not select this check box, the node uses the configured JMS destination. If you select this check box but you have not built a list of JMS destinations in the local environment, the node uses the configured JMS destination.	useDistList

The JMS Connection properties of the JMSReply node are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
JMS provider name	Yes	No	`WebSphere MQ`	Select a JMS vendor name from the list, or enter a name of your choice. When you select a name from the list, the `Initial Context Factory` property is updated automatically with the relevant Java class. If you enter your own JMS provider name, you must also enter a value for the `Initial Context Factory`. The default value is `WebSphere MQ`.	
Initial Context Factory	Yes	Yes	`com.sun.jndi.fscontext. RefFSContextFactory`	This property is the starting point for a JNDI namespace. A JMS application uses the initial context to obtain and look up the connection factory and queue or topic objects for the JMS provider. If you select a JMS provider name from the list in `JMS provider name`, the `Initial Context Factory` property is updated automatically with the relevant Java class. If you enter your own JMS provider name, you must also enter a value for the `Initial Context Factory`. The default value of `com.sun.jndi.fscontext. RefFSContextFactory` defines the file-based initial context factory for the WebSphere MQ JMS provider. If the node is set to use your own JMS Provider, and the corresponding Configurable services property of the **`mqsichangeproperties`** definition has the `InitialContextFactory` attribute set, this overrides the setting on the node.	initialContextFactory

Property	M	C	Default	Description	mqsiapplybaroverride command property
Location JNDI Bindings	Yes	Yes		This property specifies either the file system path or the LDAP location for the bindings file. The bindings file contains definitions for the JNDI-administered objects that are used by the JMSReply node. When you enter a value for Location JNDI Bindings, ensure that it complies with the following instructions: • Construct the bindings file before you deploy a message flow that contains a JMSReply node. • Do not include the file name of the bindings file in this field. • If you have specified an LDAP location that requires authentication, configure both the LDAP principal (userid) and LDAP credentials (password) separately. These values are configured at broker level. For information about configuring these values, refer to the **mqsicreatebroker** and **mqsichangebroker** commands. • The string value must include a supported URL prefix that has a URL handler that is available on the class path. For information about constructing the JNDI-administered objects bindings file, refer to the documentation that is supplied with the JMS provider. If the node is set to use your own JMS Provider, and the corresponding Configurable services property of the **mqsichangeproperties** definition has the `jndiBindingsLocation` attribute set, this overrides the setting on the node.	locationJndiBindings
Connection Factory Name	Yes	Yes		The name of the connection factory that is used by the JMSReply node to create a connection to the JMS provider. This name must already exist in the bindings file.	connectionFactoryName

The Advanced properties of the JMSReply node are described in the following table.

Property	M	C	Default	Description
New Correlation ID	No	Yes	Cleared	If the JMSReply node is required to generate a new Correlation ID for the message, select the check box. The check box is cleared by default; if you leave the check box cleared, the Correlation ID of the output message is taken from the JMSCorrelationID field in the JMSTransport_Header_Values section of the message tree.
Transaction Mode	No	No	`None`	This property controls whether the incoming message is received under sync point. To define the transactional characteristics of how the message is handled, select one of the following values: • Select `None` if the outgoing message is to be treated as non-persistent. If you select this value, the message is sent using a non-transacted JMS session that is created using the `Session.AUTO_ACKNOWLEDGE` parameter. • Select `Local` if the input node that receives the message should coordinate the commit or roll-back of JMS messages that have been sent by the JMSReply node, along with any other resources, such as DB2 or WebSphere MQ, that perform work within the message flow. If you select this value, the node uses a transacted JMS session. • Select `Global` if the JMSReply node should participate in a global message flow transaction that is managed by the broker's external sync point coordinator. The sync point coordinator is the broker's queue manager on distributed systems, and RRS (Resource Recovery Services) on z/OS. If you select this value, any messages that are received by the node are globally coordinated using an XA JMS session.
Delivery Mode	No	Yes	`Automatic`	This property controls the persistence mode that a JMS provider uses for a message. Valid values are: • `Automatic`: the mode from the input message is inherited • `Persistent`: the message survives if the JMS provider has a system failure • `Non-persistent`: the message is lost if the JMS provider has a system failure
Message Expiration (ms)	Yes	Yes	`0`	This property controls the length of time, in milliseconds, for which the JMS provider keeps the output JMS message. The default value, `0`, is used to indicate that the message must not expire. Select `Inherit from header` or enter an integer that represents a number of milliseconds. If you select `Inherit from header`, the property inherits the value of the JMSExpiry field in the JMS message, which is found at the following location: `OutputRoot.JMSTransport.Transport_Folders.Header_Values.JMSExpiration`
Message Priority	No	Yes	`4`	This property assigns relative importance to the message and it can be used for message selection by a receiving JMS client application or a JMSReply node. Select a value between `0` (lowest priority) and `9` (highest priority) or select `Inherit from header`. The default value is `4`, which indicates medium priority. Priorities in the range `0` to `4` relate to normal delivery. Priorities in the range `5` to `9` relate to graduations of expedited delivery. If you select `Inherit from header`, the property inherits the value of the JMSPriority field in the JMS message, which is found at the following location: `OutputRoot.JMSTransport.Transport_Folders.Header_Values.JMSPriority`

Property	M	C	Default	Description
Message type	No	Yes	`TextMessage`	This property controls the class of the JMS output message. The default value is `TextMessage`. Valid values are: • `TextMessage` • `BytesMessage` • `MapMessage` • `StreamMessage` • `ObjectMessage` • `Base JMS message with no payload` If you do not set this property, the node assumes the output type from the metadata PayLoadType field in the JMS message tree.

The Validation properties of the JMSReply node are described in the following table. Refer to "Validation properties" on page 4169 for a full description of these properties.

Property	M	C	Default	Description	**`mqsiapplybaroverride` command property**
Validate	No	Yes	`Inherit`	This property controls whether validation takes place. Valid values are: • `None` • `Content and Value` • `Content` • `Inherit` If a message is propagated to the Failure terminal of the node, it is not validated.	validateMaster
Failure Action	No	No	`Exception`	This property controls what happens if validation fails. You can set this property only if you set `Validate` to `Content` or `Content and Value`. Valid values are: • `User Trace` • `Local Error Log` • `Exception` (default value) • `Exception List`	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

"JMS parsers and domains" on page 1116
The JMSMap and JMSStream domains can be used for modeling messages that are produced by the implementations of the Java Messaging Service standard.

"JMS message transformation" on page 1684
The JMSInput and JMSOutput nodes expect JMS messages, and therefore expect a native JMS message tree representation.

"Configurable services" on page 1296
Configurable services are typically runtime properties. You can use them to define properties that are related to external services on which the broker relies; for example, an SMTP server or a JMS provider.

Related tasks:

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

"Using more than one input node" on page 1473
You can include more than one input node in a single message flow.

"Configuring transactionality for message flows" on page 1290
A message flow runs in a single transaction, which is started when data is received by an input node, and can be committed or rolled back when all processing has completed.

"Configuring the broker to enable a JMS provider's proprietary API" on page 748
Some JMS providers provide an alternative interface to the standard JMS specification for particular JMS API calls. In these cases, IBM supplies a Java class to interface with that proprietary API.

"Configuring the broker to use SSL with JMS nodes" on page 530
Configure your broker to work with a JMS provider that supports JMS clients that can connect by using the Secure Sockets Layer (SSL) protocol.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Validating messages" on page 1478
The broker provides validation based on the message model for predefined messages.

Related reference:

"Validation properties" on page 4169
You can control validation by setting properties on the Validate and Parser Options tabs for the nodes that are listed in the following table.

"JMS message selector" on page 1703
A message selector allows a JMS consumer to be more selective about the messages that it receives from a particular topic or queue.

"JMSInput node" on page 4532
Use the JMSInput node to receive messages from JMS destinations. JMS destinations are accessed through a connection to a JMS provider.

"JMSOutput node" on page 4549
Use the JMSOutput node to send messages to JMS destinations.

"JMSMQTransform node" on page 4547
Use the JMSMQTransform node to transform a message with a JMS message tree into a message that has a message tree structure that is compatible with the format of messages that are produced by the WebSphere MQ JMS provider.

"MQJMSTransform node" on page 4610
Use the MQJMSTransform node to receive messages that have a WebSphere MQ JMS provider message tree format, and transform them into a format that is compatible with messages that are to be sent to JMS destinations.

"Using LocalEnvironment variables with JMSOutput and JMSReply nodes" on page 4242
The LocalEnvironment data elements related to the processing of JMS Messages in the JMSOutput and JMSReply nodes.

Label node

Use the Label node to process a message that is propagated by a RouteToLabel node to dynamically determine the route that the message takes through the message flow.

This topic contains the following sections:

Purpose:
Use the Label node in combination with a RouteToLabel node to route a message through the message flow based on message content. The RouteToLabel node interrogates the LocalEnvironment of the message to determine the identifier of the Label node to which the message must be routed next. You can propagate the message by coding ESQL in a Compute node, or by coding Java in a JavaCompute or user-defined node.

Precede the RouteToLabel node in the message flow with a Compute node or JavaCompute node and populate the LocalEnvironment of the message with the identifiers of one or more Label nodes that introduce the next sequence of processing for the message.

Design your message flow so that a Label node logically follows a RouteToLabel node in a message flow, but do not connect it physically to the RouteToLabel node. The connection is made by the broker, when required, according to the contents of LocalEnvironment.

The Label node provides a target for a routing decision, and does not process the message that it handles in any way. Typically, a Label node connects to a subflow that processes each message in a specific way, and either ends in an output node or in another RouteToLabel node.

The Label node can also be used in conjunction with a SOAPExtract node or as the target of a PROPAGATE statement, which is specified in a Compute or Database node.

The Label node is contained in the **Routing** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:
Look at the following sample to see how to use this node:
- Airline Reservations

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online

information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Terminals and properties:
When you have put an instance of the Label node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The Label node terminals are described in the following table.

Terminal	Description
Out	The output terminal to which the message is routed.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Label node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The Label node Basic properties are described in the following table.

Property	M	C	Default	Description
Label Name	Yes	No		An identifier for the node. It is used as a target for a message that is routed by a RouteToLabel node. `Label Name` must not be the same as the name of the instance of the node itself, and it must be unique in the message flow in which it appears. The name of the instance can be modified by the WebSphere Message Broker Toolkit if the subflow, of which this Label node is a part, is embedded into another message flow.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

"Creating destination lists" on page 1477
Create a list of destinations to indicate where a message is sent.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Using nodes for decision making" on page 2209
You can use several built-in nodes in different ways to control the path that a message takes through the message flow.

"Creating a message processing or output node in Java" on page 3062
A message processing node is used to process a message, and an output node is used to output a message as a bit stream.

Related reference:

"FlowOrder node" on page 4458
Use the FlowOrder node to control the order in which a message is processed by a message flow.

"RouteToLabel node" on page 4673
Use the RouteToLabel node in combination with one or more Label nodes to dynamically determine the route that a message takes through the message flow, based on its content.

"PROPAGATE statement" on page 5150
The PROPAGATE statement propagates a message to the downstream nodes.

Related information:

Java user-defined extensions API

Mapping node

Use the Mapping node to construct one or more new messages and populate them with various types of information.

This topic contains the following sections:
- "Purpose"
- "Using this node in a message flow" on page 4572
- "Terminals and properties" on page 4572

Purpose:
You can populate the new messages with the following types of information:
- New information
- Modified information from the input message
- Information taken from a database

You can modify elements of the message body data, and the local environment tree.

When you first open or create a message map for the node, if you select `This map is called from a message flow node and maps properties and message body`, the headers in the input message are always copied to the output message without modification. To modify the message headers in a Mapping node, select `This map is called from a message flow node and maps properties, headers, and message body`. When you select this property, the map that is created allows additional elements, including WebSphere MQ, HTTP, and JMS headers, to be mapped.

These components of the output message can be defined by using mappings that are based on elements of both the input message and data from an external database. You create the mappings that are associated with this node, in the mapping file that is associated with this node, by mapping inputs (message or database) to outputs. You can modify the assignments made by these mappings by using supplied or user-defined functions and procedures; for example, you can convert a string value to uppercase when you assign it to the message output field.

Use the Mapping node to:
- Build a new message
- Copy messages between parsers
- Transform a message from one format to another

The Mapping node is contained in the **Transformation** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:
Look at the following sample to see how to use this node:
- Pager

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Terminals and properties:
When you have put an instance of the Mapping node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The Mapping node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the input message is propagated if a failure is detected during the computation. If you have selected `Treat Warnings as Errors`, the node propagates the message to this terminal if database warning messages are returned, even though the processing might have completed successfully.

Terminal	Description
Out	The output terminal that propagates the message following completion of the mappings.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Mapping node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The Mapping node Basic properties are described in the following table.

Property	M	C	Default	Description	**mqsiapplybaroverride command property**
Data Source	No	Yes		The ODBC data source name of the database that contains the tables to which you refer in the mappings that are associated with this node (identified by the `Mapping Module` property). This name identifies the appropriate database on the system on which this message flow is to execute. The broker connects to this database with user ID and password information that you have specified on the **mqsicreatebroker**, **mqsichangebroker**, or **mqsisetdbparms** command. If the name is different from the data source name used in the Mapping Editor, select the **Use data source from flow property** option on the **Override Data Source and Schema** dialog (see "Message mapping tips and restrictions" on page 2314). z/OS On z/OS systems, the broker uses the broker started task ID, or the user ID and password that are specified on the **mqsisetdbparms** command JCL, BIPSDBP in the customization data set hlq>.SBIPPROC.	dataSource

Property	M	C	Default	Description	mqsiapplybaroverride command property
Transaction	Yes	No	Automatic	The transaction mode for the node. The values are: • `Automatic` (the default). The message flow, of which the Mapping node is a part, is committed if it is successful; that is, the actions that you define in the mappings are performed and the message continues through the message flow. If the message flow fails, it is rolled back. If you choose `Automatic`, the ability to commit or rollback the action of the Mapping node on the database depends on the success or failure of the entire message flow. • `Commit`. To commit any uncommitted actions that are performed in this message flow on the database that is connected to this node, irrespective of the success or failure of the message flow as a whole, select `Commit`. The changes to the database are committed even if the message flow fails.	
Mapping Routine	Yes	No	Mapping	The name of the mapping routine that contains the statements to execute against the database or the message tree. By default, the name that is assigned to the mapping routine is identical to the name of the mapping file in which the routine is defined. The default name for the file is the name of the message flow concatenated with the name of the node when you include it in the message flow (for example, `MFlow1_Mapping.msgmap` for the first Mapping node in message flow MFlow1). You cannot specify a value that includes spaces. If you click **Browse** next to this entry field, a dialog box is displayed that lists all available mapping routines that this node can access. Select the routine that you want and click **OK**; the routine name is set in `Mapping Module`. To work with the mapping routine that is associated with this node, double-click the node, or right-click the node and click **Open Mappings**. If the mapping routine does not exist, it is created for you with the default name in the default file. If the file exists already, you can also open file `<flow_name>_<node_name>.msgmap` in the Broker Development view. A mapping routine is specific to the type of node with which it is associated; you cannot use a mapping routine that you have developed for a Mapping node with any other node that uses mappings (for example, a DataInsert node). If you create a mapping routine, you cannot call it from any other mapping routine, although you can call it from an ESQL routine. For more information about working with mapping files, and defining their content, see “Using message mappings” on page 2228.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Mapping Mode	Yes	No	Message	The mode that is used to process information that is passed through the Mapping node. Valid values are: • `Message` (the default): the message is generated or passed through by the Mapping node, as modified within the node. • `LocalEnvironment`: the LocalEnvironment tree structure is generated or passed through by the Mapping node, as modified within the node. • `LocalEnvironment And Message`: the LocalEnvironment tree structure and message are generated or passed through by the Mapping node, as modified by the node. • `Exception`: the ExceptionList is passed through by the Mapping node unchanged. • `Exception And Message`: the ExceptionList is passed through unchanged, and the message is generated or passed through by the Mapping node, as modified by the node. • `Exception And LocalEnvironment`: the ExceptionList is passed through unchanged, and the LocalEnvironment tree is generated or passed through by the Mapping node, as modified by the node. • `All`: the ExceptionList is passed through unchanged, the message and the localEnvironment are generated or passed through by the Mapping node, as modified by the node. You must set this property to reflect accurately the output message format that you need. If you select an option (or accept the default value) that does not include a particular component of the message, that component is not included in any output message that is constructed. You can choose any combination of `Message`, `LocalEnvironment`, and `Exception` components to be generated and modified by the Mapping node. To construct a map that propagates multiple target messages, set this property to `LocalEnvironment And Message` to ensure that the node executes correctly. LocalEnvironment was known as DestinationList in some previous versions; it is retained for compatibility. The Environment component of the message tree is not affected by the mode setting. Its contents, if any, are passed on from this node.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Treat Warnings as Errors	Yes	No	Cleared	For database warning messages to be treated as errors, and the node to propagate the output message to the Failure terminal, select `Treat Warnings as Errors`. The check box is cleared initially. When you select the check box, the node handles all positive return codes from the database as errors and generates exceptions in the same way as it does for the negative, or more serious, errors. If you do not select the check box, the node treats warnings as normal return codes, and does not raise any exceptions. The most significant warning raised is `not found`, which can be handled safely as a normal return code in most circumstances.	
Throw Exception on Database Error	Yes	No	Selected	For the broker to generate an exception when a database error is detected, select `Throw Exception on Database Error`. The check box is selected initially. If you clear the check box, you must handle the error in the message flow to ensure the integrity of the broker and the database. The error is ignored if you do not handle it through your own processing, because you have chosen not to invoke the default error handling by the broker. For example, you could connect the Failure terminal to an error processing subroutine.	

The parser options for the Mapping node are described in the following table.

Property	M	C	Default	Description
Use XMLNSC Compact Parser for XMLNS Domain	No	No	Cleared	If you select this check box, the outgoing MQRFH2 specifies the XMLNS instead of XMLNSC parser, allowing an external application to remain unchanged. If outgoing messages do not contain MQRFH2 headers, this property has no effect.

The Validation properties of Mapping node are described in the following table.

If a message is propagated to the Failure terminal of the node, it is not validated. These properties do not cause the input message to be validated. It is expected that, if such validation is required, the validation has already been performed by the input node or a preceding validation node. For more details about validating messages and validation properties, see “Validating messages” on page 1478 and “Validation properties” on page 4169.

Property	M	C	Default	Description
Validate	No	Yes	None	This property controls whether validation takes place. Valid values are None, Content and Value, Content, and Inherit.
Failure Action	No	No	Exception	This property controls what happens if a validation failure occurs. You can set this property only if Validate is set to Content or Content and Value. Valid values are User Trace, Local Error Log, Exception, and Exception List.

Related concepts:

“Message flows overview” on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Client application programming interfaces" on page 1038
You can configure the nodes in your message flows to customize the behavior of those nodes by using one or more of the supported programming interfaces.

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

Related tasks:

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

"Configuring transactionality for message flows" on page 1290
A message flow runs in a single transaction, which is started when data is received by an input node, and can be committed or rolled back when all processing has completed.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Using message mappings" on page 2228
Message mappings define the blueprint for creating a message.

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

"Validating messages" on page 1478
The broker provides validation based on the message model for predefined messages.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"DataInsert node" on page 4386
Use the DataInsert node to interact with a database in the specified ODBC data source.

"Validation properties" on page 4169
You can control validation by setting properties on the Validate and Parser Options tabs for the nodes that are listed in the following table.

"Parsing on demand" on page 4173
On-demand parsing, referred to as partial parsing, is used to parse an input message bit stream only as far as is necessary to satisfy the current reference. The parsers that can perform partial parsing of input messages are the JSON, MRM, XML, XMLNS, and XMLNSC parsers.

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

MQGet node

Use the MQGet node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and the MQI and AMI application programming interfaces.

You can also use the MQGet node to retrieve messages that were previously placed in a WebSphere MQ message queue that is defined to the broker queue manager.

This topic contains the following sections:

The topic uses the following terms:

input message
A message that enters the In terminal of the MQGet node.

queue message
A message that the MQGet node reads from the queue.

Purpose:
The MQGet node reads a message from a specified queue, and establishes the processing environment for the message. If appropriate, you can define the input queue as a WebSphere MQ clustered queue or shared queue.

You can use an MQGet node anywhere in a message flow, unlike an MQInput node, which you can use only as the first node in a message flow. The output message tree from an MQGet node is constructed by combining the input tree with the result tree from the MQGET call. You can set the properties of the MQGet node to control the way in which messages are received; for example, you can indicate that a message is to be processed under transaction control, or you can request that, when the result tree is being created, data conversion is performed on receipt of every input message.

The MQGet node handles messages in the following message domains:
- XMLNSC
- DataObject
- JSON
- BLOB
- MIME
- MRM
- JMSMap
- JMSStream
- XMLNS

The MQGet node is contained in the **WebSphere MQ** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using the MQGet node in a message flow:

For information about how to use the MQGet node in a message flow, see "A request-response scenario that uses an MQGet node" on page 1569.

Look at the following sample to see how to browse messages with the MQGet node:

- Browsing WebSphere MQ Queues

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Configuring the MQGet node:
When you have put an instance of the MQGet node into a message flow, you can configure it; for more information, see "Configuring a message flow node" on page 1503. The properties of the node are displayed in the Properties view.

All mandatory properties for which you must enter a value (those properties that do not have a default value defined) are marked with an asterisk.

Configuring for coordinated transactions:
When you include an MQGet node in a message flow, the value that you set for `Transaction mode` defines whether messages are received under sync point.

- If you set the property to `Yes` (the default), the queue message is received under sync point (that is, in a WebSphere MQ unit of work). Any messages that an output node in the same instance of the message flow sends later are put under sync point, unless the output node, or any other subsequent node, overrides this setting explicitly.
- If you set the property to `Automatic`, the queue message is received under sync point if the incoming message is marked as persistent. Otherwise, it is not received under sync point. Any message that is sent later by an output node is put under sync point, as determined by the incoming persistence property, unless the output node, or any other subsequent node, overrides this setting explicitly.
- If you set the property to `No`, the queue message is not received under sync point. Any messages that are sent later by an output node in the message flow are not put under sync point, unless an individual output node, or any other subsequent node, specifies that the message must be put under sync point.

If you set the `Browse only` property, the value that you set for the `Transaction mode` property is ignored because a message cannot be browsed under sync point. However, any derived messages that are propagated later by an output node in the same instance of the message flow follow the behavior that is described previously in accordance with the specified `Transaction mode` value.

Overriding node properties during message processing:
When you include and configure an MQGet node in a message flow, you might want to override its properties under some conditions. For example, you might want to read from a queue that is identified in another part of the message, or that is retrieved from a database record.

To override the values that you set for the MQGet node properties to achieve a more dynamic way to process messages, include a Compute or JavaCompute node in your message flow before the MQGet node. Configure this node to create an

output message, and add fields to the local environment tree to define new values for the properties that you want to change.

For example, add a Compute node into the flow and define a new queue name for the MQGet node to read for messages, by including the following ESQL statement:

```
SET LocalEnvironment.MQ.GET.QueueName = 'new_queue';
```

Use `LocalEnvironment.MQ.GET.` as the correlation name for all fields that relate to the MQGet node.

You can set the following properties under the InputLocalEnvironment.MQ.GET tree. If you have set a value for the `Input MQ parameters location` property on the MQGet node, that tree location is checked instead.

Setting	Description
QueueName	This setting overrides the MQGet node `Queue name` property; for example: `SET InputLocalEnvironment.MQ.GET.QueueName = 'myQueue';`
InitialBufferSize	This setting overrides the MQGet node `Minimum message buffer size` property; for example: `SET InputLocalEnvironment.MQ.GET.InitialBufferSize = 1024;`

MQGMO.*	This setting overrides the MQGET message options used by the MQGet node; for example:
	`SET InputLocalEnvironment.MQ.GET.MQGMO.Options = MQGMO_ACCEPT_TRUNCATED_MSG;`
	`SET InputLocalEnvironment.MQ.GET.MQGMO.WaitInterval = 10000;`
	
	This override is provided for flexibility, but should be used with caution because the MQGMO is used exactly as specified. Node properties and other local environment overrides are not considered.
	
	For more information about the MQGMO structure, see the WebSphere MQ Version 7 Information Center online.

Connecting the terminals:
Connect the Out, Warning, Failure, and No Message output terminals of this node to another node in the message flow to process the message further, process errors, or send the message to an additional destination.

The completion code (CC) that is generated by the MQGET call controls what is propagated to each of the output terminals.

- If the MQGET call is successful, the MQGet node routes each parsed output message to the Out terminal.
- If the MQGET call fails, but with a CC that indicates a warning, an unparsed output message is propagated to the Warning terminal.
- If the MQGET call fails with a CC more severe than a warning, the input message is propagated to the Failure terminal.
- If the MQGET call fails with a reason code of MQRC_NO_MSG_AVAILABLE, the output message is propagated (without a result body) to the No Message terminal. The output message that is propagated to the No Message terminal is constructed from the input message only, according to the values of the `Generate mode`, `Copy message`, and `Copy local environment` properties.
- If you do not connect the Out, Warning, or No Message terminals to another node in the message flow, any message that is propagated to those terminals is discarded.

- If you do not connect the Failure terminal to another node in the message flow, the broker generates an exception when a message is propagated to that terminal.

For more information, see "Connecting failure terminals" on page 2827.

Terminals and properties:
The terminals of the MQGet node are described in the following table.

Terminal	Description
In	The input terminal that accepts the message that is being processed by the message flow.
Warning	The output terminal to which the output tree is propagated if an error (with a CC that indicates a warning) occurs in the node while trying to get a message from the queue. The MQMD part of the message is parsed, but the rest of the message is an unparsed BLOB element. The warning is discarded if the terminal is not connected, and there is no output propagation from the node at all.
Failure	The output terminal to which the input message is routed if an error (with a CC that indicates an error that is more severe than a warning) occurs in the node while trying to get a message from the queue.
Out	The output terminal to which the message is routed if it is retrieved successfully from the WebSphere MQ queue.
No Message	The output terminal to which the input message is routed if no message is available on the queue. The output message that is propagated to the No Message terminal is constructed from the input message only, according to the values of the `Generate mode`, `Copy message`, and `Copy local environment` properties.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Description properties of the MQGet node are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, MQGet	The name of the node.
Short description	No	No	Blank	A brief description of the node.
Long description	No	No	Blank	Text that describes the purpose of the node in the message flow.

The Basic properties of the MQGet node are described in the following table.

Property	M	C	Default	Description	`mqsiapplybaroverride` command property
Queue name	Yes	Yes	None	The name of the WebSphere MQ message queue from which this node retrieves messages. You must predefine this queue to the queue manager that hosts the broker on which the message flow is deployed. If this queue is not a valid queue, the node generates an exception, and the input message is propagated to the Failure terminal.	queueName

The MQGet node Input Message Parsing properties are described in the following table.

If the queue message has an MQRFH2 header, you do not have to set values for the Input Message Parsing properties, because the values can be derived from the <mcd> folder in the MQRFH2 header; for example:

```
<mcd><Msd>MRM</Msd><Set>DHM4UO906S001</Set><Type>receiptmsg1</Type>
<Fmt>XML</Fmt></mcd>
```

If you set values, and the values differ from those values in the MQRFH2 header, the values in the MQRFH2 header take precedence.

Property	M	C	Default	Description
Message domain	No	No	BLOB	The domain that is used to parse the queue message. If the MQRFH2 header does not supply the Message domain value, then you can select a value from the list. If you do not select a value, the default value is BLOB. You can also specify a user-defined parser, if appropriate.
Message set	No	No	None	The name or identifier of the message set in which the queue message is defined. If you are using the MRM or IDOC parser, or the XMLNSC parser in validating mode, select the Message set that you want to use. If you set this property, then later update the project dependencies to remove this message set reference, a warning is issued. Either update the Message set property, or restore the reference to this message set project.
Message type	No	No	None	The name of the queue message. If you are using the MRM parser, select the correct message from the list in Message type. This list is populated with messages that are defined in the Message set that you have selected.
Message format	No	No	None	The name of the physical format of the queue message. If you are using the MRM or IDOC parser, select the format of the message from the list in Message format. This list includes all the physical formats that you have defined for this Message set. If you set the Message domain property to DataObject, you can set this property to XML or SAP ALE IDoc. Set this property to SAP ALE IDoc when you have to parse a bit stream from an external source and generate a message tree.

The Parser Options properties of the MQGet node are described in the following table.

Property	M	C	Default	Description
Parse timing	No	No	On Demand	This property controls when the queue message is parsed. Valid values are On Demand, Immediate, and Complete. By default, this property is set to On Demand, which causes parsing of the message to be delayed. To cause the message to be parsed immediately, see “Parsing on demand” on page 4173.
Use MQRFH2C compact parser for MQRFH2 header	No	No	Cleared	This property controls whether the MQRFH2C compact parser, instead of the MQRFH2 parser, is used for MQRFH2 headers. Select Use MQRFH2C compact parser for MQRFH2 header if you want the MQRFH2C parser to be used. By default, this check box is cleared, which means that the compact parser is not used.
Build tree using XML schema data types	No	No	Cleared	This property controls whether the XMLNSC parser creates syntax elements in the message tree with data types taken from the XML schema. You can select this property only if you set the Validate property on the **Validation** tab to Content or Content and Value. For more information about XMLNSC, see “Manipulating messages in the XMLNSC domain” on page 2546.

Property	M	C	Default	Description
Use XMLNSC compact parser for XMLNS domain	No	No	Cleared	This property controls whether the XMLNSC Compact Parser is used for messages in the XMLNS Domain. If you set this property, the message data is displayed under XMLNSC in nodes that are connected to the output terminal when the input MQRFH2 header or Input Message Parsing properties `Message domain` is XMLNS.
Retain mixed content	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters mixed text in the queue message message. If you select the check box, elements are created for mixed text. If you clear the check box, mixed text is ignored and no elements are created.
Retain comments	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters comments in the queue message message. If you select the check box, elements are created for comments. If you clear the check box, comments are ignored and no elements are created.
Retain processing instructions	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters processing instructions in the queue message. If you select the check box, elements are created for processing instructions. If you clear the check box, processing instructions are ignored and no elements are created.
Opaque elements	No	No	Blank	This property is used to specify a list of elements in the queue message that are to be opaquely parsed by the XMLNSC parser. Opaque parsing is performed only if validation is not enabled (that is, if `Validate` is `None`); entries that are specified in `Opaque Elements` are ignored if validation is enabled.

The Advanced properties of the MQGet node are described in the following table.

Property	M	C	Default	Description
Transaction mode	No	No	`Yes`	This property controls whether the incoming message is received under sync point. Select a value for `Transaction mode` from the list to define the transactional characteristics of how this message is handled: • If you select `Automatic`, the queue message is received under sync point if it is marked as persistent. If the message is not marked as persistent, it is not received under sync point. The persistence or non-persistence of the input message determines the transactionality of any derived messages that are later propagated by an output node, unless the output node, or any other subsequent node in the message flow, overrides the transactionality explicitly. • If you select `Yes`, the queue message is received under sync point. Any derived messages that are later propagated by an output node in the same instance of the message flow are sent transactionally, unless the output node, or any other subsequent node in the message flow, overrides the transactionality explicitly. • If you select `No`, the queue message is not received under sync point. Any derived messages that are later propagated by an output node in the same instance of the message flow are sent non-transactionally, unless the output node, or any other subsequent node in the message flow, has specified that the messages must be put under sync point.

Property	M	C	Default	Description
Generate mode	No	No	`Message`	This property controls which parts of the message from the input tree are copied. Select a value for `Generate mode` from the list to define which components of the output message are generated in the MQGet node, and which components are taken from the input message. • If you select `None`, all the components of the message from the input tree are propagated unchanged. • If you select `Message` (the default), a new Message component is created by the node, but the local environment, environment, and exception list components from the input tree are propagated unchanged. • If you select `LocalEnvironment`, a new local environment component is created by the node, but the message, environment, and exception list components from the input tree are propagated unchanged. • If you select `Message and LocalEnvironment`, new message and local environment components are created by the node, but the environment and exception list components from the input tree are propagated unchanged.
Copy message	No	No	`None`	This property controls which parts of the message from the input tree are copied. If you have set `Generate mode` to either `Message` or `Message And LocalEnvironment`, select a value for `Copy message` from the list to define which parts of the message are generated in the MQGet node, and which parts are taken from the input message. • If you select `None` (the default), no part of the input message from the input tree is propagated. • If you select `Copy Headers`, the headers from the input message in the input tree are copied to the output message. • If you select `Copy Entire Message`, the entire input message from the input tree is copied to the output message.
Copy local environment	No	No	`Copy Entire LocalEnvironment`	This property controls how the local environment is copied to the output message. If you have set `Generate mode` to either `LocalEnvironment` or `Message And LocalEnvironment`, select a value for `Copy Local Environment` from the list to define which parts of the local environment are generated in the MQGet node, and which parts are taken from the input message. • If you select `Copy Entire LocalEnvironment` (the default), at each node in the message flow, a new copy of the local environment is created in the tree, and it is populated with the contents of the local environment from the preceding node. Therefore, if a node changes the local environment, the upstream nodes do not see those changes because they have their own copies. This behavior might be an issue if you are using a FlowOrder node, or if you use the **`propagate`** command on a Compute node. The entire local environment that is defined in the input message is copied to the output message. • If you select `None`, each node does not generate its own copy of the local environment, but it uses the local environment that is passed to it by the previous node. Therefore, if a node changes the local environment, those changes are seen by the upstream nodes.

Property	M	C	Default	Description
Wait interval (ms)	Yes	No	1000	The maximum time, in milliseconds, to wait for the queue message to be obtained from the message queue. Provide a value for the `Wait interval (ms)` property to specify how many milliseconds to wait for a message to be received from the MQGET call. If you select `0`, the wait interval is disabled and there is no wait time for messages. The `Wait interval (ms)` value cannot be negative. If you do not provide a value, the default value of `1000` milliseconds is used.
Minimum message buffer size (KB)	Yes	No	4	The minimum size, in KB, of the get buffer. The minimum value of this property is 1. Provide a value for this property to specify the size of the initial buffer for the MQGET call. The buffer expands automatically to accept a message of any size, but if messages are likely to be large, specify a suitable value to reduce the frequency of the buffer being resized. If you do not provide a value, the size of the buffer is 4 KB.

The Request properties of the MQGet node are described in the following table.

Property	M	C	Default	Description
Input MQMD location	No	No	`InputRoot.MQMD`	The location in the input message assembly where the MQMD that is to be used for the MQGET can be found. The default location is InputRoot.MQMD.
Input MQ parameters location	No	No	`InputLocalEnvironment.MQ.GET`	The location in the input message assembly where the WebSphere MQ parameters (for example, the initial buffer size and the MQGMO overrides) can be found. The default location is InputLocalEnvironment.MQ.GET.
Get by correlation ID	No	No	Cleared	If you select this check box, only messages that have the specified correlation ID are retrieved. If you select `Get by correlation ID`, the `CorrelId` field of the message to be retrieved must match the `CorrelId` field in the `Input MQMD location`. By default, this check box is cleared. Setting the `CorrelId` field to MQCI_NONE has the same effect as not selecting `Get by correlation ID`.
Get by message ID	No	No	Cleared	If you select this check box, only messages that have the specified message ID are retrieved. If you select `Get by message ID`, the `MsgId` field of the message to be retrieved must match the `MsgId` field in the `Input MQMD location`. By default, this check box is cleared.

Property	M	C	Default	Description
Use all input MQMD fields	No	No	Cleared	If you select `Use all input MQMD fields`, all MQMD fields at the `Input MQMD location` are used to retrieve the message. If an MQMD bit stream is present at the `Input MQMD location`, all fields in the bit stream are used. Make sure that the MQMD of the message to be retrieved matches these fields. By default, this check box is cleared. If you do not supply an input MQMD, the default MQMD is used. If you do supply an input MQMD, the default MQMD is used after the following modifications: • If you set the property `Use all input MQMD fields`, all MQMD fields supplied are copied into the default MQMD from the input MQMD. • If you do not set the property `Use all input MQMD fields`, and the properties `Get by Message ID` or `Get by Correlation ID` are selected, the respective IDs are copied into the default MQMD from the input MQMD. For more information about how the MQMD for the MQGET call is constructed, see "A request-response scenario that uses an MQGet node" on page 1569.
Browse only	No	No	Cleared	This property controls whether a message is removed from the queue when it is read. If this check box is selected, the message is not removed from the queue when it is read. Select `Browse only` to specify that the message must be retained on the queue when it is read.
Reset browse cursor	No	No	Cleared	You can set this property only if you have selected `Browse only`. When you select `Reset browse cursor`, the node browses from the start of the MQ queue (that is, the MQGMO_BROWSE_FIRST MQ get option is specified). If you do not select this property, the node browses from the current cursor position in the MQ queue (that is, the MQGMO_BROWSE_NEXT MQ get option is specified).

The Result properties of the MQGet node are described in the following table. Set these properties to determine how the results of the MQGET call are handled.

Property	M	C	Default	Description
Output data location	No	No	`OutputRoot`	This property specifies where the output data is placed. If you leave the field blank, `OutputRoot` is used as a default. Enter the start location in the output message tree at which the parsed elements from the bit string of the queue message are stored. All elements at this location are deleted, and the default behavior is to replace the input tree message with the queue message. You can enter any valid ESQL field reference (this reference can include expressions), including new field references to create a node in the message tree for inserting the response into the message that is propagated from the input tree. For example, `OutputRoot.XMLNS.ABC.DEF` and `Environment.GotReply` are valid field references. For more detailed information, see "A request-response scenario that uses an MQGet node" on page 1569. When the queue message bit string is parsed to create the contents of the message tree, the message properties that you have specified as the Input Message Parsing properties of the node are used.
Result data location	No	No	`ResultRoot`	This property specifies which subtree (of the queue message) to use. If you leave this field blank, `ResultRoot` is used as a default, and the whole queue message is used. If, for example, ResultRoot.MQMD.ReplyToQ is specified, only that subtree is used. Set this property to control which subtree of the queue message is placed in the output message. If, for example, you want only the MQMD from the queue message, use `ResultRoot.MQMD`; this subtree is then placed at the location specified by `Output data location`.
Output MQ parameters location	No	No	`OutputLocalEnvironment.MQ.GET`	This property specifies where the output WebSphere MQ parameters are located. If you leave this field blank, OutputLocalEnvironment.MQ.GET is used as a default. Set `Generate mode` to include `LocalEnvironment` to ensure that the updated values are visible in downstream nodes. The default location is `OutputLocalEnvironment.MQ.GET`. Set this property to control where the CC (completion code), the RC (reason code), the Browsed indicator, and any other WebSphere MQ parameters (for example, the MQMD that is used by the MQGET call) are placed in the output tree.

Property	M	C	Default	Description
Warning data location	No	No	`OutputRoot`	This property specifies where the output data is placed if MQGET returns a warning code. If you leave this field blank, `OutputRoot` is used as a default. Set this property to control where the queue message is placed when the MQGET call returns a warning code. You can enter any valid ESQL field reference (see the description of the `Output data location` property). The data that is placed at this location is always the complete result tree, with the body as a BLOB element. `Result data location` is not used for warning data.
Include message contents in output message assembly	No	No	Selected	This property specifies that no result or warning data is required for the output message assembly. If you select this check box, the node gets or browses the message on the queue without completely reading or parsing its contents. If you select `Include message contents in output message assembly`, the message contents are not guaranteed to be included in the output tree because this inclusion depends on other node properties, such as the `Generate mode` property. Clear `Include message contents in output message assembly` to specify that no result or warning data is required for the output message assembly. This action gets or browses the message on the queue without reading or parsing its contents.

The Validation properties of the MQGet node are described in the following table. For a full description of these properties, see "Validation properties" on page 4169.

Property	M	C	Default	Description	**`mqsiapplybaroverride` command property**
Validate	No	Yes	`None`	This property controls whether validation takes place. Valid values are `None`, `Content`, `Content and Value`, and `Inherit`.	validateMaster
Failure action	No	No	`Exception`	This property controls what happens if validation fails. You can set this property only if you set `Validate` to `Content` or `Content and Value`. Valid values are `User Trace`, `Local Error Log`, `Exception`, and `Exception List`.	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Parsers" on page 1072
A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from the internal message tree representation.

"Specifying namespaces in the Message Type property" on page 1208
When using the MRM domain, the *Message Type* property is used to specify the name of the message.

"Multipart messages" on page 1191
A *multipart message* contains one or more other messages within its structure. The contained message is sometimes referred to as an *embedded message*.

Related tasks:

"Using MQGet nodes" on page 1564
The MQGet node processes messages in a particular way, and you can use it in request-response message flows.

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Validating messages" on page 1478
The broker provides validation based on the message model for predefined messages.

"Manipulating messages in the XMLNSC domain" on page 2546
If you are writing ESQL to process messages in the XMLNSC domain, it is helpful to learn about the structure of the message tree that the XMLNSC parser builds.

"Connecting failure terminals" on page 2827
When a node that has a failure terminal detects an internal error, it propagates the message to that terminal. If it does not have a failure terminal, or if you have not connected the failure terminal, the broker generates an exception.

"Using WebSphere MQ cluster queues for input and output" on page 1544
Design your broker network to use WebSphere MQ queues, if appropriate for your business needs.

"Using WebSphere MQ shared queues for input and output (z/OS)" on page 1546
On z/OS systems, you can define WebSphere MQ shared queues as input and output queues for message flows. You might need to serialize access to those messages.

Related reference:

"WebSphere MQ Enterprise Transport" on page 1542
WebSphere MQ Enterprise Transport is a service that connects applications to messaging middleware.

"Validation properties" on page 4169
You can control validation by setting properties on the Validate and Parser Options tabs for the nodes that are listed in the following table.

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

"MQOutput node" on page 4612
Use the MQOutput node to send messages to clients that connect to the broker using the WebSphere MQ Enterprise Transport and that use the MQI and AMI application programming interfaces.

Related information:

WebSphere MQ Version 7 Information Center online

MQHeader node

Use the MQHeader node to add, modify, or delete MQ Message Descriptor (MQMD) and MQ Dead Letter Header (MQDLH) headers.

This topic contains the following sections:
- "Purpose"
- "Using this node in a message flow"
- "Terminals and properties"

Purpose:
You can add or remove a whole header, or you can change only certain fields in a header. You can set these fields to a fixed value, or to a value specified by an XPath expression to access a value in one of the message trees. XPath is used to provide a valid location from which a value for a property can be copied. For example, the location can be the body of the message, the local environment tree, or an exception list.

The MQHeader node is contained in the **WebSphere MQ** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:
Look at the following sample for more details about how to use the node:
- MQHeader node

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Terminals and properties:
When you have put an instance of the node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view. This node has no mandatory properties.

MQHeader node terminals are described in the following table:

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the input message is routed if a failure is detected.

Terminal	Description
Out	The output terminal to which the transformed message is routed if the input message is processed successfully.

The following tables describes the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The MQHeader node Description properties are described in the following table:

Property	M	C	Default	Description
Node name	No	No	MQHeader	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The MQ Message Descriptor properties are described in the following table: Refer to *WebSphere MQ Application Programming Reference* and *WebSphere MQ Application Programming Guide* for full details of each of the MQ property and its supported values.

Property	M	C	Default	Description	mqsiapplybaroverride command property
MQMD header options	No	No	`Carry forward header`	Options to control the MQMD as a whole. Select `Carry forward header` to carry forward any values that are present in an incoming message. Select `Add header` to add a new header using the specified property values. If a header already exists, the header is modified using the specified property values. If `Inherit from header` is specified as a property value and the header does not exist, the default value for the property is used. Select `Modify header` to change an existing header using the specified property values. If a header does not exist, a new header is added first. If `Inherit from header` is specified as a property value and the header does not exist, the default value for the property is used. Select `Delete header` to delete the header, if it exists. **Note:** The `Add header` and `Modify header` options both add a header if it does not exist, and change a header if it does exist. However, the default values offered by each option differ, so use the appropriate option.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Coded Character Set Identifer	No	No	MQCCSI_Q_MGR	The character set identifier of character data in the message. A sample set of custom literals for EBCIDIC and other common Unicode values is given here: MQCCSI_INTL_EBCIDIC : 500 MQCCSI_US_EBCIDIC : 037 MQCCSI_UNICODE_1200 : 1200 MQCCSI_UNICODE_1208 : 1208 MQCCSI_UNICODE_13488 : 13488 MQCCSI_UNICODE_17584 : 17584 Refer to the *WebSphere MQ Application Programming Reference* and *WebSphere MQ Application Programming Guide* for full details.	
Format	No	No	MQFMT_NONE	A name that the sender of the message can use to indicate to the receiver the nature of the data in the message.	
Version Number	No	No	MQMD_VERSION_1	The version ID of the MQMD message.	
Message Type	No	No	MQMT_DATAGRAM	The message type.	
Message Expiry	No	No	MQEI_UNLIMITED	A period of time expressed in tenths of a second, set by the application that puts the message. The message becomes eligible to be discarded if it has not been removed from the destination queue before this period of time elapses.	
Feedback or Reason Code	No	No	MQFB_NONE	Used with a message of type MQMT_REPORT to indicate the nature of the report, and meaningful only with that type of message.	
Message Priority	No	No	MQPRI_PRIORITY_AS_Q_DEF	Message priority. 0 is the lowest value, and 9 is the highest. Custom display literals are as follows: MQPRI_PRIORITY_HIGH : 9 MQPRI_PRIORITY_8 : 8 MQPRI_PRIORITY_7 : 7 MQPRI_PRIORITY_6 : 6 MQPRI_PRIORITY_5 : 5 MQPRI_PRIORITY_MEDIUM : 4 MQPRI_PRIORITY_3 : 3 MQPRI_PRIORITY_2 : 2 MQPRI_PRIORITY_1 : 1 MQPRI_PRIORITY_LOW : 0	
Message Persistence	No	No	MQPER_PERSISTENCE_AS_Q_DEF	Indicates whether the message survives system failures and restarts of the queue manager.	
Message Identifier	No	No	MQMI_NONE	A string that is used to distinguish one message from another.	
Correlation Identifier	No	No	MQCI_NONE	A string that the application can use to relate one message to another, or to relate the message to other work that the application is performing.	
Reply To Queue	No	Yes	<No default value>	The message queue to which the application that issued the get request for the message should send Reply and Report messages.	mqmdReplyToQ
Reply To Queue Manager	No	Yes	<No default value>	The queue manager to which the reply message or report message should be sent.	mqmdReplyToQMgr

The Report properties are described in the following table:

Property	M	C	Default	Description
Inherit from header	No	No	Selected	This property is enabled only when the `Modify header` option is selected. Select this field to inherit any MQMD report property value that is present in an incoming message.
Exception	No	No	No default value	A type of MQ report message. Exception report message is generated.
Expiration	No	No	No default value	A type of MQ report message. Expiration report message is generated.
Confirm on arrival	No	No	No default value	A type of MQ report message. Confirm on arrival report message is generated.
Confirm on delivery	No	No	No default value	A type of MQ report message. Confirm on delivery report message is generated.
Notification	No	No	No default value	A type of MQ report Message. Action notification report message is generated.

The MQDLH header properties are described in the following table:

Property	M	C	Default	Description	`mqsiapplybaroverride` command property
MQDLH header options	No	No	`Carry forward the header`	Options to control the MQMD as a whole. Select `Carry forward header` to carry forward any values that are present in an incoming message. Select `Add header` to add a new header using the specified property values. If a header already exists, the header is modified using the specified property values. If `Inherit from header` is specified as a property value and the header does not exist, the default value for the property is used. Select `Modify header` to change an existing header using the specified property values. If a header does not exist, a new header is added first. If *Inherit from header* is specified as a property value and the header does not exist, the default value for the property is used. Select `Delete header` to delete the header, if it exists. **Note:** The `Add header` and `Modify header` options both add a header if it does not exist, and change a header if it does exist. However, the default values offered by each option differ, so use the appropriate option.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Coded Character Set Identifer	No	No	MQCCSI_UNDEFINED	The character set identifier of character data in the message.	
Format	No	No	MQFMT_NONE	A name that the sender of the message can use to indicate to the receiver the nature of the data in the message.	
Reason	No	No	MQRC_NONE	A code that indicates why the message is sent to the dead letter queue (DLQ).	
Destination Queue Name	No	Yes	No default value	The name of the destination queue.	mqdlhDestQName
Destination Queue Manager Name	No	Yes	No default value	The name of the destination queue manager.	mqdlhDestQMgrName
Save dead letter queue	No	No	Selected	If selected, the dead letter queue name is stored in the local environment.	
Save source queue	No	No	Selected	If selected, the original source queue name is stored in the local environment.	
Put Application Name	No	No	The name of the application that put the message on the dead-letter queue.	This property is set to WebSphereMQIntegrator and appended with the broker major version number, for example: WebSphereMQIntegrator9.	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

MQInput node

Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

This topic contains the following sections:

Purpose:
The MQInput node receives a message from a WebSphere MQ message queue that is defined on the queue manager of the broker. The node uses MQGET to read a message from a specified queue, and establishes the processing environment for the message. If appropriate, you can define the input queue as a WebSphere MQ clustered queue or shared queue.

Message flows that handle messages that are received across WebSphere MQ connections must always start with an MQInput node. You can set the properties of the MQInput node to control the way in which messages are received, by causing appropriate MQGET options to be set. For example, you can indicate that a message is to be processed under transaction control. You can also request that data conversion is performed on receipt of every input message.

The MQInput node handles messages in the following message domains:
- XMLNSC
- DataObject
- JSON
- BLOB
- MIME
- MRM
- JMSMap
- JMSStream
- XMLNS

If you include an output node in a message flow that starts with an MQInput node, the output node can be any one of the supported output nodes, including user-defined output nodes; you do not have to include an MQOutput node. You can create a message flow that receives messages from WebSphere MQ clients and generates messages for clients that use any of the supported transports to connect to the broker, because you can configure the message flow to request that the broker provides any conversion that is necessary.

If you create a message flow to use as a subflow, you cannot use a standard input node; you must use an Input node as the first node to create an In terminal for the subflow.

If your message flow does not receive messages across WebSphere MQ connections, you can choose one of the supported input nodes.

The MQInput node is contained in the **WebSphere MQ** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using the MQInput node in a message flow:
Look at the following samples to see how to use the MQInput node:
- Pager
- Airline Reservations
- Error Handler
- Aggregation
- JMS Nodes
- Large Messaging

- Message Routing
- Scribble
- Timeout Processing
- Video Rental
- XSL Transform
- Browsing WebSphere MQ Queues

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Connecting the terminals:
The MQInput node routes each message that it retrieves successfully to the Out terminal. If this action fails, the message is tried again. If the backout count is exceeded (as defined by the BackoutThreshold attribute of the input queue), the message is routed to the Failure terminal; you can connect nodes to this terminal to handle this condition. If you have not connected the Failure terminal, the message is written to the backout queue.

If the message is caught by this node after an exception has been thrown further on in the message flow, the message is routed to the Catch terminal. If you have not connected the Catch terminal, the message loops continually through the node until the problem is resolved.

You must define a backout queue or a dead-letter queue (DLQ) to prevent the message from looping continually through the node.

Configuring for coordinated transactions:
When you include an MQInput node in a message flow, the value that you set for `Transaction mode` defines whether messages are received under sync point:

- If you set the property to `Automatic`, the message is received under sync point if the incoming message is marked as persistent; otherwise, it is not received under sync point. Any message that is sent later by an output node is put under sync point, as determined by the incoming persistence property, unless the output node has overridden this property explicitly.
- If you set the property to `Yes` (the default), the message is received under sync point; that is, within a WebSphere MQ unit of work. Any messages that are sent later by an output node in the same instance of the message flow are put under sync point, unless the output node has overridden this explicitly.
- If you set the property to `No`, the message is not received under sync point. Any messages that are sent later by an output node in the message flow are not put under sync point, unless an individual output node has specified that the message must be put under sync point.

The MQOutput node is the only output node that you can configure to override this option.

If you have set the `Browse Only` property, the value that is set for the `Transaction mode` property is ignored because a message cannot be browsed under sync point. However, any derived messages that are propagated later by an output node in the same instance of the message flow follow the behavior that is described previously in accordance with the specified `Transaction mode` value.

MQGET buffer size:

The MQGET buffer size is implemented internally by the broker and you cannot change it. The following description is provided for information only. You must not rely on it when you develop your message flows, because the internal implementation might change.

When the MQInput node initializes, it sets the size of the default buffer for the first MQGET to 4 KB. The MQInput node monitors the size of messages and increases or reduces the size of the buffer:

1. If an MQGET fails because the message is larger than the size of the buffer, the node immediately increases the size of the buffer to accommodate the message, issues the MQGET again, and zeros a message count.
2. When 10 messages have been counted since the increase in the size of the buffer, the node compares the size of the largest of the 10 messages with the size of the buffer. If the size of the largest message is less than 75% of the size of the buffer, the buffer is reduced to the size of the largest of the 10 messages. If an MQGET fails during the 10 messages because the message is larger than the size of the buffer, the node takes action 1.

For example, if the first message that the node receives is 20 MB, and the next 10 messages are each 14 MB, the size of the buffer is increased from 4 KB to 20 MB and remains at 20 MB for 10 messages. However, after the 10th message the size of the buffer is reduced to 14 MB.

Terminals and properties:
When you have put an MQInput node into a message flow, you can configure the node; see . The properties of the node are displayed in the Properties view. All mandatory properties that do not have a default value defined are marked with an asterisk.

The terminals of the MQInput node are described in the following table.

Terminal	Description
Failure	The output terminal to which the message is routed if an error occurs. Even if the Validation property is set, messages propagated to this terminal are not validated.
Out	The output terminal to which the message is routed if it is successfully retrieved from the WebSphere MQ queue.
Catch	The output terminal to which the message is routed if an exception is thrown downstream and caught by this node.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Description properties of the MQInput node are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, MQInput	The name of the node.
Short description	No	No		A brief description of the node.

Property	M	C	Default	Description
Long description	No	No		Text that describes the purpose of the node in the message flow.

The Basic properties of the MQInput node are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Queue name	Yes	Yes		The name of the WebSphere MQ input queue from which this node retrieves messages (using MQGET) for processing by this message flow. You must predefine this WebSphere MQ queue to the queue manager that hosts the broker to which the message flow is deployed.	queueName

The MQInput node Input Message Parsing properties are described in the following table.

If the incoming message has an MQRFH2 header, you do not have to set values for the Input Message Parsing properties because the values are derived from the `<mcd>` folder in the MQRFH2 header; for example:

```
<mcd><Msd>MRM</Msd><Set>DHM4UO906S001</Set><Type>receiptmsg1</Type>
<Fmt>XML</Fmt></mcd>
```

If you set values, and those values differ from the values in the MQRFH2 header, and the `<Msd>` element is a valid domain, the values in the MQRFH2 header take precedence.

Property	M	C	Default	Description
Message domain	No	No	`BLOB`	The domain that is used to parse the incoming message. If no MQRFH2 header exists to supply the value for the `Message domain`, you can select the property value from the list. You can either select an option or leave the property blank, in which case the default that is used is BLOB. The following options are available: • XMLNSC • DataObject • JSON • BLOB • MIME • MRM • JMSMap • JMSStream • XMLNS You can also specify a user-defined parser, if appropriate.
Message set	No	No		If you use the MRM or IDOC parser, or the XMLNSC parser in validating mode, select the name or identifier of the message set in which the incoming message is defined. The list of message sets consists of the message sets that are available when you select MRM, XMLNSC, or IDOC as the domain. If you set this property, then later update the project dependencies to remove this message set reference, a warning is issued. Either update the `Message set` property, or restore the reference to this message set project.
Message type	No	No		If you use the MRM parser, select the type of message from the list. This list is populated with messages that are defined in the message set that you have selected.

Property	M	C	Default	Description
Message format	No	No		If you are using the MRM or IDOC parser, select the physical format of the incoming message from the list. This list includes all the physical formats that you have defined for this message set. If you set the `Message domain` property to `DataObject`, you can set this property to `XML` or `SAP ALE IDoc`. Set this property to `SAP ALE IDoc` when you have to parse a bit stream from an external source and generate a message tree.

The properties of the Parser Options for the MQInput node are described in the following table.

Property	M	C	Default	Description
Parse timing	No	No	`On Demand`	This property controls when an input message is parsed. Valid values are On Demand, Immediate, and Complete. `Parse timing` is, by default, set to `On Demand`, which causes parsing of the message to be delayed. To cause the message to be parsed immediately, see "Parsing on demand" on page 4173.
Use MQRFH2C compact parser for MQRFH2 header	No	No	Cleared	This property controls whether the MQRFH2C compact parser, instead of the MQRFH2 parser, is used for MQRFH2 headers.
Build tree using XML schema data types	No	No	Cleared	This property controls whether the XMLNSC parser creates syntax elements in the message tree with data types taken from the XML schema. You can select this property only if you set the `Validate` property on the **Validation** tab to `Content` or `Content and Value`.
Use XMLNSC compact parser for XMLNS domain	No	No	Cleared	This property controls whether the XMLNSC compact parser is used for messages in the XMLNS domain. If you set this property, the message data is displayed under XMLNSC in nodes that are connected to the output terminal when the input MQRFH2 header or the Input Message Parsing property `Message domain` is XMLNS. For more information, see "Manipulating messages in the XMLNSC domain" on page 2546.
Retain mixed content	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters mixed text in an input message. If you select the check box, elements are created for mixed text. If you clear the check box, mixed text is ignored and no elements are created.
Retain comments	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters comments in an input message. If you select the check box, elements are created for comments. If you clear the check box, comments are ignored and no elements are created.
Retain processing instructions	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters processing instructions in an input message. If you select the check box, elements are created for processing instructions. If you clear the check box, processing instructions are ignored and no elements are created.
Opaque elements	No	No	Blank	This property is used to specify a list of elements in the input message that are to be opaquely parsed by the XMLNSC parser. Opaque parsing is performed only if validation is not enabled (that is, if `Validate` is `None`); entries that are specified in `Opaque Elements` are ignored if validation is enabled.

The Advanced properties of the MQInput node are described in the following table. Set these properties to determine how the message is processed, such as its transactional characteristics. Many of these properties map to options on the MQGET call.

Property	M	C	Default	Description	**mqsiapplybaroverride command property**
Transaction mode	Yes	No	`Yes`	This property controls whether the incoming message is received under sync point. Valid values are `Automatic`, `Yes`, and `No`. • If you select `Automatic`, the incoming message is received under sync point if it is marked persistent, otherwise it is not received under sync point. The transactionality of any derived messages that are sent later by an output node is determined by the incoming persistence property, unless the output node has overridden transactionality explicitly. • If you select `Yes`, the incoming message is received under sync point. Any derived messages that are sent later by an output node in the same instance of the message flow are sent transactionally, unless the output node has overridden transactionality explicitly. • If you select `No`, the incoming message is not received under sync point. Any derived messages that are sent later by an output node in the message flow are sent non-transactionally, unless the output node has specified that the messages must be put under sync point.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Order mode	Yes	No	`Default`	The order in which messages are retrieved from the input queue and processed. Messages arriving can be processed in order, or any thread can process any message when it is ready. When ordering is imposed, a thread processes a message only if it is the first unprocessed message available with a unique ordering value.Valid values are `Default`, `By User ID`, `By Queue Order`, and `User Defined`. This property has an effect only if the message flow property `Additional instances` on the **Instances** tab, is set to greater than zero; that is, if multiple threads read the input queue. Valid values are: • `Default`. Messages are retrieved in the order that is defined by the queue attributes, but this order is not guaranteed because the messages are processed by the message flow. • `By User ID`. Messages that have the same UserIdentifier value in the MQMD are retrieved and processed in the order that is defined by the queue attributes; this order is guaranteed to be preserved when the messages are processed. A message that is associated with a particular user identifier that is being processed by one thread, is completely processed before the same thread, or another thread, can start to process another message with the same user identifier. Ensure that each message has a unique message ID in the MQMD of the incoming message. No other ordering is guaranteed to be preserved. • `By Queue Order`. Messages are retrieved and processed by this node in the order that is defined by the queue attributes; this order is guaranteed to be preserved when the messages are processed. This behavior is identical to the behavior that is exhibited if the message flow property `Additional instances` is set to zero. However, if you set `Order mode` to `By Queue Order` then redeploy the message flow, additional instances that are already running are not released. Therefore, when you set `Order mode` to `By Queue Order`, either stop and restart the message flow, or run the **mqsireload** command for the execution group after you redeploy the flow. • ⌂`User Defined`. You can specify a message element using the `Order field location` property. For more details about using this option, see “Optimizing message flow throughput” on page 587 and “Receiving messages in a WebSphere MQ message group” on page 1554.	
Order field location	N	N	""	An XPath or ESQL expression property to control which part of the message is used to impose ordering on incoming messages when `Order mode` is `User Defined`. If the field is missing, an exception is raised, and the message is rolled back. NULL and empty values are processed separately, in parallel.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Logical order	Yes	No	Selected	If you select this check box, messages are received in logical order, as defined by WebSphere MQ. This option maps to the MQGMO_LOGICAL_ORDER option of the MQGMO of the MQI. If you clear the check box, messages that are sent as part of a group are not received in a predetermined order. If a broker expects to receive messages in groups, and you have not selected this check box, either the order of the input messages is not significant, or you must design the message flow to process them appropriately. You must also select `Commit by message group` if you want message processing to be committed only after the final message of a group has been received and processed. More information about the options to which this property maps is available in the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online. For more details about using this option, see "Receiving messages in a WebSphere MQ message group" on page 1554.	
All messages available	Yes	No	Cleared	Select `All messages available` if you want message retrieval and processing to be done only when all messages in a single group are available. This property maps to the MQGMO_ALL_MSGS_AVAILABLE option of the MQGMO of the MQI. Clear this check box if message retrieval does not depend on all of the messages in a group being available before processing starts. More information about the options to which this property maps is available in the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online.	
Match message ID	No	No		A message ID that must match the message ID in the MQMD of the incoming message. Enter a message identifier if you want the input node to receive only messages that contain a matching message identifier value in the MsgId field of the MQMD. Enter an even number of hexadecimal digits (characters `0 to 9`, `A to F`, and `a to f` are valid) up to a maximum of 48 digits. If the matching message identifier that you enter is shorter than the size of the MsgId field, `Match message ID` is padded on the right with `X'00'` characters. This property maps to the MQMO_MATCH_MSG_ID option of the MQGMO of the MQI. Leave this property blank if you do not want the input node to check that the message ID matches. More information about the options to which this property maps is available in the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Match correlation ID	No	No		A correlation ID that must match the correlation ID in the MQMD of the incoming message. Enter a message identifier if you want the input node to receive only messages that contain a matching correlation identifier value in the CorrelId field of the MQMD. Enter an even number of hexadecimal digits (characters `0 to 9`, `A to F`, and `a to f` are valid) up to a maximum of 48 digits. If the ID that you enter is shorter than the size of the CorrelId field, it is padded on the right with `X'00'` characters. This property maps to the MQMO_MATCH_CORREL_ID option of the MQGMO of the MQI. Leave this property blank if you do not want the input node to check that the CorrelID matches. More information about the options to which this property maps is available in the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Convert	Yes	No	Cleared	If you select this check box, WebSphere MQ converts data in the message to be received, in conformance with the CodedCharSetId and Encoding values set in the MQMD. Select `Convert` if you want WebSphere MQ to perform data conversion on the message when it is retrieved from the queue. This option is useful if you are handling messages in the BLOB domain, or are using a user-defined parser. Do not select this option if you are parsing messages with the XML or MRM domains, because the parser does the conversion. WebSphere MQ converts the incoming message to the encoding and coded character set that is specified in the MQMD that the input node supplies on the MQGET call to retrieve the message from the input queue. The broker uses the MQGMO_CONVERT option on the MQGET call; typical rules for WebSphere MQ conversion apply. The values that you specify in the `Convert encoding` and `Convert coded character set ID` options are used in the **MsgDesc Encoding** and **CCSID** fields in the MQGET call. WebSphere MQ can convert the incoming message only if the MQMD **Format** field is a built-in WebSphere MQ value that identifies character data, or if a data conversion exit exists in WebSphere MQ. This property maps to the MQGMO_CONVERT option of the MQGMO of the MQI. Clear the check box if you do not want WebSphere MQ to convert the message. If you select this check box, you can also specify values for the `Convert encoding` and `Convert coded character set ID` properties. For more information about WebSphere MQ data conversion, and why you might choose to use this option, see the *Application Programming Guide* section of the WebSphere MQ Version 7 Information Center online.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Convert encoding	No	No		The representation used for numeric values in the message data, expressed as an integer value. This property is valid only if you have selected `Convert`. Enter the number that represents the encoding to which you want to convert numeric data in the message body. Valid values include: • Linux Windows `546` for DOS, all Windows systems, and Linux on x86 • Linux UNIX `273` for Linux on Linux on POWER, Linux on IBM z Systems, and all UNIX systems • z/OS `785` for z/OS messages that use Binary Packed Decimals, and 273 for messages that use IEEE floating point numbers The encoding is used only in the following circumstances: • If a user-defined WebSphere MQ data conversion exit uses the encoding. • If the built-in WebSphere MQ conversion exit uses the encoding to convert messages in any of the predefined WebSphere MQ formats. If you specify an incorrect value, no conversion is done. For further information about the values that you can specify for `Convert encoding`, see "Data conversion" on page 1151 and the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online.	
Convert coded character set ID	No	No		The coded character set identifier of character data in the message data, expressed as an integer value. This property is valid only if you have selected `Convert`. Enter the value that represents the character set identifier to which you want to convert character data in the message body. If you specify an incorrect value, no conversion is done. For further information about the values that you can specify for `Convert coded character set ID`, see the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Commit by message group	Yes	No	Cleared	This property controls when a transaction is committed when processing messages that are part of a message group. If you select the check box, the transaction is committed when the message group has been processed. If you leave this check box cleared, a commit is performed after each message has been propagated completely through the message flow. This property is relevant only if you have selected `Logical order`. Set the `Order mode` property to `By Queue Order` if the messages in a group must be retrieved and processed in the order in which they are displayed on the queue.	
z/OS serialization token	No	No		On z/OS only: A user-defined token for serialized application support. The value that is specified must conform to the rules for a valid ConnTag in the WebSphere MQ MQCNO structure. Enter a serialization token if you want to use the serialized access to shared resources that is provided by WebSphere MQ. The value that you provide for the serialization token must conform to the rules described in the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online. For more information about serialization and queue sharing on z/OS, see the *Concepts and Planning Guide* section of the WebSphere MQ Version 7 Information Center online.	serializationToken
Topic	No	Yes		The default topic for the input message. You can associate a message with a publish/subscribe topic by using this property. You can enter any characters as the topic name. When messages pass through the MQInput node, they take on whatever topic name you have entered. (If you are using publish/subscribe, you can subscribe to a topic and see any messages that passed through the MQInput node and were published under that topic name.) If the incoming message has an MQRFH2 header, you do not have to set a value for the `Topic` property because the value can be obtained from the `<psc>` folder in the MQRFH2 header; for example: `<psc><Topic>stockquote</Topic></psc>` If you set a `Topic` property value, and that value differs from the <Topic> value in the MQRFH2 header, the value in the MQRFH2 header takes precedence.	topicProperty
Browse Only	No	No	Cleared	This property controls whether a message is removed from the queue when it is read. If you select this check box, the message is not removed from the queue when it is read. If you select this option, `OutputLocalEnvironment.MQ.GET.Browsed` is set to true when a message is propagated to the output terminal of the MQInput node.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Reset browse timeout (ms)	Yes	Yes	-1	The time, in milliseconds, between the last eligible message being browsed on the input queue and the browse being reset to the beginning of the queue. The default value of -1 indicates that the browse is not reset.	

The Validation properties of the MQInput node are described in the following table. Set these properties if you want the parser to validate the body of messages against the Message set. (If a message is propagated to the Failure terminal of the node, it is not validated.)

For more details, see "Validating messages" on page 1478 and "Validation properties" on page 4169.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Validate	No	Yes	None	This property controls whether validation takes place. Valid values are None, Content, and Content and Value.	validateMaster
Failure action	No	No	Exception	This property controls what happens if validation fails. You can set this property only if you set Validate to Content or Content and Value. Valid values are User Trace, Local Error Log, Exception, and Exception List.	

The Security properties of the MQInput node are described in the following table. Set values for these properties to control the extraction of an identity from a message (when a security profile is associated with the node). For more information about these properties, see "Identity" on page 390, "Configuring the extraction of an identity or security token" on page 447, "Message flow security overview" on page 383, and "Setting up message flow security" on page 431.

Property	M	C	Default	Description
Identity token type	No	No	None	This property specifies the type of identity token present in the incoming message. Valid values are: • Transport Default • Username • Username + Password • SAML Assertion • X.509 Certificate If this property is not specified, the identity is retrieved from the transport headers and the type is set to Username.
Identity token location	No	No	None	This property specifies where, in the message, the identity can be found. The location is specified as an ESQL field reference, an XPath expression, or a string literal. If you use a string literal, it must be enclosed in single quotation marks and must not contain a period (.), If this property is not specified, the identity is retrieved from the MQMD.UserIdentifier transport header.
Identity password location	No	No	None	This property specifies where, in the message, the password can be found. The location is specified as an ESQL field reference, an XPath expression, or a string literal. If you use a string literal, it must be enclosed in single quotation marks and must not contain a period (.), If it is not specified, the password is not set. This property can be set only if Identity token type is set to Username + Password.

Property	M	C	Default	Description
Identity IssuedBy location	No	No	`None`	This property specifies a string or path expression that describes the issuer of the identity. The location is specified as an ESQL field reference, an XPath expression, or a string literal. If you use a string literal, it must be enclosed in single quotation marks and must not contain a period (.), The value specifies the Issuer that is passed to a WS-Trust v1.3 STS provider. If this property is not specified, the MQMD.PutApplName value is used. If you leave the `Identity issuedBy location` field blank and the MQMD.PutApplName is also blank, the string `MQ` is used.
Treat security exceptions as normal exceptions	No	No	`False`	This property specifies whether to treat security exceptions (such as "Access Denied") as normal exceptions, and propagate them to the Failure terminal (if wired). This property is turned off by default, which ensures that security exceptions cause the message to be backed out even if the Failure terminal is wired.

The Instances properties of the MQInput node are described in the following table. Set values for these properties to control the additional instances that are available for a node. For a full description of these properties, see "Configurable message flow properties" on page 4020.

Property	M	C	Default	Description	**`mqsiapplybaroverride` command property**
Additional instances pool	No	Yes	`Use Pool Associated with Message Flow`	The pool from which additional instances are obtained. • If you select `Use Pool Associated with Message Flow`, additional instances are obtained from the message flow value. • If you select `Use Pool Associated with Node`, additional instances are allocated from the additional instances of the node, based on the number specified in the `Additional instances` property.	componentLevel
Additional instances	No	Yes	`0`	The number of additional instances that the node can start if the `Additional instances pool` property is set to `Use Pool Associated with Node`. By default, no additional instances are given to the node.	additionalInstances

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

"Parsers" on page 1072
A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from the internal message tree representation.

"Message flow security overview" on page 383
WebSphere Message Broker provides a security manager, which enables you to control access to individual messages in a message flow, using the identity of the message.

"Identity" on page 390
In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

"Specifying namespaces in the Message Type property" on page 1208
When using the MRM domain, the *Message Type* property is used to specify the name of the message.

"Multipart messages" on page 1191
A *multipart message* contains one or more other messages within its structure. The contained message is sometimes referred to as an *embedded message*.

Related tasks:

"Using more than one input node" on page 1473
You can include more than one input node in a single message flow.

"Configuring transactionality for message flows" on page 1290
A message flow runs in a single transaction, which is started when data is received by an input node, and can be committed or rolled back when all processing has completed.

"Validating messages" on page 1478
The broker provides validation based on the message model for predefined messages.

"Manipulating messages in the XMLNSC domain" on page 2546
If you are writing ESQL to process messages in the XMLNSC domain, it is helpful to learn about the structure of the message tree that the XMLNSC parser builds.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Using WebSphere MQ cluster queues for input and output" on page 1544
Design your broker network to use WebSphere MQ queues, if appropriate for your business needs.

"Using WebSphere MQ shared queues for input and output (z/OS)" on page 1546
On z/OS systems, you can define WebSphere MQ shared queues as input and output queues for message flows. You might need to serialize access to those messages.

"Receiving messages in a WebSphere MQ message group" on page 1554
You can configure the MQInput node to receive messages that are in a WebSphere MQ message group.

"Setting up message flow security" on page 431
Set up security on a message flow to control access based on the identity of a message passing through the message flow.

"Configuring the extraction of an identity or security token" on page 447
You can configure the SecurityPEP node or security enabled input nodes to extract the identity or security token from a message and store it in the properties tree identity fields, enabling it to be processed throughout the message flow and propagated at output or request nodes.

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

Related reference:

"WebSphere MQ Enterprise Transport" on page 1542
WebSphere MQ Enterprise Transport is a service that connects applications to messaging middleware.

"Configurable message flow properties" on page 4020
When you add a message flow to a broker archive (BAR) file in preparation for deploying it to a broker, you can set additional properties that influence its run time operation. These properties are available for review and update when you select the **Manage and Configure** tab for the broker archive file.

"Input node" on page 4511
Use the Input node as an In terminal for an embedded message flow (a subflow).

"Validation properties" on page 4169
You can control validation by setting properties on the Validate and Parser Options tabs for the nodes that are listed in the following table.

"MQOutput node" on page 4612
Use the MQOutput node to send messages to clients that connect to the broker using the WebSphere MQ Enterprise Transport and that use the MQI and AMI application programming interfaces.

"MQGet node" on page 4578
Use the MQGet node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and the MQI and AMI application programming interfaces.

"MQReply node" on page 4621
Use the MQReply node to send a response to the originator of the input message.

"SecurityPEP node" on page 4729
Use the SecurityPEP node in a message flow to invoke the message flow security manager at any point in the message flow.

Related information:

WebSphere MQ Version 7 Information Center online

MQJMSTransform node

Use the MQJMSTransform node to receive messages that have a WebSphere MQ JMS provider message tree format, and transform them into a format that is compatible with messages that are to be sent to JMS destinations.

This topic contains the following sections:
- "Purpose"
- "Using the MQJMSTransform node in a message flow" on page 4611
- "Terminals and properties" on page 4611

Purpose:
Use the MQJMSTransform node to send messages to existing message flows and to interoperate with WebSphere MQ JMS and WebSphere Message Broker publish/subscribe.

The JMSMQTransform node handles messages in all supported message domains.

The MQJMSTransform node is contained in the **JMS** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using the MQJMSTransform node in a message flow:
The following sample contains a message flow in which the MQJMSTransform node is used. Look at this sample for an example of how to use the MQJMSTransform node.

- JMS Nodes

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Terminals and properties:
When you have put an instance of the MQJMSTransform node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view.

The terminals of the MQJMSTransform node are described in the following table.

Terminal	Description
Failure	The output terminal to which the message is routed if an error occurs. Even if the Validation property is set, messages that are propagated to this terminal are not validated.
Out	The output terminal to which the message is routed if it is successfully retrieved from the WebSphere MQ queue.
In	The input terminal that accepts a message for processing by the node.

The following table describes the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The MQJMSTransform node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, MQJMSTransform	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related reference:

"JMSInput node" on page 4532
Use the JMSInput node to receive messages from JMS destinations. JMS destinations are accessed through a connection to a JMS provider.

"JMSOutput node" on page 4549
Use the JMSOutput node to send messages to JMS destinations.

"JMSMQTransform node" on page 4547
Use the JMSMQTransform node to transform a message with a JMS message tree into a message that has a message tree structure that is compatible with the format of messages that are produced by the WebSphere MQ JMS provider.

"WebSphere Broker JMS Transport" on page 1681
The WebSphere Broker JMS Transport is a service that connects applications that send and receive messages that conform to the Java Message Service (JMS) standard.

MQOptimizedFlow node

Use the MQOptimizedFlow node to provide a high-performance publish/subscribe message flow. The node supports publishers and subscribers that use Java Message Service (JMS) application programming interfaces and the WebSphere MQ Enterprise Transport.

Restriction: z/OS You cannot use an MQOptimizedFlow node in message flows that you deploy to z/OS systems.

The MQOptimizedFlow node is deprecated. The node continues to work but is no longer required.

MQOutput node

Use the MQOutput node to send messages to clients that connect to the broker using the WebSphere MQ Enterprise Transport and that use the MQI and AMI application programming interfaces.

This topic contains the following sections:

Purpose:

The MQOutput node delivers an output message from a message flow to a WebSphere MQ queue. The node uses MQPUT to put the message to the destination queue or queues that you specify.

If appropriate, define the queue as a WebSphere MQ clustered queue or shared queue. When you use a WebSphere MQ clustered queue, leave the Queue Manager Name empty.

You can configure the MQOutput node to put a message to a specific WebSphere MQ queue that is defined on any queue manager that is accessible by the queue manager for the broker, or to the destinations identified in the local environment that is associated with the message.

Set other properties to control the way in which messages are sent, by causing appropriate MQPUT options to be set; for example, you can request that a message is processed under transaction control. You can also specify that WebSphere MQ can, if appropriate, break the message into segments in the queue manager.

If you create a message flow to use as a subflow, you cannot use a standard output node; use an instance of the Output node to create an Out terminal for the subflow through which to propagate the message.

If you do not want your message flow to send messages to a WebSphere MQ queue, choose another supported output node.

The MQOutput node checks for the presence of an MQMD (WebSphere MQ message descriptor) header in the message tree. If no MQMD header is present, the MQOutput node creates one in the message tree, and populates it with MQMD default properties. If an MQMD header is found, the MQOutput node checks that it is an WebSphere MQ type header; if it is not, the `Message Context` property is set to `Default`. The MQOutput node treats any other transport headers in the message tree as data. If such headers are not required as part of the message body, use a transformation node to remove them from the message tree before the MQOutput node. If the message tree is sourced from JMS, you can use a JMSMQTransform node to construct a message tree compatible with MQ. For further details, see “JMS message transformation” on page 1684.

The MQOutput node is contained in the **WebSphere MQ** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:
Look at the following samples to see how to use this node:
- Pager
- Airline Reservations
- Error Handler
- Aggregation
- Large Messaging
- Message Routing
- Timeout Processing
- Video Rental
- XSL Transform

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

For an example of how to use this node, assume that you have written a publishing application that publishes stock updates on a regular basis. The application sends the messages to the broker on an MQInput node, and the message flow makes the publications available to multiple subscribers through a Publication node. You configure a Compute node to create a new output message whenever one particular stock is changed, and connect this node to an MQOutput node to record each price change for this stock.

Working with WrittenDestination data

After the message has been put, the WrittenDestination folder in the local environment is updated with the destination information. WrittenDestination data for an MQOutput node has the following format:

```
WrittenDestination = (
   MQ  = (
      z`DestinationData = (
         queueName        = 'OUT'
         queueManagerName = 'MYQUEUEMANAGER'
         replyIdentifier = X'4d...2e'
         msgId           = X'3c...2c'
         correlId        = X'2a...00'
         GroupId         = X'3a...00'
      )
   )
)
```

Connecting the terminals:
Connect the In terminal to the node from which outbound messages bound are routed.

Connect the Out or Failure terminal of this node to another node in this message flow to process the message further, process errors, or send the message to an additional destination.

If you use aggregation in your message flows, you must use the output terminals.

Contents of the WebSphere MQ reply message:
The:

- Values of the following fields in MQMD are set, irrespective of the settings you make:

  ```
  MQMD.Report = 0;
  MQMD.PutApplType = MQAT_BROKER;
  MQMD.PutDate = Taken from current Timestamp
  MQMD.PutTime = Taken from current Timestamp
  MQMD.PutApplName = MsgTree.MQMD.ReplyToQMgr (first 28 chars)
  ```

- Values of the following fields are set from the values in the `Root.MQMD` folder:

  ```
  MQMD.Version
  MQMD.Format
  MQMD.Priority
  MQMD.Persistence
  MQMD.Expiry
  MQMD.Encoding
  MQMD.CodedCharSetId
  MQMD.GroupId
  ```

```
MQMD.MsgSeqNumber
MQMD.Offset
MQMD.MsgFlags
MQMD.OriginalLength
```

- Following values in MQMD are set conditionally, based on values in the MQOutput node and the `Root.MQMD` folder:

```
IF MsgTree.MQMD.MsgType = MQMT_REQUEST THEN
  MQMD.MsgType = MQMT_REPLY;
IF Nodes Message Context is Default, PassAll or PassIdentity THEN
  MQMD.UserIdentifer = MsgTree.MQMD.UserIdentifier;
IF MsgTree.MQMD.Report contains MQRO_PASS_CORREL_ID THEN
  MQMD.CorrelId = MsgTree.MQMD.CorrelId;
ELSE
  MQMD.CorrelId = MsgTree.MQMD.MsgId;
IF MsgTree.MQMD.Report contains MQRO_PASS_MSG_ID THEN
  MQMD.MsgId = MsgTree.MQMD.MsgId;
ELSE
  MQMD.MsgId = MQMI_NONE;
```

- Value of the `MQMD.Persistence` field is set based on the `Persistence` mode on the MQOutput node.

When the output MQMD structure has been constructed, the `Message Context` on the MQOutput node is ignored, and the behavior is as `set All`.

The values that are overridden, are done only in the output MQMD structure; no updates are made to the MQMD folder in the message tree.

Configuring for coordinated transactions:
When you define an MQOutput node, the option that you select for the `Transaction Mode` property defines whether the message is written under sync point:

- If you select `Yes`, the message is written under sync point (that is, within a WebSphere MQ unit of work).
- If you select `Automatic` (the default), the message is written under sync point if the incoming input message is marked as persistent.
- If you select `No`, the message is not written under sync point.

Another property of the MQOutput node, `Persistence Mode`, defines whether the output message is marked as persistent when it is put to the output queue:

- If you select `Yes`, the message is marked as persistent.
- If you select `Automatic` (the default), the message persistence is determined from the properties of the incoming message, as set in the MQMD.
- If you select `No`, the message is not marked as persistent.
- If you select `As Defined for Queue`, the message persistence is set as defined in the WebSphere MQ queue by the MQOutput node specifying the MQPER_PERSISTENCE_AS_Q_DEF option in the MQMD.

Terminals and properties:
When you have put an instance of the MQOutput node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view.

The MQOutput node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.

Terminal	Description
Failure	The output terminal to which the message is routed if a failure is detected when the message is put to the output queue.
Out	The output terminal to which the message is routed if it has been successfully put to the output queue.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file for deployment).

The MQOutput node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, MQOutput	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The MQOutput node Basic properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Queue Manager Name	No	Yes		Enter the name of the WebSphere MQ queue manager to which this output queue (which is specified by the `Queue Name` property) is defined. The `Queue Manager Name` property is only needed if the queue specified is defined on another queue manager or is a cluster queue. If it is a cluster queue, then `Queue Manager Name` can optionally be specified if you want to be prescriptive about which queue manager receives the message. If the queue is defined on a remote queue manager, the `Queue Manager Name` must be specified and it must match the transmission queue defined on the broker's local queue manager, for that remote queue manager.	queueManagerName
Queue Name	No	Yes		To send the output message to a single destination queue that is defined by this node, enter the name of the WebSphere MQ output queue to which the message flow sends messages. If you set the `Destination Mode` property to `Queue Name`, you must specify a value for the `Queue Name` property. If you set `Destination Mode` to another value, this property is ignored.	queueName

The MQOutput node Advanced properties are described in the following table. These properties define the transactional control for the message and the way in which the message is put to the queue. Many of these properties map to options on the MQPUT call.

Property	M	C	Default	Description
Destination Mode	Yes	No	`Queue Name`	The queues to which the output message is sent. • If you select `Queue Name` (the default), the message is sent to the queue that is named in the `Queue Name` property. If you select this option, you must set the `Queue Manager Name` and `Queue Name` properties. • If you select `Reply To Queue`, the message is sent to the queue that is named in the ReplyToQ field in the MQMD. When you select this value, the MQOutput node constructs a WebSphere MQ reply message. For more information about the settings that are used by the MQOutput node and the `Root.MQMD` folder in this situation, see "Contents of the WebSphere MQ reply message" on page 4614. • If you select `Destination List`, the message is sent to the list of queues that are named in the local environment that is associated with the message. The data that you have provided is used in the DestinationData subtree of the local environment. For more information about the DestinationData subtree, see "Data types for elements in the MQ DestinationData subtree" on page 4240. For more information about destination lists, see "Creating destination lists" on page 1477.
Transaction Mode	Yes	No	`Automatic`	This property controls whether the message is put transactionally. • If you select `Automatic` (the default), the message transactionality is derived from the way that it was specified at the input node. • If you select `Yes`, the message is put transactionally. • If you select `No`, the message is put non-transactionally. For more information, see "Configuring for coordinated transactions" on page 4615.
Persistence Mode	Yes	No	`Automatic`	This property controls whether the message is put persistently. • If you select `Automatic` (the default), the persistence is as specified in the incoming message. • If you select `Yes`, the message is put persistently. • If you select `No`, the message is put non-persistently. • If you select `As Defined for Queue`, the message persistence is set as defined for the WebSphere MQ queue.
New Message ID	Yes	No	Cleared	If you select this check box, WebSphere MQ generates a new message identifier to replace the contents of the **MsgId** field in the MQMD. This property maps to the MQPMO_NEW_MSG_ID option of the MQPMO of the MQI. Clear the check box if you do not want to generate a new ID. A new message ID is still generated if you select the `Request` property on the **Request** tab. For more information about the options to which this property maps, see the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online.
New Correlation ID	Yes	No	Cleared	If you select this check box, WebSphere MQ generates a new correlation identifier to replace the contents of the CorrelId field in the MQMD. This property maps to the MQPMO_NEW_CORREL_ID option of the MQPMO of the MQI. Clear the check box if you do not want to generate a new ID. For more information about the options to which this property maps, see the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online.

Property	M	C	Default	Description
Segmentation Allowed	Yes	No	Cleared	If you select this check box, WebSphere MQ breaks the message into segments in the queue manager. Clear the check box if you do not want segmentation to occur. For more information about message segmentation, see "Sending message segments in a WebSphere MQ message" on page 1557. For more information about the options to which this property maps, see the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online.
Message Context	Yes	No	`Pass All`	This property controls how origin context is handled. • `Pass All` maps to the MQPMO_PASS_ALL_CONTEXT option of the MQPMO of the MQI. • `Pass Identity` maps to the MQPMO_PASS_IDENTITY_CONTEXT option of the MQPMO of the MQI. • `Set All` maps to the MQPMO_SET_ALL_CONTEXT option of the MQPMO of the MQI. • `Set Identity` maps to the MQPMO_SET_IDENTITY_CONTEXT option of the MQPMO of the MQI. • `Default` maps to the MQPMO_DEFAULT_CONTEXT option of the MQPMO of the MQI. • `None` maps to the MQPMO_NO_CONTEXT option of the MQPMO of the MQI. When a security profile is associated with the node and is configured to perform identity propagation, the chosen context can be overridden to ensure that the outgoing identity is set. For more information about the options to which these properties map, see the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online.
Alternate User Authority	Yes	No	Cleared	If you select this check box, alternate authority is used when the output message is put and the MQOO_ALTERNATE_USER_AUTHORITY option is set in the open options (MQOO) of the MQI. If you select this check box, this option is specified when the queue is opened for output. The alternative user information is retrieved from the context information in the message. Clear the check box if you do not want to specify alternative user authority. If you clear the check box, the broker service user ID is used when the message is put.

The MQOutput node Request properties are described in the following table. These properties define the characteristics of each output message that is generated.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Request	Yes	No	Cleared	If you select the check box, each output message in the MQMD is generated as a request message (MQMT_REQUEST), and the message identifier field is cleared (and set to MQMI_NONE) so that WebSphere MQ generates a new identifier. Clear the check box to indicate that each output message is not marked as a request message. If you have set `Destination Mode` to `Reply To Queue`, you cannot select this check box. A new message identifier is generated even if the `New Message ID` check box is not selected on the **Advanced** tab.	
Reply-to Queue Manager	No	Yes		The name of the WebSphere MQ queue manager to which the output queue, which is specified in `Reply-to Queue`, is defined. This name is inserted into the MQMD of each output message as the reply-to queue manager.	replyToQMgr
Reply-to Queue	No	Yes		The name of the WebSphere MQ queue to which to put a reply to this request. This name is inserted into the MQMD of each output message as the reply-to queue.	replyToQ

The Validation properties of the MQOutput node are described in the following table.

For a full description of these properties, see "Validation properties" on page 4169.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Validate	No	Yes	`Inherit`	This property controls whether validation takes place. Valid values are `None`, `Content and Value`, `Content`, and `Inherit`.	validateMaster
Failure Action	No	No	`Exception`	This property controls what happens if validation fails. You can set this property only if you set `Validate` to `Content` or `Content and Value`. Valid values are `User Trace`, `Local Error Log`, `Exception`, and `Exception List`.	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Local environment tree structure" on page 1056
The local environment tree is a part of the logical message tree in which you can store information while the message flow processes the message.

Related tasks:

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

"Configuring transactionality for message flows" on page 1290
A message flow runs in a single transaction, which is started when data is received by an input node, and can be committed or rolled back when all processing has completed.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Validating messages" on page 1478
The broker provides validation based on the message model for predefined messages.

"Using WebSphere MQ cluster queues for input and output" on page 1544
Design your broker network to use WebSphere MQ queues, if appropriate for your business needs.

"Using WebSphere MQ shared queues for input and output (z/OS)" on page 1546
On z/OS systems, you can define WebSphere MQ shared queues as input and output queues for message flows. You might need to serialize access to those messages.

"Accessing the local environment tree" on page 2463
The local environment tree has its own correlation name, LocalEnvironment, and you must use this name in all ESQL statements that refer to or set the content of this tree.

"Configuring aggregation flows" on page 2721
Use aggregation message flows to generate and fan-out a number of related requests, fan-in the corresponding replies, and compile those replies into a single aggregated reply message, by using the AggregateControl, AggregateRequest, and AggregateReply nodes.

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

Related reference:

"Validation properties" on page 4169
You can control validation by setting properties on the Validate and Parser Options tabs for the nodes that are listed in the following table.

"MQGet node" on page 4578
Use the MQGet node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and the MQI and AMI application programming interfaces.

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

"MQReply node" on page 4621
Use the MQReply node to send a response to the originator of the input message.

"Output node" on page 4626
Use the Output node as an out terminal for an embedded message flow (a subflow).

Related information:

WebSphere MQ Version 7 Information Center online

MQReply node

Use the MQReply node to send a response to the originator of the input message.

This topic contains the following sections:

Purpose:

The MQReply node is a specialized form of the MQOutput node that puts the output message to the WebSphere MQ queue that is identified by the ReplyToQ field of the input message header. If appropriate, you can define the queue as a WebSphere MQ clustered queue or shared queue.

The MQReply node uses the options that are set in the Report field in the MQMD. By default (if no options are set), the MQReply node generates a new MsgId field in the reply message, and copies the message ID from the input message to the CorrelId field in the reply message. If the receiving application expects other values in these fields, ensure that the application that puts the message to the message flow input queue sets the required report options, or that you set the appropriate options in the MQMD during message processing in the message flow; for example, use a Compute node to set the Report options in the message.

More information about the Report field is available in the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online.

The MQReply node is contained in the **WebSphere MQ** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:

Look at the following sample to see how to use this node:

- Airline Reservations

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

You can use this node when you receive an order from a customer. When the order message is processed, a response is sent to the customer acknowledging receipt of the order and providing a possible date for delivery.

Working with data in the WrittenDestination folder

After the message has been put to the reply queue, the WrittenDestination folder in the local environment tree is updated with the destination information. A WrittenDestination folder for an MQOutput node has the following format:

```
WrittenDestination = (
   MQ  = (
      DestinationData = (
         queueName        = 'OUT'
         queueManagerName = 'MYQUEUEMANAGER'
         replyIdentifier = X'4d...2e'
         msgId           = X'3c...2c'
         correlId        = X'2a...00'
         GroupId         = X'3a...00'
      )
   )
)
```

Configuring the MQReply node:
When you have put an instance of the MQReply node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view.

Configure the MQReply node as follows:

1. Optional: On the **Description** tab, enter a short description, a long description, or both. You can also rename the node on this tab.
2. On the **Advanced** tab:
   a. Select `Segmentation Allowed` if you want WebSphere MQ to break the message into segments in the queue manager, when appropriate. You must also set `MQMF_SEGMENTATION_ALLOWED` in the MsgFlags field in the MQMD for segmentation to occur.

      More information about the options to which this property maps is available in the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online.
   b. Choose the persistence mode that you want for the output message.
      - If you select Automatic (the default), the persistence is as specified in the incoming message.
      - If you select Yes, the message is put persistently.
      - If you select No, the message is put non-persistently.
      - If you select As Defined for Queue, the message persistence is set as defined in the WebSphere MQ queue.
   c. Choose the transaction mode that you want for the output message.
      - If you select Automatic (the default), the message transactionality is derived from how it was specified at the MQInput node.
      - If you select Yes, the message is put transactionally.
      - If you select No, the message is put non-transactionally.
3. On the **Validation** tab, set the validation properties; see "Validation properties" on page 4169. If a message is propagated to the Failure terminal of the node, it is not validated.

   For more details, see "Validating messages" on page 1478.

The reply message is put (using MQPUT) to the queue named in the input message MQMD as the ReplyTo queue. You cannot change this destination.

Connecting the output terminals to another node:
Connect the Out or Failure terminal of this node to another node in the message flow to process the message further, process errors, or send the message to an additional destination.

If you use aggregation in your message flows, you must connect these output terminals.

Configuring for coordinated transactions:
When you define an MQReply node, the option that you select for the `Transaction Mode` property defines whether the message is written under sync point:

- If you select Yes, the message is written under sync point (that is, in a WebSphere MQ unit of work).
- If you select Automatic (the default), the message is written under sync point if the incoming input message is marked as persistent.
- If you select No, the message is not written under sync point.

Another property of the MQReply node, `Persistence Mode`, defines whether the output message is marked as persistent when it is put to the output queue:

- If you select Yes, the message is marked as persistent.
- If you select Automatic (the default), the message persistence is determined by the properties of the incoming message, as set in the MQMD (the WebSphere MQ message descriptor).
- If you select No, the message is not marked as persistent.
- If you select As Defined for Queue, the message persistence is set as defined in the WebSphere MQ queue; the MQReply node specifies the MQPER_PERSISTENCE_AS_Q_DEF option in the MQMD.

Terminals and properties:
The MQReply node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the message is routed if a failure is detected when the message is put to the output queue.
Out	The output terminal to which the message is routed if it has been successfully put to the output queue, and if further processing is required in this message flow.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory*; the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file for deployment).

The MQReply node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type.	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The MQReply node Advanced properties are described in the following table.

Property	M	C	Default	Description
Segmentation Allowed	Yes	No	Cleared	If you select this check box, WebSphere MQ breaks the message into segments in the queue manager.
Persistence Mode	Yes	No	Automatic	This property controls whether the message is put persistently. Valid values are Automatic, Yes, No, and As Defined for Queue.
Transaction Mode	Yes	No	Automatic	This property controls whether the message is put transactionally. Valid values are Automatic, Yes, and No.

The Validation properties of the MQReply node are described in the following table.

For a full description of these properties, see "Validation properties" on page 4169.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Validate	No	Yes	Inherit	This property controls whether validation takes place. Valid values are None, Content and Value, Content, and Inherit.	validateMaster
Failure Action	No	No	Exception	This property controls what happens if validation fails. You can set this property only if you set `Validate` to Content or Content and Value. Valid values are User Trace, Local Error Log, Exception, and Exception List.	

The MQReply node also has the following properties that you cannot access or modify through the WebSphere Message Broker Toolkit interface. However, these values are used by the broker when the message is processed in the message flow.

Property	Description
Queue Manager Name	The name of the WebSphere MQ queue manager to which the output queue, identified in `Queue Name`, is defined. This name is retrieved from the ReplyTo field of the MQMD header of the input message.
Queue Name	The name of the WebSphere MQ queue to which the output message is put. This name is retrieved from the ReplyTo field of the MQMD header of the input message.
Destination	This property always has the value reply.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an

input message is received.

"Local environment tree structure" on page 1056
The local environment tree is a part of the logical message tree in which you can store information while the message flow processes the message.

"Packaging and deployment overview" on page 3210
Deployment is the process of transferring data to an execution group on a broker so that it can take effect in the broker. Message flows and associated resources are packaged in broker archive (BAR) files for deployment.

Related tasks:

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

"Configuring transactionality for message flows" on page 1290
A message flow runs in a single transaction, which is started when data is received by an input node, and can be committed or rolled back when all processing has completed.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Validating messages" on page 1478
The broker provides validation based on the message model for predefined messages.

"Using WebSphere MQ cluster queues for input and output" on page 1544
Design your broker network to use WebSphere MQ queues, if appropriate for your business needs.

"Using WebSphere MQ shared queues for input and output (z/OS)" on page 1546
On z/OS systems, you can define WebSphere MQ shared queues as input and output queues for message flows. You might need to serialize access to those messages.

"Configuring aggregation flows" on page 2721
Use aggregation message flows to generate and fan-out a number of related requests, fan-in the corresponding replies, and compile those replies into a single aggregated reply message, by using the AggregateControl, AggregateRequest, and AggregateReply nodes.

Related reference:

"Validation properties" on page 4169
You can control validation by setting properties on the Validate and Parser Options tabs for the nodes that are listed in the following table.

"MQGet node" on page 4578
Use the MQGet node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and the MQI and AMI application programming interfaces.

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

"MQOutput node" on page 4612
Use the MQOutput node to send messages to clients that connect to the broker using the WebSphere MQ Enterprise Transport and that use the MQI and AMI application programming interfaces.

"SCADAOutput node" on page 4706
The SCADAOutput node, available in earlier versions of WebSphere Message Broker, is not supported in WebSphere Message Broker Version 7.0. See for information about migrating your message flows from WebSphere Message Broker to WebSphere Message Broker Version 7.0.

Related information:

WebSphere MQ Version 7 Information Center online

Output node

Use the Output node as an out terminal for an embedded message flow (a subflow).

This topic contains the following sections:

Purpose:

You can use a subflow for a common task that can be represented by a sequence of message flow nodes. For example, you can create a subflow to increment or decrement a loop counter, or to provide error processing that is common to a number of message flows.

You must use an Output node to provide the Out terminal to a subflow; you cannot use a standard output node (a built-in output node such as MQOutput, or a user-defined output node).

You can include one or more Output nodes in a subflow. Each one that you include provides a terminal through which you can propagate messages to subsequent nodes in the message flow in which you include the subflow.

The Output node is contained in the **Construction** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

When you select and include a subflow in a message flow, it is represented by the following icon:

When you include the subflow in a message flow, this icon exhibits a terminal for each Output node that you included in the subflow, and the name of the terminal (which you can see when you move your mouse pointer over it) matches the name of that instance of the Output node. Give your Output nodes meaningful names, you can easily recognize them when you use their corresponding terminal on the subflow node in your message flow.

Using this node in a message flow:

Look at the following sample to see how to use this node:

- Error Handler

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Terminals and properties:
When you have put an instance of the Output node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view.

The Output node terminals are described in the following table.

Terminal	Description
In	The output terminal that defines an out terminal for the subflow.

The following table describes the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Output node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, Output	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Subflows" on page 1030
You can include subflows in your message flows in the same way as you include built-in or user-defined nodes. You can also connect subflows to other nodes in the same way.

Related tasks:

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can

use in your message flows.

Related reference:

"Input node" on page 4511
Use the Input node as an In terminal for an embedded message flow (a subflow).

"MQOutput node" on page 4612
Use the MQOutput node to send messages to clients that connect to the broker using the WebSphere MQ Enterprise Transport and that use the MQI and AMI application programming interfaces.

Passthrough node

Use the Passthrough node to enable version control of a subflow at run time.

This topic contains the following sections:
- "Purpose"
- "Using this node in a message flow"
- "Terminals and properties"

Purpose:
Use the Passthrough node to add a label to your message flow or subflow. By combining this label with keyword replacement from your version control system, you can identify which version of a subflow is included in a deployed message flow. You can use this label for your own purposes. If you have included the correct version keywords in the label, you can see the value of the label:

- Stored in the broker archive (BAR) file, using the **mqsireadbar** command
- As last deployed to a particular broker, on the properties of a deployed message flow in the WebSphere Message Broker Toolkit
- At run time, if you enable user trace for that message flow

The Passthrough node does not process the message in any way. The message that it propagates on its Out terminal is the same message that it received on its In terminal.

The Passthrough node is contained in the **Construction** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:
Use this node to identify a subflow; for example, if you develop an error processing subflow to include in several message flows, you might want to modify that subflow. However, you might want to introduce the modified version initially to just a subset of the message flows in which it is included. Set a value for the instance of the Passthrough node that identifies which version of the subflow you have included.

Terminals and properties:
When you have put an instance of the Passthrough node into a message flow, you can configure it. For more information, see "Configuring a message flow node" on page 1503. The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The Passthrough node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Out	The output terminal that delivers a message to the subflow.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined), the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Passthrough node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	Passthrough	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The Passthrough node Basic properties are described in the following table.

Property	M	C	Default	Description
Label	No	No		The label (identifier) of the node. Enter a value that defines a unique characteristic; for example, the version of the subflow in which the node is included.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Subflows" on page 1030
You can include subflows in your message flows in the same way as you include built-in or user-defined nodes. You can also connect subflows to other nodes in the same way.

Related tasks:

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

Related reference:

"Input node" on page 4511
Use the Input node as an In terminal for an embedded message flow (a subflow).

PeopleSoftInput node

Use the PeopleSoftInput node to interact with a PeopleSoft application.

This topic contains the following sections:

Purpose:
Use the PeopleSoftInput node to interact with PeopleSoft applications. For example, a PeopleSoftInput node monitors a PeopleSoft system for a specified event. When that event occurs, the PeopleSoftInput node generates a message tree that represents the business object with the new event details. The message tree is propagated to the Out terminal so that the rest of the message flow can use the data to update other systems, or audit the changes.

The PeopleSoftInput node is contained in the **WebSphere Adapters** drawer of the message flow node palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:
To function correctly, the PeopleSoftInput node needs an adapter component, which you set using the `Adapter component` node property, and business object definitions, which are stored in the message set that you reference from the node. For this reason, you must provide a message set. By default, the message that is propagated from the PeopleSoftInput node is in the DataObject domain, so the `Message domain` property is set to DataObject. You cannot specify a different domain. The message type is detected automatically by the node.

The PeopleSoftInput node populates the route to label destination list with the name of the method binding. If you add a RouteToLabel node to the message flow after the PeopleSoftInput node, the RouteToLabel node can use the name of the method binding to route the message to the correct part of the message flow for processing.

You can deploy only one input node that uses a particular adapter component to an execution group, but you can deploy many input nodes that use different adapter components to an execution group.

You can use the **mqsisetdbparms** command in the following format to configure an account name with a user name and password for the Adapter for PeopleSoft Enterprise.

```
mqsisetdbparms broker name -n adapter name -u user name -p password
```

For example:

```
mqsisetdbparms BRK1 -n eis::PeopleSoftCustomerInbound.inadapter -u peoplesoftuid -p ********
```

Using configurable services for PeopleSoft nodes

PeopleSoft nodes can get PeopleSoft connection details from either the adapter component or a configurable service. By using a configurable service, you can change the connection details for an adapter without the need to redeploy the adapter. For more details about creating, changing, reporting, and deleting the configurable services for PeopleSoft, see “Changing connection details for PeopleSoft adapters” on page 722.

Terminals and properties:
When you have put an instance of the PeopleSoftInput node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view. If you double-click a PeopleSoftInput node, you open the Adapter Connection wizard. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The PeopleSoftInput node terminals are described in the following table.

Terminal	Description
Out	Business object events from the adapter are sent to the Out terminal.
Failure	If an error happens in the PeopleSoftInput node, the message is propagated to the Failure terminal. Information about the error, and business object events can also be propagated to the Failure terminal.
Catch	Business object events are sent to the Catch terminal if they cause an uncaught exception in the message flow. If the Catch terminal is not connected, the retry process is activated to handle the business object.

The following table describes the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The PeopleSoftInput node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, PeopleSoftInput.	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The PeopleSoftInput node Basic properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Primary adapter component	Yes	Yes		The name of the adapter component that contains configuration properties for the adapter. Either enter a name of an adapter file or click **Browse** to select an adapter file from the list of files that are available in referenced message set projects. When the PeopleSoftInput node receives data from the PeopleSoft system, it associates that data with a method name, depending on the service operation name that is assigned to that type of data when you run the Adapter Connection wizard. The PeopleSoftInput node attempts to handle methods that are defined in the primary adapter. If the type of data that is received does not correspond to any of the methods that are defined in the primary adapter, the node can handle methods that are defined in matching secondary adapters that are deployed to the same execution group.	adapterComponent
Secondary adapter mode	No	Yes	None	Specifies whether the node can handle methods that are defined in secondary adapters. If you set the `Secondary adapter mode` property to `None`, the node handles only methods that are defined in the primary adapter. If the type of data that is received does not correspond to any of the methods that are defined in the primary adapter, a failure occurs. If you set this property to `All adapters in execution group`, the node can handle methods that are defined in any PeopleSoft inbound adapters that are deployed to the same execution group.	secondaryAdapterMode

The PeopleSoftInput node Routing properties are described in the following table.

Property	M	C	Default	Description
Set destination list	No	No	Selected	This property specifies whether to add the method binding name to the route to label destination list. If you select this check box, the method binding name is added so that you can use a RouteToLabel node in the message flow after the PeopleSoftInput node.
Label prefix	No	No		The prefix to add to the method name when routing to label. Add a label prefix to avoid a clash of corresponding label nodes when you include multiple WebSphere Adapters input nodes in the same message flow. By default, there is no label prefix, so the method name and label name are identical.

The PeopleSoftInput node Input Message Parsing properties are described in the following table.

Property	M	C	Default	Description
Message domain	No	No	DataObject	The domain that is used to parse the incoming message. By default, the message that is propagated from the PeopleSoftInput node is in the DataObject domain. You cannot specify a different domain.

Property	M	C	Default	Description
Message set	Yes	No	Set automatically	The name of the message set in which the incoming message is defined. This field is set automatically from the Adapter component property. If you set this property, then subsequently update the project dependencies to remove this message set reference, a warning is issued. Either update the `Message set` property, or restore the reference to this message set project.
Message type	No	No		The name of the incoming message. The node detects the message type automatically. You cannot set this property.
Message format	No	No		The name of the physical format of the incoming message. You cannot set this property.

The PeopleSoftInput node Transactionality properties are described in the following table.

Property	M	C	Default	Description
Transaction mode	No		Yes	The transaction mode on this input node determines whether the rest of the nodes in the flow operate under sync point.

The Instances properties of the PeopleSoftInput node are described in the following table. For a full description of these properties, refer to "Configurable message flow properties" on page 4020.

Property	M	C	Default	Description	**mqsiapplybaroverride command property**
Additional instances pool	No	Yes	`Use Pool Associated with Message Flow`	The pool from which additional instances are obtained. • If you select `Use Pool Associated with Message Flow`, additional instances are obtained from the message flow value. • If you select `Use Pool Associated with Node`, additional instances are allocated from the node's additional instances based on the number specified in the `Additional instances` property.	componentLevel
Additional instances	No	Yes	`0`	The number of additional instances that the node can start if the `Additional instances pool` property is set to `Use Pool Associated with Node`. By default, no additional instances are given to the node.	additionalInstances

The PeopleSoftInput node Retry properties are described in the following table.

Property	M	C	Default	Description	**mqsiapplybaroverride command property**
Retry mechanism	No	No	Failure	This property specifies how retry processing is handled when a failure is rolled back to the PeopleSoftInput node. • If you select `Failure`, retry processing is not performed so you cannot set the remaining properties. • If you select `Short and long retry`, retry processing is performed first at the interval that is specified by the `Short retry interval` property, and if that is unsuccessful, it is then performed at the interval that is specified by the `Long retry interval` property.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Retry threshold	No	Yes	0	The maximum number of times that retry processing is performed for short retry.	retryThreshold
Short retry interval	No	Yes	0	The interval between short retry attempts.	shortRetryInterval
Long retry interval	No	Yes	0	The interval between long retry attempts.	longRetryInterval

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Overview of WebSphere Adapter for PeopleSoft Enterprise" on page 2013
With the adapter for PeopleSoft Enterprise you can create integrated processes that exchange information with PeopleSoft Enterprise through a standard interface. This interface shields the message flow developer from having to understand the lower level details about implementation of the application or data structures used on the PeopleSoft Enterprise server.

"DataObject parser and domain" on page 1114
Use the DataObject domain to parse and write messages for WebSphere Adapters and CORBA applications.

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Creating a custom event project in PeopleTools" on page 2083
The WebSphere Adapter requires an event project in PeopleSoft to perform asynchronous inbound event processing. Use PeopleTools to create the custom event project.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

PeopleSoftRequest node

Use the PeopleSoftRequest node to interact with a PeopleSoft application.

This topic contains the following sections:
- "Purpose"
- "Using this node in a message flow"
- "Terminals and properties" on page 4636

Purpose:
Use the PeopleSoftRequest node to interact with PeopleSoft applications. For example, a PeopleSoftRequest node requests information from a PeopleSoft Enterprise Information System (EIS). A customer business object is sent to PeopleSoft, causing PeopleSoft to retrieve information about a customer, such as an address and account details. The response information that is retrieved by the PeopleSoftRequest node can then be used by the rest of the message flow. The PeopleSoftRequest node can send and receive business data.

The PeopleSoftRequest node is contained in the **WebSphere Adapters** drawer of the message flow node palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:
To function correctly, the PeopleSoftRequest node needs an adapter component, which you set using the `Adapter component` node property, and business object definitions, which are stored in the message set that you reference from the node. For this reason, you must provide a message set. By default, the message that is propagated from the PeopleSoftRequest node is in the DataObject domain, so the `Message domain` property is set to DataObject. You cannot specify a different domain. The message type is detected automatically by the node.

The PeopleSoftRequest node supports local transactions using the broker's Local Transaction Manager, and global transactions using the broker's external syncpoint coordinator.

To effectively maintain the pool of connections to PeopleSoft, you can set a connection timeout value on a configurable service. For more information, see "Configuring EIS connections to expire after a specified time" on page 726.

You can deploy several WebSphere Adapters request nodes that use the same adapter component to an execution group.

You can use the **mqsisetdbparms** command in the following format to configure an account name with a user name and password for the Adapter for PeopleSoft Enterprise.

```
mqsisetdbparms broker name -n adapter name -u user name -p password
```

For example:

```
mqsisetdbparms BRK1 -n eis::PeopleSoftCustomerOutbound.outadapter -u peoplesoftuid -p ********
```

Using configurable services for PeopleSoft nodes

PeopleSoft nodes can get PeopleSoft connection details from either the adapter component or a configurable service. By using a configurable service, you can change the connection details for an adapter without the need to redeploy the adapter. For more details about creating, changing, reporting, and deleting the configurable services for PeopleSoft, see "Changing connection details for PeopleSoft adapters" on page 722.

Terminals and properties:
When you have put an instance of the PeopleSoftRequest node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view. If you double-click a PeopleSoftRequest node, you open the Adapter Connection wizard. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The PeopleSoftRequest node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts the request business object.
Out	The output terminal to which the response business object is sent if it represents successful completion of the request, and if further processing is required within this message flow.
Failure	If an error happens in the PeopleSoftRequest node, the message is propagated to the Failure terminal. Information about the error, and business object events can also be propagated to the Failure terminal.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk on the panel if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The PeopleSoftRequest node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, PeopleSoftRequest	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The PeopleSoftRequest node Basic properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Primary adapter component	Yes	No		The name of the adapter component that contains configuration properties for the adapter. Either enter a name of an adapter file, or click **Browse** to select an adapter file from the list of files that are available in referenced message set projects. When thePeopleSoftRequest node receives data from the PeopleSoft system, it associates that data with a method name. The PeopleSoftRequest node attempts to call methods that are defined in the primary adapter. If the method is not defined in the primary adapter, the node can call methods that are defined in matching secondary adapters that are deployed to the same execution group.	
Secondary adapter mode	No	Yes	None	Specifies whether the node can call methods that are defined in secondary adapters. If you set the `Secondary adapter mode` property to `None`, the PeopleSoftRequest node calls only methods that are defined in the primary adapter. If the method is not defined in the primary adapter, an error occurs. If you set this property to `All adapters in execution group`, the node can call methods that are defined in any PeopleSoft outbound adapter that is deployed to the same execution group.	secondaryAdapterMode
Default method	Yes	Yes		The default method binding to use.	defaultMethod

The PeopleSoftRequest node Response Message Parsing properties are described in the following table.

Property	M	C	Default	Description
Message domain	No	No	DataObject	The domain that is used to parse the response message. By default, the response message that is propagated from the PeopleSoftRequest node is in the DataObject domain. You cannot specify a different domain.
Message set	Yes	No	Set automatically	The name of the message set in which the response message is defined. This field is set automatically from the Adapter component property. If you set this property, then subsequently update the project dependencies to remove this message set reference, a warning is issued. Either update the `Message set` property, or restore the reference to this message set project.
Message type	No	No		The name of the response message. The node detects the message type automatically. You cannot set this property.
Message format	No	No		The name of the physical format of the response message. You cannot set this property.

The PeopleSoftRequest node Transactionality properties are described in the following table.

Property	M	C	Default	Description
Transaction mode	No	No	No	This property specifies that updates are performed independently, not as part of a local transaction. You cannot change this property.

The PeopleSoftRequest node Request properties are described in the following table.

Property	M	C	Default	Description
Method Location	Yes	No	$LocalEnvironment/Adapter/MethodName	The location of the business method (such as createPurchaseOrder or deletePurchaseOrder) that is used to trigger the PeopleSoftRequest node to perform an action on the external system.
Data Location	Yes	No	$Body	The location in the incoming message tree from which data is retrieved to form the request that is sent from the PeopleSoftRequest node to the EIS.

The PeopleSoftRequest node Result properties are described in the following table.

Property	M	C	Default	Description
Output data location	No	No	$OutputRoot	The message tree location to which the PeopleSoftRequest node sends output.
Copy local environment	No	No	Selected	This property controls how the local environment is copied to the output message. If you select the check box, at each node in the message flow, a new copy of the local environment is created in the tree, and it is populated with the contents of the local environment from the preceding node. So if a node changes the local environment, the upstream nodes do not see those changes because they have their own copies. This behavior might be an issue if you are using a FlowOrder node, or if you use the propagate command on a Compute node. If you clear the check box, each node does not generate its own copy of the local environment, but it uses the local environment that is passed to it by the previous node. So if a node changes the local environment, those changes are seen by the upstream nodes.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Overview of WebSphere Adapter for PeopleSoft Enterprise" on page 2013
With the adapter for PeopleSoft Enterprise you can create integrated processes that exchange information with PeopleSoft Enterprise through a standard interface. This interface shields the message flow developer from having to understand the lower level details about implementation of the application or data structures used on the PeopleSoft Enterprise server.

"DataObject parser and domain" on page 1114
Use the DataObject domain to parse and write messages for WebSphere Adapters and CORBA applications.

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Creating a custom event project in PeopleTools" on page 2083
The WebSphere Adapter requires an event project in PeopleSoft to perform asynchronous inbound event processing. Use PeopleTools to create the custom event project.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

PHPCompute node

Use the PHPCompute node to route and transform an incoming message, using the PHP scripting language.

Support for the PHP scripting language is available on all operating systems on which WebSphere Message Broker is supported.

This topic contains the following sections:

Purpose:

The PHPCompute node can use the PHP scripting language to route and transform incoming messages.

Using this node, you can achieve the following goals:

- Examine an incoming message and, depending on its content, propagate it unchanged to the node's output terminal.
- Change part of an incoming message and propagate the changed message to the output terminal by using PHP.

- Create and build a new output message that is independent of the input message by using PHP.

The PHPCompute node is contained in the **Transformation** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using the PHPCompute node in a message flow:
Look at the following sample to see how to use this node:

- PHPCompute Node

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Specifying PHP:
Create PHP statements to customize the behavior of the PHPCompute node. For example, you can customize the node to create one or more output messages by using either new data or the content of an input message or database (unchanged or modified). For example, you might want to modify a value in the input message by adding a value from a database, and store the result in a field in the output message.

Create the PHP statements that you want in a PHP script file and ensure that it exists in the workspace before you associate it with the PHPCompute node.

If the required PHP script file exists, import it into the workspace before associating it with the PHPCompute node (see "Importing file systems into the WebSphere Message Broker Toolkit" on page 2931).

If a PHP file does not exist for this node, create one in the project folder with a file extension of `.php` (for example, `myfile.php`). For more information about creating a PHP script file, see "Creating PHP code for a PHPCompute node" on page 2672.

The PHPCompute node provides support for Simple Network Management Protocol (SNMP). The Management Information Base (MIB) files are installed during the installation of WebSphere Message Broker, and their location is specified by the **`MIBDIRS`** environment variable.

Configuring the PHPCompute node:
When you have put an instance of the PHPCompute node into a message flow, you can configure it. For more information about how to configure nodes, see .

The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (the ones that do not have a default value defined) are marked with an asterisk in that view.

To configure the PHPCompute node:

1. Optional: On the **Description** tab, enter a short description, a long description, or both. You can also rename the node on this tab.

2. On the **Basic** tab, use the `PHP script` property to specify the name of the PHP file. Select the `Invoke 'evaluate()' method in PHP class definition` property if the code in your PHP script file includes an evaluate method.
3. On the **Parser options** tab, select the `Use XMLNSC compact parser for the XMLNS domain` property to specify that the XMLNSC Compact Parser is used for messages in the XMLNS Domain.
4. On the **Validation** tab, specify the parser validation properties of the node. For more information about validation, see "Validating messages" on page 1478. For information about how to complete this tab, see "Validation tab properties" on page 4169.

Terminals and properties:
The PHPCompute node terminals are described in the following table.

Terminal	Type	Description
In	Input data	The input terminal that accepts a message for processing by the node.
Out	Output data	The output terminal to which the transformed message is routed.
Failure	Output data	The output terminal to which the message is routed if a failure is detected during the computation. Even if the `Validate` property is set, messages that are propagated to the Failure terminal of the node are not validated.
* (dynamic)	Dynamic output	Zero or more dynamic output terminals can be created to support message routing.

You can define additional dynamic output terminals on the PHPCompute node. Not all dynamic output terminals that are created on a PHPCompute node need to be mapped to an expression in the filter table. If any of the dynamic output terminals are unmapped, they never have messages propagated to them. Several expressions can map to the same single dynamic output terminal. No static output terminal exists to which the message is passed straight through. For more information about using dynamic terminals, see "Using dynamic terminals" on page 1518.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Description properties of the PHPCompute node are described in the following table:

Property	M	C	Default	Description
Node name	No	No	PHPCompute	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The Basic properties of the PHPCompute node are described in the following table:

Property	M	C	Default	Description	mqsiapplybaroverride command property
PHP script	Yes	Yes		A string containing the name of the PHP script file.	ScriptName

The Parser Options properties of the PHPCompute node are described in the following table.

Property	M	C	Default	Description
Use XMLNSC compact parser for XMLNS domain	No	No	False	This property controls whether the XMLNSC Compact Parser is used for messages in the XMLNS Domain. If you set this property, the message data appears under XMLNSC in nodes that are connected to the output terminal when the input MQRFH2 header or Input Message Parsing property, `Message Domain`, is XMLNS.

The Validation properties of the PHPCompute node are described in the following table.

For a full description of these properties, see "Validation properties" on page 4169.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Validate	No	No	`None`	This property controls whether validation takes place. Valid values are: • `None` • `Content and Value` • `Content` • `Inherit`	validateMaster
Failure action	No	No	Exception	This property controls what happens if validation fails. You can set this property only if you set `Validate` to `Content` or `Content and Value`. Valid values are: • `User Trace` • `Local Error Log` • `Exception` • `Exception List`	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"PHP overview" on page 2671
WebSphere Message Broker provides support for the PHP scripting language.

Related tasks:

"Using PHP" on page 2670
You can use the PHP scripting language for message routing and transformation.

"Using dynamic terminals" on page 1518
You can add, rename, and remove dynamic terminals on a node in the Message Flow editor.

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

"Validating messages" on page 1478
The broker provides validation based on the message model for predefined messages.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

"Creating PHP code for a PHPCompute node" on page 2672
Use these instructions to create your PHP code and associate it with your PHPCompute node.

Related reference:

"Validation properties" on page 4169
You can control validation by setting properties on the Validate and Parser Options tabs for the nodes that are listed in the following table.

"PHP API" on page 5313
The WebSphere Message Broker PHP API provides classes to represent the message, the message assembly, and the element tree.

Publication node

Use the Publication node to filter output messages from a message flow and transmit them to subscribers who have registered an interest in a particular set of topics.

This information contains the following sections:

- "Purpose"
- "Using this node in a message flow" on page 4644
- "Terminals and properties" on page 4644

Purpose:

Use the Publication node to publish a message through the WebSphere MQ queue manager associated with the broker. Applications that expect to receive publications must register a subscription.

The message being published must be:

- A Publish command message
- A Delete Publication command message, or
- Have at least one topic present in the standard properties of the message.

The Publication node uses the topic, or topics, and any options present in the command message to publish the message. The WebSphere MQ queue manager delivers the publication to all subscribing applications matching the topic, and any other options specified on their subscriptions.

The Publication node is contained in the **Routing** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:
Look at the following samples to see how to use this node:
- Scribble
- JMS Nodes
- Pager

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

For an example of how to use this node, assume that you have written a publishing application that publishes stock updates on a regular basis. The application sends the messages to the broker on an MQInput node, and the message flow provides a conversion from the input currency to a number of output currencies. Include a Publication node for each currency that is supported, and set the `Subscription Point` property to a value that reflects the currency in which the stock price is published by the node; for example, Sterling, or USD.

Terminals and properties:
When you have put an instance of the Publication node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view.

The Publication node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
NoMatch	If no subscribers are matched on any of the published topics, the original message is propagated here.
Out	If at least one subscriber is matched by WebSphere MQ on at least one of the published topics, the original message is propagated here.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory*; the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Publication node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type: Publication	The name of the node.

Property	M	C	Default	Description
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The Publication node Basic properties are described in the following table.

Property	M	C	Default	Description
Implicit Stream Naming	Yes	No	Cleared	Select `Implicit Stream Naming` to take the name of the WebSphere MQ queue on which the message was received by the message flow as the stream name. This property provides compatibility with earlier versions of WebSphere MQ Publish/Subscribe, and applies to messages with an MQRFH header when MQPSStream is not specified. Clear the check box if you do not want this action to be taken.
Subscription Point	No	No		The subscription point value for the node. If you do not specify a value for this property, the default subscription point is assumed. Set a subscription point for a Publication node to restrict the forwarding of its publications to those subscribers that specify the subscription point in their subscription (as described in the example scenario in "Using this node in a message flow" on page 4644). For more information, see "Subscription points" on page 2222.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Publish/Subscribe" on page 2215
Publish/subscribe is a style of messaging application in which the providers of information (publishers) are decoupled from the consumers of that information (subscribers).

"Subscription points" on page 2222
A *subscription point* is the name that a subscriber uses to request publications from a particular set of Publication nodes. It is the property of a Publication node that differentiates that Publication node from other Publication nodes in the same message flow.

"Packaging and deployment overview" on page 3210
Deployment is the process of transferring data to an execution group on a broker so that it can take effect in the broker. Message flows and associated resources are packaged in broker archive (BAR) files for deployment.

"Routing using publish/subscribe applications" on page 2215
You can route your messages to applications using the publish/subscribe method of messaging.

Related tasks:

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Using WebSphere MQ cluster queues for input and output" on page 1544
Design your broker network to use WebSphere MQ queues, if appropriate for your business needs.

"Using WebSphere MQ shared queues for input and output (z/OS)" on page 1546
On z/OS systems, you can define WebSphere MQ shared queues as input and output queues for message flows. You might need to serialize access to those messages.

Related reference:

"MQOutput node" on page 4612
Use the MQOutput node to send messages to clients that connect to the broker using the WebSphere MQ Enterprise Transport and that use the MQI and AMI application programming interfaces.

Real-timeInput node

The Real-timeInput node, available in earlier versions of WebSphere Message Broker, is not supported in WebSphere Message Broker Version 7.0. See "Migrating from Version 6.1 products" on page 163 for information about migrating your message flows from WebSphere Message Broker to WebSphere Message Broker Version 7.0.

Real-timeOptimizedFlow node

The Real-timeOptimizedFlow node, available in earlier versions of WebSphere Message Broker, is not supported in WebSphere Message Broker Version 7.0. See "Migrating from Version 6.1 products" on page 163 for information about migrating your message flows from WebSphere Message Broker to WebSphere Message Broker Version 7.0.

RegistryLookup node

Use the RegistryLookup node to retrieve any type of entity held in the WebSphere Service Registry and Repository (WSRR). The entities that match the specified search criteria are stored in the local environment. The input message is not modified.

This topic contains the following sections:

Purpose:

The RegistryLookup node retrieves any type of entity held in WSRR.

Data is retrieved according to search criteria defined by node properties, possibly supplemented or overridden by local environment definitions at run time; see "Local environment overrides" for more details.

The retrieved data is placed in the local environment tree, making it available to subsequent nodes. The input message received by the node is propagated to the output terminal unchanged.

RegistryLookup node processing

The RegistryLookup node is contained in the **Web services** drawer of the message flow node palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

When the RegistryLookup node receives a message the following steps occur in sequence.

1. The RegistryLookup node retrieves the data from WSRR using the specified search criteria.
2. If one or more matches are found, the RegistryLookup node adds a representation of those entities to the local environment tree.
   - If `Match Policy` is set to `One`, a single entity is returned by WSRR and added to the local environment tree. If the registry contains more than one entity that matches the specified search criteria it is not possible to determine which one is returned by WSRR. A different one can be returned each time the query is issued.
   - If `Match Policy` is set to `All`, all matching entities are added to the local environment tree. The order of the entities is determined by WSRR and might vary between queries.

   The input message is propagated unchanged to the Output terminal. The local environment tree is propagated to the Out terminal, where it is available for further processing by transformation nodes. The exact representation of an entity in the local environment tree depends on the `Depth Policy` property. See "RegistryLookup node output" on page 1897 for details of the local environment output tree.

   Add a compute node to your message flow to interpret and act on the returned data. For instance, if the local environment tree contains multiple endpoint addresses for use by subsequent request nodes the Compute node should select the required address and set up any transport headers or local environment settings required by those request nodes.
3. If no matches are found, the RegistryLookup node propagates the input message to the NoMatch terminal.
4. If a processing error occurs, for example if the WSRR server configured on the `DefaultWSRR` configurable service object cannot be connected to, or the connection times out, the RegistryLookup node propagates the input message unchanged to the Failure terminal. The ExceptionList is populated with details of the error.

Local environment overrides:
You can override the RegistryLookup node properties by using local environment settings. See "Dynamically defining the search criteria" on page 1891.

Terminals and properties:
When you have put an instance of the RegistryLookup node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view.

The RegistryLookup node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the message is routed if an error occurs within the node's processing.
Out	The output terminal to which the unmodified input message and updated local environment containing the matched registry data is sent.
NoMatch	The terminal to which the input message is sent if no matching entity is found based on the specified search criteria.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The RegistryLookup node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type (RegistryLookup)	The name of the node
Short description	No	No	None	A brief description of the node
Long description	No	No	None	Text that describes the purpose of the node in the message flow

The RegistryLookup node Basic properties are described in the following table.

Property	M	C	Default	Description	**mqsiapplybaroverride command property**
Name	No	Yes	None	Enter the string values for one or more of `Name`, `Namespace`, and `Version` for the entities or artifacts that you want to retrieve from WSRR. At least one of the properties is required. If you leave all three property values blank, you will get an error message when you try to save.	name
Namespace	No	Yes	None		namespace
Version	No	Yes	None		serviceVersion

Property	M	C	Default	Description	mqsiapplybaroverride command property
User Properties	No	No	None	Allows a query to specify user-defined properties. Add `User Properties` by clicking **Add**. `User Properties` refer to the Additional Properties that are used to catalog the entities in WSRR. Enter values for `Property Name`, which is the case sensitive match of the additional property in WSRR, `Property Type`, and `Property Value`. The `Property Type` can be: • a String (the default), in which case the `Property Value` is a character string to be matched with the additional property value present in WSRR • XPATH, or ESQL, in which case the `Property Value` is a XPath or ESQL expression which locates a field in the message tree that contains the character string to be matched with the property value present in WSRR.	
Classification	No	No	None	The Web Ontology Language (OWL) classification system property. Each classifier is a class in OWL, and has a Uniform Resource Identifier (URI). Using classifications in the registry can help to make objects easier to find and can also add meaning to custom objects that are unique to a particular system. Add a `Classification` by clicking **Add** and typing the complete fully qualified OWL URI for the OWL classification. For example, it can define a particular service endpoint's lifecycle state.	
Match Policy	Yes	No	`One`	WSRR can contain multiple entities that match the search criteria specified by the properties above. If `Match Policy` is set to `One`, at most one matching entity is returned. If `Match Policy` is set to `All`, all matching entities are returned. See "RegistryLookup node output" on page 1897	

The RegistryLookup node Advanced properties are described in the following table.

Property	M	C	Default	Description
Depth Policy	Yes	No	`Return matched only (Depth = 0)`	Specify the depth of the WSRR query and the contents of the entity data to be returned. The returned entities are stored in the ServiceRegistry message tree in the LocalEnvironment. Select `Return matched only (Depth = 0)` to use a WSRR query depth of 0, and to return only the matched entities. Select `Return matched showing immediate relationships (For compatibility only)` to use a WSRR query depth of 1, and to return only the matched entities. Returned entities include elements listing the bsrURIs of related child entities. This option provides compatibility with versions of WebSphere Message Broker before Version 6.1.0.4, using the output format that was used in those prior versions. This option is deprecated, use one of the other options. Select `Return matched plus immediate related entities (Depth = 1)` to use a WSRR query depth of 1, and to return the matched entities and the immediate related child entities. Select `Return matched plus all related entities (Depth = -1)` to use a WSRR query depth of -1, and to return the matched entities and all the related child entities.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message flow aggregation" on page 2718
Aggregation is the generation and fan-out of related requests that are derived from a single input message, and the fan-in of the corresponding replies to produce a single aggregated reply message.

"Dynamically defining the search criteria" on page 1891
You can use the RegistryLookup and EndpointLookup nodes to issue WebSphere Service Registry and Repository (WSRR) queries specified in the local environment.

"RegistryLookup node output" on page 1897
Use the RegistryLookup node to retrieve any type of entity held in the WebSphere Service Registry and Repository (WSRR). The entities that match the specified search criteria are stored in the local environment. The input message is not modified.

"WebSphere Service Registry and Repository" on page 1875
The WebSphere Service Registry and Repository (WSRR) is a central repository of entities. A wide range of entities can be stored and retrieved, including user-defined concepts and definitions related specifically to Web services, such as WSDL services, service interfaces, and associated policies.

"Configurable services" on page 1296
Configurable services are typically runtime properties. You can use them to define properties that are related to external services on which the broker relies; for example, an SMTP server or a JMS provider.

Related tasks:

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Configuring aggregation flows" on page 2721
Use aggregation message flows to generate and fan-out a number of related requests, fan-in the corresponding replies, and compile those replies into a single aggregated reply message, by using the AggregateControl, AggregateRequest, and AggregateReply nodes.

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

Related reference:

"EndpointLookup node" on page 4407
Use the EndpointLookup node to retrieve service endpoint information held in the WebSphere Service Registry and Repository (WSRR). The entities that match the specified search criteria are stored in the local environment. Depending on the node configuration, the Web service URL destination used by the SOAP and HTTP request nodes can also be set. The input message is not modified.

"XSLTransform node" on page 4968
Use the XSLTransform node to transform an XML message to another form of message, according to the rules provided by an XSL (Extensible Stylesheet Language) style sheet, and to set the `Message domain`, `Message set`, `Message type`, and `Message format` for the generated message.

Resequence node

Use the Resequence node to control the sequence in which a group (or groups) of incoming messages are propagated in a message flow.

This topic contains the following sections:
- "Purpose"
- "Using the Resequence node in a message flow" on page 4653
- "Configuring the Resequence node" on page 4653
- "Terminals and properties" on page 4655

Purpose:
The Resequence node controls the sequence in which a group (or groups) of incoming messages are propagated through the node in a message flow.

You can use a Resequence node to rearrange groups of messages into sequential order according to a sequence number in the message. Each message must contain a sequence number, which can be any positive or negative integer. The sequence number is calculated by an XPath expression defined in the `Path to sequence number` property on the node, and it can be one that was added to the message by a Sequence node.

The Resequence node can reorder multiple sequence groups independently of each other, but it cannot reorder messages between sequence groups. The group to which a message belongs is determined by the properties of the Resequence node.

Each sequence group can be associated with only one Resequence node. Multiple Resequence nodes can have a sequence group with the same name, but each of those sequence groups is treated as a separate group. The combination of the execution group name, message flow name, node name, and sequence group name is used to differentiate between the sequence groups.

For example, you might have a message flow called *flow1* containing a Resequence node called *node1*, which is deployed to an execution group called *eg1*. A message is sent to it using a sequence group called *group1*. The result is *eg1/flow1/node1/group1*. Exactly the same message flow in a different execution group, for example *eg2*, would result in *eg2/flow1/node1/group1*.

You can configure a Resequence node to use multiple threads for propagating messages, but only if each message that is being propagated belongs to a different sequence group. For messages belonging to the same sequence group, only one thread at a time can be used to propagate messages. As a result, the sequential order of messages in a sequence group is preserved, but no order between groups is maintained.

A transaction break occurs at the Resequence node. Messages arriving at the node are not directly propagated to the output terminal; all messages (including the next message in the sequence) are initially serialized to an internal WebSphere MQ queue. The storing of the message occurs in the current transaction; when it has been stored, the transaction is completed. If a stored message is the next one in the sequence, it is propagated down the message flow under a new transaction. Only the serializable part of the data is propagated from the node; local environment, environment, and exception lists are not preserved.

Any exceptions that occur in nodes following the Resequence node are rolled back to the Catch terminal of the Resequence node. If the Catch terminal is not connected to any other nodes, the messages are re-delivered to the original terminal (Out, Missing, or Expired). The messages are never backed-out or discarded.

The Resequence node stores all received messages onto internal queues before propagating messages downstream, even when they are received in order. The message flow is effectively split into two running flows (one before the Resequence node and one after it. If the part of the message flow before the Resequence node runs significantly faster than the part of the message flow after it, the number of messages on the internal queue can increase more quickly than can be processed. Also, any expiry times given for end of groups or missing messages do not occur in a timely manner. When the queues are full, messages arriving at the Resequence node cause exceptions to be thrown. You can avoid this problem by following these steps:

1. Ensure that the controlling applications send only a finite amount of work at a time, which the message flow can manage.
2. Configure additional instances so that the second part of the message flow has more instances to do work with on. Use the properties on the **Instances** tab to give the Resequence node its own set of additional instances.
3. Structure the message flow so that the part of the flow that places the most demands on the CPU (transformation, for example), comes before the Resequence node.
4. If the large workload is transient, increase the maximum queue depth on the internal queues.
5. Use the Resequence configurable service to partition the queues. This can prevent the situation in which an instance of the Resequence node fills up the queues and stops another instance working.

For information about the various states and state transitions of the Resequence node, see "Resequence node state machines" on page 4658.

The Resequence node is contained in the **Routing** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using the Resequence node in a message flow:
Look at the following sample to see how to use the Resequence node:

- Healthcare

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Configuring the Resequence node:
When you have put an instance of the Resequence node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view.

All mandatory properties that do not have a default value defined are marked with an asterisk.

Configure the Resequence node:

1. Optional: On the **Description** tab, enter a `Short description`, a `Long description`, or both. You can also rename the node on this tab.
2. On the **Basic** tab, set the properties that determine how the message sequence is controlled.
   - Use the `Path to sequence number` property to specify an XPath expression that is used to calculate the sequence number of the message. The XPath expression can calculate the sequence number or it can point to an integer field in the message. Messages can also contain an optional sequence group identifier. This property is mandatory.

- Use the Path to sequence group identifier property to specify the location of the sequence group identifier. The location is specified as an XPath expression. Messages that have the same group identifier are considered part of the same sequence group.
- Use the Start of sequence definition property to identify the first sequence number in each group.
  - Select Literal to specify a literal sequence number, which can be any positive or negative numeric value. When the message with the specified sequence number arrives, the messages are propagated.
  - Select Predicate to specify an XPath expression that evaluates to either True or False, indicating whether the message is the first in the sequence.

    Messages continue to be collected while the expression evaluates to False. When the expression of a message is evaluated to True, it indicates that the message is the first in the sequence.

    Typically, the XPath expression evaluates to a Boolean; however, if other data types are returned, the predicate is determined in the following way:

Table 252.

Returned data type	True	False
Boolean	True	False
Numeric	Any non-zero value	0 or 0.0
String	Any string matching true (case-insensitive)	Any string not matching true (case insensitive)
NodeSet	Never	Always

    When a message evaluates the expression to True (and is therefore identified as the start of the sequence), the node checks that the message has the smallest sequence number collected up to that point. If messages are found with lower sequence numbers, an exception is thrown.

    When the first message that evaluates to true has been processed successfully, the XPath expressions of subsequent messages are not checked. If a message arrives with a lower sequence number than the message that was identified as the start of the sequence, an exception is thrown.
  - Select Automatic to specify a time limit for gathering messages before using the lowest numbered message.
- Use the End of sequence definition property to specify when each sequence group has been completed.
  - Select Literal to specify a literal sequence number. This value can be any positive or negative numeric value that is equal to or greater than the value of the Start of sequence definition property. When the message with the specified sequence number arrives, the sequence group remains open and waits for missing messages until the Missing message timeout property expires. Any messages that arrive within that time are included in the group (unless they are duplicates or outside the allowed range) until the sequence group is closed. When the sequence group is closed, any new messages arriving for that group are treated as belonging to a new instance of the group.
  - Select Predicate to specify an XPath expression that evaluates to either True or False, to indicate whether the message is the last in the sequence.

Typically, the XPath expression evaluates to a Boolean; however, if other data types are returned, the predicate is determined in the way shown in Table 1.

When the predicate evaluates to True, the sequence number of the message is assigned to the `End of sequence definition` property.

When a message has been received with the end of sequence predicate set to True, the sequence group remains open and waits for missing messages until the `Missing message timeout` property expires. Any messages that arrive within that time are included in the group (unless they are duplicates or outside the allowed range) until the sequence group is closed. When the sequence group is closed, any new messages arriving for that group are treated as belonging to a new instance of the group.

If a message arrives with the end of sequence predicate set to True and with a lower sequence number than a message that has already arrived, an exception is thrown.

- Select `Automatic` to specify the timeout period for the node. This option specifies how long (in seconds) the node waits for messages to arrive in an empty queue, before closing the sequence group. This option is useful for applications that cannot determine the final number in the sequence. The timer starts when there are no messages in the queue waiting to be propagated. If new messages arrive before the timeout period is reached, the timer is canceled. If no new messages arrive before the end of the specified time, the sequence group is closed and any further messages for the group are considered part of a new group.

- Use the `Missing message timeout` property to specify how long (in seconds) the node waits for the next message in the sequence before it moves on to the next message in the sequence. Messages that arrive within the specified time limit are propagated in sequential order to the Out terminal.

  When the specified time limit has been exceeded, the messages are propagated in sequential order to the Expire terminal. Subsequent messages in the sequence group are also routed to the Expire terminal. If the missing message eventually arrives, it is propagated to the Missing terminal.

3. On the **Advanced** tab:
   - Use the `Persistence mode` property to specify whether to store incomplete sequences of messages persistently.
   - Use the `Configurable service` property to specify the name of a Resequence configurable service that overrides specified properties of the Resequence node.
4. Optional: On the **Instances** tab, set values for the properties that show the additional instances (threads) that are available for a node.

   Multiple threads can be used to propagate messages from the same Resequence node, but only if each message that is being propagated belongs to a different sequence group. For messages belonging to the same sequence group, only one thread at a time can be used to propagate messages. As a result, the sequential order of messages in a sequence group is preserved, but no order between groups is maintained.

   For more information about specifying additional instances, see “Configurable message flow properties” on page 4020.

Terminals and properties:
The terminals of the Resequence node are described in the following table.

Terminal	Description
In	The input terminal through which the incoming message assembly arrives at the node.
Failure	The output terminal to which the message is routed if an error occurs. This value includes failures caused by retry processing.
Out	The output terminal to which the output message is propagated by default.
Expire	The output terminal to which the message is routed if a timeout occurs while the node is waiting for the message to arrive. All subsequent messages in the same group are also propagated to this terminal.
Missing	The output terminal to which the missing message (which caused a timeout to occur) is routed if it subsequently arrives at the node.
Catch	The output terminal to which the message is routed if an exception is issued downstream and caught by this node. Exceptions are caught only if this terminal is attached.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file for deployment).

The Description properties of the Resequence node are described in the following table.

Property	M	C	Default	Description
Node name	No	No	Resequence	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The Basic properties of the Resequence node are described in the following table.

Property	M	C	Default	Description
Path to sequence number	Yes	No		An XPath expression that calculates the sequence number of the message.
Path to sequence group identifier	No	No		An XPath expression that calculates the sequence group identifier. Messages that have the same group identifier are considered part of the same sequence group. This property functions in the same way as the `Correlation path` property in the Collector node.
Start of sequence definition	Yes	No	`Literal, 0`	Specifies the first sequence number in each group. Valid values are: • `Literal` and *number* • `Predicate` and *XPath* • `Automatic` and *time in seconds* If this property is set to `Automatic`, the associated time is overridden by the value of the `startSequenceSeconds` property, if set, in the Resequence configurable service.

Property	M	C	Default	Description
End of sequence definition	Yes	No	`Automatic`	Specifies when each sequence group has been completed. Valid values are: • `Literal` and *number* • `Predicate` and *XPath* • `Automatic` and *time in seconds* If this property is set to `Automatic`, the associated time is overridden by the value of the `endSequenceSeconds` property, if set, in the Resequence configurable service.
Missing message timeout (seconds)	No	No		Specifies how long (in seconds) the node waits for the next message in the sequence before it moves on to the following one. This property is overridden by the **`Missing message timeout`** property, if set, in the Resequence configurable service.

The Instances properties of the Resequence node are described in the following table. For a full description of these properties, see "Configurable message flow properties" on page 4020.

Property	M	C	Default	Description
Additional instances pool	No	Yes	`Use Pool Associated with Message Flow`	The pool from which additional instances are obtained. • If you select `Use Pool Associated with Message Flow`, additional instances are obtained from the message flow pool. • If you select `Use Pool Associated with Node`, additional instances are allocated from the node's additional instances based on the number specified in the `Additional instances` property.
Additional instances	No	Yes	`0`	The number of additional instances that the node can start if the `Additional instances pool` property is set to `Use Pool Associated with Node`.

The Advanced properties of the Resequence node are described in the following table.

Property	M	C	Default	Description
Persistence mode	Yes	No	`Non-persistent`	Specifies whether to store incomplete sequences of messages persistently. Valid options are: • `Non-persistent` • `Persistent`
Configurable service	No	Yes	None set	This property specifies the name of a Resequence configurable service to be used by the Resequence node. The properties set by the Resequence configurable service override the equivalent properties set on the Resequence node. For more information about the properties than you can set with this configurable service, see "Configurable services properties" on page 3766.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

You cannot use monitoring properties to configure transaction events on the following nodes:

"Collector node" on page 4333

"Resequence node" on page 4651

Use a monitoring profile instead; see "Configuring monitoring event sources using a monitoring profile" on page 762.

Related concepts:

"Message sequencing" on page 2784
Use message sequencing to ensure that messages are delivered to the receiving application in a particular order.

"Message collections" on page 2755
A message collection is a single message that contains multiple messages derived from one or more sources.

Related tasks:

"Using message sequences" on page 2783
You can maintain or change the sequence of messages in a message flow based on the sequence number and group ID contained in each message.

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"Sequence node" on page 4736
Use the Sequence node to add a sequence number to one or more groups of input messages.

"Configurable message flow properties" on page 4020
When you add a message flow to a broker archive (BAR) file in preparation for deploying it to a broker, you can set additional properties that influence its run time operation. These properties are available for review and update when you select the **Manage and Configure** tab for the broker archive file.

"Collector node" on page 4333
Use the Collector node to create message collections based on rules that you configure in the node.

Resequence node state machines:

State machine diagrams show the possible actions and states of a system, and the transitions between those states.

The following topics contain state machine diagrams showing the states, transitions, and actions of the Resequence node:

- "Automatic start and end of sequence" on page 4659
- "Literal start and end of sequence" on page 4661

- “Predicate start and end of sequence” on page 4662

In each of the diagrams, the following syntax is used:

```
<action> [ <predicate list> ] / <result list>
```

where the predicate list and result list are separated by commas.

You can use these diagrams to infer the behavior of other combinations of sequence start and end definitions for a sequence group; for example, combinations such as literal start of sequence and automatic end of sequence.

For more information about state machine diagrams, see the Agile Modeling Web site at `http://www.agilemodeling.com/artifacts/stateMachineDiagram.htm`. For information about the Unified Modeling Language (UML), see the Object Management Group (OMG) UML Web site at `http://www.uml.org/`.

Related concepts:

“Message sequencing” on page 2784
Use message sequencing to ensure that messages are delivered to the receiving application in a particular order.

“Sequence groups” on page 2786
Use sequence groups to control the way in which messages are grouped together for processing by Sequence and Resequence nodes.

Related tasks:

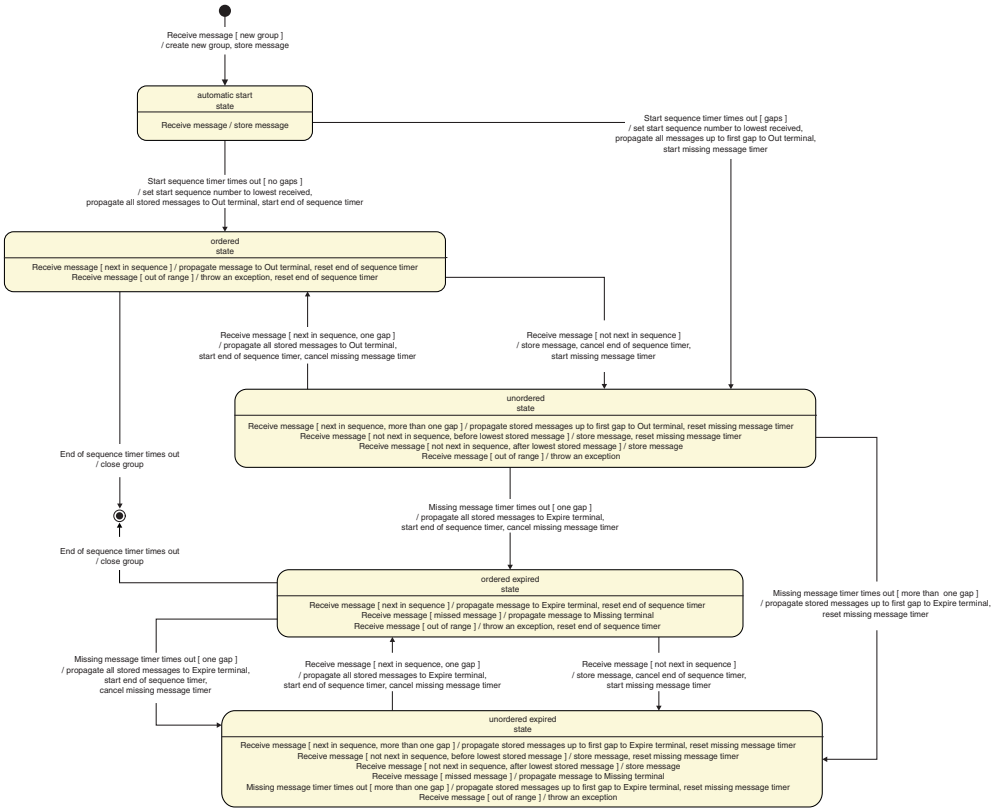
“Using message sequences” on page 2783
You can maintain or change the sequence of messages in a message flow based on the sequence number and group ID contained in each message.

Related reference:

“Resequence node” on page 4651
Use the Resequence node to control the sequence in which a group (or groups) of incoming messages are propagated in a message flow.

Automatic start and end of sequence:

This state machine diagram shows the states, transitions, and actions of the Resequence node with both start and end definitions of a sequence group set to Automatic.



Related concepts:

“Message sequencing” on page 2784
Use message sequencing to ensure that messages are delivered to the receiving application in a particular order.

“Sequence groups” on page 2786
Use sequence groups to control the way in which messages are grouped together for processing by Sequence and Resequence nodes.

Related tasks:

“Using message sequences” on page 2783
You can maintain or change the sequence of messages in a message flow based on the sequence number and group ID contained in each message.

Related reference:

“Resequence node state machines” on page 4658
State machine diagrams show the possible actions and states of a system, and the transitions between those states.

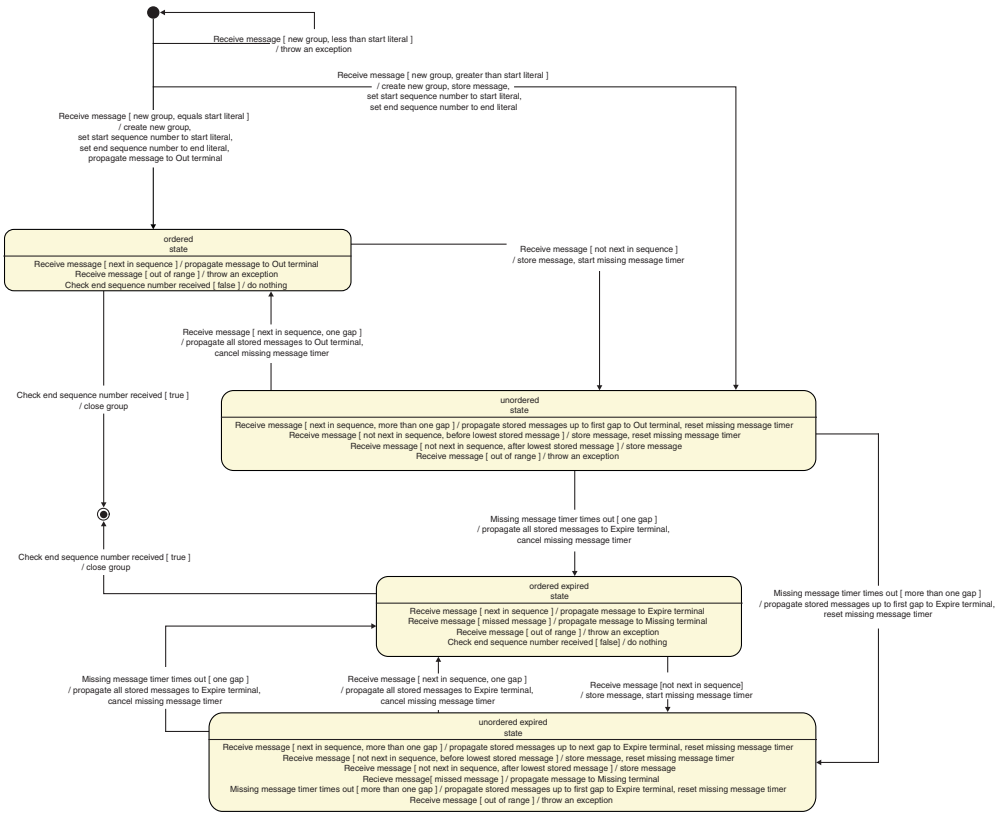
“Literal start and end of sequence” on page 4661
This state machine diagram shows the states, transitions, and actions of the Resequence node with both start and end definitions of a sequence group set to Literal.

“Predicate start and end of sequence” on page 4662
This state machine diagram shows the states, transitions, and actions of the Resequence node with both start and end definitions of a sequence group set to Predicate.

“Resequence node” on page 4651
Use the Resequence node to control the sequence in which a group (or groups) of incoming messages are propagated in a message flow.

Literal start and end of sequence:

This state machine diagram shows the states, transitions, and actions of the Resequence node with both start and end definitions of a sequence group set to Literal.



Related concepts:

"Message sequencing" on page 2784
Use message sequencing to ensure that messages are delivered to the receiving application in a particular order.

"Sequence groups" on page 2786
Use sequence groups to control the way in which messages are grouped together for processing by Sequence and Resequence nodes.

Related tasks:

"Using message sequences" on page 2783
You can maintain or change the sequence of messages in a message flow based on the sequence number and group ID contained in each message.

Related reference:

"Resequence node state machines" on page 4658
State machine diagrams show the possible actions and states of a system, and the transitions between those states.

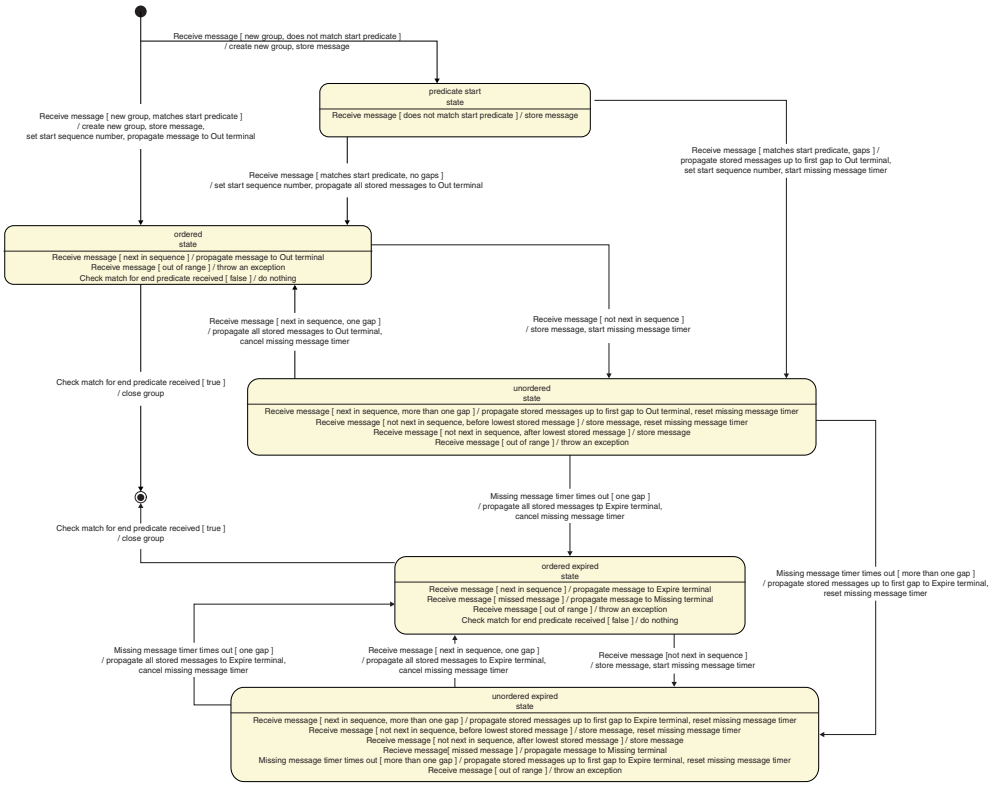
"Predicate start and end of sequence" on page 4662
This state machine diagram shows the states, transitions, and actions of the Resequence node with both start and end definitions of a sequence group set to Predicate.

“Automatic start and end of sequence” on page 4659
This state machine diagram shows the states, transitions, and actions of the Resequence node with both start and end definitions of a sequence group set to Automatic.

“Resequence node” on page 4651
Use the Resequence node to control the sequence in which a group (or groups) of incoming messages are propagated in a message flow.

Predicate start and end of sequence:

This state machine diagram shows the states, transitions, and actions of the Resequence node with both start and end definitions of a sequence group set to Predicate.



Related concepts:

“Message sequencing” on page 2784
Use message sequencing to ensure that messages are delivered to the receiving application in a particular order.

“Sequence groups” on page 2786
Use sequence groups to control the way in which messages are grouped together for processing by Sequence and Resequence nodes.

Related tasks:

“Using message sequences” on page 2783
You can maintain or change the sequence of messages in a message flow based on the sequence number and group ID contained in each message.

Related reference:

“Resequence node state machines” on page 4658
State machine diagrams show the possible actions and states of a system, and the

transitions between those states.

"Automatic start and end of sequence" on page 4659
This state machine diagram shows the states, transitions, and actions of the Resequence node with both start and end definitions of a sequence group set to Automatic.

"Literal start and end of sequence" on page 4661
This state machine diagram shows the states, transitions, and actions of the Resequence node with both start and end definitions of a sequence group set to Literal.

"Resequence node" on page 4651
Use the Resequence node to control the sequence in which a group (or groups) of incoming messages are propagated in a message flow.

ResetContentDescriptor node

Use the ResetContentDescriptor node to request that the message is reparsed by a different parser.

This topic contains the following sections:
- "Purpose"
- "Using this node in a message flow" on page 4664
- "Configuring the ResetContentDescriptor node" on page 4664
- "Terminals and properties" on page 4666

Purpose:
The node associates the new parser information with the input message bit stream. If the message has been parsed already to create a message tree, and the contents of the tree have been modified (for example, by a Compute node), the ResetContentDescriptor node must re-create the bit stream from the message tree by calling the current parser.

If your message flow has updated the message before it is received by the ResetContentDescriptor node, ensure that the changed message contents are still valid for the current parser. If the contents are not valid, the parser generates an error when it attempts to re-create the bit stream from the message tree, and the ResetContentDescriptor node raises an exception. For example, if you have added a new field to a message in the MRM domain, and the field is not present in the model, the recreation of the bit stream fails.

If you specify MRM as the new parser, you can also specify a different message template (message set, message type, and message format). This node does not reparse the message, but the properties that you set for this node determine how the message is parsed when it is next reparsed by the message flow.

The ResetContentDescriptor node does not:
- Change the message content; it changes message properties to specify the way in which the bit stream is parsed next time that the parser is started.
- Convert the message from one format to another; for example, if the incoming message has a message format of XML and the outgoing message format is binary, the ResetContentDescriptor node does not do any reformatting. It starts the parser to re-create the bit stream of the incoming XML message, which retains the XML tags in the message. When the message is reparsed by a subsequent node, the XML tags are not valid and the parse fails.

The ResetContentDescriptor node is contained in the **Construction** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:
For an example of how to use this node, assume that you want to swap between the BLOB and the MRM domains. The format of an incoming message might be unknown when it enters a message flow, therefore the BLOB parser is started. Later on in the message flow, you might decide that the message is predefined as a message in the MRM domain, and you can use the ResetContentDescriptor node to set the correct values to use when the message is parsed by a subsequent node in the message flow.

The following table shows typical ResetContentDescriptor node properties.

Property	Value
Message domain	MRM
Reset message domain	Selected
Message set	MyMessageSet
Reset message set	Selected
Message type	m_MESSAGE1
Reset message type	Selected
Message format	Text1
Reset message format	Selected

The `Message domain` is set to MRM, and the MRM parser is started when the message is next parsed. The `Message set`, `Message type`, and `Message format` are the message template values that define the message model, and all of the reset check boxes are selected because all of the properties need to change.

The ResetContentDescriptor node causes the BLOB parser that is associated with the input message to construct the physical bit stream of the message (not the logical tree representation of it), which is later passed to the MRM parser. The MRM parser then parses the bit stream using the message template (`Message set`, `Message type`, and `Message format`) specified in this ResetContentDescriptor node.

In Version 6.1, you do not need to include a ResetContentDescriptor node after an XSLTransform node in your message flow to set `Message domain`, `Message set`, `Message type`, and `Message format` of the message generated by the XSLTransform node. The XSLTransform node performs this function.

Configuring the ResetContentDescriptor node:
When you have put an instance of the ResetContentDescriptor node into a message flow, you can configure the node. For more information, see . The properties of the node are displayed in the Properties view.

All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

1. Optional: On the **Description** tab, enter a Short description, a Long description, or both. You can also rename the node on this tab.
2. On the **Basic** tab:
   a. To use a different parser associated with the message, specify the new domain in the Message domain property:
      - XMLNSC
      - DataObject
      - JSON
      - BLOB
      - MIME
      - MRM
      - JMSMap
      - JMSStream
      - XMLNS

      You can also specify a user-defined parser if appropriate.

      You must also select the Reset message domain check box.

      If you leave the Message domain property blank and do not select the Reset message domain check box, the domain is not reset. If you leave the Message domain property blank and select the Reset message domain check box, the default value is BLOB.
   b. If the MRM, XMLNSC, or IDOC parser is to reparse the message, specify the other properties of the model that are to be associated with the input message, and select the relevant reset check box beneath each field. If you select a reset check box for a property and you have not specified a value for that property, the value of that property is reset to blank. Alternatively, if you have specified a value for that property, the property is not blank. If you do not select the reset check box for a property, the value for that property is inherited from the incoming message. If the parser is associated with the input message already, specify only the properties that are to change.
      1) Define the Message set. Choose a value from the list of available message models (the name and identifier of the message model are shown), and select the Reset message set check box.
      2) For MRM domains, define the name of the message in Message type. Enter the name and select the Reset message type check box.
      3) For MRM and IDOC, define the Message format. This property specifies the physical format for the parser. You can select one of the formats from the list (which lists the names of those formats that you have defined on the Message set specified previously), and select Reset message format.

      These actions are taken only if suitable headers exist. If the message does not have an MQRFH2 header, the node does not create one.
3. On the **Parser Options** subtab:
   a. Parse timing is, by default, set to On Demand, which causes parsing of the message to be delayed.

      For more details, see "Parsing on demand" on page 4173.
   b. Select Use MQRFH2C compact parser for MQRFH2 header if you want the MQRFH2C parser to be used. By default, this check box is cleared, which means that the compact parser is not used.

c. If you are using the XMLNSC parser, set values for the properties that determine how the XMLNSC parser operates. For more information, see "Manipulating messages in the XMLNSC domain" on page 2546.

4. On the **Validation** tab, set the validation properties if you want the parser to validate the body of messages against the `Message set`. (If a message is propagated to the Failure terminal of the node, it is not validated.)

   For more details, see "Validating messages" on page 1478 and "Validation properties" on page 4169.

Terminals and properties:
The ResetContentDescriptor node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the message is routed if an error is detected by the node.
Out	The output terminal to which the message is routed if a new parser is identified by the properties.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Description properties of the ResetContentDescriptor node are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The Basic properties of the ResetContentDescriptor node are described in the following table.

The ResetContentDescriptor node Basic properties are described in the following table.

Property	M	C	Default	Description
Message domain	No	No	`BLOB`	The message domain that is associated with the message that you want to reparse.
Reset message domain	Yes	No	Cleared	If you select the reset check box, the `Message domain` property is reset. In this case, if you do not select a value for the `Message domain` property, the `Message domain` property value is `BLOB`.

Property	M	C	Default	Description
Message set	No	No		The message set that is associated with the message that you want to reparse. If you set this property, then later update the project dependencies to remove this message set reference, a warning is issued. Either update the `Message set` property, or restore the reference to this message set project.
Reset message set	Yes	No	Cleared	If you select the reset check box, the `Message set` property is reset. In this case, if you do not select a value for the `Message set` property, the `Message set` property value is blank.
Message type	No	No		The message type that is associated with the message that you want to reparse.
Reset message type	Yes	No	Cleared	If you select the reset check box, the `Message type` property is reset. In this case, if you do not select a value for the `Message type` property, the `Message type` property value is blank.
Message format	No	No		The message format that is associated with the message that you want to reparse. If you set the `Message domain` property to `DataObject`, you can set this property to `XML` or `SAP ALE IDoc`. Set this property to `SAP ALE IDoc` when you need to parse a bit stream from an external source and generate a message tree.
Reset message format	Yes	No	Cleared	If you select the reset check box, the `Message format` property is reset. In this case, if you do not select a value for the `Message format` property, the `Message format` property value is blank.

The Parser Options properties of the ResetContentDescriptor node are described in the following table.

Property	M	C	Default	Description
Parse timing	No	No	On Demand	This property controls when the reparsed message is parsed. Valid values are On Demand, Immediate, and Complete. For a full description of this property, see "Parsing on demand" on page 4173.
Use MQRFH2C compact parser for MQRFH2 header	No	No	Cleared	This property controls whether the MQRFH2C compact parser, instead of the MQRFH2 parser, is used for MQRFH2 headers.
Build tree using XML schema data types	No	No	Cleared	This property controls whether the XMLNSC parser creates syntax elements in the message tree with data types taken from the XML Schema. You can select this property only if you set the `Validate` property on the **Validation** tab to `Content` or `Content and Value`.
Use XMLNSC compact parser for XMLNS domain	No	No	Cleared	This property controls whether the XMLNSC Compact Parser is used for messages in the XMLNS Domain. If you set this property, the message data is displayed under XMLNSC in nodes that are connected to the output terminal when the input MQRFH2 header or Domain is XMLNS.
Retain mixed content	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters mixed text in the reparsed message. If you select the check box, elements are created for mixed text. If you clear the check box, mixed text is ignored and no elements are created.

Property	M	C	Default	Description
Retain comments	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters comments in the reparsed message. If you select the check box, elements are created for comments. If you clear the check box, comments are ignored and no elements are created.
Retain processing instructions	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters processing instructions in the reparsed message. If you select the check box, elements are created for processing instructions. If you clear the check box, processing instructions are ignored and no elements are created.
Opaque elements	No	No	Blank	This property is used to specify a list of elements in the reparsed message that are to be opaquely parsed by the XMLNSC parser. Opaque parsing is performed only if validation is not enabled (that is, if the value of the `Validate` property is set to `None`); entries that are specified in `Opaque Elements` are ignored if validation is enabled.

The Validation properties of the ResetContentDescriptor node are described in the following table. For a full description of these properties, see "Validation properties" on page 4169.

Property	M	C	Default	Description	**`mqsiapplybaroverride` command property**
Validate	No	Yes	None	This property controls whether validation takes place. Valid values are None, Content, Content and Value, and Inherit.	validateMaster
Failure action	No	No	Exception	This property controls what happens if validation fails. You can set this property only if you set `Validate` to Content and Value or Content. Valid values are User Trace, Local Error Log, Exception, and Exception List.	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Specifying namespaces in the Message Type property" on page 1208
When using the MRM domain, the *Message Type* property is used to specify the name of the message.

"Multipart messages" on page 1191
A *multipart message* contains one or more other messages within its structure. The contained message is sometimes referred to as an *embedded message*.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Parsers" on page 1072
A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from the internal message tree representation.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Validating messages" on page 1478
The broker provides validation based on the message model for predefined messages.

"Manipulating messages in the XMLNSC domain" on page 2546
If you are writing ESQL to process messages in the XMLNSC domain, it is helpful to learn about the structure of the message tree that the XMLNSC parser builds.

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

Related reference:

"Validation properties" on page 4169
You can control validation by setting properties on the Validate and Parser Options tabs for the nodes that are listed in the following table.

"Parsing on demand" on page 4173
On-demand parsing, referred to as partial parsing, is used to parse an input message bit stream only as far as is necessary to satisfy the current reference. The parsers that can perform partial parsing of input messages are the JSON, MRM, XML, XMLNS, and XMLNSC parsers.

Route node

Use the Route node to direct messages that meet certain criteria down different paths of a message flow.

This topic contains the following sections:

Purpose:
As an example, you can forward a message to different service providers, based on the request details. You can also use the Route node to bypass unnecessary steps. For example, you can check to see if certain data is in a message, and perform a database lookup operation only if the data is missing. If you set the `Distribution Mode` property to `All`, you can trigger multiple events that each require different

conditions. For example, you can log requests that relate to a particular account identifier, and send requests that relate to a particular product to be audited.

You use XPath filter expressions to control processing. A result of a filter expression is cast as Boolean, so the result is guaranteed to be either true or false. For more information about XPath 1.0 query syntax, see W3C XPath 1.0 Specification.

The Route node is contained in the **Routing** drawer of the message flow node palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:
Look at the following sample to see how to use this node:

- Simplified Database Routing

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

The Route node has one input terminal and a minimum of three output terminals: Match, Default, and Failure. The Default and Failure output terminals are static, so they are always present on the node. The dynamic Match terminal is created automatically each time a new Route node is selected and used in the Message Flow editor. This behavior means that you do not always have to create the first dynamic output terminal for this node, which is the minimum number of terminals needed for this node to operate. You can rename this dynamic terminal if "Match" is not an appropriate name.

A message is copied to the Default terminal if none of the filter expressions are true. If an exception occurs during filtering, the message is propagated to the Failure terminal. The Route node can define one or more dynamic output terminals. For all terminals, the associated filter expression is applied to the input message and, if the result is true, a copy of the message is routed to the specified terminal. The Route node determines the order in which the terminals are driven. The node always propagates messages to the terminals in the order in which they appear in the filter table.

Each filter expression is applied to the input message in the order that is given in the filter table. If the result is true, a copy of the message is routed to its associated dynamic output terminal. If you set the `Distribution Mode` property to `First`, application of all filter expressions might not occur.

Consider the following example input message:

```
<EmployeeRecord>
 <EmployeeNumber>00001</EmployeeNumber>
 <FamilyName>Smith</FamilyName>
 <Wage>20000</Wage>
</EmployeeRecord>
```

and the following XPath filter expressions:

```
$Root/XMLNSC/EmployeeRecord/EmployeeNumber="00002"|Match
$Root/XMLNSC/EmployeeRecord/EmployeeNumber="00001"|out_exp2
```

In this example, the `Distribution Mode` property is set to `First`. The Route node processes the XPath filter expressions, in the order in which the are displayed, against the input message. Because the `Distribution Mode` property is set to `First`, the unmodified input message is propagated only once to the dynamic output terminal that is mapped to the first filter expression that resolves to true. In the previous example, the first filter expression, which is associated with the Match terminal, is false because the employee number in the input message is not "00002". Therefore no message is propagated to the Match terminal. The second filter expression is true, therefore a copy of the input message is routed to the "out_expr2" dynamic terminal. If the employee number in the input message is "00003", and therefore does not match either filter expression, the message is propagated to the static Default output terminal. If the `Distribution Mode` property is set to `All` for this example, the same outcome is achieved because only one filter expression is true.

Terminals:
The Route node terminals are described in the following table.

Terminal	Description
In	The static input terminal that accepts a message for processing by the node.
Match	A dynamic output terminal to which the original message can be routed when processing completes successfully. You can create additional dynamic terminals; see “Dynamic terminals.”
Default	The static output terminal to which the message is routed if no filter expression resolves to true.
Failure	The static output terminal to which the message is routed if a failure is detected during processing.

Dynamic terminals

The Route node can have further dynamic output terminals. Not all dynamic output terminals that are created on a Route node have to be mapped to an expression in the filter table. Messages are never propagated to unmapped dynamic output terminals. Several expressions can map to the same single dynamic output terminal. No static output terminal exists to which the message is passed straight through. For more information about using dynamic terminals, see “Using dynamic terminals” on page 1518.

Properties:
When you have put an instance of the Route node into a message flow, you can configure it. For more information, see “Configuring a message flow node” on page 1503. The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those properties that do not have a default value defined) are marked with an asterisk.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Route node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, Route	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The Route node Basic properties are described in the following table.

Property	M	C	Default	Description	`mqsiapplybaroverride` command property
Filter table	Yes	No		A table where all rows are expressions and associated terminal names that define the switching that is performed by this node following evaluation of each filter expression. The full expression is in the format *`XPath filter expression`*`,` *`terminal name`* All XPath expressions must start with $Root, $Properties, $LocalEnvironment, $DestinationList, $ExceptionList, or $Environment. If you are creating an expression by hand, you can also start the expression with $Body. However, the XPath Expression Builder and associated validation in the WebSphere Message Broker Toolkit do not support use of the $Body variable. If you are using the XPath Expression Builder, use the $Root variable instead. Expressions are evaluated in the order in which they are displayed in the table. To improve performance, specify the expressions that are satisfied most frequently at the top of the filter table. Typically, you specify a unique terminal name for each XPath expression.	
Distribution Mode	No	Yes	`All`	This property determines the routing behavior of the node when an inbound message matches multiple filter expressions. If you set the `Distribution Mode` property to `First`, the message is propagated to the associated output terminal of the first expression in the table that resolves to true. If you set this property to `All`, the message is propagated to the associated output terminal for each expression in the table that resolves to true. If no output terminal matches, the message is propagated to the Default terminal.	distributionMode

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Message flow node terminals" on page 1034
A terminal is the point at which one node in a message flow is connected to another node.

"XPath overview" on page 1507
The XML Path Language (XPath) is used to uniquely identify or address parts of an XML document. An XPath expression can be used to search through an XML document, and extract information from any part of the document, such as an element or attribute (referred to as a *node* in XML) in it. XPath can be used alone or in conjunction with XSLT.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Using dynamic terminals" on page 1518
You can add, rename, and remove dynamic terminals on a node in the Message Flow editor.

"Using XPath" on page 1506
XPath provides an alternative method to ESQL for entering expressions in the property fields of specific built-in nodes.

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Using nodes for decision making" on page 2209
You can use several built-in nodes in different ways to control the path that a message takes through the message flow.

W3C XPath 1.0 Specification

RouteToLabel node

Use the RouteToLabel node in combination with one or more Label nodes to dynamically determine the route that a message takes through the message flow, based on its content.

This topic contains the following sections:

Purpose:

The RouteToLabel node interrogates the local environment of the message to determine the identifier of the Label node to which to route the message.

You must precede the RouteToLabel node in the message flow with a Compute node that populates the local environment of the message with the identifiers of one or more Label nodes that introduce the next sequence of processing for the message. The destinations are set up as a list of label names in the local environment tree in a specific location. This excerpt of ESQL from the Airline Reservations sample demonstrates how to set up the local environment content in a Compute node:

```
IF InputRoot.XMLNSC.PassengerQuery.ReservationNumber<>'' THEN
   SET OutputLocalEnvironment.Destination.RouterList.DestinationData[1].labelName = 'SinglePassenger';
ELSE
   SET OutputLocalEnvironment.Destination.RouterList.DestinationData[1].labelName = 'AllReservations';
END IF;
```

The label names can be any string value, and can be specified explicitly in the Compute node, taken or cast from any field in the message, or retrieved from a database. A label name in the local environment must match the `Label Name` property of a corresponding Label node.

When you configure the Compute node, you must also select a value for the `Compute Mode` property from the list that includes LocalEnvironment.

Design your message flow so that a RouteToLabel node logically precedes one or more Label nodes in a message flow, but do not physically wire the RouteToLabel node with a Label node. The connection is made by the broker, when required, according to the contents of the local environment.

The RouteToLabel node is contained in the **Routing** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:
Look at the following sample to see how to use this node:

- Airline Reservations

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Terminals and properties:
When you have put an instance of the RouteToLabel node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value are marked with an asterisk.

The RouteToLabel node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the message is routed if a failure is detected during processing.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The RouteToLabel node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type: RouteToLabel	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The RouteToLabel node Basic properties are described in the following table.

Property	M	C	Default	Description
Mode	Yes	No	Route To Last	This property controls how the RouteToLabel node processes the items in the local environment that is associated with the current message . Valid values are: • `Route To First`: removes the first item from the local environment. The current message is routed to the Label node that is identified by labelName in that list item. • `Route To Last` (the default): removes the last item from the local environment. The current message is routed to the Label node that is identified by labelName in that list item.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Local environment tree structure" on page 1056
The local environment tree is a part of the logical message tree in which you can store information while the message flow processes the message.

Related tasks:

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

"Accessing the local environment tree" on page 2463
The local environment tree has its own correlation name, LocalEnvironment, and you must use this name in all ESQL statements that refer to or set the content of this tree.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Using nodes for decision making" on page 2209
You can use several built-in nodes in different ways to control the path that a message takes through the message flow.

Related reference:

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"FlowOrder node" on page 4458
Use the FlowOrder node to control the order in which a message is processed by a message flow.

"Label node" on page 4569
Use the Label node to process a message that is propagated by a RouteToLabel node to dynamically determine the route that the message takes through the message flow.

"CAST function" on page 5245

"SET statement" on page 5159
The SET statement assigns a value to a variable.

SAPInput node

Use the SAPInput node to accept input from an SAP application.

This topic contains the following sections:

- "Purpose"
- "Using this node in a message flow"
- "Terminals and properties" on page 4678

Purpose:

Use the SAPInput node to accept input from SAP applications. For example, the SAPInput node might monitor an SAP system for new purchase orders. When a new purchase order is raised, the SAPInput node generates a message tree that represents the business object with the new purchase order details. The message tree is propagated to the Out terminal so that the rest of the message flow can use the data to update other systems, or to audit the changes.

The SAPInput node is contained in the **WebSphere Adapters** drawer of the message flow node palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:

The SAPInput node needs an adapter component to function correctly. You set the component by using the `Adapter component` node property and business object definitions, which are stored in the message set that you reference from the node. For this reason, you must provide a message set. By default, the message that is propagated from the SAPInput node is in the DataObject domain, so the `Message domain` property is set to DataObject. You cannot specify a different domain. The message type is detected automatically by the node.

The SAPInput node populates the route-to-label destination list with the name of the method binding. If you add a RouteToLabel node to the message flow after the SAPInput node, the RouteToLabel node can use the name of the method binding to route the message to the correct part of the message flow for processing.

You can deploy only one input node that uses a particular adapter component to an execution group, but you can deploy many input nodes that use different adapter components to an execution group.

You can use the **mqsisetdbparms** command in the following format to configure an account name with a user name and password for the Adapter for SAP Software.

```
mqsisetdbparms broker name -n adapter name -u user name -p password
```

For example:

```
mqsisetdbparms BRK1 -n eis::SAPCustomerInbound.inadapter -u sapuid -p ********
```

The SAP inbound adapter has a property called `Number of listeners`, which configures the adapter to have a particular number of threads listening to the SAP RFC program ID. These threads are not used directly to process messages in a message flow. When a message listener has an event to deliver to the message flow, it requests one of the instances of the flow. In general, it is sensible to keep the number of listeners equal to the number of instances (where instances equals 1 plus additional instances set on the flow or node). For example:

- If the number of listeners is 1, and additional instances is 0, you get a single-threaded message flow that processes one message at a time.
- If the number of listeners is 2, and additional instances is 1, you get two threads that process messages at the same time.
- If the number of listeners is 2, and additional instances is 0, you get two threads that receive data from the EIS, but only one message flow thread will ever run.

The listeners block processing until a message flow instance is available; the listeners do not queue multiple pieces of work. If you leave the number of listeners set to 1 (the default value), the broker ensures that the number of listeners is equal to the number of additional instances plus one. Additional threads can increase the throughput of a message flow, but consider the potential effect on message order.

Using configurable services for SAP nodes

SAP nodes can get SAP connection details from either the adapter component or a configurable service. By using a configurable service, you can change the connection details for an adapter without the need to redeploy the adapter. For more details about creating, changing, reporting, and deleting the configurable services for SAP, see “Changing connection details for SAP adapters” on page 719.

Generic IDoc routing

By using the SAPInput node in passthrough mode, WebSphere Message Broker can receive any IDoc and route it according to IDoc type. For more information, see “Generic IDoc routing” on page 1976. You can use a pattern to process IDocs of various kinds with a single RFC program ID without having to redeploy or rediscover existing message sets and adapters, even when adding new types of IDoc. For more information, look at the pattern: Data distribution SAP to WebSphere MQ: one-way (for IDoc). You can view patterns in the information center by using the links only when you use the information center that is integrated with the WebSphere Message Broker Toolkit, or when you use the online information center.

Look at the following samples to see how to use this node:

- SAP Connectivity
- SAP callout to a synchronous system

- SAP callout to an asynchronous system

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Terminals and properties:
When you have put an instance of the SAPInput node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view. If you double-click an SAPInput node, you open the Adapter Connection wizard. All mandatory properties for which you must enter a value (those properties that do not have a default value defined) are marked with an asterisk.

The SAPInput node terminals are described in the following table.

Terminal	Description
Out	Business object events from the adapter are sent to the Out terminal.
Failure	If an error occurs in the SAPInput node, the message is propagated to the Failure terminal. Information about the error, and business object events can also be propagated to the Failure terminal.
Catch	Business object events are sent to the Catch terminal if they cause an uncaught exception in the message flow. If the Catch terminal is not connected, the retry process is activated to handle the business object.

The following tables describe the node properties. The columns headed M indicate whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the columns headed C indicate whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The SAPInput node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, SAPInput	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The SAPInput node Basic properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Primary adapter component	Yes	Yes		The name of the adapter component that contains configuration properties for the adapter. Either enter a name of an adapter file, or click **Browse** to select an adapter file from the list of files that are available in referenced message set projects. When the SAPInput node receives data from the SAP system, it associates that data with a method name, depending on the service operation name that is assigned to that type of data when you run the Adapter Connection wizard. The SAPInput node attempts to handle methods that are defined in the primary adapter. If the type of data that is received does not correspond to any of the methods that are defined in the primary adapter, the node can handle methods that are defined in matching secondary adapters that are deployed to the same execution group.	adapterComponent
Secondary adapter mode	No	Yes	None	Specifies whether the node can handle methods that are defined in secondary adapters. If you set the `Secondary adapter mode` property to `None`, the node handles only methods that are defined in the primary adapter. If the type of data that is received does not correspond to any of the methods that are defined in the primary adapter, a failure occurs. If you set this property to `All adapters in execution group`, the node can handle methods that are defined in matching SAP inbound adapters that are deployed to the same execution group. For example, if the primary adapter is configured for IDoc messages, matching secondary adapters are IDoc inbound adapters.	secondaryAdapterMode

The SAPInput node Advanced properties are described in the following table.

Property	M	C	Default	Description
Maximum client wait time (secs)	No	Yes	60	The time (in seconds) for the SAP system to wait for a reply to be returned by an SAPReply node. The default value is 60 seconds. If a reply is sent to an SAPReply node after the timeout, the SAPReply node issues an exception. If the broker is stopped while the adapter is waiting for an SAPReply node to provide a reply, a system failure is sent back to the SAP system. If you set this property to zero (0), the SAP system waits indefinitely for a reply to be returned by an SAPReply node. This property is applicable to synchronous callback mode only; it does not apply to asynchronous BAPIs or IDOCs. If you set this property for an SAPInput node that is configured with an adapter that is not for synchronous callback, a warning is issued.

The SAPInput node Routing properties are described in the following table.

Property	M	C	Default	Description
Set destination list	No	No	Selected	Specifies whether to add the method binding name to the route-to-label destination list. If you select this check box, the method binding name is added so that you can use a RouteToLabel node in the message flow after the SAPInput node.
Label prefix	No	No		The prefix to add to the method name when routing to a label. Add a label prefix to avoid a clash of corresponding label nodes when you include multiple WebSphere Adapters input nodes in the same message flow. By default, there is no label prefix, so the method name and label name are identical.

The SAPInput node Input Message Parsing properties are described in the following table.

Property	M	C	Default	Description
Message domain	No	No	`DataObject`	The domain that is used to parse the incoming message. By default, the message that is propagated from the SAPInput node is in the DataObject domain. You cannot specify a different domain.
Message set	Yes	No	Set automatically	The name of the message set in which the incoming message is defined. This field is set automatically from the Adapter component property. If you set this property, then later update the project dependencies to remove this message set reference, a warning is issued. Either update the `Message set` property, or restore the reference to this message set project.
Message type	No	No		The name of the incoming message. The node detects the message type automatically. You cannot set this property.
Message format	No	No		The name of the physical format of the incoming message. You cannot set this property.

The SAPInput node Transactionality properties are described in the following table.

Property	M	C	Default	Description
Transaction mode	No	No	`Yes`	The transaction mode on this input node determines whether the rest of the nodes in the message flow are executed under sync point.

The Instances properties of the SAPInput node are described in the following table. For a full description of these properties, refer to "Configurable message flow properties" on page 4020. For more information about tuning the SAP adapter, see "Tuning the SAP adapter for scalability and performance" on page 3278.

Property	M	C	Default	Description	**`mqsiapplybaroverride` command property**
Additional instances pool	No	Yes	`Use Pool Associated with Message Flow`	The pool from which additional instances are obtained. • If you select `Use Pool Associated with Message Flow`, additional instances are obtained from the message flow value. • If you select `Use Pool Associated with Node`, additional instances are allocated from the additional instances of the node based on the number specified in the `Additional instances` property.	componentLevel
Additional instances	No	Yes	`0`	The number of additional instances that the node can start if the `Additional instances pool` property is set to `Use Pool Associated with Node`. By default, no additional instances are given to the node.	additionalInstances

The SAPInput node Retry properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Retry mechanism	No	No	`Failure`	Specifies how retry processing is handled when a failure is rolled back to the SAPInput node. • If you select `Failure`, retry processing is not performed so you cannot set the remaining properties. • If you select `Short and long retry`, retry processing is performed first at the interval that is specified by the `Short retry interval` property, and if that is unsuccessful, it is then performed at the interval that is specified by the `Long retry interval` property.	
Retry threshold	No	Yes	`0`	The maximum number of times that retry processing is performed for short retry.	retryThreshold
Short retry interval	No	Yes	`0`	The interval between short retry attempts.	shortRetryInterval
Long retry interval	No	Yes	`0`	The interval between long retry attempts.	longRetryInterval

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

"DataObject parser and domain" on page 1114
Use the DataObject domain to parse and write messages for WebSphere Adapters and CORBA applications.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Routing IDocs to separate message flows" on page 2055
You can use a pattern to process IDocs of various kinds with a single RFC program ID without having to redeploy or rediscover existing message sets and adapters, even when adding new types of IDoc.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

Related reference:

"WebSphere Adapter for SAP properties" on page 4024
Reference information to which to refer when you connect to an SAP application.

"SAPRequest node" on page 4685
Use the SAPRequest node to send requests to an SAP application.

"SAPReply node"
Use the SAPReply node to send a reply to an SAP synchronous callout. Use this node with an SAPInput node to implement a message flow that acts as a remote function call (RFC) destination.

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

SAPReply node

Use the SAPReply node to send a reply to an SAP synchronous callout. Use this node with an SAPInput node to implement a message flow that acts as a remote function call (RFC) destination.

This topic contains the following sections:
- "Purpose"
- "Using this node in a message flow" on page 4683
- "Terminals and properties" on page 4683

Purpose:
An SAP program can call functions on remote RFC servers. These functions can be called asynchronously or synchronously, where they must provide a reply. A message flow can act as an RFC destination and receives the function call through the SAPInput node. When the function is called synchronously, use the SAPReply node to send a reply back.

The SAPReply node is contained in the **WebSphere Adapters** drawer of the message flow node palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:
You can use an SAPReply node in the same message flow as an SAPInput node, or in a different flow from an SAPInput node. The SAPReply node must be deployed in the same execution group as the SAPInput node.

Look at the following samples to see how to use this node:

- SAP callout to a synchronous system
- SAP callout to an asynchronous system

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Terminals and properties:
When you have put an instance of the SAPReply node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view. To display the properties of the node in the Properties dialog, either double-click the node, or right-click the node and click **Properties**. All mandatory properties for which you must enter a value (those properties that do not have a default value defined) are marked with an asterisk.

The SAPReply node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the message is routed if a failure is detected when the message is propagated.
Out	The output terminal to which the message is routed if it has been sent to an external resource. The message is unchanged except for the addition of status information.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk in the Properties view) if you must enter a value when no default is defined). The column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The SAPReply node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type: SAPReply	The name of the node.
Short description	No	No	None	A brief description of the node.
Long description	No	No	None	Text that describes the purpose of the node in the message flow.

The SAPReply node Basic properties are described in the following table.

Property	M	C	Default	Description
SAP Reply Identifier	No	No	$LocalEnvironment/Destination/Adapter/Reply/ReplyIdentifier	Use this property to specify the location of the SAP reply identifier. The SAPReply node issues an exception in the following circumstances: • The SAPReply node receives a reply for a BAPI that has timed out • The SAPReply node receives a reply for a BAPI when the broker has been restarted or redeployment has occurred after the flow with the SAPInput node has finished, but before the flow with the SAPReply node has started.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Passing parameters and reporting errors" on page 1942
The BAPI interface is defined by its input parameters (IMPORT), output parameters (EXPORT), and tables.

"BAPI inbound scenarios" on page 1943
In SAP, you can call functions in other applications or SAP systems that are registered with SAP as remote function call (RFC) servers. In WebSphere Message Broker, you can register the SAP adapter with SAP as an RFC server so that it accepts synchronous and asynchronous calls from SAP.

"SAP adapter scalability and performance" on page 1949
You can improve performance by configuring the number of listeners on the adapter and the number of additional instances on the message flow to prevent delays when processing synchronous calls from SAP.

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

"DataObject parser and domain" on page 1114
Use the DataObject domain to parse and write messages for WebSphere Adapters and CORBA applications.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

"Tuning the SAP adapter for scalability and performance" on page 3278
You can configure the number of listeners on the adapter and the number of additional instances on the message flow to prevent delays when processing synchronous calls from SAP.

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

Related reference:

"SAPInput node" on page 4676
Use the SAPInput node to accept input from an SAP application.

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

SAPRequest node

Use the SAPRequest node to send requests to an SAP application.

This topic contains the following sections:

Purpose:

Use the SAPRequest node to send requests to SAP applications. For example, the SAPRequest node might request information from an SAP Enterprise Information System (EIS). A customer business object is sent to SAP, causing SAP to retrieve information about a customer, such as an address and account details. The response information that is retrieved by the SAPRequest node can then be used by the rest of the message flow. The SAPRequest node can send and receive business data.

The SAPRequest node is contained in the **WebSphere Adapters** drawer of the message flow node palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using the SAPRequest node in a message flow:

The SAPRequest node needs an adapter component to function correctly. You set the component by using the `Adapter component` node property and business object definitions, which are stored in the message set that you reference from the node.

For this reason, you must provide a message set. By default, the message that is propagated from the SAPRequest node is in the DataObject domain, so that the `Message domain` property is set to DataObject. You cannot specify a different domain. The node automatically detects the message type.

The SAPRequest node supports local transactions by using the Local Transaction Manager of the broker.

You can deploy several WebSphere Adapters request nodes that use the same adapter component to an execution group.

The SAPRequest node can use an identity that is present on an input message, and propagate it to SAP, by using the `Propagate` property on the security profile that is defined on the node. For more information, see "Propagating security credentials to an SAP request" on page 2065.

To effectively maintain the pool of connections to SAP, you can set a connection timeout value on a configurable service. By default, the connectionIdleTimeout property is set to zero, indicating that no timeout occurs. However, new connections to SAP are opened with different user IDs, therefore do not set this property to zero if you are using identity propagation. For more information, see "Configuring EIS connections to expire after a specified time" on page 726.

You can use the **mqsisetdbparms** command in the following format to configure an account name with a user name and password for the Adapter for SAP Software. The **mqsisetdbparms** command stores the password in case-sensitive form. However, when the SAP GUI sets a password, it automatically converts the password to uppercase. Therefore, specify an uppercase password to connect to the SAP system.

```
mqsisetdbparms broker name -n adapter name -u user name -p PASSWORD
```

For example:

```
mqsisetdbparms BRK1 -n eis::SAPCustomerOutbound.outadapter -u sapuid -p ********
```

Look at the following sample to see how to use this node:

- SAP Connectivity

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Using configurable services for SAP nodes

SAP nodes can get SAP connection details from either the adapter component or a configurable service. By using a configurable service, you can change the connection details for an adapter without the need to redeploy the adapter. For more details about creating, changing, reporting, and deleting the configurable services for SAP, see "Changing connection details for SAP adapters" on page 719.

Terminals and properties:
When you have put an instance of the SAPRequest node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view. If you double-click an SAPRequest node, you open the Adapter Connection

wizard. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The SAPRequest node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts the request business object.
Out	The output terminal to which the response business object is sent if it represents successful completion of the request, and if further processing is required within this message flow.
Failure	If an error occurs in the SAPRequest node, the message is propagated to the Failure terminal. Information about the error, and business object events can also be propagated to the Failure terminal.

The following tables describe the node properties. The columns headed M indicate whether the property is *mandatory* (marked with an asterisk on the panel if you must enter a value when no default is defined); the columns headed C indicate whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The SAPRequest node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, SAPRequest	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The SAPRequest node Basic properties are described in the following table.

Property	M	C	Default	Description	`mqsiapplybaroverride` command property
Primary adapter component	Yes	No		The name of the adapter component that contains configuration properties for the adapter. Either enter a name of an adapter file, or click **Browse** to select an adapter file from the list of files that are available in referenced message set projects. When the SAPRequest node receives data from the SAP system, it associates that data with a method name. The SAPRequest node attempts to call methods that are defined in the primary adapter. If the method is not defined in the primary adapter, the node can call methods that are defined in matching secondary adapters that are deployed to the same execution group.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Secondary adapter mode	No	Yes	None	Specifies whether the node can call methods that are defined in secondary adapters. If you set the `Secondary adapter mode` property to `None`, the SAPRequest node calls only methods that are defined in the primary adapter. If the method is not defined in the primary adapter, an error occurs. If you set this property to `All adapters in execution group`, the node can call methods that are defined in matching SAP outbound adapters that are deployed to the same execution group. For example, if the primary adapter is configured for IDoc messages, matching secondary adapters are IDoc outbound adapters.	secondaryAdapterMode
Default method	Yes	Yes		The default method binding to use. This property lists the methods that are defined by the adapter. You can override this property by setting the method name in the LocalEnvironment.Adapter subtree. For more information, see "Local environment tree structure" on page 1056. The method names correspond to the Service Operation names, which are configured by the Adapter Connection wizard. In most cases, the names are based on the name of the service that is being discovered (for example, a BAPI).	defaultMethod

The SAPRequest node Response Message Parsing properties are described in the following table.

Property	M	C	Default	Description
Message domain	No	No	`DataObject`	The domain that is used to parse the response message. By default, the message that is propagated from the SAPRequest node is in the DataObject domain. You cannot specify a different domain.
Message set	Yes	No	Set automatically	The name of the message set in which the incoming message is defined. This field is set automatically from the Adapter component property. If you set this property, then subsequently update the project dependencies to remove this message set reference, a warning is issued. Either update the `Message set` property, or restore the reference to this message set project.
Message type	No	No		The name of the response message. The node detects the message type automatically. You cannot set this property.
Message format	No	No		The name of the physical format of the response message. You cannot set this property.

The SAPRequest node Transactionality properties are described in the following table.

Property	M	C	Default	Description
Transaction mode	No	No	`Automatic`	Specifies how updates are handled. • If you select `Yes`, the SAPRequest node takes part in the local transaction that is started by the message flow's input node. • If you select `No`, the SAPRequest node does not take part in the local transaction that is started by the message flow's input node. • If you select `Automatic`, the SAPRequest node uses the value that is set on the input node that drives the message flow. For example, if the message flow is driven by an SAPInput node, the SAPRequest assumes the Transaction mode that is set on the SAPInput node. For more information about transactionality, see "SAP BAPI transaction commit" on page 1928.

The SAPRequest node Request properties are described in the following table.

Property	M	C	Default	Description
Method Location	Yes	No	`$LocalEnvironment/Adapter/MethodName`	The location of the business method (such as createPurchaseOrder or deletePurchaseOrder) that is used to trigger the SAPRequest node to perform an action on the external system.
Data Location	Yes	No	`$Body`	The location in the incoming message tree from which data is retrieved to form the request that is sent from the SAPRequest node to the EIS.

The SAPRequest node Result properties are described in the following table.

Property	M	C	Default	Description
Output data location	No	No	`$OutputRoot`	The message tree location to which the SAPRequest node sends output.
Copy local environment	No	No	Selected	This property controls how the local environment is copied to the output message. If you select this check box, a new copy of the local environment is created in the tree (at each node in the message flow), and it is populated with the contents of the local environment from the preceding node. Therefore, if a node changes the local environment, the previous nodes in the flow do not see those changes because they have their own copies. This behavior might be an issue if you are using a FlowOrder node, or if you use the propagate command on a Compute node. If you clear the check box, each node does not generate its own copy of the local environment, but it uses the local environment that is passed to it by the previous node. Therefore, if a node changes the local environment, those changes are seen by the upstream nodes.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Overview of WebSphere Adapter for SAP Software" on page 1917
With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

"SAP BAPI transaction commit" on page 1928
When the SAP adapter is used with the BAPI interface, you must consider certain factors when you design transactional flows.

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"DataObject parser and domain" on page 1114
Use the DataObject domain to parse and write messages for WebSphere Adapters and CORBA applications.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Adding external software dependencies for SAP" on page 2048
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

"Configuring the SAP server to work with the adapter" on page 2050
Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

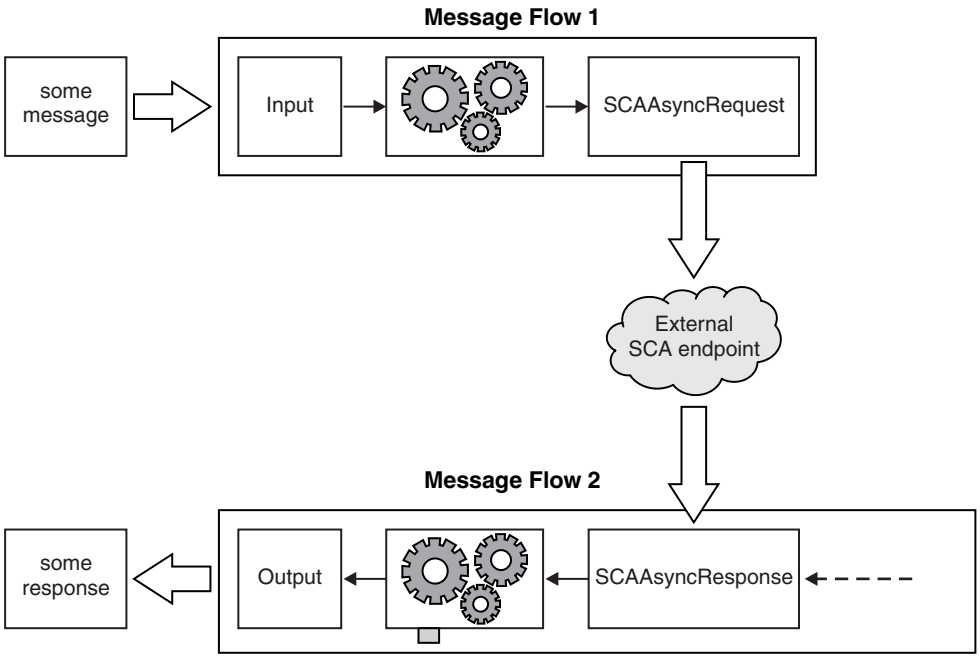
"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

SCAAsyncRequest node

Use the SCAAsyncRequest node with the SCAAsyncResponse node to construct a pair of message flows that invoke a WebSphere Process Server service component asynchronously.

This topic contains the following sections:

Purpose:
The SCAAsyncRequest node sends a request to a business process that is running on WebSphere Process Server, but the node does not wait for the associated response to be received. However, the SCAAsyncRequest node does wait for an acknowledgment before continuing with the message flow. The response to the Process Server request is received by the SCAAsyncResponse node, which can be in the same message flow or in a separate message flow. The nodes are used as a pair, and correlate responses against the original requests.



The SCAAsyncRequest node is the first half of the asynchronous request and response node pair. The SCAAsyncRequest node sends a request to a business process that is running on WebSphere Process Server. The request is sent by the SCAAsyncRequest node, but the SCAAsyncRequest node does not receive the response. The response is received by a SCAAsyncResponse node that is running on a different thread. The SCAAsyncResponse node is typically at the beginning of a different message flow; however, it must be in the same execution group as the SCAAsyncRequest node.

A Broker SCA definition is required to configure both the SCAAsyncRequest node and the SCAAsyncResponse node. The Broker SCA definition contains specific data related to the Process Server binding.

The SCAAsyncRequest node is contained in the **SCA** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:
Look at the following sample to see how to use the node:

SCA nodes

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

You can change the operation that should be invoked by changing the value in the following location in the local environment:

`LocalEnvironment.Destination.SCA.Request.Operation`

For a Web Services binding, you can change the URL to which the request is sent. Change the value in the following location in the local environment:

`LocalEnvironment.Destination.SCA.Request.Binding.WebServices.Transport.HTTP.WebServiceURL`

For an MQ binding, you can change the Request Queue Manager and Request Queue by changing the values in the following locations:

`LocalEnvironment.Destination.SCA.Request.Binding.MQ.queueManagerName`
`LocalEnvironment.Destination.SCA.Request.Binding.MQ.queueName`

You can store context data in the following location in the local environment. The SCAAysncResponse node can later retrieve this data.

`LocalEnvironment.Destination.SCA.Request.UserContext`

`LocalEnvironment.Destination.SCA.Request.KeyAlias`

Configuring the SCAAsyncRequest node:
Ensure that the message set contains a Broker SCA definition with an extension of .outsca with which to configure the SCAAsyncRequest node.

There are two methods of putting an instance of the SCAAsyncRequest node into a message flow: you can either drag an instance of the node from the node palette, or drag a Broker SCA definition with an extension of .outsca from a message set onto the message flow editor canvas.

`Dragging a node onto the canvas`

If you have dragged an instance of the node from the palette onto the canvas, use one of the following methods to start to configure it:

- Drag a Broker SCA definition with an extension of `.outsca` onto the node.
- Type the file name in the `SCA file name` option in the Properties view of the node.
- Browse to the file (`SCA file name` option in the Properties view of the node.)

`Dragging a .outsca file onto the canvas`

- If the `.outsca` file contains a WSDL that has only a single, one-way operation defined, a SCARequest node is created.
- If the `.outsca` file contains a WSDL that only has request-response operations, you first select the operation, and say whether you want the outbound request to be sent synchronously or asynchronously. For a

synchronous request, a SCARequest node is created and configured. For an asynchronous request, a pair of SCAAsyncRequest and SCAAsyncResponse nodes are created and configured.

- If the `.outsca` file contains a WSDL that has a mixture of one-way and request-response operations defined, and you select a one-way operation, a SCARequest node is created.

The values for many of the node properties are provided in the Broker SCA definition. The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (properties that do not have a default value defined) are marked with an asterisk.

1. Optional: On the **Description** tab, enter a `Short description`, a `Long description`, or both. You can also rename the node on this tab.
2. On the **Basic** tab, set the `Unique identifier` and `Broker SCA definition` properties.
   - `Unique identifier`. You must specify the unique string that is common to your pair of SCAAsyncRequest and SCAAsyncResponse nodes. This property is mandatory.
   - In `Broker SCA definition`, specify the name of the Broker SCA definition that contains configuration properties for the SCAAsyncRequest node. If you have created the node by dragging a Broker SCA definition from a message set onto the Message flow editor canvas, this property is preset to the name of the Broker SCA definition. If you created the node by selecting it from the palette, you can set this property in one of the following ways:
     - If you have a Broker SCA definition, you can select it from the Broker SCA definitions by clicking **Browse**.
     - Type a file name that is relative to the message set project in which the Broker SCA definition exists.
     - If you have Broker SCA definitions, but no message set, you can create a message set:
       a. Click **Browse** to open the Broker SCA Definition Selection pane.
       b. Click **Import/Create New** to open the Import Broker SCA definition wizard.
       c. Enter the message set name and message set project name, then click **Next**.
       d. Choose the relevant option:
          - If your Broker SCA definition exists in your workspace, click **Use resources from the workspace**, and select the Broker SCA definition.
          - If your Broker SCA definition is in the file system, click **Use external resources**, select the Broker SCA definition, click **Next**.
       e. Select the Broker SCA definition to import.
       f. Click **Finish**. A new message set project and message set are created with message definitions. The Broker SCA definition is added to the `Broker SCA Definitions` folder.
       g. Select the Broker SCA definition from the Broker SCA Definition Selection window, then click **OK**.
     - If you have a message set but no Broker SCA definition, generate a Broker SCA definition by following the instructions in "Generating a Broker SCA definition from a message set" on page 2967.
     - Drag a Broker SCA definition from a message set onto the node.

3. On the **Binding** tab, specify properties that relate to the WebSphere Process Server binding. Some of the properties on this tab are derived from the Broker SCA definition.
   - The value of the Binding type property is derived from the binding information in the Broker SCA definition. It is read-only. Possible values are:
     - `WebService`. Requests from WebSphere Process Server are sent as SOAP messages over the HTTP transport. See the WSDL properties table in "Terminals and properties" for details of binding properties that are either supplied by the Broker SCA definition, or that you need to set.
     - `MQ`. Requests from WebSphere Process Server arrive as WebSphere MQ messages. See the MQ properties table and the Transactionality table in "Terminals and properties," for details of binding properties that are either supplied by the Broker SCA definition, or that you need to set.

Terminals and properties:
The SCAAsyncRequest node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message.
Failure	The output terminal to which the message is sent if a failure is detected in the node.
Out	The output terminal to which the message is sent after the node has successfully sent the message to the SCA resource.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Description properties of the SCAAsyncRequest node are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type	The name of the node.
Short description	No	No	None	A brief description of the node.
Long description	No	No	None	Text that describes the purpose of the node in the message flow.

The Basic properties of the SCAAsyncRequest node are described in the following table.

Property	M	C	Default	Description
Unique identifier	Yes	Yes	Not set	The property specifies a unique identifier that is common to your pair of SCAAsyncRequest and SCAAsyncResponse nodes.

Property	M	C	Default	Description
Broker SCA definition	Yes	Yes	Not set	The property specifies the name of the Broker SCA definition that contains configuration properties for the SCAAsyncRequest node. You can click Browse to see a list of all relevant Broker SCA definitions in the current workspace.

The Binding property of the SCAAsyncRequest node is described in the following table.

Property	M	C	Default	Description
Binding type	Yes	No	From SCA export	This property is read-only. It is supplied by the toolkit when the SCA export configuration file is parsed. The property describes the binding type that was found in the Broker SCA definition. If the binding is Web Services, only the WSDL properties are shown. If the binding is MQ, only the MQ properties are shown.

The WSDL properties are described in the following table. These properties are shown only if the binding type is Web Services.

Note: The values for the properties must be supplied on the node. They are not marked as mandatory on the node because the Broker SCA definition might not contain all the property values, resulting in node errors being shown. You must supply values for those properties that have not been configured by the Broker SCA definition file.

Property	M	C	Default	Description
WSDL file name	Yes	No	Not set	The WSDL file name is taken from the WSDL file that is referenced by the Broker SCA definition.
Target namespace	No	No	From Broker SCA definition	This property type is String. It is supplied by the toolkit when the WSDL configuration file that is referenced by the Broker SCA definition is parsed.
Port type	Yes	No	By default, the `Port type` that is referenced in the Broker SCA definition.	This property type is String. This property is read-only. Error Conditions: • The selected `Port type` does not contain at least one operation.
Imported binding	Yes	No	From Broker SCA definition	This property type is String. It is supplied by the toolkit when the WSDL that is referenced by the Broker SCA definition is parsed. Error Conditions: • No SOAP bindings (with HTTP transport) in the WSDL file are associated with the `Port type`. • The selected binding does not have any operations.
Operation	Yes	No	From Broker SCA definition	This property type is String. The `Operation` property lists the operations that are defined by the port type. The first two-way operation in the list is selected by default. Only two-way operations are supported.

Property	M	C	Default	Description
Service port	Yes	No	From Broker SCA definition	This property type is String. It is supplied by the toolkit when the WSDL configuration file is parsed. Error Conditions: • No ports point to the selected binding.
Web service URL	Yes	Yes	From Broker SCA definition	This property type is String. This property is automatically derived from the `<soap:address>` element of the selected `Service port`. Whenever the selected port is updated, the `Web service URL` is updated accordingly. However, if you override the value, your value persists and the URL is no longer updated from the service port. If you choose to override this property you must specify it in the form http://<hostname>[:<port>]/[<path>] where: • http://<hostname> must be specified. • <port> has a default of 80. If you specify a value, you must include the : before the port number. • <path> has a default of /. If you specify a value, you must include the / before the path.

The MQ properties are described in the following table. These properties are shown only if the binding type is MQ.

Note: The values for the properties must be supplied on the node. They are not marked as mandatory on the node because the Broker SCA definition might not contain all the property values, resulting in node errors being shown. You must provide values for those properties that have not been configured by the Broker SCA definition file.

Property	M	C	Default	Description
Operation	Yes	No	From the Broker SCA definition	The available operations that are found in the interface of the Broker SCA definition. Only two-way operations are supported.
Queue name	Yes	Yes	From the Broker SCA definition	The name of the queue that receives a request message from WebSphere Message Broker. This is taken from the Broker SCA definition, but can be updated directly in the node. This can be a remote queue; see "How does distributed queuing work?" in the *Intercommunication* section of the WebSphere MQ information center.
Queue manager name	Yes	Yes	From the Broker SCA definition	The name of the queue manager that receives a message from WebSphere Message Broker. It is taken from the Broker SCA definition, but can be updated directly in the node. If a remote queue is used, leave this property blank to allow WebSphere MQ to resolve the queue manager name.
Reply-to queue name	Yes	Yes	From the Broker SCA definition	The name of the queue that receives a response message from WebSphere Process Server. It is taken from the Broker SCA definition, but can be updated directly in the node.

Property	M	C	Default	Description
Reply-to queue manager name	No	Yes	From the Broker SCA definition	The name of the queue manager that receives a response from WebSphere Process Server. It is taken from the Broker SCA definition, but can be updated directly in the node. This queue manager must be local to the broker.
Response message correlation	Yes	No	From the Broker SCA definition	This property indicates how WebSphere Process Server provides correlation ID information in the response message. This correlation information is used by the WebSphere Message Broker to determine which message is a response to which request, and is supplied by the toolkit when the SCA export file is parsed. Select From Message ID if the WebSphere Process Server is expected to copy the MQMD MSGID field in the request to the MQMD CORRELID field in the response. Select From Correl ID if the WebSphere Process Server is expected to copy the MQMD CORRELID field in the request to the MQMD CORRELID field in the response.

The Transactionality property of the SCAAsyncRequest node is described in the following table.

Property	M	C	Default	Description
Transaction mode	No	No	For MQ, 'yes'	This property can be 'automatic', 'no', or 'yes'. The property is enabled only when the Binding type is MQ.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"SOAP parser and domain" on page 1082
You can use the SOAP parser to create a common WSDL-based logical tree format for working with Web services, independent of the physical bitstream format.

"SOAP tree overview" on page 1611
This tree format allows you to access the key parts of the SOAP message in a

convenient way.

"XMLNSC parser" on page 1090
The XMLNSC parser is a flexible, general-purpose XML parser that offers high performance XML parsing and optional XML Schema validation.

"Working with Service Component Architecture (SCA)" on page 2095
Start here to find out how you can use SCA to allow interoperability with WebSphere Process Server Version 6.2.

"Samples" on page 98
The WebSphere Message Broker Toolkit provides samples that show the features that are available in WebSphere Message Broker, and how to use them. This topic provides links to the information about the individual samples.

Related tasks:

"Validating messages" on page 1478
The broker provides validation based on the message model for predefined messages.

"Manipulating messages in the XMLNSC domain" on page 2546
If you are writing ESQL to process messages in the XMLNSC domain, it is helpful to learn about the structure of the message tree that the XMLNSC parser builds.

"Importing from WSDL" on page 2946
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to create a new message definition from WSDL.

Related reference:

"Parsing on demand" on page 4173
On-demand parsing, referred to as partial parsing, is used to parse an input message bit stream only as far as is necessary to satisfy the current reference. The parsers that can perform partial parsing of input messages are the JSON, MRM, XML, XMLNS, and XMLNSC parsers.

"Validation properties" on page 4169
You can control validation by setting properties on the Validate and Parser Options tabs for the nodes that are listed in the following table.

"SCAAsyncResponse node"
Use the SCAAsyncResponse node with the SCAAsyncRequest node to construct a pair of message flows that start a component asynchronously.

"SCARequest node" on page 4719
Use the SCARequest node to send a request to WebSphere Process Server. The node is a synchronous request and response node, and blocks after sending the request until the response is received. The node can also send one-way requests.

"SCAInput node" on page 4707
Use the SCAInput node with the SCAReply node to process messages from WebSphere Process Server.

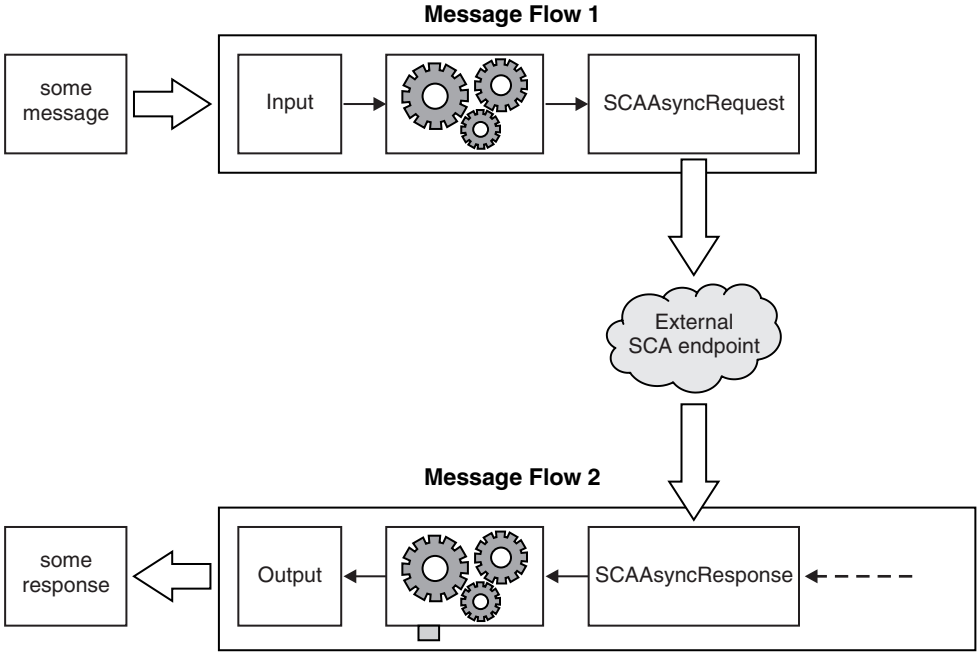
"SCAReply node" on page 4726
Use the SCAReply node to send a message from the broker to the originating client in response to a message received by a SCAInput node.

SCAAsyncResponse node

Use the SCAAsyncResponse node with the SCAAsyncRequest node to construct a pair of message flows that start a component asynchronously.

This topic contains the following sections:

Purpose:
The node allows the broker to receive the response to a previous asynchronous request made from an SCAAsyncRequest node.



The node is contained in the **SCA** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:
Look at the following sample to see how to use the node:

SCA nodes

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

You can retrieve context data that has been stored by the SCAAsyncRequest node from the following location in the local environment:

```
LocalEnvironment.SCA.Response.UserContext
```

You can access the SOAP header and context information in an inbound response in the local environment, at the following locations:

```
LocalEnvironment.SCA.Response.Binding.WebServices.SOAP.Header
LocalEnvironment.SCA.Response.Binding.WebServices.SOAP.Context
```

Configuring the node:
Ensure that the message set contains a Broker SCA definition with an extension of .outsca with which to configure the node.

There are two methods of putting an instance of the node into a message flow: You can either drag an instance of the node from the node palette, or drag a Broker SCA definition with an extension of .outsca from a message set, onto the message flow editor canvas. Dragging a Broker SCA definition with an extension of .outsca onto the canvas creates a pair of SCAAsyncRequest and SCAAsyncResponse nodes.

If you have dragged an instance of the node from the palette onto the canvas, you must then start to configure it by dragging a Broker SCA definition with an extension of .outsca onto the node. The values for many of the node properties are provided in the Broker SCA definition. If you have dragged a Broker SCA definition onto the canvas and created a pair of SCAAsyncRequest and SCAAsyncResponse nodes, many of the values for the node properties have already been supplied from the Broker SCA definition.

The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (properties that do not have a default value defined) are marked with an asterisk.

1. Optional: On the **Description** tab, enter a `Short description`, a `Long description`, or both. You can also rename the node on this tab.
2. On the **Basic** tab, set the `Unique identifier` and `Broker SCA definition` properties.
   - `Unique identifier`. You must specify the unique URL fragment that is common to your pair of SCAAsyncRequest and SCAAsyncResponse nodes. This property is mandatory.
   - In `Broker SCA definition`, specify the name of the Broker SCA definition that contains configuration properties for the node. If you have created the node by dragging a Broker SCA definition from a message set onto the Message flow editor canvas, this property is preset to the name of the Broker SCA definition. If you created the node by selecting it from the palette, you can set this property in one of the following ways:
     – If you have a Broker SCA definition, you can select it from the Broker SCA definitions by clicking **Browse**.
     – If you have Broker SCA definitions, but no message set, you can create a message set:
       a. Click **Browse** to open the Broker SCA Definition Selection pane.
       b. Click **Import/Create New** to open the Import Broker SCA definition wizard.
       c. Enter the message set name and message set project name, then click **Next**.
       d. Choose the relevant option:
          - If your Broker SCA definition exists in your workspace, click **Use resources from the workspace**, and select the Broker SCA definition.
          - If your Broker SCA definition is in the file system, click **Use external resources**, select the Broker SCA definition, then click **Next**.
       e. Select the Broker SCA definition to import.
       f. Click **Finish**. A new message set project and message set are created with message definitions. The Broker SCA definition is added to the `Broker SCA Definitions` folder.
       g. Select the Broker SCA definition from the Broker SCA Definition Selection window, then click **OK**.

- If you have a message set but no Broker SCA definition, generate a Broker SCA definition by following the instructions in "Generating a Broker SCA definition from a message set" on page 2967.
- Drag a Broker SCA definition from a message set onto the node.
- Type a file name that is relative to the message set project in which the Broker SCA definition exists.

3. On the **Binding** tab, specify properties that relate to the binding. Some of the properties on this tab are derived from the Broker SCA definition.

   The value of the Binding type property is derived from the binding information in the Broker SCA definition, and is read-only. Possible values are:

   - `WebService`. Web service responses are sent as SOAP messages over the HTTP transport.
   - `MQ`. MQ responses arrive as messages.

   The `Propagate only SOAP body (owned by XMLNSC domain)` check box is shown only when the Binding type is Web Services. It is not shown when the Binding type is MQ; there are no MQ-specific binding properties.
4. On the **Response Message Parsing** tab, the properties are set automatically from the Broker SCA Definition file.
   - If the Binding type is Web Services, the Message domain is always SOAP. If the Binding type is MQ, you can change the domain to MRM, XMLNSC, XMLNS, MIME, JSON or BLOB.
   - If the Binding type is MQ, the Message domain defaults to XMLNSC if the data bindings for all operations are using XML. Otherwise the default domain is BLOB.
5. On the **Parser Options** sub tab, set properties that are associated with the parser.
   - `Parse timing` is, by default, set to `On Demand`, which causes parsing of the message to be delayed. For information about how to cause the message to be parsed immediately, see "Parsing on demand" on page 4173.
   - `XMLNSC Parser Options`. Set values for the properties that determine how the XMLNSC parser operates. For more information, see "Manipulating messages in the XMLNSC domain" on page 2546.
6. Use the **Validation** tab to provide validation based on the message set for predefined messages. For more information about validation, see "Validating messages" on page 1478. For information about how to complete this tab, see "Validation tab properties" on page 4169.
7. Use the **Instances** tab to specify how additional threads are handled for the message flow.
   - The `Additional instances pool` property specifies whether additional instance threads are allocated from a thread pool for the whole message flow, or from a thread pool for use by that node only. By default, this property is set to `Use Pool Associated with Message Flow`.
   - The `Additional instances` property specifies the number of additional threads that the broker can use to service the message flow and has the default value `0`.

Terminals and properties:
The terminals of the node are described in the following table.

Terminal	Description
Failure	The output terminal to which the message is routed if a failure is detected when the message is propagated.
Out	The output terminal to which the message is routed if it has been propagated successfully, and if further processing is required within this message flow.
Fault	The output terminal to which a SOAP fault message is routed if the Binding type is Web Services. The Fault terminal is not used by any other type of Binding type.
Catch	The output terminal to which the message is routed if an exception is thrown downstream and caught by this node.

When you have put an instance of the node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (properties that do not have a default value defined) are marked with an asterisk.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Description properties of the node are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type: SOAPAsyncResponse	The name of the node.
Short description	No	No	None	A brief description of the node.
Long description	No	No	None	Text that describes the purpose of the node in the message flow.

The Basic properties of the node are described in the following table:

Property	M	C	Default	Description
Unique identifier	Yes	No	Not set	Specify the unique identifier that is common to your pair of nodes.
Broker SCA definition	Yes	Yes	Not set	The property specifies the name of the Broker SCA definition that contains configuration properties for the node. You can click Browse to see a list of all relevant Broker SCA definitions in the current workspace.

.

The Binding properties of the node are described in the following table:

Property	M	C	Default	Description
Binding type	Yes	No	Not set	The binding type that was found in the SCA Import.
extractSOAPBody	No	No	Cleared	This option is available if the binding is `Web Services`. If the check box is selected, only the SOAP body is propagated. If it is cleared, the entire SOAP message is propagated.

The Response Message Parsing properties of the node are described in the following table. The node sets these properties automatically; the table describes when you can change them.

Property	M	C	Default	Description
Message domain	No	No	Set automatically according to the binding that is defined in the corresponding node.	The domain that is used to parse the response message. It is determined according to the Binding type. You can change this property if the Binding type is MQ. The property is read-only when the Binding type is Web Services.
Message set	No	No	Picked automatically according to the Broker SCA definition that is chosen in the corresponding node.	The name of the message set in which the response message is defined. `Message set` is set automatically to the message set that contains the SCA file that is configured on the corresponding node. This property is read-only.
Message type	No	No	Not set	The node detects the message type automatically. You can change this property if the Binding type is MQ and the message domain is MRM. You cannot change this property if the Binding type is Web Services.
Message format	No	No	Not set	The name of the physical format of the response message. You can change this property if the Binding type is MQ and the message domain is MRM. You cannot change this property if the Binding type is Web Services.

The Parser Options properties of the node are described in the following table.

Property	M	C	Default	Description
Parse timing	Yes	No	`On Demand`	This property controls when a response message is parsed. Valid values are `On Demand`, `Immediate`, and `Complete`. By default, parse timing is set to `On demand`, which causes parsing of the input message to be delayed. For a full description of this property, see "Parsing on demand" on page 4173.
Build tree using XML schema data types	No	No	Selected	This property controls whether the XMLNSC parser creates syntax elements in the message tree with data types taken from the XML Schema.
Retain mixed content	Yes	No	Cleared	This property controls whether the parser creates elements in the message tree when it encounters mixed text in a response message. If you select the check box, elements are created for mixed text. If the check box is cleared, mixed text is ignored and no elements are created.
Retain comments	Yes	No	Cleared	This property controls whether the parser creates elements in the message tree when it encounters comments in a response message. If you select the check box, elements are created for comments. If the check box is cleared, comments are ignored and no elements are created.
Retain processing instructions	Yes	No	Cleared	This property controls whether the parser creates elements in the message tree when it encounters processing instructions in a response message. If you select the check box, elements are created for processing instructions. If the check box is cleared, processing instructions are ignored and no elements are created.

Property	M	C	Default	Description
Opaque elements	No	No	Not set	This property is used to specify a list of elements in the response message that are to be opaquely parsed. Opaque parsing is performed only if validation is not enabled (that is, if `Validate` is `None`); entries that are specified in `Opaque Elements` are ignored if validation is enabled.

The Validation properties of the node are described in the following table.

If validation fails, the message is propagated to the failure terminal, if this terminal is wired. For more details, see "Validating messages" on page 1478 and "Validation properties" on page 4169.

Property	M	C	Default	Description
Validate	Yes	Yes	`Content and value`	This property controls whether validation takes place. Valid values are `None`, `Content and value`, and `Content`.
Failure action	Yes	Yes	`Exception`	This property controls what happens if validation fails. You can set this property only if you set `Validate` to `Content` or `Content and value`. Valid values are `User trace`, `Exception list`, `Local error log`, and `Exception`.

The Security properties of the node are described in the following table. Set values for these properties to control the extraction of an identity from a message (when a security profile is associated with the node). For more information about these properties, see "Identity" on page 390, "Configuring the extraction of an identity or security token" on page 447, "Message flow security overview" on page 383, and "Setting up message flow security" on page 431.

Property	M	C	Default	Description
Identity token type	No	No	`None`	This property specifies the type of identity token present in the incoming message. Valid values are: `Transport Default`, `Username`, `Username + Password`, `SAML Assertion`, and `X.509 Certificate`. If this property is not specified, the identity is retrieved from the Basic-Auth transport header and the type is set to `Username + Password`.
Identity token location	No	No	`None`	This property specifies where, in the message, the identity can be found. The location is specified as an ESQL field reference, an XPath expression, or a string literal. If you use a string literal, it must be enclosed in single quotation marks and must not contain a period (.), If this property is not specified, the identity is retrieved from the MQMD.UserIdentifier transport header.
Identity password location	No	No	`None`	This property specifies where, in the message, the password can be found. The location is specified as an ESQL field reference, an XPath expression, or a string literal. If you use a string literal, it must be enclosed in single quotation marks and must not contain a period (.), If it is not specified, the password is not set. This property can be set only if `Identity token type` is set to `Username + Password`.
Identity IssuedBy location	No	No	`None`	This property specifies a string or path expression that describes the issuer of the identity. The location is specified as an ESQL field reference, an XPath expression, or a string literal. If you use a string literal, it must be enclosed in single quotation marks and must not contain a period (.), The value specifies the Issuer that is passed to a WS-Trust v1.3 STS provider. If this property is not specified, the MQMD.PutApplName value is used. If you leave the `Identity issuedBy location` field blank and the MQMD.PutApplName is also blank, the string `MQ` is used.

Property	M	C	Default	Description
Treat security exceptions as normal exceptions	No	No	`False`	This property specifies whether to treat security exceptions (such as "Access Denied") as normal exceptions, and propagate them to the Failure terminal (if wired). This property is turned off by default, which ensures that security exceptions cause the message to be backed out even if the Failure terminal is wired.

The Instances properties of the node are described in the following table.

Property	M	C	Default	Description
Additional instances pool	No	Yes	`Flow`	The pool from which additional instances are obtained. • If you select `Flow`, additional instances are obtained from the message flow value. • If you select `Node`, additional instances are allocated from the additional instances pool for that node; how many are allocated is specified in the `Additional instances` property.
Additional instances	No	Yes	`0`	The number of additional instances that the node can start if the `Additional instances pool` property is set to `Node`. By default, no additional instances are given to the node.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see “Configuring monitoring event sources using monitoring properties” on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

“Message modeling” on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

“Parsers” on page 1072
A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from the internal message tree representation.

“SOAP parser and domain” on page 1082
You can use the SOAP parser to create a common WSDL-based logical tree format for working with Web services, independent of the physical bitstream format.

“XMLNSC parser” on page 1090
The XMLNSC parser is a flexible, general-purpose XML parser that offers high performance XML parsing and optional XML Schema validation.

“Working with Service Component Architecture (SCA)” on page 2095
Start here to find out how you can use SCA to allow interoperability with WebSphere Process Server Version 6.2.

“SOAP parser and domain” on page 1082
You can use the SOAP parser to create a common WSDL-based logical tree format

for working with Web services, independent of the physical bitstream format.

"Samples" on page 98
The WebSphere Message Broker Toolkit provides samples that show the features that are available in WebSphere Message Broker, and how to use them. This topic provides links to the information about the individual samples.

Related tasks:

"Manipulating messages in the XMLNSC domain" on page 2546
If you are writing ESQL to process messages in the XMLNSC domain, it is helpful to learn about the structure of the message tree that the XMLNSC parser builds.

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

Related reference:

"Parsing on demand" on page 4173
On-demand parsing, referred to as partial parsing, is used to parse an input message bit stream only as far as is necessary to satisfy the current reference. The parsers that can perform partial parsing of input messages are the JSON, MRM, XML, XMLNS, and XMLNSC parsers.

"SCAAsyncRequest node" on page 4690
Use the SCAAsyncRequest node with the SCAAsyncResponse node to construct a pair of message flows that invoke a WebSphere Process Server service component asynchronously.

"SCAInput node" on page 4707
Use the SCAInput node with the SCAReply node to process messages from WebSphere Process Server.

"SCAReply node" on page 4726
Use the SCAReply node to send a message from the broker to the originating client in response to a message received by a SCAInput node.

"SCARequest node" on page 4719
Use the SCARequest node to send a request to WebSphere Process Server. The node is a synchronous request and response node, and blocks after sending the request until the response is received. The node can also send one-way requests.

"SecurityPEP node" on page 4729
Use the SecurityPEP node in a message flow to invoke the message flow security manager at any point in the message flow.

SCADAInput node

The SCADAInput node, available in earlier versions of WebSphere Message Broker, is not supported in WebSphere Message Broker Version 7.0. See "Migrating from Version 6.1 products" on page 163 for information about migrating your message flows from WebSphere Message Broker to WebSphere Message Broker Version 7.0.

SCADAOutput node

The SCADAOutput node, available in earlier versions of WebSphere Message Broker, is not supported in WebSphere Message Broker Version 7.0. See "Migrating from Version 6.1 products" on page 163 for information about migrating your message flows from WebSphere Message Broker to WebSphere Message Broker Version 7.0.

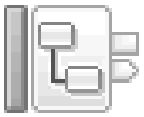
SCAInput node

Use the SCAInput node with the SCAReply node to process messages from WebSphere Process Server.

This topic contains the following sections:

Purpose:
If you use service components in WebSphere Process Server, you can use this node to start a service provided by WebSphere Message Broker. The flow acts as a SCA endpoint for the service component to use through an import binding.

The SCAInput node is contained in the **SCA** drawer of the message flow node palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:



Using this node in a message flow:
Look at the following sample to see how to use the node:

- SCA nodes

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

You can access the operation that is being called on an inbound request in the following location in the local environment:

```
LocalEnvironment.SCA.Input.Operation
```

You can access the SOAP Header information that is present in an inbound request from the following location in the local environment:

```
LocalEnvironment.SCA.Input.Binding.WebServices.SOAP.Header
```

You can also access the SOAP context information that is present in an inbound request in the local environment. The SOAP context is stored in the following location:

```
LocalEnvironment.SCA.Input.Binding.WebServices.SOAP.Context
```

Configuring the SCAInput node:
Ensure that the message set contains a Broker SCA definition with an extension of `.insca` with which to configure the SCAInput node.

You can put an instance of the SCAInput node into a message flow in either of the following ways:

- Drag a Broker SCA definition with an extension of `.insca` from a message set onto the message flow editor canvas. If the `.insca` file contains only one-way operations, dragging a `.insca` file onto the canvas creates a SCAInput node.

Otherwise, a pair of SCAInput and SCAReply nodes is created. If you use this method, many of the values for the properties of the node or nodes are supplied by the Broker SCA definition.

- Drag an instance of the node from the node palette onto the canvas. You then configure the node by dragging a Broker SCA definition with an extension of `.insca` onto the node.

The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (properties that do not have a default value defined) are marked with an asterisk.

1. Optional: On the **Description** tab, enter a `Short description`, a `Long description`, or both. You can also rename the node on this tab.
2. On the **Basic** tab, set the `Broker SCA definition` and `Message routing` properties.
   - In `Broker SCA definition`, specify the name of the Broker SCA definition that contains configuration properties for the SCAInput node. If you have created the node by dragging a Broker SCA definition from a message set onto the Message flow editor canvas, this property is preset to the name of the Broker SCA definition. If you created the node by selecting it from the palette, you can set this property in one of the following ways:
     – If you have a Broker SCA definition, you can select it from the Broker SCA definitions by clicking **Browse**.
     – If you have Broker SCA definitions, but no message set, you can create a message set:
       a. Click **Browse** to open the Broker SCA Definition Selection pane.
       b. Click **Import/Create New** to open the Import Broker SCA definition wizard.
       c. Enter the message set name and message set project name, then click **Next**.
       d. Choose the relevant option:
          - If your Broker SCA definition exists in your workspace, click **Use resources from the workspace**, and select the Broker SCA definition.
          - If your Broker SCA definition is in the file system, click **Use external resources**, select the Broker SCA definition, then click **Next**.
       e. Select the Broker SCA definition to import.
       f. Click **Finish**. A new message set project and message set are created with message definitions. The Broker SCA definition is added to the `Broker SCA Definitions` folder.
       g. Select the Broker SCA definition from the Broker SCA Definition Selection window, then click **OK**.
     – If you have a message set but no Broker SCA definition, generate a Broker SCA definition by following the instructions in “Generating a Broker SCA definition from a message set” on page 2967.
     – Drag a Broker SCA definition from a message set onto the node.
     – Type a file name that is relative to the message set project in which the Broker SCA definition exists.
   - Use `Message routing` to specify whether to allow message propagation to dynamic terminals for the operations defined in the SCA message or to the common Out terminal. By default, each operation is routed to its own terminal. You can select:

- `Route each operation to its own terminal`. The default. One terminal is generated on the node for each operation defined in the interface of the Broker SCA definition and messages are propagated to the appropriate terminal.
- `Route to a common out terminal`. One Out terminal is generated on the node. All messages are propagated to this terminal. If you select this option, the `Propagate only SOAP body (owned by XMLNSC domain)` check box is disabled and you cannot select it.

If you specify the `Broker SCA definition` property by typing a file name, or by selecting a Broker SCA definition by using the **Browse** facility, on an SCAInput node that already has dynamic terminals configured on it, the existing dynamic terminals are replaced. If you specify a file name that is not valid, previously existing dynamic terminals are removed and no terminals are added.

If you drag a Broker SCA definition onto an SCAInput node that already has dynamic terminals configured on it, a dialog box is displayed allowing you to confirm whether you want to override the existing dynamic terminals with dynamic terminals that might be generated from the Broker SCA definition. If you choose to override existing dynamic terminals, any existing connections that are configured for these terminals are also deleted. You can specify that this dialog box is not displayed again.

If you drag a Broker SCA definition onto an SCAInput node which already has an Out terminal, no dynamic terminals which be generated from the Broker SCA definition are added and the node continues to operate with its Out terminal.

3. On the **Binding** tab, properties that relate to the WebSphere Process Server binding are specified. Some of the properties on this tab are derived from the Broker SCA definition.
   - The value of the Binding type property is derived from the binding information in the Broker SCA definition. It is read-only. Possible values are:
     - `Web Service`. Requests from WebSphere Process Server are sent as SOAP messages over the HTTP transport. See the WSDL properties table in "Terminals and properties" on page 4710 for details of binding properties that are either supplied by the Broker SCA definition, or that you must set.
     - `MQ`. Requests from WebSphere Process Server arrive as WebSphere MQ messages. See the MQ properties table and the Transactionality table in "Terminals and properties" on page 4710, for details of binding properties that are either supplied by the Broker SCA definition, or that you must set.
4. On the **Input Message Parsing** tab, the properties are set automatically when the WSDL file is configured.
   - If the Binding type is Web Services, the Message domain is always SOAP.
   - If the Binding type is MQ, the Message domain defaults to XMLNSC if the data bindings for all operations are using XML. Otherwise the default domain is BLOB. You can change the domain to MRM, XMLNSC, XMLNS, MIME, JSON, or BLOB.
5. On the **Parser Options** sub tab, set properties that are associated with the parser.
   - `Parse timing` is, by default, set to `On Demand`, which causes parsing of the message to be delayed. For information about how to cause the message to be parsed immediately, see "Parsing on demand" on page 4173.

- `XMLNSC Parser Options`. Set values for the properties that determine how the XMLNSC parser operates. For more information, see "Manipulating messages in the XMLNSC domain" on page 2546.

6. Use the **Validation** tab to provide validation based on the message set for predefined messages. For more information about validation, see "Validating messages" on page 1478. For information about how to complete this tab, see "Validation tab properties" on page 4169.
7. Use the **Instances** tab to specify how additional threads are handled for the message flow.
   - The `Additional instances pool` property specifies whether additional instance threads are allocated from a thread pool for the whole message flow, or from a thread pool for use by that node only. By default, this property is set to `Use Pool Associated with Message Flow`.
   - The `Additional instances` property specifies the number of additional threads that the broker can use to service the message flow and has the default value `0`.
8. Use the **Retry** tab to define how retry processing is carried out when a failure gets rolled back to the input node.
   - The `Retry mechanism` defines the format of the mechanism, and its type is Enumerate. Set the property to either `Failure` or `Short and long retry`.
   - The `Retry threshold` property defines the number of retries to correct the failure, and its type is Integer.
   - The `Short retry interval` property defines the time the client waits in seconds before attempting to correct the failure, and its type is Integer.
   - The `Long retry interval` property defines the time the client waits in seconds before attempting to correct the failure, and its type is Integer.

Terminals and properties:
The following table describes the terminals of the SCAInput node.

Terminal	Description
Failure	The output terminal to which the received message is propagated when a failure (such as a message validation failure) is detected
Out	The output terminal to which the message is routed if you have selected `Route to common out terminal` on the `Message routing` property.
Catch	The output terminal to which the message is routed if an exception is thrown downstream and caught by this node.
* (dynamic)	(Applicable only when the Binding type is Web Services.) A dynamic terminal is generated for each operation that is supported by the port type and implemented by the imported binding. The dynamic operation terminal to which the SCA message is routed depends on the operation that is defined in the SCA message when it is received.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The following table describes the Description properties of the SCAInput node.

Property	M	C	Default	Description
Node name	No	No	The node type	The name of the node.
Short description	No	No	None	A brief description of the node.
Long description	No	No	None	Text that describes the purpose of the node in the message flow.

The following table describes the Basic properties of the SCAInput node.

Property	M	C	Default	Description
Broker SCA definition	Yes	No	<None>	This property type is String. The name of the Broker SCA definition that contains configuration properties for the SCAInput node.
Message routing	Yes	No	`Route each operation to its own terminal`	This property determines whether a common Out terminal is to be used, or whether each operation is to be routed to its own terminal. • If the Binding type is MQ, the common out terminal is always used. • If the Binding type is Web Services, you can choose whether to route messages to a dynamic terminal, or to the common Out terminal.

The following table describes the Binding properties of the SCAInput node; the binding that is found in Broker SCA definition is defined here.

Property	M	C	Default	Description
Binding type	Yes	No	Derived from the Broker SCA definition	This property type is String. Its value derives from the Broker SCA definition.

The following table describes the Web Service properties of the SCAInput node. These properties are only relevant if Binding type is Web Services.

Property	M	C	Default	Description
WSDL file name	Yes	No	<None>	This property type is String. A value is given to this property when the WSDL configuration file is parsed.
Target namespace	No	No	Derived from the Broker SCA definition	This property type is String. `Target namespace` displays the namespace of the selected WSDL file.
Port type	Yes	No	The default value is the port type from the Broker SCA definition.	This property type is String. This property is read-only. Error conditions: • Selected `Port type` does not contain at least one operation.

Property	M	C	Default	Description
Imported binding	Yes	No	Derived from the Broker SCA definition.	This property type is String. The `Imported binding` box lists all the SOAP bindings that are associated with the selected port type. Bindings are listed in the order that they are displayed in the WSDL file. By default, the first binding that is pointed to by the port in the Broker SCA definition is selected. This property is updated every time the `Port type` value changes, and an information message is displayed that states that corresponding changes must be made in the WebSphere Process Server system. Error conditions: • No SOAP bindings in the WSDL file are associated with the `Port type`. • Selected binding does not have any operations.
Service port	Yes	No	Derived from the Broker SCA definition	This property type is String. The `Service port` box lists all the service ports that point to the selected binding. The service port that is referenced in the Broker SCA definition is selected by default. This property is updated every time the selected binding value changes, and an information message is displayed that states that corresponding changes must be made in the WebSphere Process Server system. Error conditions: • No ports point to the selected binding.
URL selector	Yes	Yes	Derived from the Broker SCA definition	This property type is String. `URL selector` is the HTTP path selector upon which the node accepts inbound messages.
Use HTTPS	No	Yes	Derived from the Broker SCA definition	This property type is Boolean. Its value is `True` if the HTTP location is an HTTPS; otherwise, its value is `False`.
Propagate only SOAP Body	No	No	True	This property type is Boolean. Select the check box if only the body of the SOAP message is to be propagated.
Use WS-Addressing	No	No	False	This property type is Boolean. Select the check box if WS-Addressing is to be used.

The following table describes the MQ properties of the SCAInput node. These properties are only relevant if the Binding type is MQ.

Property	M	C	Default	Description
Queue name	Yes	Yes	Derived from the Broker SCA definition	This property type is String. The value of this property can be changed directly on the node. If the property is changed, an information message is displayed that states that corresponding changes must be made to the WebSphere Process Server system.

The SCAInput node Input Message Parsing properties are described in the following table.

Property	M	C	Default	Description
Message domain	No	No	Set automatically according to the binding that is defined in the corresponding node.	The domain that is used to parse the incoming message. The domain is determined according to the Binding type type. You can change this property if the Binding type is MQ. The property is read-only when the Binding type is Web Services.
Message set	No	No	Picked automatically according to the Broker SCA definition that is chosen in the corresponding node.	The name of the message set in which the incoming message is defined. `Message set` is set automatically to the message set that contains the SCA file that is configured on the corresponding node. This property is read-only.
Message type	No	No	Not set	The node detects the message type automatically. You can change this property if the Binding type is MQ and the message domain is MRM. You cannot change this property if the Binding type is Web Services.
Message format	No	No	Not set	The name of the physical format of the incoming message. You can change this property if the Binding type is MQ and the message domain is MRM. You cannot change this property if the Binding type is Web Services.

The following table describes the Parser Options properties of the SCAInput node.

Property	M	C	Default	Description
Parse timing	No	No	`On demand`	This property controls when an input message is parsed. Valid values are `On demand`, `Immediate`, and `Complete`. For a full description of this property, see "Parsing on demand" on page 4173.
Build tree using XML Schema data types	No	No	Selected	This property controls whether the syntax elements in the message tree have data types taken from the XML Schema.
Retain mixed content	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters mixed text in an input message. If you select the check box, elements are created for mixed text. If you clear the check box, mixed text is ignored and no elements are created.
Retain comments	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters comments in an input message. If you select the check box, elements are created for comments. If you clear the check box, comments are ignored and no elements are created.
Retain processing instructions	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters processing instructions in an input message. If you select the check box, elements are created for processing instructions. If you clear the check box, processing instructions are ignored and no elements are created.

Property	M	C	Default	Description
Opaque elements	No	No	Blank	This property is used to specify a list of elements in the input message that are to be opaquely parsed. Opaque parsing is performed only if validation is not enabled (that is, if `Validate` is `None`); entries that are specified in `Opaque Elements` are ignored if validation is enabled.

The following table describes the Validation properties of the SCAInput node. For more information, see "Validation properties" on page 4169.

Property	M	C	Default	Description
Validate	No	Yes	`Content and value`	This property controls whether validation takes place. Valid values are `None`, `Content and value`, and `Content`.
Failure action	No	No	`Exception`	This property controls what happens if validation fails. You can set this property only if you set `Validate` to `Content` or `Content and value`. Valid values are `User Trace`, `Local Error Log`, `Exception`, and `Exception List`.

The Security properties of the node are described in the following table. Set values for these properties to control the extraction of an identity from a message (when a security profile is associated with the node). For more information about these properties, see "Identity" on page 390, "Configuring the extraction of an identity or security token" on page 447, "Message flow security overview" on page 383, and "Setting up message flow security" on page 431.

Property	M	C	Default	Description
Identity token type	No	No	`None`	This property specifies the type of identity token present in the incoming message. Valid values are: `Transport Default`, `Username`, `Username + Password`, `SAML Assertion`, and `X.509 Certificate`. If this property is not specified, the identity is retrieved from the Basic-Auth transport header and the type is set to `Username`.
Identity token location	No	No	`None`	This property specifies where, in the message, the identity can be found. The location is specified as an ESQL field reference, an XPath expression, or a string literal. If you use a string literal, it must be enclosed in single quotation marks and must not contain a period (.), If this property is not specified, the identity is retrieved from the MQMD.UserIdentifier transport header.
Identity password location	No	No	`None`	This property specifies where, in the message, the password can be found. The location is specified as an ESQL field reference, an XPath expression, or a string literal. If you use a string literal, it must be enclosed in single quotation marks and must not contain a period (.), If it is not specified, the password is not set. This property can be set only if `Identity token type` is set to `Username + Password`.
Identity IssuedBy location	No	No	`None`	This property specifies a string or path expression that describes the issuer of the identity. The location is specified as an ESQL field reference, an XPath expression, or a string literal. If you use a string literal, it must be enclosed in single quotation marks and must not contain a period (.), The value specifies the Issuer that is passed to a WS-Trust v1.3 STS provider. If this property is not specified, the MQMD.PutApplName value is used. If you leave the `Identity issuedBy location` field blank and the MQMD.PutApplName is also blank, the string `MQ` is used.

Property	M	C	Default	Description
Treat security exceptions as normal exceptions	No	No	`False`	This property specifies whether to treat security exceptions (such as "Access Denied") as normal exceptions, and propagate them to the Failure terminal (if wired). This property is turned off by default, which ensures that security exceptions cause the message to be backed out even if the Failure terminal is wired.

The following table describes the Instances properties of the SCAInput node.

Property	M	C	Default	Description
Additional instances pool	No	Yes	`Use Pool Associated with Message Flow`	The pool from which additional instances are obtained. • If you select `Use Pool Associated with Message Flow`, additional instances are obtained from the message flow value. • If you select `Use Pool Associated with Node`, additional instances are allocated from the additional instances of the node, based on the number specified in the `Additional instances` property.
Additional instances	No	Yes	`0`	The number of additional instances that the node can start if the `Additional instances pool` property is set to `Use Pool Associated with Node`. By default, no additional instances are given to the node.

The following table describes the Retry properties of the SCAInput node.

Property	M	C	Default	Description
Retry mechanism	No	No	`Failure`	This property specifies how retry processing is handled when a failure is rolled back to the SCAInput node. • If you select `Failure`, retry processing is not performed so you cannot set the remaining properties. • If you select `Short and long retry`, retry processing is performed first at the interval that is specified by the `Short retry interval` property, and if that is unsuccessful, it is then performed at the interval that is specified by the `Long retry interval` property.
Retry threshold	No	Yes	`0`	The maximum number of times that retry processing is performed for short retry.
Short retry interval	No	Yes	`0`	The interval between short retry attempts.
Long retry interval	No	Yes	`0`	The interval between long retry attempts.

The Transactionality property of the SCAInput node is described in the following table.

Property	M	C	Default	Description
Transaction mode	No	No	For MQ, 'yes'	This property can be 'automatic', 'no', or 'yes'.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Working with Service Component Architecture (SCA)" on page 2095
Start here to find out how you can use SCA to allow interoperability with WebSphere Process Server Version 6.2.

"Message flow security overview" on page 383
WebSphere Message Broker provides a security manager, which enables you to control access to individual messages in a message flow, using the identity of the message.

"Identity" on page 390
In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

"Service Component Architecture (SCA) overview" on page 2096
Service Component Architecture (SCA) is a specification that describes a model for building applications and systems using a service-oriented architecture (SOA).

"SOAP parser and domain" on page 1082
You can use the SOAP parser to create a common WSDL-based logical tree format for working with Web services, independent of the physical bitstream format.

"Samples" on page 98
The WebSphere Message Broker Toolkit provides samples that show the features that are available in WebSphere Message Broker, and how to use them. This topic provides links to the information about the individual samples.

Related tasks:

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

"Setting up message flow security" on page 431
Set up security on a message flow to control access based on the identity of a message passing through the message flow.

"Configuring the extraction of an identity or security token" on page 447
You can configure the SecurityPEP node or security enabled input nodes to extract the identity or security token from a message and store it in the properties tree identity fields, enabling it to be processed throughout the message flow and propagated at output or request nodes.

"Importing from WSDL" on page 2946
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to create a new message definition from WSDL.

"Creating an application based on SCA import or export files" on page 1422
You can create a new application that is based on existing SCA import or export files.

Related reference:

"SCAAsyncRequest node" on page 4690
Use the SCAAsyncRequest node with the SCAAsyncResponse node to construct a pair of message flows that invoke a WebSphere Process Server service component asynchronously.

"SCAReply node" on page 4726
Use the SCAReply node to send a message from the broker to the originating client in response to a message received by a SCAInput node.

"SCAAsyncResponse node" on page 4698
Use the SCAAsyncResponse node with the SCAAsyncRequest node to construct a pair of message flows that start a component asynchronously.

"SCARequest node" on page 4719
Use the SCARequest node to send a request to WebSphere Process Server. The node is a synchronous request and response node, and blocks after sending the request until the response is received. The node can also send one-way requests.

"Parsing on demand" on page 4173
On-demand parsing, referred to as partial parsing, is used to parse an input message bit stream only as far as is necessary to satisfy the current reference. The parsers that can perform partial parsing of input messages are the JSON, MRM, XML, XMLNS, and XMLNSC parsers.

"WS-Addressing with the SOAPInput node" on page 1651
Various options are available when you use WS-Addressing with the SOAPInput node.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"Message domains used by SCA nodes"
Use the `Message domain` property of the SCAInput node to determine which domain to use to parse the incoming message.

"SecurityPEP node" on page 4729
Use the SecurityPEP node in a message flow to invoke the message flow security manager at any point in the message flow.

Message domains used by SCA nodes:

Use the `Message domain` property of the SCAInput node to determine which domain to use to parse the incoming message.

If the Binding type is `MQ`, you must set the message domain. The SCAInput node supports all MQ data formats used by WebSphere Process Server. If the data format in the Service Component Definition Language (SCDL) file is either `com.ibm.websphere.sca.mq.data.impl.MQDataBindingImplXML` or `com.ibm.wbiserver.datahandler.xml.XMLDataHandler`, the default domain is XMLNSC; otherwise the default domain is BLOB.

See the following table for the relationship between the MQ data format and default message domain:

Data format	Description	Default message domain
Delimited MQ	Serializes the business object to and from a delimited format in the message that is sent and received from the MQ client.	BLOB

Data format	Description	Default message domain
Fixed Width MQ	Serializes the business object to and from a fixed width format in the message that is sent and received from the MQ client.	BLOB
JSON	Sends and receives a business object that is based on JavaScript Object Notation (JSON) from the MQ client.	BLOB
JMS adapter language	Needed if your message body contains a message in C, or COBOL, or PL/I.	BLOB
MQ serialized business object XML	Serializes a business object to an XML document, and deserializes an XML document to a business object. Wrapped data objects must be a complex type; they cannot be a simple type.	XMLNSC
MQ serialized Java	Serializes a business object to a Java object and deserializes a Java object to a business object.	BLOB
MQ unstructured binary message	Sets the incoming bytes into a business object property called `value` on the inbound message, and gets the bytes from the business object property called `value`, and sets it in the output stream on the outbound message.	BLOB
MQ unstructured text message	Sets the incoming text message into a business object property called `value` on the inbound message, and gets the text message from the business object property called `value`, and sets it in the output stream on the outbound message.	BLOB
MQRFH header	Contains fixed size and variable sized pieces of information that are specified by the MQRFH header standard.	BLOB
MQRFH2 header	Contains fixed size and variable sized pieces of information that are specified by the MQRFH2 header standard.	BLOB

Data format	Description	Default message domain
WTX	Allows the use of WebSphere Transformation Extender (WTX), a universal validation and transformation engine, to convert business objects to many data formats, and many data formats to business objects.	BLOB

If the Binding type is `Web Service`, the message domain is always `SOAP`.

SCARequest node

Use the SCARequest node to send a request to WebSphere Process Server. The node is a synchronous request and response node, and blocks after sending the request until the response is received. The node can also send one-way requests.

This topic contains the following sections:
- "Purpose"
- "Using this node in a message flow"
- "Configuring the SCARequest node" on page 4720
- "Terminals and properties" on page 4721

Purpose:
The SCARequest node sends synchronous outbound two-way (request-response) requests, and one-way (request only) operations, to a business process on WebSphere Process Server.

An outbound Broker SCA definition (`.outsca`) file is required to configure the SCARequest node. The Broker SCA definition contains specific data related to the binding that is used by the SCA Export component.

The SCARequest node is contained in the **SCA** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:
You can use the node in any message flow that needs to call an application running on WebSphere Process Server.

`LocalEnvironment.Destination.SCA.Request.KeyAlias`

You can change the timeout by changing the value in the following location in the local environment:

`LocalEnvironment.Destination.SCA.Request.Timeout`

You can change the timeout by changing the value in the following location in the local environment:

`LocalEnvironment.Destination.SCA.Request.Timeout`

For a Web Services binding, you can change the URL to which the request is sent. Change the value in the following location in the local environment:

`LocalEnvironment.Destination.SCA.Request.Binding.WebServices.Transport.HTTP.WebServiceURL`

For an MQ binding, you can change the Request Queue Manager and Request Queue by changing the values in the following locations:

```
LocalEnvironment.Destination.SCA.Request.Binding.MQ.queueManagerName
LocalEnvironment.Destination.SCA.Request.Binding.MQ.queueName
```

Configuring the SCARequest node:
Ensure that the message set contains a Broker SCA definition with an extension of `.outsca` with which to configure the SCARequest node.

There are two methods of putting an instance of the SCARequest node into a message flow: You can either drag an instance of the node from the node palette, or drag a Broker SCA definition with an extension of `.outsca` from a message set, onto the message flow editor canvas.

Dragging a node onto the canvas

If you have dragged an instance of the node from the palette onto the canvas, use one of the following methods to start to configure it:

- Drag a Broker SCA definition with an extension of `.outsca` onto the node.
- Type the file name in the `SCA file name` option in the Properties view of the node.
- Browse to the file (`SCA file name` option in the Properties view of the node.)

Dragging a .outsca file onto the canvas

- If the `.outsca` file contains a WSDL that has only a single, one-way operation defined, a SCARequest node is created.
- If the `.outsca` file contains a WSDL that only has request-response operations, you first select the operation, and say whether you want the outbound request to be sent synchronously or asynchronously. For a synchronous request, a SCARequest node is created and configured. For an asynchronous request, a pair of SCAAsyncRequest and SCAAsyncResponse nodes are created and configured.
- If the `.outsca` file contains a WSDL that has a mixture of one-way and request-response operations defined, and you select a one-way operation, a SCARequest node is created.

If the request contains a two-way operation, the node blocks until a response message is received, or for the time defined by the timeout property (120 seconds by default). If the timeout period is exceeded, the message received on the input terminal of the SCARequest is propagated to the Failure terminal.

The SOAP, XMLNSC, XMLNS, MRM, MIME, JSON, and BLOB domains are available to parse the response.

- If the Binding type is `Web Service`, the message domain is SOAP.
- If the Binding type is `MQ`, the default domain is BLOB if any operations have non-XML data bindings. If all operations have XML data binding the default domain is XMLNSC. The data binding is identified as XML if the SCA Export uses `com.ibm.websphere.sca.mq.data.impl.mqdatabindingimplxml` or `com.ibm.wbiserver.datahandler.xml.XMLDataHandler`.

If the request contains a one-way operation, the node sends the request message, then routes the input message through to the Out terminal. In this case, the properties in the Response Message Parsing tab are unavailable.

The values for many of the properties of the node are provided by the Broker SCA definition. The properties of the node are displayed in the Properties view. All

mandatory properties for which you must enter a value (properties that do not have a default value defined) are marked with an asterisk.

Terminals and properties:
The SCARequest node terminals are described in the following table.

Name	Type	Description
In	Input data	The node is driven by a message arriving on the In terminal.
Out	Output data	When a message has been sent to an external resource it is sent to the Out terminal unchanged, except for the addition of status information.
Failure	Output data	A failure in the node is sent to the failure terminal.
Fault	Output data	A SOAP fault from the external system is sent to this terminal. This terminal is only used when the Binding type is Web Services.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Description properties of the SCARequest node are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type	The name of the node.
Short description	No	No	None	A brief description of the node.
Long description	No	No	None	Text that describes the purpose of the node in the message flow.

The Basic properties of the SCARequest node are described in the following table.

Property	M	C	Default	Description
Broker SCA Definition	Yes	No	none	The name of the Broker SCA Definition that contains configuration properties for the SCA Request node. Click the browse button to list all relevant Broker SCA Definitions in the current workspace.

The Binding properties of the SCARequest node are described in the following table.

Property	M	C	Default	Description
Binding type	Yes	No	none	The binding type that was found in the SCA Export.
Request Timeout	No	Yes	120	The time in seconds before the request times out waiting for a response from WebSphere Process Server. If the timeout period is exceeded, the message received on the input terminal of the SCARequest is propagated to the Failure terminal. This property is disabled if the Broker SCA definition (inbound) contains only one-way operations.

The Web Services properties are described in the following table. These properties are shown only if the binding type is Web Services.

Property	M	C	Default	Description
WSDL file name	Yes	No	<none>	This property type is String. When you select a WSDL file for the `WSDL file name` field, the WSDL is validated to ensure that it is WS-I compliant. Only Deployable WSDL can be used to configure the SCA nodes. After a valid WSDL file is selected, the message set project to which WSDL file belongs is added as a referenced project to the corresponding flow project, if the reference does not exist.
Port type	Yes	No	From WSDL	This property type is String. The field lists all the `Port types` defined in WSDL file selected in the `WSDL file name` property. Error Conditions: • Selected `Port type` does not contain at least one operation. WSDL properties are disabled when the node is configured to act in gateway mode.
Imported Binding	Yes	No	From WSDL	This property type is String. This property is updated every time that the `Port type` value changes. The field lists the imported SOAP bindings with HTTP or JMS transport associated with the selected `Port type`. When you select a binding, the property tab for the associated transport is enabled; otherwise, it is disabled. Bindings are listed in the same order in which they appear in the WSDL file. The selected binding is the one that has both ports and operations. If there is no such binding, then binding with ports is selected. If no bindings have ports then the first binding in the list is selected. Error Conditions: • No SOAP bindings (with HTTP or JMS transport) in the WSDL file are associated with the `Port type`. • The selected binding does not have any operations. WSDL properties are disabled when the node is configured to act in gateway mode.

Property	M	C	Default	Description
Operation	Yes	No	From WSDL	This property type is String. The `Binding operation` box contains all the operations defined by the selected binding. The first operation in the list is selected by default. This property is updated every time the selected binding value changes WSDL properties are disabled when the node is configured to act in gateway mode.
Service port	Yes	No	From WSDL	This property type is String. This field is updated every time that the selected binding is updated. This field lists all the WSDL ports that point to the selected binding. The first service port for the binding is selected by default. This property is updated every time the selected binding value changes. Error Conditions: • No ports point to the selected binding. WSDL properties are disabled when the node is configured to act in gateway mode.
Target namespace	Yes	No	From WSDL	`Target namespace` is implemented as a read-only field. This hidden property type is String. It is updated with the `Target namespace` of the WSDL file when the `WSDL file name` is configured. WSDL properties are disabled when the node is configured to act in gateway mode.
Web service URL	Yes	Yes	none	The URL of the SOAP address selected. This property is automatically derived from the `<soap:address>` element of the selected `Service port`. Whenever the selected port is updated, the `Web service URL` is updated accordingly. However, if you override the value then your value persists and the URL is no longer updated from the service port. If you choose to override this property you must specify it in the form http://<hostname>[:<port>]/[<path>] where: • http://<hostname> must be specified. • <port> has a default of 80. If you specify a value, you must include the colon : before the port number. • <path> has a default of /. If you specify a value, you must include the / before the path. For more details of how to override this property, see Changing the default URL for a SOAPRequest node or a SOAPAsyncRequest node request.

Property	M	C	Default	Description
Propagate only SOAP Body, owned by XMLNSC domain	No	No	false	If selected, only the SOAP body is propagated, otherwise the entire SOAP message is propagated.
Use WS-Addressing	No	No	false	Select the check box if you want to use WS-Addressing.

The MQ properties are described in the following table. These properties are shown only if the binding type is MQ.

Property	M	C	Default	Description
Operation	Yes	No	From WSDL	The selected operation from the selected binding in the WSDL file. The WSDL is not displayed; the WSDL is in the Broker SCA Definition file.
Queue name	Yes	Yes	none	The queue that receives a request message from WebSphere Message Broker.
Queue manager name	Yes	Yes	none	The Queue Manager that receives a request from WebSphere Message Broker.
Reply-to queue name	Yes	Yes	none	The queue that receives a response message from WebSphere Process Server.
Reply-to queue manager name	No	Yes	none	The Queue Manager that receives a response from WebSphere Process Server.
Response Message Correlation	Yes	No	FromMsgId	This property indicates how WebSphere Message Broker is to complete correlation ID information in the response message.

The Response Message Parsing properties of the node are described in the following table. The node sets these properties automatically; the table describes when you can change them.

Property	M	C	Default	Description
Message domain	No	No	Set automatically according to the binding that is defined in the corresponding node.	The domain that is used to parse the response message. It is determined according to the Binding type. You can change this property if the Binding type is MQ. The property is read-only when the Binding type is Web Services.
Message set	No	No	Picked automatically according to the Broker SCA definition that is chosen in the corresponding node.	The name of the message set in which the response message is defined. `Message set` is set automatically to the message set that contains the SCA file that is configured on the corresponding node. This property is read-only.
Message type	No	No	Not set	The node detects the message type automatically. You can change this property if the Binding type is MQ and the message domain is MRM. You cannot change this property if the Binding type is Web Services.

Property	M	C	Default	Description
Message format	No	No	Not set	The name of the physical format of the response message. You can change this property if the Binding type is MQ and the message domain is MRM. You cannot change this property if the Binding type is Web Services.

The validation properties are described in the following table. For more information, see "Validation properties" on page 4169.

Property	M	C	Default	Description
Validate	No	Yes	`Content and value`	This property controls whether validation takes place. Valid values are `None`, `Content and value`, and `Content`.
Failure action	No	No	`Exception`	This property controls what happens if validation fails. You can set this property only if you set `Validate` to `Content` or `Content and value`. Valid values are `User Trace`, `Local Error Log`, `Exception`, and `Exception List`.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"SOAP parser and domain" on page 1082
You can use the SOAP parser to create a common WSDL-based logical tree format for working with Web services, independent of the physical bitstream format.

"SOAP tree overview" on page 1611
This tree format allows you to access the key parts of the SOAP message in a convenient way.

"XMLNSC parser" on page 1090
The XMLNSC parser is a flexible, general-purpose XML parser that offers high performance XML parsing and optional XML Schema validation.

"Working with Service Component Architecture (SCA)" on page 2095
Start here to find out how you can use SCA to allow interoperability with WebSphere Process Server Version 6.2.

"Samples" on page 98
The WebSphere Message Broker Toolkit provides samples that show the features that are available in WebSphere Message Broker, and how to use them. This topic provides links to the information about the individual samples.

Related tasks:

"Validating messages" on page 1478
The broker provides validation based on the message model for predefined messages.

"Manipulating messages in the XMLNSC domain" on page 2546
If you are writing ESQL to process messages in the XMLNSC domain, it is helpful to learn about the structure of the message tree that the XMLNSC parser builds.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Importing from WSDL" on page 2946
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to create a new message definition from WSDL.

Related reference:

"Parsing on demand" on page 4173
On-demand parsing, referred to as partial parsing, is used to parse an input message bit stream only as far as is necessary to satisfy the current reference. The parsers that can perform partial parsing of input messages are the JSON, MRM, XML, XMLNS, and XMLNSC parsers.

"Validation properties" on page 4169
You can control validation by setting properties on the Validate and Parser Options tabs for the nodes that are listed in the following table.

"SCAAsyncResponse node" on page 4698
Use the SCAAsyncResponse node with the SCAAsyncRequest node to construct a pair of message flows that start a component asynchronously.

"SCAInput node" on page 4707
Use the SCAInput node with the SCAReply node to process messages from WebSphere Process Server.

"SCAReply node"
Use the SCAReply node to send a message from the broker to the originating client in response to a message received by a SCAInput node.

SCAReply node

Use the SCAReply node to send a message from the broker to the originating client in response to a message received by a SCAInput node.

This topic contains the following sections:
- "Purpose"
- "Using this node in a message flow" on page 4727
- "Terminals and properties" on page 4727

Purpose:
The SCAReply node can be used in a message flow to send a reply back to the client that used an SCAInput node to send an SCA request to the WebSphere Message Broker. The SCAReply node and the SCAInput node must be in the same execution group.

The SCAReply node must use the same transport as the SCAInput node that originated the SCA request.

The SCAReply node is contained in the **SCA** drawer of the message flow node palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:
Look at the following sample to seem how to use the node:

SCA nodes

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

A ReplyIdentifier is created on the SCAInput node and is set in the message context and the local environment.

The value needs to be preserved throughout the flow especially if the SCAInput and SCAReply nodes are in different flows, so that the reply can reach the originating WebSphere Process Server client. The reply identifier is stored in the following location:

```
LocalEnvironment.Destination.SCA.Reply.ReplyIdentifier
```

Terminals and properties:
When you have put an instance of the SCAReply node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (properties that do not have a default value defined) are marked with an asterisk.

The SCAReply node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the message is routed if a failure is detected in this node.
Out	The output terminal to which the message is routed if it has been propagated successfully, and if further processing is required within this message flow. The message is propagated unchanged, except for the addition of status information.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The following table describes the Description properties of the SCAReply node.

Property	M	C	Default	Description
Node name	No	No	The node type: SCAReply	The name of the node.
Short description	No	No	None	A brief description of the node.
Long description	No	No	None	Text that describes the purpose of the node in the message flow.

The following table describes the Validation properties of the SCAReply node.

Property	M	C	Default	Description
Validate	Yes	Yes	Inherit	Valid values are `None`, `Content`, `Content and Value`, and `Inherit`.
Failure action	Yes	No	`Exception`	This property controls what happens if validation fails. You can set this property only if you set `Validate` to `Content` or `Content and Value`. Valid values are `User Trace`, `Local Error`, `Exception`, and `Exception List`.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Working with Service Component Architecture (SCA)" on page 2095
Start here to find out how you can use SCA to allow interoperability with WebSphere Process Server Version 6.2.

"Message flow aggregation" on page 2718
Aggregation is the generation and fan-out of related requests that are derived from a single input message, and the fan-in of the corresponding replies to produce a single aggregated reply message.

"Service Component Architecture (SCA) overview" on page 2096
Service Component Architecture (SCA) is a specification that describes a model for building applications and systems using a service-oriented architecture (SOA).

"Samples" on page 98
The WebSphere Message Broker Toolkit provides samples that show the features that are available in WebSphere Message Broker, and how to use them. This topic provides links to the information about the individual samples.

Related tasks:

"Configuring aggregation flows" on page 2721
Use aggregation message flows to generate and fan-out a number of related requests, fan-in the corresponding replies, and compile those replies into a single aggregated reply message, by using the AggregateControl, AggregateRequest, and AggregateReply nodes.

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

"Importing from WSDL" on page 2946
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to create a new message definition from WSDL.

"Importing WSDL definitions from the command line" on page 2948
WSDL definitions can be imported using the (**`mqsicreatemsgdefsfromwsdl`**) command.

"Creating an application based on SCA import or export files" on page 1422
You can create a new application that is based on existing SCA import or export files.

Related reference:

"SCAInput node" on page 4707
Use the SCAInput node with the SCAReply node to process messages from WebSphere Process Server.

"SCAAsyncRequest node" on page 4690
Use the SCAAsyncRequest node with the SCAAsyncResponse node to construct a pair of message flows that invoke a WebSphere Process Server service component asynchronously.

"SCAAsyncResponse node" on page 4698
Use the SCAAsyncResponse node with the SCAAsyncRequest node to construct a pair of message flows that start a component asynchronously.

"SCARequest node" on page 4719
Use the SCARequest node to send a request to WebSphere Process Server. The node is a synchronous request and response node, and blocks after sending the request until the response is received. The node can also send one-way requests.

SecurityPEP node

Use the SecurityPEP node in a message flow to invoke the message flow security manager at any point in the message flow.

This topic contains the following sections:
- "Purpose"
- "Using the SecurityPEP node in a message flow" on page 4731
- "Configuring the SecurityPEP node" on page 4731
- "Terminals and properties" on page 4732

Purpose:
The SecurityPEP node enables you to invoke the message flow security manager at any point in the message flow between an input node and an output (or request) node.

Message flow security enables the broker to perform end-to-end processing of an identity or security token carried in a message through a message flow. This capability enables you to configure security for a message flow, allowing you to control access based on the identity or security token associated with the message and providing a security mechanism that is independent of both transport type and message format.

The SecurityPEP node enables you to invoke the security manager even if your message flow input nodes do not support message flow security (for example, TCPIPClientInput or SAPInput nodes). If you use input nodes that do not support message flow security, or you require some processing or routing of the message before the required security operation can be identified, you can use the SecurityPEP node to invoke the message flow security.

The SecurityPEP node also enables you to invoke different aspects of security at different points in the message flow. For example, you might require authentication to occur at a security enabled input node, whereas token mapping and authorization might be required after some logic in the message flow has determined the necessary business operation. Alternatively, you might have a message flow with multiple request nodes that require SecurityPEP nodes to enable one type of security token to be mapped to another type for propagation by the request nodes.

You can use the node properties to specify the location of the security tokens in the message tree. These properties contain an XPath expression or ESQL path that defines the location of the security tokens in the message tree. The message flow security manager extracts this information from the message and sends it to the external Policy Decision Point (PDP), which uses the information for authentication, authorization, or mapping. The PDP to be used is configured by the associated security profile.

Alternatively, you can configure the SecurityPEP node to use the current tokens, which have already been extracted by an upstream input node or SecurityPEP node, and stored in the Properties folder. When the node is configured with the token type as `Current token`, if a mapped token exists, the mapped token is used; otherwise, the source token is used.

The SecurityPEP node must be associated with a security profile, which specifies the security operations to be enforced by the node, including authentication, authorization, and mapping. If no security profile is associated with the node, the node propagates the message to the Output terminal without enforcing any security. Security profiles are configured by the broker administrator before deploying a message flow, and are accessed by the security manager at run time. If a security profile is specified on either a message flow or a node, the profile must be available when the message flow is deployed; otherwise, a deployment error occurs.

The associated security profile also allows you to specify the external security provider to be used (LDAP, WS-Trust V1.3 STS, or TFIM V6.1), and to configure the way in which they are used. The security profile is associated with the SecurityPEP node (or its owning message flow) during deployment, by editing the **`Security Profile`** property with the Broker Archive editor.

For information about the types of security token that are supported by the SecurityPEP node, see “Identity” on page 390.

The SecurityPEP node propagates messages to the Out terminal only if all the configured security operations complete successfully. The input messages are propagated unmodified, apart from the population of the source identity and mapped identity (if one exists).

If any of the configured security operations are unsuccessful, the SecurityPEP node throws a security exception wrapped in a recoverable exception (unlike the security enabled input nodes), which invokes the error handling that is provided by the message flow. This enables the exception to be caught and processed. Alternatively, you can handle SecurityPEP node exceptions by wiring the node's failure terminal into specific security failure processing logic. When a security operation fails, the input messages are unmodified apart from the population of the exception list.

The SecurityPEP node is contained in the **Security** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Message structure

The SecurityPEP node handles messages in the following message domains:
- MRM
- XMLNSC
- DataObject
- XMLNS
- JMSMap
- JMSStream
- MIME
- BLOB
- SOAP
- XML (this domain is deprecated; use XMLNSC)
- IDOC (this domain is deprecated; use MRM)

Using the SecurityPEP node in a message flow:
Look at the following sample to see how to use the SecurityPEP node:
- Security Policy Enforcement Point (PEP)

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Configuring the SecurityPEP node:
When you have put an instance of the SecurityPEP node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view.

All mandatory properties that do not have a default value defined are marked with an asterisk.

Configure the SecurityPEP node:

1. Optional: On the **Description** tab, enter a `Short description`, a `Long description`, or both. You can also rename the node on this tab.
2. On the **Basic** tab, set values for the properties that control the extraction of an identity or security token from a message (when a security profile is associated with the node).
   - Select an option from the `Identity token type` list to specify the type of identity in the incoming message tree. If you leave this option to the default (`Current token`), the token type that exists in the identity mapped or source field in the Properties folder is used.

     If `Current token` is selected, the `Identity token location` and `Identity password location` fields are disabled.
   - In `Identity token location`, enter the XPath expression, ESQL field reference, or string literal that specifies where, in the message, the identity or token is located. If you use a string literal, it must be enclosed in single quotes and must not contain a period (.),

     This property is disabled if the `Identity token type` property is set to `Current token`.
   - In `Identity password location`, enter the XPath expression, ESQL field reference, or string literal that specifies where, in the message, the password can be found. If you use a string literal, it must be enclosed in single quotes and must not contain a period (.),

This option can be set only if the `Identity token type` is set to `Username + Password`.

This property is disabled if the `Identity token type` property is set to `Current token`.

- In `Identity issuedBy location`, specify an XPath expression, an ESQL field reference, or a string literal that specifies the location in the message of the issuer value. If you use a string literal, it must be enclosed in single quotes and must not contain a period (.),

  If the associated security profile specifies a WS-Trust v1.3 STS provider (for example, TFIM V6.2) and this field is left blank, no WS-Trust Issuer.Address element is included in the WS-Trust request.

3. On the **Advanced** tab, set the properties to override the default settings for a WS-Trust v1.3 STS. These properties can be set only if the security profile associated with the SecurityPEP node specifies a WS-Trust v1.3 STS.
   - Use the `WS-Trust Applies-To Address` property to specify the Address for the **/wst:RequestSecurityToken/wsp:AppliesTo** element of the WS-Trust message. You can use this property to provide the URI of the service for which the security token is to be validated or issued.

     You can specify this value as an ESQL field reference, an XPath expression, or a string literal. If you use a string literal, it must be enclosed in single quotes and must not contain a period (.),

     By default, this value is a URI for the fully qualified name of the message flow, in the form *uri:Brokername.Execution Group Name.Message Flow Name*
   - Use the `WS-Trust Applies-To Service` property to specify the Service Name for the **/wst:RequestSecurityToken/wsp:AppliesTo** element of the WS-Trust message. You can use this property to provide the Service Name of the service for which the security token is to be validated or issued.

     You can specify this value as an ESQL field reference, an XPath expression, or a string literal. If you use a string literal, it must be enclosed in single quotes and must not contain a period (.),

     By default, this value is left blank, which means that the WS-Trust request will not include this element.
   - Use the `WS-Trust Applies-To PortType` property to specify the Port Type for the **/wst:RequestSecurityToken/wsp:AppliesTo** element of the WS-Trust message. You can use this property to provide the Port Type of the service for which the security token is to be validated or issued.

     You can specify this value as an ESQL field reference, an XPath expression, or a string literal. If you use a string literal, it must be enclosed in single quotes and must not contain a period (.),

For more information, see "Message flow security overview" on page 383 and "Setting up message flow security" on page 431.

Terminals and properties:
The terminals of the SecurityPEP node are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing.
Out	The output terminal to which the message is routed if the SecurityPEP node is associated with a security profile and if all the configured security operations complete successfully. The propagated properties folder identity elements are updated by the configured security operations.

Terminal	Description
Failure	The output terminal to which the message is routed if there is a failure in the node; for example, if the configured security operations return an exception.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the bar file to deploy it).

The Description properties of the SecurityPEP node are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, SecurityPEP	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The Basic properties of the SecurityPEP node are described in the following table. Set values for these properties to control the extraction of an identity from a message (when a security profile is associated with the node). For more information about these properties, see "Identity" on page 390, "Configuring the extraction of an identity or security token" on page 447, "Message flow security overview" on page 383, and "Setting up message flow security" on page 431.

Property	M	C	Default	Description
Identity token type	No	No	`None`	This property specifies the type of identity token present in the incoming message. Valid values are: • `Current token` • `Username` • `Username and password` • `X.509 Certificate` • `SAML assertion` • `Kerberos GSS v5 AP_REQ` • `LTPA v2 token` • `Universal WSSE token` If this property is set to `Current token`, the identity in the Properties folder is used. You can also specify the `Username and password` value to validate a RACF PassTicket using a WS-Trust V1.3 STS such as TFIM V6.2.
Identity token location	No	No	`None`	This property specifies where, in the message, the identity or security token can be found. The location is specified as an ESQL field reference, an XPath expression, or a string literal. If you use a string literal, it must be enclosed in single quotes and must not contain a period (.),

Property	M	C	Default	Description
Identity password location	No	No	`None`	This property specifies where, in the message, the password can be found. The location is specified as an ESQL field reference, an XPath expression, or a string literal. If you use a string literal, it must be enclosed in single quotes and must not contain a period (.), This property can be set only if `Identity token type` is set to `Username and password`.
Identity IssuedBy location	No	No	`None`	This property specifies an XPath expression or ESQL path that describes the issuer of the identity or security token. The location is specified as an ESQL field reference, an XPath expression, or a string literal. If you use a string literal, it must be enclosed in single quotes and must not contain a period (.), This option is used when the associated security profile specifies a WS-Trust V1.3 STS provider (for example, TFIM V6.2) for authentication, mapping or authorization. In this case, when this field is left blank, no WS-Trust Issuer.Address element is sent.

The Advanced properties of the SecurityPEP node are described in the following table. These properties are used only if the security profile specifies a WS-Trust V1.3 STS (for example, TFIM V6.2).

Property	M	C	Default	Description
WS-Trust Applies-To Address	No	Yes	Not set	This property sets the Address for the **`/wst:RequestSecurityToken/wsp:AppliesTo`** element of the WS-Trust message. Use this property to provide the URI of the service for which the security token is to be validated or issued. This value can be specified as an ESQL field reference, an XPath expression, or a string literal. If you use a string literal, it must be enclosed in single quotes and must not contain a period (.), By default, this value is a URI for the fully qualified name of the message flow, in the form *uri:Brokername.Execution Group Name.Message Flow Name*.
WS-Trust Applies-To Service	No	Yes	Not set	This property sets the Service Name for the **`/wst:RequestSecurityToken/wsp:AppliesTo`** element of the WS-Trust message. Use this property to provide the Service Name of the service for which the security token is to be validated or issued. This value can be specified as an ESQL field reference, an XPath expression, or a string literal. If you use a string literal, it must be enclosed in single quotes and must not contain a period (.), By default, this value is left blank, which means that the WS-Trust request does not include this element.

Property	M	C	Default	Description
WS-Trust Applies-To Port Type	No	Yes	Not set	This property sets the Port Type for the **/wst:RequestSecurityToken/wsp:AppliesTo** element of the WS-Trust message. Use this property to provide the Port Type of the service for which the security token is to be validated or issued. This value can be specified as an ESQL field reference, an XPath expression, or a string literal. If you use a string literal, it must be enclosed in single quotes and must not contain a period (.), By default, this value is left blank, which means that the WS-Trust request does not include this element.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Message flow security overview" on page 383
WebSphere Message Broker provides a security manager, which enables you to control access to individual messages in a message flow, using the identity of the message.

"Identity" on page 390
In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

"Security profiles" on page 387
A security profile defines the security operations that are to be performed in a message flow at SecurityPEP nodes and security enabled input and output nodes.

"WS-Security" on page 765
Web Services Security (WS-Security) describes enhancements to SOAP messaging to provide quality of protection through message integrity, message confidentiality, and single message authentication. WS-Security mechanisms can be used to accommodate a wide variety of security models and encryption technologies.

Related tasks:

"Setting up message flow security" on page 431
Set up security on a message flow to control access based on the identity of a message passing through the message flow.

"Creating a security profile" on page 433
You can create a security profile for use with Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant security token server (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2. Support is also provided for TFIM V6.1, for compatibility with previous versions of WebSphere Message Broker. You can create the security profile by using either the **mqsicreateconfigurableservice** command or an editor in the WebSphere Message Broker Explorer.

Related reference:

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

"HTTPInput node" on page 4474
Use the HTTPInput node to receive an HTTP message from an HTTP client for processing by a message flow.

"SOAPInput node" on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

"SCAInput node" on page 4707
Use the SCAInput node with the SCAReply node to process messages from WebSphere Process Server.

Sequence node

Use the Sequence node to add a sequence number to one or more groups of input messages.

This topic contains the following sections:
- "Purpose"
- "Using the Sequence node in a message flow" on page 4737
- "Configuring the Sequence node" on page 4737
- "Terminals and properties" on page 4738

Purpose:
The Sequence node enables you to receive groups of messages from an input source, and preserve the order in which the messages in each group arrived.

Use a Sequence node to generate a monotonically increasing sequence number for each sequence group. As each message in the group arrives at the Sequence node, the sequence number for the group is incremented and stored with the message in a location specified by the node property `Path to store sequence number` (for example, LocalEnvironment, MQRFH2 header, message body).

The sequencing applies to messages within the same sequence group only. You can use properties in the Sequence node to organize messages into groups according to a specified condition; for example, grouping all messages with the same value in a customer number field in the message. If no sequence group is specified, a single default group is used for all messages.

A Sequence node can receive input from multiple input nodes in the message flow or from input nodes that have additional instances. The Sequence node can process multiple sequence groups in parallel, but it processes only one request at a time for sequence numbers from the same sequence group.

The Sequence node allocates a sequence number to each message in a sequence group, and the next sequence number in the group is not allocated until the current message in the group has finished processing (either by being committed or rolled back). Only one thread at a time can process messages in the same sequence group downstream of the Sequence node, which ensures that sequencing is maintained for the group when there are multiple threads in the message flow.

When the Sequence node receives messages from multiple threads, the order in which the messages reach the Sequence node is preserved. However, the order in

which the messages arrive at the Sequence node might be different from the order in which they are taken from the transport by the input node. This situation can occur because messages on one thread might be overtaken by messages on other threads between the input node and Sequence node.

Each sequence group can be associated with only one Sequence node. Multiple Sequence nodes can have a sequence group with the same name, but each of those sequence groups is treated as a separate group. The combination of the execution group name, message flow name, node name, and sequence group name is used to differentiate between the sequence groups.

For example, you might have a message flow called *flow1* containing a Sequence node called *node1*, which is deployed to an execution group called *eg1*. A message is sent to it using a sequence group called *group1*. The result is *eg1/flow1/node1/group1*. Exactly the same message flow in a different execution group, for example *eg2*, would result in *eg2/flow1/node1/group1*.

The Sequence node is contained in the **Routing** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using the Sequence node in a message flow:
Look at the following sample to see how to use the Sequence node:

- Healthcare

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Configuring the Sequence node:
When you have put an instance of the Sequence node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view.

All mandatory properties that do not have a default value defined are marked with an asterisk.

Configure the Sequence node:

1. Optional: On the **Description** tab, enter a `Short description`, a `Long description`, or both. You can also rename the node on this tab.
2. On the **Basic** tab, set the properties that determine how the message sequence is controlled.
   - Use the `Path to store sequence number` property to specify the location in which to save the sequence number of the message. The location is specified as an XPath expression.
   - Use the `Path to sequence group identifier` property to specify the location of the sequence group identifier. The location is specified as an XPath expression. Messages that have the same group identifier are considered part of the same sequence group.

- Use the Start of sequence definition property to specify the first sequence number in each group. In the Literal field, specify any positive or negative numeric value as the first sequence number in the group. The default value is 0.

  When the first message of a specific sequence group is propagated by the Sequence node, the specified starting sequence number is used to assign the first sequence number; successive messages contain monotonically increasing sequence numbers. This value is overridden by the LocalEnvironment.Sequence.StartOfSequenceNumber variable.
- Use the End of sequence definition property to specify when each sequence group has been completed.
  - Select Literal to specify a literal sequence number. This value can be any positive or negative numeric value that is greater than the value of the Start of sequence definition property. When the message with the specified sequence number arrives, the sequence group is closed.
  - Select Predicate to specify an XPath expression that evaluates to either True or False, indicating whether the message is the last in the sequence. When the first message in the sequence has been determined, messages for that sequence group continue to have sequence numbers assigned to them until the end of sequence predicate evaluates to True.

    If the XPath expression is valid but is not present in the message, it evaluates to False and the next message in the sequence group is assigned a sequence number. When the predicate evaluates to True, the message becomes the last in the sequence and no more messages are assigned to the sequence for that sequence group.

    If the XPath expression is invalid, it fails and the message rolls back
  - Select Automatic to specify the timeout period for the node. This option specifies how long (in seconds) the node waits for messages to arrive in an empty queue, before closing the sequence group. This option is useful for applications that cannot determine the final number in the sequence. The timer starts when there are no messages in the queue waiting to be propagated. If new messages arrive before the timeout period is reached, the timer is canceled. If no new messages arrive before the end of the specified time, the sequence group is closed and any further messages for the group are considered part of a new group.

3. On the **Advanced** tab:
   - Use the Persistence mode property to specify whether to store sequence group state persistently. The default value is Non-persistent.

Terminals and properties:
The terminals of the Sequence node are described in the following table.

Terminal	Description
In	The input terminal through which the incoming message assembly arrives at the node.
Failure	The output terminal to which the message is routed if an error occurs. This value includes failures caused by retry processing.
Out	The output terminal to which the output message is propagated by default.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a

value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file for deployment).

The Description properties of the Sequence node are described in the following table.

Property	M	C	Default	Description
Node name	No	No	Sequence	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The Basic properties of the Sequence node are described in the following table.

Property	M	C	Default	Description
Path to store sequence number	No	No	`$OutputLocalEnvironment/Sequence/Number`	An XPath expression that specifies the location in which to save the sequence number of the message.
Path to sequence group identifier	No	No	Not set	An XPath expression that points to the location of the sequence group identifier. Messages that have the same group identifier are considered part of the same sequence group. This property functions in the same way as the `Correlation path` property in the Collector node.
Start of sequence definition	Yes	No	`0`	Specifies the first sequence number in each group. Valid values are positive or negative integers. The default value is 0.
End of sequence definition	Yes	No	`Automatic`	Specifies when each sequence group has been completed. Valid values are: • `Literal` and *number* • `Predicate` and *XPath* • `Automatic` and *time in seconds* The default is `Automatic`.

The Advanced properties of the Sequence node are described in the following table.

Property	M	C	Default	Description
Persistence mode	Yes	No	`Non-persistent`	Specifies whether to store sequence group state persistently. Valid options are: • `Non-persistent` • `Persistent`

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Message sequencing" on page 2784
Use message sequencing to ensure that messages are delivered to the receiving application in a particular order.

Related tasks:

"Using message sequences" on page 2783
You can maintain or change the sequence of messages in a message flow based on the sequence number and group ID contained in each message.

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"Resequence node" on page 4651
Use the Resequence node to control the sequence in which a group (or groups) of incoming messages are propagated in a message flow.

"Configurable message flow properties" on page 4020
When you add a message flow to a broker archive (BAR) file in preparation for deploying it to a broker, you can set additional properties that influence its run time operation. These properties are available for review and update when you select the **Manage and Configure** tab for the broker archive file.

SiebelInput node

Use the SiebelInput node to interact with a Siebel application.

This topic contains the following sections:

Purpose:

Use the SiebelInput node to interact with Siebel applications. For example, a SiebelInput node monitors a Siebel system for a specified event. When that event occurs, the SiebelInput node generates a message tree that represents the business object with the new event details. The message tree is propagated to the Out terminal so that the rest of the message flow can use the data to update other systems, or audit the changes.

The SiebelInput node is contained in the **WebSphere Adapters** drawer of the message flow node palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:
To use the SiebelInput node, you must first create the Siebel event table. For instructions, see "Creating the event store manually" on page 2072.

To function correctly, the SiebelInput node needs an adapter component, which you set using the `Adapter component` node property, and business object definitions, which are stored in the message set that you reference from the node. For this reason, you must provide a message set. By default, the message that is propagated from the SiebelInput node is in the DataObject domain, so the `Message domain` property is set to DataObject. You cannot specify a different domain. The message type is detected automatically by the node.

The SiebelInput node populates the route to label destination list with the name of the method binding. If you add a RouteToLabel node to the message flow after the SiebelInput node, the RouteToLabel node can use the name of the method binding to route the message to the correct part of the message flow for processing.

You can deploy only one input node that uses a particular adapter component to an execution group, but you can deploy many input nodes that use different adapter components to an execution group.

You can use the **mqsisetdbparms** command in the following format to configure an account name with a user name and password for the Adapter for Siebel Business Applications.

```
mqsisetdbparms broker name -n adapter name -u user name -p password
```

For example:

```
mqsisetdbparms BRK1 -n eis::SiebelCustomerInbound.inadapter -u siebeluid -p ********
```

Using configurable services for Siebel nodes

Siebel nodes can get Siebel connection details from either the adapter component or a configurable service. By using a configurable service, you can change the connection details for an adapter without the need to redeploy the adapter. For more details about creating, changing, reporting, and deleting the configurable services for Siebel, see "Changing connection details for Siebel adapters" on page 720.

You can also connect to different versions of Siebel by creating a custom EISProviders configurable service and setting the location of the appropriate library files. For more information, see "Connecting to different versions of Siebel" on page 2077.

Terminals and properties:
When you have put an instance of the SiebelInput node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view. If you double-click a SiebelInput node, you open the Adapter Connection wizard. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The SiebelInput node terminals are described in the following table.

Terminal	Description
Out	Business object events from the adapter are sent to the Out terminal.

Terminal	Description
Failure	If an error happens in the SiebelInput node, the message is propagated to the Failure terminal. Information about the error, and business object events can also be propagated to the Failure terminal.
Catch	Business object events are sent to the Catch terminal if they cause an uncaught exception in the message flow. If the Catch terminal is not connected, the retry process is activated to handle the business object.

The following table describes the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The SiebelInput node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, SiebelInput.	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The SiebelInput node Basic properties are described in the following table.

Property	M	C	Default	Description	`mqsiapplybaroverride` command property
Primary adapter component	Yes	Yes		The name of the adapter component that contains configuration properties for the adapter. Either enter a name of an adapter file or click **Browse** to select an adapter file from the list of files that are available in referenced message set projects. When theSiebelInput node receives data from the Siebel system, it associates that data with a method name, depending on the service operation name that is assigned to that type of data when you run the Adapter Connection wizard. The SiebelInput node attempts to handle methods that are defined in the primary adapter. If the type of data that is received does not correspond to any of the methods that are defined in the primary adapter, the node can handle methods that are defined in matching secondary adapters that are deployed to the same execution group.	adapterComponent

Property	M	C	Default	Description	mqsiapplybaroverride command property
Secondary adapter mode	No	Yes	None	Specifies whether the node can handle methods that are defined in secondary adapters. If you set the `Secondary adapter mode` property to `None`, the node handles only methods that are defined in the primary adapter. If the type of data that is received does not correspond to any of the methods that are defined in the primary adapter, a failure occurs. If you set this property to `All adapters in execution group`, the node can handle methods that are defined in any Siebel inbound adapters that are deployed to the same execution group.	secondaryAdapterMode

The SiebelInput node Routing properties are described in the following table.

Property	M	C	Default	Description
Set destination list	No	No	Selected	This property specifies whether to add the method binding name to the route to label destination list. If you select this check box, the method binding name is added so that you can use a RouteToLabel node in the message flow after the SiebelInput node.
Label prefix	No	No		The prefix to add to the method name when routing to label. Add a label prefix to avoid a clash of corresponding label nodes when you include multiple WebSphere Adapters input nodes in the same message flow. By default, there is no label prefix, so the method name and label name are identical.

The SiebelInput node Input Message Parsing properties are described in the following table.

Property	M	C	Default	Description
Message domain	No	No	DataObject	The domain that is used to parse the incoming message. By default, the message that is propagated from the SiebelInput node is in the DataObject domain. You cannot specify a different domain.
Message set	Yes	No	Set automatically	The name of the message set in which the incoming message is defined. This field is set automatically from the Adapter component property. If you set this property, then subsequently update the project dependencies to remove this message set reference, a warning is issued. Either update the `Message set` property, or restore the reference to this message set project.
Message type	No	No		The name of the incoming message. The node detects the message type automatically. You cannot set this property.
Message format	No	No		The name of the physical format of the incoming message. You cannot set this property.

The SiebelInput node Transactionality properties are described in the following table.

Property	M	C	Default	Description
Transaction mode	No	No	Yes	The transaction mode on this input node determines whether the rest of the nodes in the flow operate under sync point.

The Instances properties of the SiebelInput node are described in the following table. For a full description of these properties, see "Configurable message flow properties" on page 4020.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Additional instances pool	No	Yes	`Use Pool Associated with Message Flow`	The pool from which additional instances are obtained. • If you select `Use Pool Associated with Message Flow`, additional instances are obtained from the message flow value. • If you select `Use Pool Associated with Node`, additional instances are allocated from the node's additional instances based on the number specified in the `Additional instances` property.	componentLevel
Additional instances	No	Yes	`0`	The number of additional instances that the node can start if the `Additional instances pool` property is set to `Use Pool Associated with Node`. By default, no additional instances are given to the node.	additionalInstances

The SiebelInput node Retry properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Retry mechanism	No	No	Failure	This property specifies how retry processing is handled when a failure is rolled back to the SiebelInput node. • If you select `Failure`, retry processing is not performed so you cannot set the remaining properties. • If you select `Short and long retry`, retry processing is performed first at the interval that is specified by the `Short retry interval` property, and if that is unsuccessful, it is then performed at the interval that is specified by the `Long retry interval` property.	
Retry threshold	No	Yes	0	The maximum number of times that retry processing is performed for short retry.	retryThreshold
Short retry interval	No	Yes	0	The interval between short retry attempts.	shortRetryThreshold
Long retry interval	No	Yes	0	The interval between long retry attempts.	longRetryThreshold

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Overview of WebSphere Adapter for Siebel Business Applications" on page 2002
With WebSphere Adapter for Siebel Business Applications, you can create integrated processes that exchange information with a Siebel application, without

special coding.

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"DataObject parser and domain" on page 1114
Use the DataObject domain to parse and write messages for WebSphere Adapters and CORBA applications.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Creating the event store manually" on page 2072
To configure the Siebel application, create an event table and a Siebel business object.

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

SiebelRequest node

Use the SiebelRequest node to interact with a Siebel application.

This topic contains the following sections:
- "Purpose"
- "Using this node in a message flow" on page 4746
- "Terminals and properties" on page 4746

Purpose:
Use the SiebelRequest node to interact with Siebel applications. For example, a SiebelRequest node requests information from a Siebel Enterprise Information System (EIS). A customer business object is sent to Siebel, causing Siebel to retrieve information about a customer, such as an address and account details. The response information that is retrieved by the SiebelRequest node can then be used by the rest of the message flow. The SiebelRequest node can send and receive business data.

The SiebelRequest node is contained in the **WebSphere Adapters** drawer of the message flow node palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:
To function correctly, the SiebelRequest node needs an adapter component, which you set using the `Adapter component` node property, and business object definitions, which are stored in the message set that you reference from the node. For this reason, you must provide a message set. By default, the message that is propagated from the SiebelRequest node is in the DataObject domain, so the `Message domain` property is set to DataObject. You cannot specify a different domain. The message type is detected automatically by the node.

The SiebelRequest node supports local transactions by using the local transaction manager for the broker, and global transactions by using the external syncpoint coordinator for the broker.

To effectively maintain the pool of connections to Siebel, you can set a connection timeout value on a configurable service. For more information, see "Configuring EIS connections to expire after a specified time" on page 726.

You can deploy several WebSphere Adapters request nodes that use the same adapter component to an execution group.

You can use the **mqsisetdbparms** command in the following format to configure an account name with a user name and password for the Adapter for Siebel Business Applications.

```
mqsisetdbparms broker name -n adapter name -u user name -p password
```

For example:

```
mqsisetdbparms BRK1 -n eis::SiebelCustomerOutbound.outadapter -u siebeluid -p ********
```

Using configurable services for Siebel nodes

Siebel nodes can get Siebel connection details from either the adapter component or a configurable service. By using a configurable service, you can change the connection details for an adapter without the need to redeploy the adapter. For more details about creating, changing, reporting, and deleting the configurable services for Siebel, see "Changing connection details for Siebel adapters" on page 720.

You can also connect to different versions of Siebel by creating a custom EISProviders configurable service and setting the location of the appropriate library files. For more information, see "Connecting to different versions of Siebel" on page 2077.

Terminals and properties:
When you have put an instance of the SiebelRequest node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view. If you double-click a SiebelRequest node, you open the Adapter Connection wizard. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The SiebelRequest node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts the request business object.
Out	The output terminal to which the response business object is sent if it represents successful completion of the request, and if further processing is required within this message flow.
Failure	If an error happens in the SiebelRequest node, the message is propagated to the Failure terminal. Information about the error, and business object events can also be propagated to the Failure terminal.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk on the panel if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The SiebelRequest node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, for example, SiebelRequest	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The SiebelRequest node Basic properties are described in the following table.

Property	M	C	Default	Description	`mqsiapplybaroverride` command property
Primary adapter component	Yes	No		The name of the adapter component that contains configuration properties for the adapter. Either enter a name of an adapter file, or click **Browse** to select an adapter file from the list of files that are available in referenced message set projects. When theSiebelRequest node receives data from the Siebel system, it associates that data with a method name. The SiebelRequest node attempts to call methods that are defined in the primary adapter. If the method is not defined in the primary adapter, the node can call methods that are defined in matching secondary adapters that are deployed to the same execution group.	
Secondary adapter mode	No	Yes	None	Specifies whether the node can call methods that are defined in secondary adapters. If you set the `Secondary adapter mode` property to `None`, the SiebelRequest node calls only methods that are defined in the primary adapter. If the method is not defined in the primary adapter, an error occurs. If you set this property to `All adapters in execution group`, the node can call methods that are defined in any Siebel outbound adapter that is deployed to the same execution group.	secondaryAdapterMode

Property	M	C	Default	Description	mqsiapplybaroverride command property
Default method	Yes	Yes		The default method binding to use.	defaultMethod

The SiebelRequest node Response Message Parsing properties are described in the following table.

Property	M	C	Default	Description
Message domain	No	No	DataObject	The domain that is used to parse the response message. By default, the response message that is propagated from the SiebelRequest node is in the DataObject domain. You cannot specify a different domain.
Message set	Yes	No	Set automatically	The name of the message set in which the incoming message is defined. This field is set automatically from the Adapter component property. If you set this property, then subsequently update the project dependencies to remove this message set reference, a warning is issued. Either update the `Message set` property, or restore the reference to this message set project.
Message type	No	No		The name of the response message. The node detects the message type automatically. You cannot set this property.
Message format	No	No		The name of the physical format of the response message. You cannot set this property.

The SiebelRequest node Transactionality properties are described in the following table.

Property	M	C	Default	Description
Transaction mode	No	No	No	This property specifies that updates are performed independently, not as part of a local transaction. You cannot change this property.

The SiebelRequest node Request properties are described in the following table.

Property	M	C	Default	Description
Method Location	Yes	No	$LocalEnvironment/ Adapter/MethodName	The location of the business method (such as createPurchaseOrder or deletePurchaseOrder) that is used to trigger the SiebelRequest node to perform an action on the external system.
Data Location	Yes	No	$Body	The location in the incoming message tree from which data is retrieved to form the request that is sent from the SiebelRequest node to the EIS.

The SiebelRequest node Result properties are described in the following table.

Property	M	C	Default	Description
Output data location	No	No	$OutputRoot	The message tree location to which the SiebelRequest node sends output.

Property	M	C	Default	Description
Copy local environment	No	No	Selected	This property controls how the local environment is copied to the output message. If you select the check box, at each node in the message flow, a new copy of the local environment is created in the tree, and it is populated with the contents of the local environment from the preceding node. So if a node changes the local environment, the upstream nodes do not see those changes because they have their own copies. This behavior might be an issue if you are using a FlowOrder node, or if you use the propagate command on a Compute node. If you clear the check box, each node does not generate its own copy of the local environment, but it uses the local environment that is passed to it by the previous node. So if a node changes the local environment, those changes are seen by the upstream nodes.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Overview of WebSphere Adapter for Siebel Business Applications" on page 2002
With WebSphere Adapter for Siebel Business Applications, you can create integrated processes that exchange information with a Siebel application, without special coding.

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

"DataObject parser and domain" on page 1114
Use the DataObject domain to parse and write messages for WebSphere Adapters and CORBA applications.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

SOAPAsyncRequest node

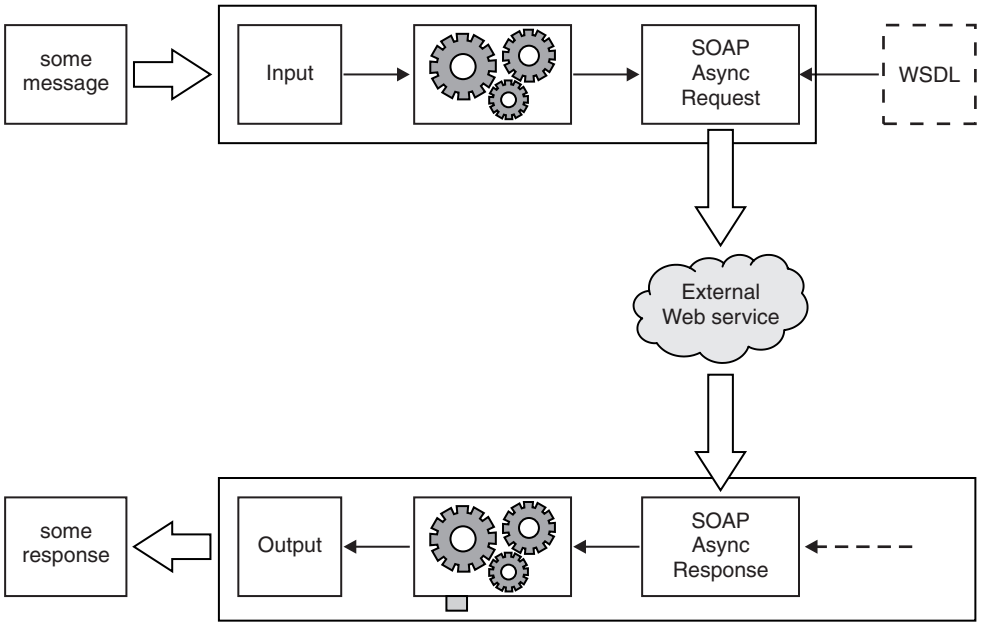
Use the SOAPAsyncRequest node with the SOAPAsyncResponse node to construct a pair of message flows that call a Web service asynchronously.

This topic contains the following sections:

Purpose:

The SOAPAsyncRequest node can use HTTP or JMS transport. It is linked as a pair with a SOAPAsyncResponse using a unique identifier, and WS-Addressing, to correlate response messages with the original request. The SOAPAsyncRequest and SOAPAsyncResponse nodes cannot be used with one-way operations.

The SOAPAsyncRequest node sends a Web service request, but the node does not wait for the associated Web service response to be received. This asynchronous functionality enables multiple outbound requests to be made almost in parallel because the outbound request is not blocked waiting for the response. However, when using HTTP transport, the SOAPAsyncRequest node does wait for the HTTP 202 acknowledgment before continuing with the message flow, and the SOAPAsyncRequest node blocks if the acknowledgment is not received. The Web service response is received by the SOAPAsyncResponse node, which can be in a separate message flow. The nodes are used as a pair, and correlate responses against the original requests.



The SOAPAsyncRequest node is the first half of the asynchronous request and response node pair. The SOAPAsyncRequest node calls a remote SOAP-based Web service. The request is sent by the SOAPAsyncRequest node, but the SOAPAsyncRequest node does not receive the response. The response is received by a SOAPAsyncResponse node that is running on a different thread. The SOAPAsyncResponse node is typically at the beginning of a different message flow; however, it must be in the same execution group as the SOAPAsyncRequest node.

The SOAPAsyncRequest node is WSDL-driven, in a similar manner to the SOAPRequest node.

The SOAPAsyncRequest node is contained in the **Web Services** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:
A SOAPAsyncRequest node is linked as a pair with a SOAPAsyncResponse using a unique identifier to correlate request and response messages.

The following sample demonstrates how to use the asynchronous SOAP nodes when you call a Web service. The Web service simulates an order service, and the client shows how existing WebSphere MQ interfaces can be extended to make Web service requests.

- Asynchronous Consumer

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Using WSDL with the SOAPAsyncRequest node:
A SOAPAsyncRequest node must be associated with a WSDL file unless it is operating in gateway mode. For more information about gateway mode, see "Gateway operation mode for SOAP nodes" on page 1645.

In WSDL mode, when you select a WSDL file for the `WSDL file name` field, the WSDL is validated to ensure that it is WS-I compliant. If the WSDL uses a SOAP/JMS transport URI it is not WS-I compliant, but by default no error is shown. To enable strict WS-I validation and display a warning when a SOAP/JMS transport is used, click **Window > Preferences > Broker Development > Message Sets > Validation** and clear the **WS-I BP 1.1: Allow SOAP/JMS as transport URI** check box.

After a valid WSDL file is selected, the message set project to which WSDL file belongs is added as a referenced project to the corresponding flow project, if the reference does not exist. If the WSDL file is not valid, or an incorrect file name is entered, an error message is displayed in the Properties view and all WSDL properties are blank.

If the node was created by dropping a WSDL file from a message set onto the message flow editor, this property is preset to the name of the WSDL file. If the name of the WSDL file is not preset, you can set this property in one of the following ways.

- If you have Deployable WSDL, you can select from the Deployable WSDL files by clicking **Browse**.
- If you have WSDL definitions, but no message set, then you can create a message set:
  1. Click **Browse** to open the WSDL Selection window.
  2. Click **Import/Create New** to open the Import WSDL file wizard.
  3. Enter the message set name and message set project name. Click **Next**.
  4. Select the relevant option:
     - If your WSDL file exists in your workspace, select **Use resources from the workspace**, and select the WSDL file.
     - If your WSDL file is in the file system, select **Use external resources**. Select the WSDL file. Click **Next**.
  5. Select the WSDL bindings to import. Any warnings or errors are displayed in the wizard banner.
  6. Click **Finish**. Result: Creates a new message set project and message set, with message definitions. The WSDL definitions are added to the Deployable WSDL folder.
  7. You can now select the WSDL file from the WSDL Selection window. Click **OK**.
- If you have a message set but no WSDL definition, you must generate a WSDL definition. See "Generating a WSDL definition from a message set" on page 2968.
- Drag a WSDL file from a message set onto the node.
- Type in a file name that is relative to the message set project in which the deployable WSDL file exists.

When you save the flow file, it is validated that the `WSDL file name` exists in the message set. If it is does not, an error is generated, and you will not be able to add a flow that contains this SOAPAsyncRequest node to the broker archive (BAR) file.

Terminals and properties:
The SOAPAsyncRequest node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a SOAP request message for dispatch to the target by the node.
Failure	The output terminal to which the message is routed if a failure is detected when the SOAP request message is dispatched to the target (such as a message validation failure).
Out	The output terminal to which the message is routed if it has been successfully dispatched to the target, and if further processing is required within this message flow. The message that leaves the Out terminal is the same as the message that arrived at the In terminal.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined). The column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

Some SOAPAsyncRequest node properties are initially set from properties in the imported WSDL. These properties are parsed differently depending on which URI format is used by the `address` element in the WSDL. For details, see "WSDL URI formats for JMS" on page 1668.

The SOAPAsyncRequest node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type	The name of the node.
Short description	No	No	None	A brief description of the node.
Long description	No	No	None	Text that describes the purpose of the node in the message flow.

The SOAPAsyncRequest node Basic properties are described in the following table.

Property	M	C	Default	Description	**`mqsiapplybaroverride` command property**
Unique identifier	Yes	No		This property is the unique identifier that links a pair of SOAPAsyncRequest and SOAPAsyncResponse nodes. When using HTTP transport, this identifier is used as a unique URL fragment to identify incoming response messages for the SOAPAsyncResponse node. When using JMS transport, this property is used as a unique identifier only if the `Get Response By Correl ID` property is checked. If the `Get Response By Correl ID` property is cleared and JMS transport is used, the `Reply To Destination` queue acts as the unique identifier instead, and therefore must be unique to this pair of nodes.	asyncResponseCorrelator

Property	M	C	Default	Description	mqsiapplybaroverride command property
Operation mode	Yes	Yes	`Invoke a specific web service defined by a WSDL interface`	This property allows you to specify the operation mode of the node, which determines whether it acts in WSDL mode or in gateway mode. In WSDL mode, the node performs operations according to the WSDL it is configured with. However, gateway mode allows you to configure your flow to handle generic SOAP request/response and one-way messages, or to act as a façade between multiple web services clients and multiple back-end web services providers. **Invoke a specific web service defined by a WSDL interface** Configure the node with a deployable WSDL by setting the WSDL file name property or by dragging a WSDL onto the node. This is the default option. **Invoke a generic web service** Configure the node to act in gateway mode with no WSDL required. See "Gateway operation mode for SOAP nodes" on page 1645 for a fuller explanation of gateway mode.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
WSDL file name	Yes	No	None	This property indicates the location of the WSDL file that you want to use to configure the node. Enter the full path to the WSDL file, or click **Browse** to locate the WSDL file in your workspace. When you select a WSDL file for the `WSDL file name` property, the WSDL is validated to ensure that it is WS-I compliant. If the WSDL has a binding using SOAP/JMS which is not WS-I compliant, by default no error is shown. To enable strict WS-I validation and display a warning when a SOAP/JMS transport is used, click **Window** > **Preferences** > **Broker Development** > **Message Sets** > **Validation** and clear the **WS-I BP 1.1: Allow SOAP/JMS as transport URI** check box. Only deployable WSDL files can be used to configure the SOAP nodes. After a valid WSDL file is selected, the message set project to which the WSDL file belongs is added as a referenced project to the corresponding message flow project, if the reference does not exist. If the WSDL file is not valid, or an incorrect file name is entered, an error message is displayed in the Properties view and all WSDL properties are blank. If the node was created by dropping a WSDL file from a message set onto the Message Flow editor, this property is preset to the name of the WSDL file. This property takes a string value. The following situations result in an error condition: • The WSDL file does not belong to a message set project, or the WSDL file was not imported correctly; see "Importing from WSDL" on page 2946 and "Importing WSDL definitions from the command line" on page 2948. • The WSDL file contains no HTTP or JMS bindings. • The WSDL file contains no port type. • The WSDL file that is specified in the field does not exist. WSDL properties are disabled when the node is configured to act in gateway mode.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Port type	Yes	No	By default, the first `Port type` found in the WSDL file, that has an associated HTTP or JMS binding with it, is selected.	This property type is String. This field lists all the port types defined by the specified WSDL file. By default, the first port type found in the WSDL file that has an associated HTTP or JMS binding is selected. When you save the flow file, it is validated that the selected `Port type` is valid within the content of the selected WSDL file. If it is not valid, an error is generated, and you will not be able to add a flow that contains this SOAPAsyncRequest node to the broker archive (BAR) file. Error Conditions: • Selected `Port type` does not contain at least one operation. WSDL properties are disabled when the node is configured to act in gateway mode.	
Imported binding	Yes	No		This property type is String. The `Imported binding` box lists all the SOAP bindings associated with the selected port type. Only HTTP or JMS transport is supported. Bindings are listed in the order that they are displayed in the WSDL file. By default, the first binding that implements the operation and has an associated service port is selected. This property is updated every time the `Port type` value changes. Error Conditions: • No SOAP bindings (with HTTP or JMS transport) in the WSDL file are associated with the `Port type`. • Selected binding does not have any operations. WSDL properties are disabled when the node is configured to act in gateway mode.	
Binding operation	Yes	No		This property type is String. The `Binding operation` box contains all the operations defined by the selected binding. The first operation in the list is selected by default. This property is updated every time the selected binding value changes.When you save the flow file, it is validated that the selected `Binding operation` is valid within the content of the selected WSDL file. If it is not valid, an error is generated, and you will not be able to add a flow that contains this SOAPAsyncRequest node to the broker archive (BAR) file. WSDL properties are disabled when the node is configured to act in gateway mode.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Service port	Yes	No		This property type is String. The `Service port` box lists all the service ports that point to the selected binding. The first service port for the binding is selected by default. This property is updated every time the selected binding value changes. When you save the flow file, it is validated that the selected `Service port` is valid within the content of the selected WSDL file. If it is not valid, an error is generated, and you will not be able to add a flow that contains this SOAPAsyncRequest node to the broker archive (BAR) file. Error Conditions: • No ports point to the selected binding. WSDL properties are disabled when the node is configured to act in gateway mode.	
Target namespace	No	No		This property type is String. `Target namespace` displays the namespace of the selected WSDL file. WSDL properties are disabled when the node is configured to act in gateway mode.	
Transport	No	No	HTTP	This property is set automatically when the `Imported binding` property is selected. The value of this property shows the transport used by the selected WSDL binding; for example, `HTTP` or `JMS`. If you choose to switch the transport from `JMS` to `HTTP`, a dialog box displays, which allows you to reset the JMS-specific properties. You must reset the JMS properties to deploy the message flow to a runtime environment version prior to fix pack V7.0.0.1.	

The SOAPAsyncRequest node HTTP Transport properties are described in the following table. These settings are used only when the node uses HTTP transport.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Web service URL	Yes	No		This property type is String. This property is automatically derived from the <soap:address> element of the selected Service port. Whenever the selected port is updated, the Web service URL is updated accordingly. However, if you override the value then your value persists and the URL is no longer updated from the service port. If you choose to override this property you must specify it in the form http://<hostname>[:<port>]/[<path>] where: • http://<hostname> must be specified. • <port> has a default of 80. If you specify a value, you must include the colon (:) before the port number. • <path> has a default of forward slash (/). If you specify a value, you must include the forward slash (/) before the path. For more details of how to override this property, see Changing the default URL for a SOAPRequest node or a SOAPAsyncRequest node request.	webServiceURL
Request timeout (in seconds)	No	Yes	120	This property type is Integer. This property has the value of the wait time for the remote server to respond with an acknowledgment that the message has been received. The time in seconds that the node waits for a response from the Web service. The valid range is 1 to (2^{31})-1. You cannot enter a value that represents an unlimited wait.	requestTimeout
HTTP(S) proxy location	No	Yes		This property type is String. The location of the proxy server to which requests are sent. This value must be in the form hostname:port.	httpProxyLocation

Property	M	C	Default	Description	mqsiapplybaroverride command property
SSL Protocol (if using SSL)	No	Yes	TLS	This property type is Enumerate. The SSL protocol to use when making an HTTPS request. Valid values are: **SSL** This option attempts to connect by using the SSLv3 protocol first, but the handshake can fall back to the SSLv2 protocol where the SSLv2 protocol is supported by the underlying JSSE provider. **SSLv3** This option attempts to connect with the SSLv3 protocol only. The handshake cannot fall back to SSLv2. **TLS** The default. This option attempts to connect with the TLS protocol only. The handshake cannot fall back to SSLv3 or SSLv2. **TLSv1** This option attempts to connect with the TLS v1.0 protocol only. Fallback to SSLv3 or SSLv2 is not allowed. **TLSv1.1** This option attempts to connect with the TLS v1.1 protocol only. Fallback to SSLv3, SSLv2, or TLSv1.0 is not allowed. **TLSv1.2** This option attempts to connect with the TLS v1.2 protocol only. Fallback to SSLv3, SSLv2, TLSv1.0, or TLSv1.1 is not allowed. **SSL_TLS** This option enables all SSL v3.0 and TLS v1.0 protocols. Fallback to SSLv2 is not allowed. **SSL_TLSv2** This option enables all SSL v3.0 and TLS v1.0, v1.1, and v1.2 protocols. Fallback to SSLv2 is not allowed. Both ends of an SSL connection must use the same protocol. The protocol must be one that the remote server can accept.	sslProtocol
Allowed SSL ciphers (if using SSL)	No	Yes	Empty	This property type is String. A comma-separated list of ciphers to use when making an SSL request. This setting enables you to specify a single cipher (such as SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA), or a list of ciphers that are the only ones used by the connection. This list of ciphers must include one or more that are accepted by the remote server. The default value is an empty string, which allows the node to use any, or all, of the available ciphers during the SSL connection handshake. This method enables the greatest scope for making a successful SSL connection.	allowedSSLCiphers

Property	M	C	Default	Description	mqsiapplybaroverride command property
Use compression	No	No	None	This property controls whether the content of the HTTP request is compressed. Valid values are none, gzip, zlib (deflate) and deflate. If the request is compressed, the Content-Encoding header is set to indicate that the content is compressed. zlib (deflate) represents RFC 1950 + RFC 1951 combined. deflate represents RFC 1951 only.	requestCompressionType
Accept compressed responses by default	No	Yes	Cleared	This property indicates whether the request accepts compressed responses. If this option is selected, it is possible for the request to receive responses with a Content-Encoding of gzip or deflate. If such a response is received the content is decoded and the Content-Encoding header is removed. If the Request Header does not contain an Accept-Encoding header then selecting this option sets the Accept-Encoding header to "gzip, deflate".	acceptCompressedResponses
Enable SSL certificate hostname checking	No	Yes	No	This property specifies if the host name of the server that is receiving the request must match the host name in the SSL certificate.	hostnameChecking

The SOAPAsyncRequest node JMS Transport properties are described in the following table. These settings are used only when the node uses JMS transport.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Destination	Yes	Yes	None	The destination to which the node sends outgoing messages. If the SOAPAsyncRequest node is to be used to send point-to-point messages, enter the Destination queue name for the JMS queue name that is listed in the bindings file. This property takes its initial value from a WSDL URI property, depending on whether the WSDL `address` URI is formatted in the W3C (standards) style, or the IBM (proprietary) style. `Destination` is set to the value of `destinationName` found in the WSDL if a W3C-style URI is found, or `destination` if an IBM-style URI is found.	jmsDestination

Property	M	C	Default	Description	mqsiapplybaroverride command property
Reply To Destination	Yes	Yes	None	The name of the JMS destination to which the receiving application must send a reply message. For a reply message to be returned to this JMS destination, the JMS destination name must be known to the domain of the JMS provider that is used by the receiving client. If the `Get Response By Correl ID` property is cleared, this queue uniquely identifies messages destined for the paired SOAPAsyncResponse node. If the `Get Response By Correl ID` property is checked, this queue can then be shared between multiple nodes. This property takes its initial value from a WSDL URI property, depending on whether the WSDL `address` URI is formatted in the W3C (standards) style, or the IBM (proprietary) style. `Reply To Destination` is set to the value of `replyToName` found in the WSDL if a W3C-style URI is found, or to the first of `replyToName`, `replyTo`, `replyToDestination`, or `replyDestination` if an IBM-style URI is found. If any of these other properties are present, they display as a name-value pair in the `User Parameters` table.	jmsReplyToDestination
JMS provider name	Yes	No	`WebSphere MQ`	Select a JMS vendor name from the list, or enter a name of your choice. The name must match the name of a configurable service that is defined for the broker to which you deploy the message flow. When you select a name from the list, the `Initial context factory` property is updated automatically with the relevant Java class. If you enter your own JMS provider name, you must also enter a value for the `Initial context factory`.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Initial context factory	Yes	Yes	com.sun.jndi.fscontext.RefFSContextFactory	The starting point for a JNDI namespace. A JMS application uses the initial context to obtain and look up the connection factory and queue or topic objects for the JMS provider. If you select a JMS provider name from the list in JMS provider name, the Initial context factory property is updated automatically with the relevant Java class. If you enter your own JMS provider name, you must also enter a value for the Initial context factory. The default value is com.sun.jndi.fscontext.RefFSContextFactory, which defines the file-based Initial context factory for the WebSphere MQ JMS provider. This property takes its initial value from a WSDL URI property, depending on whether the WSDL address URI is formatted in the W3C (standards) style, or the IBM (proprietary) style. Initial context factory is set to the value of jndiInitialContextFactory found in the WSDL if a W3C-style URI is found, or initialContextFactory if an IBM-style URI is found.	initialContextFactory

Property	M	C	Default	Description	**mqsiapplybaroverride** command property
JNDI URL bindings location	Yes	Yes		The system path or the LDAP location for the bindings file. The bindings file contains definitions for the JNDI administered objects that are used by the SOAPAsyncRequest node. This property is disabled when the `Initial context factory` is `com.ibm.mq.jms.Nojndi`. When you enter a value for `JNDI URL bindings location`, ensure that it complies with the following instructions: • Construct the bindings file before you deploy a message flow that contains a SOAPAsyncRequest node. • Do not include the file name of the bindings file in this field. • If you have specified an LDAP location that requires authentication, configure the LDAP principal (userid) and LDAP credentials (password) separately. These values are configured at broker level. For information about configuring these values, see "**mqsicreatebroker** command" on page 3831 and "**mqsichangebroker** command" on page 3723. • The string value must include a supported URL prefix that has a URL handler that is available on the class path. For information about constructing the JNDI administered objects bindings file, see the JMS provider documentation. This property takes its initial value from a WSDL URI property, depending on whether the WSDL `address` URI is formatted in the W3C (standards) style, or the IBM (proprietary) style. `JNDI URL bindings location` is set to the value of `jndiURL` found in the WSDL if a W3C-style URI is found, or `jndiProviderURL` if an IBM-style URI is found.	locationJndiBindings

Property	M	C	Default	Description	mqsiapplybaroverride command property
Connection factory name	Yes	Yes		The name of the connection factory that is used by the SOAPAsyncRequest node to create a connection to the JMS provider. This property is initially configured from the imported WSDL. This name must exist in the bindings file. The `Connection factory name` must be a JMS QueueConnectionFactory. This property takes its initial value from a WSDL URI property, depending on whether the WSDL `address` URI is formatted in the W3C (standards) style, or the IBM (proprietary) style. `Connection factory name` is set to the value of `jndiConnectionFactoryName` found in the WSDL if a W3C-style URI is found, or `connectionFactory` if an IBM-style URI is found.	connectionFactoryName
User Parameters	No	No		This table describes user properties that will be sent in the `requestURI` property of the outgoing request message. The properties are name-value pairs that exist in the WSDL and are not used by other properties of the SOAPAsyncRequest node.	
JNDI parameters	No	No		A table mapping JNDI context parameters to their type. These properties take their initial values from any W3C-style WSDL properties starting with `jndi-`. IBM-style WSDL does not support JNDI parameters, but you can set these properties on the node.	
Get Response By Correl ID	No	Yes	Cleared	If this property is checked, the SOAPAsyncRequest node sends the request message with the Correl ID specified in the `Unique identifier` property, and the SOAPAsyncResponse node receives only response messages that match that Correl ID. This allows a single `Reply To Destination` queue to be shared between several pairs of SOAPAsyncRequest and SOAPAsyncResponse nodes, if this property is checked for all those nodes. This functionality works only with web service providers that support reading the Correl ID from the request message and using it as the Correl ID in the response message.	
Backout destination	No	Yes		The SOAPAsyncResponse node sends response messages to this destination when errors prevent the response message flow from processing the response message. The message is removed from the reply to destination. The backout properties are set by the SOAPAsyncRequest node, but used only by its paired SOAPAsyncResponse node.	backoutDestination

Property	M	C	Default	Description	mqsiapplybaroverride command property
Backout threshold	No	Yes	0	This value controls when a message is put to the backout destination. For example, if the value is 3, the JMS provider attempts to deliver the message to the input destination three times. After the third attempted delivery, the message is not rolled back to the reply to destination and is sent to the Backout destination. The backout properties are set by the SOAPAsyncRequest node, but used only by its paired SOAPAsyncResponse node. See "Configuring the backout threshold property" on page 4770.	

The SOAPAsyncRequest node Message Delivery properties are described in the following table. This sub tab is enabled only if the selected binding in the Basic tab uses JMS transport.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Target Service	No	No	None	Used by the SOAPAsyncRequest node when dispatching the service request. This property takes its initial value from the `targetService` WSDL property.	targetService
Delivery mode	No	Yes	Persistent	This property controls the persistence mode that a JMS provider uses for a message. Valid values are: • `Persistent`: the message survives if the JMS provider has a system failure. • `Non Persistent`: the message is lost if the JMS provider has a system failure. This property takes its initial value from a WSDL URI property, depending on whether the WSDL `address` URI is formatted in the W3C (standards) style, or the IBM (proprietary) style. `Delivery mode` is set to the value of `deliveryMode` found in the WSDL if a W3C-style URI is found, or to the first of `deliveryMode` or `persistence` if an IBM-style URI is found. If both these properties are present, the second property displays as a name-value pair in the `User Parameters` table.	deliveryMode

Property	M	C	Default	Description	mqsiapplybaroverride command property
Message Priority	No	Yes	4	This property assigns relative importance to the message and can be used for message selection by a receiving web service. Select a value between 0 (lowest priority) and 9 (highest priority). The default value is 4, which indicates medium priority. Priorities in the range 0 - 4 indicate typical delivery. Priorities in the range 5 - 9 indicate faster delivery. This property takes its initial value from a WSDL URI property, depending on whether the WSDL `address` URI is formatted in the W3C (standards) style, or the IBM (proprietary) style. `Priority` is set to the value of `priority` found in the WSDL if a W3C-style URI is found, or to the first of `priority` or `Priority` if an IBM-style URI is found. If both these properties are present, the second property displays as a name-value pair in the `User Parameters` table.	messagePriority
Message Expiration (ms)	No	Yes	0	This property controls the length of time, in milliseconds, for which the JMS provider keeps the output JMS message. The default value, 0, is used to indicate that the message must not expire. This property takes its initial value from the `timeToLive` WSDL property.	messageExpiration
Message Type	No	Yes	bytes	Select a value from the list to configure the type of JMS message that is produced by the SOAPAsyncRequest node. Valid values are `text` and `bytes`.	messageType

The SOAPAsyncRequest node Transactions properties are described in the following table. This setting does not apply when the node uses HTTP transport.

Property	M	C	Default	Description
Transaction Mode	Yes	No	Automatic	This property controls whether the message is output under a JMS transaction. Valid values are `Yes`, `No`, and `Automatic`. Select `No` to output the message using a non-transactional JMS session. Select `Yes` to output the message using a transactional JMS session. The JMS transaction can be either local or global. To use a global transaction, using an XA JMS session, you must also select the message flow property `Coordinated Transaction` in the BAR file properties. Select `Automatic` if you want the message transactionality to be inherited from the `Transaction mode` setting on the Input node at the start of the flow. See “Configuring for coordinated JMS transactions” on page 4770.

The SOAPAsyncRequest node Advanced properties are described in the following table.

SOAP headers that are part of the must understand headers list are incorporated into the flow rather than causing a SOAP fault. Adding headers to the must understand headers list stops SOAP faults being generated by SOAP headers.

You do not need to add must understand headers for WS-Addressing and WS-Security because these are understood if you configure **WS Extensions**.

The must understand headers list that is configured on this node is applied to the corresponding SOAPAsyncResponse node when the SOAPAsyncResponse node receives the reply from the remote server.

Property	M	C	Default	Description
WSDL-defined SOAP response headers	No	No		The WSDL-defined SOAP response headers table is read-only, and is populated based on the SOAP headers defined in the output part of the selected operations. By default, the check boxes, in the second column of the table, are cleared for all entries in the WSDL-defined SOAP response headers table. You must select the relevant check box to add the header to the must understand headers list. When the node is configured to act in gateway mode, with no WSDL required, this table is cleared. The original values of these fields are restored if the operation mode of the node is changed back to WSDL mode.
User-defined SOAP response headers	No	No		You can add custom headers (headers that are not defined in the WSDL file) in the User-defined SOAP headers table. Use **Add**, **Edit**, and **Delete** for this table. You must select the relevant check box, in the second column of the table, to ensure that the newly added custom header is added to the must understand headers list.

The SOAPAsyncRequest node WS Extensions properties are described in the following table.

Property	M	C	Default	Description	
Use WS-Addressing	No	No	Selected	You cannot edit this property. This property indicates that WS-Addressing is always engaged on the SOAPAsyncRequest node. For more details about WS-Addressing with the SOAPAsyncRequest node, see "WS-Addressing with the SOAPAsyncRequest and SOAPAsyncResponse nodes" on page 1655.	
Allow MTOM	No	Yes	No	This property controls whether MTOM is enabled for the parser. Valid values are Yes, No, and Inherit. For more information about using SOAP MTOM with the SOAPReply, SOAPRequest, and SOAPAsyncRequest nodes; see "Using SOAP MTOM with the SOAPReply, SOAPRequest, and SOAPAsyncRequest nodes" on page 1678. For more information about MTOM, see "SOAP MTOM" on page 6697. MTOM support is disabled when the node is configured to act in gateway mode.	allowMTOM

Property	M	C	Default	Description	
WS-Security	No	No		This table and features two columns: • `Alias` • `XPath Expression` You can add XPath expressions with an associated `Alias` value to the `WS-Security` table. The `Alias` is resolved in a Policy Set that is created by the administrator. The Policy Set resolves the `Alias` to either encrypt or sign the part of the message referred to by the `XPath Expression`. You can **Add**, **Edit**, and **Delete** in this table.	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

LocalEnvironment overrides:
You can dynamically override set values in the local environment in the same way as setting values in other elements of a message. For a full list of values you can override in the local environment, see Local environment overrides.

Working with WrittenDestination data:
After the request has been made, the WrittenDestination folder in LocalEnvironment is updated with the WS-Addressing, compression details (if in use), and transport details. A WrittenDestination for a SOAPAsyncRequest node has the following format, with WS-Addressing and Compression present only if it is used:

```
WrittenDestination = (
   SOAP  = (
      Request = (
         WSA = (
            To = 'URI'
            ReplyTo = 'http://server:7800/reply'
            MessageID = 'id'
            Action = 'doAllTheStuff'
         )
         Transport = (
            HTTP = (
               WebServiceURL = 'http://server:8080/service'
               Compression   = (
                  OriginalSize   = 775
                  CompressedSize = 411
               )
            )
      )
   )
)
```

The following example uses JMS transport:

```
WrittenDestination = (
   SOAP  = (
      Request = (
         WSA = (
            To = 'URI'
            ReplyTo = 'http://server:7800/reply'
            MessageID = 'id'
            Action = 'doAllTheStuff'
         )
         Transport = (
            JMS = (
               Destination = 'jms:jndi:B2BQUEUEIN'
            )
         )
      )
   )
)
```

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"SOAP parser and domain" on page 1082
You can use the SOAP parser to create a common WSDL-based logical tree format for working with Web services, independent of the physical bitstream format.

"SOAP tree overview" on page 1611
This tree format allows you to access the key parts of the SOAP message in a convenient way.

"XMLNSC parser" on page 1090
The XMLNSC parser is a flexible, general-purpose XML parser that offers high performance XML parsing and optional XML Schema validation.

"Using compression with HTTP and SOAP nodes" on page 1597
You can configure HTTP and SOAP nodes to use HTTP compression and decompression when sending and receiving messages.

Related tasks:

"Configuring authentication with HTTP basic authentication" on page 451
Use a security profile to configure HTTP basic authentication in the HTTPRequest or SOAPInput nodes.

"Validating messages" on page 1478
The broker provides validation based on the message model for predefined messages.

"Manipulating messages in the XMLNSC domain" on page 2546
If you are writing ESQL to process messages in the XMLNSC domain, it is helpful to learn about the structure of the message tree that the XMLNSC parser builds.

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Importing from WSDL" on page 2946
You can use the New Message Definition File wizard in the WebSphere Message

Broker Toolkit to create a new message definition from WSDL.

Related reference:

"Configuring the backout threshold property"
You can set the backout threshold property on nodes that use JMS transport to specify how many attempts are made to deliver the message to the input destination.

"Configuring for coordinated JMS transactions" on page 4544
Configure your message flow to receive or output messages under coordinated transactions.

"Parsing on demand" on page 4173
On-demand parsing, referred to as partial parsing, is used to parse an input message bit stream only as far as is necessary to satisfy the current reference. The parsers that can perform partial parsing of input messages are the JSON, MRM, XML, XMLNS, and XMLNSC parsers.

"Validation properties" on page 4169
You can control validation by setting properties on the Validate and Parser Options tabs for the nodes that are listed in the following table.

"SOAPAsyncResponse node" on page 4777
Use the SOAPAsyncResponse node in conjunction with the SOAPAsyncRequest node to construct a pair of message flows that call a Web service asynchronously.

"WS-Addressing with the SOAPAsyncRequest and SOAPAsyncResponse nodes" on page 1655
The remote Web service must understand WS-Addressing to be able to work with SOAPAsyncRequest and SOAPAsyncResponse nodes.

Configuring the backout threshold property:

You can set the backout threshold property on nodes that use JMS transport to specify how many attempts are made to deliver the message to the input destination.

If the Backout threshold is set to `0` then redelivery is not attempted. If the Backout threshold is 1 or greater, the message will be redelivered the specified number of times.

Set the value of Backout threshold depending on the capabilities of the JMS provider.

If the JMS provider supports `JMSXDeliveryCount`, you can set the Backout threshold to any value.

If the JMS provider does not support `JMSXDeliveryCount`, the Backout threshold must only be set to 0 or 1. If the JMS provider does not support `JMSXDeliveryCount` and the value is set to greater than 1, a redelivered message is repeatedly backed out and reprocessed, and is never delivered to the backout destination.

Configuring for coordinated JMS transactions:

Configure your message flow to receive or output messages under coordinated transactions.

When you include a node using JMS transport in a message flow, such as the JMSInput or SOAPInput node when using JMS transport, the value that you set for `Transaction mode` defines whether messages are received under sync point.

- If you set this property to `Global`, the message is received under external sync point coordination; that is, within a WebSphere MQ unit of work. Any messages that are sent later, by an output node in the same instance of the message flow, are put under sync point, unless the output node overrides this setting explicitly.
- If you set this property to `Local`, the message is received under the local sync point control of the node. Any messages that are sent later, by an output node in the flow, are not put under local sync point, unless an individual output node specifies that the message must be put under local sync point.
- If you set this property to `None`, the message is not received under sync point. Any messages that are sent later, by an output node in the flow, are not put under sync point, unless an individual output node specifies that the message must be put under sync point.

To receive messages under external sync point, you must take additional configuration steps, which need be applied only the first time that a specific node using JMS transport is deployed to the broker for a particular JMS provider.

- On distributed systems, the external sync point coordinator for the broker is WebSphere MQ. Before you deploy a message flow in which the `Transaction mode` property is set to `Global` or `Yes`, and is intended to use globally coordinated transactions, modify the queue manager `.ini` file to include extra definitions for each JMS provider resource manager that participates in globally coordinated transactions.
  - Windows On Windows:
    1. Start WebSphere MQ Explorer.
    2. Right-click the queue manager name in the left pane and click **Properties**.
    3. Click **XA resource managers** in the left pane.
    4. Click **Add...** .
    5. Set the options as follows:
       - Set `Name` to any value.
       - On Windows on x86 systems, set the `SwitchFile` property to *`install_dir`*`\bin\JMSSwitch.dll`. On Windows on x86-64 systems, set the `SwitchFile` property to `JMSSwitch.dll`.
       - Set the `XAOpenString` property to a string value as follows: *`Initial Context`*`,`*`location JNDI`*`,`*`Optional_parms`*.
       - Set the `ThreadOfControl` property to `Thread`.
    6. On Windows on x86-64 systems only, copy the switch file *JMSSwitch32.dll* to the `\exits` subdirectory in the WebSphere MQ installation directory, and rename it to `JMSSwitch.dll`. Copy the switch file `JMSSwitch.dll` to the `\exits64` subdirectory in the WebSphere MQ installation directory.

    For more information, see the *System Administration Guide* section of the WebSphere MQ Version 7 Information Center online.
  - Linux UNIX On Linux and UNIX systems, add a stanza to the queue manager `.ini` file for each JMS provider.

    For example:

    ```
    XAResourceManager:
    Name=Jms_Provider_Name
    SwitchFile=/install_dir/bin/ JMSSwitch.so
    XAOpenString=Initial Context,location JNDI,Optional_parms
    ThreadOfControl=THREAD
    ```

    Where:

Name is an installation defined name that identifies a JMS provider resource manager.

SwitchFile
is the file system path to the JMSSwitch library that is supplied in the `bin` directory of the broker.

XAOpenString can have the following values:

– *Initial Context* is the value that is set in the JMSInput node property `Initial context factory`.

– *location JNDI* is the value that is set in the JMSInput node property `Location JNDI bindings`. This value must include a supported URL prefix that has a URL handler that is available on the class path.

The following parameters are optional:

– `LDAP Principal` matches the value that is set for the broker by using the **mqsicreatebroker** or **mqsichangebroker** commands.

– `LDAP Credentials` matches the value that is set for the broker by using the **mqsicreatebroker** or **mqsichangebroker** commands.

– `Recovery Connection Factory Name` is the JNDI administered connection factory that is defined in the bindings file. If a value is not specified, you must add a default value for `recoverXAQCF` to the bindings file. In either case, the Recovery Connection Factory must be defined as an XA Queue Connection Factory for the JMS provider that is associated with the `Initial context factory`.

The optional parameters are comma-separated and are positional. Therefore, any parameters that are missing must be represented by a comma. For example:

```
com.sun.jndi.fscontext.RefFSContextFactory,file:/C:/webservices/SOAP/JMS/JNDIXA,,,QCF
```

1. Update the Java CLASSPATH environment variable for the queue manager of the broker to include a reference to `xarecovery.jar`; for example:

   ```
   install_dir/classes/xarecovery.jar
   ```

2. Update the Java PATH environment variable for the queue manager of the broker to point to the bin directory in which the SwitchFile is located; for example:

   ```
   install_dir/bin
   ```

Finally, ensure that you have taken the following configuration steps:

– In the message flow, ensure that the coordinated property is enabled by using the WebSphere Message Broker Archive editor.

– Ensure that each node that must be part of the XA transaction is set to the global transaction mode.

– Ensure that the service ID that is used for the broker and the queue manager is the same user ID.

– Ensure that the JNDI connection factory objects that the JMS nodes use for a global transaction are configured to be of the type MQXAConnectionFactory, MQXAQueueConnectionFactory, or MQXATopicConnectionFactory.

  - If you create the bindings using WebSphere Message Broker Explorer, ensure the `Support XA Transactions` option is checked when you define your connection factory.
  - If you create the bindings using JMSAdmin, use the command **DEF XAQCF** or **DEF XATCF**, instead of **DEF QCF** or **DEF TCF**, when you define your connection factory.

For more information, see the *System Administration Guide* section of the WebSphere MQ Version 7 Information Center online.

- z/OS On z/OS, the external sync point manager is Resource Recovery Services (RRS). The only JMS provider that is supported on z/OS is WebSphere MQ JMS. The only transport option that is supported for WebSphere MQ JMS on z/OS is the bind option.

  Sync point control for the JMS provider is managed with RRS sync point coordination of the queue manager of the broker. You do not have to modify the `.ini` file.

Local environment overrides for the SOAPAsyncRequest node:

You can dynamically override values in the local environment in the same way as setting values in other elements of a message. These local environment overrides are used only by the SOAPAsyncRequest node and not by the SOAPAsyncResponse node.

This topic contains the following sections:

Local environment overrides for the SOAPAsyncRequest node:
You can set the following properties under LocalEnvironment.Destination.SOAP.Request:

Setting	Description

TransportType	Overrides the `Transport` property on the node to switch transport. For example, if the node is configured to use the JMS transport, use the following to switch to HTTP transport:
	`SET OutputLocalEnvironment.Destination.SOAP.Request.TransportType = 'http';`
	
	To switch to JMS transport:
	`SET OutputLocalEnvironment.Destination.SOAP.Request.TransportType = 'jms';`
	
	This overrides only the request transport for this message. The response transport is not changed from the property set on the SOAPAsyncResponse node.

Operation	Overrides the `Operation` property on the node. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Operation = 'myOperation';`
UserContext	You can store context data in the following location in the local environment. The SOAPAsyncResponse node can later retrieve this data. `SET OutputLocalEnvironment.Destination.SOAP.Request.UserContext = 'myData';`

LocalEnvironment overrides for HTTP transport:
You can set the following properties under LocalEnvironment.Destination.SOAP.Request.Transport.HTTP. These properties apply only when using HTTP transport.

You can switch between HTTP and JMS transport using the `TransportType` override, or the WS-Addressing `To` field; see "WS-Addressing information in the local environment" on page 1656.

Setting	Description
WebServiceURL	Overrides the `Web service URL` property on the node. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.HTTP.WebServiceURL = 'http://ibm.com/abc/';`

Setting	Description
RequestURI	Overrides the RequestURI, which is the path after the URL and port. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.HTTP.RequestURI = '/abc/def?x=y&g=h';`
Timeout	Overrides the Request timeout (in seconds) property on the node. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.HTTP.Timeout = 42;` This is the time that the node waits to receive the HTTP 202 acknowledgment, rather than the time to wait for the associated Web service response.
ProxyURL	Overrides the HTTP(S) proxy location property on the node. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.HTTP.ProxyURL = 'my.proxy';`
SSLProtocol	Overrides the SSLProtocol property on the node. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.HTTP.SSLProtocol = 'TLS';` Valid values are: SSL, SSLv3, and TLS.
SSLCiphers	Overrides the Allowed SSL Ciphers (if using SSL) property on the node. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.HTTP.SSLCiphers = 'SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA';`
HTTPVersion	Overrides the HTTPVersion. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.HTTP.HTTPVersion = 'HTTP/1.1';`
Method	Overrides the Method. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.HTTP.Method = 'GET';`

ProxyConnectHeaders	Specifies additional headers that are used if the outbound request is an SSL connection through a proxy. These additional headers are sent with the initial CONNECT request to the proxy. For example, you can send proxy authentication information to a proxy server when you are using SSL. You can send multiple headers but each one must be separated by a carriage return and a line feed (ASCII 0x0D 0x0A), in accordance with RFC2616; for example:
	`DECLARE CRLF CHAR CAST(X'0D0A' AS CHAR CCSID 1208);`
	`SET OutputLocalEnvironment.Destination.HTTP.ProxyConnectHeaders = 'Proxy-Authorization: Basic Zm5lcmJsZTpwYXNzd29yZA==' || CRLF || 'Proxy-Connection: Keep-Alive' || CRLF;`
	
	This setting is used only if the request is an SSL request through a proxy server. To send proxy authentication information for a non-SSL request, specify the individual headers in the HTTPRequestHeader folder, as shown in the following example:
	`SET OutputRoot.HTTPRequestHeader."Proxy-Authorization" = 'Basic Zm5lcmJsZTpwYXNzd29yZA==';`
	`SET OutputRoot.HTTPRequestHeader."Proxy-Connection" = 'Keep-Alive';`

Compression	Overrides the Use compression property on the node. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.HTTP.Compression = 'gzip';`

Local environment overrides for JMS transport:
You can set the following JMS properties in the SOAPAsyncRequest node under LocalEnvironment.Destination.SOAP.Request.Transport.JMS. These properties apply only when using JMS transport.

You can switch between HTTP and JMS transport using the TransportType override, or the WS-Addressing To field; see "WS-Addressing information in the local environment" on page 1656.

Some JMS local environment overrides for the SOAPAsyncRequest node have equivalent properties in the JMSTransport header. If you specify a local environment override, it takes precedence over any equivalent property set in the JMSTransport header.

Setting	Description
CorrelationID	Sets the request message CorrelID. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.JMS.CorrelationID = 'myCorrelID';`
DeliveryMode	Overrides the DeliveryMode property on the node. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.JMS.DeliveryMode = 'NON_PERSISTENT';` Allowed values for this property are PERSISTENT and NON_PERSISTENT. If the UriFormat is ibm, 1 and 0 are additional allowed values for DeliveryMode.
Destination	Overrides the Destination property on the node. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.JMS.Destination = 'REPLYTOQ2';`
DestinationURI	You can override multiple JMS properties at the same time in the local environment using the DestinationURI setting. Properties that you set in this way can be overridden by setting local environment overrides for individual JMS properties as shown in the following tables. `SET OutputLocalEnvironment.Destination.SOAP.Reply.Transport.JMS.DestinationURI = 'jms:jndi:INPUTQ1?jndiConnectionFactoryName=QCF& replyToName=REPLYTOQ2& jndiInitialContextFactory=com.sun.jndi.fscontext.RefFSContextFactory& jndiURL=file:/C:/Webservices/SOAP/JMS/JNDI& userParam1=value1& userParam2=value2& timeToLive=30000';` This local environment override can be set with either a W3C-style or IBM-style URI format. For more information, see "WSDL URI formats for JMS" on page 1668.
Expiration	Overrides the Expiration property on the node. This property is specified in milliseconds. For example, to set an expiration of 100 milliseconds: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.JMS.Expiration = '100';`
MessagePriority	Overrides the MessagePriority property on the node. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.JMS.MessagePriority = '7';`
MessageType	Overrides the MessageType property on the node. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.JMS.MessageType = 'text';` Allowed values for this property are textand bytes.
ProviderName	Overrides the JMS provider name property on the node. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.JMS.ProviderName = 'WebSphere MQ';`
TargetService	Overrides the TargetService property on the node. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.JMS.TargetService = 'testService';`

Setting	Description
TransactionMode	Overrides the Transaction mode property on the node. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.JMS.TransactionMode = 'Yes';` Allowable values for this property are Yes, No and ForceLocal. • No means that no transaction takes place, and is equivalent to None. • Yes means that a local transaction takes place if the flow's Coordinated Transaction is not selected, or a global transaction takes place if the flow's Coordinated Transaction property is selected. • ForceLocal means that a local transaction is always used, even if the flow's Coordinated Transaction property is selected.
UriFormat	Overrides the UriFormat property on the node. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.JMS.UriFormat = 'w3c';` Allowable values for this property are w3c and ibm.

Local environment overrides for JNDI

You can set the following JMS properties in the SOAPAsyncRequest node under LocalEnvironment.Destination.SOAP.Request.Transport.JMS.JNDI. These properties apply only when using JMS transport.

Setting	Description
BindingsLocation	Overrides the JNDI URL bindings location property on the node. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.JMS.JNDI.BindingsLocation = 'file:/C:/Webservices/SOAP/JMS/JNDI';`
ConnectionFactoryName	Overrides the Connection factory name property on the node. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.JMS.JNDI.ConnectionFactoryName = 'QCF';`

ContextParameters	Specify JNDI context parameters in addition to the JNDI context parameters defined on the node. You can define specific JNDI context parameters, for example:
	`SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.JMS.JNDI.ContextParameters.property1 = 'value1';`
	`SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.JMS.JNDI.ContextParameters.property2 = 'value2';`

InitialContextFactory	Overrides the Initial context factory property on the node. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.JMS.JNDI.InitialContextFactory = 'com.sun.jndi.fscontext.RefFSContextFactory';`

UserProperties	Specify user context parameters in addition to the user context parameters defined on the node. You can define specific user context parameters, for example:
	`SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.JMS.JNDI.UserProperties.property1 = 'value1';`
	`SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.JMS.JNDI.UserProperties.property2 = 'value2';`

Local environment overrides for WS-Addressing ReplyTo

You can set the following JMS WS-Addressing ReplyTo properties in the SOAPAsyncRequest node under

LocalEnvironment.Destination.SOAP.Request.Transport.JMS.AsyncReply. Set these properties if you want to override how the remote server locates the JNDI definitions for the response queue.

These properties apply only when using JMS transport. You can switch between HTTP and JMS transport using the TransportType override, or the WS-Addressing To field; see "WS-Addressing information in the local environment" on page 1656.

Setting	Description
BindingsLocation	Overrides the BindingsLocation property in the WS-Addressing ReplyTo. `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.JMS.AsyncReply.JNDI.BindingsLocation 'file:/C:/webservices/SOAP/JMS/JNDI';`
ConnectionFactoryName	Overrides the ConnectionFactoryName property in the WS-Addressing ReplyTo. `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.JMS.AsyncReply.JNDI.ConnectionFactory 'QCF';`

ContextParameters	Specify JNDI context parameters in the WS-Addressing ReplyTo. You can define specific JNDI context parameters, for example:
	`SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.JMS.AsyncReply.JNDI.ContextParameters 'value1';`
	`SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.JMS.AsyncReply.JNDI.ContextParameters 'value2';`

InitialContextFactory	Overrides the InitialContextFactory property in the WS-Addressing ReplyTo. `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.JMS.AsyncReply.JNDI.InitialContextFac 'com.sun.jndi.fscontext.RefFSContextFactory';`

Setting	Description
OneWay	Instructs the node that the inbound message was a one-way message and that no reply message is needed. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Gateway.OneWay = True` If the inbound message used HTTP transport, the node sends an acknowledgment HTTP 202 message. If the inbound message used JMS transport, no response is expected. The outbound message has no reply-to queue, and the node does not wait for a response. For more information, see "One-way messages in Gateway mode" on page 1648.

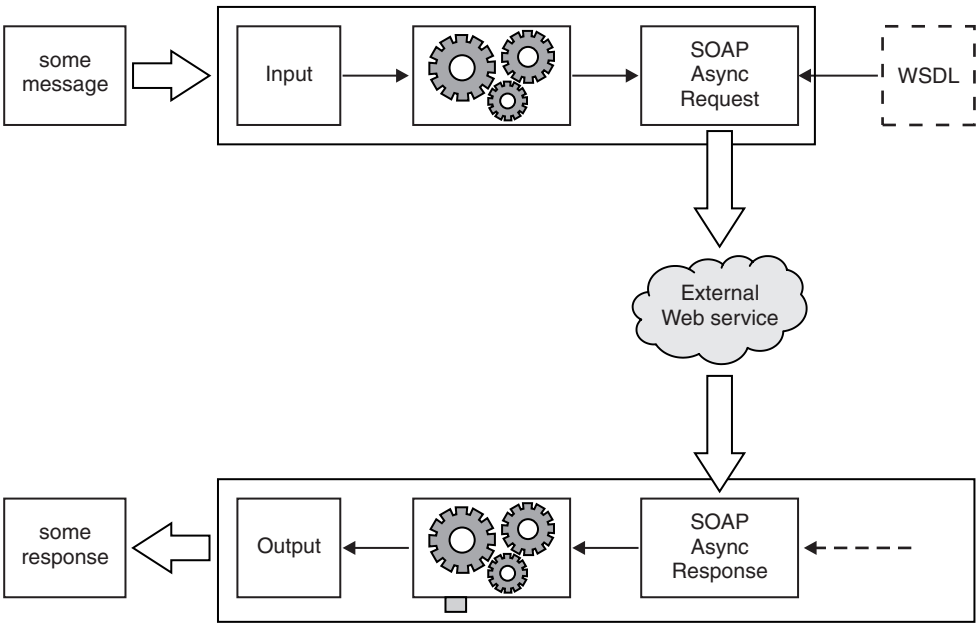
SOAPAsyncResponse node

Use the SOAPAsyncResponse node in conjunction with the SOAPAsyncRequest node to construct a pair of message flows that call a Web service asynchronously.

This topic contains the following sections:

Purpose:
The SOAPAsyncRequest node sends a Web service request, but the node does not wait for the associated Web service response to be received. However, the SOAPAsyncRequest node does wait for the HTTP 202 acknowledgment before continuing with the message flow, and the SOAPAsyncRequest node blocks if the acknowledgment is not received. The Web service response is received by the SOAPAsyncResponse node, which can be in a separate message flow. The nodes

are used as a pair, and correlate responses against the original requests.



The SOAP parser invokes the XMLNSC parser to parse the XML content of the SOAP Web service, and to validate the XML body of the SOAP Web service. The SOAP parser options are passed through to the XMLNSC parser. For more information, see “Manipulating messages in the XMLNSC domain” on page 2546.

The SOAPAsyncResponse node is contained in the **Web Services** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:
Configuration of the SOAPAsyncResponse node is not WSDL-driven, although the 'must understand headers' list configured on the corresponding SOAPAsyncRequest node is applicable to the SOAPAsyncResponse node.

Most configuration options for this node are set on its paired SOAPAsyncRequest node, including the `Backout destination` and `Backout threshold` properties. No SOAP fault is sent when the backout threshold is reached.

You can retrieve context data that has been stored by the SOAPAsyncRequest node from the following location in the local environment:

```
LocalEnvironment.SOAP.Response.UserContext
```

The following sample demonstrates how to use the asynchronous SOAP nodes when you call a Web service. The Web service simulates an order service, and the client shows how existing WebSphere MQ interfaces can be extended to make Web service requests.

- Asynchronous Consumer

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

The SOAPAsyncResponse node can receive a response that has a Content-Encoding of gzip or deflate. When such a response is received, the content is decoded and the Content-Encoding header is removed.

Terminals and properties:
When you have put an instance of the SOAPAsyncResponse node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The SOAPAsyncResponse node terminals are described in the following table.

Terminal	Description
Failure	The output terminal to which an asynchronous SOAP response message is routed if a failure is detected when the message received is propagated to the Out flow (such as a message validation failure).
Out	The output terminal to which the asynchronous SOAP response message is routed if it has been successfully received, and if further processing is required in this message flow. If no errors occur in the node, a valid none fault SOAP response message received from an external resource is always sent to the Out terminal first.
Fault	The output terminal to which an asynchronous SOAP fault response is routed if it has been successfully received, and if further processing of the fault is required in this message flow.
Catch	The output terminal to which the message is routed if an exception is thrown downstream and caught by this node.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The SOAPAsyncResponse node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type: SOAPAsyncResponse	The name of the node.
Short description	No	No	None	A brief description of the node.
Long description	No	No	None	Text that describes the purpose of the node in the message flow.

The SOAPAsyncResponse node Basic properties are described in the following table.

Property	M	C	Default	Description	`mqsiapplybaroverride` command property
Unique identifier	Yes	No		Specify the unique URL fragment that is common to your pair of SOAPAsyncRequest and SOAPAsyncResponse nodes.	asyncRequestCorrelator

The SOAPAsyncResponse node Transactions property is described in the following table. This setting does not apply when the node uses HTTP transport.

Property	M	C	Default	Description
Transaction mode	Yes	No	No	This property controls whether the message is received under a JMS transaction. Valid values are `Yes` and `No`. Select `No` to receive the message using a non-transactional JMS session. Select `Yes` to receive the message using a transactional JMS session. The JMS transaction can be either local or global. To use a global transaction, using an XA JMS session, you must also select the message flow property `Coordinated Transaction` in the BAR file properties. See "Configuring for coordinated JMS transactions" on page 4784.

The SOAPAsyncResponse node Advanced property is described in the following table.

Property	M	C	Default	Description
Set destination list	No	No	Selected	This property indicates whether to add the incoming SOAP operation to the route to label destination list.
Label prefix	No	No		Use this property to add a prefix to the SOAP Operation name in the destination list. You must add a `Label prefix` if you want to use multiple SOAPAsyncResponse nodes in the same message flow without causing their corresponding Label nodes to clash. By default, the prefix is an empty string so that the operation name and the label name are identical. This property is not available if the `Set destination list` property is cleared.
Place WS-Addressing headers into LocalEnvironment	No	No	Cleared	This property specifies whether the node puts WS-Addressing headers from the response message into the local environment tree. WS-Addressing headers are not accessible to the flow if this check box is cleared because by default, all headers are processed and removed.

The SOAPAsyncResponse node Instances properties are described in the following table.

Property	M	C	Default	Description	**`mqsiapplybaroverride` command property**
Additional instances pool	No	Yes	`Use Pool Associated with Message Flow`	The pool from which additional instances are obtained. • If you select `Use Pool Associated with Message Flow`, additional instances are obtained from the message flow value. • If you select `Use Pool Associated with Node`, additional instances are allocated from the node's additional instances based on the number specified in the `Additional instances` property.	componentLevel
Additional instances	No	Yes	`0`	The number of additional instances that the node can start if the `Additional instances pool` property is set to `Use Pool Associated with Node`. By default, no additional instances are given to the node.	additionalInstances

The SOAPAsyncResponse node Response Message Parsing properties are described in the following table. The SOAPAsyncResponse node sets these properties automatically; you cannot set them yourself.

Property	M	C	Default	Description
Message domain	No	No	`SOAP`	The domain that is used to parse the response message. By default, the message that is propagated from the SOAPAsyncResponse node is in the SOAP domain. You cannot specify a different domain. For more information, see "SOAP parser and domain" on page 1082. The Response Message Parsing properties are ignored when the paired SOAPAsyncRequest node is configured to act in gateway mode.
Message set	Yes	No	Set automatically from the `WSDL file name` property that is provided by the SOAPAsyncRequest node.	The name of the message set in which the response message is defined. `Message set` is set automatically to the message set that contains the WSDL file that is configured on the corresponding SOAPAsyncRequest node. If you set this property, and then subsequently update the project dependencies to remove this message set reference, a warning is issued. Either update the `Message set` property, or restore the reference to this message set project. The Response Message Parsing properties are ignored when the paired SOAPAsyncRequest node is configured to act in gateway mode.
Message type	No	No		The name of the response message. The node detects the message type automatically. You cannot set this property.
Message format	No	No		The name of the physical format of the response message. You cannot set this property.

The SOAPAsyncResponse node Parser Options properties are described in the following table. The properties are passed through to the XMLNSC parser.

Property	M	C	Default	Description
Parse timing	No	No	`On Demand`	This property controls when a response message is parsed. Valid values are `On Demand`, `Immediate`, and `Complete`. By default, parse timing is set to `On demand`, which causes parsing of the input message to be delayed. For a full description of this property, see "Parsing on demand" on page 4173.
Build tree using XML schema data types	No	No	Selected	This property controls whether the XMLNSC parser creates syntax elements in the message tree with data types taken from the XML Schema. This property is ignored when the paired SOAPAsyncRequest node is configured to act in gateway mode.
Retain mixed content	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters mixed text in a response message. If you select the check box, elements are created for mixed text. If you clear the check box, mixed text is ignored and no elements are created.
Retain comments	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters comments in an response message. If you select the check box, elements are created for comments. If you clear the check box, comments are ignored and no elements are created.

Property	M	C	Default	Description
Retain processing instructions	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters processing instructions in a response message. If you select the check box, elements are created for processing instructions. If you clear the check box, processing instructions are ignored and no elements are created.
Opaque elements	No	No	Blank	This property is used to specify a list of elements in the response message that are to be opaquely parsed by the XMLNSC parser. Opaque parsing is performed only if validation is not enabled (that is, if `Validate` is `None`); entries that are specified in `Opaque Elements` are ignored if validation is enabled.

The SOAPAsyncResponse node Validation properties are described in the following table. By default, validation is enabled.

If a message is propagated to the Failure terminal of the node, it is not validated. For more details, see "Validating messages" on page 1478 and "Validation properties" on page 4169.

Property	M	C	Default	Description	**mqsiapplybaroverride command property**
Validate	No	Yes	`Content and value`	This property controls whether validation takes place. Valid values are `None`, `Content and value`, and `Content`. Validation properties are ignored when the paired SOAPAsyncRequest node is configured to act in gateway mode.	validateMaster
Failure action	No	No	`Exception`	This property controls what happens if validation fails. You can set this property only if you set `Validate` to `Content` or `Content and value`. Valid values are `User trace`, `Exception list`, `Local error log`, and `Exception`. Validation properties are ignored when the paired SOAPAsyncRequest node is configured to act in gateway mode.	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

LocalEnvironment overrides:
You can retrieve information that was set by the paired SOAPAsyncRequest node from the following property under LocalEnvironment.Destination.SOAP.Response :

Setting	Description
UserContext	You can retrieve context data that was stored by the SOAPAsyncRequest node from the following location in the local environment: `SET myVar = InputLocalEnvironment.Destination.SOAP.Response.UserContext;`

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"Parsers" on page 1072
A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from the internal message tree representation.

"SOAP parser and domain" on page 1082
You can use the SOAP parser to create a common WSDL-based logical tree format for working with Web services, independent of the physical bitstream format.

"XMLNSC parser" on page 1090
The XMLNSC parser is a flexible, general-purpose XML parser that offers high performance XML parsing and optional XML Schema validation.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Validating messages" on page 1478
The broker provides validation based on the message model for predefined messages.

"Manipulating messages in the XMLNSC domain" on page 2546
If you are writing ESQL to process messages in the XMLNSC domain, it is helpful to learn about the structure of the message tree that the XMLNSC parser builds.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

Related reference:

"Configuring for coordinated JMS transactions" on page 4544
Configure your message flow to receive or output messages under coordinated transactions.

"Parsing on demand" on page 4173
On-demand parsing, referred to as partial parsing, is used to parse an input message bit stream only as far as is necessary to satisfy the current reference. The parsers that can perform partial parsing of input messages are the JSON, MRM, XML, XMLNS, and XMLNSC parsers.

"Validation properties" on page 4169
You can control validation by setting properties on the Validate and Parser Options tabs for the nodes that are listed in the following table.

"SOAPAsyncRequest node" on page 4750
Use the SOAPAsyncRequest node with the SOAPAsyncResponse node to construct a pair of message flows that call a Web service asynchronously.

"WS-Addressing with the SOAPAsyncRequest and SOAPAsyncResponse nodes" on page 1655
The remote Web service must understand WS-Addressing to be able to work with SOAPAsyncRequest and SOAPAsyncResponse nodes.

Configuring for coordinated JMS transactions:

Configure your message flow to receive or output messages under coordinated transactions.

When you include a node using JMS transport in a message flow, such as the JMSInput or SOAPInput node when using JMS transport, the value that you set for `Transaction mode` defines whether messages are received under sync point.

- If you set this property to `Global`, the message is received under external sync point coordination; that is, within a WebSphere MQ unit of work. Any messages that are sent later, by an output node in the same instance of the message flow, are put under sync point, unless the output node overrides this setting explicitly.
- If you set this property to `Local`, the message is received under the local sync point control of the node. Any messages that are sent later, by an output node in the flow, are not put under local sync point, unless an individual output node specifies that the message must be put under local sync point.
- If you set this property to `None`, the message is not received under sync point. Any messages that are sent later, by an output node in the flow, are not put under sync point, unless an individual output node specifies that the message must be put under sync point.

To receive messages under external sync point, you must take additional configuration steps, which need be applied only the first time that a specific node using JMS transport is deployed to the broker for a particular JMS provider.

- On distributed systems, the external sync point coordinator for the broker is WebSphere MQ. Before you deploy a message flow in which the `Transaction mode` property is set to `Global` or `Yes`, and is intended to use globally coordinated transactions, modify the queue manager `.ini` file to include extra definitions for each JMS provider resource manager that participates in globally coordinated transactions.
  - Windows On Windows:
    1. Start WebSphere MQ Explorer.
    2. Right-click the queue manager name in the left pane and click **Properties**.
    3. Click **XA resource managers** in the left pane.
    4. Click **Add...** .
    5. Set the options as follows:
       - Set `Name` to any value.
       - On Windows on x86 systems, set the `SwitchFile` property to *`install_dir`*`\bin\JMSSwitch.dll`. On Windows on x86-64 systems, set the `SwitchFile` property to `JMSSwitch.dll`.
       - Set the `XAOpenString` property to a string value as follows: *`Initial Context`*`,`*`location JNDI`*`,`*`Optional_parms`*.
       - Set the `ThreadOfControl` property to `Thread`.
    6. On Windows on x86-64 systems only, copy the switch file *JMSSwitch32.dll* to the `\exits` subdirectory in the WebSphere MQ installation directory, and rename it to `JMSSwitch.dll`. Copy the switch file `JMSSwitch.dll` to the `\exits64` subdirectory in the WebSphere MQ installation directory.

For more information, see the *System Administration Guide* section of the WebSphere MQ Version 7 Information Center online.

– Linux UNIX On Linux and UNIX systems, add a stanza to the queue manager `.ini` file for each JMS provider.

  For example:

```
XAResourceManager:
Name=Jms_Provider_Name
SwitchFile=/install_dir/bin/ JMSSwitch.so
XAOpenString=Initial Context,location JNDI,Optional_parms
ThreadOfControl=THREAD
```

  Where:

  Name is an installation defined name that identifies a JMS provider resource manager.

  SwitchFile
  is the file system path to the JMSSwitch library that is supplied in the `bin` directory of the broker.

XAOpenString can have the following values:

– *Initial Context* is the value that is set in the JMSInput node property `Initial context factory`.

– *location JNDI* is the value that is set in the JMSInput node property `Location JNDI bindings`. This value must include a supported URL prefix that has a URL handler that is available on the class path.

The following parameters are optional:

– `LDAP Principal` matches the value that is set for the broker by using the **mqsicreatebroker** or **mqsichangebroker** commands.

– `LDAP Credentials` matches the value that is set for the broker by using the **mqsicreatebroker** or **mqsichangebroker** commands.

– `Recovery Connection Factory Name` is the JNDI administered connection factory that is defined in the bindings file. If a value is not specified, you must add a default value for `recoverXAQCF` to the bindings file. In either case, the Recovery Connection Factory must be defined as an XA Queue Connection Factory for the JMS provider that is associated with the `Initial context factory`.

The optional parameters are comma-separated and are positional. Therefore, any parameters that are missing must be represented by a comma. For example:

```
com.sun.jndi.fscontext.RefFSContextFactory,file:/C:/webservices/SOAP/JMS/JNDIXA,,,QCF
```

1. Update the Java CLASSPATH environment variable for the queue manager of the broker to include a reference to `xarecovery.jar`; for example:

   install_dir`/classes/xarecovery.jar`

2. Update the Java PATH environment variable for the queue manager of the broker to point to the bin directory in which the SwitchFile is located; for example:

   install_dir`/bin`

Finally, ensure that you have taken the following configuration steps:

– In the message flow, ensure that the coordinated property is enabled by using the WebSphere Message Broker Archive editor.

– Ensure that each node that must be part of the XA transaction is set to the global transaction mode.

– Ensure that the service ID that is used for the broker and the queue manager is the same user ID.

– Ensure that the JNDI connection factory objects that the JMS nodes use for a global transaction are configured to be of the type MQXAConnectionFactory, MQXAQueueConnectionFactory, or MQXATopicConnectionFactory.
  - If you create the bindings using WebSphere Message Broker Explorer, ensure the `Support XA Transactions` option is checked when you define your connection factory.
  - If you create the bindings using JMSAdmin, use the command **DEF XAQCF** or **DEF XATCF**, instead of **DEF QCF** or **DEF TCF**, when you define your connection factory.

  For more information, see the *System Administration Guide* section of the WebSphere MQ Version 7 Information Center online.

- z/OS On z/OS, the external sync point manager is Resource Recovery Services (RRS). The only JMS provider that is supported on z/OS is WebSphere MQ JMS. The only transport option that is supported for WebSphere MQ JMS on z/OS is the bind option.

  Sync point control for the JMS provider is managed with RRS sync point coordination of the queue manager of the broker. You do not have to modify the `.ini` file.

SOAPEnvelope node

Use the SOAPEnvelope node to add a SOAP envelope onto an existing message. This node is designed to be used with the SOAPExtract node.

This topic contains the following sections:

Purpose:
The default behavior of the SOAPEnvelope node is to attach the SOAP envelope from a standard location ($LocalEnvironment/SOAP/Envelope) in the local environment tree; you can specify an explicit location by using an XPath expression.

You can also use the node in a flow without a corresponding SOAPExtract node; the node has an option to create a default SOAP envelope.

The SOAPEnvelope node is contained in the **Web Services** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using the SOAPEnvelope node in a message flow:
This node is designed to be used in conjunction with the SOAPExtract node; see "SOAPExtract node" on page 4790.

Configuring the SOAPEnvelope node:
When you have put an instance of the SOAPEnvelope node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view.

Supported parsers:
This node is designed to work with SOAP messages. Use one of the following parsers:

- XMLNSC
- MRM
- XMLNS

Other XML parsers are not supported because they do not support namespaces. An exception is thrown if a message is received which is not using the correct parser or does not conform to the basic structure of a SOAP message.

Full validation is not done on the SOAP message, which just needs to contain a body element.

As the SOAP domain is not supported by the SOAPEnvelope node, you cannot add the envelope extracted by the SOAPExtract node, from the SOAP domain, back into the message flow again; that is, a flow such as the following example is not supported:

```
SOAPInput node-> SOAPExtract node->SOAPEnvelope node
```

Example SOAP messages:
Incoming SOAP envelope

```
<?xml version="1.0"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns="http://ws3.st.mqsi.ibm.com/App/DocLiteral1"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Header>
        <tns:requestHeader>
            <tns:assessorUrl>header1</tns:assessorUrl>
        </tns:requestHeader>
    </soapenv:Header>
</soapenv:Envelope>
```

Incoming SOAP message body

```
<?xml version="1.0"?>
<tns:requestAvailability
xmlns:tns="http://ws3.st.mqsi.ibm.com/App/DocLiteral1"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <tns:carDetails>body1</tns:carDetails>
    <tns:claimID>body2</tns:claimID>
    <tns:location>body3</tns:location>
    <tns:reqDate>body4</tns:reqDate>
</tns:requestAvailability>
```

Outgoing SOAP message

```
<?xml version="1.0"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns="http://ws3.st.mqsi.ibm.com/App/DocLiteral1"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Header>
        <tns:requestHeader>
```

```
            <tns:assessorUrl>header1</tns:assessorUrl>
        </tns:requestHeader>
    </soapenv:Header>
    <soapenv:Body>
        <tns:requestAvailability>
            <tns:carDetails>body1</tns:carDetails>
            <tns:claimID>body2</tns:claimID>
            <tns:location>body3</tns:location>
            <tns:reqDate>body4</tns:reqDate>
        </tns:requestAvailability>
    </soapenv:Body>
</soapenv:Envelope>
```

Terminals and properties:
The terminals of the SOAPEnvelope node are described in the following table:

Terminal	Description
In	The input terminal that accepts a SOAP message for processing by the node.
Out	The output terminal that outputs the SOAP message that was constructed from the SOAP message body and a SOAP envelope.
Failure	The output terminal to which the message is routed if a failure is detected during processing.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Description properties of the SOAPEnvelope node are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The Basic properties of the SOAPEnvelope node are described in the following table.

Property	M	C	Default	Description
Create new envelope	No	No	Cleared	This property controls whether the node creates a SOAP envelope, or gets an existing one from the message tree. If you select the check box, the node creates a new envelope. If you clear the check box, the node copies the envelope from the value entered in the `Existing Envelope Location` property.

Property	M	C	Default	Description
Existing Envelope Location	No	No	`$LocalEnvironment/SOAP/Envelope`	An XPath expression that represents the location from which the node will copy the SOAP envelope. The following correlation names are available: **`$Root`** The root of the message tree. **`$Body`** The last child of the root of the message tree (equivalent to /). **`$LocalEnvironment`** The root of the local environment tree. **`$Environment`** The root of the global environment tree.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"SOAP nodes" on page 1609
The SOAP nodes act as points in the flow where Web service processing is configured and applied. Properties on the SOAP nodes control the processing carried out and can be configured by supplying a WSDL definition, or by manually configuring properties, or both.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"Parsers" on page 1072
A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from the internal message tree representation.

"SOAP parser and domain" on page 1082
You can use the SOAP parser to create a common WSDL-based logical tree format for working with Web services, independent of the physical bitstream format.

"XMLNSC parser" on page 1090
The XMLNSC parser is a flexible, general-purpose XML parser that offers high performance XML parsing and optional XML Schema validation.

Related tasks:

Chapter 9, “Developing message flow applications,” on page 1019
Develop message flows to process your business messages and data.

“Manipulating messages in the XMLNSC domain” on page 2546
If you are writing ESQL to process messages in the XMLNSC domain, it is helpful to learn about the structure of the message tree that the XMLNSC parser builds.

“Handling errors in message flows” on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

Related reference:

“SOAPExtract node”
Use the SOAPExtract node to remove SOAP envelopes, allowing just the body of a SOAP message to be processed. It can also route a SOAP message based on its operation name. Both functions are optional; they are contained in one node because they are often used together.

“Parsing on demand” on page 4173
On-demand parsing, referred to as partial parsing, is used to parse an input message bit stream only as far as is necessary to satisfy the current reference. The parsers that can perform partial parsing of input messages are the JSON, MRM, XML, XMLNS, and XMLNSC parsers.

SOAPExtract node

Use the SOAPExtract node to remove SOAP envelopes, allowing just the body of a SOAP message to be processed. It can also route a SOAP message based on its operation name. Both functions are optional; they are contained in one node because they are often used together.

This topic contains the following sections:
- “Purpose”
- “Using the SOAPExtract node in a message flow” on page 4791
- “Configuring the SOAPExtract node” on page 4791
- “Supported parsers” on page 4791
- “Terminals and properties” on page 4792
- “Example SOAP messages” on page 4793

Purpose:
The SOAPExtract node can perform two functions:

`Extract function`
The default behavior is to detach the SOAP envelope to a standard location ($LocalEnvironment/SOAP/Envelope) in the LocalEnvironment tree. Alternatively, you can specify an explicit location using an XPath expression. Any existing SOAP envelope at the chosen location is replaced.

`Routing function`
The SOAP message is routed to a Label node in the message flow as identified by the SOAP operation in the message. The SOAP Operation is identified in the SOAP body tag.

Ensure that the message parser options in the properties folder of the outgoing message are correctly set up to parse the message, by copying the message set and message format from the incoming message. The message type is derived from the SOAP envelope message body first child.

Only a single child of the SOAP message body is supported.

The SOAPExtract node is contained in the **Web Services** drawer of the palette, and is represented in the workbench by the following icon:

Using the SOAPExtract node in a message flow:
Look at the following sample to see how to use this node:

- SOAP Nodes

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Configuring the SOAPExtract node:
When you have put an instance of the SOAPExtract node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view.

1. Optional: On the **Description** tab, enter a `Short description`, a `Long description`, or both. You can also rename the node on this tab.
2. On the **Basic** tab:
   a. Specify in `Remove envelope` whether the node must remove the soap envelope and place it in the location given in `Envelope Destination`, or leave it on the message. The default value is that the node removes the envelope.
   b. In `Envelope Destination`, enter an XPath expression that represents the destination to which the node will copy the envelope. By default, the node copies the envelope to the LocalEnvironment ($LocalEnvironment/SOAP/Envelope).
   c. In `Destination path mode`, specify the behavior of the Envelope Destination property.
      - `Create path`: The node creates the tree if the path specifies a location that does not already exist. Only simple expressions of the form `aaa/bbb/ccc` in `Envelope Destination` are supported. The default.
      - `XPath location of existing element`: If you know that the destination element exists, you can use any valid XPath 1.0 expression in `Envelope Destination`.
   d. In `Route to 'operation' label`, specify whether the node must route the message to the SOAP operation given in the message. The default setting is for the node to send the message to the Out terminal.
   e. In `Label Prefix`, enter the value to prefix to the label used for routing by the node. Entering a prefix allows for name spacing between subflows. By default, no value is prefixed to the label name used for routing the message.

Supported parsers:
This node is designed to work with SOAP messages. Use one of the following parsers:

- SOAP
- XMLNSC
- MRM
- XMLNS

Other XML parsers are not supported because they do not support namespaces. An exception is thrown if a message is received which is not using the correct parser or does not conform to the basic structure of a SOAP message.

Full validation is not done on the SOAP message, which just needs to contain a body element.

Terminals and properties:
The terminals of the SOAPExtract node are described in the following table:

Terminal	Description
In	The input terminal that accepts a SOAP message for processing by the node.
Out	The output terminal that outputs the SOAP message body (without the envelope if the `Remove envelope` check box is selected on the node properties).
Failure	The output terminal to which the message is routed if a failure is detected during processing.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Description properties of the SOAPExtract node are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The Basic properties of the SOAPExtract node are described in the following table.

Property	M	C	Default	Description
Remove envelope	No	No	Selected	If you select the check box, the node removes the SOAP header from the message. For a SOAP tree, the node outputs to the Out terminal the first child of SOAP.body from the SOAP tree. It outputs to `Envelope Destination` the full SOAP tree minus the first child of SOAP.body. If you clear the check box, the node leaves the envelope on the message. In the case of a SOAP tree, the full tree is propagated to the Out terminal.

Property	M	C	Default	Description
Envelope Destination	No	No	`$LocalEnvironment/SOAP/Envelope`	An XPath expression that represents the destination to which the node will copy the SOAP envelope. The following correlation names are available: **`$Root`** The root of the message tree. **`$Body`** The last child of the root of the message tree (equivalent to /). **`$LocalEnvironment`** The root of the LocalEnvironment tree. **`$Environment`** The root of the Global Environment tree.
Destination path mode	No	No	`Create path`	This determines the behavior of the `Envelope Destination` property. Set this property: **`Create path`** The default. The tree is created if the path specifies a location that does not exist. Only simple expressions of the form `aaa/bbb/ccc` are supported. **`XPath location of existing element`** If you know that the destination element exists, you can enter any valid XPath 1.0 expression.
Route to 'operation' label	No	No	Cleared	This property determines whether the node must route the message to the SOAP operation given in the message. If you select the check box, the message is routed to a Label node that matches the SOAP operation. An exception is thrown if no Label node matches. The name of the first child element of the SOAP body is used to determine the RouteToLabel name. For the 'RPC literal' and 'wrapped doc literal' WSDL types, this is the 'operation' name. For a SOAP tree, the first child of SOAP.Body supplies the operation name. If you clear the check box, the node sends the message to the Out terminal.
Label Prefix	No	No		The value to prefix to the label that the node uses for routing. Entering a prefix allows for name spacing between subflows.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Example SOAP messages:

Incoming SOAP message

```
<?xml version="1.0"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns="http://ws3.st.mqsi.ibm.com/App/DocLiteral1"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Header>
    <tns:requestHeader>
        <tns:assessorUrl>header1</tns:assessorUrl>
    </tns:requestHeader>
    </soapenv:Header>
    <soapenv:Body>
        <tns:requestAvailability>
            <tns:carDetails>body1</tns:carDetails>
            <tns:claimID>body2</tns:claimID>
            <tns:location>body3</tns:location>
            <tns:reqDate>body4</tns:reqDate>
        </tns:requestAvailability>
    </soapenv:Body>
</soapenv:Envelope>
```

De-enveloped message

The operation name is requestAvailability. Note that the namespacing is removed from the operation.

```
<?xml version="1.0"?>
<tns:requestAvailability
xmlns:tns="http://ws3.st.mqsi.ibm.com/App/DocLiteral1"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <tns:carDetails>body1</tns:carDetails>
    <tns:claimID>body2</tns:claimID>
    <tns:location>body3</tns:location>
    <tns:reqDate>body4</tns:reqDate>
</tns:requestAvailability>
```

Removed envelope

```
<?xml version="1.0"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns="http://ws3.st.mqsi.ibm.com/App/DocLiteral1"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Header>
        <tns:requestHeader>
            <tns:assessorUrl>header1</tns:assessorUrl>
        </tns:requestHeader>
    </soapenv:Header>
</soapenv:Envelope>
```

Related concepts:

"SOAP nodes" on page 1609
The SOAP nodes act as points in the flow where Web service processing is configured and applied. Properties on the SOAP nodes control the processing carried out and can be configured by supplying a WSDL definition, or by manually configuring properties, or both.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an

input message is received.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"Parsers" on page 1072
A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from the internal message tree representation.

"SOAP parser and domain" on page 1082
You can use the SOAP parser to create a common WSDL-based logical tree format for working with Web services, independent of the physical bitstream format.

"XMLNSC parser" on page 1090
The XMLNSC parser is a flexible, general-purpose XML parser that offers high performance XML parsing and optional XML Schema validation.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Manipulating messages in the XMLNSC domain" on page 2546
If you are writing ESQL to process messages in the XMLNSC domain, it is helpful to learn about the structure of the message tree that the XMLNSC parser builds.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

Related reference:

"SOAPEnvelope node" on page 4786
Use the SOAPEnvelope node to add a SOAP envelope onto an existing message. This node is designed to be used with the SOAPExtract node.

"Parsing on demand" on page 4173
On-demand parsing, referred to as partial parsing, is used to parse an input message bit stream only as far as is necessary to satisfy the current reference. The parsers that can perform partial parsing of input messages are the JSON, MRM, XML, XMLNS, and XMLNSC parsers.

SOAPInput node

Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

This topic contains the following sections:

Purpose:

The SOAPInput node is typically used with the SOAPReply node, which can be included in the same message flow, or a different flow in the same execution group.

You can connect a SOAPReply node to the Out terminal to handle successful responses. If you also want your message flow to handle reply processing after a timeout, connect a SOAPReply node to the HTTP Timeout terminal.

You cannot use an HTTPReply node to respond to a Web service request that is received by a SOAPInput node; the broker raises an exception when the reply is attempted.

If you choose to handle HTTP messages by using the execution group listener, you must carefully check the URL specifications in your HTTPInput and SOAPInput nodes. If both URL specifications match an incoming message, the wrong type of node might get the message, and processing might fail or produce unexpected results. This situation occurs if you specify identical values for the Path suffix for URL properties of the HTTPInput node and the SOAPInput node. It can also occur if you use wildcards in either or both specifications, and an incoming message matches both properties.

The SOAPInput node is contained in the **Web Services** drawer of the message flow node palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:
The SOAPInput node can be used in a message flow that accepts and processes SOAP messages. The node is configured using deployable WSDL. Look at the following sample to see how to use this node:

- SOAP Nodes

A client can send an HTTP GET to the web service endpoint exposed by the SOAPInput node, suffixed with a query string `?wsdl`, and receive a response with the WSDL definition used to configure the flow; see "Using WSDL to configure message flows" on page 1664.

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

A client can send a request that has a Content-Encoding of gzip or deflate. When such a request is received the content is decoded and the Content-Encoding header is removed.

Using port numbers when deploying a SOAPInput node to an execution group:
When processing HTTP messages, you can use either the broker-wide listener or execution group (embedded) listeners for your HTTP message flows. See "HTTP listeners" on page 1589. However, SOAP nodes always use the embedded listener.

Each execution group that contains a SOAPInput node has one listener and two ports, an HTTP port and an HTTPS port. The default SOAP node port numbers are 7800 for HTTP and 7843 for HTTPS. If you deploy the flow to multiple execution groups, the port number is incremented by one for each successive deployment. The message flow that is deployed to the first execution group receives requests on port 7800 (by default), the next one uses port 7801, and so on, up to the specified

limit of 7842. In this scenario, you typically use an intermediary router that listens on one port, then distributes the requests across the range of ports that you are using.

If you do not want the port to be allocated dynamically, you can define a specific port by using the **mqsichangeproperties** command. You can also change the default range of port numbers by using this command.

Connecting the terminals:
The SOAPInput node routes each message that it retrieves successfully to the Out terminal. If message validation fails, the message is routed to the Failure terminal; you can connect nodes to this terminal to handle this condition. If you have not connected the Failure terminal, the message is discarded, the `Maximum client wait time` expires, and an error is returned to the client.

If the message is caught by this node after an exception has been thrown further on in the message flow, the message is routed to the Catch terminal. If you have not connected the Catch terminal, the message is discarded, the `Maximum client wait time` expires, and an error is returned to the client.

If the Maximum client wait time expires, by default, the listener sends a SOAP fault message to the client, indicating that its timeout has expired. If you have connected the HTTP Timeout terminal, and you are using the HTTP transport, the SOAP Fault message is propagated through the HTTP Timeout terminal. You must include a SOAPReply node in the sequence of nodes connected to the HTTP Timeout terminal and this node must send a valid SOAP Fault message. The listener waits again for the interval defined by the `Maximum client wait time (sec)` property, or for 10 seconds, whichever is the shorter interval:

- If a response is received before this second interval expires, the listener propagates the response to the client.
- If a response is not received before this second interval expires, the listener sends a SOAP fault message to the client, indicating that its timeout has expired.

Because the listener waits for only a brief interval after the message has been propagated through the HTTP Timeout terminal, you must ensure that the sequence of nodes that you connect to the HTTP Timeout terminal includes a SOAPReply node, which sends a response before this interval expires. If you have connected the HTTP Timeout terminal, but you are not using the HTTP transport, the message is not propagated through the HTTP Timeout terminal. The listener sends a SOAP fault message to the client, indicating that its timeout has expired.

Terminals and properties:
The SOAPInput node terminals are described in the following table.

Name	Type	Description
Failure	Output data	The output terminal to which a SOAP message is routed if a failure is detected when the message received is propagated to the Out terminal (such as a message validation failure).
Out	Output data	The output terminal to which the SOAP message is routed when it is received successfully and if further processing is required in this message flow. If no errors occur in the input node, a valid SOAP message that is received from an external resource is always sent to the `Out` terminal first.

Name	Type	Description
HTTPTimeout	Output data	The output terminal to which a timeout SOAP fault message is routed if the SOAPReply node that is connected to the Out terminal does not respond within the time interval specified by the Maximum client wait time property. This terminal is used only if the message is sent across the HTTP transport, and WS-RM is not being used.
Catch	Output data	The output terminal to which the message is routed if an exception is thrown downstream and is caught by this node.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

Some SOAPInput node properties are initially set from properties in the imported WSDL. These properties are parsed differently depending on which URI format is used by the `address` element in the WSDL. For details, see "WSDL URI formats for JMS" on page 1668.

The following Retry properties are no longer available on the SOAPInput node:

- Retry mechanism
- Retry threshold
- Short retry interval
- Long retry interval

When a SOAP over HTTP message fails, a SOAP fault is sent back to the client; the message exchange pattern is complete and no message exists to retry. When a SOAP over JMS message fails, retry processing is handled according to the backout properties defined on the JMS Transport tab.

The SOAPInput node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type; SOAPInput	The name of the node.
Short description	No	No	None	A brief description of the node.
Long description	No	No	None	Text that describes the purpose of the node in the message flow.

The SOAPInput node Basic properties are described in the following table.

Property	M	C	Default	Description
Operation mode	Yes	Yes	`Specify WSDL interface to expose`	This property allows you to specify the operation mode of the node, which determines whether it acts in WSDL mode or in gateway mode. In WSDL mode, the node performs operations according to the WSDL it is configured with. However, gateway mode allows you to configure your flow to handle generic SOAP request/response and one-way messages, or to act as a façade between multiple web services clients and multiple back-end web services providers. **Specify WSDL interface to expose** Configure the node with a deployable WSDL by setting the WSDL file name property or by dragging a WSDL onto the node. This is the default option. **Operate in gateway mode** Configure the node to act in gateway mode with no WSDL required. See "Gateway operation mode for SOAP nodes" on page 1645 for a fuller explanation of gateway mode.
WSDL file name	Yes	No	None	This property indicates the location of the WSDL file that you want to use to configure the node. Enter the full path to the WSDL file, or click **Browse** to locate the WSDL file in your workspace. When you select a WSDL file for the `WSDL file name` property, the WSDL is validated to ensure that it is WS-I compliant. If the WSDL has a binding using SOAP/JMS which is not WS-I compliant, by default no error is shown. To enable strict WS-I validation and display a warning when a SOAP/JMS transport is used, click **Window** > **Preferences** > **Broker Development** > **Message Sets** > **Validation** and clear the **WS-I BP 1.1: Allow SOAP/JMS as transport URI** check box. Only deployable WSDL files can be used to configure the SOAP nodes. After a valid WSDL file is selected, the message set project to which the WSDL file belongs is added as a referenced project to the corresponding message flow project, if the reference does not exist. If the WSDL file is not valid, or an incorrect file name is entered, an error message is displayed in the Properties view and all WSDL properties are blank. If the node was created by dropping a WSDL file from a message set onto the Message Flow editor, this property is preset to the name of the WSDL file. This property takes a string value. The following situations result in an error condition: • The WSDL file does not belong to a • message set project, or the WSDL file was not imported correctly; for more details, see "Importing from WSDL" on page 2946 and "Importing WSDL definitions from the command line" on page 2948. • The WSDL file contains no HTTP or JMS bindings. • The WSDL file contains no port type. • The WSDL file that is specified in the field does not exist. WSDL properties are disabled when the node is configured to act in gateway mode.

Property	M	C	Default	Description
Port type	Yes	No	The first `Port type` found in the WSDL file (that has an associated HTTP binding with it).	This property lists all the port types that are defined by the specified WSDL file. By default, the first port type found in the WSDL file that has an associated HTTP or JMS binding is selected. This property takes a string value. The following situation causes an error condition: • The selected `Port type` does not contain at least one operation. When you save the message flow file, validation of some of the WSDL-related properties occur to ensure that: • The WSDL file exists in the message set. • The selected `Port type`, `Binding operation`, and `Service port` are all valid in the content of the selected WSDL file. If one or more of these conditions are not met, an error is generated, and you cannot add a message flow that contains this SOAPInput node to a broker archive (BAR) file. WSDL properties are disabled when the node is configured to act in gateway mode.
Imported binding	Yes	No		The `Imported binding` property lists the imported SOAP bindings associated with the selected port type. Only HTTP or JMS transport is supported. When you select a binding, the property tab for the associated transport is enabled; otherwise, it is disabled. Bindings are listed in the order in which they are displayed in the WSDL file. By default, the first imported binding that implements the operation, and has an associated service port, is selected. This property is updated every time the `Port type` value changes. This property type is String. The following situations cause an error condition: • No imported SOAP bindings (with HTTP or JMS transport) in the WSDL file are associated with the `Port type`. • The selected binding does not have any operations. WSDL properties are disabled when the node is configured to act in gateway mode.
Service port	Yes	No		The `Service port` field lists all the service ports that point to the selected binding. By default, the first service port for the binding is selected. This property is updated every time the selected binding value changes. This property type is String. The following situation causes an error condition: • No ports point to the selected binding. WSDL properties are disabled when the node is configured to act in gateway mode.
Target namespace	Yes	No		This property displays the namespace of the selected WSDL file. This property type is String. WSDL properties are disabled when the node is configured to act in gateway mode.

Property	M	C	Default	Description
Transport	No	No		This property is set automatically when the `Imported binding` property is selected. The value of this property shows the transport used by the selected WSDL binding if one is selected; for example, `HTTP` or `JMS`. If you choose to switch the transport from `JMS` to `HTTP`, a dialog box displays, which allows you to reset the JMS-specific properties. You must reset the JMS properties to deploy the message flow to a runtime environment version prior to fix pack V7.0.0.1.

The SOAPInput node HTTP Transport properties are described in the following table. These settings are used only when the node uses HTTP transport.

Property	M	C	Default	Description	**`mqsiapplybaroverride` command property**
Path suffix for URL	Yes	Yes	None	`Path suffix for URL` is the HTTP path selector on which the node accepts inbound messages. This property is set automatically from the `<soap:address>` element of the selected `Service port`. Whenever the selected port is updated, `URL Selector` is updated accordingly. However, if you override this value, your value persists, and the URL is no longer updated from the service port. If you choose to override this property, you must specify the [<path>]. If you configure the same HTTP path for nodes in multiple message flows, an inbound request to that endpoint might be processed by any of those message flows. If the message flows are configured differently, they process the message differently and so this can lead to unpredictable behavior. If you require additional resources for a single HTTP path, you can configure additional instances instead of creating multiple message flows. For more information, see “Configurable message flow properties” on page 4020.	urlSelector

Property	M	C	Default	Description	**mqsiapplybaroverride command property**
Use HTTPS	No	Yes	Cleared	This property type is Boolean and is configured automatically from the `<soap:address>` element of the selected `Service port`. If the address contains an HTTPS URL, the check box is selected; otherwise it is cleared. However, if you manually override this property value, it is no longer updated from the corresponding service port. To enable the HTTPS protocol when this property is selected, perform the following steps: 1. Create a new key store of type "jks" and choose a password. 2. Create a new self signed certificate with a label of your choice. 3. Run the following command: **mqsichangeproperties** *brokername* `-o BrokerRegistry -n brokerKeystoreFile -v` *keystoreFile* 4. Run the following command: **mqsichangeproperties** *brokername* `-o BrokerRegistry -n brokerTruststoreFile -v` *keystoreFile* 5. Run the following command: **mqsisetdbparms** *brokername* `-n brokerKeystore::password -u na -p` *keystorePassword* 6. Run the following command: **mqsisetdbparms** *brokername* `-n brokerTruststore::password -u na -p` *keystorePassword* 7. Deploy your message flow to the broker. 8. This process uses TLS rather than SSL. To enable SSL, run the following command: **mqsichangeproperties** *brokername* `-e` *executionGroup* `-o HTTPSConnector -n sslProtocol -v SSL`. If your flow uses a SOAPRequest node, you should also change the value of the `Protocol` property on the SOAPRequest node.	useHTTPS

Property	M	C	Default	Description	mqsiapplybaroverride command property
Maximum client wait time (sec)	Yes	Yes	180	The time that the client waits for a remote server to respond with a "message received" acknowledgment. The valid range is zero (which means a short wait) through (2^{31})-1. This property specifies the maximum length of time that the TCP/IP listener that received the input message from the Web service client waits for a response from a SOAPReply node. If a response is received within this time, the listener propagates the response to the client. If a response is not received in this time, a SOAP fault message is generated indicating that the timeout has expired. This fault message is either sent by the listener or when using the HTTP transport, the timeout terminal processing. See the "Connecting the terminals" on page 4797 and "Terminals and properties" on page 4797 sections for more information on the HTTP Timeout terminal.	maxClientWaitTime
Enable support for ?wsdl	N	Y	Cleared	If this property is selected, the broker returns WSDL and XML Schema information relating to this endpoint in response to an HTTP GET request with a `?wsdl` query string. This allows you to control the distribution of your WSDL. For a full description, see "Using WSDL to configure message flows" on page 1664. This property is disabled when the node is configured to act in gateway mode.	

The SOAPInput node JMS Transport properties are described in the following table. These settings are used only when the node uses JMS transport

Property	M	C	Default	Description	mqsiapplybaroverride command property
Source	Yes	No	None	The name of the queue from which the node receives incoming messages. This property takes its initial value from a WSDL URI property, depending on whether the WSDL `address` URI is formatted in the W3C (standards) style, or the IBM (proprietary) style. `Source` is set to the value of `destinationName` found in the WSDL if a W3C-style URI is found, or `destination` if an IBM-style URI is found.	source
JMS provider name	Yes	No	`WebSphere MQ`	Select a JMS vendor name from the list, or enter a name of your choice. When you select a name from the list, the `Initial context factory` property is updated automatically with the relevant Java class. If you enter your own JMS provider name, you must also enter a value for the `Initial context factory`. The name must match the name of a configurable service that is defined for the broker to which you deploy the message flow.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Initial context factory	Yes	Yes	com.sun.jndi.fscontext.RefFSContextFactory	The starting point for a JNDI namespace. A JMS application uses the initial context to obtain and look up the connection factory and queue or topic objects for the JMS provider. If you select a JMS provider name from the list in JMS provider name, the Initial context factory property is updated automatically with the relevant Java class. If you enter your own JMS provider name, you must also enter a value for the Initial context factory. The default value is com.sun.jndi.fscontext.RefFSContextFactory, which defines the file-based Initial context factory for the WebSphere MQ JMS provider. This property takes its initial value from a WSDL URI property, depending on whether the WSDL address URI is formatted in the W3C (standards) style, or the IBM (proprietary) style. Initial context factory is set to the value of jndiInitialContextFactory found in the WSDL if a W3C-style URI is found, or initialContextFactory if an IBM-style URI is found.	initialContextFactory

Property	M	C	Default	Description	**mqsiapplybaroverride command property**
JNDI URL bindings location	Yes	Yes		The system path or the LDAP location for the bindings file. The bindings file contains definitions for the JNDI administered objects that are used by the SOAPInput node. This property is disabled when the `Initial context factory` is `com.ibm.mq.jms.Nojndi`. When you enter a value for `JNDI URL bindings location`, ensure that it complies with the following instructions: • Construct the bindings file before you deploy a message flow that contains a SOAPInput node. • Do not include the file name of the bindings file in this field. • If you have specified an LDAP location that requires authentication, configure the LDAP principal (userid) and LDAP credentials (password) separately. These values are configured at broker level. For information about configuring these values, see "**mqsicreatebroker** command" on page 3831 and "**mqsichangebroker** command" on page 3723. • The string value must include a supported URL prefix that has a URL handler that is available on the classpath. For information about constructing the JNDI administered objects bindings file, see the JMS provider documentation. This property takes its initial value from a WSDL URI property, depending on whether the WSDL `address` URI is formatted in the W3C (standards) style, or the IBM (proprietary) style. `JNDI URL bindings location` is set to the value of `jndiURL` found in the WSDL if a W3C-style URI is found, or `jndiProviderURL` if an IBM-style URI is found.	locationJndiBindings
Connection factory name	Yes	Yes		The name of the connection factory that is used by the SOAPInput node to create a connection to the JMS provider. This property is initially configured from the imported WSDL. This name must exist in the bindings file. The `Connection factory name` must be a JMS QueueConnectionFactory. This property takes its initial value from a WSDL URI property, depending on whether the WSDL `address` URI is formatted in the W3C (standards) style, or the IBM (proprietary) style. `Connection factory name` is set to the value of `jndiConnectionFactoryName` found in the WSDL if a W3C-style URI is found, or `connectionFactory` if an IBM-style URI is found.	connectionFactoryName

Property	M	C	Default	Description	mqsiapplybaroverride command property
Backout destination	No	Yes		The SOAPInput node sends input messages to this destination when errors prevent the message flow from processing the message, and the message must be removed from the input destination. The backout destination name must exist in the bindings file.	backoutDestination
Backout threshold	No	Yes	0	The value that controls when a message is put to the backout destination. For example, if the value is 3, the JMS provider attempts to deliver the message to the input destination three times. After the third attempted delivery, the message is not rolled back to the input destination and is sent to the `Backout destination`. A SOAP fault is sent only when the backout threshold has been reached. See "Configuring the backout threshold property" on page 4816.	
JNDI context parameters	No	No		A table that maps JNDI context parameters to their type. These properties are initially configured from the imported WSDL. These properties take their initial values from any W3C-style WSDL properties starting with `jndi-`. IBM-style WSDL URIs do not support JNDI context parameters, but you can set these properties on the node.	

The SOAPInput node Message Selectors properties are described in the following table. This tab is enabled only if the selected binding in the Basic tab uses JMS transport.

For a description of how to construct the JMS message selector, see JMS message selector.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Application property	No	Yes		The message selector that filters messages according to the application property value. If the JMS provider is required to filter messages, based on message properties that are set by the originating JMS client application, enter a selector string for `Application property`, specifying both the property name and the selection conditions; for example, `OrderValue > 200`. Leave `Application property` blank if you do not want the input node to make a selection based on application property.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Timestamp	No	Yes		The message selector that filters messages according to the JMSTimestamp. If the JMS provider is required to filter messages that have been generated at specific times, enter a selector string for `Timestamp`, where the value is an unqualified Java millisecond time; for example, `105757642321`. Qualify the selector with operators, such as `=`, `BETWEEN` or `AND`. Leave `Timestamp` blank if you do not want the input node to make a selection based on the JMSTimeStamp.	
Delivery mode	No	Yes	`All`	The message selector that filters messages according to the message delivery mode. If the JMS provider is required to filter messages based on the JMSDeliveryMode header value in the JMS messages, select an option for `Delivery mode` from the list: • Select `Non Persistent` to receive messages that are marked as non-persistent by the originating JMS client application. • Select `Persistent` to receive messages that are marked as persistent by the originating JMS client application. • Select `All` to receive both persistent and non-persistent messages. (This value is the default.) This property takes its initial value from a WSDL URI property, depending on whether the WSDL `address` URI is formatted in the W3C (standards) style, or the IBM (proprietary) style. `Delivery mode` is set to the value of `deliveryMode` found in the WSDL if a W3C-style URI is found, or to the first of `deliveryMode` or `persistence` if an IBM-style URI is found.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Priority	No	Yes		The message selector that filters messages according to the message priority. If the JMS provider is required to filter messages based on the JMSPriority header value in the JMS message, enter a selector string for `Priority`. Valid values for `Priority` are from `0` (lowest) to `9` (highest). For example, enter `= 5` to receive messages of priority 5, `> 4` to receive messages with a priority greater than 4, or `BETWEEN 4 AND 8` to receive messages with a priority in the range 4 - 8. Leave `Priority` blank if you do not want the input node to make a selection based on the JMSPriority. This property takes its initial value from a WSDL URI property, depending on whether the WSDL `address` URI is formatted in the W3C (standards) style, or the IBM (proprietary) style. `Priority` is set to the value of `priority` found in the WSDL if a W3C-style URI is found, or to the first of `priority` or `Priority` if an IBM-style URI is found.	
Message ID	No	Yes		The message selector that filters messages according to the message ID. If the JMS provider is required to filter messages based on the JMSMessageID header, enter a selector string for `Message ID`. For example, enter `> WMBRK123456` to return messages where the `Message ID` is greater than `WMBRK123456`. Leave `Message ID` blank if you do not want the input node to make a selection based on JMSMessageID.	
Redelivered	No	Yes		If the JMS provider is required to filter messages based on the JMSRedelivered header, enter a selector string for `Redelivered`: • Enter `= FALSE` if the input node accepts only messages that have not been redelivered by the JMS provider. • Enter `= TRUE` if the input node accepts only messages that have been redelivered by the JMS provider. • Leave `Redelivered` blank if you do not want the input node to make a selection based on JMSRedelivered.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Correlation ID	No	Yes		The message selector that filters messages according to the correlation ID. If the JMS provider is required to filter messages based on the JMSCorrelationID header, enter a selector string for `Correlation ID`. For example, `= WMBRKABCDEFG` returns messages with a `Correlation ID` that matches this value. Leave `Correlation ID` blank if you do not want the input node to make a selection based on JMSCorrelationID.	
Target service	No	No		This property is configured from the value of the `targetService` property found in the JMS endpoint location URL that is contained in the port section of the WSDL. The SOAP/JMS message will be read from the source queue only if the message has a `targetService` value that matches the value defined on the node. This value is used by the server component to determine the port component to which the request is dispatched. This property takes its initial value from the `targetService` WSDL property.	targetService

The SOAPInput node Transactions properties are described in the following table. This setting does not apply when the node is using HTTP transport.

Property	M	C	Default	Description
Transaction Mode	Yes	No	No	This property controls whether the message is received under a JMS transaction. Valid values are `Yes` and `No`. Select `No` to receive the message using a non-transactional JMS session. Select `Yes` to output the message using a transactional JMS session. The JMS transaction can be either local or global. To use a global transaction, using an XA JMS session, you must also select the message flow property `Coordinated Transaction` in the BAR file properties. See “Configuring for coordinated JMS transactions” on page 4817. The value set for `Transaction mode` on the SOAPInput node is inherited by nodes downstream in the message flow that have their `Transaction mode` set to `Automatic`. Other resources that perform work within the message flow, such as DB2 or WebSphere MQ, use transactions regardless of the node's `Transaction mode` setting, and commit their transaction after the message is processed.

The SOAPInput node Advanced properties are described in the following table.

Property	M	C	Default	Description	
SOAP 1.1 actor (SOAP 1.2 role)	Yes	No	`Ultimate Destination (Ultimate Receiver)`	Use this property to configure the SOAP actor (SOAP 1.1 protocol) or SOAP role (SOAP 1.2 protocol) that is performed by the SOAPInput node. The default value is `Ultimate Destination (Ultimate Receiver)`. (`Ultimate Destination` relates to SOAP 1.1 and `Ultimate Receiver` relates to SOAP 1.2). You can enter any predefined or user-defined value. Predefined roles are specified in the respective SOAP 1.1 or SOAP 1.2 specifications, and are used to process SOAP headers that are targeted at the specific role. If you select `empty`, an error occurs when the message flow is validated. This property takes a string value.	
Set destination list	No	No	Selected	This property specifies whether to add the method binding name to the route-to-label destination list. If you select this check box, the method binding name is added so that you can use a RouteToLabel node in the message flow after the SOAPInput node. This property type is Boolean.	
Label prefix	No	No	None	The prefix to add to the method name when routing to label. Add a `Label prefix` to avoid a clash of corresponding label nodes when you include multiple input nodes in the same message flow. By default, no label prefix exists; therefore, the method name and label name are identical. The default prefix is an empty string so that the operation name and the label name are identical, but the field displays the user instruction: `<enter a prefix if required>`. This property is enabled only if the `setDestinationList` property is enabled.	

Property	M	C	Default	Description	
Route inbound processing failures to failure terminal	No	Yes	Cleared	Select this check box to send any fault to the Failure terminal during inbound SOAP processing. If this property is selected, in a situation during inbound SOAP processing that results in a SOAP fault, instead of immediately sending the SOAP fault back to the client, the fault is sent to the Failure terminal instead to allow logging and recovery processing. In this situation, an exception list is sent to the Failure terminal with the inbound message as a BLOB. By default, this check box is cleared.	sendProcessingFaultsToFailure
WSDL defined SOAP headers	No	No		This table is read-only and is populated by the SOAP headers that are defined in the output part of the selected operations. The table is updated automatically when the selected operation is updated. By default, the check boxes in the second column of the table are cleared for all entries in the WSDL-defined table. You must select the check boxes for those headers that you want to add to the `must understand headers` list. SOAP headers that are part of the must understand headers list are incorporated into the flow rather than causing a SOAP fault. Adding headers to the `must understand headers` list stops SOAP faults being generated by SOAP headers. You do not have to add `must understand headers` for WS-Addressing and WS-Security because these elements are understood if you configure WS Extensions. This property is generated in the CMF file. When the node is configured to act in gateway mode, with no WSDL required, this table is cleared. The original values of these fields are restored if the operation mode of the node is changed back to WSDL mode.	

Property	M	C	Default	Description	
User defined SOAP headers	No	Yes	None	You can add custom headers in this table. Use the **Add**, **Edit** and **Delete** controls to manipulate rows in this table. You must ensure that the check box for the custom header you have added is selected (in the second column of the table), so that the header is added to the `must understand headers` list. This property is generated in the CMF file. When the node is configured to act in gateway mode, with no WSDL required, custom headers in this table have their `Operation` set to `*`. The original values of these fields are restored if the operation mode of the node is changed back to WSDL mode.	

The SOAPInput node WS Extensions properties are described in the following table.

Property	M	C	Default	Description
Use WS-Addressing	No	No	Cleared	This property indicates whether to engage WS-Addressing on the SOAPInput node. By default, this check box is cleared. If this property is selected, a reply can be sent back to a different web service client as specified in the WS-Addressing properties. The reply can be sent using a transport that is different than the one used for the incoming message.
Place WS-Addressing headers into LocalEnvironment	No	No	Cleared	This property specifies whether the node puts WS-Addressing headers received in the message into the local environment tree. WS-Addressing headers are not accessible to the flow if this check box is cleared because, by default, all headers are processed and removed.
WS-Security	No	Yes		This complex property is in the form of a table and consists of two columns: • `Alias` • `XPath Expression` You can add XPath expressions with an associated `Alias` value to the `WS-Security` table. The `Alias` is resolved in a policy set that is created by the administrator. The policy set resolves the `Alias` to either encrypt or sign the part of the message that is referenced by the `XPath Expression`. You can use the **Add**, **Edit** and **Delete** controls in this table.

The SOAPInput node Input Message Parsing properties are described in the following table.

Property	M	C	Default	Description
Message domain	No	No	SOAP	The domain that is used to parse the incoming message. By default, the message that is propagated from the SOAPInput node is in the SOAP domain. You cannot specify a different domain. Input message parsing properties are disabled when the node is configured to act in gateway mode.
Message set	Yes	No	Set automatically from the WSDL file name property.	The name of the message set in which the incoming message is defined. This value is set automatically to the message set that contains the WSDLfile when the WSDL is associated with the node. If you set this property, then later update the project dependencies to remove this message set reference, a warning is issued. Either update the Message set property, or restore the reference to this message set project. Input message parsing properties are disabled when the node is configured to act in gateway mode.
Message type	No	No		The name of the incoming message. The node detects the message type automatically. You cannot set this property.
Message format	No	No		The name of the physical format of the incoming message. You cannot set this property.

The SOAPInput node Parser Options properties are described in the following table.

Property	M	C	Default	Description
Parse timing	No	No	On demand	This property controls when an input message is parsed. Valid values are On demand, Immediate, and Complete. The default value, On demand, causes parsing of the message to be delayed. For a full description of this property, see "Parsing on demand" on page 4173.
Build tree using XML Schema data types	No	No	Selected	This property controls whether the syntax elements in the message tree have data types taken from the XML Schema. This property is cleared and disabled when the node is configured to act in gateway mode.
Retain mixed content	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters mixed text in an input message. If you select the check box, elements are created for mixed text. If you clear the check box, mixed text is ignored and no elements are created.
Retain comments	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters comments in an input message. If you select the check box, elements are created for comments. If you clear the check box, comments are ignored and no elements are created.
Retain processing instructions	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters processing instructions in an input message. If you select the check box, elements are created for processing instructions. If you clear the check box, processing instructions are ignored and no elements are created.

Property	M	C	Default	Description
Opaque elements	No	No	Blank	This property is used to specify a list of elements in the input message that are to be parsed opaquely. Opaque parsing is performed only if validation is not enabled (that is, if `Validate` is `None`); entries that are specified in `Opaque Elements` are ignored if validation is enabled.

The SOAPInput node Validation properties are described in the following table. See "Validation properties" on page 4169.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Validate	No	Yes	`Content and value`	This property controls whether the SOAP parser validates the body of each input message against XML schema generated from the message set. Valid values are `None`, `Content and value`, and `Content`. The SOAP parser invokes the XMLNSC parser to validate the XML body of the SOAP Web Service. If a message is propagated to the Failure terminal of the node, it is not validated. For more details, see "Validating messages" on page 1478 and "Validation properties" on page 4169. Validation properties are disabled, and the `Validate` property is set to `None`, when the node is configured to act in gateway mode.	validateMaster
Failure action	No	No	`Exception`	This property controls what happens if validation fails. You can set this property only if you set `Validate` to `Content` or `Content and value`. Valid values are `User Trace`, `Local Error Log`, `Exception`, and `Exception List`. Validation properties are disabled when the node is configured to act in gateway mode.	

The SOAPInput node Instances properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Additional instances pool	No	Yes	`Use Pool Associated with Message Flow`	This property specifies whether additional instance threads are allocated from a thread pool for the whole message flow, or from a thread pool for use by that node only. • If you select `Use Pool Associated with Message Flow`, additional instances are obtained from the message flow value. • If you select `Use Pool Associated with Node`, additional instances are allocated based on the number specified in the `Additional instances` property.	
Additional instances	No	Yes	`0`	The number of additional instances that the node can start if the `Additional instances pool` property is set to `Use Pool Associated with Node`. By default, no additional instances are given to the node.	additionalInstances

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Identity" on page 390
In WebSphere Message Broker, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

"What is SOAP?" on page 1604
SOAP is an XML message format used in Web service interactions. SOAP messages are typically sent over HTTP or JMS, but other transport protocols can be used. The use of SOAP in a specific Web service is described by a WSDL definition.

"SOAP parser and domain" on page 1082
You can use the SOAP parser to create a common WSDL-based logical tree format for working with Web services, independent of the physical bitstream format.

"WSDL URI formats for JMS" on page 1668
You must use WSDL to configure SOAP nodes. When using WSDL with a JMS transport, different URI formats can exist in the `address` element in the WSDL, which affect how properties are parsed and applied to the configured nodes.

"SOAP nodes" on page 1609
The SOAP nodes act as points in the flow where Web service processing is configured and applied. Properties on the SOAP nodes control the processing carried out and can be configured by supplying a WSDL definition, or by manually configuring properties, or both.

Related tasks:

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

"Setting up message flow security" on page 431
Set up security on a message flow to control access based on the identity of a message passing through the message flow.

"Configuring the extraction of an identity or security token" on page 447
You can configure the SecurityPEP node or security enabled input nodes to extract the identity or security token from a message and store it in the properties tree identity fields, enabling it to be processed throughout the message flow and propagated at output or request nodes.

"Importing from WSDL" on page 2946
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to create a new message definition from WSDL.

"Importing WSDL definitions from the command line" on page 2948
WSDL definitions can be imported using the (**mqsicreatemsgdefsfromwsdl**) command.

"Validating messages" on page 1478
The broker provides validation based on the message model for predefined messages.

"Using timeouts with HTTP and SOAP nodes" on page 1595
Connect the HTTP Timeout terminal of the HTTPInput or SOAPInput nodes to further nodes to process timeouts.

Related reference:

"Configuring the backout threshold property"
You can set the backout threshold property on nodes that use JMS transport to specify how many attempts are made to deliver the message to the input destination.

"Configuring for coordinated JMS transactions" on page 4817
Configure your message flow to receive or output messages under coordinated transactions.

"Validation properties" on page 4169
You can control validation by setting properties on the Validate and Parser Options tabs for the nodes that are listed in the following table.

"Parsing on demand" on page 4173
On-demand parsing, referred to as partial parsing, is used to parse an input message bit stream only as far as is necessary to satisfy the current reference. The parsers that can perform partial parsing of input messages are the JSON, MRM, XML, XMLNS, and XMLNSC parsers.

"WS-Addressing with the SOAPInput node" on page 1651
Various options are available when you use WS-Addressing with the SOAPInput node.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"SOAPAsyncResponse node" on page 4777
Use the SOAPAsyncResponse node in conjunction with the SOAPAsyncRequest node to construct a pair of message flows that call a Web service asynchronously.

"SOAPRequest node" on page 4828
Use the SOAPRequest node to send a SOAP request to the remote Web service.

"SOAPReply node" on page 4819
Use the SOAPReply node to send SOAP messages from the broker to the originating client in response to a message received by a SOAPInput node.

"SOAPExtract node" on page 4790
Use the SOAPExtract node to remove SOAP envelopes, allowing just the body of a SOAP message to be processed. It can also route a SOAP message based on its operation name. Both functions are optional; they are contained in one node because they are often used together.

Configuring the backout threshold property:

You can set the backout threshold property on nodes that use JMS transport to specify how many attempts are made to deliver the message to the input destination.

If the Backout threshold is set to `0` then redelivery is not attempted. If the Backout threshold is 1 or greater, the message will be redelivered the specified number of times.

Set the value of Backout threshold depending on the capabilities of the JMS provider.

If the JMS provider supports `JMSXDeliveryCount`, you can set the Backout threshold to any value.

If the JMS provider does not support `JMSXDeliveryCount`, the Backout threshold must only be set to 0 or 1. If the JMS provider does not support `JMSXDeliveryCount` and the value is set to greater than 1, a redelivered message is repeatedly backed out and reprocessed, and is never delivered to the backout destination.

Configuring for coordinated JMS transactions:

Configure your message flow to receive or output messages under coordinated transactions.

When you include a node using JMS transport in a message flow, such as the JMSInput or SOAPInput node when using JMS transport, the value that you set for `Transaction mode` defines whether messages are received under sync point.

- If you set this property to `Global`, the message is received under external sync point coordination; that is, within a WebSphere MQ unit of work. Any messages that are sent later, by an output node in the same instance of the message flow, are put under sync point, unless the output node overrides this setting explicitly.
- If you set this property to `Local`, the message is received under the local sync point control of the node. Any messages that are sent later, by an output node in the flow, are not put under local sync point, unless an individual output node specifies that the message must be put under local sync point.
- If you set this property to `None`, the message is not received under sync point. Any messages that are sent later, by an output node in the flow, are not put under sync point, unless an individual output node specifies that the message must be put under sync point.

To receive messages under external sync point, you must take additional configuration steps, which need be applied only the first time that a specific node using JMS transport is deployed to the broker for a particular JMS provider.

- On distributed systems, the external sync point coordinator for the broker is WebSphere MQ. Before you deploy a message flow in which the `Transaction mode` property is set to `Global` or `Yes`, and is intended to use globally coordinated transactions, modify the queue manager `.ini` file to include extra definitions for each JMS provider resource manager that participates in globally coordinated transactions.
  - Windows On Windows:
    1. Start WebSphere MQ Explorer.
    2. Right-click the queue manager name in the left pane and click **Properties**.
    3. Click **XA resource managers** in the left pane.
    4. Click **Add...** .
    5. Set the options as follows:
       - Set `Name` to any value.

- On Windows on x86 systems, set the SwitchFile property to *install_dir*\bin\JMSSwitch.dll. On Windows on x86-64 systems, set the SwitchFile property to JMSSwitch.dll.
- Set the XAOpenString property to a string value as follows: *Initial Context*,*location JNDI*,*Optional_parms*.
- Set the ThreadOfControl property to Thread.

6. On Windows on x86-64 systems only, copy the switch file *JMSSwitch32.dll* to the \exits subdirectory in the WebSphere MQ installation directory, and rename it to JMSSwitch.dll. Copy the switch file JMSSwitch.dll to the \exits64 subdirectory in the WebSphere MQ installation directory.

For more information, see the *System Administration Guide* section of the WebSphere MQ Version 7 Information Center online.

– Linux UNIX On Linux and UNIX systems, add a stanza to the queue manager .ini file for each JMS provider.

For example:

```
XAResourceManager:
Name=Jms_Provider_Name
SwitchFile=/install_dir/bin/ JMSSwitch.so
XAOpenString=Initial Context,location JNDI,Optional_parms
ThreadOfControl=THREAD
```

Where:

Name is an installation defined name that identifies a JMS provider resource manager.

SwitchFile
is the file system path to the JMSSwitch library that is supplied in the bin directory of the broker.

XAOpenString can have the following values:

– *Initial Context* is the value that is set in the JMSInput node property Initial context factory.
– *location JNDI* is the value that is set in the JMSInput node property Location JNDI bindings. This value must include a supported URL prefix that has a URL handler that is available on the class path.

The following parameters are optional:

– LDAP Principal matches the value that is set for the broker by using the **mqsicreatebroker** or **mqsichangebroker** commands.
– LDAP Credentials matches the value that is set for the broker by using the **mqsicreatebroker** or **mqsichangebroker** commands.
– Recovery Connection Factory Name is the JNDI administered connection factory that is defined in the bindings file. If a value is not specified, you must add a default value for recoverXAQCF to the bindings file. In either case, the Recovery Connection Factory must be defined as an XA Queue Connection Factory for the JMS provider that is associated with the Initial context factory.

The optional parameters are comma-separated and are positional. Therefore, any parameters that are missing must be represented by a comma. For example:

```
com.sun.jndi.fscontext.RefFSContextFactory,file:/C:/webservices/SOAP/JMS/JNDIXA,,,QCF
```

1. Update the Java CLASSPATH environment variable for the queue manager of the broker to include a reference to xarecovery.jar; for example:

```
install_dir/classes/xarecovery.jar
```

2. Update the Java PATH environment variable for the queue manager of the broker to point to the bin directory in which the SwitchFile is located; for example:

   install_dir/bin

   Finally, ensure that you have taken the following configuration steps:

   – In the message flow, ensure that the coordinated property is enabled by using the WebSphere Message Broker Archive editor.
   – Ensure that each node that must be part of the XA transaction is set to the global transaction mode.
   – Ensure that the service ID that is used for the broker and the queue manager is the same user ID.
   – Ensure that the JNDI connection factory objects that the JMS nodes use for a global transaction are configured to be of the type MQXAConnectionFactory, MQXAQueueConnectionFactory, or MQXATopicConnectionFactory.
     - If you create the bindings using WebSphere Message Broker Explorer, ensure the `Support XA Transactions` option is checked when you define your connection factory.
     - If you create the bindings using JMSAdmin, use the command **DEF XAQCF** or **DEF XATCF**, instead of **DEF QCF** or **DEF TCF**, when you define your connection factory.

   For more information, see the *System Administration Guide* section of the WebSphere MQ Version 7 Information Center online.

- z/OS On z/OS, the external sync point manager is Resource Recovery Services (RRS). The only JMS provider that is supported on z/OS is WebSphere MQ JMS. The only transport option that is supported for WebSphere MQ JMS on z/OS is the bind option.

  Sync point control for the JMS provider is managed with RRS sync point coordination of the queue manager of the broker. You do not have to modify the `.ini` file.

SOAPReply node

Use the SOAPReply node to send SOAP messages from the broker to the originating client in response to a message received by a SOAPInput node.

This topic contains the following sections:

Purpose:
The SOAPReply node is typically used with the SOAPInput node, which can be included in the same message flow, or a different flow in the same execution group.

You cannot use a SOAPReply node to respond to a Web service request that is received by an HTTPInput node; the broker generates an exception when the reply is attempted.

The SOAPReply node is contained in the **Web Services** drawer of the message flow node palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:
The SOAPReply node can be used in any message flow that needs to send SOAP messages from the broker to the originating client in response to a message received by a SOAPInput node. If a SOAPReply node is connected in a message flow that receives a one-way message, the message propagates to the Failure terminal of the SOAPReply node, and an exception is raised.

Look at the following sample to see how to use this node:

- SOAP Nodes

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Working with WrittenDestination data:
After the reply has been made, the WrittenDestination folder in the LocalEnvironment is updated if WS-Addressing is in use, and with transport details if WS-Addressing is in non-anonymous mode. A WrittenDestination for a SOAPReply node has the following format:

```
WrittenDestination = (
   SOAP  = (
      Reply = (
         WSA = (
            To = 'URI'
            MessageID = 'id'
            Action = 'doAllTheStuff'
         )      )
   )
)
```

LocalEnvironment overrides:
You can dynamically override set values in the local environment in the same way as setting values in other elements of a message. For a full list of values you can override in the local environment, see Local environment overrides.

Terminals and properties:
When you have put an instance of the SOAPReply node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The SOAPReply node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the message is routed if a failure is detected when the message is propagated.
Out	The output terminal to which the message is routed if it has been propagated successfully, and if further processing is required within this message flow.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The SOAPReply node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type: SOAPReply	The name of the node.
Short description	No	No	None	A brief description of the node.
Long description	No	No	None	Text that describes the purpose of the node in the message flow.

The SOAPReply node Transactions property is described in the following table. This setting does not apply when the node uses HTTP transport.

Property	M	C	Default	Description
Transaction mode	Yes	No	Automatic	This property controls whether the message is sent under a JMS transaction. Valid values are `Yes`, `No`, and `Automatic`. Select `No` to send the message using a non-transactional JMS session. Select `Yes` to output the message using a transactional JMS session. The JMS transaction can be either local or global. To use a global transaction, using an XA JMS session, you must also select the message flow property `Coordinated Transaction` in the BAR file properties. Select `Automatic` if you want the message transactionality to be inherited from the `Transaction mode` setting on the Input node at the start of the flow. See "Configuring for coordinated JMS transactions" on page 4823. The Transactions properties apply only to messages sent using JMS transport.

The SOAPReply node Validation properties are described in the following table. By default, validation is enabled.

If a message is propagated to the Failure terminal of the node, it is not validated. For more details, see "Validating messages" on page 1478 and "Validation properties" on page 4169.

Property	M	C	Default	Description	**`mqsiapplybaroverride` command property**
Validate	No	Yes	`Inherit`	This property controls whether validation takes place. Valid values are `None`, `Content and Value`, `Content`, and `Inherit`.	validateMaster
Failure action	No	No	`User trace`	This property controls what happens if validation fails. You can set this property only if you set `Validate` to `Content` or `Content and Value`. Valid values are `User trace`, `Local error log`, `Exception`, and `Exception list`.	

The SOAPReply node WS Extensions property is described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Allow MTOM	No	Yes	No	This property controls whether MTOM is enabled for the parser. Valid values are `Yes`, `No`, and `Inherit`. For more information about using SOAP MTOM with the SOAPReply, SOAPRequest, and SOAPAsyncRequest nodes, see "Using SOAP MTOM with the SOAPReply, SOAPRequest, and SOAPAsyncRequest nodes" on page 1678.	allowMTOM

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"What is SOAP?" on page 1604
SOAP is an XML message format used in Web service interactions. SOAP messages are typically sent over HTTP or JMS, but other transport protocols can be used. The use of SOAP in a specific Web service is described by a WSDL definition.

"SOAP nodes" on page 1609
The SOAP nodes act as points in the flow where Web service processing is configured and applied. Properties on the SOAP nodes control the processing carried out and can be configured by supplying a WSDL definition, or by manually configuring properties, or both.

Related tasks:

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

"Importing from WSDL" on page 2946
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to create a new message definition from WSDL.

"Importing WSDL definitions from the command line" on page 2948
WSDL definitions can be imported using the (**`mqsicreatemsgdefsfromwsdl`**) command.

Related reference:

"Configuring for coordinated JMS transactions"
Configure your message flow to receive or output messages under coordinated transactions.

"WS-Addressing with the SOAPReply node" on page 1653
Various options are available when you use WS-Addressing with the SOAPReply node.

"SOAPInput node" on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

"SOAPExtract node" on page 4790
Use the SOAPExtract node to remove SOAP envelopes, allowing just the body of a SOAP message to be processed. It can also route a SOAP message based on its operation name. Both functions are optional; they are contained in one node because they are often used together.

"SOAPRequest node" on page 4828
Use the SOAPRequest node to send a SOAP request to the remote Web service.

Configuring for coordinated JMS transactions:

Configure your message flow to receive or output messages under coordinated transactions.

When you include a node using JMS transport in a message flow, such as the JMSInput or SOAPInput node when using JMS transport, the value that you set for `Transaction mode` defines whether messages are received under sync point.

- If you set this property to `Global`, the message is received under external sync point coordination; that is, within a WebSphere MQ unit of work. Any messages that are sent later, by an output node in the same instance of the message flow, are put under sync point, unless the output node overrides this setting explicitly.
- If you set this property to `Local`, the message is received under the local sync point control of the node. Any messages that are sent later, by an output node in the flow, are not put under local sync point, unless an individual output node specifies that the message must be put under local sync point.
- If you set this property to `None`, the message is not received under sync point. Any messages that are sent later, by an output node in the flow, are not put under sync point, unless an individual output node specifies that the message must be put under sync point.

To receive messages under external sync point, you must take additional configuration steps, which need be applied only the first time that a specific node using JMS transport is deployed to the broker for a particular JMS provider.

- On distributed systems, the external sync point coordinator for the broker is WebSphere MQ. Before you deploy a message flow in which the `Transaction mode` property is set to `Global` or `Yes`, and is intended to use globally coordinated transactions, modify the queue manager `.ini` file to include extra definitions for each JMS provider resource manager that participates in globally coordinated transactions.
  - Windows On Windows:
    1. Start WebSphere MQ Explorer.
    2. Right-click the queue manager name in the left pane and click **Properties**.
    3. Click **XA resource managers** in the left pane.
    4. Click **Add...** .

5. Set the options as follows:
   - Set `Name` to any value.
   - On Windows on x86 systems, set the `SwitchFile` property to *`install_dir`*`\bin\JMSSwitch.dll`. On Windows on x86-64 systems, set the `SwitchFile` property to `JMSSwitch.dll`.
   - Set the `XAOpenString` property to a string value as follows: *`Initial Context`*`,`*`location JNDI`*`,`*`Optional_parms`*.
   - Set the `ThreadOfControl` property to `Thread`.
6. On Windows on x86-64 systems only, copy the switch file *JMSSwitch32.dll* to the `\exits` subdirectory in the WebSphere MQ installation directory, and rename it to `JMSSwitch.dll`. Copy the switch file `JMSSwitch.dll` to the `\exits64` subdirectory in the WebSphere MQ installation directory.

For more information, see the *System Administration Guide* section of the WebSphere MQ Version 7 Information Center online.

– Linux UNIX On Linux and UNIX systems, add a stanza to the queue manager `.ini` file for each JMS provider.

For example:

```
XAResourceManager:
Name=Jms_Provider_Name
SwitchFile=/install_dir/bin/ JMSSwitch.so
XAOpenString=Initial Context,location JNDI,Optional_parms
ThreadOfControl=THREAD
```

Where:

Name is an installation defined name that identifies a JMS provider resource manager.

SwitchFile
is the file system path to the JMSSwitch library that is supplied in the `bin` directory of the broker.

XAOpenString can have the following values:

– *Initial Context* is the value that is set in the JMSInput node property `Initial context factory`.

– *location JNDI* is the value that is set in the JMSInput node property `Location JNDI bindings`. This value must include a supported URL prefix that has a URL handler that is available on the class path.

The following parameters are optional:

– `LDAP Principal` matches the value that is set for the broker by using the **`mqsicreatebroker`** or **`mqsichangebroker`** commands.

– `LDAP Credentials` matches the value that is set for the broker by using the **`mqsicreatebroker`** or **`mqsichangebroker`** commands.

– `Recovery Connection Factory Name` is the JNDI administered connection factory that is defined in the bindings file. If a value is not specified, you must add a default value for `recoverXAQCF` to the bindings file. In either case, the Recovery Connection Factory must be defined as an XA Queue Connection Factory for the JMS provider that is associated with the `Initial context factory`.

The optional parameters are comma-separated and are positional. Therefore, any parameters that are missing must be represented by a comma. For example:

```
com.sun.jndi.fscontext.RefFSContextFactory,file:/C:/webservices/SOAP/JMS/JNDIXA,,,QCF
```

1. Update the Java CLASSPATH environment variable for the queue manager of the broker to include a reference to `xarecovery.jar`; for example:

install_dir/classes/xarecovery.jar

2. Update the Java PATH environment variable for the queue manager of the broker to point to the bin directory in which the SwitchFile is located; for example:

   install_dir/bin

Finally, ensure that you have taken the following configuration steps:

- In the message flow, ensure that the coordinated property is enabled by using the WebSphere Message Broker Archive editor.
- Ensure that each node that must be part of the XA transaction is set to the global transaction mode.
- Ensure that the service ID that is used for the broker and the queue manager is the same user ID.
- Ensure that the JNDI connection factory objects that the JMS nodes use for a global transaction are configured to be of the type MQXAConnectionFactory, MQXAQueueConnectionFactory, or MQXATopicConnectionFactory.
  - If you create the bindings using WebSphere Message Broker Explorer, ensure the `Support XA Transactions` option is checked when you define your connection factory.
  - If you create the bindings using JMSAdmin, use the command **DEF XAQCF** or **DEF XATCF**, instead of **DEF QCF** or **DEF TCF**, when you define your connection factory.

For more information, see the *System Administration Guide* section of the WebSphere MQ Version 7 Information Center online.

- z/OS On z/OS, the external sync point manager is Resource Recovery Services (RRS). The only JMS provider that is supported on z/OS is WebSphere MQ JMS. The only transport option that is supported for WebSphere MQ JMS on z/OS is the bind option.

  Sync point control for the JMS provider is managed with RRS sync point coordination of the queue manager of the broker. You do not have to modify the `.ini` file.

Local environment overrides for the SOAPReply node:

You can dynamically override set values in the local environment in the same way as setting values in other elements of a message.

Other local environment overrides are available for WS-Addressing. See "WS-Addressing with the SOAPReply node" on page 1653.

You can set the following HTTP properties in the SOAPReply node under `LocalEnvironment.Destination.SOAP.Request.Transport.HTTP.AsyncReply`. These properties apply only when you make an HTTP reply with WS-Addressing. You can switch between HTTP and JMS transport using the WS-Addressing `To` field; see "WS-Addressing information in the local environment" on page 1656.

Setting	Description
HTTPVersion	Overrides the `HTTPVersion`. For example: `SET OutputLocalEnvironment.Destination.SOAP.Reply.Transport.HTTP.AsyncReply.HTTPVersion = 'HTTP/1.1';`
Method	Overrides the reply message `Method`. For example: `SET OutputLocalEnvironment.Destination.SOAP.Reply.Transport.HTTP.AsyncReply.Method = 'GET';`

Setting	Description

ProxyConnectHeaders	Specifies additional headers that are used if the outbound request is an SSL connection through a proxy. These additional headers are sent with the initial CONNECT request to the proxy. For example, you can send proxy authentication information to a proxy server when you are using SSL. You can send multiple headers but each one must be separated by a carriage return and a line feed (ASCII 0x0D 0x0A), in accordance with RFC2616; for example:
	`DECLARE CRLF CHAR CAST(X'0D0A' AS CHAR CCSID 1208);`
	`SET OutputLocalEnvironment.Destination.SOAP.Reply.Transport.HTTP.AsyncReply.ProxyConnectHeaders = 'Proxy-Authorization: Basic Zm5lcmJsZTpwYXNzd29yZA==' || CRLF || 'Proxy-Connection: Keep-Alive' || CRLF;`
	
	This setting is used only if the request is an SSL request through a proxy server. To send proxy authentication information for a non-SSL request, specify the individual headers in the HTTPRequestHeader folder, as shown in the following example:
	`SET OutputRoot.HTTPReplyHeader."Proxy-Authorization" = 'Basic Zm5lcmJsZTpwYXNzd29yZA==';`
	`SET OutputRoot.HTTPReplyHeader."Proxy-Connection" = 'Keep-Alive';`

ProxyURL	Overrides the reply message HTTP(S) proxy location. For example: `SET OutputLocalEnvironment.Destination.SOAP.Reply.Transport.HTTP.AsyncReply.ProxyURL = 'my.proxy';`
RequestURI	Overrides the reply message RequestURI, which is the path after the URL and port. For example: `SET OutputLocalEnvironment.Destination.SOAP.Reply.Transport.HTTP.AsyncReply.RequestURI = '/abc/def?x=y&g=h';`
SSLCiphers	Overrides the reply message Allowed SSL Ciphers (if using SSL). For example: `SET OutputLocalEnvironment.Destination.SOAP.Reply.Transport.HTTP.AsyncReply.SSLCiphers = 'SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA';`
SSLProtocol	Overrides the reply message SSLProtocol. For example: `SET OutputLocalEnvironment.Destination.SOAP.Reply.Transport.HTTP.AsyncReply.SSLProtocol = 'TLS';` Valid values are SSL, SSLv3, and TLS.
Timeout	Overrides the reply message Request timeout (in seconds). For example: `SET OutputLocalEnvironment.Destination.SOAP.Reply.Transport.HTTP.AsyncReply.Timeout = 42;`
WebServiceURL	Overrides the reply message Web service URL. For example: `SET OutputLocalEnvironment.Destination.SOAP.Reply.Transport.HTTP.AsyncReply.WebServiceURL = 'http://ibm.com/abc/';`

You can set the following JMS properties in the SOAPReply node under LocalEnvironment.Destination.SOAP.Reply.Transport.JMS. These properties apply only when you make a JMS reply with WS-Addressing. You can switch between HTTP and JMS transport using the WS-Addressing To field; see "WS-Addressing information in the local environment" on page 1656.

Setting	Description
BindingsLocation	Overrides the reply message JNDI URL bindings location. For example: `SET OutputLocalEnvironment.Destination.SOAP.Reply.Transport.JMS.JNDI.BindingsLocation = 'file:/C:/mqsi6/Webservices/SOAP/JMS/JNDI';`
ConnectionFactoryName	Overrides the reply message Connection factory name. For example: `SET OutputLocalEnvironment.Destination.SOAP.Reply.Transport.JMS.JNDI.ConnectionFactoryName = 'QCF';`

Setting	Description

ContextParameters	Overrides the reply message JNDI context parameters. You can override specific JNDI context parameters, for example:
	`SET OutputLocalEnvironment.Destination.SOAP.Reply.Transport.JMS.JNDI.ContextParameters.foo = 'bar';`
	`SET OutputLocalEnvironment.Destination.SOAP.Reply.Transport.JMS.JNDI.ContextParameters.foo2 = 'baz';`

CorrelationID	Sets the reply message CorrelID. For example: `SET OutputLocalEnvironment.Destination.SOAP.Reply.Transport.JMS.CorrelationID = 'myCorrelID';`
Destination	Overrides the reply message Destination. For example: `SET OutputLocalEnvironment.Destination.SOAP.Reply.Transport.JMS.Destination = 'REPLYTOQ2';`
DeliveryMode	Overrides the reply message DeliveryMode. For example: `SET OutputLocalEnvironment.Destination.SOAP.Reply.Transport.JMS.DeliveryMode = 'NON_PERSISTENT';` Allowed values for this property are PERSISTENT and NON_PERSISTENT. If the UriFormat is ibm, 1 and 0 are additional allowed values for DeliveryMode.
Expiration	Overrides the reply message Expiration. This property is specified in milliseconds. For example, to set an expiration of 100 milliseconds: `SET OutputLocalEnvironment.Destination.SOAP.Reply.Transport.JMS.Expiration = '100';`
InitialContextFactory	Overrides the reply message Initial context factory. For example: `SET OutputLocalEnvironment.Destination.SOAP.Reply.Transport.JMS.JNDI.InitialContextFactory = 'com.sun.jndi.fscontext.RefFSContextFactory';`
MessagePriority	Overrides the reply message MessagePriority. For example: `SET OutputLocalEnvironment.Destination.SOAP.Reply.Transport.JMS.MessagePriority = '7';`
MessageType	Overrides the reply message MessageType. For example: `SET OutputLocalEnvironment.Destination.SOAP.Reply.Transport.JMS.MessageType = 'text';` Allowed values for this property are text and bytes.
ProviderName	Overrides the reply message JMS provider name. For example: `SET OutputLocalEnvironment.Destination.SOAP.Reply.Transport.JMS.ProviderName = 'WebSphere MQ';`
TransactionMode	Overrides the Transaction mode property on the node. For example: `SET OutputLocalEnvironment.Destination.SOAP.Reply.Transport.JMS.TransactionMode = 'Yes';` Allowable values for this property are Yes, No and ForceLocal. • No means that no transaction takes place, and is equivalent to None. • Yes means that a local transaction takes place if the flow's Coordinated Transaction is not selected, or a global transaction takes place if the flow's Coordinated Transaction property is selected. • ForceLocal means that a local transaction is always used, even if the flow's Coordinated Transaction property is selected.
UriFormat	Overrides the reply message UriFormat. For example: `SET OutputLocalEnvironment.Destination.SOAP.Reply.Transport.JMS.UriFormat = 'w3c';` Allowable values for this property are w3c and ibm.

Setting	Description

UserProperties	Overrides the `User Context Parameters` on the node. You can override specific user context parameters, for example:
	`SET OutputLocalEnvironment.Destination.SOAP.Reply.Transport.JMS.JNDI.UserProperties.property1 = 'value1';`
	`SET OutputLocalEnvironment.Destination.SOAP.Reply.Transport.JMS.JNDI.UserProperties.property2 = 'value2';`

Setting	Description
OneWay	Instructs the node that the inbound message was a one-way message. The node resources are cleared because no reply message is needed. For example: `SET OutputLocalEnvironment.Destination.SOAP.Reply.Gateway.OneWay = True` Additionally, if the inbound message used HTTP transport, the node sends an acknowledgment HTTP 202 message. For more information, see "One-way messages in Gateway mode" on page 1648.

SOAPRequest node

Use the SOAPRequest node to send a SOAP request to the remote Web service.

This topic contains the following sections:
- "Purpose"
- "Using this node in a message flow"
- "Terminals and properties"
- "Working with WrittenDestination data" on page 4843
- "Local environment overrides" on page 4844

Purpose:
The SOAPRequest node is a synchronous request and response node, which blocks processing after sending the request until the response is received. This node enables the HTTP 1.1 Keep-Alive method by default.

The SOAPRequest node is contained in the **Web Services** drawer of the message flow node palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:
The SOAPRequest node can be used in any message flow that needs to call a Web service. Look at the following sample to see how to use this node:
- SOAP Nodes

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Terminals and properties:
The SOAPRequest node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which a message is routed if a failure is detected when the message is propagated to the Out flow (such as a message validation failure). Failures routed to this terminal include failures caused by the retry processing occurring before the retry propagates the message to the Out flow.
Out	The output terminal to which the message is routed if the SOAP request has been sent and responded to successfully, and if further processing is required within this message flow. If no errors occur within the SOAPRequest node and a none fault SOAP response is received from the external resource it is always sent to the Out terminal first.
Fault	SOAP fault messages received in response to the sent request are directed to the Fault terminal. If no connection is provided to the Fault terminal no further processing occurs for a received fault within this message flow.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined; the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

Some SOAPRequest node properties are initially set from properties in the imported WSDL. These properties are parsed differently depending on which URI format is used by the `address` element in the WSDL. For details, see "WSDL URI formats for JMS" on page 1668.

The SOAPRequest node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type	The name of the node.
Short description	No	No	None	A brief description of the node.
Long description	No	No	None	Text that describes the purpose of the node in the message flow.

The SOAPRequest node Basic properties are described in the following table.

Property	M	C	Default	Description
Operation mode	Yes	Yes	`Invoke a specific web service defined by a WSDL interface`	This property allows you to specify the operation mode of the node, which determines whether it acts in WSDL mode or in gateway mode. In WSDL mode, the node performs operations according to the WSDL it is configured with. However, gateway mode allows you to configure your flow to handle generic SOAP request/response and one-way messages, or to act as a façade between multiple web services clients and multiple back-end web services providers. **Invoke a specific web service defined by a WSDL interface** Configure the node with a deployable WSDL by setting the WSDL file name property or by dragging a WSDL onto the node. This is the default option. **Invoke a generic web service** Configure the node to act in gateway mode with no WSDL required. See “Gateway operation mode for SOAP nodes” on page 1645 for a fuller explanation of gateway mode.
WSDL file name	Yes	No	None	This property indicates the location of the WSDL file that you want to use to configure the node. Enter the full path to the WSDL file, or click **Browse** to locate the WSDL file in your workspace. When you select a WSDL file for the `WSDL file name` property, the WSDL is validated to ensure that it is WS-I compliant. If the WSDL has a binding using SOAP/JMS which is not WS-I compliant, by default no error is shown. To enable strict WS-I validation and display a warning when a SOAP/JMS transport is used, click **Window > Preferences > Broker Development > Message Sets > Validation** and clear the **WS-I BP 1.1: Allow SOAP/JMS as transport URI** check box. Only deployable WSDL files can be used to configure the SOAP nodes. After a valid WSDL file is selected, the message set project to which the WSDL file belongs is added as a referenced project to the corresponding message flow project, if the reference does not exist. If the WSDL file is not valid, or an incorrect file name is entered, an error message is displayed in the Properties view and all WSDL properties are blank. If the node was created by dropping a WSDL file from a message set onto the Message Flow editor, this property is preset to the name of the WSDL file. This property takes a string value. The following situations result in an error condition: • The WSDL file does not belong to a message set project, or the WSDL file was not imported correctly; see “Importing from WSDL” on page 2946 and “Importing WSDL definitions from the command line” on page 2948. • The WSDL file contains no HTTP or JMS bindings. • The WSDL file contains no port type. • The WSDL file that is specified in the field does not exist. WSDL properties are disabled when the node is configured to act in gateway mode.

Property	M	C	Default	Description
Port type	Yes	No	By default, the first `Port type` found in the WSDL file, that has an associated HTTP binding with it, is selected.	This property type is String. The field lists all the `Port types` defined in WSDL file selected in the `WSDL file name` property. Error Conditions: • Selected `Port type` does not contain at least one operation. WSDL properties are disabled when the node is configured to act in gateway mode.
Imported binding	Yes	No		This property type is String. This property is updated every time that the `Port type` value changes. The field lists the imported SOAP bindings with HTTP or JMS transport associated with the selected `Port type`. When you select a binding, the property tab for the associated transport is enabled; otherwise, it is disabled. Bindings are listed in the same order in which they appear in the WSDL file. The selected binding is the one that has both ports and operations. If there is no such binding, then binding with ports is selected. If no bindings have ports then the first binding in the list is selected. Error Conditions: • No SOAP bindings (with HTTP or JMS transport) in the WSDL file are associated with the `Port type`. • The selected binding does not have any operations. WSDL properties are disabled when the node is configured to act in gateway mode.
Binding operation	Yes	No		This property type is String. The `Binding operation` box contains all the operations defined by the selected binding. The first operation in the list is selected by default. This property is updated every time the selected binding value changes WSDL properties are disabled when the node is configured to act in gateway mode.
Service port	Yes	No		This property type is String. This field is updated every time that the selected binding is updated. This field lists all the WSDL ports that point to the selected binding. The first service port for the binding is selected by default. This property is updated every time the selected binding value changes. Error Conditions: • No ports point to the selected binding. WSDL properties are disabled when the node is configured to act in gateway mode.
Target namespace	Yes	No		`Target namespace` is implemented as a read-only field. This hidden property type is String. It is updated with the `Target namespace` of the WSDL file when the `WSDL file name` is configured. WSDL properties are disabled when the node is configured to act in gateway mode.

Property	M	C	Default	Description
Transport	No	No		This property is set automatically when the `Imported binding` property is selected. The value of this property shows the transport used by the selected WSDL binding; for example, `HTTP` or `JMS`. If you choose to switch the transport from `JMS` to `HTTP`, a dialog box displays, which allows you to reset the JMS-specific properties. You must reset the JMS properties to deploy the message flow to a runtime environment version prior to fix pack V7.0.0.1.

The SOAPRequest node HTTP Transport properties are described in the following table. These settings are used only when the node uses HTTP transport

Property	M	C	Default	Description	**`mqsiapplybaroverride` command property**
Web service URL	Yes	Yes	SOAP address of the selected port	The URL of the SOAP address selected. This property is automatically derived from the `<soap:address>` element of the selected `Service port`. Whenever the selected port is updated, the `Web service URL` is updated accordingly. However, if you override the value then your value persists and the URL is no longer updated from the service port. If you choose to override this property you must specify it in the form http://<hostname>[:<port>]/[<path>] where: • http://<hostname> must be specified. • <port> has a default of 80. If you specify a value, you must include the colon : before the port number. • <path> has a default of /. If you specify a value, you must include the / before the path. For more details of how to override this property, see Changing the default URL for a SOAPRequest node or a SOAPAsyncRequest node request.	webServiceURL
Request timeout (in seconds)	No	Yes	120	The number of seconds that the client waits for a remote server to respond with a 'message received' acknowledgment. The timeout might take up to one second longer than the value specified. If no response is received in this time, a SOAP Fault exception is raised and is propagated to the Failure terminal.	
HTTP(S) proxy location	No	Yes	Blank	The location of the proxy server to which requests are sent.	httpProxyLocation

Property	M	C	Default	Description	mqsiapplybaroverride command property
Protocol (if using SSL)	No	Yes	TLS	The selected protocol if you use SSL. This property type is Enumerate. The following options are available: **SSL** This option attempts to connect by using the SSLv3 protocol first, but allows the handshake to fall back to the SSLv2 protocol where the SSLv2 protocol is supported by the underlying JSSE provider. **SSLv3** This option attempts to connect with the SSLv3 protocol only. Fallback to SSLv2 is not allowed. **TLS** The default. This option attempts to connect with the TLS protocol only. Fallback to SSLv3 or SSLv2 is not allowed. **TLSv1** This option attempts to connect with the TLS v1.0 protocol only. Fallback to SSLv3 or SSLv2 is not allowed. **TLSv1.1** This option attempts to connect with the TLS v1.1 protocol only. Fallback to SSLv3, SSLv2, or TLSv1.0 is not allowed. **TLSv1.2** This option attempts to connect with the TLS v1.2 protocol only. Fallback to SSLv3, SSLv2, TLSv1.0, or TLSv1.1 is not allowed. **SSL_TLS** This option enables all SSL v3.0 and TLS v1.0 protocols. Fallback to SSLv2 is not allowed. **SSL_TLSv2** This option enables all SSL v3.0 and TLS v1.0, v1.1, and v1.2 protocols. Fallback to SSLv2 is not allowed. Both ends of an SSL connection must agree on the protocol to use; therefore, the chosen protocol must be one that the remote server can accept.	sslProtocol

Property	M	C	Default	Description	mqsiapplybaroverride command property
Allowed SSL ciphers (if using SSL)	No	Yes	None	The specific SSL cipher, or ciphers, that you are using. This setting allows you to specify a single cipher (such as SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA) or a list of ciphers that are the only ones used by the connection. This set of ciphers must include one or more that are accepted by the remote server. You can specify a list of ciphers in priority order. The server selects the first acceptable cipher in the list. If none of the ciphers in the list are suitable, the server returns a handshake failure alert and closes the connection. A comma is used as a separator between the ciphers. The default value is an empty string, which allows the node to use any, or all, of the available ciphers during the SSL connection handshake. This method allows the greatest scope for making a successful SSL connection.	allowedSSLCiphers
Use compression	No	No	None	This property controls whether the content of the HTTP request is compressed. Valid values are none, gzip, zlib (deflate) and deflate. If the request is compressed, the Content-Encoding header is set to indicate that the content is compressed. zlib (deflate) represents RFC 1950 + RFC 1951 combined. deflate represents RFC 1951 only.	requestCompressionType
Accept compressed responses by default	No	Yes	Cleared	This property indicates whether the request accepts compressed responses. If this option is selected, it is possible for the request to receive responses with a Content-Encoding of gzip or deflate. If such a response is received, the content is decoded and the Content-Encoding header is removed. If the Request Header does not contain an Accept-Encoding header then selecting this option sets the Accept-Encoding header to "gzip, deflate".	acceptCompressedResponses
Perform hostname checking (if using SSL)	No	Yes	No	This property specifies if the host name of the server that is receiving the request must match the host name in the SSL certificate.	hostnameChecking

The SOAPRequest node JMS Transport properties are described in the following table. These settings are used only when the node uses JMS transport

Property	M	C	Default	Description	mqsiapplybaroverride command property
Destination	Yes	Yes	None	The destination to which the node sends outgoing messages. If the SOAPRequest node is to be used to send point-to-point messages, enter the Destination queue name for the JMS queue name that is listed in the bindings file. This property takes its initial value from a WSDL URI property, depending on whether the WSDL `address` URI is formatted in the W3C (standards) style, or the IBM (proprietary) style. `Destination` is set to the value of `destinationName` found in the WSDL if a W3C-style URI is found, or `destination` if an IBM-style URI is found.	jmsDestination
Reply To Destination	No	Yes	None	The name of the JMS destination to which the receiving application must send a reply message. This property is not used if the JMS operation is one way. For a reply message to be returned to this JMS destination, the JMS destination name must be known to the domain of the JMS provider that is used by the receiving client. If you do not specify a reply-to destination, a temporary dynamic queue is used as the reply-to destination. If you do not specify a reply-to destination and you are using the WebSphere MQ JMS provider, you must configure JMS temporary dynamic queues. For more information about configuring JMS temporary dynamic queues, see "Configuring JMS temporary dynamic queues for the WebSphere MQ JMS provider" on page 4848. This property takes its initial value from a WSDL URI property, depending on whether the WSDL `address` URI is formatted in the W3C (standards) style, or the IBM (proprietary) style. `Reply To Destination` is set to the value of `replyToName` found in the WSDL if a W3C-style URI is found, or to the first of `replyToName`, `replyTo`, `replyToDestination`, or `replyDestination` if an IBM-style URI is found. If any of these other properties are present, they display as a name-value pair in the `User Parameters` table.	jmsReplyToDestination
Request timeout (in seconds)	No	Yes	120	The time the client waits for a remote server to send a response message before timing out.	requestTimeout

Property	M	C	Default	Description	mqsiapplybaroverride command property
JMS provider name	Yes	No	WebSphere MQ	Select a JMS vendor name from the list, or enter a name of your choice. The name must match the name of a configurable service that is defined for the broker to which you deploy the message flow. When you select a name from the list, the Initial context factory property is updated automatically with the relevant Java class. If you enter your own JMS provider name, you must also enter a value for the Initial context factory.	
Initial context factory	Yes	Yes	com.sun.jndi.fscontext.RefFSContextFactory	The starting point for a JNDI namespace. A JMS application uses the initial context to obtain and look up the connection factory and queue or topic objects for the JMS provider. If you select a JMS provider name from the list in JMS provider name, the Initial context factory property is updated automatically with the relevant Java class. If you enter your own JMS provider name, you must also enter a value for the Initial context factory. The default value is com.sun.jndi.fscontext.RefFSContextFactory, which defines the file-based Initial context factory for the WebSphere MQ JMS provider. This property takes its initial value from a WSDL URI property, depending on whether the WSDL address URI is formatted in the W3C (standards) style, or the IBM (proprietary) style. Initial context factory is set to the value of jndiInitialContextFactory found in the WSDL if a W3C-style URI is found, or initialContextFactory if an IBM-style URI is found.	initialContextFactory

Property	M	C	Default	Description	**mqsiapplybaroverride** command property
JNDI URL bindings location	Yes	Yes		The system path or the LDAP location for the bindings file. The bindings file contains definitions for the JNDI administered objects that are used by the SOAPRequest node. This property is disabled when the `Initial context factory` is `com.ibm.mq.jms.Nojndi`. When you enter a value for `JNDI URL bindings location`, ensure that it complies with the following instructions: • Construct the bindings file before you deploy a message flow that contains a SOAPRequest node. • Do not include the file name of the bindings file in this field. • If you have specified an LDAP location that requires authentication, configure the LDAP principal (userid) and LDAP credentials (password) separately. These values are configured at broker level. For information about configuring these values, see "**mqsicreatebroker** command" on page 3831 and "**mqsichangebroker** command" on page 3723. • The string value must include a supported URL prefix that has a URL handler that is available on the class path. For information about constructing the JNDI administered objects bindings file, see the JMS provider documentation. This property takes its initial value from a WSDL URI property, depending on whether the WSDL `address` URI is formatted in the W3C (standards) style, or the IBM (proprietary) style. `JNDI URL bindings location` is set to the value of `jndiURL` found in the WSDL if a W3C-style URI is found, or `jndiProviderURL` if an IBM-style URI is found.	locationJndiBindings

Property	M	C	Default	Description	mqsiapplybaroverride command property
Connection factory name	Yes	Yes		The name of the connection factory that is used by the SOAPRequest node to create a connection to the JMS provider. This property is initially configured from the imported WSDL. This name must exist in the bindings file. The `Connection factory name` must be a JMS QueueConnectionFactory. This property takes its initial value from a WSDL URI property, depending on whether the WSDL `address` URI is formatted in the W3C (standards) style, or the IBM (proprietary) style. `Connection factory name` is set to the value of `jndiConnectionFactoryName` found in the WSDL if a W3C-style URI is found, or `connectionFactory` if an IBM-style URI is found.	connectionFactoryName
User Parameters	No	No		This table describes user properties that have been defined in the request. The properties are name-value pairs that exist in the WSDL and are not used by other properties of the SOAPRequest node.	
JNDI parameters	No	No		A table mapping JNDI context parameters to their type. These properties take their initial values from any W3C-style WSDL properties starting with `jndi-`. IBM-style WSDL does not support JNDI parameters, but you can set these properties on the node.	

The SOAPRequest node Message Delivery properties are described in the following table. This sub tab is enabled only if the selected binding in the Basic tab uses JMS transport.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Target Service	No	No	None	Used by the SOAPRequest node when dispatching the service request. This property takes its initial value from the `targetService` WSDL property.	targetService

Property	M	C	Default	Description	mqsiapplybaroverride command property
Delivery mode	No	Yes	Persistent	This property controls the persistence mode that a JMS provider uses for a message. Valid values are: • `Persistent`: the message survives if the JMS provider has a system failure. • `Non Persistent`: the message is lost if the JMS provider has a system failure. This property takes its initial value from a WSDL URI property, depending on whether the WSDL `address` URI is formatted in the W3C (standards) style, or the IBM (proprietary) style. `Delivery mode` is set to the value of `deliveryMode` found in the WSDL if a W3C-style URI is found, or to the first of `deliveryMode` or `persistence` if an IBM-style URI is found. If both these properties are present, the second property displays as a name-value pair in the `User Parameters` table.	deliveryMode
Message Priority	No	Yes	4	This property assigns relative importance to the message and can be used for message selection by a receiving web service. Select a value between 0 (lowest priority) and 9 (highest priority). The default value is 4, which indicates medium priority. Priorities in the range 0 - 4 indicate typical delivery. Priorities in the range 5 - 9 indicate faster delivery. This property takes its initial value from a WSDL URI property, depending on whether the WSDL `address` URI is formatted in the W3C (standards) style, or the IBM (proprietary) style. `Priority` is set to the value of `priority` found in the WSDL if a W3C-style URI is found, or to the first of `priority` or `Priority` if an IBM-style URI is found. If both these properties are present, the second property displays as a name-value pair in the `User Parameters` table.	messagePriority
Message Expiration (ms)	No	Yes	0	This property controls the length of time, in milliseconds, for which the JMS provider keeps the output JMS message. The default value, 0, is used to indicate that the message must not expire. This property takes its initial value from the `timeToLive` WSDL property.	messageExpiration
Message Type	No	Yes	bytes	Select a value from the list to configure the type of JMS message that is produced by the SOAPRequest node.	messageType

The SOAPRequest node Transactions properties are described in the following table. This setting does not apply when the node uses HTTP transport.

Property	M	C	Default	Description
Transaction Mode	Yes	No	Automatic	This property controls whether the message is output under a JMS transaction. Valid values are `Yes`, `No`, and `Automatic`. Select `No` to output the message using a non-transactional JMS session. Select `Yes` to output the message using a transactional JMS session. The JMS transaction can be either local or global. To use a global transaction, using an XA JMS session, you must also select the message flow property `Coordinated Transaction` in the BAR file properties. Select `Automatic` if you want the message transactionality to be inherited from the `Transaction mode` setting on the Input node at the start of the flow. This value is only used when the selected operation is one-way. See "Configuring for coordinated JMS transactions" on page 4846.

The SOAPRequest node Advanced properties are described in the following table.

Property	M	C	Default	Description
WSDL-defined SOAP response headers	No	No		This table is read-only, and is populated by the SOAP headers defined in the output part of the selected operations. By default, the check boxes, in the second column of the table, are cleared for all entries in the `WSDL-defined SOAP response headers` table. You must select the relevant check box to add the header to the `must understand headers` list. SOAP headers that are part of the `must understand headers` list are incorporated into the flow rather than causing a SOAP fault. Adding headers to the `must understand headers` list stops SOAP faults being generated by SOAP headers. You do not need to add `must understand headers` for WS-Addressing and WS-Security as they are understood if you configure **WS Extensions**.The table is updated automatically when the selected operation is updated. This property is generated in the CMF file. When the node is configured to act in gateway mode, with no WSDL required, this table is cleared. The original values of these fields are restored if the operation mode of the node is changed back to WSDL mode.
User-defined SOAP response headers	No	Yes	None	You can add custom headers (headers that are not defined in the WSDL file) in this table. Use **Add**, **Edit**, and **Delete** for this table. You must select the check box, in the second column of the table, to ensure that the newly added custom header is added to the `must understand headers` list. This property is generated in the CMF file.

The SOAPRequest node WS Extensions properties are described in the following table.

Property	M	C	Default	Description	
Use WS-Addressing	No	No		This property specifies whether to use WS-Addressing. For more details about WS-Addressing with the SOAPRequest node, see "WS-Addressing with the SOAPRequest node" on page 1653.	

Property	M	C	Default	Description	
Place WS-Addressing headers into LocalEnvironment	No	No	Cleared	This property specifies whether the node puts WS-Addressing headers received in the response message into the local environment tree. WS-Addressing headers are not accessible to the flow if this check box is cleared because by default, all headers are processed and removed.	
Allow MTOM	No	Yes	No	This property controls whether MTOM is enabled for the parser. Valid values are `Yes`, `No`, and `Inherit`. For more information about using SOAP MTOM with the SOAPReply, SOAPRequest, and SOAPAsyncRequest nodes, see "Using SOAP MTOM with the SOAPReply, SOAPRequest, and SOAPAsyncRequest nodes" on page 1678. MTOM support is disabled when the node is configured to act in gateway mode.	allowMTOM
WS-Security	No	Yes		This complex property is in the form of a table and consists of two columns: • `Alias` • `XPath Expression` You can add XPath expressions with an associated `Alias` value to the `WS-Security` table. The `Alias` is resolved in a Policy Set that is created by the administrator. The Policy Set resolves the `Alias` to either encrypt or sign the part of the message referred to by the `XPath Expression`. You can **Add**, **Edit**, and **Delete** in this table.	

The SOAPRequest node Response Message Parsing properties are described in the following table.

Property	M	C	Default	Description
Message domain	No	No	SOAP	The domain that is used to parse the response message. By default, the message that is propagated from the SOAPInput node is in the SOAP domain. You cannot specify a different domain. For more information, see "SOAP parser and domain" on page 1082. The Response Message Parsing properties are disabled when the node is configured to act in gateway mode.

Property	M	C	Default	Description
Message set	Yes	No	Set automatically from the `WSDL file name` property.	The name of the message set in which the response message is defined. This property is automatically set to the message set that contains the WSDL file, when the WSDL is associated with the node. If you set this property, then later update the project dependencies to remove this message set reference, a warning is issued. Either update the `Message set` property, or restore the reference to this message set project. The Response Message Parsing properties are disabled, and this property is cleared, when the node is configured to act in gateway mode.
Message type	No	No		The name of the response message. The node detects the message type automatically. You cannot set this property.
Message format	No	No		The name of the physical format of the response message. You cannot set this property.

The SOAPRequest node Parser Options properties are described in the following table.

Property	M	C	Default	Description
Parse timing	No	No	`On demand`	This property controls when a response message is parsed. Valid values are `On demand`, `Immediate`, and `Complete`. The default value, `On demand`, causes parsing of the message to be delayed. For a full description of this property, see "Parsing on demand" on page 4173.
Build tree using XML schema data types	No	No	Set	This property controls whether the syntax elements in the message tree have data types taken from the XML Schema. The `SOAP Parser Options` properties determine how the SOAP parser operates. The SOAP parser options are passed through to the XMLNSC parser. For more information, see "Manipulating messages in the XMLNSC domain" on page 2546. This property is cleared and disabled when the node is configured to act in gateway mode.
Retain mixed content	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters mixed text in a response message. If you select the check box, elements are created for mixed text. If you clear the check box, mixed text is ignored and no elements are created.
Retain comments	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters comments in a response message. If you select the check box, elements are created for comments. If you clear the check box, comments are ignored and no elements are created.
Retain processing instructions	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters processing instructions in a response message. If you select the check box, elements are created for processing instructions. If you clear the check box, processing instructions are ignored and no elements are created.

Property	M	C	Default	Description
Opaque elements	No	No	Blank	This property is used to specify a list of elements in the input message that are to be opaquely parsed. Opaque parsing is performed only if validation is not enabled (that is, if `Validate` is `None`); entries that are specified in `Opaque Elements` are ignored if validation is enabled.

The SOAPRequest node Validation properties are described in the following table. These properties apply only to the response message; the request message is not validated.

Property	M	C	Default	Description	**mqsiapplybaroverride command property**
Validate	No	Yes	`Content and value`	This property controls whether the SOAP parser validates the body of each response message against XML Schema generated from the message. Valid values are `None`, `Content and value`, and `Content`. By default, validation is enabled. The SOAP parser starts the XMLNSC parser to validate the XML body of the SOAP Web service. If a message is propagated to the Failure terminal of the node, it is not validated. For more details, see "Validating messages" on page 1478 and "Validation properties" on page 4169. Validation properties are disabled, and the `Validate` property is set to `None`, when the node is configured to act in gateway mode.	validateMaster
Failure action	No	No	`Exception`	This property controls what happens if validation fails. You can set this property only if you set `Validate` to `Content` or `Content and value`. Valid values are `User trace`, `Local error log`, `Exception`, and `Exception list`. The Validation properties are disabled when the node is configured to act in gateway mode.	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Working with WrittenDestination data:
After the request has been made, the WrittenDestination folder in the local environment is updated with the WS-Addressing (if in use), compression details (if in use) and transport details. A WrittenDestination for a SOAPRequest node has

the following format, with WS-Addressing and Compression present only if it is used, where *jmsDestination* is the name of your JMS destination:

```
WrittenDestination = (
   SOAP  = (
      Request = (
         WSA = (
            To = 'URI'
            MessageID = 'id'
            Action = 'doAllTheStuff'
         )
         Transport = (
            HTTP = (
               WebServiceURL = 'http://server:8080/thing'
               Compression   = (
                  OriginalSize   = 775
                  CompressedSize = 411
            )
            JMS = (
               Destination = jmsDestination
            )
         )
      )
   )
)
```

Local environment overrides:
You can dynamically override set values in the local environment in the same way as setting values in other elements of a message. For a full list of values you can override in the local environment, see Local environment overrides.

HTTPRequest headers:
To control the contents of a HTTPRequest header that is included in a message, you must include a Compute node to add a HTTPRequest header to the input message before the HTTPRequest node in the message flow.

If a SOAPAction is set to an empty string in the HTTPInput header, it is overridden, unless the action is set explicitly in the HTTPRequest header, for example:

```
SET OutputRoot.HTTPRequestHeader.SOAPAction = InputRoot.HTTPInputHeader.SOAPAction
```

Related concepts:

"Message flow aggregation" on page 2718
Aggregation is the generation and fan-out of related requests that are derived from a single input message, and the fan-in of the corresponding replies to produce a single aggregated reply message.

"What is SOAP?" on page 1604
SOAP is an XML message format used in Web service interactions. SOAP messages are typically sent over HTTP or JMS, but other transport protocols can be used. The use of SOAP in a specific Web service is described by a WSDL definition.

"SOAP tree overview" on page 1611
This tree format allows you to access the key parts of the SOAP message in a convenient way.

"EndpointLookup node output" on page 1894
Use an EndpointLookup node to retrieve the endpoint addresses for WSDL service definitions held in WebSphere Service Registry and Repository (WSRR).

"SOAP nodes" on page 1609
The SOAP nodes act as points in the flow where Web service processing is configured and applied. Properties on the SOAP nodes control the processing carried out and can be configured by supplying a WSDL definition, or by

manually configuring properties, or both.

"WSDL URI formats for JMS" on page 1668
You must use WSDL to configure SOAP nodes. When using WSDL with a JMS transport, different URI formats can exist in the `address` element in the WSDL, which affect how properties are parsed and applied to the configured nodes.

"Using compression with HTTP and SOAP nodes" on page 1597
You can configure HTTP and SOAP nodes to use HTTP compression and decompression when sending and receiving messages.

Related tasks:

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Configuring aggregation flows" on page 2721
Use aggregation message flows to generate and fan-out a number of related requests, fan-in the corresponding replies, and compile those replies into a single aggregated reply message, by using the AggregateControl, AggregateRequest, and AggregateReply nodes.

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

"Configuring authentication with HTTP basic authentication" on page 451
Use a security profile to configure HTTP basic authentication in the HTTPRequest or SOAPInput nodes.

"Importing from WSDL" on page 2946
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to create a new message definition from WSDL.

"Importing WSDL definitions from the command line" on page 2948
WSDL definitions can be imported using the (**`mqsicreatemsgdefsfromwsdl`**) command.

"Configuring JMS temporary dynamic queues for the WebSphere MQ JMS provider" on page 4848
Configure JMS temporary dynamic queues for the WebSphere MQ JMS provider, so that you can use them with SOAPRequest nodes, by using WebSphere MQ Explorer.

Related reference:

"SOAPAsyncResponse node" on page 4777
Use the SOAPAsyncResponse node in conjunction with the SOAPAsyncRequest node to construct a pair of message flows that call a Web service asynchronously.

"SOAPAsyncRequest node" on page 4750
Use the SOAPAsyncRequest node with the SOAPAsyncResponse node to construct a pair of message flows that call a Web service asynchronously.

"SOAPInput node" on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

"SOAPExtract node" on page 4790
Use the SOAPExtract node to remove SOAP envelopes, allowing just the body of a SOAP message to be processed. It can also route a SOAP message based on its operation name. Both functions are optional; they are contained in one node because they are often used together.

"SOAPReply node" on page 4819
Use the SOAPReply node to send SOAP messages from the broker to the originating client in response to a message received by a SOAPInput node.

"HTTPRequest node" on page 4488
Use the HTTPRequest node to interact with a web service.

"EndpointLookup node" on page 4407
Use the EndpointLookup node to retrieve service endpoint information held in the WebSphere Service Registry and Repository (WSRR). The entities that match the specified search criteria are stored in the local environment. Depending on the node configuration, the Web service URL destination used by the SOAP and HTTP request nodes can also be set. The input message is not modified.

"WS-Addressing with the SOAPRequest node" on page 1653
Various options are available when you use WS-Addressing with the SOAPRequest node.

"Configuring for coordinated JMS transactions" on page 4544
Configure your message flow to receive or output messages under coordinated transactions.

Configuring for coordinated JMS transactions:

Configure your message flow to receive or output messages under coordinated transactions.

When you include a node using JMS transport in a message flow, such as the JMSInput or SOAPInput node when using JMS transport, the value that you set for `Transaction mode` defines whether messages are received under sync point.

- If you set this property to `Global`, the message is received under external sync point coordination; that is, within a WebSphere MQ unit of work. Any messages that are sent later, by an output node in the same instance of the message flow, are put under sync point, unless the output node overrides this setting explicitly.
- If you set this property to `Local`, the message is received under the local sync point control of the node. Any messages that are sent later, by an output node in the flow, are not put under local sync point, unless an individual output node specifies that the message must be put under local sync point.
- If you set this property to `None`, the message is not received under sync point. Any messages that are sent later, by an output node in the flow, are not put under sync point, unless an individual output node specifies that the message must be put under sync point.

To receive messages under external sync point, you must take additional configuration steps, which need be applied only the first time that a specific node using JMS transport is deployed to the broker for a particular JMS provider.

- On distributed systems, the external sync point coordinator for the broker is WebSphere MQ. Before you deploy a message flow in which the `Transaction mode` property is set to `Global` or `Yes`, and is intended to use globally coordinated transactions, modify the queue manager `.ini` file to include extra definitions for each JMS provider resource manager that participates in globally coordinated transactions.
  - Windows On Windows:
    1. Start WebSphere MQ Explorer.
    2. Right-click the queue manager name in the left pane and click **Properties**.
    3. Click **XA resource managers** in the left pane.
    4. Click **Add...** .

5. Set the options as follows:
   - Set `Name` to any value.
   - On Windows on x86 systems, set the `SwitchFile` property to *`install_dir`*`\bin\JMSSwitch.dll`. On Windows on x86-64 systems, set the `SwitchFile` property to `JMSSwitch.dll`.
   - Set the `XAOpenString` property to a string value as follows: *`Initial Context`*`,`*`location JNDI`*`,`*`Optional_parms`*.
   - Set the `ThreadOfControl` property to `Thread`.
6. On Windows on x86-64 systems only, copy the switch file *JMSSwitch32.dll* to the `\exits` subdirectory in the WebSphere MQ installation directory, and rename it to `JMSSwitch.dll`. Copy the switch file `JMSSwitch.dll` to the `\exits64` subdirectory in the WebSphere MQ installation directory.

For more information, see the *System Administration Guide* section of the WebSphere MQ Version 7 Information Center online.

– Linux UNIX On Linux and UNIX systems, add a stanza to the queue manager `.ini` file for each JMS provider.

For example:

```
XAResourceManager:
Name=Jms_Provider_Name
SwitchFile=/install_dir/bin/ JMSSwitch.so
XAOpenString=Initial Context,location JNDI,Optional_parms
ThreadOfControl=THREAD
```

Where:

Name is an installation defined name that identifies a JMS provider resource manager.

SwitchFile
is the file system path to the JMSSwitch library that is supplied in the `bin` directory of the broker.

XAOpenString can have the following values:

– *Initial Context* is the value that is set in the JMSInput node property `Initial context factory`.

– *location JNDI* is the value that is set in the JMSInput node property `Location JNDI bindings`. This value must include a supported URL prefix that has a URL handler that is available on the class path.

The following parameters are optional:

– `LDAP Principal` matches the value that is set for the broker by using the **`mqsicreatebroker`** or **`mqsichangebroker`** commands.

– `LDAP Credentials` matches the value that is set for the broker by using the **`mqsicreatebroker`** or **`mqsichangebroker`** commands.

– `Recovery Connection Factory Name` is the JNDI administered connection factory that is defined in the bindings file. If a value is not specified, you must add a default value for `recoverXAQCF` to the bindings file. In either case, the Recovery Connection Factory must be defined as an XA Queue Connection Factory for the JMS provider that is associated with the `Initial context factory`.

The optional parameters are comma-separated and are positional. Therefore, any parameters that are missing must be represented by a comma. For example:

```
com.sun.jndi.fscontext.RefFSContextFactory,file:/C:/webservices/SOAP/JMS/JNDIXA,,,QCF
```

1. Update the Java CLASSPATH environment variable for the queue manager of the broker to include a reference to `xarecovery.jar`; for example:

install_dir/classes/xarecovery.jar

2. Update the Java PATH environment variable for the queue manager of the broker to point to the bin directory in which the SwitchFile is located; for example:

   install_dir/bin

Finally, ensure that you have taken the following configuration steps:

- In the message flow, ensure that the coordinated property is enabled by using the WebSphere Message Broker Archive editor.
- Ensure that each node that must be part of the XA transaction is set to the global transaction mode.
- Ensure that the service ID that is used for the broker and the queue manager is the same user ID.
- Ensure that the JNDI connection factory objects that the JMS nodes use for a global transaction are configured to be of the type MQXAConnectionFactory, MQXAQueueConnectionFactory, or MQXATopicConnectionFactory.
  - If you create the bindings using WebSphere Message Broker Explorer, ensure the `Support XA Transactions` option is checked when you define your connection factory.
  - If you create the bindings using JMSAdmin, use the command **DEF XAQCF** or **DEF XATCF**, instead of **DEF QCF** or **DEF TCF**, when you define your connection factory.

For more information, see the *System Administration Guide* section of the WebSphere MQ Version 7 Information Center online.

- z/OS On z/OS, the external sync point manager is Resource Recovery Services (RRS). The only JMS provider that is supported on z/OS is WebSphere MQ JMS. The only transport option that is supported for WebSphere MQ JMS on z/OS is the bind option.

  Sync point control for the JMS provider is managed with RRS sync point coordination of the queue manager of the broker. You do not have to modify the `.ini` file.

Configuring JMS temporary dynamic queues for the WebSphere MQ JMS provider:

Configure JMS temporary dynamic queues for the WebSphere MQ JMS provider, so that you can use them with SOAPRequest nodes, by using WebSphere MQ Explorer.

About this task

Complete this task before using JMS temporary dynamic queues with the WebSphere MQ JMS provider in your message flow applications.

If the model queue SYSTEM.JMS.MODEL.QUEUE already exists on your system, start at step 6 on page 4849 and enter `SYSTEM.JMS.MODEL.QUEUE` when you are asked to enter your JMS temporary dynamic model queue name.

Procedure

To set up JMS temporary dynamic queues:

1. In the Navigator view of WebSphere MQ Explorer, expand **Queue Managers** > ***QueueManager***, where *QueueManager* is the name of your queue manager, click **Queues**.

2. Right-click **Queues**, then click **New** > **Model Queue**. The New Model Queue wizard opens.
3. Enter a name for your JMS temporary dynamic model queue, for example `SYSTEM.JMS.MODEL.QUEUE`, then select `SYSTEM.DEFAULT.MODEL.QUEUE` as the object from which to copy attributes, and click **Next**.
4. Under **Change Properties**, click **Extended** and change the following properties:
   a. In the **Shareability** field, select `Shareable`.
   b. In the **Default input open options** field, select `Input shared`.
5. To create your model queue, click **Finish**.
6. In the Navigator view of WebSphere MQ Explorer, expand **JMS Administered Objects** > ***JNDIRepository***, where *JNDIRepository* is the name of your JNDI repository, click **Connection Factories**. A list of Connection Factories is displayed in the Content view.
7. Right-click the Connection Factory that you want to use, then click **Properties**.
8. In the Properties window, click **Temporary queues**.
9. In the **Temporary model queue** field, enter the name of the JMS temporary dynamic model queue that you created previously, then click **Apply**.
10. To save your changes, click **OK**.

Results

You have created a model queue, and configured your queue manager to create JMS temporary dynamic queues you can use with the WebSphere MQ JMS provider.

What to do next

You can now use JMS temporary dynamic queues in your message flow applications with the WebSphere MQ JMS provider.

Related concepts:

"Processing JMS messages" on page 1679
JMS is the standard J2EE messaging API for building enterprise messaging applications. WebSphere Message Broker provides built-in input and output nodes for its supported protocols.

"What is SOAP?" on page 1604
SOAP is an XML message format used in Web service interactions. SOAP messages are typically sent over HTTP or JMS, but other transport protocols can be used. The use of SOAP in a specific Web service is described by a WSDL definition.

"SOAP over JMS" on page 6698
SOAP over Java Message Service 1.0 is a specification that describes how SOAP can bind to a messaging system that supports the Java Message Service (JMS).

"The SOAP body" on page 1608
The SOAP body (the <Body> element) is a mandatory sub-element of the SOAP envelope, which contains information intended for the ultimate recipient of the message.

Related reference:

"SOAPRequest node" on page 4828
Use the SOAPRequest node to send a SOAP request to the remote Web service.

"WebSphere Broker JMS Transport" on page 1681
The WebSphere Broker JMS Transport is a service that connects applications that send and receive messages that conform to the Java Message Service (JMS) standard.

Related information:

WebSphere MQ Version 7 Information Center online

Local environment overrides for the SOAPRequest node:

You can dynamically override values in the local environment in the same way as setting values in other elements of a message.

Other local environment overrides are available for WS-Addressing. See "WS-Addressing with the SOAPRequest node" on page 1653.

You can set the following properties in the SOAPRequest node under `LocalEnvironment.Destination.SOAP.Request`. These local environment overrides also apply to the SOAPAsyncRequest node.

Setting	Description
Operation	Overrides the `Operation` property on the node. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Operation = 'myOperation';`

TransportType	Overrides the `Transport` property on the node to switch transport. For example, if the node is configured to use the JMS transport, use the following to switch to HTTP transport:
	`SET OutputLocalEnvironment.Destination.SOAP.Request.TransportType = 'http';`
	
	To switch to JMS transport:
	`SET OutputLocalEnvironment.Destination.SOAP.Request.TransportType = 'jms';`
	
	This overrides the request and response transport for this message.

UserContext	You can store BLOB context data in the following location in the local environment. The SOAPAsyncResponse node can later retrieve this data. `SET OutputLocalEnvironment.Destination.SOAP.Request.UserContext = x'aabbccddeeff1122334455667788990';` Data stored in the UserContext must be in BLOB format. This field is included in the message bitstream, and therefore increases the message size. This setting applies only to the SOAPAsyncRequest node and is ignored by a SOAPRequest node.

LocalEnvironment overrides for HTTP transport

You can set the following HTTP properties in the SOAPRequest node under `LocalEnvironment.Destination.SOAP.Request.Transport.HTTP`. These properties apply only when using HTTP transport. You can switch between HTTP and JMS transport using the `TransportType` override, or the WS-Addressing `To` field; see "WS-Addressing information in the local environment" on page 1656.

Setting	Description
Compression	Overrides the `Use compression` property on the node. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.HTTP.Compression = 'gzip';`

Setting	Description
HTTPVersion	Overrides the HTTPVersion. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.HTTP.HTTPVersion = 'HTTP/1.1';`
Method	Overrides the Method. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.HTTP.Method = 'GET';`

ProxyConnectHeaders	Specifies additional headers that are used if the outbound request is an SSL connection through a proxy. These additional headers are sent with the initial CONNECT request to the proxy. For example, you can send proxy authentication information to a proxy server when you are using SSL. You can send multiple headers but each one must be separated by a carriage return and a line feed (ASCII 0x0D 0x0A), in accordance with RFC2616; for example:
	`DECLARE CRLF CHAR CAST(X'0D0A' AS CHAR CCSID 1208);`
	`SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.HTTP.ProxyConnectHeaders = 'Proxy-Authorization: Basic Zm5lcmJsZTpwYXNzd29yZA==' || CRLF ||`
	`'Proxy-Connection: Keep-Alive' || CRLF;`
	
	This setting is used only if the request is an SSL request through a proxy server. To send proxy authentication information for a non-SSL request, specify the individual headers in the HTTPRequestHeader folder, as shown in the following example:
	`SET OutputRoot.HTTPRequestHeader."Proxy-Authorization" = 'Basic Zm5lcmJsZTpwYXNzd29yZA==';`
	`SET OutputRoot.HTTPRequestHeader."Proxy-Connection" = 'Keep-Alive';`

ProxyURL	Overrides the HTTP(S) proxy location property on the node. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.HTTP.ProxyURL = 'my.proxy';`
RequestURI	Overrides the RequestURI, which is the path after the URL and port. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.HTTP.RequestURI = '/abc/def?x=y&g=h';`
SSLCiphers	Overrides the Allowed SSL Ciphers (if using SSL) property on the node. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.HTTP.SSLCiphers = 'SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA';`
SSLProtocol	Overrides the SSLProtocol property on the node. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.HTTP.SSLProtocol = 'TLS';` Valid values are SSL, SSLv3, and TLS.
Timeout	Overrides the Request timeout (in seconds) property on the node. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.HTTP.Timeout = 42;`
TimeoutMillis	Overrides the Request timeout (in seconds) property on the node. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.HTTP.TimeoutMillis = 5000;` This property defines the timeout in milliseconds. The value of TimeoutMillis overrides the value for Timeout if both values are set.
WebServiceURL	Overrides the Web service URL property on the node. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.HTTP.WebServiceURL = 'http://ibm.com/abc/';`

LocalEnvironment overrides for JMS transport

You can set the following JMS properties in the SOAPRequest node under `LocalEnvironment.Destination.SOAP.Request.Transport.JMS`. These properties apply only when using JMS transport. You can switch between HTTP and JMS transport using the `TransportType` override, or the WS-Addressing `To` field; see "WS-Addressing information in the local environment" on page 1656.

Some JMS local environment overrides for the SOAPRequest node have equivalent properties in the JMSTransport header. If you specify a local environment override, it takes precedence over any equivalent property set in the JMSTransport header.

Setting	Description
BindingsLocation	Overrides the JNDI URL bindings location property on the node. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.JMS.JNDI.BindingsLocation = 'file:/C:/mqsi6/Webservices/SOAP/JMS/JNDI';`
ConnectionFactoryName	Overrides the Connection factory name property on the node. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.JMS.JNDI.ConnectionFactoryName = 'QCF';`

ContextParameters	Specify JNDI context parameters in addition to the JNDI context parameters defined on the node. You can define specific JNDI context parameters, for example:
	`SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.JMS.JNDI.ContextParameters.property1 = 'value1';`
	`SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.JMS.JNDI.ContextParameters.property2 = 'value2';`

CorrelationID	Sets the request message CorrelID. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.JMS.CorrelationID = 'myCorrelID';`
CorrelationPattern	Sets the request message correlation pattern. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.JMS.CorrelationPattern = 1;` The allowed values are the integers 1 and 2: • 1 indicates that the messages should be automatically correlated by the CorrelID if one is present. Otherwise, the messages are correlated by the message ID. This is the default behavior. • 2 forces the messages to be correlated by the message ID, even if a CorrelID is present.
DeliveryMode	Overrides the DeliveryMode property on the node. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.JMS.DeliveryMode = 'NON_PERSISTENT';` Allowed values for this property are PERSISTENT and NON_PERSISTENT. If the UriFormat is ibm, 1 and 0 are additional allowed values for DeliveryMode.
Destination	Overrides the Destination property on the node. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.JMS.Destination = 'REPLYTOQ2';`
DestinationURI	You can override multiple JMS properties at the same time in the local environment using the DestinationURI setting. Properties that you set in this way can be overridden by setting local environment overrides for individual JMS properties as shown in the following tables. `SET OutputLocalEnvironment.Destination.SOAP.Reply.Transport.JMS.DestinationURI = 'jms:jndi:INPUTQ1?jndiConnectionFactoryName=QCF& replyToName=REPLYTOQ2& jndiInitialContextFactory=com.sun.jndi.fscontext.RefFSContextFactory& jndiURL=file:/C:/Webservices/SOAP/JMS/JNDI& userParam1=value1& userParam2=value2& timeToLive=30000';` This local environment override can be set with either a W3C-style or IBM-style URI format. For more information, see “WSDL URI formats for JMS” on page 1668.
Expiration	Overrides the Expiration property on the node. This property is specified in milliseconds. For example, to set an expiration of 100 milliseconds: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.JMS.Expiration = '100';`

Setting	Description
InitialContextFactory	Overrides the Initial context factory property on the node. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.JMS.JNDI.InitialContextFactory = 'com.sun.jndi.fscontext.RefFSContextFactory';`
MessagePriority	Overrides the MessagePriority property on the node. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.JMS.MessagePriority = '7';`
MessageType	Overrides the MessageType property on the node. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.JMS.MessageType = 'text';` Allowed values for this property are textand bytes.
ProviderName	Overrides the JMS provider name property on the node. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.JMS.ProviderName = 'WebSphere MQ';`
ReplyToDestination	Overrides the ReplyToDestination property on the node. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.JMS.ReplyToDestination = 'REPLYTOQ3';`
TargetService	Overrides the TargetService property on the node. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.JMS.TargetService = 'testService';`
Timeout	Overrides the Timeout property on the node. This value is specified in seconds. For example, to set a timeout value of 30 seconds: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.JMS.Timeout = '30';`
TransactionMode	Overrides the Transaction mode property on the node. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.JMS.TransactionMode = 'Yes';` Allowable values for this property are Yes, No and ForceLocal. • No means that no transaction takes place, and is equivalent to None. • Yes means that a local transaction takes place if the flow's Coordinated Transaction is not selected, or a global transaction takes place if the flow's Coordinated Transaction property is selected. • ForceLocal means that a local transaction is always used, even if the flow's Coordinated Transaction property is selected.
UriFormat	Overrides the UriFormat property on the node. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.JMS.UriFormat = 'w3c';` Allowable values for this property are w3c and ibm.

UserProperties	Specify user context parameters in addition to the user context parameters defined on the node. You can define specific user context parameters, for example:
	`SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.JMS.JNDI.UserProperties.property1 = 'value1';`
	`SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.JMS.JNDI.UserProperties.property2 = 'value2';`

Setting	Description
OneWay	Instructs the node that the inbound message was a one-way message and that no reply message is needed. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Gateway.OneWay = True` If the message is sent over the HTTP transport, the node waits for an acknowledgment HTTP 202 response message from the remote server. If the message is sent over the JMS transport no response is expected. In this example the outbound message has no Reply-To queue, and the node does not wait for any response. For more information, see "One-way messages in Gateway mode" on page 1648.

TCPIPClientInput node

Use the TCPIPClientInput node to create a client connection to a raw TCP/IP socket, and to receive data over that connection.

This topic contains the following sections:

Purpose:

The TCPIPClientInput node opens connections to a remote server application that is listening on a TCP/IP port. The connections are not made directly by the node but are obtained from a connection pool managed by the WebSphere Message Broker execution group. The execution group uses the default TCPIPClient configurable service to determine which attributes are used for the socket connection. However, if the configurable service is set on the node, the configurable service is used for all the properties, including the host and port number.

You can configure the broker to use SSL for TCP/IP nodes; see "SSL and the TCP/IP nodes" on page 551.

When a connection is opened by the connection pool, it is sent to a TCPIPClientInput node (if the Open terminal of the node is connected). The input event is sent to only one TCPIPClientInput node on the connection.

The node requests a client connection that contains data ready for reading. Until such a connection is available, the node is paused, waiting for data (in a similar way to the MQInput node). Therefore, two criteria must be met before the node becomes available:

- A client connection has been made
- At least one byte of data is available to be processed

By default (as set in the configurable service), no client connections are made by the input node. The node relies on the creation of client connections by output or request nodes. In this mode of operation, an input node is never started until an output or request node starts an interaction.

You can change the mode on the configurable service to create a pool of client connections ready for processing. To use this function, `minimumConnections` must be set to a value larger than zero. The execution group then ensures that the specified

number of connections are always available by creating them at the start, and continuing to create the connections until the minimum value is reached.

This behavior is different from the TCPIPServerInput node, which does not attempt to make a minimum number of connections. For more information, see "TCPIPServerInput node" on page 4890.

The client node also has a maximum value, which limits how many connections it can create. More connections than the minimum value can exist as a result of output nodes creating connections.

When connections are available, the second criterion is met when there is at least one byte of data to be processed; otherwise, the connection closes. In either case, the connection is given to the node and the event is processed.

The first record of data is detected in accordance with properties on the node and then sent to the Out terminal. If an error occurs, including a timeout waiting for data or the closure of a connection while waiting for the full record, the data is sent to the Failure terminal. If the connection closes and there is no data, a message is sent to the Close terminal. Although the message has no data, the local environment does have details of the connection that closed.

For both data and close events, the following local environment is created:

Table 253. Location in local environment

Location in local environment	Description
$LocalEnvironment/TCPIP/Input/ConnectionDetails/Type	The client.
$LocalEnvironment/TCPIP/Input/ConnectionDetails/Hostname	The host name used to make a connection.
$LocalEnvironment/TCPIP/Input/ConnectionDetails/Port	The port number used to make a connection.
$LocalEnvironment/TCPIP/Input/ConnectionDetails/OpenTimestamp	The time stamp when the connection was first opened.
$LocalEnvironment/TCPIP/Input/ConnectionDetails/CloseTimestamp	The time stamp when the connection was closed (null if not yet closed).
$LocalEnvironment/TCPIP/Input/ConnectionDetails/SequenceNumber/InputRecord	The sequence number of the message received on this connection. The first record has a sequencing number of 1; the second record has a sequencing number of 2, and so on.
$LocalEnvironment/TCPIP/Input/ConnectionDetails/SequenceNumber/OutputRecord	The sequence number of the message sent on this connection. The first record has a sequencing number of 1; the second record has a sequencing number of 2, and so on.
$LocalEnvironment/TCPIP/Input/ConnectionDetails/Id	The ID of the socket being used. This ID is an internal identifier used by the message broker to uniquely identify a connection.
$LocalEnvironment/TCPIP/Input/ConnectionDetails/ReplyId	The reply ID that has been stored on this connection. The value can be any text string.

When the node has constructed the record from the connection stream it releases the connection back to the connection pool for use by other nodes. Properties on the Advanced tab show how that connection can be used by other nodes in the

future. By default, the Advanced properties mark the input stream on the TCP/IP connection as being reserved, which means that no other input node can use it, until the current use of the message flow is finished. Alternatively, you can reserve the connection until it is unreserved by another node, or not to reserve it at all and permit any other node (or thread in this node) to use the connection straight away. Similar options are available on the output stream but it is kept unreserved by default.

Another node can access a reserved stream only if the ID of the connection is known. This behavior allows all the nodes in a message flow to access the same connection using the same ID while stopping any other flow acquiring the connection.

The TCPIPClientInput node is contained in the **TCPIP** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Message structure

The TCPIPClientInput node handles messages in the following message domains:

- XMLNSC
- DataObject
- JSON
- BLOB
- MIME
- MRM
- JMSMap
- JMSStream
- XMLNS

Using the TCPIPClientInput node in a message flow:
Look at the following samples to see how to use the TCPIPClientInput node:

- TCPIP Client Nodes
- TCPIP Handshake

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Configuring the TCPIPClientInput node:
When you have put an instance of the TCPIPClientInput node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view.

All mandatory properties that do not have a default value defined are marked with an asterisk.

Configure the TCPIPClientInput node:

1. Optional: On the **Description** tab, enter a `Short description`, a `Long description`, or both. You can also rename the node on this tab.

2. On the **Basic** tab, set the properties that determine how the TCP/IP connection is controlled.
   - Use the Connection details property to specify either the host name and port number to be used, or the name of a configurable service. This property is mandatory. The following formats are supported:
     - Configurable service name. This value is used to look up the port and host name in configurable services. For example, TCPIPProfile1.
     - <Hostname>:<Port>. This value is the host name followed by the port number (separated by a colon). For example, tcpip.server.com:1111.
     - <Port>. This value is the port number. In this case, the host name is assumed to be localhost.
   - Use the Timeout waiting for a data record (seconds) property to specify how long the node listens on a connection for more data after the first byte of data has arrived. You can specify any length of time in seconds. The default is 60 seconds. When the specified time has been exceeded, all available data is sent to the Failure terminal.
3. On the **Advanced** tab, set the properties that determine how the data stream is controlled.
   - Use the Close connection property to specify when and how to close the connection.
     - Select No to leave the connection open. This value is the default.
     - Select After timeout to close the connection when a timeout occurs.
     - Select After data has been received to close the connection when the end of the record is found.
     - Select At end of flow to close the connection after the flow has been run.
   - Select Close input stream after a record has been received to close the input stream as soon as the data has been retrieved. When the connection input stream is reserved, no other node can use it without specifying the ID. This property is not selected by default.
   - Use the Input Stream Modification property to specify whether to reserve the input stream for use only by input and receive nodes that specify the connection ID, or to release the input stream at the end of the flow. These options are available only if you have not selected the Close input stream after a record has been received property.
     - Select Leave unchanged to leave the input stream as it was when it entered the node. This value is selected by default.
     - Select Reserve input stream (for use by future TCPIP input and receive nodes) to specify that this input stream can be used only by this node and by other receive nodes that request it by specifying the connection ID. When the connection input stream is reserved, no other nodes can use it without specifying the correct connection ID.
     - Select Reserve input stream (for use by future TCPIP input and receive nodes) then release at end of flow to specify that this input stream can be used only by this node and receive nodes that request it by specifying the correct connection ID. After the flow has been run, this input stream is returned to the pool and becomes available for use by any input or receive node.
   - Use the Output Stream Modification property to specify whether to release the output stream.
     - Select Leave unchanged to leave the output stream as it was when it entered the node. This value is selected by default.

- Select `Release output stream and reset ReplyID` to specify that this output stream is returned to the pool and is available for use by any output node. The ReplyID is passed in the LocalEnvironment when leaving this node, but is reset for the next record on this connection.

4. On the **Input Message Parsing** tab, set values for the properties that the node uses to determine how to parse the incoming message.

   If the incoming message has an MQRFH2 header, you do not have to set values for the Input Message Parsing properties because the values are derived from the `<mcd>` folder in the MQRFH2 header; for example:

   ```
   <mcd><Msd>MRM</Msd><Set>DHM4UO906S001</Set><Type>receiptmsg1</Type>
   <Fmt>XML</Fmt></mcd>
   ```

   If you set values that differ from those in the MQRFH2 header, the values in the MQRFH2 header take precedence.

   - In `Message domain`, select the name of the parser that you are using from the list. The default is `BLOB`. You can choose from the following options:
     - XMLNSC
     - DataObject
     - JSON
     - BLOB
     - MIME
     - MRM
     - JMSMap
     - JMSStream
     - XMLNS

     You can also specify a user-defined parser, if appropriate.
   - If you are using the MRM or IDOC parser, or the XMLNSC parser in validating mode, select the `Message set` that you want to use. This list is populated with available message sets when you select MRM, XMLNSC, or IDOC as the domain.
   - If you are using the MRM parser, select the correct message type from the list in `Message type`. This list is populated with available message types when you select the MRM parser.
   - If you are using the MRM or IDOC parser, select the correct message format from the list in `Message format`. This list is populated with available message formats when you select the MRM or IDOC parser.
   - Specify the message coded character set ID in `Message coded character set ID`.
   - Select the message encoding from the list in `Message encoding` or specify a numeric encoding value. The default is `Broker System Determined`. You can choose from the following options:
     - Little Endian, with IEEE Floating Point (546)
     - Big Endian, with IEEE Floating Point (273)
     - Big Endian, with S390 Floating Point (785)
     - Broker System Determined

     For more information about encoding, see "Data conversion" on page 1151.

5. On the **Parser Options** subtab:

   - `Parse timing` is, by default, set to `On Demand`, which causes parsing of the message to be delayed. To cause the message to be parsed immediately, see "Parsing on demand" on page 4173.
   - If you are using the XMLNSC parser, set values for the properties that determine how the XMLNSC parser operates. For more information, see "Manipulating messages in the XMLNSC domain" on page 2546.

6. Use the **Retry** tab to define how retry processing is performed when a flow fails. You can set the following retry processing:
   - `Retry mechanism` determines the action that occurs if the flow fails. The following choices can be set:
     - Select `Failure` for the node to report a failure without any retry attempts.
     - Select `Short retry` for the node to retry before reporting a failure if the condition persists. The number of times that it retries is specified in `Retry threshold`.
     - Select `Short retry and long retry` for the node to retry, first using the value in `Retry threshold` as the number of attempts to make. If the condition persists after the `Retry threshold` has been reached, the node uses the `Long retry interval` between attempts.
   - Specify the `Retry threshold`. The number of times the node retries the flow transaction if the `Retry mechanism` property is set to either `Short retry` or `Short retry and long retry`.
   - Specify the `Short retry interval`. The length of time, in seconds, to wait between short retry attempts.
   - Specify the `Long retry interval`. The length of time to wait between long retry attempts until a message is successful, the message flow is stopped, or the message flow is redeployed. The broker property **`MinLongRetryInterval`** defines the minimum value that the `Long retry interval` can take. If the value is lower than the minimum, the broker value is used.
7. Use the **Records and Elements** tab to specify how the data is interpreted as records:
   - Use the `Record detection` property to determine how the data is split into records, each of which generates a single message. Choose from the following options:
     - `End of stream` specifies that all of the data sent in the data stream is a single record.
     - `Fixed Length` specifies that each record is a fixed number of bytes in length. Each record must contain the number of bytes specified in the `Length` property, except possibly a shorter final record in the file.
     - Select `Delimited` if the records you are processing are separated, or terminated, by a DOS or UNIX line end or by a sequence of user-defined delimiter bytes. Specify the delimiter and delimiter type in the `Delimiter` and `Delimiter type` properties.
     - Select `Parsed Record Sequence` if the data contains a sequence of one or more records that are serially recognized by the parser that is specified in `Message domain`. The node propagates each recognized record as a separate message. If you select this `Record detection` option, the parser specified in `Message domain` must be either XMLNSC or MRM (either CWF or TDS physical format).
   - If you set `Record detection` to `Fixed Length`, use `Length` to specify the required length of the output record. This value must be in the range 1 byte through 100 MB. The default is `80` bytes.

     If you set `Record detection` to `Connection closed`, `Fixed Length`, or `Delimited`, a limit of 100 MB applies to the length of the records. If you set `Record detection` to `Parsed Record Sequence`, the TCPIPClientInput node does not determine or limit the length of a record. Nodes that are downstream in the message flow might try to determine the record length, or process a long record. If you intend to process large records in this way, ensure that your broker has sufficient memory. You might have to apply

flow techniques described in the Large Messaging sample to best use the available memory; see Large Messaging.

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

- If you set `Record detection` to `Delimited`, use `Delimiter` to specify the delimiter to be used. Choose from:
  - `DOS or UNIX Line End`, which, on UNIX systems, specifies the line feed character (<LF>, X'0A'), and, on Windows systems, specifies a carriage return character followed by a line feed character (<CR><LF>, X'0D0A'). The node treats both of these strings as delimiters, irrespective of the system on which the broker is running. If they are both displayed in the same record, the node recognizes both strings as delimiters. The node does not recognize X'15', which, on z/OS systems, is the 'newline' byte; specify a value of `Custom Delimiter` in this property and a value of 15 in the `Custom delimiter` property if your input file is coded using EBCDIC new lines, such as EBCDIC files from a z/OS system.
  - `Custom Delimiter`, which permits a sequence of bytes to be specified in `Custom delimiter`.
- In `Custom delimiter`, specify the delimiter byte or bytes to be used when `Custom delimiter` is set in the `Delimiter` property. Specify this value as an even-numbered string of hexadecimal digits. The default is X'0A' and the maximum length of the string is 16 bytes (represented by 32 hexadecimal digits).
- If you specify `Delimited` in `Record detection`, use `Delimiter type` to specify the type of delimiter. Permitted values are:
  - `Infix`. If you select this value, each delimiter separates records. If the data ends with a delimiter, the (zero length) data following the final delimiter is still propagated, although it contains no data.
  - `Postfix`. If you specify this value, each delimiter terminates records. If the data ends with a delimiter, no empty record is propagated after the delimiter. If the data does not end with a delimiter, it is processed as if a delimiter follows the final bytes of the data. `Postfix` is the default value.
- The TCPIPClientInput node considers each occurrence of the delimiter in the input as either separating (infix) or terminating (postfix) each record. If the data begins with a delimiter, the node treats the (zero length) contents preceding that delimiter as a record and propagates an empty record to the flow. The delimiter is never included in the propagated message.

8. Use the **Validation** tab to provide validation based on the message set for predefined messages. For more information about validation, see "Validating messages" on page 1478. For information about how to complete this tab, see "Validation tab properties" on page 4169.
9. On the **Transactions** tab, set the transaction mode. Although TCP/IP operations are non-transactional, the transaction mode on this input node determines whether the rest of the nodes in the flow are to be run under point of consistency. Select `Yes` if you want the flow updates to be treated transactionally (if possible) or `No` if you do not. The default for this property is `No`.
10. Optional: On the **Instances** tab, set values for the properties that show the additional instances (threads) that are available for a node. For more details, see "Configurable message flow properties" on page 4020.

Terminals and properties:
The terminals of the TCPIPClientInput node are described in the following table.

Terminal	Description
Open	The output terminal to which a message is routed when a connection is first opened. Use the Open terminal if processing is required when a connection is opened rather than when data first arrives. The connection associated with the message is reserved from the general connection pool until propagation to the Open terminal has finished. However, the connection can be accessed using the connectionId specified in the local environment. Each connection that is created is sent to the Open terminal, including any connections that are created mid-flow by a TCPIPClientReceive node or TCPIPClientOutput node. If the Open terminal is not attached, open events are automatically made available in the connection pool.
Failure	The output terminal to which the message is routed if an error occurs. This value includes failures caused by retry processing. Even if the Validation property is set, messages propagated to this terminal are not validated.
Out	The output terminal to which the message is routed if it is successfully retrieved from an external resource. If no errors occur within the input node, a message received from an external resource is always sent to the Out terminal first.
Close	The output terminal to which the message is routed if the connection closes.
Catch	The output terminal to which the message is routed if an exception is issued downstream and caught by this node. Exceptions are caught only if this terminal is attached.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file for deployment).

The Description properties of the TCPIPClientInput node are described in the following table.

Property	M	C	Default	Description
Node name	No	No	TCPIPClientInput	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The Basic properties of the TCPIPClientInput node are described in the following table.

Property	M	C	Default	Description	`mqsiapplybaroverride` command property
Connection details	Yes	Yes		A string containing either the host name and port number to be used, or the name of a configurable service.	connectionDetails
Timeout waiting for a data record (seconds)	Yes	Yes	`60`	Specifies how long the node listens on a connection for more data after the first byte of data has arrived. You can specify any length of time in seconds.	timeoutWaitingForData

The Advanced properties of the TCPIPClientInput node are described in the following table.

Property	M	C	Default	Description
Close connection	Yes	No	`No`	Controls when the connection is closed, or if it remains open. Valid options are: • `No` • `After Timeout` • `After Data has been Received` • `At End of Flow`
Close input stream after a record has been received	Yes	No	Cleared	Specifies whether to close the input stream as soon as the data has been retrieved. When the connection input stream is reserved, no other node can use it without knowing the ID. This property is not selected by default.
Input Stream Modification	No	No	`Leave unchanged`	Specifies whether to reserve the input stream for use only by input and receive nodes that specify the connection ID, or to release it at the end of the flow. Valid options are: • `Leave unchanged` • `Reserve input stream (for use by future TCPIP nodes)` • `Reserve input stream (for use by future TCPIP nodes) then release at end of flow` When the connection input stream is reserved, no other nodes can use it without specifying the correct connection ID. If the input stream is released at the end of the flow, it is returned to the pool and becomes available for use by any input or receive node.
Output Stream Modification	No	No	`Leave unchanged`	Specifies whether this output stream is released and returned to the pool for use by any output node. Valid options are: • `Leave unchanged` • `Release output stream and reset ReplyID` If you select `Release output stream and reset ReplyID`, the ReplyID is passed in the LocalEnvironment when leaving this node, but is reset for the next record on this connection.

The TCPIPClientInput node Input Message Parsing properties are described in the following table.

Property	M	C	Default	Description	`mqsiapplybaroverride` command property
Message domain	No	No		The domain that is used to parse the incoming message.	
Message set	No	No		The name or identifier of the message set in which the incoming message is defined. If you set this property, then later update the project dependencies to remove this message set reference, a warning is issued. Either update the `Message Set` property, or restore the reference to this message set project.	
Message type	No	No		The name of the incoming message.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Message format	No	No		The name of the physical format of the incoming message.	
Message coded character set ID	Yes	No	`Broker System Default`	The ID of the coded character set used to interpret the data being read.	messageCodedCharSetIdProperty
Message encoding	Yes	No	`Broker System Determined`	The encoding scheme for numbers and large characters used to interpret the data being read. Valid values are `Broker System Determined` or a numeric encoding value. For more information about encoding, see "Data conversion" on page 1151.	messageEncodingProperty

The Parser Options properties of the TCPIPClientInput node are described in the following table.

Property	M	C	Default	Description
Parse timing	No	No	`On Demand`	This property controls when an input message is parsed. Valid values are: • `On Demand` • `Immediate` • `Complete` For a full description of this property, see "Parsing on demand" on page 4173.
Build tree using XML schema data types	No	No	Cleared	This property controls whether the syntax elements in the message tree have data types taken from the XML schema.
Use XMLNSC compact parser for XMLNS domain	No	No	Cleared	This property controls whether the XMLNSC Compact Parser is used for messages in the XMLNS Domain. If you set this property, the message data is displayed under XMLNSC in nodes that are connected to the output terminal when the input MQRFH2 header or Input Message Parsing property, `Message Domain`, is XMLNS.
Retain mixed content	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters mixed text in an input message. If you select the check box, elements are created for mixed text. If you clear the check box, mixed text is ignored and no elements are created.
Retain comments	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters comments in an input message. If you select the check box, elements are created for comments. If you clear the check box, comments are ignored and no elements are created.
Retain processing instructions	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters processing instructions in an input message. If you select the check box, elements are created for processing instructions. If you clear the check box, processing instructions are ignored and no elements are created.
Opaque elements	No	No	Blank	This property is used to specify a list of elements in the input message that are to be opaquely parsed by the XMLNSC parser.

The Records and Elements properties of the TCPIPClientInput node are described in the following table:

Property	M	C	Default	Description
Record detection	Yes	No	`End of Stream`	The mechanism used to identify records in the input data. Valid options are: • `End of Stream` • `Fixed Length` • `Delimited` • `Parsed Record Sequence`
Length (bytes)	Yes	No	`0`	The length of each record, in bytes, when `Fixed Length` record detection is selected.
Delimiter	Yes	No	`DOS or UNIX Line End`	The type of delimiter bytes that separate, or end, each record when `Delimited` record detection is selected. Valid options are: • `DOS or UNIX Line End` • `Custom Delimiter (Hexadecimal)`
Custom delimiter (hexadecimal)	No	No		The delimiter bytes, expressed in hexadecimal, when `Delimited` record detection and `Custom Delimiter` are selected. This property is mandatory only if the `Delimiter` property is set to `Custom Delimiter (Hexadecimal)`.
Delimiter type	Yes	No	`Postfix`	The location of the delimiter when `Delimited` record detection and `Custom Delimiter (Hexadecimal)` are selected. Valid options are: • `Infix` • `Postfix` This property is ignored unless the `Delimiter` property is set to `Custom Delimiter (Hexadecimal)`.

The Retry properties of the TCPIPClientInput node are described in the following table:

Property	M	C	Default	Description	**`mqsiapplybaroverride` command property**
Retry mechanism	Yes	No	`Failure`	How the node handles a flow failure. Valid options are: • `Failure` • `Short Retry` • `Short and Long Retry`	
Retry threshold	Yes	Yes	`0`	The number of times to retry the flow transaction when `Retry mechanism` is `Short retry`.	retryThreshold
Short retry interval	No	Yes	`0`	The interval, in seconds, between each retry if `Retry threshold` is not zero.	shortRetryInterval
Long retry interval	No	Yes	`300`	The interval between retries if `Retry mechanism` is `Short and long retry` and the retry threshold has been exhausted.	longRetryInterval

The Validation properties of the TCPIPClientInput node are described in the following table.

For a full description of these properties, see “Validation properties” on page 4169.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Validate	No	Yes	None	This property controls whether validation takes place. Valid values are • None • Content and Value • Content	validateMaster
Failure action	No	No	Exception	This property controls what happens if validation fails. Valid values are: • User Trace • Local Error Log • Exception • Exception List	

The Transactions properties of the TCPIPClientInput node are described in the following table:

Property	M	C	Default	Description
Transaction mode	No	Yes	No	The transaction mode on this input node determines whether the rest of the nodes in the flow are run under point of consistency. Valid options are: • No • Yes

The Instances properties of the TCPIPClientInput node are described in the following table. For a full description of these properties, see "Configurable message flow properties" on page 4020.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Additional instances pool	No	Yes	Use Pool Associated with Message Flow	The pool from which additional instances are obtained. • If you select Use Pool Associated with Message Flow, additional instances are obtained from the message flow pool. • If you select Use Pool Associated with Node, additional instances are allocated from the additional instances of the node based on the number specified in the Additional instances property.	componentLevel
Additional instances	No	Yes	0	The number of additional instances that the node can start if the Additional instances pool property is set to Use Pool Associated with Node.	additionalInstances

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"WebSphere Broker TCP/IP Transport" on page 1735
The WebSphere Broker TCP/IP Transport is a service that connects applications that use raw TCP/IP sockets for transferring data.

"TCP/IP nodes" on page 1738
WebSphere Message Broker implements access to the TCP/IP input and output streams through a series of nodes.

"Scenarios for WebSphere Message Broker and TCP/IP" on page 1746
Two example scenarios show how you might use TCP/IP and WebSphere Message Broker as part of a business solution.

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

"Parsers" on page 1072
A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from the internal message tree representation.

"Specifying namespaces in the Message Type property" on page 1208
When using the MRM domain, the *Message Type* property is used to specify the name of the message.

Related tasks:

"Working with TCP/IP" on page 1750
You can use WebSphere Message Broker TCPIP nodes and TCP/IP configurable services to perform various tasks.

"Validating messages" on page 1478
The broker provides validation based on the message model for predefined messages.

"Manipulating messages in the XMLNSC domain" on page 2546
If you are writing ESQL to process messages in the XMLNSC domain, it is helpful to learn about the structure of the message tree that the XMLNSC parser builds.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

Related reference:

"Parsing on demand" on page 4173
On-demand parsing, referred to as partial parsing, is used to parse an input message bit stream only as far as is necessary to satisfy the current reference. The parsers that can perform partial parsing of input messages are the JSON, MRM, XML, XMLNS, and XMLNSC parsers.

"Configurable message flow properties" on page 4020
When you add a message flow to a broker archive (BAR) file in preparation for deploying it to a broker, you can set additional properties that influence its run time operation. These properties are available for review and update when you select the **Manage and Configure** tab for the broker archive file.

"Validation properties" on page 4169
You can control validation by setting properties on the Validate and Parser Options tabs for the nodes that are listed in the following table.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"TCPIPClientOutput node"
Use the TCPIPClientOutput node to create a client connection to a raw TCP/IP socket, and to send data over that connection to an external application.

"TCPIPServerOutput node" on page 4903
Use the TCPIPServerOutput node to create a server connection to a raw TCP/IP socket, and to send data over the connection to an external application.

"TCPIPClientReceive node" on page 4877
Use the TCPIPClientReceive node to receive data over a client TCP/IP connection.

"TCPIPServerReceive node" on page 4913
Use the TCPIPServerReceive node to receive data over a server TCP/IP connection.

"TCPIPServerInput node" on page 4890
Use the TCPIPServerInput node to create a server connection to a raw TCPIP socket, and to receive data over that connection.

TCPIPClientOutput node

Use the TCPIPClientOutput node to create a client connection to a raw TCP/IP socket, and to send data over that connection to an external application.

This topic contains the following sections:
- "Purpose"
- "Using the TCPIPClientOutput node in a message flow" on page 4869
- "Configuring the TCPIPClientOutput node" on page 4869
- "Terminals and properties" on page 4873

Purpose:
The TCPIPClientOutput node opens connections to a remote server application that is listening on a TCP/IP port. The connections are not made directly by the node but are obtained from a connection pool managed by the WebSphere Message Broker execution group. The execution group uses the default TCPIPClient configurable service to determine which attributes are used for the socket

connection. However, if the configurable service is set on the node, the configurable service is used for all the properties, including the host and port number.

You can configure the broker to use SSL for TCP/IP nodes; see "SSL and the TCP/IP nodes" on page 551.

The TCPIPClient configurable service is used to create a pool of client connections ready for processing. To use this function, the `minimumConnections` property must be set to a value larger than zero. The execution group ensures that the specified number of connections are always available by creating them at the start, and continuing to create the connections until the minimum value is reached.

The node requests a client connection, and, if no connections are available for sending data, the output node requests that the pool creates a new connection. If the `maximumConnections` property has not been exceeded, a new connection is created.

When the connection has been established, the data is sent. If the data has not been sent successfully within the time limit specified by the node's `Timeout sending a data record` property, an exception is thrown.

Properties in the local environment can override the TCP/IP connection used by the node:

Table 254. Input local environment properties

Location in local environment	Description
$LocalEnvironment/Destination/TCPIP/Output/Hostname	The host name used to make a connection.
$LocalEnvironment/Destination/TCPIP/Output/Port	The port number used to make a connection.
$LocalEnvironment/Destination/TCPIP/Output/Id	The ID of the socket being used. This value is an internal identifier used by WebSphere Message Broker to uniquely identify a connection.
$LocalEnvironment/Destination/TCPIP/Output/ReplyId	The Reply ID that has been stored on this connection. It can be any text string.
$LocalEnvironment/Destination/TCPIP/Output/Timeout	The timeout value used when sending data to the TCP/IP client connection. This value overrides the `Timeout sending a data record` property specified on the node.

You can dynamically choose the connection details (host name and port number), and the connection used (ID), by using this property. You can also set the Reply ID on the connection. The Reply ID enables a string to be stored in the connection and to be seen in the local environment. You can use this connection to store Reply IDs from other TCPIP nodes or from other transports, such as WebSphere MQ

The output of the node contains the same information as the input, and any fields that were missing from the input are updated with details from the connection used. For example, if the `Id` property is not provided as input (because you want to create a new connection or reuse a pool connection), the output local environment contains the ID of the connection that is used.

Table 255. Output local environment properties

Location in local environment	Description
$LocalEnvironment/WrittenDestination/TCPIP/Output/ConnectionDetails/Hostname	The host name used to make a connection.
$LocalEnvironment/WrittenDestination/TCPIP/Output/ConnectionDetails/Port	The port number used to make a connection.
$LocalEnvironment/WrittenDestination/TCPIP/Output/ConnectionDetails/OpenTimestamp	The time stamp when the connection was first opened.
$LocalEnvironment/WrittenDestination/TCPIP/Output/ConnectionDetails/CloseTimestamp	The time stamp when the connection was closed (null if not yet closed).
$LocalEnvironment/WrittenDestination/TCPIP/Output/ConnectionDetails/SequenceNumber	The sequence number of the message received on this connection. The first record has a sequencing number of 1; the second record has a sequencing number of 2, and so on.
$LocalEnvironment/WrittenDestination/TCPIP/Output/ConnectionDetails/Id	The ID of the socket being used. This ID is an internal identifier used by WebSphere Message Broker to uniquely identify a connection.
$LocalEnvironment/WrittenDestination/TCPIP/Output/ConnectionDetails/ReplyId	The Reply ID that has been stored on this connection. It can be any text string.

If the connection closes (or any other type of exception occurs) while using the TCP/IP transport, an exception is thrown. This exception goes to the Failure terminal if it is connected, otherwise the exception returns back down the flow.

The node also has a Close input terminal. If a message is sent to this terminal, the connection is closed using a combination of the details provided in the node and the local environment.

The TCPIPClientOutput node is contained in the **TCPIP** drawer of the palette and is represented in the workbench by the following icon:

Using the TCPIPClientOutput node in a message flow:
The TCPIPClientOutput node can be used in any message flow that needs to send data to an external application. Look at the following samples to see how to use this node:

- TCPIP Client Nodes
- TCPIP Handshake

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Configuring the TCPIPClientOutput node:
When you have put an instance of the TCPIPClientOutput node into a message flow, you can configure it (for more information, see “Configuring a message flow node” on page 1503). The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk in that view.

To configure the TCPIPClientOutput node:

1. Optional: On the **Description** tab, enter a short description, a long description, or both. You can also rename the node on this tab.
2. On the **Basic** tab, set the properties that determine how the TCP/IP connection is controlled.
   - Use the `Connection details` property to specify either the host name and port number to be used, or the name of a configurable service. This property is mandatory. The following formats are supported:
     - `Configurable service name`. This value is used to look up the port and host name in configurable services. For example, `TCPIPProfile1`.
     - `<Hostname>:<Port>`. This value is the host name followed by the port number (separated by a colon). For example, `tcpip.server.com:1111`.
     - `<Port>`. This value is the port number. In this case, the host name is assumed to be `localhost`.
   - Use the `Timeout sending a data record (seconds)` property to specify how long the node waits when trying to send data. You can specify any length of time in seconds. When the specified time has been exceeded, all available data is sent to the Failure terminal. The default is `60` seconds.
3. On the **Advanced** tab, set the properties that determine how the data stream is controlled.
   - Use the `Send to` property to specify whether the data is to be sent to one connection or to all available connections:
     - Select `One connection` to send the message to only one connection, as specified by the node properties and message overrides. This value is the default.
     - Select `All available connections` to send the data to all available connections.
   - Use the `Close connection` property to specify when and how to close the connection.
     - Select `No` to leave the connection open. This value is the default.
     - Select `After timeout` to close the connection when a timeout occurs.
     - Select `After data has been sent` to close the connection when the end of the record has been sent.
   - Select `Close output stream after a record has been sent` to close the output stream as soon as the data has been sent. This property is not selected by default.
   - Use the `Output Stream Modification` property to specify whether to reserve or release the output stream. These options are available only if you have not selected the `Close output stream after a record has been sent` property.
     - Select `Leave unchanged` to leave the output stream as it was when it entered the node. This value is selected by default.
     - Select `Release output stream` to specify that this output stream is returned to the pool and is available for use by any output node.
     - Select `Reserve output stream (for use by future TCPIP output nodes)` to specify that this output stream can be used only by this node and by other output nodes that request it by specifying the connection ID. When the connection input stream is reserved, no other nodes can use it without specifying the correct connection ID.
     - Select `Reserve output stream (for use by future TCPIP output nodes) then release after propagate` to specify that this output stream can be used only by this node and output nodes that request it by specifying the

correct connection ID. After the message has been propagated, this output stream is returned to the pool and becomes available for use by any output node.

- Use the `Input Stream Modification` property to specify whether to reserve the input stream for use only by input and receive nodes that specify the connection ID, or to release it at the end of the flow.
  - Select `Leave unchanged` to leave the input stream as it was when it entered the node. This value is selected by default.
  - Select `Release input stream` to specify that this input stream is returned to the pool and is available for use by any input or receive node.
  - Select `Reserve input stream (for use by future TCPIP input and receive nodes)` to specify that this input stream can be used only by this node and by other input or receive nodes that request it by specifying the connection ID. When the connection input stream is reserved, no other nodes can use it without specifying the correct connection ID.
  - Select `Reserve input stream (for use by future TCPIP input and receive nodes) then release after propagate` to specify that this input stream can be used only by this node and receive nodes that request it by specifying the correct connection ID. After the message has been propagated, this input stream is returned to the pool and becomes available for use by any input or receive node.

4. On the **Request** tab, specify the location of the data to be written. You can specify the properties on this tab as XPath or ESQL expressions. Content Assist is available in the properties pane and also in the XPath Expression Builder, which you can invoke by clicking **Edit** to the right of each property.
   a. In `Data location`, specify the input data location. This value is the location in the input message tree that contains the record to be written. The default value is `$Body`, which is the entire message body (`$InputRoot.Body`).

      When you are specifying this property, and the data in the message tree that it identifies is owned by a model-driven parser (such as the MRM parser or XMLNSC parser), consider the following issues:

      - If you are using MRM CWF format, ensure that the identified message tree exists as a message definition. If this value is defined as a global element only, exceptions BIP5180 and BIP5167 are generated.
      - If you are using MRM TDS format, the serialization of the identified message is successful if the element is defined as a global element or message. However, if the identified field is not found as a global element or message, note that:
        - If this value is a leaf field in the message tree, the field is written as self-defining. No validation occurs even if validation is enabled.
        - If this value is a complex element, an internal exception is generated, BIP5522, indicating that the logical type cannot be converted to a string.
      - If you are using MRM XML, the events are similar as for the MRM TDS format except that, if the field is a complex element, it is written as self-defining.
      - If you use the XMLNSC parser, no validation occurs even if validation is enabled.

   b. In `Hostname location`, specify the location of the value to override the `Hostname` set in the `Connection details` property of the **Basic** tab. If you do not specify a location, the default value is `$LocalEnvironment/Destination/TCPIP/Output/Hostname`.

c. In Port location, specify the location of the value to override the Port number set in the Connection details property of the **Basic** tab. If you do not specify a location, the default value is $LocalEnvironment/Destination/TCPIP/Output/Port.

d. In ID location, specify the location of the Id of the socket being used. This internal identifier is used by WebSphere Message Broker to uniquely identify a connection. If you do not specify a location, the default value is $LocalEnvironment/Destination/TCPIP/Output/Id.

e. In Reply ID location, specify the location of the Reply ID that is stored on the connection being used. The Reply ID can be used when data is returned in an input node. If you do not specify a location, the default value is $LocalEnvironment/Destination/TCPIP/Output/ReplyId.

5. Use the **Records and Elements** tab to specify how the TCPIPClientOutput node writes the record that is derived from the message.
   - In Record definition, choose from:
     - Record is Unmodified Data to specify that records are left unchanged. This value is the default.
     - Record is Fixed Length Data to specify that records are padded to a specified length if necessary. You specify this length in the Length property. If the record is longer than the value specified in Length, the node generates an exception. Use the Padding byte (hexadecimal) property to specify the byte to be used for padding the message to the required length.
     - Record is Delimited Data to specify that records are separated by a delimiter and accumulated by concatenation. The delimiter is specified by the Delimiter, Custom delimiter, and Delimiter type properties. The file is finished only when a message is received on the Finish File terminal.
   - In Length (bytes), specify the length (in bytes) of records when Record is Fixed Length Data is specified in Record definition. Records longer than this value cause an exception to be issued. This value must be in the range 1 byte through 100 MB. The default is 80 bytes.
   - When Record is Fixed Length Data is specified in Record definition, use Padding byte (hexadecimal) to specify the byte to be used when padding records to the specified length if they are shorter than this length. Specify this value as 2 hexadecimal digits. The default value is X'20'.
   - In Delimiter, specify the delimiter to be used if you specify Record is Delimited Data in Record definition. Choose from:
     - Broker System Line End to specify that a line end sequence of bytes is used as the appropriate delimiter for the file system on which the broker is to run. This value is the default. For example, on Windows systems, this value is a 'carriage-return, line-feed' pair (X'0D0A'); on UNIX systems, this value is a single 'line-feed' byte (X'0A'); on z/OS systems, this value is a 'newline' byte (X'15').
     - Custom Delimiter (hexadecimal) to specify that the explicit delimiter sequence defined in the Custom delimiter property is to be used to delimit records.
   - In Custom delimiter (hexadecimal), specify the delimiter sequence of bytes to be used to delimit records when Custom Delimiter is specified in the Delimiter property. Specify this value as an even-numbered string of hexadecimal digits. The default is X'0A' and the maximum length of the string is 16 bytes.

- If you specify `Record is Delimited Data` in `Record definition`, use `Delimiter type` to specify how the delimiter is to separate records. Choose from:
  - `Postfix` to specify that the delimiter is added after each record that is written. This value is the default.
  - `Infix` to specify that the delimiter is only inserted between any two adjacent records.

6. On the **Validation** tab, specify the parser validation properties of the node. For more information about validation, see "Validating messages" on page 1478. For information about how to complete this tab, see "Validation tab properties" on page 4169.

Terminals and properties:
The TCPIPClientOutput node terminals are described in the following table.

Terminal	Type	Description
In	Input data	The input terminal that accepts a message for processing by the node.
Close	Input control	The input terminal to which a message is routed when the connection given in the local environment is closed.
Out	Output data	The output terminal to which the message is routed if it is successfully sent to an external resource. The message received on the In terminal is propagated to the Out terminal and is left unchanged except for the addition of status information.
Close	Output control	The output terminal to which a message propagated from the Close input terminal is routed.
Failure	Output data	The output terminal to which the message is routed if a failure is detected in the node.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Description properties of the TCPIPClientOutput node are described in the following table:

Property	M	C	Default	Description
Node name	No	No	TCPIPClientOutput	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The Basic properties of the TCPIPClientOutput node are described in the following table:

Property	M	C	Default	Description	`mqsiapplybaroverride` command property
Connection details	Yes	Yes		A string containing either the host name and port number to be used, or the name of a configurable service.	connectionDetails

Property	M	C	Default	Description	mqsiapplybaroverride command property
Timeout sending a data record (seconds)	Yes	Yes	60	Specifies how long the node waits when attempting to send data. You can specify any length of time in seconds.	timeoutSendingData

The Advanced properties of the TCPIPClientOutput node are described in the following table.

Property	M	C	Default	Description
Close connection	Yes	No	No	Controls when the connection is closed, or if it remains open. Valid options are: • No • After Timeout • After Data has been Sent
Close output stream after a record has been sent	Yes	No	Cleared	Specifies whether to close the output stream as soon as the data has been sent. This property is not selected by default.
Output Stream Modification	No	No	Leave unchanged	Specifies whether to reserve this output stream or release it and return it to the pool for use by any output node. Valid options are: • Leave unchanged • Release output stream • Reserve output stream (for use by future TCPIP nodes) • Reserve output stream (for use by future TCPIP nodes) then release at end of flow
Input Stream Modification	No	No	Leave unchanged	Specifies whether to reserve the input stream for use only by input and receive nodes that specify the connection ID, or to release it at the end of the message flow. Valid options are: • Leave unchanged • Release input stream • Reserve input stream (for use by future TCPIP nodes) • Reserve input stream (for use by future TCPIP nodes) then release at end of flow When the connection input stream is reserved, no other nodes can use it without specifying the correct connection ID. If the input stream is released after the message has been propagated, it is returned to the pool and becomes available for use by any input or receive node.
Send to:	Yes	No	One Connection	Specifies whether the data is to be sent to one connection or to available connections. Valid options are: • One Connection • All Available Connections

The Request properties of the TCPIPClientOutput node are described in the following table:

Property	M	C	Default	Description
Data location	Yes	No	`$Body`	The location in the input message tree containing the record to be written.
Hostname location	Yes	No	`$LocalEnvironment/Destination/TCPIP/Output/Hostname`	The message element location containing the host name.
Port location	Yes	No	`$LocalEnvironment/Destination/TCPIP/Output/Port`	The message element location containing the port.
ID location	Yes	No	`$LocalEnvironment/Destination/TCPIP/Output/Id`	The message element location containing the ID.
Reply ID location	Yes	No	`$LocalEnvironment/Destination/TCPIP/Output/ReplyId`	The message element location containing the reply ID.

The Records and Elements properties of the TCPIPClientOutput node are described in the following table:

Property	M	C	Default	Description
Record definition	Yes	No	`Record is Unmodified Data`	This property controls how the records derived from the message are written. Valid options are: • `Record is Unmodified Data` • `Record is Fixed Length Data` • `Record is Delimited Data`
Length (bytes)	Yes	No	`0`	The required length of the output record. This property applies only when `Record is Fixed Length Data` is specified in `Record definition`.
Padding byte (hexadecimal)	Yes	No	`20`	The two-digit hexadecimal byte to be used to pad short messages when `Record is Fixed Length Data` is specified in `Record definition`.
Delimiter	Yes	No	`Broker System Line End`	The delimiter to be used when `Record is Delimited Data` is specified in `Record definition`. Valid options are: • `Broker System Line End` • `Custom Delimiter (Hexadecimal)`
Custom delimiter (hexadecimal)	No	No	None	The delimiter byte sequence to be used when `Record is Delimited Data` is specified in the `Record definition` property and `Custom Delimiter (Hexadecimal)` is specified in the `Delimiter` property.
Delimiter type	Yes	No	`Postfix`	This property specifies the way in which the delimiters are to be inserted between records when `Record is Delimited Data` is specified in `Record definition`. Valid options are: • `Infix` • `Postfix`

The Validation properties of the TCPIPClientOutput node are described in the following table.

For a full description of these properties, see “Validation properties” on page 4169.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Validate	No	Yes	Inherit	This property controls whether validation takes place. Valid values are: • None • Content and Value • Content • Inherit	validateMaster
Failure action	No	No	Exception	This property controls what happens if validation fails. You can set this property only if you set Validate to Content or Content and Value. Valid values are: • User Trace • Local Error Log • Exception • Exception List	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"WebSphere Broker TCP/IP Transport" on page 1735
The WebSphere Broker TCP/IP Transport is a service that connects applications that use raw TCP/IP sockets for transferring data.

"TCP/IP nodes" on page 1738
WebSphere Message Broker implements access to the TCP/IP input and output streams through a series of nodes.

"Scenarios for WebSphere Message Broker and TCP/IP" on page 1746
Two example scenarios show how you might use TCP/IP and WebSphere Message Broker as part of a business solution.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Working with TCP/IP" on page 1750
You can use WebSphere Message Broker TCPIP nodes and TCP/IP configurable services to perform various tasks.

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

"Validating messages" on page 1478
The broker provides validation based on the message model for predefined messages.

Related reference:

"Validation properties" on page 4169
You can control validation by setting properties on the Validate and Parser Options tabs for the nodes that are listed in the following table.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"TCPIPClientInput node" on page 4854
Use the TCPIPClientInput node to create a client connection to a raw TCP/IP socket, and to receive data over that connection.

"TCPIPServerOutput node" on page 4903
Use the TCPIPServerOutput node to create a server connection to a raw TCP/IP socket, and to send data over the connection to an external application.

"TCPIPServerInput node" on page 4890
Use the TCPIPServerInput node to create a server connection to a raw TCPIP socket, and to receive data over that connection.

"TCPIPClientReceive node"
Use the TCPIPClientReceive node to receive data over a client TCP/IP connection.

"TCPIPServerReceive node" on page 4913
Use the TCPIPServerReceive node to receive data over a server TCP/IP connection.

TCPIPClientReceive node

Use the TCPIPClientReceive node to receive data over a client TCP/IP connection.

This topic contains the following sections:
- "Purpose"
- "Using the TCPIPClientReceive node in a message flow" on page 4879
- "Configuring the TCPIPClientReceive node" on page 4879
- "Terminals and properties" on page 4884

Purpose:
The TCPIPClientReceive node waits for data to be received on a TCP/IP connection, and retrieves the data. If the connection is closed, an exception is thrown.

You can configure the broker to use SSL for TCP/IP nodes; see "SSL and the TCP/IP nodes" on page 551.

When a connection is established, the data is sent to the TCPIPClientReceive node. If the TCPIPClientReceive node fails to receive all of the data within the time

specified in the `Timeout waiting for a data record` property, the message is sent to the timeout terminal; if no timeout terminal is connected, an exception is thrown.

Properties in the local environment can override the TCP/IP connection used by the node.

Table 256. Input local environment properties

Location in local environment for input to node	Description
$LocalEnvironment//TCPIP/Receive/Hostname	The host name used to make a connection.
$LocalEnvironment//TCPIP/Receive/Port	The port number used to make a connection.
$LocalEnvironment/TCPIP/Receive/Id	The ID of the socket being used. This ID is an internal identifier used by WebSphere Message Broker to uniquely identify a connection.
$LocalEnvironment/TCPIP/Receive/ReplyId	The Reply ID to be stored on this connection. This ID can then be used when data is returned on an input node. The Reply ID can be any text string.
$LocalEnvironment/TCPIP/Receive/Timeout	The timeout value used when waiting for data on the TCP/IP client connection. This value overrides the `Timeout waiting for a data record` property specified on the node.
$LocalEnvironment/TCPIP/Receive/Length	The value used to override the number of bytes to be read when using fixed size records. This value overrides the `Length (bytes)` property specified on the node. If the `Record detection` property is set to anything other than `Fixed Length`, the local environment field is ignored. If this field is not present or evaluates to null, it is ignored and the value on the node is used.

These properties enable the connection details (host name and port number) and the connection used (ID) to be selected dynamically. You can also set the Reply ID on the connection, which enables a string to be stored in the connection and to be seen in the local environment of any data that is received back from this connection. You can use this connection to store Reply IDs from other TCPIP nodes or from other transports, such as WebSphere MQ.

When a record has been retrieved, the ConnectionDetails field in the local environment is populated with the details of the connection that is being used.

Table 257. Output local environment properties

Location in local environment for output from node	Description
$LocalEnvironment/TCPIP/Receive/ConnectionDetails/Type	The client.
$LocalEnvironment/TCPIP/Receive/ConnectionDetails/Hostname	The host name used to make a connection.
$LocalEnvironment/TCPIP/Receive/ConnectionDetails/Port	The port number used to make a connection.
$LocalEnvironment/TCPIP/Receive/ConnectionDetails/OpenTimestamp	The time stamp when the connection was first opened.
$LocalEnvironment/TCPIP/Receive/ConnectionDetails/CloseTimestamp	The time stamp when the connection was closed (null if not yet closed).

Table 257. Output local environment properties (continued)

Location in local environment for output from node	Description
$LocalEnvironment/TCPIP/Receive/ConnectionDetails/ SequenceNumber/InputRecord	The sequence number of the message that is received on this connection. The first record has a sequencing number of 1; the second record is 2; and so on.
$LocalEnvironment/TCPIP/Receive/ConnectionDetails/ SequenceNumber/OutputRecord	The sequence number of the message sent on this connection. The first record has a sequencing number of 1; the second record is 2; and so on.
$LocalEnvironment/TCPIP/Receive/ConnectionDetails/ Id	The ID of the socket being used. This ID is an internal identifier used by WebSphere Message Broker to uniquely identify a connection.
$LocalEnvironment/TCPIP/Receive/ConnectionDetails/ ReplyId	The Reply ID that is stored on this connection. This ID can be any text string.

The TCPIPClientReceive node is contained in the **TCPIP** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Message structure

The TCPIPClientReceive node handles messages in the following message domains:

- XMLNSC
- DataObject
- JSON
- BLOB
- MIME
- MRM
- JMSMap
- JMSStream
- XMLNS

Using the TCPIPClientReceive node in a message flow:
Look at the following samples to see how to use the TCPIPClientReceive node:

- TCPIP Client Nodes
- TCPIP Handshake

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Configuring the TCPIPClientReceive node:
When you have put an instance of the TCPIPClientReceive node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view.

All mandatory properties that do not have a default value defined are marked with an asterisk.

Configure the TCPIPClientReceive node:

1. Optional: On the **Description** tab, enter a `Short description`, a `Long description`, or both. You can also rename the node on this tab.
2. On the **Basic** tab, set the properties that determine how the TCP/IP connection is controlled.
   - Use the `Connection details` property to specify either the host name and port number to be used, or the name of a configurable service. This property is mandatory. The following formats are supported:
     - `Configurable service name`. This value is used to look up the port and host name in configurable services. For example, `TCPIPProfile1`.
     - `<Hostname>:<Port>`. This value is the host name followed by the port number (separated by a colon); for example, `tcpip.server.com:1111`
     - `<Port>`. This value is the port number. In this case, the host name is assumed to be `localhost`.
   - Use the `Timeout waiting for a data record (seconds)` property to specify how long the node listens on a connection for more data after the first byte of data has arrived. You can specify any length of time in seconds. The default is `60` seconds. When the specified time has been exceeded, all available data is sent to the Failure terminal.
3. On the **Advanced** tab, set the properties that determine how the data stream is controlled.
   - Use the `Close connection` property to specify when and how to close the connection.
     - Select `No` to leave the connection open. This value is the default.
     - Select `After timeout` to close the connection when a timeout occurs.
     - Select `After data has been received` to close the connection when the end of the record is found.
   - Select `Close input stream after a record has been received` to close the input stream as soon as the data has been retrieved. By default this property is not selected. When the connection input stream is reserved, no other node can use it without knowing the ID.
   - Use the `Input Stream Modification` property to specify whether to reserve the input stream for use only by input and receive nodes that specify the connection ID, or to release the input stream at the end of the flow.
     - Select `Leave unchanged` to leave the input stream as it was when it entered the node. This value is selected by default.
     - Select `Release input stream` to specify that this input stream is returned to the pool and is available for use by any input or receive node.
     - Select `Reserve input stream (for use by future TCPIP input and receive nodes)` to specify that this input stream can be used only by this node and by other input or receive nodes that request it by specifying the connection ID. When the connection input stream is reserved, no other nodes can use it without specifying the correct connection ID.
     - Select `Reserve input stream (for use by future TCPIP input and receive nodes) then release after propagate` to specify that this input stream can be used only by this node and receive nodes that request it by specifying the correct connection ID. After the message has been propagated, this input stream is returned to the pool and becomes available for use by any input or receive node.

- Use the Output Stream Modification property to specify whether to reserve or release the output stream. These options are available only if you have not selected the Close output stream after a record has been sent property.
  - Select Leave unchanged to leave the output stream as it was when it entered the node. This value is selected by default.
  - Select Release output stream to specify that this output stream is returned to the pool and is available for use by any output node.
  - Select Reserve output stream (for use by future TCPIP output nodes) to specify that this output stream can be used only by this node and by other output nodes that request it by specifying the connection ID. When the connection input stream is reserved, no other nodes can use it without specifying the correct connection ID.
  - Select Reserve output stream (for use by future TCPIP output nodes) then release after propagate to specify that this output stream can be used only by this node and output nodes that request it by specifying the correct connection ID. After the message has been propagated, this output stream is returned to the pool and becomes available for use by any output node.

4. On the **Request** tab, specify the location of the data to be written. You can specify the properties on this tab as XPath or ESQL expressions. Content Assist is available in the Properties view and also in the XPath Expression Builder, which you can run by clicking **Edit** to the right of each property.
   - In Hostname location, specify the location of the value to override the Hostname that is set in the Connection details property of the **Basic** tab. If you do not specify a location, the default value is $LocalEnvironment/TCPIP/Receive/Hostname.
   - In Port location, specify the location of the value to override the Port that is set in the Connection details property of the **Basic** tab. If you do not specify a location, the default value is $LocalEnvironment/TCPIP/Receive/Port.
   - In ID location, specify the location of the Id of the socket being used. This internal identifier is used by WebSphere Message Broker to uniquely identify a connection. If you do not specify a location, the default value is $LocalEnvironment/TCPIP/Receive/Id.
   - In Reply ID location, specify the location of the Reply ID that is stored on the connection that is being used. The Reply ID can be used when data is returned in an input node. If you do not specify a location, the default value is $LocalEnvironment/TCPIP/Receive/ReplyId.
5. On the **Result** tab, set values for the properties that determine where the reply is stored.
   - Use the Output data location property to specify the start location in the output message tree where the parsed elements from the bit string of the message are stored. The default value is $OutputRoot.
   - Use the Copy local environment property to specify whether the local environment is copied to the output message.
     - If Copy local environment is selected, a new copy of the local environment is created in the tree, and it is populated with the contents of the local environment from the preceding node. Therefore, if a node changes the local environment, the upstream nodes are not affected by those changes because they have their own copies. This value is the default.

- If Copy local environment is not selected, the node does not generate its own copy of the local environment, but uses the local environment that is passed to it by the preceding node. Therefore, if a node changes the local environment, the changes are reflected by the upstream nodes.

6. On the **Input Message Parsing** tab, set values for the properties that the node uses to determine how to parse the incoming message.

   If the incoming message has an MQRFH2 header, you do not have to set values for the Input Message Parsing properties because the values are derived from the <mcd> folder in the MQRFH2 header; for example:

   ```
   <mcd><Msd>MRM</Msd><Set>DHM4UO906S001</Set><Type>receiptmsg1</Type>
   <Fmt>XML</Fmt></mcd>
   ```

   If you set values, and if they differ from the values in the MQRFH2 header, the values in the MQRFH2 header take precedence.

   - In Message domain, select the name of the parser that you are using from the list. The default is BLOB. You can choose from the following options:
     - XMLNSC
     - DataObject
     - JSON
     - BLOB
     - MIME
     - MRM
     - JMSMap
     - JMSStream
     - XMLNS

     You can also specify a user-defined parser, if appropriate.
   - If you are using the MRM or IDOC parser, or the XMLNSC parser in validating mode, select the Message set that you want to use. This list is populated with available message sets when you select MRM, XMLNSC, or IDOC as the domain.
   - If you are using the MRM parser, select the correct message type from the list in Message type. This list is populated with available message types when you select the MRM parser.
   - If you are using the MRM or IDOC parser, select the correct message format from the list in Message format. This list is populated with available message formats when you select the MRM or IDOC parser.
   - Specify the message coded character set ID in Message coded character set ID.
   - Select the message encoding from the list in Message encoding or specify a numeric encoding value. For more information about encoding, see "Data conversion" on page 1151.

7. On the **Parser Options** subtab:
   - Parse timing is, by default, set to On Demand, which causes parsing of the message to be delayed. To cause the message to be parsed immediately, see "Parsing on demand" on page 4173.
   - If you are using the XMLNSC parser, set values for the properties that determine how the XMLNSC parser operates. For more information, see "Manipulating messages in the XMLNSC domain" on page 2546.

8. Use the **Records and Elements** tab to specify how the data is interpreted as records. Only one record is retrieved each time the TCPIPClientReceive node is invoked; therefore, if the TCP/IP stream contains multiple logical messages, you must call the node multiple times to receive all the messages.

- Use the Record detection property to determine how the data is split into records, each of which generates a single message. Choose from the following options:
  - Connection closed specifies that all of the data sent during a connection is a single record.
  - Fixed Length specifies that each record is a fixed number of bytes in length. Each record contains the number of bytes specified in the Length property, except possibly a shorter final record in the file.
  - Select Delimited if the records that you are processing are separated, or terminated, by a DOS or UNIX line end or by a sequence of user-defined delimiter bytes. Specify the delimiter and delimiter type in the Delimiter and Delimiter type properties.
  - Select Parsed Record Sequence if the data contains a sequence of one or more records that are serially recognized by the parser specified in Message domain. The node propagates each recognized record as a separate message. If you select this Record detection option, the parser specified in Message domain must be either XMLNSC or MRM (either CWF or TDS physical format).
- If you set Record detection to Fixed Length, use Length to specify the required length of the output record. This value must be in the range 1 byte through 100 MB. The default is 80 bytes.

  If you set Record detection to Connection closed, Fixed Length, or Delimited, a limit of 100 MB applies to the length of the records. If you set Record detection to Parsed Record Sequence, the TCPIPClientReceive node does not determine or limit the length of a record. Nodes that are downstream in the message flow might try to determine the record length or process a long record. If you intend to process large records in this way, ensure that your broker has sufficient memory. You might have to apply flow techniques described in the Large Messaging sample to best use the available memory.
- If you set Record detection to Delimited, use Delimiter to specify the delimiter to be used. Choose from the following options:
  - DOS or UNIX Line End, on UNIX systems, specifies the line feed character (<LF>, X'0A'), and, on Windows systems, specifies a carriage return character followed by a line feed character (<CR><LF>, X'0D0A'). The node treats both of these strings as delimiters, irrespective of the system on which the broker is running. If both strings can be seen in the same record, the node recognizes both as delimiters. The node does not recognize X'15' which, on z/OS systems, is the 'newline' byte; set this property to Custom Delimiter and set Custom delimiter to 15 if your input file is coded using EBCDIC new lines.
  - Custom Delimiter (hexadecimal), permits a sequence of bytes to be specified in Custom delimiter (hexadecimal)
- In Custom delimiter (hexadecimal), specify the delimiter byte or bytes to be used when Delimiter is set to Custom delimiter (hexadecimal). Specify this value as an even-numbered string of hexadecimal digits. The default is X'0A' and the maximum length of the string is 16 bytes (represented by 32 hexadecimal digits).
- If you set Record detection to Delimited, use Delimiter type to specify the type of delimiter. Permitted values are:
  - Infix. If you select this value, each delimiter separates records. If the data ends with a delimiter, the (zero length) data following the final delimiter is still propagated, although it contains no data.

– `Postfix`. If you specify this value, each delimiter terminates records. If the data ends with a delimiter, no empty record is propagated after the delimiter. If the data does not end with a delimiter, it is processed as if a delimiter follows the final bytes of the data. `Postfix` is the default value.

- The TCPIPClientReceive node considers each occurrence of the delimiter in the input as either separating (infix) or terminating (postfix) each record. If the data begins with a delimiter, the node treats the (zero length) contents preceding that delimiter as a record and propagates an empty record to the flow. The delimiter is never included in the propagated message.

9. Use the **Validation** tab to provide validation based on the message set for predefined messages. For more information about validation, see "Validating messages" on page 1478. For information about how to complete this tab, see "Validation tab properties" on page 4169.

Terminals and properties:
The terminals of the TCPIPClientReceive node are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Out	The output terminal to which the message is routed if it is successfully retrieved from an external resource. If no errors occur in the input node, a message received from an external resource is always sent to the Out terminal first.
Timeout	The terminal to which a message is sent when the time specified in the `Timeout waiting for a data record` property has been exceeded. The message text is `timeout value is exceeded`.
Failure	The output terminal to which the message is routed if an error occurs. These errors include failures caused by retry processing. Even if the Validation property is set, messages propagated to this terminal are not validated.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Description properties of the TCPIPClientReceive node are described in the following table.

Property	M	C	Default	Description
Node name	No	No	TCPIPClientReceive	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The Basic properties of the TCPIPClientReceive node are described in the following table.

Property	M	C	Default	Description	`mqsiapplybaroverride` command property
Connection details	Yes	Yes		A string containing either the host name and port number to be used, or the name of a configurable service.	connectionDetails

Property	M	C	Default	Description	mqsiapplybaroverride command property
Timeout waiting for a data record (seconds)	Yes	Yes	60	Specifies how long the node listens on a connection for more data after the first byte of data has arrived. You can specify any length of time in seconds.	timeoutWaitingForData

The Advanced properties of the TCPIPClientReceive node are described in the following table.

Property	M	C	Default	Description
Close connection	Yes	No	No	Controls when the connection is closed, or if it remains open. Valid options are: • No • After Timeout • After Data has been Received
Close input stream after a record has been received	Yes	No	Cleared	Specifies whether to close the input stream as soon as the data has been retrieved. When the connection input stream is reserved, no other node can use it without knowing the ID. By default, this property is not selected.
Input Stream Modification	No	No	Leave unchanged	Specifies whether to reserve the input stream for use only by input and receive nodes that specify the connection ID, or to release the input stream at the end of the flow. Valid options are: • Leave unchanged • Release input stream • Reserve input stream (for use by future TCPIP nodes) • Reserve input stream (for use by future TCPIP nodes) then release at end of flow When the connection input stream is reserved, no other nodes can use it without specifying the correct connection ID. If the input stream is released after the message has been propagated, it is returned to the pool and becomes available for use by any input or receive node.
Output Stream Modification	No	No	Leave unchanged	Specifies whether to reserve this output stream or release it and return it to the pool for use by any output node. Valid options are: • Leave unchanged • Release output stream • Reserve output stream (for use by future TCPIP nodes) • Reserve output stream (for use by future TCPIP nodes) then release at end of flow

The Request properties of the TCPIPClientReceive node are described in the following table:

Property	M	C	Default	Description
Host name location	Yes	No	$LocalEnvironment/TCPIP/Receive/Hostname	The message element location that contains the host name.

Property	M	C	Default	Description
Port location	Yes	No	`$LocalEnvironment/TCPIP/Receive/Port`	The message element location that contains the port.
ID location	Yes	No	`$LocalEnvironment//TCPIP/Receive/Id`	The message element location that contains the ID.
Reply ID location	Yes	No	`$LocalEnvironment/TCPIP/Receive/ReplyId`	The message element location that contains the Reply ID.
Record length location	No	No	`$LocalEnvironment/TCPIP/Receive/Length`	The message element location that contains the record length to be read. Specify the location of the value to override the `Length (bytes)` property on the **Records and elements** tab. If you do not specify a location, the default value is `$LocalEnvironment/TCPIP/Receive/Length`.

The Result properties of the TCPIPClientReceive node are described in the following table:

Property	M	C	Default	Description
Output data location	No	No	`$OutputRoot`	The start location in the output message tree where the parsed elements from the bit string of the message are stored.
Copy local environment	No	No	Selected	Specifies if the local environment is copied to the output message.

The Input Message Parsing properties of the TCPIPClientReceive node are described in the following table.

Property	M	C	Default	Description	**`mqsiapplybaroverride` command property**
Message domain	No	No		The domain that is used to parse the incoming message.	
Message set	No	No		The name or identifier of the message set in which the incoming message is defined. If you set this property, then later update the project dependencies to remove this message set reference, a warning is issued. Either update the `Message Set` property, or restore the reference to this message set project.	
Message type	No	No		The name of the incoming message.	
Message format	No	No		The name of the physical format of the incoming message.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Message coded character set ID	Yes	No	Broker System Default	The ID of the coded character set that is used to interpret the data being read.	messageCodedCharSetIdProperty
Message encoding	Yes	No	Broker System Determined	The encoding scheme for numbers and large characters used to interpret the data that is being read. Valid values are Broker System Determined or a numeric encoding value. For more information about encoding, see "Data conversion" on page 1151.	messageEncodingProperty

The Parser Options properties of the TCPIPClientReceive node are described in the following table.

Property	M	C	Default	Description
Parse timing	No	No	On Demand	This property controls when an input message is parsed. Valid values are: • On Demand • Immediate • Complete For a full description of this property, see "Parsing on demand" on page 4173.
Build tree using XML schema data types	No	No	Cleared	This property controls whether the syntax elements in the message tree have data types taken from the XML Schema.
Use XMLNSC compact parser for XMLNS domain	No	No	Cleared	This property controls whether the XMLNSC Compact Parser is used for messages in the XMLNS domain. If you set this property, the message datais displayed under XMLNSC in nodes that are connected to the output terminal when the input MQRFH2 header or Input Message Parsing property, Message Domain, is XMLNS.
Retain mixed content	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters mixed text in an input message. If you select the check box, elements are created for mixed text. If you clear the check box, mixed text is ignored and no elements are created.
Retain comments	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters comments in an input message. If you select the check box, elements are created for comments. If you clear the check box, comments are ignored and no elements are created.
Retain processing instructions	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters processing instructions in an input message. If you select the check box, elements are created for processing instructions. If you clear the check box, processing instructions are ignored and no elements are created.
Opaque elements	No	No	Blank	This property is used to specify a list of elements in the input message that are to be parsed opaquely by the XMLNSC parser.

The Records and Elements properties of the TCPIPClientReceive node are described in the following table:

Property	M	C	Default	Description
Record detection	Yes	No	`Connection Closed`	The mechanism used to identify records in the input data. Valid options are: • `Connection Closed` • `Fixed Length` • `Delimited` • `Parsed Record Sequence`
Length (bytes)	Yes	No	`0`	The length of each record, in bytes, when `Fixed Length` record detection is selected.
Delimiter	Yes	No	`DOS or UNIX Line End`	The type of delimiter bytes that separate, or ends, each record when `Delimited` record detection is selected. Valid options are: • `DOS or UNIX Line End` • `Custom Delimiter (Hexadecimal)`
Custom delimiter (hexadecimal)	No	No		The delimiter bytes, expressed in hexadecimal, when `Delimited` record detection and `Custom Delimiter (Hexadecimal)` are selected. This property is mandatory only if the `Delimiter` property is set to `Custom Delimiter (Hexadecimal)`.
Delimiter type	Yes	No	`Postfix`	The location of the delimiter when `Delimited` record detection and `Custom Delimiter (Hexadecimal)` are selected. Valid options are: • `Infix` • `Postfix` This property is ignored unless the `Delimiter` property is set to `Custom Delimiter (Hexadecimal)`.

The Validation properties of the TCPIPClientReceive node are described in the following table.

For a full description of these properties, see "Validation properties" on page 4169.

Property	M	C	Default	Description	**`mqsiapplybaroverride` command property**
Validate	No	Yes	`None`	This property controls whether validation takes place. Valid values are • `None` • `Content and Value` • `Content` • `inherit`	validateMaster
Failure action	No	No	`Exception`	This property controls what happens if validation fails. You can set this property only if you set `Validate` to `Content` or `Content and Value`. Valid values are: • `User Trace` • `Local Error Log` • `Exception` • `Exception List`	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"WebSphere Broker TCP/IP Transport" on page 1735
The WebSphere Broker TCP/IP Transport is a service that connects applications that use raw TCP/IP sockets for transferring data.

"TCP/IP nodes" on page 1738
WebSphere Message Broker implements access to the TCP/IP input and output streams through a series of nodes.

"Scenarios for WebSphere Message Broker and TCP/IP" on page 1746
Two example scenarios show how you might use TCP/IP and WebSphere Message Broker as part of a business solution.

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

"Parsers" on page 1072
A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from the internal message tree representation.

"Specifying namespaces in the Message Type property" on page 1208
When using the MRM domain, the *Message Type* property is used to specify the name of the message.

"Samples" on page 98
The WebSphere Message Broker Toolkit provides samples that show the features that are available in WebSphere Message Broker, and how to use them. This topic provides links to the information about the individual samples.

Related tasks:

"Working with TCP/IP" on page 1750
You can use WebSphere Message Broker TCPIP nodes and TCP/IP configurable services to perform various tasks.

"Validating messages" on page 1478
The broker provides validation based on the message model for predefined messages.

"Manipulating messages in the XMLNSC domain" on page 2546
If you are writing ESQL to process messages in the XMLNSC domain, it is helpful to learn about the structure of the message tree that the XMLNSC parser builds.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

Related reference:

"Parsing on demand" on page 4173
On-demand parsing, referred to as partial parsing, is used to parse an input message bit stream only as far as is necessary to satisfy the current reference. The parsers that can perform partial parsing of input messages are the JSON, MRM, XML, XMLNS, and XMLNSC parsers.

"Configurable message flow properties" on page 4020
When you add a message flow to a broker archive (BAR) file in preparation for deploying it to a broker, you can set additional properties that influence its run time operation. These properties are available for review and update when you select the **Manage and Configure** tab for the broker archive file.

"Validation properties" on page 4169
You can control validation by setting properties on the Validate and Parser Options tabs for the nodes that are listed in the following table.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"TCPIPClientInput node" on page 4854
Use the TCPIPClientInput node to create a client connection to a raw TCP/IP socket, and to receive data over that connection.

"TCPIPClientOutput node" on page 4867
Use the TCPIPClientOutput node to create a client connection to a raw TCP/IP socket, and to send data over that connection to an external application.

"TCPIPServerOutput node" on page 4903
Use the TCPIPServerOutput node to create a server connection to a raw TCP/IP socket, and to send data over the connection to an external application.

"TCPIPServerInput node"
Use the TCPIPServerInput node to create a server connection to a raw TCPIP socket, and to receive data over that connection.

"TCPIPServerReceive node" on page 4913
Use the TCPIPServerReceive node to receive data over a server TCP/IP connection.

TCPIPServerInput node

Use the TCPIPServerInput node to create a server connection to a raw TCPIP socket, and to receive data over that connection.

This topic contains the following sections:
- "Purpose"
- "Using the TCPIPServerInput node in a message flow" on page 4892
- "Configuring the TCPIPServerInput node" on page 4892
- "Terminals and properties" on page 4896

Purpose:
The TCPIPServerInput node listens on a port and, when a client socket connects to the port, the server socket creates a connection for the client. Unlike the TCPIPClientInput node, the TCPIPServerInput node does not attempt to make a

minimum number of connections, because the server end of the socket cannot initiate new connections, it can only accept them. The TCPIPServerInput node accepts connections up to a maximum value, which is specified in the MaximumConnections property of the TCPIPServer configurable service. By default, the broker can accept up to 100 server connections. For more information, see “**mqsicreateconfigurableservice** command” on page 3849 and “**mqsireportproperties** command” on page 3937.

You can configure the broker to use SSL for TCP/IP nodes; see “SSL and the TCP/IP nodes” on page 551.

When a connection is opened, a message containing details of the connection is sent to the Open terminal; no data is routed to this terminal. Use the Open terminal if processing is required when a connection is opened rather than when data first arrives. The output stream for the connection is reserved until the propagation of the open message has finished (which happens immediately if the terminal is not attached to any other nodes). The options on the **Advanced** tab of the node do not apply to open events; they are applicable only when the first data arrives on the connection and is propagated down the Out terminal.

The first record of data is detected in accordance with properties on the node and then sent to the Out terminal. If an error occurs, including a timeout waiting for data or the closure of a connection while waiting for the full record, the data is sent to the Failure terminal. If the connection closes and no data exists, a message is sent to the Close terminal. Although the message has no data, the local environment does have details of the connection that closed.

For both data and close events, the following local environment is created.

Table 258. Location in local environment

Location in local environment	Description
$LocalEnvironment/TCPIP/Input/ConnectionDetails/Type	The server.
$LocalEnvironment/TCPIP/Input/ConnectionDetails/Hostname	The host name used to make a connection.
$LocalEnvironment/TCPIP/Input/ConnectionDetails/Port	The port number used to make a connection.
$LocalEnvironment/TCPIP/Input/ConnectionDetails/OpenTimestamp	The time stamp when the connection was first opened.
$LocalEnvironment/TCPIP/Input/ConnectionDetails/CloseTimestamp	The time stamp when the connection was closed (null if not yet closed).
$LocalEnvironment/TCPIP/Input/ConnectionDetails/SequenceNumber/InputRecord	The sequence number of the message received on this connection. The first record has a sequencing number of 1; the second record has a sequencing number of 2; and so on.
$LocalEnvironment/TCPIP/Input/ConnectionDetails/SequenceNumber/OutputRecord	The sequence number of the message sent on this connection. The first record has a sequencing number of 1; the second record has a sequencing number of 2; and so on.
$LocalEnvironment/TCPIP/Input/ConnectionDetails/Id	The ID of the socket being used. This ID is an internal identifier used by WebSphere Message Broker to uniquely identify a connection.
$LocalEnvironment/TCPIP/Input/ConnectionDetails/ReplyId	The Reply ID that is stored on this connection. It can be any text string.

Table 258. Location in local environment (continued)

Location in local environment	Description
$LocalEnvironment/TCPIP/Input/ConnectionDetails/ ClientDetails/Hostname	The fully qualified domain name of the computer from which the client connected.
$LocalEnvironment/TCPIP/Input/ConnectionDetails/ ClientDetails/Address	The IP address of the computer from which the client connected.

The TCPIPServerInput node is contained in the **TCPIP** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Message structure

The TCPIPServerInput node handles messages in the following message domains:

- XMLNSC
- DataObject
- JSON
- BLOB
- MIME
- MRM
- JMSMap
- JMSStream
- XMLNS

Using the TCPIPServerInput node in a message flow:
Look at the following samples to see how to use the TCPIPServerInput node:

- TCPIP Client Nodes
- TCPIP Handshake

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Configuring the TCPIPServerInput node:
When you have put an instance of the TCPIPServerInput node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view.

All mandatory properties that do not have a default value defined are marked with an asterisk.

Configure the TCPIPServerInput node:

1. Optional: On the **Description** tab, enter a `Short description`, a `Long description`, or both. You can also rename the node on this tab.
2. On the **Basic** tab, set the properties that determine how the TCPIP connection is controlled.

- Use the Connection details property to specify the port number to be used, or the name of a configurable service. This property is mandatory. The following formats are supported:
  - Configurable service name. This value is used to look up the port number in configurable services. For example, TCPIPProfile1.
  - <Port>. This value is the port number. For example, 1111
- Use the Timeout waiting for a data record (seconds) property to specify how long the node listens on a connection for more data after the first byte of data has arrived. You can specify any length of time in seconds. The default is 60 seconds. When the specified time has been exceeded, all available data is sent to the Failure terminal.

3. On the **Advanced** tab, set the properties that determine how the data stream is controlled.
   - Use the Close connection property to specify when and how to close the connection.
     - Select No to leave the connection open. This value is the default.
     - Select After timeout to close the connection when a timeout occurs.
     - Select After data has been received to close the connection when the end of the record is found.
     - Select At end of flow to close the connection after the flow has been run.
   - Select Close input stream after a record has been received to close the input stream as soon as the data has been retrieved. When the connection input stream is reserved, no other node can use it without specifying the ID. This property is not selected by default.
   - Use the Input Stream Modification property to specify whether to reserve the input stream for use only by input and receive nodes that specify the connection ID, and, if reserved, whether to release the input stream at the end of the flow. These options are available only if you have not selected the Close input stream after a record has been received property.
     - Select Leave unchanged to leave the input stream as it was when it entered the node. This value is selected by default.
     - Select Reserve input stream (for use by future TCPIP input and receive nodes) to specify that this input stream can be used only by this node and by other receive nodes that request it by specifying the connection ID. When the connection input stream is reserved, no other nodes can use it without specifying the correct connection ID.
     - Select Reserve input stream (for use by future TCPIP input and receive nodes) then release at end of flow to specify that this input stream can be used only by this node and receive nodes that request it by specifying the connection ID. After the flow has been run, this input stream is returned to the pool and becomes available for use by any input or receive node.
   - Use the Output Stream Modification property to specify whether to release the output stream.
     - Select Leave unchanged to leave the output stream as it was when it entered the node. This value is selected by default.
     - Select Release output stream and reset ReplyID to specify that this output stream is returned to the pool and is available for use by any output node. The ReplyID is passed in the local environment when leaving this node, but is reset for the next record on this connection.

4. On the **Input Message Parsing** tab, set values for the properties that the node uses to determine how to parse the incoming message.

   If the incoming message has an MQRFH2 header, you do not have to set values for the Input Message Parsing properties because the values are derived from the <mcd> folder in the MQRFH2 header; for example:

   ```
   <mcd><Msd>MRM</Msd><Set>DHM4UO906S001</Set><Type>receiptmsg1</Type>
   <Fmt>XML</Fmt></mcd>
   ```

   If you set values, and those values differ from those in the MQRFH2 header, the values in the MQRFH2 header take precedence.

   - In Message domain, select the name of the parser that you are using from the list. The default is BLOB. You can choose from the following options:
     - XMLNSC
     - DataObject
     - JSON
     - BLOB
     - MIME
     - MRM
     - JMSMap
     - JMSStream
     - XMLNS

     You can also specify a user-defined parser, if appropriate.
   - If you are using the MRM or IDOC parser, or the XMLNSC parser in validating mode, select the Message set that you want to use. This list is populated with available message sets when you select MRM, XMLNSC, or IDOC as the domain.
   - If you are using the MRM parser, select the correct message type from the list in Message type. This list is populated with available message types when you select the MRM parser.
   - If you are using the MRM or IDOC parser, select the correct message format from the list in Message format. This list is populated with available message formats when you select the MRM or IDOC parser.
   - Specify the message coded character set ID in Message coded character set ID.
   - Select the message encoding from the list in Message encoding or specify a numeric encoding value. For more information about encoding, see "Data conversion" on page 1151.
5. On the **Parser Options** subtab:
   a. Parse timing is, by default, set to On Demand, which causes parsing of the message to be delayed. To cause the message to be parsed immediately, see "Parsing on demand" on page 4173.
   b. If you are using the XMLNSC parser, set values for the properties that determine how the XMLNSC parser operates. For more information, see "Manipulating messages in the XMLNSC domain" on page 2546.
6. Use the **Retry** tab to define how retry processing is performed when a flow fails. You can set the following properties:
   - Retry mechanism determines the action that occurs if the flow fails. Choose from the following values:
     - Select Failure for the node to report a failure without any retry attempts.
     - Select Short retry for the node to retry before reporting a failure if the condition persists. The number of times that it retries is specified in Retry threshold.

- Select Short retry and long retry for the node to retry, first using the value in Retry threshold as the number of attempts it should make. If the condition persists after the Retry threshold has been reached, the node then uses the Long retry interval between attempts.

- Specify the Retry threshold:The number of times the node retries the flow transaction if the Retry mechanism property is set to either Short retry or Short retry and long retry.
- Specify the Short retry interval:The length of time, in seconds, to wait between short retry attempts.
- Specify the Long retry interval:The length of time to wait between long retry attempts until a message is successful, the message flow is stopped, or the message flow is redeployed. The broker property **MinLongRetryInterval** defines the minimum value that the Long retry interval can take. If the value is lower than the minimum, the broker value is used.

7. Use the **Records and Elements** tab to specify how the data is interpreted as records.
   - Use the Record detection property to determine how the data is split into records, each of which generates a single message. Choose from the following options:
     - End of stream specifies that all of the data sent in the data stream is a single record.
     - Fixed Length specifies that each record is a fixed number of bytes in length. Each record contains the number of bytes specified in the Length property.
     - Select Delimited if the records that you are processing are separated, or ended, by a DOS or UNIX line end or by a sequence of user-defined delimiter bytes. Specify the delimiter and delimiter type in the Delimiter and Delimiter type properties.
     - Select Parsed Record Sequence if the data contains a sequence of one or more records that are serially recognized by the parser specified in Message domain. The node propagates each recognized record as a separate message. If you select the Record detection option, the parser specified in Message domain must be either XMLNSC or MRM (either CWF or TDS physical format).
   - If you set Record detection to Fixed Length, use Length to specify the required length of the output record. This value must be in the range 1 byte through 100 MB. The default is 80 bytes.

     If you set Record detection to End of stream, Fixed Length, or Delimited, a limit of 100 MB applies to the length of the records. If you set Record detection toParsed Record Sequence, the TCPIPServerInput node does not determine or limit the length of a record. Nodes that are downstream in the message flow might try to determine the record length or process a long record. If you intend to process large records in this way, ensure that your broker has sufficient memory. You might have to apply flow techniques described in the Large Messaging sample to best use the available memory.
   - If you set Record detection to Delimited, use Delimiter to specify the delimiter to be used. Choose from the following values:
     - DOS or UNIX Line End, which, on UNIX systems, specifies the line feed character (<LF>, X'0A'), and, on Windows systems, specifies a carriage return character followed by a line feed character (<CR><LF>, X'0D0A'). The node treats both of these strings as delimiters, irrespective of the system on which the broker is running. If both strings are seen in the same record, the node recognizes both as delimiters. The node does not

recognize X'15', which, on z/OS systems, is the 'newline' byte; specify a value of `Custom Delimiter` in this property and a value of `15` in the `Custom delimiter` property if your input data is coded using EBCDIC new lines.

– `Custom Delimiter`, which permits a sequence of bytes to be specified in `Custom delimiter`

- In `Custom delimiter`, specify the delimiter byte or bytes to be used when `Delimiter` is set to `Custom delimiter`. Specify this value as an even-numbered string of hexadecimal digits. The default is X'0A' and the maximum length of the string is 16 bytes (represented by 32 hexadecimal digits).
- If you set `Record detection` to `Delimited`, use `Delimiter type` to specify the type of delimiter. Permitted values are:
  – `Infix`. If you select this value, each delimiter separates records. If the data ends with a delimiter, the (zero length) data that follows the final delimiter is still propagated although it contains no data.
  – `Postfix`. If you specify this value, each delimiter ends records. If the data ends with a delimiter, no empty record is propagated after the delimiter. If the data does not end with a delimiter, it is processed as if a delimiter follows the final bytes of the data. `Postfix` is the default value.
- The TCPIPServerInput node considers each occurrence of the delimiter in the input as either separating (infix) or terminating (postfix) each record. If the data begins with a delimiter, the node treats the (zero length) contents preceding that delimiter as a record and propagates an empty record to the flow. The delimiter is never included in the propagated message.

8. Use the **Validation** tab to provide validation based on the message set for predefined messages. For more information about validation, see "Validating messages" on page 1478. For information about how to provide validation for this tab, see "Validation tab properties" on page 4169.
9. On the **Transactions** tab, set the transaction mode. Although TCPIP operations are non-transactional, the transaction mode on this input node determines whether the rest of the nodes in the flow are to be run under point of consistency or not. Select `Yes` if you want the flow updates to be treated transactionally (if possible) or `No` if you do not. The default for this property is `No`.
10. Optional: On the **Instances** tab, set values for the properties that determine the additional instances (threads) that are available for a node. For more details, see "Configurable message flow properties" on page 4020.

Terminals and properties:
The terminals of the TCPIPServerInput node are described in the following table.

Terminal	Description
Failure	The output terminal to which the message is routed if an error occurs. These errors include failures caused by retry processing. Even if the Validation property is set, messages propagated to this terminal are not validated.
Open	The output terminal to which a message is routed when it is first opened. Use the Open terminal if processing is required when a connection is opened rather than when data first arrives.
Out	The output terminal to which the message is routed if it is successfully retrieved from an external resource. If no errors occur within the input node, a message that is received from an external resource is always sent to the Out terminal first.
Close	The output terminal to which the message is routed if the connection closes.

Terminal	Description
Catch	The output terminal to which the message is routed if an exception is thrown downstream and caught by this node. Exceptions are caught only if this terminal is attached.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Description properties of the TCPIPServerInput node are described in the following table.

Property	M	C	Default	Description
Node name	No	No	TCPIPServerInput	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The Basic properties of the TCPIPServerInput node are described in the following table.

Property	M	C	Default	Description	**mqsiapplybaroverride command property**
Connection details	Yes	Yes		A string containing the port number to be used, or the name of a configurable service.	connectionDetails
Timeout waiting for a data record (seconds)	Yes	Yes	`60`	Specifies how long the node listens on a connection for more data after the first byte of data has arrived. You can specify any length of time in seconds.	timeoutWaitingForData

The Advanced properties of the TCPIPServerInput node are described in the following table.

Property	M	C	Default	Description
Close connection	Yes	No	`No`	Controls when the connection is closed, or if it remains open. Valid options are: • `No` • `After Timeout` • `After Data has been Received` • `At End of Flow`
Close input stream after a record has been received	Yes	No	Cleared	Specifies whether to close the input stream as soon as the data has been retrieved. When the connection input stream is reserved, no other node can use it without knowing the ID. By default, this option is not selected.

Property	M	C	Default	Description
Input Stream Modification	No	No	`Leave unchanged`	Specifies whether to reserve the input stream for use only by input and receive nodes that specify the connection ID, and, if reserved, whether to release the input stream at the end of the flow. Valid options are: • `Leave unchanged` • `Reserve input stream (for use by future TCPIP nodes)` • `Reserve input stream (for use by future TCPIP nodes) then release at end of flow` When the connection input stream is reserved, no other nodes can use it without specifying the correct connection ID. If the input stream is released at the end of the flow, it is returned to the pool and becomes available for use by any input or receive node.
Output stream modification	No	No	`Leave unchanged`	Specifies whether this output stream is released and returned to the pool for use by any output node. Valid options are: • `Leave unchanged` • `Release output stream and reset ReplyID` If you select `Release output stream and reset ReplyID`, the ReplyID is passed in the local environment when leaving this node, but is reset for the next record on this connection.

The TCPIPServerInput node Input Message Parsing properties are described in the following table.

Property	M	C	Default	Description	**`mqsiapplybaroverride` command property**
Message domain	No	No		The domain that is used to parse the incoming message.	
Message set	No	No		The name or identifier of the message set in which the incoming message is defined. If you set this property, then later update the project dependencies to remove this message set reference, a warning is issued. Either update the `Message Set` property, or restore the reference to this message set project.	
Message type	No	No		The name of the incoming message.	
Message format	No	No		The name of the physical format of the incoming message.	
Message coded character set ID	Yes	No	`Broker System Default`	The ID of the coded character set used to interpret the data being read.	messageCodedCharSetIdProperty
Message encoding	Yes	No	`Broker System Determined`	The encoding scheme for numbers and large characters used to interpret the data being read. Valid values are `Broker System Determined` or a numeric encoding value. For more information about encoding, see "Data conversion" on page 1151.	messageEncodingProperty

The Parser Options properties of the TCPIPServerInput node are described in the following table.

Property	M	C	Default	Description
Parse timing	No	No	`On Demand`	This property controls when an input message is parsed. Valid values are: • `On Demand` • `Immediate` • `Complete` For a full description of this property, see "Parsing on demand" on page 4173.
Build tree using XML schema data types	No	No	Cleared	This property controls whether the syntax elements in the message tree have data types taken from the XML Schema.
Use XMLNSC compact parser for XMLNS domain	No	No	Cleared	This property controls whether the XMLNSC Compact Parser is used for messages in the XMLNS Domain. If you set this property, the message data is displayed under XMLNSC in nodes that are connected to the output terminal when the input MQRFH2 header or Input Message Parsing property, `Message Domain`, is XMLNS.
Retain mixed content	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters mixed text in an input message. If you select the check box, elements are created for mixed text. If you clear the check box, mixed text is ignored and no elements are created.
Retain comments	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters comments in an input message. If you select the check box, elements are created for comments. If you clear the check box, comments are ignored and no elements are created.
Retain processing instructions	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters processing instructions in an input message. If you select the check box, elements are created for processing instructions. If you clear the check box, processing instructions are ignored and no elements are created.
Opaque elements	No	No	Blank	This property is used to specify a list of elements in the input message that are to be opaquely parsed by the XMLNSC parser.

The Records and Elements properties of the TCPIPServerInput node are described in the following table:

Property	M	C	Default	Description
Record detection	Yes	No	`End of Stream`	The mechanism used to identify records in the input data. Valid options are: • `End of Stream` • `Fixed Length` • `Delimited` • `Parsed Record Sequence`
Length (bytes)	Yes	No	`0`	The length of each record, in bytes, when `Fixed Length` record detection is selected.

Property	M	C	Default	Description
Delimiter	Yes	No	DOS or UNIX Line End	The type of delimiter bytes that separates, or ends, each record when Delimited record detection is selected. Valid options are: • DOS or UNIX Line End • Custom Delimiter (Hexadecimal)
Custom delimiter (hexadecimal)	No	No		The delimiter bytes, expressed in hexadecimal, when Delimited record detection and Custom Delimiter (Hexadecimal) are selected. This property is mandatory only if the Delimiter property is set to Custom Delimiter (Hexadecimal).
Delimiter type	Yes	No	Postfix	The location of the delimiter when Delimited record detection and Custom Delimiter (Hexadecimal) are selected. Valid options are: • Infix • Postfix This property is ignored unless the Delimiter property is set to Custom Delimiter (Hexadecimal).

The Retry properties of the TCPIPServerInput node are described in the following table:

Property	M	C	Default	Description	mqsiapplybaroverride command property
Retry mechanism	Yes	No	Failure	How the node handles a flow failure. Valid options are: • Failure • Short Retry • Short and Long Retry	
Retry threshold	Yes	Yes	0	The number of times to retry the flow transaction when Retry mechanism is Short retry.	retryThreshold
Short retry interval (seconds)	No	Yes	0	The interval, in seconds, between each retry if Retry threshold is not zero.	shortRetryThreshold
Long retry interval (seconds)	No	Yes	300	The interval between retries if Retry mechanism is Short and Long Retry and the retry threshold has been exhausted.	longRetryThreshold

The Validation properties of the TCPIPServerInput node are described in the following table.

For a full description of these properties, see "Validation properties" on page 4169.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Validate	No	Yes	None	This property controls whether validation takes place. Valid values are • None • Content and Value • Content	validateMaster

Property	M	C	Default	Description	mqsiapplybaroverride command property
Failure action	No	No	Exception	This property controls what happens if validation fails. Valid values are: • User Trace • Local Error Log • Exception • Exception List	

The Transactions properties of the TCPIPServerInput node are described in the following table:

Property	M	C	Default	Description
Transaction mode	No	Yes	No	The transaction mode on this input node determines whether the rest of the nodes in the flow are executed under point of consistency. Valid options are: • Yes • No

The Instances properties of the TCPIPServerInput node are described in the following table. For a full description of these properties, see "Configurable message flow properties" on page 4020.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Additional instances pool	No	Yes	Use Pool Associated with Message Flow	The pool from which additional instances are obtained. • If you select Use Pool Associated with Message Flow, additional instances are obtained from the message flow pool. • If you select Use Pool Associated with Node, additional instances are allocated from the additional instances of the node, based on the number specified in the Additional instances property.	componentLevel
Additional instances	No	Yes	0	The number of additional instances that the node can start if the Additional instances pool property is set to Use Pool Associated with Node.	additionalInstances

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"WebSphere Broker TCP/IP Transport" on page 1735
The WebSphere Broker TCP/IP Transport is a service that connects applications

that use raw TCP/IP sockets for transferring data.

"TCP/IP nodes" on page 1738
WebSphere Message Broker implements access to the TCP/IP input and output streams through a series of nodes.

"Scenarios for WebSphere Message Broker and TCP/IP" on page 1746
Two example scenarios show how you might use TCP/IP and WebSphere Message Broker as part of a business solution.

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

"Parsers" on page 1072
A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from the internal message tree representation.

"Data conversion" on page 1151
Convert data that your message flows are transferring between different environments by using WebSphere MQ or WebSphere Message Broker facilities.

"Samples" on page 98
The WebSphere Message Broker Toolkit provides samples that show the features that are available in WebSphere Message Broker, and how to use them. This topic provides links to the information about the individual samples.

Related tasks:

"Working with TCP/IP" on page 1750
You can use WebSphere Message Broker TCPIP nodes and TCP/IP configurable services to perform various tasks.

"Using more than one input node" on page 1473
You can include more than one input node in a single message flow.

"Validating messages" on page 1478
The broker provides validation based on the message model for predefined messages.

"Manipulating messages in the XMLNSC domain" on page 2546
If you are writing ESQL to process messages in the XMLNSC domain, it is helpful to learn about the structure of the message tree that the XMLNSC parser builds.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

Related reference:

"Parsing on demand" on page 4173
On-demand parsing, referred to as partial parsing, is used to parse an input message bit stream only as far as is necessary to satisfy the current reference. The parsers that can perform partial parsing of input messages are the JSON, MRM, XML, XMLNS, and XMLNSC parsers.

"Configurable message flow properties" on page 4020
When you add a message flow to a broker archive (BAR) file in preparation for deploying it to a broker, you can set additional properties that influence its run time operation. These properties are available for review and update when you

select the **Manage and Configure** tab for the broker archive file.

"Input node" on page 4511
Use the Input node as an In terminal for an embedded message flow (a subflow).

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"TCPIPClientInput node" on page 4854
Use the TCPIPClientInput node to create a client connection to a raw TCP/IP socket, and to receive data over that connection.

"TCPIPClientOutput node" on page 4867
Use the TCPIPClientOutput node to create a client connection to a raw TCP/IP socket, and to send data over that connection to an external application.

"TCPIPServerOutput node"
Use the TCPIPServerOutput node to create a server connection to a raw TCP/IP socket, and to send data over the connection to an external application.

"Validation properties" on page 4169
You can control validation by setting properties on the Validate and Parser Options tabs for the nodes that are listed in the following table.

TCPIPServerOutput node

Use the TCPIPServerOutput node to create a server connection to a raw TCP/IP socket, and to send data over the connection to an external application.

This topic contains the following sections:
- "Purpose"
- "Using the TCPIPServerOutput node in a message flow" on page 4905
- "Configuring the TCPIPServerOutput node" on page 4905
- "Terminals and properties" on page 4908

Purpose:
The TCPIPServerOutput node listens on a TCP/IP port and waits for a client node to make a connection with the port. When the client node connects to the port, the server node creates a connection for the client. The connections are not made directly by the node but are obtained from a connection pool managed by the WebSphere Message Broker execution group.

The execution group uses the default TCPIPServer configurable service to determine which attributes are used for the socket connection. However, if the configurable service is set on the node, the configurable service is used for all the properties, including the host and port number.

When the connection has been established, the data is sent. If the data is not sent successfully within the time limit specified by the node's `Timeout sending a data record` property, an exception is thrown.

You can configure the broker to use SSL for TCP/IP nodes; see "SSL and the TCP/IP nodes" on page 551.

Properties in the local environment can override the TCP/IP connection used by the node.

Table 259. Input local environment properties

Location in local environment	Description
$LocalEnvironment/Destination/TCPIP/Output/Hostname	The host name used to make a connection.
$LocalEnvironment/Destination/TCPIP/Output/Port	The port number used to make a connection.
$LocalEnvironment/Destination/TCPIP/Output/Id	The ID of the socket being used. This ID is an internal identifier used by WebSphere Message Broker to uniquely identify a connection.
$LocalEnvironment/Destination/TCPIP/Output/ReplyId	The Reply ID that is stored on this connection. It can be any text string.
$LocalEnvironment/Destination/TCPIP/Output/Timeout	The timeout value used when sending data to the TCP/IP server connection. This value overrides the `Timeout sending a data record` property specified on the node.

You can dynamically select the connection details (host name and port number), and the connection used (ID), using these properties. The Reply ID can also be set on the connection, which enables a string to be stored in the connection and to be displayed in the local environment. This behavior can be used to store Reply IDs from other TCPIP nodes or from other transports such as WebSphere MQ.

The output of the node contains the same information as the input, and any fields that are missing from the input are updated with details from the connection used. For example, if the `Id` property is not provided as input (because you want to create a connection or reuse a pool connection), the output local environment contains the ID of the connection that is used.

Table 260. Output local environment properties

Location in local environment	Description
$LocalEnvironment/WrittenDestination/TCPIP/Output/ConnectionDetails/Hostname	The host name used to make a connection.
$LocalEnvironment/WrittenDestination/TCPIP/Output/ConnectionDetails/Port	The port number used to make a connection.
$LocalEnvironment/WrittenDestination/TCPIP/Output/ConnectionDetails/OpenTimestamp	The time stamp when the connection was first opened.
$LocalEnvironment/WrittenDestination/TCPIP/Output/ConnectionDetails/CloseTimestamp	The time stamp when the connection was closed (null if not yet closed).
$LocalEnvironment/WrittenDestination/TCPIP/Output/ConnectionDetails/SequenceNumber	The sequence number of the message received on this connection. The first record has a sequencing number 1, the second record has a sequencing number 2, and so on.
$LocalEnvironment/WrittenDestination/TCPIP/Output/ConnectionDetails/Id	The ID of the socket being used. This ID is an internal identifier used by WebSphere Message Broker to uniquely identify a connection.
$LocalEnvironment/WrittenDestination/TCPIP/Output/ConnectionDetails/ReplyId	The Reply ID that is stored on this connection. It can be any text string.
$LocalEnvironment/WrittenDestination/TCPIP/Output/ConnectionDetails/ClientDetails/Hostname	The fully qualified domain name of the computer from which the client connected.
$LocalEnvironment/WrittenDestination/TCPIP/Output/ConnectionDetails/ClientDetails/Address	The IP address of the computer from which the client connected.

If the connection closes (or any other type of exception occurs) while using the TCP/IP transport, an exception is thrown. This exception goes to the Failure terminal if it is connected, otherwise the exception goes back down the message flow.

The node also has a Close input terminal. If a message is sent to this terminal, the connection is closed using a combination of the details provided in the node and the local environment.

The TCPIPServerOutput node is contained in the **TCPIP** drawer of the palette and is represented in the workbench by the following icon:

Using the TCPIPServerOutput node in a message flow:
You can use the TCPIPServerOutput node in any message flow that must send data to an external application. Look at the following samples to see how to use it:

- TCPIP Client Nodes
- TCPIP Handshake

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Configuring the TCPIPServerOutput node:
When you have put an instance of the TCPIPServerOutput node into a message flow, you can configure it (for more information, see "Configuring a message flow node" on page 1503). The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk in that view.

To configure the TCPIPServerOutput node:

1. Optional: On the **Description** tab, enter a short description, a long description, or both. You can also rename the node on this tab.
2. On the **Basic** tab, set the properties that determine how the TCP/IP connection is controlled.
   - Use the `Connection details` property to specify the port number to be used, or the name of a configurable service. This property is mandatory. The following formats are supported:
     - `Configurable service name`. This value is used to look up the port number in configurable services; for example, `TCPIPProfile1`.
     - `<Port>`. This value is the port number; for example, `1111`
     - `<Port>`. This value is the port number. In this case the host name is assumed to be `localhost`.
   - Use the `Timeout sending a data record (seconds)` property to specify how long the node waits when trying to send data. You can specify any length of time in seconds. When the specified time has been exceeded, all available data is sent to the Failure terminal. The default is `60` seconds.
3. On the **Advanced** tab, set the properties that determine how the data stream is controlled.

- Use the Send to property to specify whether the data is to be sent to one connection or to all available connections.
  - Select One connection to send the message to only one connection, as specified by the node properties and message overrides. This value is the default.
  - Select All available connections to send the data to all available connections.
- Use the Close connection property to specify when and how to close the connection.
  - Select No to leave the connection open. This value is the default.
  - Select After timeout to close the connection when a timeout occurs.
  - Select After data has been sent to close the connection when the end of the record has been sent.
- Select Close output stream after a record has been sent to close the output stream as soon as the data has been sent. By default, this property is not selected.
- Use the Output Stream Modification property to specify whether to reserve or release the output stream. These options are available only if you have not selected the Close output stream after a record has been sent property.
  - Select Leave unchanged to leave the output stream as it was when it entered the node. This value is selected by default.
  - Select Release output stream to specify that this output stream is returned to the pool and is available for use by any output node.
  - Select Reserve output stream (for use by future TCPIP output nodes) to specify that this output stream can be used only by this node and by other output nodes that request it by specifying the connection ID. When the connection input stream is reserved, no other nodes can use it without specifying the correct connection ID.
  - Select Reserve output stream (for use by future TCPIP output nodes) then release after propagate to specify that this output stream can be used only by this node and output nodes that request it by specifying the correct connection ID. After the message has been propagated, this output stream is returned to the pool and becomes available for use by any output node.
- Use the Input Stream Modification property to reserve the input stream for use only by input and receive nodes that specify the connection ID, or to release the input stream at the end of the message flow.
  - Select Leave unchanged to leave the input stream as it was when it entered the node. This value is selected by default.
  - Select Release input stream to specify that this input stream is returned to the pool and is available for use by any input or receive node.
  - Select Reserve input stream (for use by future TCPIP input and receive nodes) to specify that this input stream can be used only by this node and by other input or receive nodes that request it by specifying the connection ID. When the connection input stream is reserved, no other nodes can use it without specifying the correct connection ID.
  - Select Reserve input stream (for use by future TCPIP input and receive nodes) then release after propagate to specify that this input stream can be used only by this node and receive nodes that request it by specifying the correct connection ID. After the message has been propagated, this input stream is returned to the pool and becomes available for use by any input or receive node.

4. On the **Request** tab, specify the location of the data to be written. You can specify the properties on this tab as XPath or ESQL expressions. Content Assist is available in the Properties view and also in the XPath Expression Builder, which you can call by using the **Edit** button to the right of each property.
   a. In `Data location`, specify the input data location, which is the location in the input message tree that contains the record to be written. The default value is `$Body`, which is the entire message body (`$InputRoot.Body`).

      When you specify this property, if the data in the message tree that it identifies is owned by a model-driven parser (such as the MRM parser or XMLNSC parser,) be aware of the following considerations:

      - If you are using MRM CWF format, ensure that the identified message tree exists as a message definition. If this message tree is defined as a global element only, exceptions BIP5180 and BIP5167 are generated.
      - If you are using MRM TDS format, the serialization of the identified message is successful if the element is defined as a global element or message. However, if the identified field is not found as a global element or message, note that:
        – If this field is a leaf field in the message tree, the field is written as self-defining. No validation occurs even if validation is enabled.
        – If this field is a complex element, an internal exception is generated, BIP5522, indicating that the logical type cannot be converted to a string.
      - If you are using MRM XML, the events are similar to the MRM TDS format except that, if the field is a complex element, it is written as self-defining.
      - If you use the XMLNSC parser, no validation occurs, even if validation is enabled.

   b. In `Port location`, specify the location of the value to override the `Port` that is set in the `Connection details` property of the **Basic** tab. If you do not specify a location, the default value is `$LocalEnvironment/Destination/TCPIP/Output/Port`.
   c. In `ID location`, specify the location of the `Id` of the socket being used. This internal identifier is used by WebSphere Message Broker to uniquely identify a connection. If you do not specify a location, the default value is `$LocalEnvironment/Destination/TCPIP/Output/Id`.
   d. In `Reply ID location`, specify the location of the Reply ID that is stored on the connection that is being used. The Reply ID can be used when data is returned in an input node. If you do not specify a location, the default value is `$LocalEnvironment/Destination/TCPIP/Output/ReplyId`.
5. Use the **Records and Elements** tab to specify how the TCPIPServerOutput node writes the record that is derived from the message.
   - In `Record definition`, choose from the following values:
     – `Record is Unmodified Data` specifies that records are left unchanged. This value is the default.
     – `Record is Fixed Length Data` specifies that records are padded to a specified length if necessary. You specify this length in the `Length` property. If the record is longer than the value specified in `Length`, the node generates an exception. Use the `Padding byte` property to specify the byte to be used for padding the message to the required length.
     – `Record is Delimited Data` specifies that records are separated by a delimiter and accumulated by concatenation. The delimiter is specified by

the Delimiter, Custom delimiter, and Delimiter type properties. The file is finished only when a message is received on the Finish File terminal.

- In Length, specify the length (in bytes) of records when Record definition is set to Record is Fixed Length Data. Records longer than this value cause an exception to be thrown. This value must be in the range 1 byte through 100 MB. The default is 80 bytes.
- When Record definition is set to Record is Fixed Length Data, use Padding byte to specify the byte to be used when padding records to the specified length if they are shorter than this length. Specify this value as two hexadecimal digits. The default value is X'20'.
- In Delimiter, specify the delimiter to be used if you set Record definition to Record is Delimited Data. Choose from:
  - Broker System Line End specifies that a line end sequence of bytes is used as the delimiter as appropriate for the file system on which the broker is running. This value is the default. For example, on Windows systems, this line end is a 'carriage-return, line-feed' pair (X'0D0A'); on UNIX systems, it is a single 'line-feed' byte (X'0A'); on z/OS systems, it is a 'newline' byte (X'15').
  - Custom Delimiter specifies that the explicit delimiter sequence defined in the Custom delimiter property is to be used to delimit records.
- In Custom delimiter, specify the delimiter sequence of bytes to be used to delimit records when Delimiter is set to Custom Delimiter. Specify this value as an even-numbered string of hexadecimal digits. The default is X'0A' and the maximum length of the string is 16 bytes.
- If you set Record definition to Record is Delimited Data, use Delimiter type to specify how the delimiter is to separate records. Choose from the following values:
  - Postfix specifies that the delimiter is added after each record that is written. This value is the default.
  - Infix specifies that the delimiter is inserted between any two adjacent records only.

6. On the **Validation** tab, specify the parser validation properties of the node. For more information about validation, see "Validating messages" on page 1478. For information about how to provide validation for this tab, see "Validation tab properties" on page 4169.

Terminals and properties:
The TCPIPServerOutput node terminals are described in the following table.

Terminal	Type	Description
In	Input data	The input terminal that accepts a message for processing by the node.
Close	Input control	The input terminal to which a message is routed when the connection given in the local environment is closed.
Out	Output data	The output terminal to which the message is routed if it is successfully sent to an external resource. The message received on the In terminal is propagated to the Out terminal and is left unchanged except for the addition of status information.
Close	Output control	The output terminal to which a message propagated from the Close input terminal is routed.
Failure	Output data	The output terminal to which the message is routed if a failure is detected in the node.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file for deployment).

The Description properties of the TCPIPServerOutput node are described in the following table:

Property	M	C	Default	Description
Node name	No	No	TCPIPServerOutput	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The Basic properties of the TCPIPServerOutput node are described in the following table:

Property	M	C	Default	Description	**mqsiapplybaroverride command property**
Connection details	Yes	Yes		A string containing the port number to be used, or the name of a configurable service.	connectionDetails
Timeout sending a data record (seconds)	Yes	Yes	`60`	Specifies how long the node waits when trying to send data. You can specify any length of time in seconds.	timeoutSendingData

The Advanced properties of the TCPIPServerOutput node are described in the following table.

Property	M	C	Default	Description
Close connection	Yes	No	`No`	Controls when the connection is closed, or if it remains open. Valid options are: • `No` • `After Timeout` • `After Data has been Sent`
Close output stream after a record has been sent	Yes	No	Cleared	Specifies whether to close the output stream as soon as the data has been sent. By default, this property is not selected.
Output Stream Modification	No	No	`Leave unchanged`	Specifies whether to reserve this output stream or release it and return it to the pool for use by any output node. Valid options are: • `Leave unchanged` • `Release output stream` • `Reserve output stream (for use by future TCPIP nodes)` • `Reserve output stream (for use by future TCPIP nodes) then release at end of flow`

Property	M	C	Default	Description
Input Stream Modification	No	No	`Leave unchanged`	Specifies whether to reserve the input stream for use only by input and receive nodes that specify the connection ID, or to release the input stream at the end of the flow. Valid options are: • `Leave unchanged` • `Release input stream` • `Reserve input stream (for use by future TCPIP nodes)` • `Reserve input stream (for use by future TCPIP nodes) then release at end of flow` When the connection input stream is reserved, no other nodes can use it without specifying the correct connection ID. If the input stream is released after the message has been propagated, it is returned to the pool and becomes available for use by any input or receive node.
Send to:	Yes	No	`One connection`	Specifies whether the data is to be sent to one connection or to all available connections. Valid options are: • One Connection • All Available Connections

The Request properties of the TCPIPServerOutput node are described in the following table:

Property	M	C	Default	Description
Data location	Yes	No	`$Body`	The location in the input message tree that contains the record to be written.
Port location	Yes	No	`$LocalEnvironment/Destination/TCPIP/Output/Port`	The message element location that contains the port.
ID	Yes	No	`$LocalEnvironment/Destination/TCPIP/Output/Id`	The message element location that contains the ID.
Reply ID location	Yes	No	`$LocalEnvironment/Destination/TCPIP/Output/ReplyId`	The message element location that contains the Reply ID.

The Records and Elements properties of the TCPIPServerOutput node are described in the following table:

Property	M	C	Default	Description
Record definition	Yes	No	`Record is Unmodified Data`	This property controls how the records derived from the message are written. Valid options are: • `Record is Unmodified Data` • `Record is Fixed Length Data` • `Record is Delimited Data`
Length (bytes)	Yes	No	`0`	The required length of the output record. This property applies only when `Record definition` is set to `Record is Fixed Length Data`.
Padding byte (hexadecimal)	Yes	No	`20`	The two-digit hexadecimal byte to be used to pad short messages when `Record definition` is set to `Record is Fixed Length Data`.

Property	M	C	Default	Description
Delimiter	Yes	No	Broker System Line End	The delimiter to be used when Record definition is set to Record is Delimited Data. Valid options are: • Broker System Line End • Custom Delimiter (Hexadecimal)
Custom delimiter (hexadecimal)	No	No	None	The delimiter byte sequence to be used when Record definition is set to Record is Delimited Data and Delimiter is set to Custom Delimiter (Hexadecimal).
Delimiter type	Yes	No	Postfix	This property specifies the way in which the delimiters are inserted between records when Record definition is set to Record is Delimited Data. Valid options are: • Infix • Postfix

The Validation properties of the TCPIPServerOutput node are described in the following table.

For a full description of these properties, see "Validation properties" on page 4169.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Validate	No	Yes	Inherit	This property controls whether validation takes place. Valid values are: • None • Content and Value • Content • Inherit	validateMaster
Failure action	No	No	Exception	This property controls what happens if validation fails. You can set this property only if you set Validate to Content or Content and Value. Valid values are: • User Trace • Local Error Log • Exception • Exception List	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"WebSphere Broker TCP/IP Transport" on page 1735
The WebSphere Broker TCP/IP Transport is a service that connects applications that use raw TCP/IP sockets for transferring data.

"TCP/IP nodes" on page 1738
WebSphere Message Broker implements access to the TCP/IP input and output streams through a series of nodes.

"Scenarios for WebSphere Message Broker and TCP/IP" on page 1746
Two example scenarios show how you might use TCP/IP and WebSphere Message Broker as part of a business solution.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message flow aggregation" on page 2718
Aggregation is the generation and fan-out of related requests that are derived from a single input message, and the fan-in of the corresponding replies to produce a single aggregated reply message.

Related tasks:

"Working with TCP/IP" on page 1750
You can use WebSphere Message Broker TCPIP nodes and TCP/IP configurable services to perform various tasks.

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Configuring aggregation flows" on page 2721
Use aggregation message flows to generate and fan-out a number of related requests, fan-in the corresponding replies, and compile those replies into a single aggregated reply message, by using the AggregateControl, AggregateRequest, and AggregateReply nodes.

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

"Validating messages" on page 1478
The broker provides validation based on the message model for predefined messages.

Related reference:

"Validation properties" on page 4169
You can control validation by setting properties on the Validate and Parser Options tabs for the nodes that are listed in the following table.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"TCPIPClientInput node" on page 4854
Use the TCPIPClientInput node to create a client connection to a raw TCP/IP socket, and to receive data over that connection.

"TCPIPClientOutput node" on page 4867
Use the TCPIPClientOutput node to create a client connection to a raw TCP/IP socket, and to send data over that connection to an external application.

"TCPIPServerInput node" on page 4890
Use the TCPIPServerInput node to create a server connection to a raw TCPIP socket, and to receive data over that connection.

"TCPIPClientReceive node" on page 4877
Use the TCPIPClientReceive node to receive data over a client TCP/IP connection.

"TCPIPServerReceive node"
Use the TCPIPServerReceive node to receive data over a server TCP/IP connection.

TCPIPServerReceive node

Use the TCPIPServerReceive node to receive data over a server TCP/IP connection.

This topic contains the following sections:
- "Purpose"
- "Using the TCPIPServerReceive node in a message flow" on page 4915
- "Configuring the TCPIPServerReceive node" on page 4915
- "Terminals and properties" on page 4919

Purpose:
The TCPIPServerReceive node waits for data to be received on a TCP/IP connection, and retrieves the data. If the connection is closed, an exception is thrown.

When a connection is established, the data is sent to the TCPIPServerReceive node. If the TCPIPServerReceive node fails to receive all the data within the time specified in the `Timeout waiting for a data record` property, the message is sent to the Timeout terminal; if no Timeout terminal is connected, an exception is thrown.

You can configure the broker to use SSL for TCP/IP nodes; see "SSL and the TCP/IP nodes" on page 551.

Properties in the local environment can override the TCP/IP connection used by the node.

Table 261. Input local environment properties

Location in local environment for input to node	Description
$LocalEnvironment//TCPIP/Receive/Hostname	The host name used to make a connection.
$LocalEnvironment//TCPIP/Receive/Port	The port number used to make a connection.
$LocalEnvironment/TCPIP/Receive/Id	The ID of the socket being used. This ID is an internal identifier used by WebSphere Message Broker to uniquely identify a connection.
$LocalEnvironment/TCPIP/Receive/ReplyId	The Reply ID to be stored on this connection. This ID can then be used when data is returned on an input node. The Reply ID can be any text string.
$LocalEnvironment/TCPIP/Receive/Timeout	The timeout value used when waiting for data on the TCP/IP server connection. This value overrides the `Timeout waiting for a data record` property specified on the node.

Table 261. Input local environment properties (continued)

Location in local environment for input to node	Description
$LocalEnvironment/TCPIP/Receive/Length	The value used to override the number of bytes to be read when using fixed size records. This value overrides the `Length (bytes)` property specified on the node. If the `Record detection` property is set to anything other than `Fixed Length`, the local environment field is ignored. If this field is not present or evaluates to null, it is ignored and the value on the node is used.

These properties enable the connection details (host name and port number) and the connection used (ID) to be selected dynamically. The Reply ID can also be set on the connection, which enables a string to be stored in the connection and to be displayed in the local environment. In this way, you can store Reply IDs from other TCPIP nodes or from other transports, such as WebSphere MQ.

When a record has been retrieved, the ConnectionDetails field in the local environment tree is populated with the details of the connection that is being used.

Table 262. Output local environment properties

Location in local environment for output from node	Description
$LocalEnvironment/TCPIP/Receive/ConnectionDetails/Type	The Server.
$LocalEnvironment/TCPIP/Receive/ConnectionDetails/Hostname	The host name used to make a connection.
$LocalEnvironment/TCPIP/Receive/ConnectionDetails/Port	The port number used to make a connection.
$LocalEnvironment/TCPIP/Receive/ConnectionDetails/OpenTimestamp	The time stamp when the connection was first opened.
$LocalEnvironment/TCPIP/Receive/ConnectionDetails/CloseTimestamp	The time stamp when the connection was closed (null if not yet closed).
$LocalEnvironment/TCPIP/Receive/ConnectionDetails/SequenceNumber/InputRecord	The sequence number of the message that is received on this connection. The first record has a sequencing number of 1; the second record is 2; and so on.
$LocalEnvironment/TCPIP/Receive/ConnectionDetails/SequenceNumber/OutputRecord	The sequence number of the message that is sent on this connection. The first record has a sequencing number of 1; the second record is 2; and so on.
$LocalEnvironment/TCPIP/Receive/ConnectionDetails/Id	The ID of the socket being used. This ID is an internal identifier used by WebSphere Message Broker to uniquely identify a connection.
$LocalEnvironment/TCPIP/Receive/ConnectionDetails/ReplyId	The Reply ID that is stored on this connection. It can be any text string.
$LocalEnvironment/TCPIP/Receive/ConnectionDetails/ClientDetails/Hostname	The fully qualified domain name of the computer from which the client connected.
$LocalEnvironment/TCPIP/Receive/ConnectionDetails/ClientDetails/Address	The IP address of the computer from which the client connected.

The TCPIPServerReceive node is contained in the **TCPIP** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Message structure

The TCPIPServerReceive node handles messages in the following message domains:

- XMLNSC
- DataObject
- JSON
- BLOB
- MIME
- MRM
- JMSMap
- JMSStream
- XMLNS

Using the TCPIPServerReceive node in a message flow:
Look at the following samples to see how to use the TCPIPServerReceive node:

- TCPIP Client Nodes
- TCPIP Handshake

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Configuring the TCPIPServerReceive node:
When you have put an instance of the TCPIPServerReceive node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view.

All mandatory properties that do not have a default value defined are marked with an asterisk.

Configure the TCPIPServerReceive node:

1. Optional: On the **Description** tab, enter a `Short description`, a `Long description`, or both. You can also rename the node on this tab.
2. On the **Basic** tab, set the properties that determine how the TCP/IP connection is controlled.
   - Use the `Connection details` property to specify either the host name and port number to be used, or the name of a configurable service. This property is mandatory. The following formats are supported:
     - `Configurable service name`. This value is used to look up the port and host name in configurable services. For example, `TCPIPProfile1`.
     - `<Hostname>:<Port>`. This value is the host name followed by the port number (separated by a colon); for example, `tcpip.server.com:1111`
     - `<Port>`. This value is the port number. In this case, the host name is assumed to be `localhost`.
   - Use the `Timeout waiting for a data record (seconds)` property to specify how long the node listens on a connection for more data after the first byte

of data has arrived. You can specify any length of time in seconds. The default is 60 seconds. When the specified time has been exceeded, all available data is sent to the Failure terminal.

3. On the **Advanced** tab, set the properties that determine how the data stream is controlled.
   - Use the Close connection property to specify when and how to close the connection.
     - Select No to leave the connection open. This value is the default.
     - Select After timeout to close the connection when a timeout occurs.
     - Select After data has been received to close the connection when the end of the record is found.
   - Select Close input stream after a record has been received to close the input stream as soon as the data has been retrieved. By default this property is not selected. When the connection input stream is reserved, no other node can use it without knowing the ID.
   - Use the Input Stream Modification property to specify whether to reserve the input stream for use only by input and receive nodes that specify the connection ID, or to release the input stream at the end of the flow.
     - Select Leave unchanged to leave the input stream as it was when it entered the node. This value is selected by default.
     - Select Release input stream to specify that this input stream is returned to the pool and is available for use by any input or receive node.
     - Select Reserve input stream (for use by future TCPIP input and receive nodes) to specify that this input stream can be used only by this node and by other input or receive nodes that request it by specifying the connection ID. When the connection input stream is reserved, no other nodes can use it without specifying the correct connection ID.
     - Select Reserve input stream (for use by future TCPIP input and receive nodes) then release after propagate to specify that this input stream can be used only by this node and receive nodes that request it by specifying the correct connection ID. After the message has been propagated, this input stream is returned to the pool and becomes available for use by any input or receive node.
   - Use the Output Stream Modification property to specify whether to reserve or release the output stream. These options are available only if you have not selected the Close output stream after a record has been sent property.
     - Select Leave unchanged to leave the output stream as it was when it entered the node. This value is selected by default.
     - Select Release output stream to specify that this output stream is returned to the pool and is available for use by any output node.
     - Select Reserve output stream (for use by future TCPIP output nodes) to specify that this output stream can be used only by this node and by other output nodes that request it by specifying the connection ID. When the connection input stream is reserved, no other nodes can use it without specifying the correct connection ID.
     - Select Reserve output stream (for use by future TCPIP output nodes) then release after propagate to specify that this output stream can be used only by this node and output nodes that request it by specifying the correct connection ID. After the message has been propagated, this output stream is returned to the pool and becomes available for use by any output node.

4. On the **Request** tab, specify the location of the data to be written. You can specify the properties on this tab as XPath or ESQL expressions. Content Assist is available in the Properties view and also in the XPath Expression Builder, which you can run by clicking **Edit** to the right of each property.
   - In `Hostname location`, specify the location of the value to override the `Hostname` that is set in the `Connection details` property of the **Basic** tab. If you do not specify a location, the default value is `$LocalEnvironment/TCPIP/Receive/Hostname`.
   - In `Port location`, specify the location of the value to override the `Port` that is set in the `Connection details` property of the **Basic** tab. If you do not specify a location, the default value is `$LocalEnvironment/TCPIP/Receive/Port`.
   - In `ID location`, specify the location of the `Id` of the socket being used. This internal identifier is used by WebSphere Message Broker to uniquely identify a connection. If you do not specify a location, the default value is `$LocalEnvironment/TCPIP/Receive/Id`.
   - In `Reply ID location`, specify the location of the Reply ID that is stored on the connection that is being used. The Reply ID can be used when data is returned in an input node. If you do not specify a location, the default value is `$LocalEnvironment/TCPIP/Receive/ReplyId`.
5. On the **Result** tab, set values for the properties that determine where the reply is stored.
   - Use the `Output data location` property to specify the start location in the output message tree where the parsed elements from the bit string of the message are stored. The default value is `$OutputRoot`.
   - Use the `Copy local environment` property to specify whether the local environment is copied to the output message.
     – If `Copy local environment` is selected, a new copy of the local environment is created in the tree, and it is populated with the contents of the local environment from the preceding node. Therefore, if a node changes the local environment, the upstream nodes are not affected by those changes because they have their own copies. This value is the default.
     – If `Copy local environment` is not selected, the node does not generate its own copy of the local environment, but uses the local environment that is passed to it by the preceding node. Therefore, if a node changes the local environment, the changes are reflected by the upstream nodes.
6. On the **Input Message Parsing** tab, set values for the properties that the node uses to determine how to parse the incoming message.

   If the incoming message has an MQRFH2 header, you are not required to set values for the Input Message Parsing properties because the values are derived from the `<mcd>` folder in the MQRFH2 header; for example:

   ```
   <mcd><Msd>MRM</Msd><Set>DHM4UO906S001</Set><Type>receiptmsg1</Type>
   <Fmt>XML</Fmt></mcd>
   ```

   If you set values, and if they differ from the values in the MQRFH2 header, the values in the MQRFH2 header take precedence.
   - In `Message domain`, select the name of the parser that you are using from the list. The default is `BLOB`. You can choose from the following options:
     – XMLNSC
     – DataObject
     – JSON
     – BLOB
     – MIME

- MRM
- JMSMap
- JMSStream
- XMLNS

You can also specify a user-defined parser, if appropriate.

- If you are using the MRM or IDOC parser, or the XMLNSC parser in validating mode, select the `Message set` that you want to use. This list is populated with available message sets when you select MRM, XMLNSC, or IDOC as the domain.
- If you are using the MRM parser, select the correct message type from the list in `Message type`. This list is populated with available message types when you select the MRM parser.
- If you are using the MRM or IDOC parser, select the correct message format from the list in `Message format`. This list is populated with available message formats when you select the MRM or IDOC parser.
- Specify the message coded character set ID in `Message coded character set ID`.
- Select the message encoding from the list in `Message encoding` or specify a numeric encoding value. For more information about encoding, see "Data conversion" on page 1151.

7. On the **Parser Options** subtab:
   - `Parse timing` is, by default, set to `On Demand`, which causes parsing of the message to be delayed. To cause the message to be parsed immediately, see "Parsing on demand" on page 4173.
   - If you are using the XMLNSC parser, set values for the properties that determine how the XMLNSC parser operates. For more information, see "Manipulating messages in the XMLNSC domain" on page 2546.
8. Use the **Records and Elements** tab to specify how the data is interpreted as records. Only one record is retrieved each time the TCPIPServerReceive node is started; therefore, if the TCP/IP stream contains multiple logical messages, you must start the node multiple times to receive all the messages.
   - Use the `Record detection` property to determine how the data is split into records, each of which generates a single message. Choose from the following options:
     - `Connection closed` specifies that all of the data sent during a connection is a single record.
     - `Fixed Length` specifies that each record is a fixed number of bytes in length. Each record contains the number of bytes specified in the `Length` property, except possibly a shorter final record in the file.
     - Select `Delimited` if the records that you are processing are separated, or terminated, by a DOS or UNIX line end or by a sequence of user-defined delimiter bytes. Specify the delimiter and delimiter type in the `Delimiter` and `Delimiter type` properties.
     - Select `Parsed Record Sequence` if the data contains a sequence of one or more records that are serially recognized by the parser that is specified in `Message domain`. The node propagates each recognized record as a separate message. If you select this `Record detection` option, the parser specified in `Message domain` must be either XMLNSC or MRM (either CWF or TDS physical format).
   - If you set `Record detection` to `Fixed Length`, use `Length` to specify the required length of the output record. This value must be between 1 byte and 100 MB. The default is `80` bytes.

If you set Record detection to Connection closed, Fixed Length, or Delimited, a limit of 100 MB applies to the length of the records. If you set Record detection to Parsed Record Sequence, the TCPIPServerReceive node does not determine or limit the length of a record. Nodes that are downstream in the message flow might try to determine the record length or process a long record. If you intend to process large records in this way, ensure that your broker has sufficient memory. You might have to apply message flow techniques described in the Large Messaging sample to make the best use of the available memory.

- If you set Record detection to Delimited, use Delimiter to specify the delimiter to be used. Choose from the following options:
  - DOS or UNIX Line End, on UNIX systems, specifies the line feed character (<LF>, X'0A'), and, on Windows systems, specifies a carriage return character followed by a line feed character (<CR><LF>, X'0D0A'). The node treats both of these strings as delimiters, irrespective of the system on which the broker is running. If both strings can be seen in the same record, the node recognizes both as delimiters. The node does not recognize X'15' which, on z/OS systems, is the 'newline' byte; set this property to Custom Delimiter and set Custom delimiter to 15 if your input file is coded using EBCDIC new lines.
  - Custom Delimiter (hexadecimal), permits a sequence of bytes to be specified in Custom delimiter (hexadecimal)
- In Custom delimiter (hexadecimal), specify the delimiter byte or bytes to be used when Delimiter is set to Custom delimiter (hexadecimal). Specify this value as an even-numbered string of hexadecimal digits. The default is X'0A' and the maximum length of the string is 16 bytes (represented by 32 hexadecimal digits).
- If you set Record detection to Delimited, use Delimiter type to specify the type of delimiter. Permitted values are:
  - Infix. If you select this value, each delimiter separates records. If the data ends with a delimiter, the (zero length) data following the final delimiter is still propagated, although it contains no data.
  - Postfix. If you specify this value, each delimiter terminates records. If the data ends with a delimiter, no empty record is propagated after the delimiter. If the data does not end with a delimiter, it is processed as if a delimiter follows the final bytes of the data. Postfix is the default value.
- The TCPIPServerReceive node considers each occurrence of the delimiter in the input as either separating (infix) or terminating (postfix) each record. If the data begins with a delimiter, the node treats the (zero length) contents preceding that delimiter as a record and propagates an empty record to the flow. The delimiter is never included in the propagated message.

9. Use the **Validation** tab to provide validation based on the message set for predefined messages. For more information about validation, see "Validating messages" on page 1478. For information about how to complete this tab, see "Validation tab properties" on page 4169.

Terminals and properties:
The terminals of the TCPIPServerReceive node are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.

Terminal	Description
Out	The output terminal to which the message is routed if it is successfully retrieved from an external resource. If no errors occur within the input node, a message received from an external resource is always sent to the Out terminal first.
Timeout	The terminal to which a message is sent when the time specified in the `Timeout waiting for a data record` property has been exceeded. The message text is `Timeout value is exceeded`.
Failure	The output terminal to which the message is routed if an error occurs. These errors include failures caused by retry processing. Even if the Validation property is set, messages propagated to this terminal are not validated.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Description properties of the TCPIPServerReceive node are described in the following table.

Property	M	C	Default	Description
Node name	No	No	TCPIPServerReceive	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The Basic properties of the TCPIPServerReceive node determine how the TCP/IP connection is controlled, and are described in the following table.

Property	M	C	Default	Description	**mqsiapplybaroverride command property**
Connection details	Yes	Yes		A string containing the port number to be used, or the name of a configurable service. The following formats are supported: • `Configurable service name`. This value is used to look up the port and host name in configurable services. For example, `TCPIPProfile1`. • `<Port>`. This value is the port number; for example, `1111`. • `<Port>`. This value is the port number. In this case, the host name is assumed to be `localhost`.	connectionDetails
Timeout waiting for a data record (seconds)	Yes	Yes	`60`	Specifies how long the node listens on a connection for more data after the first byte of data has arrived. You can specify any length of time in seconds. The default is `60` seconds. When the specified time has been exceeded, all available data is sent to the Failure terminal.	timeoutWaitingForData

The Advanced properties of the TCPIPServerReceive node determine how the data stream is controlled, and are described in the following table.

Property	M	C	Default	Description
Close connection	Yes	No	No	Controls when the connection is closed, or if it remains open. Valid options are: • Select No to leave the connection open. This value is the default. • Select After timeout to close the connection when a timeout occurs. • Select After data has been received to close the connection when the end of the record is found.
Close input stream after a record has been received	Yes	No	Cleared	Specifies whether to close the input stream as soon as the data has been retrieved. When the connection input stream is reserved, no other node can use it without knowing the ID. By default this property is not selected.
Input Stream Modification	No	No	Leave unchanged	Specifies whether to reserve the input stream for use only by input and receive nodes that specify the connection ID, or to release the input stream at the end of the flow. Valid options are: • Select Leave unchanged to leave the input stream as it was when it entered the node. This value is selected by default. • Select Release input stream to specify that this input stream is returned to the pool and is available for use by any input or receive node. • Select Reserve input stream (for use by future TCPIP input and receive nodes) to specify that this input stream can be used only by this node and by other input or receive nodes that request it by specifying the connection ID. When the connection input stream is reserved, no other nodes can use it without specifying the correct connection ID. • Select Reserve input stream (for use by future TCPIP input and receive nodes) then release after propagate to specify that this input stream can be used only by this node and receive nodes that request it by specifying the correct connection ID. After the message has been propagated, this input stream is returned to the pool and becomes available for use by any input or receive node.
Output Stream Modification	No	No	Leave unchanged	Specifies whether to reserve this output stream or release it and return it to the pool for use by any output node. These options are available only if you have not selected the Close output stream after a record has been sent property. • Select Leave unchanged to leave the output stream as it was when it entered the node. This value is selected by default. • Select Release output stream to specify that this output stream is returned to the pool and is available for use by any output node. • Select Reserve output stream (for use by future TCPIP output nodes) to specify that this output stream can be used only by this node and by other output nodes that request it by specifying the connection ID. When the connection input stream is reserved, no other nodes can use it without specifying the correct connection ID. • Select Reserve output stream (for use by future TCPIP output nodes) then release after propagate to specify that this output stream can be used only by this node and output nodes that request it by specifying the correct connection ID. After the message has been propagated, this output stream is returned to the pool and becomes available for use by any output node.

The Request properties of the TCPIPServerReceive node specify the location of the data to be written. You can specify the properties on this tab as XPath or ESQL expressions. Content Assist is available in the Properties view and also in the XPath Expression Builder, which you can run by clicking **Edit** to the right of each property. The Request properties are described in the following table:

Property	M	C	Default	Description
Port location	Yes	No	`$LocalEnvironment/TCPIP/Receive/Port`	The message element location that contains the Port. Specify the location of the value to override the `Port` that is set in the `Connection details` property of the **Basic** tab. If you do not specify a location, the default value is `$LocalEnvironment/TCPIP/Receive/Port`.
ID location	Yes	No	`$LocalEnvironment/TCPIP/Receive/Id`	The message element location that contains the ID. Specify the location of the `Id` of the socket that is being used. This internal identifier is used by WebSphere Message Broker to uniquely identify a connection. If you do not specify a location, the default value is `$LocalEnvironment/TCPIP/Receive/Id`.
Reply ID location	Yes	No	`$LocalEnvironment/TCPIP/Receive/ReplyId`	The message element location that contains the Reply ID. Specify the location of the Reply ID that is stored on the connection being used. The Reply ID can be used when data is returned in an input node. If you do not specify a location, the default value is `$LocalEnvironment/TCPIP/Receive/ReplyId`.
Record length location	No	No	`$LocalEnvironment/TCPIP/Receive/Length`	The message element location that contains the record length to be read. Specify the location of the value to override the `Length (bytes)` property on the **Records and elements** tab. If you do not specify a location, the default value is `$LocalEnvironment/TCPIP/Receive/Length`.

The Result properties of the TCPIPServerReceive node determine where the reply is to be stored, and are described in the following table:

Property	M	C	Default	Description
Output data location	No	No	`$OutputRoot`	The start location in the output message tree where the parsed elements from the bit string of the message are stored.

Property	M	C	Default	Description
Copy local environment	No	No	Selected	Specifies whether the local environment is copied to the output message. • If `Copy local environment` is selected, a new copy of the local environment is created in the tree, and it is populated with the contents of the local environment from the preceding node. Therefore, that if a node changes the local environment, the upstream nodes are not affected by those changes because they have their own copies. This value is the default. • If `Copy local environment` is not selected, the node does not generate its own copy of the local environment, but uses the local environment that is passed to it by the preceding node. Therefore, if a node changes the local environment, the changes are reflected by the upstream nodes.

The Input Message Parsing properties of the TCPIPServerReceive node determine how to parse the incoming message.

If the incoming message has an MQRFH2 header, you do not have to set values for the Input Message Parsing properties because the values are derived from the `<mcd>` folder in the MQRFH2 header; for example:

```
<mcd><Msd>MRM</Msd><Set>DHM4UO906S001</Set><Type>receiptmsg1</Type>
<Fmt>XML</Fmt></mcd>
```

If you set values, and if they differ from the values in the MQRFH2 header, the values in the MQRFH2 header take precedence.

The TCPIPServerReceive node Input Message Parsing properties are described in the following table.

Property	M	C	Default	Description	**mqsiapplybaroverride command property**
Message domain	No	No	`BLOB`	The domain that is used to parse the incoming message. The default is `BLOB`. You can choose from the following options: • XMLNSC • DataObject • JSON • BLOB • MIME • MRM • JMSMap • JMSStream • XMLNS You can also specify a user-defined parser, if appropriate.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Message set	No	No		If you are using the MRM or IDOC parser, or the XMLNSC parser in validating mode, select the `Message set` in which the incoming message is defined. The list contains the message sets that are available when you select MRM, XMLNSC, or IDOC as the domain. If you set this property, then later update the project dependencies to remove this message set reference, a warning is issued. Either update the `Message Set` property, or restore the reference to this message set project.	
Message type	No	No		The name of the incoming message. If you are using the MRM parser, select the correct message type from the list in `Message type`. This list is populated with available message types when you select the MRM parser.	
Message format	No	No		The name of the physical format of the incoming message. If you are using the MRM or IDOC parser, select the correct message format from the list in `Message format`. This list is populated with available message formats when you select the MRM or IDOC parser.	
Message coded character set ID	Yes	No	`Broker System Default`	The ID of the coded character set used to interpret the data being read.	messageCodedCharSetIdProperty
Message encoding	Yes	No	`Broker System Determined`	The encoding scheme for numbers and large characters used to interpret the data being read. Valid values are `Broker System Determined` or a numeric encoding value. For more information about encoding, see "Data conversion" on page 1151.	messageEncodingProperty

The Parser Options properties of the TCPIPServerReceive node are described in the following table.

Property	M	C	Default	Description
Parse timing	No	No	`On Demand`	This property controls when an input message is parsed. Valid values are: • `On Demand` • `Immediate` • `Complete` `Parse timing` is, by default, set to `On Demand`, which causes parsing of the message to be delayed. To cause the message to be parsed immediately, see "Parsing on demand" on page 4173.
Build tree using XML schema data types	No	No	Cleared	This property controls whether the syntax elements in the message tree have data types taken from the XML Schema.

Property	M	C	Default	Description
Use XMLNSC compact parser for XMLNS domain	No	No	Cleared	This property controls whether the XMLNSC Compact Parser is used for messages in the XMLNS Domain. If you set this property, the message data is displayed under XMLNSC in nodes that are connected to the output terminal when the input MQRFH2 header or Input Message Parsing property, `Message Domain`, is XMLNS.
Retain mixed content	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters mixed text in an input message. If you select the check box, elements are created for mixed text. If you clear the check box, mixed text is ignored and no elements are created.
Retain comments	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters comments in an input message. If you select the check box, elements are created for comments. If you clear the check box, comments are ignored and no elements are created.
Retain processing instructions	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters processing instructions in an input message. If you select the check box, elements are created for processing instructions. If you clear the check box, processing instructions are ignored and no elements are created.
Opaque elements	No	No	Blank	This property is used to specify a list of elements in the input message that are to be opaquely parsed by the XMLNSC parser.

The Records and Elements properties of the TCPIPServerReceive node specify how the data is interpreted as records, and are described in the following table:

Property	M	C	Default	Description
Record detection	Yes	No	`Connection closed`	The mechanism used to identify records in the input data. • `Connection closed` specifies that all of the data sent during a connection is a single record. • `Fixed Length` specifies that each record is a fixed number of bytes in length. Each record contains the number of bytes specified in the `Length` property, except possibly a shorter final record in the file. • Select `Delimited` if the records you are processing are separated, or ended, by a DOS or UNIX line end or by a sequence of user-defined delimiter bytes. Specify the delimiter and delimiter type in the `Delimiter` and `Delimiter type` properties. • Select `Parsed Record Sequence` if the data contains a sequence of one or more records that are serially recognized by the parser specified in `Message domain`. The node propagates each recognized record as a separate message. If you select this `Record detection` option, the parser specified in `Message domain` must be either XMLNSC or MRM (either CWF or TDS physical format).

Property	M	C	Default	Description
Length (bytes)	Yes	No	0	If you set Record detection to Fixed Length, use Length to specify the required length of the output record in bytes. This value must be in the range 1 byte through 100 MB. The default is 80 bytes. If you set Record detection to Connection closed, Fixed Length, or Delimited, a limit of 100 MB applies to the length of the records. If you set Record detection to Parsed Record Sequence, the TCPIPServerReceive node does not determine or limit the length of a record. Nodes that are downstream in the message flow might try to determine the record length or process a long record. If you intend to process large records in this way, ensure that your broker has sufficient memory. You might have to apply message flow techniques described in the Large Messaging sample to make the best use of the available memory.
Delimiter	Yes	No	DOS or UNIX Line End	If you set Record detection to Delimited, use Delimiter to specify the delimiter to be used. Choose from the following options: • DOS or UNIX Line End, on UNIX systems, specifies the line feed character (<LF>, X'0A'), and, on Windows systems, specifies a carriage return character followed by a line feed character (<CR><LF>, X'0D0A'). The node treats both of these strings as delimiters, irrespective of the system on which the broker is running. If both strings are displayed in the same record, the node recognizes both as delimiters. The node does not recognize X'15' which, on z/OS systems, is the 'newline' byte; set this property to Custom Delimiter and set Custom delimiter to 15 if your input file is coded using EBCDIC new lines, such as EBCDIC files from a z/OS system. • Custom Delimiter (hexadecimal), permits a sequence of bytes to be specified in Custom delimiter (hexadecimal)
Custom delimiter (hexadecimal)	No	No		The delimiter byte or bytes to be used when Delimited record detection and Custom Delimiter (Hexadecimal) are selected. This property is mandatory only if the Delimiter property is set to Custom Delimiter (Hexadecimal). Specify this value as an even-numbered string of hexadecimal digits. The default is X'0A' and the maximum length of the string is 16 bytes (represented by 32 hexadecimal digits).
Delimiter type	Yes	No	Postfix	The location of the delimiter when Delimited record detection and Custom Delimiter (Hexadecimal) are selected. Valid options are: • Infix. If you select this value, each delimiter separates records. If the data ends with a delimiter, the (zero length) data following the final delimiter is still propagated, although it contains no data. • Postfix. If you specify this value, each delimiter ends records. If the data ends with a delimiter, no empty record is propagated after the delimiter. If the data does not end with a delimiter, it is processed as if a delimiter follows the final bytes of the data. Postfix is the default value. The TCPIPServerReceive node considers each occurrence of the delimiter in the input as either separating (infix) or terminating (postfix) each record. If the data begins with a delimiter, the node treats the (zero length) contents preceding that delimiter as a record and propagates an empty record to the flow. The delimiter is never included in the propagated message. This property is ignored unless the Delimiter property is set to Custom Delimiter (Hexadecimal).

The Validation properties of the TCPIPServerReceive node are described in the following table.

For a full description of these properties, see "Validation properties" on page 4169.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Validate	No	Yes	None	This property controls whether validation takes place. Valid values are • None • Content and Value • Content • Inherit	validateMaster
Failure action	No	No	Exception	This property controls what happens if validation fails. You can set this property only if you set Validate to Content or Content and Value. Valid values are: • User Trace • Local Error Log • Exception • Exception List	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"WebSphere Broker TCP/IP Transport" on page 1735
The WebSphere Broker TCP/IP Transport is a service that connects applications that use raw TCP/IP sockets for transferring data.

"TCP/IP nodes" on page 1738
WebSphere Message Broker implements access to the TCP/IP input and output streams through a series of nodes.

"Scenarios for WebSphere Message Broker and TCP/IP" on page 1746
Two example scenarios show how you might use TCP/IP and WebSphere Message Broker as part of a business solution.

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

"Parsers" on page 1072
A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from the internal message tree representation.

"Specifying namespaces in the Message Type property" on page 1208
When using the MRM domain, the *Message Type* property is used to specify the name of the message.

"Samples" on page 98
The WebSphere Message Broker Toolkit provides samples that show the features that are available in WebSphere Message Broker, and how to use them. This topic provides links to the information about the individual samples.

Related tasks:

"Working with TCP/IP" on page 1750
You can use WebSphere Message Broker TCPIP nodes and TCP/IP configurable services to perform various tasks.

"Using more than one input node" on page 1473
You can include more than one input node in a single message flow.

"Validating messages" on page 1478
The broker provides validation based on the message model for predefined messages.

"Manipulating messages in the XMLNSC domain" on page 2546
If you are writing ESQL to process messages in the XMLNSC domain, it is helpful to learn about the structure of the message tree that the XMLNSC parser builds.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

Related reference:

"Parsing on demand" on page 4173
On-demand parsing, referred to as partial parsing, is used to parse an input message bit stream only as far as is necessary to satisfy the current reference. The parsers that can perform partial parsing of input messages are the JSON, MRM, XML, XMLNS, and XMLNSC parsers.

"Configurable message flow properties" on page 4020
When you add a message flow to a broker archive (BAR) file in preparation for deploying it to a broker, you can set additional properties that influence its run time operation. These properties are available for review and update when you select the **Manage and Configure** tab for the broker archive file.

"Input node" on page 4511
Use the Input node as an In terminal for an embedded message flow (a subflow).

"Validation properties" on page 4169
You can control validation by setting properties on the Validate and Parser Options tabs for the nodes that are listed in the following table.

"**mqsicreateconfigurableservice** command" on page 3849
Use the **mqsicreateconfigurableservice** command to create an object name for a broker external resource.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"TCPIPClientInput node" on page 4854
Use the TCPIPClientInput node to create a client connection to a raw TCP/IP socket, and to receive data over that connection.

"TCPIPClientOutput node" on page 4867
Use the TCPIPClientOutput node to create a client connection to a raw TCP/IP socket, and to send data over that connection to an external application.

"TCPIPServerInput node" on page 4890
Use the TCPIPServerInput node to create a server connection to a raw TCPIP socket, and to receive data over that connection.

"TCPIPServerOutput node" on page 4903
Use the TCPIPServerOutput node to create a server connection to a raw TCP/IP socket, and to send data over the connection to an external application.

"TCPIPClientReceive node" on page 4877
Use the TCPIPClientReceive node to receive data over a client TCP/IP connection.

Throw node

Use the Throw node to throw an exception in a message flow.

This topic contains the following sections:

Purpose:

An exception can be caught and processed by:

- A preceding TryCatch node
- The message flow input node (the built-in nodes, for example HTTPInput and MQInput, have Catch terminals)
- A preceding AggregateReply node

Include a Throw node to force an error path through the message flow if the content of the message contains unexpected data. For example, to back out a message that does not contain a particular field, you can check (using a Filter node) that the field exists; if the field does not exist, the message can be passed to a Throw node that records details about the exception in the exception list subtree in the message.

The Throw node is contained in the **Construction** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:

Look at the following samples to see how to use this node:

- Airline Reservations
- Error Handler
- Large Messaging

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Include a Throw node with a TryCatch node in your message flow to alert the systems administrator of a potential error situation; for example, if you have a Compute node that calculates a number, test the result of this calculation and

throw an exception if the result exceeds a certain amount. The TryCatch node catches this exception and propagates the message to a sequence of nodes that process the error.

Terminals and properties:
When you have put an instance of the Throw node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The Throw node terminal is described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Throw node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type: Throw	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The Throw node Basic properties are described in the following table.

Property	M	C	Default	Description
Message Catalog	No	No		The name of the message catalog from which the error text for the error number of the exception is extracted. Enter the fully-qualified path and file name of the message catalog that contains the message source. This file can be your own message catalog, or the default message catalog that is supplied with WebSphere Message Broker. To use the default supplied catalog, leave this property blank.
Message Number	No	No	`3001`	The error number of the exception that is being thrown. • If you have created your own message catalog, enter the number for the message in the catalog that you want to use when this exception is thrown. • If you are using the default message catalog, specify a number between 3001 (the default) and 3049. These numbers are reserved in the default catalog for your use. The text of each of these messages in the default message catalog is identical, but you can use a different number in this range for each situation in which you throw an exception; use the number to identify the exact cause of the error.

Property	M	C	Default	Description
Message Text	No	No		Additional text that explains the cause of the error. Enter any additional free format text that contains information that you want to include with the message when it is written to the local error log; for example, if you have checked for the existence of a particular field in a message and thrown an exception when that field is not found, you might include the text: `The message did not contain the required field: Branch number` If you are using the default message catalog, this text is inserted as `&1` in the message text.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Exception list tree structure" on page 1066
The exception list tree is a part of the logical message tree in which the message flow writes information about exceptions that occur when a message is processed.

Related tasks:

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Creating message catalogs" on page 3138
Create your own message catalogs to write tailored entries to the local error log.

Related reference:

"Filter node" on page 4452
Use the Filter node to route a message according to message content.

"HTTPInput node" on page 4474
Use the HTTPInput node to receive an HTTP message from an HTTP client for processing by a message flow.

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

"SCADAInput node" on page 4706
The SCADAInput node, available in earlier versions of WebSphere Message Broker, is not supported in WebSphere Message Broker Version 7.0. See for information about migrating your message flows from WebSphere Message Broker to WebSphere Message Broker Version 7.0.

"Trace node" on page 4942
Use the Trace node to generate trace records that you can use to monitor the behavior of a message flow.

"TryCatch node" on page 4949
Use the TryCatch node to provide a special handler for exception processing.

TimeoutControl node

Use the TimeoutControl node to process an input message that contains a timeout request.

This topic contains the following sections:
- "Purpose"
- "Using this node in a message flow"
- "Terminals and properties" on page 4933

Purpose:
The TimeoutControl node validates the timeout request message, stores the message, and propagates the message (unchanged) to the next node in the message flow. For more information, see "Sending timeout request messages" on page 2810.

The TimeoutControl node is contained in the **Timer** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:
Use a TimeoutControl node and a TimeoutNotification node together in a message flow for an application that requires events to occur at particular times, or at regular intervals.

Examples of when you can use the timeout nodes in a message flow include:
- You need to run a batch job every day at midnight.
- You want information about currency exchange rates to be sent to banks at hourly intervals.
- You want to confirm that important transactions are processed within a certain time period and perform some other specified actions to warn when a transaction has not been processed in that time period.

You can use more than one TimeoutControl node with a TimeoutNotification node. Timeout requests that are initiated by those TimeoutControl nodes are all processed by the same TimeoutNotification node if the same `Unique identifier` is used for the TimeoutNotification node and each of the TimeoutControl nodes.

You can use TimeoutControl nodes with a TimeoutNotification node that is in a separate message flow only if the following conditions are met:
- The same `Unique identifier` is used for your TimeoutNotification node and each of your TimeoutControl nodes

- The message flow that contains your TimeoutControl nodes and the message flow that contains your TimeoutNotification node are deployed to the same execution group

Look at the following sample for more details about how to use the timeout processing nodes:

- Timeout Processing

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Terminals and properties:
When you have put an instance of the TimeoutControl node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The TimeoutControl node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message tree for processing (which includes validating the timeout request specified in the message tree at `Request location`) and adds it to the control queue.
Failure	The output terminal to which the input message is propagated if a failure is detected during processing in this node. If this terminal is not connected to another node, error information is passed back to the previous node in the message flow.
Out	The output terminal to which incoming messages are propagated, unchanged, after successful timeout request processing. If this terminal is not connected to another node, no propagation occurs. If propagation of the message fails, the message is propagated to the Failure terminal.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Description properties of the TimeoutControl node are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, TimeoutControl	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The Basic properties of the TimeoutControl node are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Unique identifier	Yes	Yes	None	This is the only mandatory property for the node. Its value must be unique within the broker. The equivalent property of the TimeoutNotification node with which it is paired must have the same value. The maximum length of this identifier is 12 characters. This name is also used to identify a Timer configurable service (if one exists) to be used by the node.	uniqueIdentifier
Request location	No	No	None	This property describes where to find the timeout request information in the incoming message. This value can be any valid location in the input message tree and is validated at run time. If you do not specify a request location, InputLocalEnvironment.TimeoutRequest is assumed. For more information about the timeout request message, see "Sending timeout request messages" on page 2810.	
Request persistence	No	No	Automatic	This property controls whether an incoming timeout request survives a restart of either the broker or the message flow that contains the TimeoutNotification node that is paired with this TimeoutControl node. Select `Yes` if you want the incoming request to persist; select `No` if you do not. If you select `Automatic` (the default), the Persistence setting in the Properties folder of the incoming message is used.	

The Message properties of the TimeoutControl node are described in the following table.

The TimeoutControl node Message properties are described in the following table.

Property	M	C	Default	Description
Stored message location	No	No	None	This property identifies the location of the part of the request message that you want to store for propagation by the TimeoutNotification node with which this node is paired. If you do not specify a value, the entire message is stored. You can specify any valid location in the message tree. If you choose to store the entire message, you do not need to specify any values in `Message domain`, `Message set`, `Message type`, or `Message format`.

Property	M	C	Default	Description
Message domain	No	No	BLOB	The domain that is used to parse the stored timeout request message by the TimeoutNotification node. If you do not specify a value and the message location is stored, the default value is BLOB. Select the name of the parser that you are using. This value, and the three corresponding values in Message set, Message type, and Message format, are used by the TimeoutNotification node with which it is paired when it rebuilds the stored message for propagation. If you have stored the entire request message (by leaving Stored message location blank), do not specify any values here. If you choose to store part of the request message, specify values here that reflect the stored request message fragment as if it were the entire message, which is the case when it is processed by the TimeoutNotification node. Choose from the following parsers: • XMLNSC • JSON • BLOB • MRM • XMLNS You can also specify a user-defined parser, if appropriate.
Message set	No	No	None	The name or identifier of the message set in which the stored timeout request message is defined. If you are using the MRM parser, or the XMLNSC parser in validating mode, select the Message set that you want to use from the list. If you set this property, then later update the project dependencies to remove this message set reference, a warning is issued. Either update the Message set property, or restore the reference to this message set project.
Message type	No	No	None	The name of the stored timeout request message. If you are using the MRM parser, select the correct message from the list in Message type. This list is populated with messages that are defined in the Message set that you have selected.
Message format	No	No	None	The name of the physical format of the stored timeout request message. If you are using the MRM parser, select the format of the message from the list in Message format. This list includes all the physical formats that you have defined for this Message set.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Parsers" on page 1072
A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from

the internal message tree representation.

"Specifying namespaces in the Message Type property" on page 1208
When using the MRM domain, the *Message Type* property is used to specify the name of the message.

"Multipart messages" on page 1191
A *multipart message* contains one or more other messages within its structure. The contained message is sometimes referred to as an *embedded message*.

"Configuring timeout flows" on page 2809
Use the TimeoutControl and TimeoutNotification nodes in message flows to process timeout requests or to generate timeout notifications at specified intervals.

"Sending timeout request messages" on page 2810
To set a controlled timeout, send a message with a set of elements with well known names to a TimeoutControl node. These elements control the properties of the timeout to be created or deleted.

Related tasks:

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

Related reference:

"TimeoutNotification node"
Use the TimeoutNotification node to manage timeout-dependent message flows.

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

TimeoutNotification node

Use the TimeoutNotification node to manage timeout-dependent message flows.

This topic contains the following sections:
- "Purpose"
- "Using this node in a message flow"
- "Terminals and properties" on page 4937

Purpose:
The TimeoutNotification node is an input node that you can use in two ways:

- A TimeoutNotification node can be paired with one or more TimeoutControl nodes.

  The TimeoutNotification node processes timeout request messages that are sent by the TimeoutControl nodes with which it is paired, and propagates copies of the messages (or selected fragments of the messages) to the next node in the message flow.
- A TimeoutNotification node can be used as a stand-alone node.

  Generated messages are propagated to the next node in the message flow at time intervals that are specified in the configuration of this node.

The TimeoutNotification node is contained in the **Timer** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:

Use a TimeoutControl node and a TimeoutNotification node together in a message flow for an application that requires events to occur at a particular time, or at regular intervals; for example, when you want a batch job to run every day at midnight, or you want information about currency exchange rates to be sent to banks at hourly intervals.

You can use more than one TimeoutControl node with a TimeoutNotification node. Timeout requests that are initiated by those TimeoutControl nodes are all processed by the same TimeoutNotification node if the same `Unique identifier` is used for the TimeoutNotification node and each of the TimeoutControl nodes. However, do not use the same `Unique identifier` for more than one TimeoutNotification node.

You can use TimeoutControl nodes with a TimeoutNotification node that is in a separate message flow only if the following conditions are met:

- The same `Unique identifier` is used for your TimeoutNotification node and each of your TimeoutControl nodes
- The message flow that contains your TimeoutControl nodes and the message flow that contains your TimeoutNotification node are deployed to the same execution group

Timeout request messages are stored for processing on a queue used by the TimeoutNotification node. By default, this queue is the SYSTEM.BROKER.TIMEOUT.QUEUE. However, you can use a Timer configurable service to specify an alternative timeout queue, which provides greater control over the storage of messages. For information about using an alternative timeout queue, see "Configuring the storage of events for timeout nodes" on page 760.

When a TimeoutNotification node is started as a result of the broker starting, or by the message flow that contains the node starting, it scans its internal timeout store and purges any non-persistent timeout requests. Notifications are issued for any persistent timeout requests that are now past and that have the IgnoreMissed property set to `False`.

If you use a TimeoutNotification node to generate a WebSphere MQ message to an output node, such as theMQOutput node, provide a valid MQMD. You must also provide a valid MQMD if the TimeoutNotification node is running in automatic mode (as a stand-alone node). If the TimeoutNotification node is running in controlled mode (that is, it is paired with one or more TimeoutControl nodes), you must provide a valid MQMD only if the stored messages do not already have an MQMD. The following ESQL shows how to provide a valid MQMD:

```
CREATE NEXTSIBLING OF OutputRoot.Properties DOMAIN 'MQMD';
SET OutputRoot.MQMD.Version = MQMD_CURRENT_VERSION;
SET OutputRoot.MQMD.Format = 'XML';
```

Look at the following sample for more details about how to use the timeout processing nodes:

- Timeout Processing

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Terminals and properties:

When you have put an instance of the TimeoutNotification node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The terminals of the TimeoutNotification node are described in the following table.

Terminal	Description
Failure	The output terminal to which the message is propagated if a failure is detected during processing in this node. Nodes can be connected to this terminal to process these failures. If this terminal is not connected to another node, messages are not propagated and no logging or safe storage of data occurs.
Out	The output terminal to which messages are propagated after timeouts expire. • If the TimeoutNotification node is running in `Automatic` mode (that is, there are no TimeoutControl nodes paired with this node), the propagated messages contain only a Properties folder and a LocalEnvironment that is populated with the timeout information. • If the TimeoutNotification node is running in `Controlled` mode (that is, TimeoutControl nodes that are paired with this node store timeout requests), the propagated messages contain what was stored by the TimeoutControl nodes, which might be entire request messages or fragments of them. If the TimeoutNotification node is used as the input node to a message flow that generates a WebSphere MQ message (for example, by using an MQOutput node), the message flow must create the necessary MQ headers and data (for example, MQMD).
Catch	The output terminal to which the message is propagated if an exception is thrown downstream. If this terminal is not connected to another node, the following events occur: 1. The TimeoutNotification node writes the error to the local error log. 2. The TimeoutNotification node repeatedly tries to process the request until the problem that caused the exception is resolved.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Description properties of the TimeoutNotification node are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type: TimeoutNotification	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The Basic properties of the TimeoutNotification node are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Unique Identifier	Yes	Yes	None	This property specifies a value that is unique within the broker and that is the same as the identifier that is specified for the TimeoutControl nodes with which this node is paired (if there are any). The maximum length of this identifier is 12 characters. This name is also used to identify a Timer configurable service (if one exists) to be used by the node. Do not use the same `Unique Identifier` for more than one TimeoutNotification node.	uniqueIdentifier
Transaction Mode	No	No	`Yes`	The transaction mode for the node. If the transaction mode is `Automatic`, a transaction is based on the persistence of the stored messages, which is controlled by the `Request Persistence` property of the TimeoutControl node with which it is paired. You can set this property to one of the following values: • Select `Yes` if you want a transaction to be started. • Select `No` if you do not want a transaction to be started. • Select `Automatic` only if you have set `Operation Mode` to `Controlled`. Whether a transaction is started depends on the persistence of the stored timeout requests, which is controlled by the value of `Request Persistence` in the TimeoutControl node with which it is paired.	
Operation Mode	No	No	`Automatic`	This property indicates whether this node is paired with any paired TimeoutControl nodes. Valid values are: • If you select `Automatic` the node is not paired with any TimeoutControl nodes. The node generates timeout requests with an interval that is controlled by the setting of the `Timeout Value` property, which must be a positive integer. • If you select `Controlled` the node processes all timeout requests that have been stored by the TimeoutControl nodes with which it is paired.	
Timeout Interval	No	Yes	1	The interval (in seconds) between timeout requests. This property is relevant only if `Operation Mode` is set to `Automatic`. The value of this property must be a positive integer. If the `Operation Mode` is set to `Automatic`, the value of the `Timeout Interval` property is overridden by the **`Timeout interval`** property, if set, in the Timer configurable service.	timeoutInterval

The properties of the Parser Options for the TimeoutNotification node are described in the following table.

Property	M	C	Default	Description
Parse Timing	No	No	On Demand	This property controls when the timeout message is parsed. Valid values are On Demand, Immediate, and Complete. By default, this property is set to On Demand, which causes parsing of the message to be delayed. To cause the message to be parsed immediately, see "Parsing on demand" on page 4173.
Build tree using XML schema data types	No	No	Cleared	This property controls whether the XMLNSC parser creates syntax elements in the message tree with data types taken from the XML Schema. You can select this property only if you set the Validate property on the **Validation** tab to Content or Content and Value.
Use MQRFH2C Compact Parser for MQRFH2 Domain	No	No	Cleared	This property controls whether the MQRFH2C Compact Parser, instead of the MQRFH2 parser, is used for MQRFH2 headers.
Use XMLNSC Compact Parser for XMLNS Domain	No	No	Cleared	This property controls whether the XMLNSC Compact Parser is used for messages in the XMLNS Domain. If you set this property, the message data appears under XMLNSC in nodes that are connected to the output terminal when the input RFH2 header or default properties Domain is XMLNS.
Retain Mixed Content	No	No	None	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters mixed text in a timeout message. If you select the check box, elements are created for mixed text. If you clear the check box, mixed text is ignored and no elements are created.
Retain Comments	No	No	None	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters comments in a timeout message. If you select the check box, elements are created for comments. If you clear the check box, comments are ignored and no elements are created.
Retain Processing Instructions	No	No	None	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters processing instructions in a timeout message. If you select the check box, elements are created for processing instructions. If you clear the check box, processing instructions are ignored and no elements are created.
Opaque elements	No	No	Blank	This property is used to specify a list of elements in the timeout message that are to be opaquely parsed by the XMLNSC parser. Opaque parsing is performed only if validation is not enabled (that is, if Validate is None); entries that are specified in Opaque Elements are ignored if validation is enabled.

The Validation properties of the TimeoutNotification node are described in the following table.

If a message is propagated to the Failure terminal of the node, it is not validated. For more information, see "Validating messages" on page 1478 and "Validation properties" on page 4169.

Property	M	C	Default	Description	**mqsiapplybaroverride command property**
Validate	No	Yes	None	This property controls whether validation takes place. Valid values are None, Content, and Content And Value.	validateMaster

Property	M	C	Default	Description	mqsiapplybaroverride command property
Failure Action	No	No	Exception	This property controls what happens if validation fails. You can set this property only if you set Validate to Content or Content and Value. Valid values are User Trace, Local Error Log, Exception, and Exception List.	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Parsers" on page 1072
A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from the internal message tree representation.

"Configuring timeout flows" on page 2809
Use the TimeoutControl and TimeoutNotification nodes in message flows to process timeout requests or to generate timeout notifications at specified intervals.

"Handling timeout notification errors" on page 2833
The TimeoutNotification node takes various actions when it handles errors with transactional messages, depending on whether the Failure and Catch terminals are connected.

"Sending timeout request messages" on page 2810
To set a controlled timeout, send a message with a set of elements with well known names to a TimeoutControl node. These elements control the properties of the timeout to be created or deleted.

"Considering performance for timeout flows" on page 2822
When you design timeout flows, the decisions that you make can affect the performance of your brokers and applications.

Related tasks:

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

"Validating messages" on page 1478
The broker provides validation based on the message model for predefined messages.

"Manipulating messages in the XMLNSC domain" on page 2546
If you are writing ESQL to process messages in the XMLNSC domain, it is helpful to learn about the structure of the message tree that the XMLNSC parser builds.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

Related reference:

"Parsing on demand" on page 4173
On-demand parsing, referred to as partial parsing, is used to parse an input message bit stream only as far as is necessary to satisfy the current reference. The parsers that can perform partial parsing of input messages are the JSON, MRM, XML, XMLNS, and XMLNSC parsers.

"Validation properties" on page 4169
You can control validation by setting properties on the Validate and Parser Options tabs for the nodes that are listed in the following table.

"TimeoutControl node" on page 4932
Use the TimeoutControl node to process an input message that contains a timeout request.

"Example XML timeout request message" on page 2812
The format used here is XML, but you can use any format that is supported by an installed parser.

"Configurable services properties" on page 3766
The supplied configurable services, and the configurable services that you create, are defined by their names and properties. You can use the supplied services.

Trace node

Use the Trace node to generate trace records that you can use to monitor the behavior of a message flow.

This topic contains the following sections:
- "Purpose"
- "Using this node in a message flow" on page 4943
- "Terminals and properties" on page 4943

Purpose:
Trace records can incorporate text, message content, and date and time information, to help you to monitor the behavior of the message flow.

You can write the records to the user trace file, another file, or the local error log (which contains error and information messages written by all other WebSphere Message Broker components). If you write traces to the local error log, you can issue a message from the default message catalog that is supplied with WebSphere Message Broker, or you can create your own message catalog.

The operation of the Trace node is independent of the setting of user tracing for the message flow that contains it. In particular, records that are written by the Trace node to the user trace log are written even if user trace is not currently active for the message flow.

The Trace node is contained in the **Construction** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:
Look at the following samples to see how to use this node:
- Airline Reservations
- Aggregation
- Timeout Processing

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Include a Trace node to help diagnose errors in your message flow. By tracing the contents of the message at various points in the flow, you can determine the sequence of processing. You can configure the Trace node to record the content of a message, and to check the action of a specific node on the message. For example, you can include a Trace node immediately after a Compute node to check that the output message has the expected format.

You can also use the Trace node to provide information in error handling in your message flows. For example, you can use this node to record failures in processing because of errors in the content or format of a message.

When you have tested the message flow and proved that its operation is correct, remove Trace nodes from your message flow, or switch them off.

Terminals and properties:
When you have put an instance of the Trace node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The terminals of the Trace node are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Out	The output terminal through which the message is propagated.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Description properties of the Trace node are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type: Trace	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The Basic properties of the Trace node are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Destination	Yes	No	`User Trace`	The destination of the trace record that is written by the node. The `Destination` refers to the computer that hosts the broker on which the Trace node is deployed: • To write the trace record to the local system error log, select `Local Error Log`. The information that you include in the trace record is written to one of the following locations: – Windows On Windows systems, data is written to the Event log (Application View) – Linux UNIX On Linux and UNIX systems, data is written to the `syslog` – z/OS On z/OS systems, data is written to the operator console UNIX On UNIX systems, syslog entries are restricted in length and messages are truncated by the newline character. To record a large amount of data in a log, set the destination to `File` or `User Trace` instead. If you select `Local Error Log`, indicate the number of the trace message that is to be written, and the message catalog in which the message is defined. – If you leave `Message Catalog` blank, the default message catalog is used as the source of the message that is to be written. You must also enter the error number of the record in `Message Number`. Numbers `3051` to `3099` are reserved in the default catalog for this use. The text of each of these messages in the default message catalog is identical, but if you use a different number in this range for each situation that you trace, you can identify the exact cause of the error. The default message number is `3051`. – If you create your own message catalog, enter the fully qualified file name for your catalog in `Message Catalog`. You must also enter the appropriate number for the message in the catalog that you want to write to the local error log in `Message Number`. On some systems, message numbers that end `00` are reserved for system use; do not include messages with numbers such as `3100` in your message catalog.	

Property	M	C	Default	Description	mqsiapplybaroverride command property

				• To write the trace record to the system-generated user trace log, select `User Trace`.
				These records are written regardless of the setting of the User Trace property for the deployed message flow.
				The location of the trace logs depends on your environment:
				Windows **Windows**
				If you set the work path using the **-w** parameter of the **mqsicreatebroker** command, the location is `workpath\log`.
				If you have not specified the broker work path, the location is:`%ALLUSERSPROFILE%\Application Data\IBM\MQSI\common\log` where `%ALLUSERSPROFILE%` is the environment variable that defines the system working directory. The default directory depends on the operating system:
				– On Windows XP and Windows Server 2003: `C:\Documents and Settings\All Users\Application Data\IBM\MQSI\common\log`
				– On Windows Vista and Windows Server 2008: `C:\ProgramData\IBM\MQSI\common\log`
				The actual value might be different on your computer.
				Linux UNIX **Linux and UNIX**
				`/var/mqsi/common/log`
				z/OS **z/OS**
				`/component_filesystem/log`
				The file name is made up of the broker name, the broker UUID, and a suffix of `userTrace.bin` (for example, `broker.e51906cb-dd00-0000-0080-b10e69a5d551.userTrace.bin.0`). Use the **mqsireadlog** and **mqsiformatlog** commands before you view the user trace log.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
				• To write the trace record to a file of your choice, select `File`. If you select this option, you must also set `File Path` to the fully qualified path name for the trace. If you do not set the path, the location of the file depends on the system; for example, on z/OS, the file is created in the home directory of the broker service ID. You can use any name for the trace file; for example, `c:\user\trace\trace.log` If you specify a file that does not exist already, the file is created. However, directories are not created by this process, so the full path must exist. The file is written as text, in the format specified by the `Pattern` property. You do not need to run the **mqsireadlog** or **mqsiformatlog** commands against the file. If a file write error occurs during processing (because of an out of space condition, for example), a single warning message, BIP4065, is written to the local system error log, and the message flow continues to process messages without logging further errors. Check the error log carefully for such a message. • If you do not want to write trace records, select `None`. You can also switch off Trace nodes.	
File Path	No	Yes		The fully qualified file name of the file to which to write records. This property is valid only if `Destination` is set to `File`.	filePath

Property	M	C	Default	Description	mqsiapplybaroverride command property

Pattern	No	No		The data that is to be included in the trace record. Create an ESQL pattern to specify what information to write. If you write the trace record to the local error log, the pattern governs the information that is written in the text of the message number that is selected. If you use the default message catalog, and a number between 3051 and 3099, the pattern information is inserted as &1 in the message text.
				• You can write plain text, which is copied into the trace record exactly as you have entered it.
				• You can identify parts of the message to write to the trace record, specifying the full field identifiers enclosed between the characters ${ and }. To record the entire message, specify ${Root}.
				• Use the ESQL functions to provide additional information; for example, use the ESQL function CURRENT_DATE to record the date, time, or both, at which the trace record is written.
				
				The pattern shown here includes some of the options that are available. The pattern writes an initial line of text, records two elements of the current message, and adds a simple timestamp:
				`Message passed through with the following fields:`
				`Store name is ${Body.storedetailselement.storename}`
				`Total sales are ${Body.totalselement.totalsales}`
				`Time is: ${EXTRACT(HOUR FROM CURRENT_TIMESTAMP)}`
				`:${EXTRACT(MINUTE FROM CURRENT_TIMESTAMP)}`
				
				The resulting trace record is:
				`Message passed through with the following fields:`
				`Store name is 'SRUCorporation'`
				`Total sales are '34.98'`
				`Time is: 11:19`
				
				A pattern that contains syntax errors does not prevent a message flow that contains the Trace node from deploying, but the node writes no trace records.	

Message Catalog	No	No		The name of the message catalog from which the error text for the error number of the exception is extracted. The default value (blank) indicates that the message is taken from the message catalog that is supplied with WebSphere Message Broker. See “Creating message catalogs” on page 3138 for more information.	
Message Number	No	No	3051	The error number of the message that is written.	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Flow debugger overview" on page 3158
Use the flow debugger in the WebSphere Message Broker Toolkit to track messages through your message flows.

Related tasks:

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Creating message catalogs" on page 3138
Create your own message catalogs to write tailored entries to the local error log.

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

Chapter 13, "Troubleshooting and support," on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

"Switching Trace nodes on and off" on page 3555
Use the **`mqsichangetrace`** command or the WebSphere Message Broker Explorer to switch Trace nodes on and off.

Related reference:

"Throw node" on page 4929
Use the Throw node to throw an exception in a message flow.

"TryCatch node" on page 4949
Use the TryCatch node to provide a special handler for exception processing.

"Local error logs" on page 6867
WebSphere Message Broker components use the local error log (also known as the system log) to record information about major activities within the system. When an error occurs, check the local error log first.

"User trace" on page 6873
User trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Administration log. User trace is inactive by default; you must activate it explicitly by using a command, or by selecting options in the

WebSphere Message Broker Toolkit.

"**mqsireadlog** command" on page 3905
Use the **mqsireadlog** command to retrieve trace records for the specified component.

"**mqsiformatlog** command" on page 3880
Use the **mqsiformatlog** command to process the XML log created by **mqsireadlog**. The command retrieves and formats any messages that the XML log contains into a form suitable for the locale of the user who runs the command.

"CURRENT_DATE function" on page 5179

TryCatch node

Use the TryCatch node to provide a special handler for exception processing.

This topic contains the following sections:

Purpose:
Initially, the input message is routed on the Try terminal, which you must connect to the remaining non-error processing nodes of the message flow. If a downstream node (which can be a Throw node) throws an exception, the TryCatch node catches it and routes the original message to its Catch terminal. Connect the Catch terminal to further nodes to provide error processing for the message after an exception. If the Catch terminal is connected, the message is propagated to it. If the Catch terminal is not connected, the message is discarded.

The TryCatch node is contained in the **Construction** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:
Look at the following sample to see how to use this node:

- Error Handler

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Use the Throw and TryCatch nodes when you use the Compute node to calculate a total. You can create a message that is sent to your system administrator when the total that is calculated exceeds the maximum value for the Total field.

Connecting the terminals:
The TryCatch node has no configurable properties that affect its operation. You determine how it operates by connecting the output terminals to subsequent nodes in your message flow.

1. Connect the Try terminal to the first node in the sequence of nodes that provides the normal (non-error) phase of processing of this message. This

sequence can contain one or more nodes that perform any valid processing. The sequence of nodes can optionally conclude with an output node.

2. Connect the Catch terminal to the first node in the sequence of nodes that provides the error processing for this message flow. This sequence can contain one or more nodes that perform any valid processing. The sequence of nodes can optionally conclude with an output node.

   When an exception is thrown in the message flow, either by the explicit use of the Throw node or the ESQL THROW statement, or by the broker raising an implicit exception when it detects an error that the message flow is not programmed to handle, control returns to the TryCatch node.

   The node propagates the message to the sequence of nodes connected to the Catch terminal (the catch flow) and the error handling that you have designed is initiated. The content of the message tree that is propagated is identical to the content that was propagated to the Try terminal, which is the content of the tree when the TryCatch node first received it. The node enhances the message tree with the new exception information that it has written to the exception list tree. Any modifications or additions that the nodes in try flow made to the message tree are not present in the message tree that is propagated to the catch flow.

Terminals and properties:
When you have put an instance of the TryCatch node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view.

The TryCatch node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Catch	The output terminal to which the message is propagated if an exception is thrown downstream and caught by this node.
Try	The output terminal to which the message is propagated if it is not caught.

The following table describes the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The TryCatch node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type: TryCatch	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Exception list tree structure" on page 1066
The exception list tree is a part of the logical message tree in which the message flow writes information about exceptions that occur when a message is processed.

Related tasks:

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

Related reference:

"Throw node" on page 4929
Use the Throw node to throw an exception in a message flow.

"Trace node" on page 4942
Use the Trace node to generate trace records that you can use to monitor the behavior of a message flow.

"THROW statement" on page 5161
Use the THROW statement to generate a user exception.

TwineballInput node

Use the TwineballInput node to discover how the WebSphere Adapters nodes work.

This topic contains the following sections:

Purpose:
The TwineballInput node is provided for educational purposes and helps you to see how the WebSphere Adapters nodes work. The TwineballInput node is a sample node with its own sample EIS. You cannot use the TwineBall nodes to connect to the external SAP, Siebel, and PeopleSoft EIS systems. Do not use this node in production.

The TwineballInput node is contained in the **WebSphere Adapters** drawer of the message flow node palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

You can use the **mqsisetdbparms** command in the following format to configure an account name with a user name and password for the Twineball adapter.

```
mqsisetdbparms broker name -n adapter name -u user name -p password
```

For example:

```
mqsisetdbparms BRK1 -n eis::TwineballInbound.inadapter -u mqbroker -p ********
```

Look at the following sample to see how to use this node:

- Twineball Example EIS Adapter

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Terminals and properties:
When you have put an instance of the TwineballInput node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view. If you double-click a TwineballInput node, you open the Adapter Connection wizard. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The TwineballInput node terminals are described in the following table.

Terminal	Description
Out	Business object events from the adapter are sent to the Out terminal.
Failure	If an error happens in the TwineballInput node, the message is propagated to the Failure terminal. Information about the error, and business object events can also be propagated to the Failure terminal.
Catch	Business object events are sent to the Catch terminal if they cause an uncaught exception in the message flow. If the Catch terminal is not connected, the retry process is activated to handle the business object.

The following table describes the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file at deployment).

The TwineballInput node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, TwineballInput.	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The TwineballInput node Basic properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Adapter component	Yes	Yes		The name of the adapter component that contains configuration properties for the adapter. Either enter a name of an adapter file or click **Browse** to select an adapter file from the list of files that are available in referenced message set projects.	adapterComponent

The TwineballInput node Routing properties are described in the following table.

Property	M	C	Default	Description
Set destination list	No	No	Selected	This property specifies whether to add the method binding name to the route to label destination list. If you select this check box, the method binding name is added so that you can use a RouteToLabel node in the message flow after the TwineballInput node.
Label prefix	No	No		The prefix to add to the method name when routing to label. Add a label prefix to avoid a clash of corresponding label nodes when you include multiple WebSphere Adapters input nodes in the same message flow. By default, there is no label prefix, so that the method name and label name are identical.

The TwineballInput node Input Message Parsing properties are described in the following table.

Property	M	C	Default	Description
Message domain	No	No	DataObject	The domain that is used to parse the incoming message. By default, the message that is propagated from the TwineballInput node is in the DataObject domain. You cannot specify a different domain.
Message set	Yes	No	Set automatically	The name of the message set in which the incoming message is defined. This field is set automatically from the Adapter component property. If you set this property, then subsequently update the project dependencies to remove this message set reference, a warning is issued. Either update the `Message set` property, or restore the reference to this message set project.
Message type	No	No		The name of the incoming message. The node detects the message type automatically. You cannot set this property.
Message format	No	No		The name of the physical format of the incoming message. You cannot set this property.

The TwineballInput node Transactional properties are described in the following table.

Property	M	C	Default	Description
Transaction mode	No	No	Yes	The transaction mode on this input node determines whether the rest of the nodes in the flow operate under sync point.

The Instances properties of the TwineballInput node are described in the following table. For a full description of these properties, see “Configurable message flow properties” on page 4020.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Additional instances pool	No	Yes	`Use Pool Associated with Message Flow`	The pool from which additional instances are obtained. • If you select `Use Pool Associated with Message Flow`, additional instances are obtained from the message flow value. • If you select `Use Pool Associated with Node`, additional instances are allocated from the node's additional instances based on the number specified in the `Additional instances` property.	componentLevel
Additional instances	No	Yes	`0`	The number of additional instances that the node can start if the `Additional instances pool` property is set to `Use Pool Associated with Node`. By default, no additional instances are given to the node.	additionalInstances

The TwineballInput node Retry properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Retry mechanism	No	No	`Failure`	This property specifies how retry processing is handled when a failure is rolled back to the TwineballInput node. • If you select `Failure`, retry processing is not performed so that you cannot set the remaining properties. • If you select `Short and long retry`, retry processing is performed first at the interval that is specified by the `Short retry interval` property, and if that is unsuccessful, it is then performed at the interval that is specified by the `Long retry interval` property.	
Retry threshold	No	Yes	0	The maximum number of times that retry processing is performed for short retry.	retryThreshold
Short retry interval	No	Yes	0	The interval between short retry attempts.	shortRetryThreshold
Long retry interval	No	Yes	0	The interval between long retry attempts.	longRetryThreshold

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"DataObject parser and domain" on page 1114
Use the DataObject domain to parse and write messages for WebSphere Adapters

and CORBA applications.

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

TwineballRequest node

Use the TwineballRequest node to discover out how WebSphere Adapters nodes work.

This topic contains the following sections:
- "Purpose"
- "Terminals and properties" on page 4956

Purpose:
The TwineballRequest node is provided for educational purposes and helps you to see how the WebSphere Adapters nodes work. The TwineballRequest node is a sample node with its own sample EIS. You cannot use the TwineBall nodes to connect to the external SAP, Siebel, and PeopleSoft EIS systems. Do not use this node in production.

The TwineballRequest node is contained in the **WebSphere Adapters** drawer of the message flow node palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

You can use the **mqsisetdbparms** command in the following format to configure an account name with a user name and password for the Twineball adapter.

```
mqsisetdbparms broker name -n adapter name -u user name -p password
```

For example:

```
mqsisetdbparms BRK1 -n eis::TwineballOutbound.outadapter -u mqbroker -p ********
```

Look at the following sample to see how to use this node:

- Twineball Example EIS Adapter

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Terminals and properties:
When you have put an instance of the TwineballRequest node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view. If you double-click a TwineballRequest node, you open the Adapter Connection wizard. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The TwineballRequest node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts the request business object.
Out	The output terminal to which the response business object is sent if it represents successful completion of the request, and if further processing is required within this message flow.
Failure	If an error happens in the TwineballRequest node, the message is propagated to the Failure terminal. Information about the error, and business object events can also be propagated to the Failure terminal.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk on the panel if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The TwineballRequest node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, TwineballRequest	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The TwineballRequest node Basic properties are described in the following table.

Property	M	C	Default	Description	`mqsiapplybaroverride` command property
Adapter component	Yes	No		The name of the adapter component that contains configuration properties for the adapter. Either enter a name of an adapter file, or click **Browse** to select an adapter file from the list of files that are available in referenced message set projects.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Default method	Yes	Yes		The default method binding to use.	defaultMethod

The TwineballRequest node Response Message Parsing properties are described in the following table.

Property	M	C	Default	Description
Message domain	No	No	DataObject	The domain that is used to parse the response message. By default, the response message that is propagated from the TwineballRequest node is in the DataObject domain. You cannot specify a different domain.
Message set	Yes	No	Set automatically	The name of the message set in which the incoming message is defined. This field is set automatically from the Adapter component property. If you set this property, then subsequently update the project dependencies to remove this message set reference, a warning is issued. Either update the `Message set` property, or restore the reference to this message set project.
Message type	No	No		The name of the response message. The node detects the message type automatically. You cannot set this property.
Message format	No	No		The name of the physical format of the response message. You cannot set this property.

The TwineballRequest node Transactionality properties are described in the following table.

Property	M	C	Default	Description
Transaction mode	No	No	Automatic	This property specifies how updates are handled. If you select `Yes`, updates are performed in a single transaction. If you select `No`, updates are performed independently.

The TwineballRequest node Request properties are described in the following table.

Property	M	C	Default	Description
Method Location	Yes	No	$LocalEnvironment/ Adapter/MethodName	The location of the business method (such as createPurchaseOrder or deletePurchaseOrder) that is used to trigger the TwineballRequest node to perform an action on the external system.
Data Location	Yes	No	$Body	The location in the incoming message tree from which data is retrieved to form the request that is sent from the TwineballRequest node to the EIS.

The TwineballRequest node Result properties are described in the following table.

Property	M	C	Default	Description
Output data location	No	No	$OutputRoot	The message tree location to which the TwineballRequest node sends output.

Property	M	C	Default	Description
Copy local environment	No	No	Selected	This property controls how the local environment is copied to the output message. If you select the check box, at each node in the message flow, a new copy of the local environment is created in the tree, and it is populated with the contents of the local environment from the preceding node. If a node changes the local environment, the upstream nodes do not see those changes because they have their own copies. This behavior might be an issue if you are using a FlowOrder node, or if you use the propagate command on a Compute node. If you clear the check box, each node does not generate its own copy of the local environment, but it uses the local environment that is passed to it by the previous node. If a node changes the local environment, those changes are seen by the upstream nodes.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"DataObject parser and domain" on page 1114
Use the DataObject domain to parse and write messages for WebSphere Adapters and CORBA applications.

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

Related tasks:

"Preparing the environment for WebSphere Adapters nodes" on page 717
Before you can use the WebSphere Adapters nodes, you must set up the broker runtime environment so that you can access the Enterprise Information System (EIS).

"Developing message flows that use WebSphere Adapters" on page 2033
For information about how to develop message flows that use WebSphere Adapters, see the following topics.

"Deploying a message flow that uses WebSphere Adapters" on page 3240
Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

"Debugging message flows that contain WebSphere Adapters nodes" on page 3192
You can use various methods to monitor message flows that include WebSphere Adapters nodes.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

Validate node

Use the Validate node to check that the message that arrives on its input terminal is as expected. You can use this node to check that the message has the expected message template properties, and to check that the content of the message is correct by selecting message validation.

This topic contains the following sections:

Purpose:
The checks that you can perform depend on the domain of the message.

Check	Domain
Check message domain	All domains
Check message set	XMLNSC, MRM, and IDOC only
Check message type	MRM only
Validate message body	XMLNSC, MRM and IDOC only

You can check the message against one or more of message domain, message set, or message type. The property is checked only if you select its corresponding check box, which means that a property that contains an empty string can be compared.

You can check the content of the message by giving a value to the `Validate` property. Validation takes place if the `Validate` property is set to a value other than `None`, which is the default value.

For validation failures to be returned to the Validate node from the parser, set the `Failure Action` property to either `Exception` or `Exception List`. Otherwise, validation failures are just logged.

If all the specified checks pass, the message is propagated to the Match terminal of the node.

If any of the checks fail, the message is propagated to the Failure terminal. If the Failure terminal is not connected to some failure handling processing, an exception is generated.

The Validate node replaces the Check node, which is deprecated in WebSphere Message Broker Version 6.0 and subsequent releases. The Validate node works in the same way as the Check node, but it has additional Validation properties to allow the validation of message content by parsers that support that capability.

The Validate node is contained in the **Validation** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:
Use the Validate node to confirm that a message has the correct message template properties, and has valid content, before propagating the message to the rest of the flow. Subsequent nodes can then rely on the message being correct, without doing their own error checking.

You can also use the Validate node to ensure that the message is routed appropriately through the message flow. For example, configure the node to direct a message that requests stock purchases through a different route from that required for a message that requests stock sales.

Another routing example is the receipt of electronic messages from your staff at your head office. These messages are used for multiple purposes (for example, to request technical support or stationery, or to advise you about new customer leads). These messages can be processed automatically because your staff complete a standard form. If you want these messages to be processed separately from other messages that are received, use the Validate node to ensure that only staff messages that have a specific message type are processed by this message flow.

Terminals and properties:
When you have put an instance of the Validate node into a message flow, you can configure it. For more information, see "Configuring a message flow node" on page 1503. The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The terminals of the Validate node are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the message is routed if the incoming message does not match the specified properties.
Match	The output terminal to which the message is routed if the incoming message matches the specified properties.

The following tables describe the properties of the node. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Description properties of the Validate node are described in the following table.

Property	M	C	Default	Description
Node name	No	No	Validate	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The Validate node Basic properties are described in the following table.

Property	M	C	Default	Description
Domain	No	No		The name of the domain. Select one of the following values from the list of the `Domain` property: • XMLNSC • SOAP • DataObject • XMLNS • JMSMap • JMSStream • MIME • MRM • BLOB You can also specify a user-defined parser, if appropriate.
Check domain	Yes	No	Cleared	If you select this check box, the incoming message is checked against the `Domain` property.
Set	No	No		The name or identifier of the message set to which the incoming message belongs. If you are using the XMLNSC, DataObject, SOAP, MRM, or IDOC parser and want to check that the incoming message belongs to a particular message set, select `Check set` and select one of the values from the list of the `Set` property. This list is populated when you select XMLNSC, DataObject, SOAP, MRM, or IDOC as the message domain. Leave `Set` clear for the other parsers. If you set this property, then subsequently update the project dependencies to remove this message set reference, a warning is issued. Either update the `Set` property, or restore the reference to this message set project.
Check set	Yes	No	Cleared	If you select the check box, the incoming message is checked against the `Set` property. If you are using the XMLNSC,DataObject, SOAP, MRM, or IDOC parser and want to check that the incoming message belongs to a particular message model, select `Check set` and select one of the values from the list of the `Set` property.
Type	No	No		The message name. If you are using the MRM parser and want to check that the incoming message is a particular message type, select `Check type` and enter the name of the message in the `Type` property. Leave `Type` clear for the other parsers.
Check type	Yes	No	Cleared	If you select the check box, the incoming message is checked against the `Type` property. If you are using the MRM parser and want to check that the incoming message is a particular message type, select `Check type` and enter the name of the message in the `Type` property.

The Validation properties of the Validate node are described in the following table.

If you are using the XMLNSC, DataObject, SOAP, MRM, or IDOC parser and want to validate the body of messages against the message set, select the required validation properties on the **Validation** tab. For more details, see “Validating messages” on page 1478 and “Validation properties” on page 4169.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Validate	No	Yes	None	This property controls whether validation takes place. Valid values are None, Content and Value, Content, and Inherit.	validateMaster
Failure Action	No	No	Exception	This property controls what happens if validation fails. You can set this property only if you set Validate to Content or Content and Value. Valid values are User Trace, Local Error Log, Exception, and Exception List.	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Parsers" on page 1072
A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from the internal message tree representation.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"Specifying namespaces in the Message Type property" on page 1208
When using the MRM domain, the *Message Type* property is used to specify the name of the message.

Related tasks:

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Validating messages" on page 1478
The broker provides validation based on the message model for predefined messages.

Related reference:

"Validation properties" on page 4169
You can control validation by setting properties on the Validate and Parser Options

tabs for the nodes that are listed in the following table.

"Check node" on page 4318
Use the Check node to compare the template of a message that is arriving on its input terminal with a message template that you supply when you configure the Check node.

Warehouse node

Use the Warehouse node to interact with a database in the specified ODBC data source.

This topic contains the following sections:
- "Purpose"
- "Using this node in a message flow"
- "Terminals and properties" on page 4964

Purpose:
The Warehouse node is a specialized form of the Database node that stores the entire message, parts of the message, or both, in a table within the database. You define what is stored by creating mappings that use the data from the input message to identify the action that is required.

You can use the Warehouse node for various purposes. For example:
- To maintain an audit trail of messages that are passing through the broker
- For offline or batch processing of messages that have passed through the broker (data mining)
- As a source from which to reprocess selected messages in the broker

Use standard database query and mining techniques to retrieve messages that you have stored in the warehouse. (No explicit support is provided by WebSphere Message Broker.)

You must have created or identified the following items:
- Input data in the form of a message set and message
- An ODBC connection to the database
- A database and database table to store the message
- At least two columns in the table: one for the binary object (the message), and one for the timestamp

The Warehouse node is contained in the **Database** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:
When you use the Warehouse node, you can store the following elements in the database that is associated with the node:
- The entire message, optionally with an associated timestamp. The message is stored as a binary object, with the timestamp in a separate column. This option has two advantages:
  - There is no need to decide beforehand how to use the warehoused data because after the data is stored, it can be retrieved, and data mining tools applied.

  - You do not need to define a specific database schema for every type of message that might pass through the broker. In a complex system, many different message types might be processed, and the time involved in defining a unique schema for each message type can become prohibitive. You can precede each Warehouse node with a Compute node that converts each message into a canonical warehouse format with a common schema, or you can store the whole message as a binary object.
- Selected parts of the message, optionally with an associated timestamp, which requires a defined database schema for that message type. The message is mapped to true type so, for example, a character string in the message is stored as a character string in the database.

Terminals and properties:
When you have put an instance of the Warehouse node into a message flow, you can configure it. For more information, see "Configuring a message flow node" on page 1503. The properties of the node are displayed in the Properties view. (If you double-click a Warehouse node, you open the New Message Map dialog box.) All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The terminals of the Warehouse node are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the input message is propagated if a failure is detected during the computation. If you have selected `Treat warnings as errors`, the node propagates the message to this terminal even if the processing completes successfully.
Out	The output terminal that outputs the message following the execution of the database statement.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Warehouse node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	Warehouse	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The Warehouse node Basic properties are described in the following table.

Property	M	C	Default	Description	**mqsiapplybaroverride command property**
Data source	No	Yes		The ODBC data source name of the database that contains the tables to which you refer in the mappings that are associated with this node (identified by the `Field mapping` property). This name identifies the appropriate database as it is known on the system on which this message flow is to run. The broker connects to this database with user ID and password information that you have specified on the **mqsisetdbparms** command.	dataSource

Property	M	C	Default	Description	mqsiapplybaroverride command property
Field mapping	Yes	No	`Warehouse`	The name of the mapping routine that contains the statements that are to be executed against the database or the message tree. By default, the name that is assigned to the mapping routine is identical to the name of the mappings file in which the routine is defined. The default name for the file is the name of the message flow concatenated with the name of the node when you include it in the message flow (for example, `MFlow1_Warehouse.msgmap` for the first Warehouse node in message flow MFlow1). You cannot specify a value that includes spaces. If you click **Browse** next to this entry field, a dialog box displays that lists all available mapping routines that can be accessed by this node. Select the routine that you want and click **OK**; the routine name is set in `Field mapping`. To work with the mapping routine that is associated with this node, double-click the node, or right-click the node and select **Open Mappings**. If the mapping routine does not exist, it is created for you with the default name in the default file. If the file exists, you can also open file *`flow_name_node_name`*`.msgmap` in the Broker Development view. The content of the mapping routine determines what is stored in the database, and in what format. You can, for example, store all or just a part of each message. You can also store the data as binary data, or store each field in the same format as it is in the message (for example, a character field in the message is stored as character in the database). A mapping routine is specific to the type of node with which it is associated; you cannot use a mapping routine that you have developed for a Warehouse node with any other node that uses mappings (for example, a DataInsert node). If you create a mapping routine, you cannot call it from any other mapping routine, although you can call it from an ESQL routine. For more information about working with mapping files, and defining their content, see “Using message mappings” on page 2228.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Transaction	Yes	No	Automatic	The transaction mode for the node. Select the value that you require: • If you select Automatic (the default), the message flow, of which the Warehouse node is a part, is committed if it is successful; that is, the actions that you define in the mappings are taken and the message continues through the message flow. If the message flow fails, it is rolled back. Therefore, selecting Automatic means that the ability to commit or roll back the action of the Warehouse node on the database depends on the success or failure of the entire message flow. • If you select Commit, any uncommitted actions that are taken in this message flow are committed on the database that is connected to this node, irrespective of the success or failure of the message flow as a whole. The changes to the database are committed even if the message flow itself fails.	
Treat warnings as errors	Yes	No	Cleared	For database warning messages to be treated as errors, and the node to propagate the output message to the Failure terminal, select Treat warnings as errors. The check box is cleared by default. When you select the check box, the node handles all positive return codes from the database as errors and generates exceptions in the same way as it does for the negative, or more serious, errors. If you do not select the check box, the node treats warnings as typical return codes, and does not raise any exceptions. The most significant warning raised is not found, which can be handled as a typical return code safely in most circumstances.	
Throw exception on database error	Yes	No	Selected	For the broker to generate an exception when a database error is detected, select Throw exception on database errors. The check box is selected by default. If you clear the check box, you must handle the error in the message flow to ensure the integrity of the broker and the database; the error is ignored if you do not handle it through your own processing, because you have chosen not to invoke the default error handling by the broker; for example, you can connect the Failure terminal to an error processing subroutine.	

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Client application programming interfaces" on page 1038
You can configure the nodes in your message flows to customize the behavior of those nodes by using one or more of the supported programming interfaces.

Related tasks:

"Deciding which nodes to use" on page 1457
WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

"Configuring transactionality for message flows" on page 1290
A message flow runs in a single transaction, which is started when data is received by an input node, and can be committed or rolled back when all processing has completed.

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Using message mappings" on page 2228
Message mappings define the blueprint for creating a message.

"Editing configurable properties" on page 3227
You can edit configurable properties in the deployment descriptor file (`broker.xml`) of your broker archive.

Related reference:

"**mqsichangebroker** command" on page 3723
Use the **mqsichangebroker** command to change one or more of the configuration parameters of the broker.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

"DataInsert node" on page 4386
Use the DataInsert node to interact with a database in the specified ODBC data source.

XSLTransform node

Use the XSLTransform node to transform an XML message to another form of message, according to the rules provided by an XSL (Extensible Stylesheet Language) style sheet, and to set the `Message domain`, `Message set`, `Message type`, and `Message format` for the generated message.

This node was formerly known as the XMLTransformation node.

This topic contains the following sections:
- "Purpose"
- "Using this node in a message flow" on page 4969
- "Configuring the XSLTransform node" on page 4970
- "Terminals and properties" on page 4970

Purpose:

You can specify the location of the style sheet to apply to this transformation in three ways:

- Use the content of the XML data within the message itself to transform the message according to a style sheet that the message itself defines.
- Set a value in the LocalEnvironment folder. You must set this value in a node that precedes the XSLTransform node (for example, a Compute node). You can therefore use a variety of inputs to determine which style sheet to use for this message, such as the content of the message data, or a value in a database.
- Use node properties to ensure that the transformation that is defined by this single style sheet is applied to every message that is processed by this node.

An XSLT (Extensible Stylesheet Language for Transformations) compiler is used for the transformation if the style sheet is not embedded within the message, and the node cache level (node property Stylesheet Cache Level) is greater than zero. If the XSLT is cached, the performance is improved because the XSLT is not parsed every time it is used.

If the prologue of the input message body contains an XML encoding declaration, the XSLTransform node ignores the encoding, and always uses the CodedCharSetId in the message property folder to decode the message.

The XSLTransform node is contained in the **Transformation** drawer of the palette, and is represented in the WebSphere Message Broker Toolkit by the following icon:

Using this node in a message flow:
For an example of how to use this node, consider two news organizations that exchange information on a regular basis. One might be a television station, the other a newspaper. Although the information is similar, the vocabulary that is used by the two is different. This node can transform one format to the other by applying the rules of the specified style sheet. If you specify the style sheet in the message (either the XML data or the LocalEnvironment), the same node can perform both transformations.

You cannot use relative path external entities that are defined in the DTD of input messages (for example, `<!DOCTYPE note [<!ENTITY chap1 SYSTEM "chap1.xml">]>`). Use an absolute path definition.

The XSLTransform node supports a number of local environment message tree variables, which you can use to dynamically alter the values that are set in the node's properties. For more details, see “Using local environment variables to set properties” on page 4976.

You can use style sheets in two different ways with the XSLTransform node. For more details, see “Deployed and non-deployed style sheets” on page 4978.

If you have large XML messages and receive an `out of memory` error, use the **`mqsireportproperties`** command to see the current value of the Java Heap size for the XSLT engine:

```
mqsireportproperties brokerName -e executionGroupLabel
                                -o ComIbmJVMManager -n jvmMaxHeapSize
```

Use the **mqsichangeproperties** command to increase the Java Heap size:

```
mqsichangeproperties brokerName -e executionGroupLabel
                                -o ComIbmJVMManager -n jvmMaxHeapSize -v newSize
```

In the previous examples, replace *brokerName*, *executionGroupLabel*, and *newSize* with the appropriate values.

The value that you choose for *newSize* depends on the amount of physical memory that your computer has, and how much you are using Java. A value in the range 512 MB (536870912) to 1 GB (1073741824) is typically sufficient.

Look at the following sample for more details about how to use the XSLTransform node:

- XSL Transform

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Configuring the XSLTransform node:
When you have put an instance of the XSLTransform node into a message flow, you can configure it; see . The properties of the node are displayed in the Properties view.

All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

Terminals and properties:
The XSLTransform node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts the message for processing by the node.
Failure	The output terminal to which the original message is routed if an error is detected during transformation.
Out	The output terminal to which the successfully transformed message is routed.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The XSLTransform node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The XSLTransform node Stylesheet properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Stylesheet name	No	Yes		The name of the style sheet, which is used if the style sheet specification is searched for in node properties. To specify a style sheet by using node properties, enter the required value for Stylesheet name. Specify a principal style sheet using one of the following methods: • Click **Browse** next to `Stylesheet name`. The identified principal style sheet and all its relatively-referenced descendant style sheets are added automatically to the BAR file when you add a message flow to a BAR file (if both they and their parent style sheets are available). • To identify an already deployed, or ready to be deployed, style sheet, use the `Stylesheet name` property, and leave the `Stylesheet directory` property blank. • In the Message Flow editor, drag an `.xslt` file onto the XSLTransform node; the `Stylesheet name` is set automatically.	stylesheetName

The XSLTransform node Advanced properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Stylesheet directory	No	Yes		The path where the style sheet is located. This path is used by all location methods. If the style sheet identification is fully qualified, the `Stylesheet directory` property is ignored; otherwise, the value that you set in this property is added to the beginning of the specification, regardless of where it is found.	stylesheetPath

Property	M	C	Default	Description	mqsiapplybaroverride command property
Stylesheet cache level	No	No	5	The number of compiled or parsed style sheets that are stored in this instance of the node. Enter a positive integer between zero (0) and 100. The default value is 5. If you set this property to zero (0), no style sheet is cached, and style sheets are interpreted rather than compiled. If your message flow does not set the style sheet dynamically by using the local environment, the same style sheet is always used, and any value greater than zero ensures that it is compiled. If your message flow does set the style sheet dynamically, you can tune the value to improve performance. If you set this property to a character other than a positive integer, a flow configuration exception error message is issued. The style sheet cache is retained for the life of the node, and is cleared when the node is deleted from the flow, when the flow is deleted, or when the execution group is stopped. If you change a cached style sheet (by redeploying or replacing the file in the file system), the XSLTransform node that is holding the cache replaces the cached version with the modified (latest) version before a new message is processed. If a deployed style sheet is redeployed, or an external style sheet is replaced in the file system, the update is detected but you cannot replace a style sheet that is deployed in a BAR file by changing the deployed style sheet on disk. To change a style sheet that is deployed in a BAR file, you must redeploy the BAR file. If you are changing several style sheets, stop the relevant message flows before you make any changes. If you do not stop the relevant message flows before you make the changes, the order of the changes cannot be guaranteed by running message flows, which might cause an incompatibility between the style sheets that are changed. Use the **mqsireload** command to reload a style sheet; however, this command does not prevent incompatibility. Consider performance when you set this property. Typically, when you set this property to a number greater than the default value of 5, performance is faster because style sheet re-compilation is less likely. However, cached style sheets use Java virtual machine (JVM) heap space; if you keep too many cached style sheets, the JVM is likely to slow. In addition, if the cached style sheets are not used regularly and you set this property to a large number, performance can be affected because compiling style sheets increases processor usage. Therefore, to decide on the most suitable value for this property, you must balance how many style sheets you use with the size of the style sheets, the size of the JVM heap space, and your usage pattern. For more information about the JVM heap space, see "JVM heap sizing" on page 3269.	

The XSLTransform node Output Message Parsing properties are described in the following table.

Property	M	C	Default	Description
Message domain	No	No	BLOB	The message domain that is associated with the output message. To associate a specific parser with the output message, specify the new domain in Message domain. The default value is BLOB. This domain is applied to the output message. Alternatively, use Inherit to associate the parser that owned the input message. XML is deprecated; use XMLNSC instead. You can also specify a user-defined parser if appropriate.
Message set	No	No		The message set that is associated with the output message. If you are using the MRM parser, or the XMLNSC parser in validating mode, select the Message set that you want to use. This list is populated with available message sets when you select MRM or XMLNSC as the domain. If you set this property, then subsequently update the project dependencies to remove this message set reference, a warning is issued. Either update the Message set property, or restore the reference to this message set project.
Message type	No	No		The message type that is associated with the output message. If you are using the MRM parser, select the correct message from the list in Message type. This list is populated with messages that are defined in the Message set that you have selected.
Message format	No	No		The message format that is associated with the output message. If you are using the MRM parser, select the XML physical format for the output message from the list in Message format. This list includes all the physical formats that you have defined for this Message set.
Character set	No	No		The numeric value of the character set for the output message. To specify a character set for the output message by using node properties, specify the required value for Character set. The value that you specify must be numeric; for example, specify 1200 to encode the output message as UTF-16.

The XSLTransform node Parser Options are described in the following table.

Property	M	C	Default	Description
Parse timing	No	No	On Demand	This property controls when an output message is parsed. Valid values are On Demand, Immediate, and Complete. Parse timing is, by default, set to On Demand, which causes parsing of the message to be delayed. To cause the message to be parsed immediately, see “Parsing on demand” on page 4173.
Build tree using XML schema data types	No	No	Cleared	This property controls whether the XMLNSC parser creates syntax elements in the message tree with data types taken from the XML Schema. You can select this property only if you set the Validate property to Content or Content and Value. For more information, see “Manipulating messages in the XMLNSC domain” on page 2546.
Use XMLNSC compact parser for XMLNS domain	No	No	No	This property controls whether the XMLNSC Compact Parser is used for output messages in the XMLNS Domain. If you set this property, the message data appears under XMLNSC in nodes that are connected to the output terminal when the input MQRFH2 header or Domain is XMLNS.

The XSLTransform node Validation properties are described in the following table. Set the validation properties for the parser to validate the body of messages against the Message set. (If a message is propagated to the Failure terminal of the node, it is not validated.) For more details, see “Validating messages” on page 1478 and “Validation properties” on page 4169.

Property	M	C	Default	Description
Validate	No	Yes	`None`	This property controls whether validation takes place of the output message. Valid values are `None`, `Content`, `Content and Value`, and `Inherit`.
Failure action	No	No	`Exception`	This property controls what happens if validation of the output message fails. You can set this property only if you set `Validate` to `Content and Value` or `Content`. Valid values are `User Trace`, `Local Error Log`, `Exception`, and `Exception List`.

The XSLTransform node Detail Trace properties are described in the following table.

Property	M	C	Default	Description
Trace setting	Yes	No	`Off`	This property is deprecated. Start user trace instead. The user trace contains the same XSL trace information. If you set this property, it does not affect user trace. In previous versions of WebSphere Message Broker, this property controls whether tracing is on or off. If tracing is on, low level tracing is recorded in a file.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 3327 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Related concepts:

"Parsers" on page 1072
A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from the internal message tree representation.

"Specifying namespaces in the Message Type property" on page 1208
When using the MRM domain, the *Message Type* property is used to specify the name of the message.

"Multipart messages" on page 1191
A *multipart message* contains one or more other messages within its structure. The contained message is sometimes referred to as an *embedded message*.

"Local environment tree structure" on page 1056
The local environment tree is a part of the logical message tree in which you can store information while the message flow processes the message.

Related tasks:

"Handling errors in message flows" on page 2823
The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

"Adding files to a broker archive" on page 3223
To deploy files to an execution group, include them in a broker archive (BAR) file.

"Setting the JVM heap size" on page 3254
When you start an execution group, it creates a Java virtual machine (JVM) for executing a Java user-defined node.

Related reference:

"User trace" on page 6873
User trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Administration log. User trace is inactive by default; you must activate it explicitly by using a command, or by selecting options in the WebSphere Message Broker Toolkit.

"Adding keywords to XSL style sheets"
Embedded keywords in an XSL style sheet; their location is not restricted. You can also add a keyword as an XML comment.

"**mqsichangeproperties** command" on page 3756
Use the **mqsichangeproperties** command to modify broker properties and properties of broker resources.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"**mqsireload** command" on page 3909
Use the **mqsireload** command to request the broker to stop and restart execution groups.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"RegistryLookup node" on page 4646
Use the RegistryLookup node to retrieve any type of entity held in the WebSphere Service Registry and Repository (WSRR). The entities that match the specified search criteria are stored in the local environment. The input message is not modified.

Adding keywords to XSL style sheets:

Embedded keywords in an XSL style sheet; their location is not restricted. You can also add a keyword as an XML comment.

XML comments must have the following format:

```
$MQSI keyword = value MQSI$
```

The example shows how to add the keyword of **author** with the value `John` to an XML style sheet:

```
<?xml version="1.0" encoding="UTF-8">
<!-- $MQSI author = John MQSI$ -->
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="text" encoding="UTF-8"/>
<xsl:template match="/">
<xsl:value-of select="message"/>
</xsl:template>
</xsl:stylesheet>
```

Restrictions within keywords

Do not use the following characters within keywords, because they might cause unpredictable behavior:

```
^ $ . | \ < > ? + * = & [ ] ( )
```

You can use these characters in the values that are associated with keywords; for example:

- `$MQSI RCSVER=$id$ MQSI$` is acceptable
- `$MQSI $name=Fred MQSI$` is not acceptable

Related concepts:

"Message flow version and keywords" on page 1445
When you are developing a message flow, you can define the version of the message flow as well as other key information that you want to be associated with it.

"Version and keyword information for deployable objects" on page 1443
Use the Broker Archive editor to view the version and keyword information of deployable objects.

Related reference:

"Guidance for defining keywords" on page 4017
You can add extra information to an object in the form of one or more keywords.

"XSLTransform node" on page 4968
Use the XSLTransform node to transform an XML message to another form of message, according to the rules provided by an XSL (Extensible Stylesheet Language) style sheet, and to set the `Message domain`, `Message set`, `Message type`, and `Message format` for the generated message.

Using local environment variables to set properties:

The XSLTransform node supports a number of local environment message tree variables, which you can use to dynamically alter the values that are set in the node's properties.

The following table lists each local environment variable against the name of the node property that it overrides:

Local environment variable name	Node property name
XSL.StyleSheetName	Stylesheet name
XSL.MessageDomain	Message domain
XSL.MessageSet	Message set
XSL.MessageType	Message type
XSL.MessageFormat	Message format
XSL.OutputCharSet	Character set

This node searches for the name of the style sheet to be used by interrogating, in the following order:

1. The input message.

   The node searches the message XML data for information about the location of the style sheet. For example, the XML data might contain:

   ```
   <?xml-stylesheet type="text/xsl" href="me.xsl"?>
   ```

and "me.xsl" is then used as the name of the style sheet.

2. The local environment.

   If no style sheet name is found in the input message, the node searches the local environment that is associated with the current message for style sheet information stored in an element called XSL.StyleSheetName.

   This node was available in Version 6.0, and element ComIbmXslXmltStylesheetname was used for the name of the style sheet, therefore the current node checks both elements. If both are present, the value in XSL.StyleSheetName takes precedence.

3. The node properties.

   If no style sheet name is found in the input message or local environment, the node uses the `Stylesheet name` and `Stylesheet directory` properties to determine the correct values.

The node searches for the message domain, message set, message type, and message format to use for the output message by interrogating, in the following order:

1. The local environment.

   The node searches the local environment that is associated with the current message for message domain, message set, message type, and message format information stored in elements called XSL.MessageDomain, XSL.MessageSet, XSL.MessageType, and XSL.MessageFormat.

2. The node's properties.

   If no message domain, message set, message type, or message format information is found in these local environment variables, the node uses the `Message domain`, `Message set`, `Message type`, and `Message format` properties to determine the correct values.

If the node cannot determine the message domain from either XSL.MessageDomain or the `Message domain` property, the default value of BLOB is used. No default values exist for message set, message type, and message format.

The node searches for the character set to use for the output message by interrogating, in the following order:

1. The local environment.

   The node searches the local environment that is associated with the current message for character set information stored in an element called XSL.OutputCharSet; for example, to encode the output of the transformation as UTF-8, enter the value `1208` as a string in this element.

   This node was available in Version 6.0, and element ComIbmXslXmltOutputcharset was used for the output character set, therefore the current node checks both elements. If both are present, the value in XSL.OutputCharSet takes precedence.

2. The node's properties.

   If no character set information is found in the local environment, the node uses the `Character set` property to determine the correct value.

   If you set a value for `Character set`, the value that you enter must be numeric; for example, to encode the output of the transformation as UTF-16, enter `1200`.

If the node cannot determine the output character set from either of these two sources, because either no value is set or the selection priorities are set to zero, the default value of 1208 (UTF-8) is used.

Be aware of the following factors if the input to the XSLTransform node is generated from the XMLNSC parser or the MRM parser. The XMLNSC parser discards certain information in XML documents, such as processing instructions and comments, if you do not set properties to retain this information in a preceding node. To ensure that the XSLTransform node transforms the message correctly, set the `Retain mixed content`, `Retain comments`, and `Retain processing instructions` properties correctly on the preceding node (for example, an MQInput node). The MRM parser also discards this information, but you cannot retain information for this parser, therefore avoid using the MRM domain if such information is vital to your transformation.

Related concepts:

"Parsers" on page 1072
A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from the internal message tree representation.

"Local environment tree structure" on page 1056
The local environment tree is a part of the logical message tree in which you can store information while the message flow processes the message.

Related reference:

"User trace" on page 6873
User trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Administration log. User trace is inactive by default; you must activate it explicitly by using a command, or by selecting options in the WebSphere Message Broker Toolkit.

"Adding keywords to XSL style sheets" on page 4975
Embedded keywords in an XSL style sheet; their location is not restricted. You can also add a keyword as an XML comment.

"XSLTransform node" on page 4968
Use the XSLTransform node to transform an XML message to another form of message, according to the rules provided by an XSL (Extensible Stylesheet Language) style sheet, and to set the `Message domain`, `Message set`, `Message type`, and `Message format` for the generated message.

Deployed and non-deployed style sheets:

You can use style sheets in two different ways with the XSLTransform node.

Deployed style sheets
: Deployed style sheets are style sheets that you import into a broker archive (BAR) file and deploy to target systems. Deployed style sheets are managed by the broker. A *principal style sheet* is the root style sheet that is referenced in a message flow; for example, a reference to a principal style sheet in the Eclipse workspace, `C:\\project1\a\b.xsl` must be specified as `a/b.xsl` (or `./a/b.xsl`). A principal style sheet can reference (include or import) its descendent style sheets.

Non-deployed style sheets
: Non-deployed style sheets are style sheets that you store in a location where the XSLTransform node can access them. Non-deployed style sheets are not managed by the broker.

Deployment of deployed style sheets or XML files

Before you can configure the XSLTransform node, you must understand how to work with style sheets. A style sheet can refer to both another XML file and a style sheet. To use deployed style sheets or XML files:

1. Make sure that the files have the correct file name extensions.

   Style sheets that are to be deployed must have either `.xsl` or `.xslt` as their file extension, and XML files to be deployed must have `.xml` as their file extension.

2. Import the files into the Eclipse workspace.

   Import into an Eclipse workspace project all style sheets and XML files that are to be deployed. Put location-dependent descendant style sheets, or XML files that are to be deployed, in the correct directory structure relative to their parent style sheets. Do not put in the Eclipse workspace location-dependent descendants that you do not want to deploy.

3. Make sure that all references to the files are relative.

   Typically, all references to a deployed style sheet must be made relative, no matter where they are displayed. A reference to a principal style sheet must be made relative to the root of the relevant Eclipse workspace project.

   The only exception is when you specify a principal style sheet as the `Stylesheet name` property on an XSLTransform node; you can use an absolute path that points to the correct directory structure in the Eclipse workspace. If the principal style sheet is found, the system resets the node property automatically to the correct relative value.

   The system also performs an automatic deployment of the principal style sheet, together with all of its location-dependent descendant style sheets that are available in the relevant Eclipse workspace project. All references to the location-dependent descendant style sheets (or XML files) of a principal style sheet must be made relative to the location of their parent style sheets. For example, if style sheet `//project1/a/b.xsl` references style sheet `//project1/a/c/d.xsl`, the reference must be changed to `c/d.xsl` (or `./c/d.xsl`).

4. Handle non-deployed child style sheets or XML files.

   Style sheets can refer to other style sheets. If you have a relatively-referenced child style sheet (or XML file) that is not to be deployed, yet its parent is, make sure that the child style sheet is placed in the correct location under *workpath*`/XSL/external` (*workpath*`/XML/external`), where *workpath* is the full path to the working directory of the broker. You can use the `MQSI_WORKPATH` environment variable to find the location of the workpath on your system; for example, on Windows XP systems, the default workpath is `MQSI_WORKPATH=C:\Documents and Settings\All Users\Application Data\IBM\MQSI`.

   A broker automatically associates the execution group deployed storage tree, *workpath*`/`**`XSL`**`/external`, and *workpath*`/`**`XML`**`/external` tree, together. Therefore if, for example, the document `b/c.xml` is not found in the broker's deployed storage, the broker automatically searches for a reference to it in the *workpath*`/XML/external/a/b` directory in the deployed principal style sheet `a/style.xsl`. Relative path references must also be used for files that have been deployed but which are not available in the workspace.

5. Deploy the files.

   Deploy manually only those style sheets or XML files that are not picked up by the system (the WebSphere Message Broker Toolkit provides warnings about

these files). If you click **Browse** for the node, or provide the full path of the location of the style sheet in the Eclipse workspace, the style sheet is included automatically in the BAR file.

To deploy manually, add the files to be deployed to a broker archive. For more information, see "Adding files to a broker archive" on page 3223 and "Adding keywords to XSL style sheets" on page 4975.

For every execution group that uses the XSLTransform node, perform one of the following actions:

- Include the style sheet in the *workpath*/XSL/external directory on the broker; do not include the style sheet in the BAR file.

  If a style sheet in the *workpath*/XSL/external directory shares the same path and name with a deployed style sheet, the deployed style sheet is used.
- Include the style sheet in the BAR file and deploy the BAR file. If multiple BAR files include the same style sheet name, the style sheet from the last BAR file that was deployed is used.
- Deploy the style sheet in its own BAR file. If the BAR files use XSLTransform nodes, but do not include the style sheet, the WebSphere Message Broker Toolkit issues warning messages.

Related concepts:

"Parsers" on page 1072
A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from the internal message tree representation.

"Local environment tree structure" on page 1056
The local environment tree is a part of the logical message tree in which you can store information while the message flow processes the message.

Related reference:

"User trace" on page 6873
User trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Administration log. User trace is inactive by default; you must activate it explicitly by using a command, or by selecting options in the WebSphere Message Broker Toolkit.

"Adding keywords to XSL style sheets" on page 4975
Embedded keywords in an XSL style sheet; their location is not restricted. You can also add a keyword as an XML comment.

"XSLTransform node" on page 4968
Use the XSLTransform node to transform an XML message to another form of message, according to the rules provided by an XSL (Extensible Stylesheet Language) style sheet, and to set the Message domain, Message set, Message type, and Message format for the generated message.

Transformation interfaces

View the reference material associated with the different ways in which you can transform messages in message flows.

- "Message mappings" on page 4981
- "ESQL reference" on page 5019
- "Java reference" on page 5312
- "PHP API" on page 5313

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Processing messages" on page 1021
Process your business messages and data by interacting with a broker, which you can configure to provide services and to communicate with other applications and systems.

"Transforming and enriching messages" on page 2227
Transform and enrich messages by using one or more of the techniques described in this section.

Message mappings

Edit and configure message maps using the Message Mapping editor.

This section contains topics that provide reference information about message mapping:

- "Message Mapping editor"
  - Source pane
  - Target pane
  - Edit pane
  - Spreadsheet pane
- "Mapping node" on page 4994
  - Syntax
  - Functions
  - Casts

Related concepts:

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

Related tasks:

"Using message mappings" on page 2228
Message mappings define the blueprint for creating a message.

Related reference:

"Mapping node" on page 4571
Use the Mapping node to construct one or more new messages and populate them with various types of information.

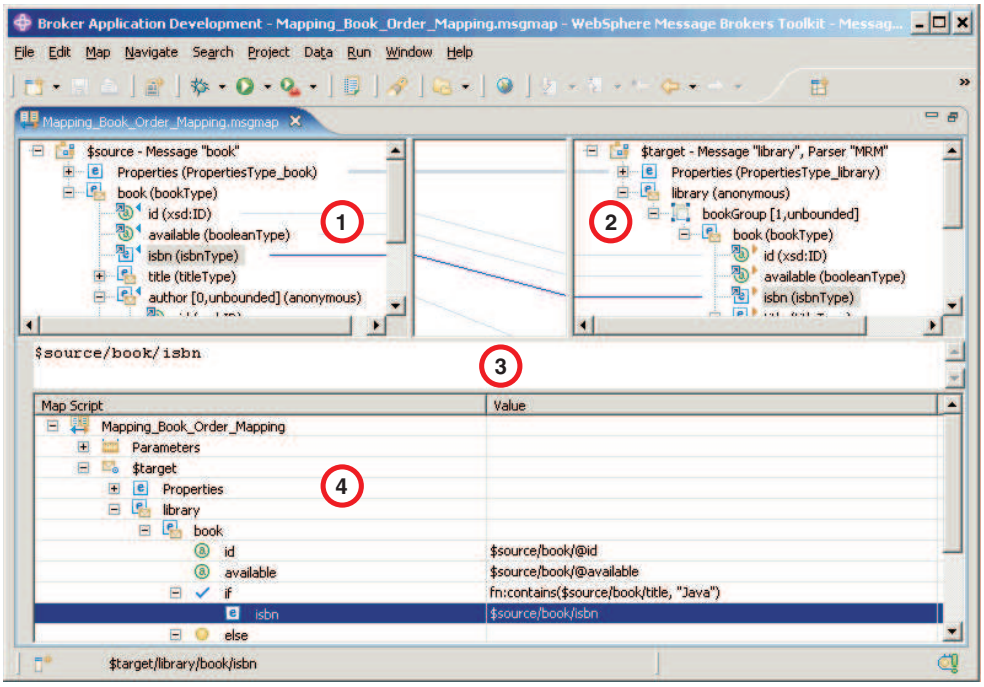
Message Mapping editor:

You use the Message Mapping editor to create and edit message mappings.

You can use the Message Mapping editor to set values for:

- The message destination
- Message headers
- Message content

Here is an example of the Message Mapping editor. There are separate panes for working with sources, targets and expressions, as well as a spreadsheet view.



1. **Source pane:** displays a source message or database table
2. **Target pane:** displays a target message
3. **Edit pane:** displays the expression to be used to derive the target element value
4. **Spreadsheet pane:** displays a summary of the mappings in spreadsheet columns (each target field and its value)

Use the Message Mapping editor to perform various mapping tasks.

Wizards and dialog boxes are provided for tasks such as adding mappable elements, working with ESQL, and working with submaps. Mappings that are created with the Message Mapping editor are automatically validated and compiled, ready for adding to a broker archive (BAR) file, and subsequent deployment to WebSphere Message Broker.

Related tasks:

"Using message mappings" on page 2228
Message mappings define the blueprint for creating a message.

"Configuring message mappings" on page 2237
Use the Message Mapping editor to configure a message mapping.

Related reference:

"Message Mapping editor Source pane" on page 4983
Details of the elements present in the Source pane of the Message Mapping Editor.

"Message Mapping editor Target pane" on page 4987
Details of the elements present in the Target pane of the Message Mapping Editor.

"Message Mapping editor Edit pane" on page 4990
Details of how you use the Edit pane of the Message Mapping Editor.

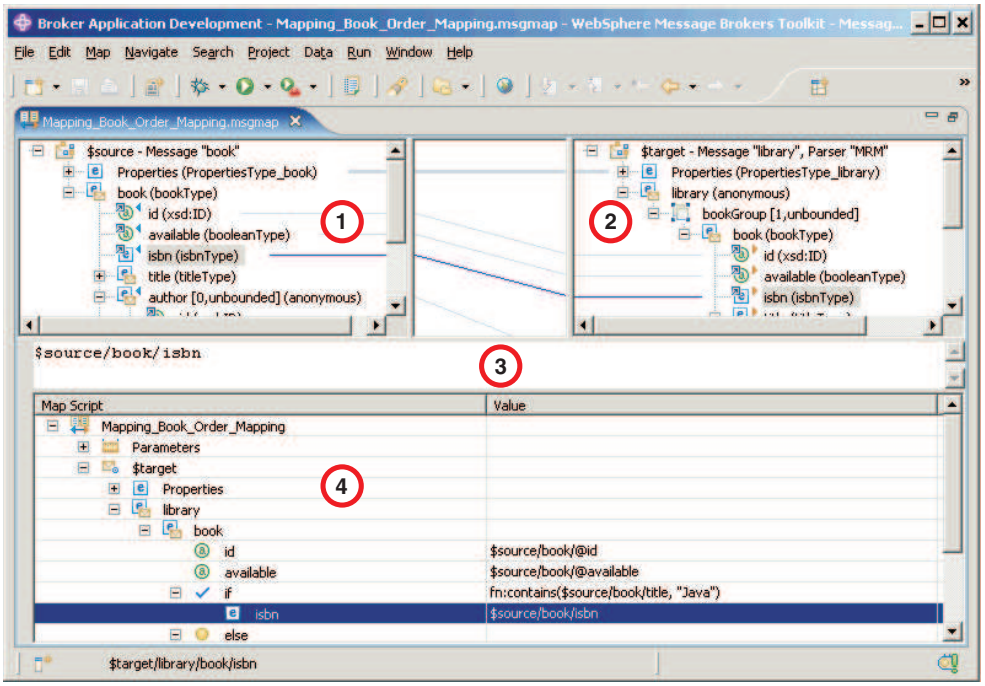
"Message Mapping editor Spreadsheet pane" on page 4991
Details of the actions available to you in the Spreadsheet pane of the Message Mapping Editor.

"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.

Message Mapping editor Source pane:

Details of the elements present in the Source pane of the Message Mapping Editor.

The following example shows the "Message Mapping editor" on page 4981. The pane that is labelled as 1 in the example is the Source pane:



The following list describes the elements that are present in the Source pane:

- A source message is identified by $source.
- A source database is identified by $db:select.
- A database stored procedure is identified by $db:proc.
- A database user-defined function is identified by $db:func.
- A mapped entry is indicated by a blue triangle alongside the element. In this example, Customer_ID and Order_Date are mapped.
- Square brackets contain minimum and maximum occurrences of an element.
- An optional field is indicated by [0,1]. In this example, First_Class is optional.
- A repeating field is indicated by [minoccurs, maxoccurs].
- A choice field is indicated by a choice line; under the choice line are the possible choices. In this example, First_Class, Second_Class, and Airmail are choices of Delivery_Method.
- The type of each element is indicated in round brackets after the element name.
- If the message schema uses namespaces, the namespace prefix is shown before the element name, separated by a colon.

Use the Source pane to invoke a number of actions, a list of which is displayed when you right-click within the Source pane. The following table describes the available actions.

Action	Description	Related tasks
Undo	Undo previous action	
Redo	Redo previous action	
Revert	Discard	
Open Declaration (message)	Display the element definition from the message set. For this action to be available, select any source message element except LocalEnvironment or Headers.	
Open Declaration (database)	Display the database, schema, or table definition from the database. For this action to be available, select any source database object.	
Show Derived Types	Hide or display derived types for an element in the source or target pane. For this action to be available, select a target element displayed as a specialization folder in the source pane.	
Show Substituting elements	Hide or display the substituting elements of the head element in the source or target pane. For this action to be available, select a target element displayed as a substitutions folder in the source pane.	
Add Sources	Add a message definition or a database table to a source. For this action to be available, select any source object.	"Adding messages or message components to the source or target" on page 2273, "Adding a database as a source or target" on page 2274
Go To	For this action to be available, select any source object.	
Delete (message)	Remove a message and any existing maps from the source. For this action to be available, select the source message ($source).	

Action	Description	Related tasks
Delete (database)	Remove a database and any existing maps from the source. For this action to be available, select the source database root ($db:select).	
Delete (database stored procedure)	Remove a database stored procedure and any existing maps from the source. For this action to be available, select the database stored procedure root ($db:proc).	
Delete (database user-defined function)	Remove a database user-defined function and any existing maps from the source. For this action to be available, select the database user-defined function root ($db:func).	
Map from Source	Create a map between the focus source element and the focus target element. For this action to be available, select compatible source and target elements.	"Mapping a target element from source message elements" on page 2254, "Mapping from source: by selection" on page 2240
Map by Name	Create a map between the focus source element and the focus target element. For this action to be available, select compatible source and target elements.	"Mapping a target element from source message elements" on page 2254, "Mapping from source: by name" on page 2241
Accumulate	If the source and target fields contain numeric data types, this action maps all occurrences of a repeating source field to a non-repeating target, resulting in the sum of all the source elements. For this action to be available, select the source and target element.	"Configuring a repeating source and a non-repeating target" on page 2266
Create New Submap	For this action to be available, select source and target elements that are either elements of complex types or wildcard elements.	"Creating and calling submaps and subroutines" on page 2298, "Creating a new submap" on page 2299, "Creating a new submap for a wildcard source" on page 2300

Action	Description	Related tasks
Create New Database Submap	Create a submap to modify a database	"Creating a submap to modify a database" on page 2301
Call Existing Submap	Call an existing submap	"Creating and calling submaps and subroutines" on page 2298, "Calling a submap" on page 2305
Call ESQL Routine	Call an ESQL routine	"Creating and calling submaps and subroutines" on page 2298, "Calling an ESQL routine" on page 2308
Call Java Method	Call a Java Method	"Calling a Java method" on page 2310
Add or Remove Headers and Folders	Include message headers and folders for source messages in a message map	"Mapping headers and folders" on page 2271
Toggle Add/Remove Stored Procedure Return Value	Specify if a database stored procedure sets a return value. DB2 on z/OS and Oracle stored procedures do not set a return value.	"Mapping a target element from database stored procedures" on page 2290
Add or Remove Result Set Columns	Specify the Result Set Columns for a database stored procedure	"Mapping a target element from database stored procedures" on page 2290
Save	Save the .msgmap file	

Related concepts:

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

Related tasks:

"Using message mappings" on page 2228
Message mappings define the blueprint for creating a message.

"Configuring a repeating source and a non-repeating target" on page 2266
To map a repeating source element to a non-repeating target element, drag elements between the Message Mapping editor Source and Target panes.

"Mapping a target element from source message elements" on page 2254

"Adding messages or message components to the source or target" on page 2273

"Adding a database as a source or target" on page 2274
Add a database as a source, and database tables as targets, to message maps that support database mappings.

Related reference:

"Message Mapping editor Target pane" on page 4987
Details of the elements present in the Target pane of the Message Mapping Editor.

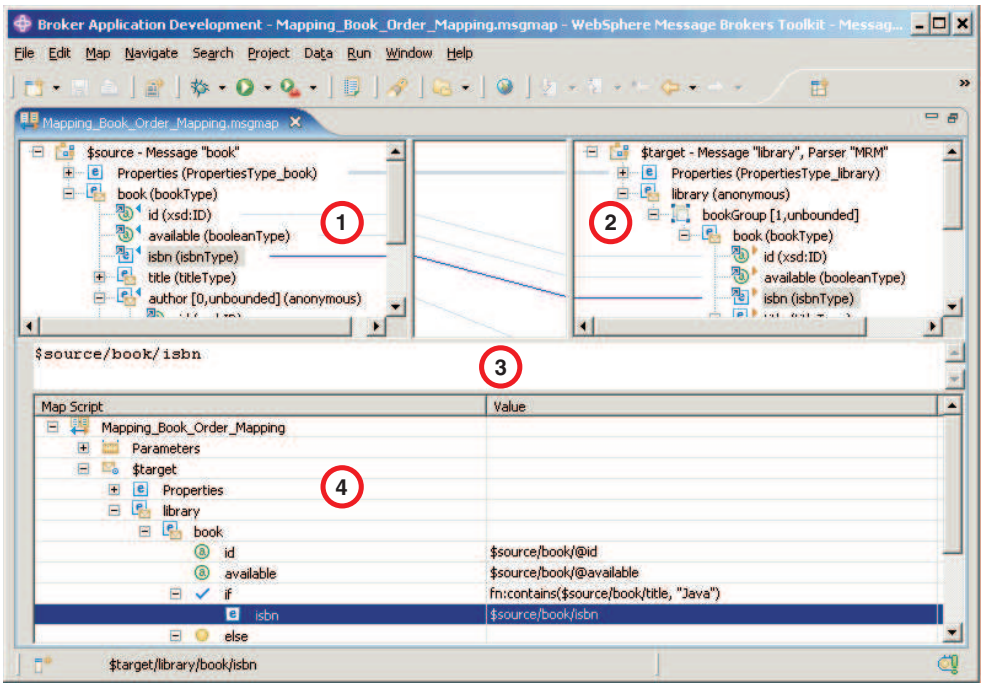
"Message Mapping editor Edit pane" on page 4990
Details of how you use the Edit pane of the Message Mapping Editor.

"Message Mapping editor Spreadsheet pane" on page 4991
Details of the actions available to you in the Spreadsheet pane of the Message Mapping Editor.

Message Mapping editor Target pane:

Details of the elements present in the Target pane of the Message Mapping Editor.

The following example shows the "Message Mapping editor" on page 4981. The pane that is labelled as 2 in the example is the Target pane:



The following list describes the elements that are present in the Target pane:

- A target message is identified by $target.
- A mapped entry is indicated by a yellow triangle alongside the element. In this example, Customer_ID, Order_Number, and Order_Date are mapped.
- Square brackets contain minimum and maximum occurrences of an element.
- An optional field is indicated by [0,1]. In this example, First_Class is optional.
- A repeating field is indicated by [minoccurs, maxoccurs].
- A choice field is indicated by a choice line; under the choice line are the possible choices. In this example, First_Class, Second_Class, and Airmail are choices of Delivery_Method.
- The type of each element is indicated in round brackets after the element name.
- If the message schema uses namespaces, the namespace prefix is shown before the element name, separated by a colon.

Use the Target pane to invoke a number of actions, a list of which is displayed when you right-click within the Target pane. The following table describes the available actions.

Action	Description	Related tasks
Undo	Undo previous action	
Redo	Redo previous action	
Revert	Discard	

Action	Description	Related tasks
Open Declaration (message)	Display the element definition from the message set. For this action to be available, select any target message element except LocalEnvironment or Headers.	
Open Declaration (database)	Display the database, schema, or table definition from the database. For this action to be available, select any target database object.	
Show Derived Types	Hide or display derived types for an element in the source or target pane. For this action to be available, select a target element displayed as a specialization folder in the target pane.	
Show Substituting elements	Hide or display the substituting elements of the head element in the source or target pane. For this action to be available, select a target element displayed as a substitutions folder in the target pane.	
Add Sources and Targets	Add a message definition or a database table to a source. For this action to be available, select any target object.	"Adding messages or message components to the source or target" on page 2273, "Adding a database as a source or target" on page 2274
Go To	For this action to be available, select any target object.	
Delete (message)	Remove a message and any existing maps from the source. For this action to be available, select the target message ($target).	

Action	Description	Related tasks
Map from Source	Create a map between the focus source element and the focus target element. For this action to be available, select compatible source and target elements.	"Mapping a target element from source message elements" on page 2254, "Mapping from source: by selection" on page 2240
Map by Name	Create a map between the focus source element and the focus target element. For this action to be available, select compatible source and target elements.	"Mapping a target element from source message elements" on page 2254, "Mapping from source: by name" on page 2241
Enter Expression	For this action to be available, select any target object except $target	"Setting the value of a target element to a constant" on page 2257, "Setting the value of a target element using an expression or function" on page 2261
Accumulate	If the source and target fields contain numeric data types, this action maps all occurrences of a repeating source field to a non-repeating target, resulting in the sum of all the source elements. For this action to be available, select the source and target element.	"Configuring a repeating source and a non-repeating target" on page 2266
Create New Submap	For this action to be available, select source and target elements that are either elements of complex types or wildcard elements.	"Creating and calling submaps and subroutines" on page 2298, "Creating a new submap" on page 2299, "Creating a new submap for a wildcard source" on page 2300
Call Existing Submap	Call an existing submap	"Creating and calling submaps and subroutines" on page 2298, "Calling a submap" on page 2305
Call ESQL Routine	Call an existing ESQL routine	"Creating and calling submaps and subroutines" on page 2298, "Calling an ESQL routine" on page 2308
Save	Save the .msgmap file	

Related concepts:

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

Related tasks:

"Using message mappings" on page 2228
Message mappings define the blueprint for creating a message.

"Adding messages or message components to the source or target" on page 2273

"Adding a database as a source or target" on page 2274
Add a database as a source, and database tables as targets, to message maps that support database mappings.

"Mapping a target element from source message elements" on page 2254

"Setting the value of a target element to a constant" on page 2257
Use the Message Mapping editor to set the value of a target element to a constant.

"Setting the value of a target element using an expression or function" on page 2261

"Configuring a repeating source and a non-repeating target" on page 2266
To map a repeating source element to a non-repeating target element, drag elements between the Message Mapping editor Source and Target panes.

Related reference:

"Message Mapping editor Source pane" on page 4983
Details of the elements present in the Source pane of the Message Mapping Editor.

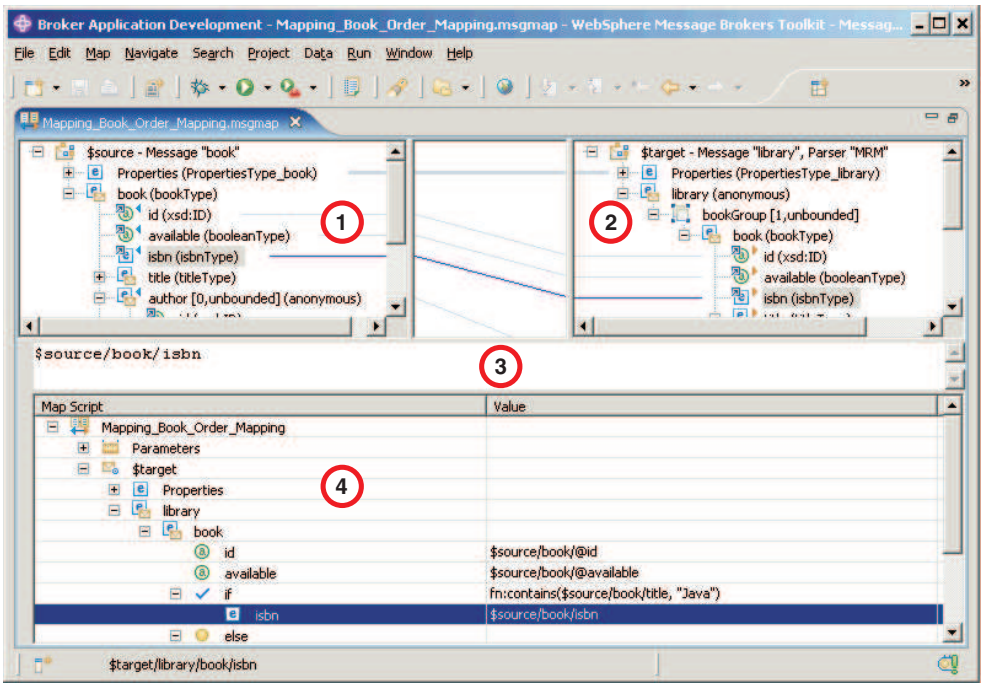
"Message Mapping editor Edit pane"
Details of how you use the Edit pane of the Message Mapping Editor.

"Message Mapping editor Spreadsheet pane" on page 4991
Details of the actions available to you in the Spreadsheet pane of the Message Mapping Editor.

Message Mapping editor Edit pane:

Details of how you use the Edit pane of the Message Mapping Editor.

The following example shows the "Message Mapping editor" on page 4981. The pane that is labelled as 3 in the example is the Edit pane:



When you have selected a source or target element, use the Edit pane to enter an expression. Right-click inside the Edit pane to invoke a list of available actions, most of which are standard Windows functions, such as cut, copy, and paste. Click **Edit** > **Content Assist** (or press Ctrl+Space) to access ESQL Content Assist, which provides a drop-down list of functions that are available in a Mapping node.

To display the definition associated with a selected element or database object, right-click in the Edit pane, and click **Open Declaration**. The appropriate editor opens to display the definition associated with the element or database definition.

Related concepts:

"Mapping node syntax" on page 4995

"Mapping node functions" on page 4997
You can configure your message mappings to use a variety of predefined and user-defined functions.

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

Related tasks:

"Using message mappings" on page 2228
Message mappings define the blueprint for creating a message.

Related reference:

"Mapping node" on page 4571
Use the Mapping node to construct one or more new messages and populate them with various types of information.

"Message Mapping editor Source pane" on page 4983
Details of the elements present in the Source pane of the Message Mapping Editor.

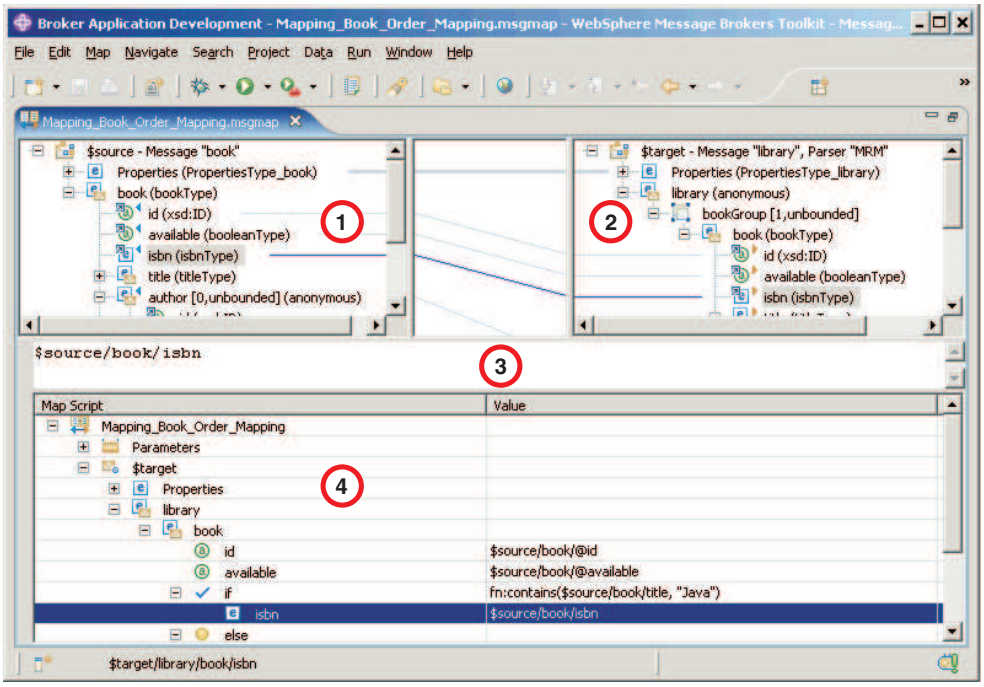
"Message Mapping editor Target pane" on page 4987
Details of the elements present in the Target pane of the Message Mapping Editor.

"Message Mapping editor Spreadsheet pane"
Details of the actions available to you in the Spreadsheet pane of the Message Mapping Editor.

Message Mapping editor Spreadsheet pane:

Details of the actions available to you in the Spreadsheet pane of the Message Mapping Editor.

The following example shows the "Message Mapping editor" on page 4981. The pane that is labelled as 4 in the example is the Spreadsheet pane:



Use the Spreadsheet pane to invoke a number of actions, a list of which is displayed when you right-click within the Spreadsheet pane. The following table describes the available actions.

Action	Description	Related tasks
Undo	Undo previous action	
Redo	Redo previous action	
Revert	Discard	
Open Declaration (message)	Display the element definition from the message set. For this action to be available, select any message element except LocalEnvironment or Headers.	
Open Declaration (database)	Display the database, schema, or table definition from the database. For this action to be available, select any database object.	
Add Sources and Targets	Add a message definition to a target.	"Adding messages or message components to the source or target" on page 2273, "Adding a database as a source or target" on page 2274
Copy	Copy the selected item to the clipboard.	

Action	Description	Related tasks
Paste	Paste the item from the clipboard.	
Delete	Remove a row from the Spreadsheet.	
For	Define a repeating condition.	“Configuring a repeating source and a non-repeating target” on page 2266, “Configuring a repeating source and a repeating target” on page 2269
If	Define what must evaluate to 'true' to process subsequent mappings.	“Configuring a repeating source and a non-repeating target” on page 2266, “Configuring conditional mappings” on page 2265
ElseIf	Define what must evaluate to 'true' to process subsequent mappings if previous If or Elseif does not evaluate to 'true'..	“Configuring a repeating source and a non-repeating target” on page 2266, “Configuring conditional mappings” on page 2265
Else	Placeholder to process subsequent mappings if previous If or ElseIf does not evaluate to 'true'.	“Configuring conditional mappings” on page 2265
Select Data Source	Define a database to be used in the mapping.	
Insert Children	Expand a structure so that each of its children has a row in the spreadsheet.	
Insert Sibling After	Create a number of new rows in the spreadsheet to set the values of specific instances of a repeating field. Can also be used to insert any non-repeating element, attribute or database column if valid at the selected location.	“Configuring a non-repeating source and a repeating target” on page 2268
Insert Sibling Before	Create a number of new rows in the spreadsheet to set the values of specific instances of a repeating field. Can also be used to insert any non-repeating element, attribute or database column if valid at the selected location.	“Configuring a non-repeating source and a repeating target” on page 2268
Replace	Substitute an element, attribute or database column in the spreadsheet with a similar item, retaining the mapping expression and any child mapping statements.	
Save	Save the .msgmap file.	

Related concepts:

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

Related tasks:

"Using message mappings" on page 2228
Message mappings define the blueprint for creating a message.

"Adding messages or message components to the source or target" on page 2273

"Adding a database as a source or target" on page 2274
Add a database as a source, and database tables as targets, to message maps that support database mappings.

"Configuring a repeating source and a non-repeating target" on page 2266
To map a repeating source element to a non-repeating target element, drag elements between the Message Mapping editor Source and Target panes.

"Configuring a repeating source and a repeating target" on page 2269

"Configuring conditional mappings" on page 2265
How to set the value of a target element conditionally in a Mapping node.

"Configuring a non-repeating source and a repeating target" on page 2268
To map a non-repeating source element to a repeating target element, drag elements between the Message Mapping editor Source and Target panes.

Related reference:

"Message Mapping editor Source pane" on page 4983
Details of the elements present in the Source pane of the Message Mapping Editor.

"Message Mapping editor Target pane" on page 4987
Details of the elements present in the Target pane of the Message Mapping Editor.

"Message Mapping editor Edit pane" on page 4990
Details of how you use the Edit pane of the Message Mapping Editor.

Mapping node:

The Mapping node has one or more mappings that are stored in message map files (with a `.msgmap` file extension). These files are configured using the Message Mapping editor.

A Mapping node must contain the following inputs and outputs:

- Zero or one source (input) messages
- Zero or more source (input) databases
- One or more target (output) messages

You must define, in message definition files in a message set, the source and target messages that are to be mapped. You can specify the parser of the source message at run time (for example, in an MQRFH2 header), but the target message is built using the runtime parser that is specified by the *Message Domain* property of the message set.

If a message mapping is between elements of different types, you might need to include casts in your mapping definitions, depending on which runtime parser is specified by the *Message Domain* property of your message set.

The Mapping node uses a language to manipulate messages that are based on XPath.

To develop message mappings for a Mapping node, use the Message Mapping editor, which provides separate panes for working with sources, targets and expressions.

Related concepts:

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

"Mapping node syntax"

"Mapping node functions" on page 4997
You can configure your message mappings to use a variety of predefined and user-defined functions.

"Mapping node casts" on page 5009
Source and target elements can be of different types in a Mapping node.

Related tasks:

"Using message mappings" on page 2228
Message mappings define the blueprint for creating a message.

Related reference:

"Mapping node" on page 4571
Use the Mapping node to construct one or more new messages and populate them with various types of information.

Mapping node syntax: In a Mapping node, the source message, if present, is identified in the "Message Mapping editor" on page 4981 by $source.

The message tree is represented in XPath format. For example, if you have an element called Body within a source message called Envelope, this is represented in the Mapping node as:

```
$source/soap11:Envelope/soap11:Body
```

where *soap11* is a namespace prefix.

The first target message is identified by $target; additional target messages are identified by $target_1, $target_2, and so on.

The first source database is identified by $db:select; additional source databases are identified by $db:select_1, $db:select_2, and so on.

The first source database stored procedure is identified by $db:proc; additional source stored procedures are identified by $db:proc_1, $db:proc_2, and so on.

The first source database user-defined function is identified by $db:func; additional source user-defined functions are identified by $db:func_1, $db:func_2, and so on.

The database element is represented in the following format:

```
$db:select.DB.SCH.TAB.COL1
```

where:

DB is the database name

SCH is the database schema name

TAB is the table name

COL1 is the column name

You can also use the Mapping node to:

- make comparisons

- perform arithmetic
- create complex conditions

The comparison operators are:

= equals

!= not equals

> greater than

>= greater than or equals

< less than

<= less than or equals

The arithmetic operators are:

+ plus

- minus

* multiply

div divide

Conditional operators ‘or' and ‘and' are supported (these are case-sensitive).

The following objects can be mapped:

- LocalEnvironment
  - Destination
  - WrittenDestination
  - File
  - SOAP
  - TCPIP
  - ServiceRegistry
  - Adapter
  - Wildcard
  - Variables
- Message headers (optional)
  - MQ Headers
  - HTTP Headers
  - JMSTransport
  - email Headers
- Message elements
- Database columns

Database objects with names that do not conform to the XML NCName format

Some database objects have names that do not conform to the XML NCName format (for example, the names contains characters like '#', or '$'). To reference such database objects use the `msgmap:db-path` function.

For more information, see “Predefined mapping functions” on page 5007

Related concepts:

“Mapping node functions” on page 4997
You can configure your message mappings to use a variety of predefined and user-defined functions.

"Mapping node casts" on page 5009
Source and target elements can be of different types in a Mapping node.

"Message Mapping editor" on page 4981
You use the Message Mapping editor to create and edit message mappings.

"Mapping node" on page 4994
The Mapping node has one or more mappings that are stored in message map files (with a `.msgmap` file extension). These files are configured using the Message Mapping editor.

"Local environment tree structure" on page 1056
The local environment tree is a part of the logical message tree in which you can store information while the message flow processes the message.

Related tasks:

"Using message mappings" on page 2228
Message mappings define the blueprint for creating a message.

"Configuring the LocalEnvironment" on page 2271

Related reference:

"Mapping node" on page 4571
Use the Mapping node to construct one or more new messages and populate them with various types of information.

Mapping node functions:

You can configure your message mappings to use a variety of predefined and user-defined functions.

The following predefined functions are available to use in your message maps:

- ESQL - prefixed esql:
- XPath - prefixed fn:
- Mapping - prefixed msgmap:
- Schema casts - prefixed xs:

Not all ESQL functions can be used in a Mapping node. For information about which functions are supported, and for a description of how to achieve equivalent processing for ESQL functions that are not supported, see the ESQL topics. For information about the predefined ESQL functions, see "ESQL mapping functions" on page 4998.

The fn:true() function (which always returns true) and the fn:false() function (which always returns false) are examples of XPath functions. You can get more information about the other XPath functions and XPath syntax from the online W3C XML Path Language document. For information about the predefined XPath functions, see "Predefined XPath mapping functions" on page 5001.

For information about the predefined mapping functions, see "Predefined mapping functions" on page 5007. See "Mapping node casts" on page 5009 for a list of the schema casts.

The Mapping node can also:

- Set the value of a target to a WebSphere MQ constant. The expression to set the value looks similar to a function with $mq: used as the prefix.
- Call a Java method directly. The expression to set the value looks similar to a function with java: used as a prefix.

Related concepts:

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

"Mapping node" on page 4994
The Mapping node has one or more mappings that are stored in message map files (with a `.msgmap` file extension). These files are configured using the Message Mapping editor.

"Mapping node syntax" on page 4995

"Mapping node casts" on page 5009
Source and target elements can be of different types in a Mapping node.

Related tasks:

"Using message mappings" on page 2228
Message mappings define the blueprint for creating a message.

"Calling a Java method" on page 2310
To call an existing Java method from a mapping node, select the method from the Call Existing Java Method wizard, or enter an XPath expression in the Edit pane.

Related reference:

"Mapping node" on page 4571
Use the Mapping node to construct one or more new messages and populate them with various types of information.

"ESQL mapping functions"
Some predefined ESQL functions are available for use with message maps.

"Predefined XPath mapping functions" on page 5001
Some XPath functions are available for use with message maps.

"Predefined mapping functions" on page 5007
Some predefined mapping functions are provided for use with message maps.

ESQL mapping functions:

Some predefined ESQL functions are available for use with message maps.

This table details the predefined ESQL mapping functions that are available to use with message maps:

Name	ESQL equivalent	Notes
Numeric functions: abs absval acos asin atan atan2 bitand bitnot bitor bitxor ceil ceiling cos cosh cot degrees exp floor in log log10 mod power radians rand sign sin sinh sqrt tan tanh truncate	ESQL function of the name same name such as ABS and ABSVAL.	The same parameters apply as for ESQL.
String functions: left length lower lcase ltrim replace replicate right rtrim space translate upper ucase	ESQL function of the same name such as LEFT and LENGTH.	The same parameters apply as for ESQL.
Field functions: bitstream fieldname fieldnamespace fieldtype fieldvalue lastmove samefield	ESQL function of the same name such as BITSTREAM and FIELDNAME.	The same parameters apply as for ESQL.

Name	ESQL equivalent	Notes
asbitstream	These signatures are supported: ASBITSTREAM(*FieldRef*) ASBITSTREAM(*FieldRef*, *typeExp*, *setExp*, *formatExp*) ASBITSTREAM(*FieldRef*, *typeExp*, *setExp*, *formatExp*, *encodingExp*, *ccsidExp*) ASBITSTREAM(*FieldRef*, *typeExp*, *setExp*, *formatExp*, *encodingExp*, *ccsidExp*, *options*)	*FieldRef* is a source field reference such as $source/po:PurchaseOrder *typeExp* is a string literal of the name of the message body, such as purchaseOrder, optionally qualified with a namespace URI, such as {http://www.ibm.com}:purchaseOrder *setExp* is a string literal of the name of the message set, such as PurchaseOrder *formatExp* is a string literal of the wire format of the message, such as XML1 *encodingExp* and *ccsidExp* evaluate to integers with values corresponding to ESQL ENCODING and CCSID constants. *options* is an ESQL constant or bit-or of ESQL constant that evaluates to an integer.
cardinality	CARDINALITY	The same parameters apply as for ESQL.
coalesce	COALESCE	The same parameters apply as for ESQL.
current-date	CURRENT_DATE	No parameters apply.
current-gmtdate	CURRENT_GMTDATE	No parameters apply.
current-gmttime	CURRENT_GMTTIME	No parameters apply.
current-gmttimestamp	CURRENT_ GMTTIMESTAMP	No parameters apply.
current-time	CURRENT_TIME	No parameters apply.
current-timestamp	CURRENT_TIMESTAMP	No parameters apply.
date	DATE	
for	FOR (expression)	Optional parameters are not supported.
gmttime	GMTTIME	
gmttimestamp	GMTTIMESTAMP	

interval-year	INTERVAL YEAR	The same parameters apply as for ESQL. Some examples:
		
		`esql:interval-minute('90')`
		`esql:interval-year-to-month('1-06')`

interval-year-to-month	INTERVAL YEAR TO MONTH	
interval-month	INTERVAL MONTH	
interval-day	INTERVAL DAY	
interval-day-to-hour	INTERVAL DAY TO HOUR	
interval-day-to-minute	INTERVAL DAY TO MINUTE	
interval-day-to-second	INTERVAL DAY TO SECOND	
interval-hour	INTERVAL HOUR	
interval-hour-to-minute	INTERVAL HOUR TO MINUTE	
interval-hour-to-second	INTERVAL HOUR TO SECOND	
interval-minute	INTERVAL MINUTE	
interval-minute-to-second	INTERVAL MINUTE TO SECOND	
interval-second	INTERVAL SECOND	

is-null	Operand IS NULL	Some examples:
		
		`esql:is-null($source/po:purchaseOrder/po:comment)`
		`esql:is-null ($db:select.ACME.PARTS.INVENTORY.LAST_TRANSACTION)`

like	source LIKE pattern	For example: `esql:like ($source/po:purchaseOrder/shipTo/first_name,'Fred')`
	source LIKE pattern ESCAPE EscapeChar	For example: `esql:like ($source/po:purchaseOrder/shipTo /zip,'L6F$_1C7','$')`
local-timezone	LOCAL_TIMEZONE	

Name	ESQL equivalent	Notes
nullif	NULLIF	The same parameters apply as for ESQL.
overlay	OVERLAY Str1 PLACING Str2 FROM Start	For example: `esql:overlay ($source/po:purchaseOrder/shipTo/city,'abc',2)`
	OVERLAY Str1 PLACING Str2 FROM Start For Length	For example: `esql:overlay ($source/po:purchaseOrder/shipTo/city,'abcde',2,3)`
position	POSITION searchExp IN SourceExp	For example: `esql:position ('aet',$source/po:purchaseOrder/shipTo/first_name)`
	POSITION searchExp IN SourceExp FROM FromExp	For example: `esql:position ('do',$source/po:purchaseOrder/shipTo/last_name,1)`
	POSITION searchExp IN SourceExp FROM FromExp REPEAT RepeatExp	For example: `esql:position ('a',$source/po:purchaseOrder/billTo /first_name,1,2)`
round	ROUND	Optional parameters are not supported.
sqlcode	SQLCODE	No parameters apply.
sqlerrortext	SQLERRORTEXT	
sqlnativeerror	SQLNATIVEERROR	
sqlstate	SQLSTATE	
time	TIME	
timestamp	TIMESTAMP	The same parameters apply as for ESQL. For example: `esql:gmttimestamp ('1999-12-31 23:59:59.999999')`
trim-leading	TRIM LEADING FROM Source	For example: `esql:trim-leading ($source/po:purchaseOrder/shipTo/state)`
	TRIM LEADING Singleton FROM Source	For example: `esql:trim-leading ('G',$source/po:purchaseOrder/shipTo/zip)`
trim-trailing	TRIM TRAILING FROM Source	For example: `esql:trim-trailing ($source/po:purchaseOrder/billTo/last_name)`
	TRIM TRAILING Singleton FROM Source	For example: `esql:trim-trailing ('e',$source/po:purchaseOrder/billTo/street)`
trim-both	TRIM BOTH FROM Source	For example: `esql:trim-both ($source/po:purchaseOrder/shipTo/city)`
	TRIM BOTH Singleton FROM Source	For example: `esql:trim-both (",$source/po:purchaseOrder/shipTo/city)`

Name	ESQL equivalent	Notes
trim	TRIM Source	For example: `esql:trim ($source/po:purchaseOrder/shipTo/city)`
	TRIM Singleton FROM Source	For example: `esql:trim (",$source/po:purchaseOrder/shipTo/city)`
uuidasblob	UUIDASBLOB	Takes zero or more parameters as in ESQL.
uuidaschar	UUIDASCHAR	

Related concepts:

"Mapping node" on page 4994
The Mapping node has one or more mappings that are stored in message map files (with a `.msgmap` file extension). These files are configured using the Message Mapping editor.

Related tasks:

"Using message mappings" on page 2228
Message mappings define the blueprint for creating a message.

Related reference:

"ESQL constants" on page 5302
Use these constants to make or parse a bit stream.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"Predefined XPath mapping functions"
Some XPath functions are available for use with message maps.

"Predefined mapping functions" on page 5007
Some predefined mapping functions are provided for use with message maps.

"Mapping node" on page 4571
Use the Mapping node to construct one or more new messages and populate them with various types of information.

Predefined XPath mapping functions:

Some XPath functions are available for use with message maps.

This table details the predefined XPath functions that are available for use with message maps. You can get more information about XPath functions and XPath syntax from the online W3C XML Path Language document.

Name	Parameters	Notes
true		
false		

Name	Parameters	Notes
sum	Source field from the message or database or “XPath "for" expression” on page 5005.	Source supports an XPath predicate. An XPath predicate is an expression enclosed in square brackets, serving to filter a sequence, retaining some items and discarding others. Predicates are supported on the XPath aggregate functions of `avg`, `count`, `max`, `min`, and `sum`. Predicates must be in one of the two forms: • An integer literal – aggregation is done for elements whose context position equals the integer. • A Boolean expression – aggregation is done for elements that make the Boolean expression evaluate to true. See “Aggregating XPath expressions conditionally” on page 5003 for more information.
avg		
max		
min		
count		
concat	Two or more strings.	You cannot use `fn:concat($source/myElem)` to concatenate instances of 'myElem'.
not	1- Expression resolved to a Boolean value.	
exists	Source field from the message or database.	
empty		
substring	1- String 2- One-based starting index 3- Length	For example: `fn:substring ($source/po:purchaseOrder/billTo/street, 3, 5)`
substring-before	Two strings	Returns the substring of the first string that precedes the first occurrence of the second string.
substring-after	Two strings	Returns the substring of the first string that follows the first occurrence of the second string.
starts-with	Two strings	Returns a Boolean value indicating whether the first string starts with the second string.
ends-with	Two strings	Returns a Boolean value indicating whether the first string ends with the second string.
contains	Two strings	Returns a Boolean value indicating whether the first string contains the second string.
year-from-dateTime	1- xs:dateTime	For example: `fn:month-from-dateTime (xs:dateTime($source/po:purchaseOrder /shipTo/datetime))` where `$source/po:purchaseOrder/shipTo/datetime` is xs:string.
month-from-dateTime		
day-from-dateTime		
hours-from-dateTime		
minutes-from-dateTime		
seconds-from-dateTime		
year-from-date	1-xs:date	For example: `fn:year-from-date(xs:date ($source/po:purchaseOrder/billTo/date))` where `$source/po:purchaseOrder/billTo/date` is xs:string.
month-from-date		
day-from-date		

Name	Parameters	Notes

hours-from-time	1- xs:time	Some examples:
		
		`fn:hours-from-time(xs:time("13:20:10:5"))`
		`fn:hours-from-time(xs:time ($source/po:purchaseOrder/shipTo/time))`

minutes-from-time		
seconds-from-time		
years-from-duration	1- xdt:dayTimeDuration	For example: `fn:minutes-from-duration (xdt:dayTimeDuration(PT47H30M))`
months-from-duration		
days-from-duration		
hours-from-duration		
minutes-from-duration		
seconds-from-duration		

Related concepts:

"Mapping node" on page 4994
The Mapping node has one or more mappings that are stored in message map files (with a `.msgmap` file extension). These files are configured using the Message Mapping editor.

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

Related tasks:

"Aggregating XPath expressions conditionally"
You can perform calculations using conditions, called predicates in XPath, on the aggregate functions in the Mapping node. The aggregate functions are `avg`, `count`, `max`, `min`, and `sum`.

"XPath "for" expression" on page 5005
You can use the `for` expression to perform specific calculations as the argument of an aggregation function in the Mapping node.

"Using message mappings" on page 2228
Message mappings define the blueprint for creating a message.

Related reference:

"ESQL mapping functions" on page 4998
Some predefined ESQL functions are available for use with message maps.

"Predefined mapping functions" on page 5007
Some predefined mapping functions are provided for use with message maps.

"Mapping node" on page 4571
Use the Mapping node to construct one or more new messages and populate them with various types of information.

Aggregating XPath expressions conditionally:

You can perform calculations using conditions, called predicates in XPath, on the aggregate functions in the Mapping node. The aggregate functions are `avg`, `count`, `max`, `min`, and `sum`.

About this task

XPath aggregate functions

XPath aggregate functions take a sequence as their argument:

```
fn:count($arg as item()*) as xs:integer
fn:avg($arg as xs:anyAtomicType*) as xs:anyAtomicType?
fn:max($arg as xs:anyAtomicType*) as xs:anyAtomicType?
fn:min($arg as xs:anyAtomicType*) as xs:anyAtomicType?
fn:sum($arg as xs:anyAtomicType*) as xs:anyAtomicType?
```

where the atomic value can be, for example, an integer, a string, or a Boolean value.

In an aggregation

```
fn:sum($source/inventory/category/product/price)
```

the node

```
$source/inventory/category/product/price
```

is put into its typed value, which is a sequence of `Atomictype` `price` values.

In the following aggregation, the argument is a node `tns:price`, which is contained in `tns:product`, which is in turn contained in `tns:category`, and so on.

```
fn:sum($source/tns:Inventory/tns:category/tns:product/tns:price)
```

How aggregation works in this sequence depends upon the scope of the iteration (or `for`) loop for `product` and `category`. For example, if there is no iteration loop, summation is done for all instances of all `prices` in all `products` in all `categories`.

However, if there is an iteration loop on `category`, one summation is calculated for each `category`. The summation is obtained from all `prices` of all `products` of a particular `category` being iterated upon.

You can add a predicate (see predicates for further information) to make the result more specific. For example:

```
fn:sum($source/tns:Inventory/tns:category[$source/tns:Inventory/tns:category/
  tns:c_id='abc' ]/tns:product[3]/tns:price)
```

has a predicate of `tns:c_id='abc'` on `category`, and a predicate of `3` on `product`. The result that you obtain is the third `product` in the `category` that has `tns:c_id='abc'`.

In both of the preceding examples, whether the expression contains a condition or not, you reference one source item only; it is `price` that is aggregated.

Predicates are supported only when they are used in message sources. For example, the `[ boolean expression ]` must be used within a segment of a path that represents a message source. Predicates are not supported in `$db:select` or `$db:proc`.

Consider a more complicated scenario, where product prices are kept in a database table, and the input message contains the quantity of the product being ordered. The objective is to calculate a total price by adding up all prices multiplied by quantity.

It might seem to be straightforward to write the following aggregation function:

```
fn:sum($source/inventory/product/quantity × $db:select/PRICE_TB/PRICE)
```

However, the first step of evaluating an XPath arithmetic expression is to evaluate its operands. If any operand is evaluated into a sequence of more than one item,

either only the first item is used in the arithmetic expression, or an error is raised; for further information, see Arithmetic expressions.

You, therefore, cannot add up the product of `quantity` and `PRICE` for each `product`. If you want to aggregate the result of an arithmetic expression, see "XPath "for" expression."

Related concepts:

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

Related tasks:

"XPath "for" expression"
You can use the `for` expression to perform specific calculations as the argument of an aggregation function in the Mapping node.

"Using message mappings" on page 2228
Message mappings define the blueprint for creating a message.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"Predefined XPath mapping functions" on page 5001
Some XPath functions are available for use with message maps.

"Mapping node" on page 4571
Use the Mapping node to construct one or more new messages and populate them with various types of information.

XPath "for" expression:

You can use the `for` expression to perform specific calculations as the argument of an aggregation function in the Mapping node.

About this task

For expressions

To perform a specific operation you can use the XPath `for` expression iteration facility. For more information, see the for expression online document.

You can express a sequence of price multiplied by quantity in many ways using the `for` expression. For example:

- `for $i in $source/inventory/category/product return $i/price * $i/quantity`

  You can use a similar expression when all operands of the return expression (`price` and `quantity`) are defined in the same repeatable container (`product`). Only one variable is needed in the `for` expression.
- ```
  for $i in $db:select, $j in $source/inventory/category/product
  [$i.db1.sch2.PRICE_TB.PROD_ID=$j/product_id]
  return $i.db1.sch2.PRICE_TB.PRICE * $j/quantity
  ```

  You can use a similar expression when some operands of the return expression (`price`) are kept in a database table, and other operands (`quantity`) are kept in a message. At least one variable is needed for the path to a database result set, and another variable is needed for the path to a repeatable XML element.
  ```

- `for $i in $source/order_info/product1, $j in $source/price_record/product2 [$i/product_id=$j/product_id] return $i/quantity * $j/price`

  You can use a similar expression when the operands (`price` and `quantity`) are defined in different repeatable elements (`product1` and `product2`) in the source message.

When you have an expression that represents a sequence of items, aggregation can be done by using the `for` expression as the argument of the aggregation function.

Example

Examples

Obtain the average cost, which is price multiplied by quantity, for all products:

```
fn:avg(for $i in $source/inventory/category/product
   return $i/price * $i/quantity)
```

Obtain the total cost of all products where prices are from a database, and quantities are from a message and paired up based on a product identifier:

```
fn:sum(for $i in $db:select, $j in $source/inventory/category/product
   [$i.db1.sch2.PRICE_TB.PROD_ID=$j/product_id]
   return $i.db1.sch2.PRICE_TB.PRICE * $j/quantity)
```

Obtain the length of the longest product identifier text:

```
fn:max(for $i in $source/inventory/category/product
   return esql:length($i/id))
```

Obtain the minimum price when the price was stored in a message as a string, and convert each source item into a numeric value before using an aggregate function:

```
fn:min(for $i in $source/inventory/category/product
   return xs:decimal($i/price))
```

The aggregation functions that are supported in WebSphere Message Broker can be expressed using a `for` expression.

For example, you might have:

```
fn:sum($source/inventory/category/product/price)
```

This expression is equivalent to:

```
fn:sum(for $i in $source/inventory/category/product
   return $i/price)
```

Related concepts:

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

Related tasks:

"Aggregating XPath expressions conditionally" on page 5003
You can perform calculations using conditions, called predicates in XPath, on the aggregate functions in the Mapping node. The aggregate functions are `avg`, `count`, `max`, `min`, and `sum`.

"Using message mappings" on page 2228
Message mappings define the blueprint for creating a message.

Related reference:

"Predefined XPath mapping functions" on page 5001
Some XPath functions are available for use with message maps.
"Mapping node" on page 4571
Use the Mapping node to construct one or more new messages and populate them with various types of information.

Predefined mapping functions:

Some predefined mapping functions are provided for use with message maps.

This table details the predefined mapping functions that are available to use with message maps.

Name	Parameters	Return	Notes
cdata-element	One string	Nothing	Create an XML element with CData content in the following target message domains: • XMLNSC • SOAP • XMLNS • XML • JMSMap • JMSStream For example: `msgmap:cdata-element('<date><month>05 </month><day>11</day><year>2008</year></date>')`

db-path	These signatures are supported:
	msgmap:db-path(*databasePath*)
	msgmap:db-path(*databasePath, delimiter*)
		Nothing	Used when the database path does not conform to the XML NCName format. For example, the database path contains characters like '#', or '$'.
			databasePath is a reference to either a `$db:select` statement, a `$db:insert` statement, a `$db:update` statement, a `$db:delete` statement, a `$db:proc` statement, or a `$db:func` statement.
			delimiter is a string literal used between the segments in the database path. *delimiter* has a default value of '.'.
			For example:
			`msgmap:db-path("$db:select.DB#1.SCH$#1. CustInfo.Name")`
			In this case, the default value, '.', is used as the delimiter.
			`msgmap:db-path("$db:select/DB#1/SCH$#1/ Cust.Info/Name", "/")`
			In this case, '/' is used as the delimiter because the database table name includes the default value.

Name	Parameters	Return	Notes

occurrence	Source field from the message or database	The index of the source field.	Often used in a condition statement when source repeats to execute specific statements for a specific occurrence. For example:
			`msgmap:occurrence ($source/po:purchaseOrder /items)=2`
			
			means the second field,
			`po:purchaseOrder`
			
			is being processed. Use code similar to the previous example if you want to specify a particular source occurrence. If you only want to assign the index value to a target you can specify the following code:
			`msgmap:occurrence ($source/po:purchaseOrder /items)`

exact-type	1- Source field from the message or database
	2- Namespace prefix
	3- Name of the type
		True if the source is of the specified type in the specified namespace.	Often used in a condition to execute specific statements for a specific source type. For example:
			`msgmap:exact-type ($source/tn1:msg2,'tn1', 'extendedMsgType')`
			
			The namespace prefix can be '*', indicating that only the name of the type is to be checked. For example:
			`msgmap:exact-type($source/tn1:msg2,'*', 'extendedMsgType')`
			
			returns true if the element
			`$source/tn1:msg2`
			
			has a type of
			`extendedMsgType`
			
			in any namespace.

empty-element()	None	Nothing	Creates an XML element with an empty tag. For example, if an element is named MyElement and the mapping expression for MyElement is set to msgmap:empty-element(), the output message will contain an element with no content: `<MyElement/>` Call this function only for an XML element.

Name	Parameters	Return	Notes
element-from-bitstream	These signatures are supported: msgmap:element-from-bitstream(*StreamRef*) msgmap:element-from-bitstream(*StreamRef*, *typeExp*, *setExp*, *formatExp*) msgmap:element-from-bitstream(*StreamRef*, *typeExp*, *setExp*, *formatExp*, *encodingExp*, *ccsidExp*) msgmap:element-from-bitstream(*StreamRef*, *typeExp*, *setExp*, *formatExp*, *encodingExp*, *ccsidExp*, *options*)	Nothing	Used to parse a bit stream. This function can be called only for a message element target. The parsed bit stream is placed in the target message tree as the target element. *StreamRef* is a reference to a BLOB of stream, such as $source/BLOB or $db:select.dsn.schema.table.column *typeExp* is a string literal of the name of the message body, such as purchaseOrder, optionally qualified with a namespace URI, such as {http://www.ibm.com}:purchaseOrder *setExp* is a string literal of the name of the message set, such as PurchaseOrder *formatExp* is a string literal of the wire format of the message, such as XML1 *encodingExp* and *ccsidExp* evaluate to integers with values corresponding to ESQL ENCODING and CCSID constants *options* is an ESQL constant or bit-or of ESQL constants that evaluate to an integer.

Related concepts:

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

Related tasks:

"Using message mappings" on page 2228
Message mappings define the blueprint for creating a message.

Related reference:

"ESQL constants" on page 5302
Use these constants to make or parse a bit stream.

"ESQL mapping functions" on page 4998
Some predefined ESQL functions are available for use with message maps.

"Predefined XPath mapping functions" on page 5001
Some XPath functions are available for use with message maps.

"Mapping node" on page 4571
Use the Mapping node to construct one or more new messages and populate them with various types of information.

Mapping node casts:

Source and target elements can be of different types in a Mapping node.

Depending on which runtime parsers are used, automatic casting cannot be done. In these cases, use one of the cast functions shown in the following table.

Function name	Signature	Parameter type
xs:anyURI	xs:anyURI(*&exp*) xs:anyURI(*&exp*, *&ccsid*) xs:anyURI(*&exp*, *&ccsid*, *&encoding*)	*$exp* is a reference to a source element of one of the following types: • xs:base64Binary • xs:boolean • xs:date • xs:dateTime • xs:dayTimeDuration • xs:decimal • xs:double • xs:duration • xs:float • xs:gDay • xs:gMonth • xs:gMonthDay • xs:gYear • xs:gYearMonth • xs:hexBinary • xs:int • xs:integer • xs:long • xs:QName • xs:string • xs:time • xs:yearMonthDuration *&ccsid* is one of the following values: • $source/Properties/CodedCharSetId • $source/MQMD/CodedCharSetId • An integer. For example, 1208 for UTF-8 *&encoding* is one of the following values: • $source/Properties/Encoding • $source/MQMD/Encoding • $mq:MQENC_WINDOWS (546) • $mq:MQENC_UNIX (273) • $mq:MQENC_390 (785) • Other $mq:MQENC_* constants and their BIT OR

Function name	Signature	Parameter type
xs:base64Binary	xs:base64Binary(*&exp*) xs:base64Binary(*&exp*, *&ccsid*) xs:base64Binary(*&exp*, *&ccsid*, *&encoding*)	*$exp* is a reference to a source element of one of the following types: • xs:hexBinary • xs:int • xs:string *&ccsid* is one of the following values: • $source/Properties/CodedCharSetId • $source/MQMD/CodedCharSetId • An integer. For example, 1208 for UTF-8 *&encoding* is one of the following values: • $source/Properties/Encoding • $source/MQMD/Encoding • $mq:MQENC_WINDOWS (546) • $mq:MQENC_UNIX (273) • $mq:MQENC_390 (785) • Other $mq:MQENC_* constants and their BIT OR
xs:boolean	xs:boolean(*&exp*)	*$exp* is a reference to a source element of type xs:string.
xs:date	xs:date(*&exp*)	*$exp* is a reference to a source element of one of the following types: • xs:dateTime • xs:gDay • xs:gMonth • xs:gMonthDay • xs:gYear • xs:gYearMonth • xs:string
xs:dateTime	xs:dateTime(*&exp*)	*$exp* is a reference to a source element of one of the following types: • xs:date • xs:string • xs:time
xs:dayTimeDuration	xs:dayTimeDuration(*&exp*)	*$exp* is a reference to a source element of type xs:string.

Function name	Signature	Parameter type
xs:decimal	xs:decimal(*&exp*)	*$exp* is a reference to a source element of one of the following types: • xs:decimal • xs:float • xs:int • xs:integer • xs:string
xs:double	xs:double(*&exp*)	*$exp* is a reference to a source element of one of the following types: • xs:duration • xs:float • xs:int • xs:string
xs:duration	xs:duration(*&exp*)	*$exp* is a reference to a source element of one of the following types: • xs:decimal • xs:float • xs:int • xs:string
xs:float	xs:float(*&exp*)	*$exp* is a reference to a source element of one of the following types: • xs:double • xs:duration • xs:int • xs:string
xs:gDay	xs:gDay(*&exp*)	*$exp* is a reference to a source element of one of the following types: • xs:date • xs:dateTime • xs:gMonth • xs:gMonthDay • xs:gYear • xs:gYearMonth • xs:string

Function name	Signature	Parameter type
xs:gMonth	xs:gMonth(*&exp*)	*$exp* is a reference to a source element of one of the following types: • xs:date • xs:dateTime • xs:gDay • xs:gMonthDay • xs:gYear • xs:gYearMonth • xs:string
xs:gMonthDay	xs:gMonthDay(*&exp*)	*$exp* is a reference to a source element of one of the following types: • xs:date • xs:dateTime • xs:gDay • xs:gMonth • xs:gYear • xs:gYearMonth • xs:string
xs:gYear	xs:gYear(*&exp*)	*$exp* is a reference to a source element of one of the following types: • xs:date • xs:dateTime • xs:gDay • xs:gMonth • xs:gMonthDay • xs:gYearMonth • xs:string
xs:gYearMonth	xs:gYearMonth(*&exp*)	*$exp* is a reference to a source element of one of the following types: • xs:date • xs:dateTime • xs:gDay • xs:gMonth • xs:gMonthDay • xs:gYear • xs:string

Function name	Signature	Parameter type
xs:hexBinary	xs:hexBinary(*&exp*) xs:hexBinary(*&exp*, *&ccsid*) xs:hexBinary(*&exp*, *&ccsid*, *&encoding*)	*$exp* is a reference to a source element of one of the following types: • xs:base64Binary • xs:int • xs:string *&ccsid* is one of the following values: • $source/Properties/CodedCharSetId • $source/MQMD/CodedCharSetId • An integer. For example, 1208 for UTF-8 *&encoding* is one of the following values: • $source/Properties/Encoding • $source/MQMD/Encoding • $mq:MQENC_WINDOWS (546) • $mq:MQENC_UNIX (273) • $mq:MQENC_390 (785) • Other $mq:MQENC_* constants and their BIT OR
xs:int	xs:int(*&exp*)	*$exp* is a reference to a source element of one of the following types: • xs:base64Binary • xs:decimal • xs:duration • xs:float • xs:hexBinary • xs:string
xs:integer	xs:integer(*&exp*)	*$exp* is a reference to a source element of one of the following types: • xs:base64Binary • xs:decimal • xs:duration • xs:int • xs:string

Function name	Signature	Parameter type
xs:long	xs:long(*&exp*)	*$exp* is a reference to a source element of one of the following types: • xs:base64Binary • xs:decimal • xs:duration • xs:float • xs:hexBinary • xs:string

Function name	Signature	Parameter type
xs:QName	xs:QName(*&exp*) xs:QName(*&exp*, *&ccsid*) xs:QName(*&exp*, *&ccsid*, *&encoding*)	*$exp* is a reference to a source element of one of the following types: • xs:anyURI • xs:base64Binary • xs:boolean • xs:date • xs:dateTime • xs:dayTimeDuration • xs:decimal • xs:double • xs:duration • xs:float • xs:gDay • xs:gMonth • xs:gMonthDay • xs:gYear • xs:gYearMonth • xs:hexBinary • xs:int • xs:integer • xs:long • xs:string • xs:time • xs:yearMonthDuration *&ccsid* is one of the following values: • $source/Properties/CodedCharSetId • $source/MQMD/CodedCharSetId • An integer. For example, 1208 for UTF-8 *&encoding* is one of the following values: • $source/Properties/Encoding • $source/MQMD/Encoding • $mq:MQENC_WINDOWS (546) • $mq:MQENC_UNIX (273) • $mq:MQENC_390 (785) • Other $mq:MQENC_* constants and their BIT OR

Function name	Signature	Parameter type
xs:string	xs:string(*&exp*) xs:string(*&exp*, *&ccsid*) xs:string(*&exp*, *&ccsid*, *&encoding*)	*$exp* is a reference to a source element of one of the following types: • xs:anyURI • xs:base64Binary • xs:boolean • xs:date • xs:dateTime • xs:dayTimeDuration • xs:decimal • xs:double • xs:duration • xs:float • xs:gDay • xs:gMonth • xs:gMonthDay • xs:gYear • xs:gYearMonth • xs:hexBinary • xs:int • xs:integer • xs:long • xs:QName • xs:time • xs:yearMonthDuration *&ccsid* is one of the following values: • $source/Properties/CodedCharSetId • $source/MQMD/CodedCharSetId • An integer. For example, 1208 for UTF-8 *&encoding* is one of the following values: • $source/Properties/Encoding • $source/MQMD/Encoding • $mq:MQENC_WINDOWS (546) • $mq:MQENC_UNIX (273) • $mq:MQENC_390 (785) • Other $mq:MQENC_* constants and their BIT OR
xs:time	xs:time(*&exp*)	*$exp* is a reference to a source element of one of the following types: • xs:dateTime • xs:string

Function name	Signature	Parameter type
xs:yearMonthDuration	xs:yearMonthDuration(*&exp*)	*$exp* is a reference to a source element of type xs:string.

Related concepts:

"Mapping node" on page 4994
The Mapping node has one or more mappings that are stored in message map files (with a `.msgmap` file extension). These files are configured using the Message Mapping editor.

"Mapping node syntax" on page 4995

"Mapping node functions" on page 4997
You can configure your message mappings to use a variety of predefined and user-defined functions.

Related tasks:

"Using message mappings" on page 2228
Message mappings define the blueprint for creating a message.

Related reference:

"Mapping node" on page 4571
Use the Mapping node to construct one or more new messages and populate them with various types of information.

Headers and Mapping node:

This topic lists the headers that can be manipulated by the Mapping node.

You can map these headers:

- MQ Headers
  - MQMD
  - MQCFH header with root element MQPCF
  - MQCIH
  - MQDLH
  - MQIIH
  - MQMDE
  - MQRFH
  - MQRFH header with MQRFH2 or MQRFH2C parser
  - MQRMH
  - MQSAPH
  - MQWIH
  - SMQ_BMH
- Email Headers
  - EmailOutputHeader
- HTTP Headers
  - HTTPInputHeader
  - HTTPReplyHeader
  - HTTPRequestHeader
  - HTTPResponseHeader
- JMSTransport

Related concepts:

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

Related tasks:

"Using message mappings" on page 2228
Message mappings define the blueprint for creating a message.

Related reference:

"Mapping node" on page 4571
Use the Mapping node to construct one or more new messages and populate them with various types of information.

ESQL reference

SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

This section covers the following topics:

"Syntax diagrams" on page 3677
This describes the formats that are available for viewing ESQL syntax diagrams.

"ESQL data types in message flows" on page 5020
This describes the valid data types for ESQL.

"ESQL field reference overview" on page 5049
This topic describes the syntax of field references.

"Special characters, case sensitivity, and comments in ESQL" on page 5305
This describes the special characters you use when writing ESQL statements.

"ESQL operators" on page 5056
This describes the operators that are available.

"ESQL reserved keywords" on page 5307
This lists the reserved keywords which you cannot use for variable names.

"ESQL non-reserved keywords" on page 5307
This lists the keywords that are not reserved, as well as those reserved for future releases, which you can use if you choose.

"ESQL functions" on page 5168
This topic lists the functions available in ESQL, and what they do.

"ESQL constants" on page 5302
This topic lists the constants available in ESQL, and what they do.

"ESQL statements" on page 5067
This topic lists the different statement types available in ESQL, and what they do.

"Calling ESQL functions" on page 5168
This topic describes all the ESQL functions in detail.

"ESQL variables" on page 5048
This topic describes the types of ESQL variable and their lifetimes.

"Broker properties that are accessible from ESQL and Java" on page 5302
This topic lists the broker attributes that can be accessed from ESQL code.

An XML format message that is used in many of the ESQL examples in these topics is shown in “Example message” on page 5311.

For information about how you can use ESQL statements and functions to configure Compute, Database, and Filter nodes, see “Writing ESQL” on page 2413.

Syntax diagrams:
The syntax for commands and ESQL statements and functions is presented in the form of a railroad diagram. The diagram tells you what you can do with the command, statement, or function and indicates relationships between different options and, sometimes, different values of an option.

For details about how to read a railroad diagram, see “How to read railroad diagrams” on page 3677.

Related reference:

“Commands” on page 3672
All WebSphere Message Broker Toolkit and runtime commands that are provided on distributed systems are listed, grouped by function, with references to command details.

ESQL data types in message flows:

All data that is referred to in message flows must be one of the defined types.

The defined types are:
- “ESQL BOOLEAN data type” on page 5021
- “ESQL datetime data types” on page 5021
- “ESQL NULL data type” on page 5032
- “ESQL numeric data types” on page 5033
- “ESQL REFERENCE data type” on page 5037
- “ESQL ROW data type” on page 5038
- “ESQL string data types” on page 5040

Related concepts:

“Message flows overview” on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

“ESQL data types” on page 2373
A data type defines the characteristics of an item of data, and determines how that data is processed. ESQL supports six data types, listed later in this section. Data that is retrieved from databases, received in a self-defining message, or defined in a message model (using MRM data types), is mapped to one of these basic ESQL types when it is processed in ESQL expressions.

Related tasks:

Chapter 9, “Developing message flow applications,” on page 1019
Develop message flows to process your business messages and data.

Related reference:

“DECLARE statement” on page 5117
Use the DECLARE statement to define a variable, the data type of the variable and, optionally, its initial value.

“ESQL-to-Java data-type mapping table” on page 5043
Table summarizing the mappings from ESQL to Java.

ESQL BOOLEAN data type:

The BOOLEAN data type holds a Boolean value.

The Boolean value can have the following values:
- TRUE
- FALSE
- UNKNOWN

Boolean literals consist of the keywords TRUE, FALSE, and UNKNOWN. The literals can appear in uppercase or lowercase. For further information about UNKNOWN, see the "IF statement" on page 5134.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL data types" on page 2373
A data type defines the characteristics of an item of data, and determines how that data is processed. ESQL supports six data types, listed later in this section. Data that is retrieved from databases, received in a self-defining message, or defined in a message model (using MRM data types), is mapped to one of these basic ESQL types when it is processed in ESQL expressions.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"DECLARE statement" on page 5117
Use the DECLARE statement to define a variable, the data type of the variable and, optionally, its initial value.

"IF statement" on page 5134
The IF statement executes one set of statements based on the result of evaluating condition expressions.

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

ESQL datetime data types:

ESQL supports several data types that handle datetime values.

The following data types are collectively known as **datetime** data types:
- "ESQL DATE data type" on page 5022
- "ESQL TIME data type" on page 5023
- "ESQL GMTTIME data type" on page 5024
- "ESQL TIMESTAMP data type" on page 5025
- "ESQL GMTTIMESTAMP data type" on page 5026
- "ESQL INTERVAL data type" on page 5027

For information about datetime functions see "ESQL datetime functions" on page 5176.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an

input message is received.

"ESQL data types" on page 2373
A data type defines the characteristics of an item of data, and determines how that data is processed. ESQL supports six data types, listed later in this section. Data that is retrieved from databases, received in a self-defining message, or defined in a message model (using MRM data types), is mapped to one of these basic ESQL types when it is processed in ESQL expressions.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Using numeric operators with datetime values" on page 2439
The following examples show the ESQL that you can code to manipulate datetime values with numeric operators.

"Calculating a time interval" on page 2441
You can use ESQL to calculate the time interval between two events, and to set a timer to be triggered after a specified interval.

Related reference:

"ESQL DATE data type"
The `DATE` data type holds a Gregorian calendar date (year, month, and day).

"ESQL GMTTIME data type" on page 5024
The `GMTTIME` data type is similar to the `TIME` data type, except that its values are interpreted as values in Greenwich Mean Time.

"ESQL GMTTIMESTAMP data type" on page 5026
The `GMTTIMESTAMP` data type is similar to the `TIMESTAMP` data type, except that the values are interpreted as values in Greenwich Mean Time.

"ESQL INTERVAL data type" on page 5027
The `INTERVAL` data type holds an interval of time.

"ESQL TIME data type" on page 5023
The `TIME` data type holds a time of day in hours, minutes, seconds, and fractions of a second.

"ESQL TIMESTAMP data type" on page 5025
The `TIMESTAMP` data type holds a DATE and a TIME in years, months, days, hours, minutes, seconds, and fractions of a second.

ESQL DATE data type:

The `DATE` data type holds a Gregorian calendar date (year, month, and day).

The format of a `DATE` literal is the word `DATE` followed by a space, followed by a date in single quotation marks in the form 'yyyy-MM-dd'. For example:

```
DECLARE MyDate DATE;
SET MyDate = DATE '2000-02-29';
```

Do not omit leading zeros from the year, month, and day.

Related concepts:

"ESQL data types" on page 2373
A data type defines the characteristics of an item of data, and determines how that data is processed. ESQL supports six data types, listed later in this section. Data that is retrieved from databases, received in a self-defining message, or defined in a message model (using MRM data types), is mapped to one of these basic ESQL types when it is processed in ESQL expressions.

Related reference:

“ESQL TIME data type”
The TIME data type holds a time of day in hours, minutes, seconds, and fractions of a second.

“ESQL GMTTIME data type” on page 5024
The GMTTIME data type is similar to the TIME data type, except that its values are interpreted as values in Greenwich Mean Time.

“ESQL TIMESTAMP data type” on page 5025
The TIMESTAMP data type holds a DATE and a TIME in years, months, days, hours, minutes, seconds, and fractions of a second.

“ESQL GMTTIMESTAMP data type” on page 5026
The GMTTIMESTAMP data type is similar to the TIMESTAMP data type, except that the values are interpreted as values in Greenwich Mean Time.

“ESQL INTERVAL data type” on page 5027
The INTERVAL data type holds an interval of time.

“DECLARE statement” on page 5117
Use the DECLARE statement to define a variable, the data type of the variable and, optionally, its initial value.

ESQL TIME data type:

The TIME data type holds a time of day in hours, minutes, seconds, and fractions of a second.

The format of a TIME literal is the word TIME followed by a space, followed by a time in single quotation marks in the form 'hh:mm:ss.ffffff'. For example:

```
DECLARE MyTime TIME;
SET MyTime = TIME '11:49:23.656';
```

Each of the hour, minute, and second fields in a TIME literal must always be two digits; the optional fractional seconds field can be up to 6 digits in length.

The PutTime reported by WebSphere MQ on z/OS and other times or timestamps can be inconsistent if the CVT field is not set correctly.

Related concepts:

“Message flows overview” on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

“ESQL data types” on page 2373
A data type defines the characteristics of an item of data, and determines how that data is processed. ESQL supports six data types, listed later in this section. Data that is retrieved from databases, received in a self-defining message, or defined in a message model (using MRM data types), is mapped to one of these basic ESQL types when it is processed in ESQL expressions.

“Message mapping overview” on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

Related tasks:

Chapter 9, “Developing message flow applications,” on page 1019
Develop message flows to process your business messages and data.

“Developing ESQL” on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"ESQL DATE data type" on page 5022
The DATE data type holds a Gregorian calendar date (year, month, and day).

"ESQL GMTTIME data type"
The GMTTIME data type is similar to the TIME data type, except that its values are interpreted as values in Greenwich Mean Time.

"ESQL TIMESTAMP data type" on page 5025
The TIMESTAMP data type holds a DATE and a TIME in years, months, days, hours, minutes, seconds, and fractions of a second.

"ESQL GMTTIMESTAMP data type" on page 5026
The GMTTIMESTAMP data type is similar to the TIMESTAMP data type, except that the values are interpreted as values in Greenwich Mean Time.

"ESQL INTERVAL data type" on page 5027
The INTERVAL data type holds an interval of time.

"DECLARE statement" on page 5117
Use the DECLARE statement to define a variable, the data type of the variable and, optionally, its initial value.

ESQL GMTTIME data type:

The GMTTIME data type is similar to the TIME data type, except that its values are interpreted as values in Greenwich Mean Time.

GMTTIME literals are defined in a similar way to TIME values. For example:

```
DECLARE MyGetGmttime GMTTIME;
SET MyGetGmttime = GMTTIME '12:00:00';
```

The PutTime reported by WebSphere MQ on z/OS and other times or timestamps can be inconsistent if the CVT field is not set correctly.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL data types" on page 2373
A data type defines the characteristics of an item of data, and determines how that data is processed. ESQL supports six data types, listed later in this section. Data that is retrieved from databases, received in a self-defining message, or defined in a message model (using MRM data types), is mapped to one of these basic ESQL types when it is processed in ESQL expressions.

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"ESQL DATE data type" on page 5022
The DATE data type holds a Gregorian calendar date (year, month, and day).

"ESQL TIME data type" on page 5023
The TIME data type holds a time of day in hours, minutes, seconds, and fractions of a second.

"ESQL TIMESTAMP data type"
The TIMESTAMP data type holds a DATE and a TIME in years, months, days, hours, minutes, seconds, and fractions of a second.

"ESQL GMTTIMESTAMP data type" on page 5026
The GMTTIMESTAMP data type is similar to the TIMESTAMP data type, except that the values are interpreted as values in Greenwich Mean Time.

"ESQL INTERVAL data type" on page 5027
The INTERVAL data type holds an interval of time.

"DECLARE statement" on page 5117
Use the DECLARE statement to define a variable, the data type of the variable and, optionally, its initial value.

ESQL TIMESTAMP data type:

The TIMESTAMP data type holds a DATE and a TIME in years, months, days, hours, minutes, seconds, and fractions of a second.

The format of a TIMESTAMP literal is the word TIMESTAMP followed by a space, followed by a time stamp in single quotation marks in the form 'yyyy-MM-dd HH:mm:ss.SSSSSS'. For example:

```
DECLARE MyTimeStamp TIMESTAMP;
SET MyTimeStamp = TIMESTAMP '1999-12-31 23:59:59';
```

The year field must always be four digits in length. The month, day, hour, and minute fields must always be two digits. (Do not omit leading zeros.) The optional fractional seconds field can be 0 - 6 digits long.

For a description of the characters used when formatting a time stamp in the ESQL CAST function, see "Formatting and parsing dateTimes as strings" on page 5253

The PutTime reported by WebSphere MQ on z/OS and other times or time stamps can be inconsistent if the CVT field is not set correctly.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL data types" on page 2373
A data type defines the characteristics of an item of data, and determines how that data is processed. ESQL supports six data types, listed later in this section. Data that is retrieved from databases, received in a self-defining message, or defined in a message model (using MRM data types), is mapped to one of these basic ESQL types when it is processed in ESQL expressions.

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

Related tasks:

Chapter 9, “Developing message flow applications,” on page 1019
Develop message flows to process your business messages and data.

“Developing ESQL” on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

“Built-in nodes” on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

“ESQL DATE data type” on page 5022
The `DATE` data type holds a Gregorian calendar date (year, month, and day).

“ESQL TIME data type” on page 5023
The `TIME` data type holds a time of day in hours, minutes, seconds, and fractions of a second.

“ESQL GMTTIME data type” on page 5024
The `GMTTIME` data type is similar to the `TIME` data type, except that its values are interpreted as values in Greenwich Mean Time.

“ESQL GMTTIMESTAMP data type”
The `GMTTIMESTAMP` data type is similar to the `TIMESTAMP` data type, except that the values are interpreted as values in Greenwich Mean Time.

“ESQL INTERVAL data type” on page 5027
The `INTERVAL` data type holds an interval of time.

“DECLARE statement” on page 5117
Use the DECLARE statement to define a variable, the data type of the variable and, optionally, its initial value.

ESQL GMTTIMESTAMP data type:

The `GMTTIMESTAMP` data type is similar to the `TIMESTAMP` data type, except that the values are interpreted as values in Greenwich Mean Time.

`GMTTIMESTAMP` values are defined in a similar way to `TIMESTAMP` values, for example:

```
DECLARE MyGetGMTTimeStamp GMTTIMESTAMP;
SET MyGetGMTTimeStamp = GMTTIMESTAMP '1999-12-31 23:59:59.999999';
```

The PutTime reported by WebSphere MQ on z/OS and other times or timestamps can be inconsistent if the CVT field is not set correctly.

Related concepts:

“Message flows overview” on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

“ESQL data types” on page 2373
A data type defines the characteristics of an item of data, and determines how that data is processed. ESQL supports six data types, listed later in this section. Data that is retrieved from databases, received in a self-defining message, or defined in a message model (using MRM data types), is mapped to one of these basic ESQL types when it is processed in ESQL expressions.

“Message mapping overview” on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

Related tasks:

Chapter 9, “Developing message flow applications,” on page 1019
Develop message flows to process your business messages and data.

“Developing ESQL” on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

“Built-in nodes” on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

“ESQL DATE data type” on page 5022
The `DATE` data type holds a Gregorian calendar date (year, month, and day).

“ESQL TIME data type” on page 5023
The `TIME` data type holds a time of day in hours, minutes, seconds, and fractions of a second.

“ESQL GMTTIME data type” on page 5024
The `GMTTIME` data type is similar to the `TIME` data type, except that its values are interpreted as values in Greenwich Mean Time.

“ESQL TIMESTAMP data type” on page 5025
The `TIMESTAMP` data type holds a DATE and a TIME in years, months, days, hours, minutes, seconds, and fractions of a second.

“ESQL INTERVAL data type”
The `INTERVAL` data type holds an interval of time.

“DECLARE statement” on page 5117
Use the DECLARE statement to define a variable, the data type of the variable and, optionally, its initial value.

ESQL INTERVAL data type:

The `INTERVAL` data type holds an interval of time.

This data type has a number of subtypes:
- YEAR
- YEAR TO MONTH
- MONTH
- DAY
- DAY TO HOUR
- DAY TO MINUTE
- DAY TO SECOND
- HOUR
- HOUR TO MINUTE
- HOUR TO SECOND
- MINUTE
- MINUTE TO SECOND
- SECOND

All these subtypes describe intervals of time and all can take part in the full range of operations of the INTERVAL type; for example, addition and subtraction operations with values of type DATE, TIME, or TIMESTAMP.

Use the CAST function to convert from one subtype to another, except for intervals described in years and months, or months, which cannot be converted to those described in days, hours, minutes, and seconds.

The split between months and days arises because the number of days in each month varies. An interval of one month and a day is not meaningful, and cannot be sensibly converted into an equivalent interval in numbers of days only.

An interval literal is defined by the syntax:

```
INTERVAL <interval string> <interval qualifier>
```

The format of interval string and interval qualifier are defined by the following table.

Interval qualifier	Interval string format	Example
YEAR	'<year>' or '<sign> <year>'	'10' or '-10'
YEAR TO MONTH	'<year>-<month>' or '<sign> <year>-<month>'	'2-06' or '- 2-06'
MONTH	'<month>' or '<sign> <month>'	'18' or '-18'
DAY	'<day>' or '<sign> <day>'	'30' or '-30'
DAY TO HOUR	'<day> <hour>' or '<sign> <day> <hour>'	'1 02' or '-1 02'
DAY TO MINUTE	'<day> <hour>:<minute>' or '<sign> <day> <hour>:<minute>'	'1 02:30' or '-1 02:30'
DAY TO SECOND	'<day> <hour>:<minute>:<second>' or '<sign> <day> <hour>:<minute>:<second>'	'1 02:30:15' or '-1 02:30:15.333'
HOUR	'<hour>' or '<sign> <hour>'	'24' or '-24'
HOUR TO MINUTE	'<hour>:<minute>' or '<sign> <hour>:<minute>'	'1:30' or '-1:30'
HOUR TO SECOND	'<hour>:<minute>:<second>' or '<sign> <hour>:<minute>:<second>'	'1:29:59' or '-1:29:59.333'
MINUTE	'<minute>' or '<sign> <minute>'	'90' or '-90'
MINUTE TO SECOND	'<minute>:<second>' or '<sign> <minute>:<second>'	'89:59' or '-89:59'
SECOND	'<second>' or '<sign> <second>'	'15' or '-15.7'

Where an interval contains both a year and a month value, a hyphen is used between the two values. In this instance, the month value must be within the range `[0, 11]`. If an interval contains a month value and no year value, the month value is unconstrained.

A space is used to separate days from the rest of the interval.

If an interval contains more than one of HOUR, MINUTE, and SECOND, a colon is needed to separate the values and all except the leftmost are constrained as follows:

HOUR
: 0-23

MINUTE
: 0-59

SECOND
: 0-59.999...

The largest value of the left-most value in an interval is +/- 2147483647.

Some examples of valid interval values are:

- 72 hours
- 3 days: 23 hours

- 3600 seconds
- 90 minutes: 5 seconds

Some examples of invalid interval values are:

- 3 days: 36 hours

  A day field is specified, so the hours field is constrained to `[0,23]`.
- 1 hour: 90 minutes

  An hour field is specified, so minutes are constrained to `[0,59]`.

Here are some examples of interval literals:

```
INTERVAL '1' HOUR
INTERVAL '90' MINUTE
INTERVAL '1-06' YEAR TO MONTH
```

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL data types" on page 2373
A data type defines the characteristics of an item of data, and determines how that data is processed. ESQL supports six data types, listed later in this section. Data that is retrieved from databases, received in a self-defining message, or defined in a message model (using MRM data types), is mapped to one of these basic ESQL types when it is processed in ESQL expressions.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Using numeric operators with datetime values" on page 2439
The following examples show the ESQL that you can code to manipulate datetime values with numeric operators.

"Calculating a time interval" on page 2441
You can use ESQL to calculate the time interval between two events, and to set a timer to be triggered after a specified interval.

Related reference:

"ESQL DATE data type" on page 5022
The `DATE` data type holds a Gregorian calendar date (year, month, and day).

"ESQL TIME data type" on page 5023
The `TIME` data type holds a time of day in hours, minutes, seconds, and fractions of a second.

"ESQL GMTTIME data type" on page 5024
The `GMTTIME` data type is similar to the `TIME` data type, except that its values are interpreted as values in Greenwich Mean Time.

"ESQL TIMESTAMP data type" on page 5025
The `TIMESTAMP` data type holds a DATE and a TIME in years, months, days, hours, minutes, seconds, and fractions of a second.

"ESQL GMTTIMESTAMP data type" on page 5026
The `GMTTIMESTAMP` data type is similar to the `TIMESTAMP` data type, except that the values are interpreted as values in Greenwich Mean Time.

"DECLARE statement" on page 5117
Use the DECLARE statement to define a variable, the data type of the variable and, optionally, its initial value.

"CAST function" on page 5245

Representation of ESQL datetime data types:

When your application sends a message to a broker, the way in which the message data is interpreted depends on the content of the message itself and the configuration of the message flow. If your application sends a message to be interpreted either by the generic XML parser, or the MRM parser, that is tailored by an XML physical format, the application can include date or time data that is represented by any of the XML Schema primitive datetime data types.

The XML Schema data type of each piece of data is converted to an ESQL data type, and the element that is created in the logical message tree is of the converted type. If the datetime data in an input message does not match the rules of the chosen schema data type, the values that the parser writes to the logical message tree are modified even if the message is in the MRM domain and you have configured the message flow to validate the input message. (Validation is not available for generic XML messages.)

This has the following effect on the subfields of the input datetime data:

- If any of the subfields of the input message are missing, a default value is written to the logical message tree. This default is substituted from the full timestamp that refers to the beginning of the current epoch: 1970-01-01 00:00:00.
- If the input message contains information for subfields that are not present in the schema, the additional data is discarded. If this occurs, no exception is raised, even if a message in the MRM domain is validated.
- After the data is parsed, it is cast to one of three ESQL datetime data types. These are DATE, TIME, and TIMESTAMP.
  - If a datetime value contains only date subfields, it is cast to an ESQL DATE.
  - If a datetime value contains only time subfields, it is cast to an ESQL TIME.
  - If a datetime value contains both date and time subfields, it is cast to an ESQL TIMESTAMP.

The following examples illustrate these points.

Input data XML Schema data type	**Schema rules**	**Input value in the bit stream**	**Value written to the logical tree (ESQL data type in brackets)**
xsd:dateTime	CCYY-MM-DDThh:mm:ss	2002-12-31T23:59:59	2002-12-31 23:59:59 (TIMESTAMP)
		--24	1970-01-24 (DATE)
		23:59:59	23:59:59 (TIME)
xsd:date	CCYY-MM-DD	2002-12-31	2002-12-31 (DATE)
		2002-12-31T23:59:59	2002-12-31 (DATE)
		-06-24	1970-06-24 (DATE)
xsd:time	hh:mm:ss	14:15:16	14:15:16 (TIME)
xsd:gDay	---DD	---24	1970-01-24 (DATE)
xsd:gMonth	--MM	--12	1970-12-01 (DATE)
xsd:gMonthDay	--MM-DD	--12-31	1970-12-31 (DATE)
xsd:gYear	CCYY	2002	2002-01-01 (DATE)
xsd:gYearMonth	CCYY-MM	2002-12	2002-12-01 (DATE)

Validation with missing subfields: When you consider which schema datetime data type to use, consider that, if the message is in the MRM domain, and you configure the message flow to validate messages, missing subfields can cause validation exceptions.

The schema data types Gday, gMonth, gMonthDay, gYear, and gYearMonth are used to record particular recurring periods of time. There is potential confusion when validation is turned on, because the recurring periods of time that are used in these schema data types are stored by ESQL as specific points in time.

For example, when the 24th of the month, which is a gDay (a monthly day) type, is written to the logical tree, the missing month and year subfields are supplied from the epoch (January 1970) to provide the specific date 1970-01-24. If you code ESQL to manipulate this date, for example by adding an interval of 10 days, and then generate an output message that is validated, an exception is raised. This is because the result of the calculation is 1970-02-03 which is invalid because the month subfield of the date no longer matches the epoch date.

Use within an MRM domain: In MRM it is possible to define an element that has the logical type of dateTime.

When a dateTime element is parsed, a field is created in the message tree that has the ESQL datatype of CURRENT_TIME or CURRENT_TIMESTAMP. However, the CURRENT_TIME and CURRENT_TIMESTAMP data types do not have the functionality to store timezone information, and the MRM does not adjust the time according to the input timezone and the timezone of the broker.

Although the CURRENT_TIME and CURRENT_TIMESTAMP data types cannot store timezone information, the MRM stores this information as part of the underlying field. This means that if the field is copied between message trees, the timezone information is copied with it, allowing this information to be preserved on output.

Note that the information is preserved only if the field is copied to a field of the same name.

However, if any new field is derived from the original field, the new field does not have the timezone information. This means that if such a field is cast as a character, the new field assumes the timezone of the broker, but its value is not adjusted for any difference between the input timezone and the timezone of the broker.

For example, an input dateTime element containing `2009-02-20T06:08:07-08:00` could be copied from the input message tree to the output message tree and appear in an output message in exactly the same format. However, if the element is cast as character, using format `IU`, by a broker running GMT the result would be `2009-02-20T06:08:07.000Z`.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL data types" on page 2373
A data type defines the characteristics of an item of data, and determines how that data is processed. ESQL supports six data types, listed later in this section. Data that is retrieved from databases, received in a self-defining message, or defined in a message model (using MRM data types), is mapped to one of these basic ESQL types when it is processed in ESQL expressions.

Related tasks:

"Defining input message characteristics" on page 1475
When a message is received by an input node in a message flow, the node detects how to interpret that message by determining the domain in which the message is defined and starting the appropriate parser.

"Validating messages" on page 1478
The broker provides validation based on the message model for predefined messages.

Related reference:

"ESQL datetime functions" on page 5176

"DateTime formats" on page 6310
When you create an element or attribute with a simple type of dateTime, you must specify a format string in the object's *Format String* property for each physical format layer (CWF, TDS, XML).

"CURRENT_TIME function" on page 5179

"CURRENT_TIMESTAMP function" on page 5180

ESQL NULL data type:

All ESQL data types (except REFERENCE) support the concept of the null value. A value of null means that the value is unknown, undefined, or uninitialized. Null values can arise when you refer to message fields that do not exist, access database columns for which no data has been supplied, or use the keyword NULL, which supplies a null literal value.

Null is a distinct state and is not the same as any other value. In particular, for integers it is not the same thing as the value 0 and for character variables it is not the same thing as a string of zero characters. The rules of ESQL arithmetic take null values into account, and you are typically unaware of their existence. Generally, but not always, these rules mean that, if any operand is null, the result is null.

If an expression returns a null value its data type is not, in general, known. All null values, whatever their origin, are therefore treated equally.

This can be regarded as their belonging to the data type NULL , which is a data type that can have just one value, null.

An expression always returns NULL if any of its elements are NULL.

Testing for null values

To test whether a field contains a null value, use the IS operator described in **Operator=**.

The effect of setting a field to NULL:
Take care when assigning a null value to a field. For example, the following command *deletes* the `Name` field:

```
SET OutputRoot.XMLNS.Msg.Data.Name = NULL;  -- this deletes the field
```

The correct way to assign a null value to a field is as follows:

```
SET OutputRoot.XMLNS.Msg.Data.Name VALUE = NULL;
-- this assigns a NULL value to a field without deleting it
```

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL data types" on page 2373
A data type defines the characteristics of an item of data, and determines how that data is processed. ESQL supports six data types, listed later in this section. Data that is retrieved from databases, received in a self-defining message, or defined in a message model (using MRM data types), is mapped to one of these basic ESQL types when it is processed in ESQL expressions.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"DECLARE statement" on page 5117
Use the DECLARE statement to define a variable, the data type of the variable and, optionally, its initial value.

ESQL numeric data types:

ESQL supports several data types that handle numeric values, including DECIMAL, FLOAT, and INTEGER.

The following data types are collectively known as **numeric** data types:
- "ESQL DECIMAL data type" on page 5034
- "ESQL FLOAT data type" on page 5036
- "ESQL INTEGER data type" on page 5037

Notes:

1. INTEGER and DECIMAL types are represented exactly inside the broker; FLOAT types are inherently subject to rounding error without warning. Do not use FLOAT if you need absolute accuracy, for example, to represent money.
2. Various casts are possible between different numeric types. These can result in loss of precision, if exact types are cast into FLOAT.

For information about numeric functions see "ESQL numeric functions" on page 5183.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL data types" on page 2373
A data type defines the characteristics of an item of data, and determines how that data is processed. ESQL supports six data types, listed later in this section. Data that is retrieved from databases, received in a self-defining message, or defined in a message model (using MRM data types), is mapped to one of these basic ESQL types when it is processed in ESQL expressions.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"ESQL DECIMAL data type"

"ESQL FLOAT data type" on page 5036

"ESQL INTEGER data type" on page 5037

"ESQL numeric functions" on page 5183
A list of the numeric functions that ESQL supports.

ESQL DECIMAL data type:
The **DECIMAL** data type holds an exact representation of a decimal number. Decimals have precision, scale, and rounding. Precision is the total number of digits of a number:

- The minimum precision is 1
- The maximum precision is 34

Scale is the number of digits to the right of the decimal point:

- The minimum scale (-exponent) is -999,999,999
- The maximum scale (-exponent) is +999,999,999

You cannot define precision and scale when declaring a DECIMAL, because they are assigned automatically. It is only possible to specify precision and scale when casting to a DECIMAL.

Scale, precision, and rounding:
The following scale, precision, and rounding rules apply:

- Unless rounding is required to keep within the maximum precision, the scale of the result of an addition or subtraction is the greater of the scales of the two operands.
- Unless rounding is required to keep within the maximum precision, the scale of the result of a multiplication is the sum of the scales of the two operands.
- The precision of the result of a division is the smaller of the number of digits needed to represent the result exactly and the maximum precision.
- All addition, subtraction, multiplication, and division calculations round the least significant digits, as necessary, to stay within the maximum precision
- All automatic rounding is *banker's* or *half even symmetric* rounding. The rules of this are:
  - When the first dropped digit is 4 or less, the first retained digit is unchanged
  - When the first dropped digit is 6 or more, the first retained digit is incremented

- When the first dropped digit is 5, the first retained digit is incremented if it is odd, and unchanged if it is even. Therefore, both 1.5 and 2.5 round to 2 while 3.5 and 4.5 both round to 4
- Negative numbers are rounded according to the same rule

Decimal literals:
Decimal literals that consist of an unquoted string of digits only, that is, that contain neither a decimal point nor an exponent (for example 12345) are of type INTEGER if they are small enough to be represented as integers. Otherwise they are of type DECIMAL.

Decimal literals that consist of an unquoted string of digits, optionally a decimal point, and an exponent (for example 123e1), are of type FLOAT if they are small enough to be represented as floats. Otherwise they are of type DECIMAL.

Decimal literals that consist of the keyword DECIMAL and a quoted string of digits, with or without a decimal point and with or without an exponent, are of type DECIMAL, for example, DECIMAL '42', DECIMAL '1.2346789e+203'.

The strings in this type of literal can also have the values:
- 'NAN', not a number
- 'INF', 'INFINITY'
- '+INF', '+INFINITY'
- '-INF', '-INFINITY'
- 'MAX'
- 'MIN'

(in any mixture of case) to denote the corresponding values.

Note, if you do not specify sufficient precision digits, that INF is returned, as shown in the following example:

```
SET VAL = CAST('123456' AS DECIMAL(3,0))
```

Related concepts:

“Message flows overview” on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

“ESQL data types” on page 2373
A data type defines the characteristics of an item of data, and determines how that data is processed. ESQL supports six data types, listed later in this section. Data that is retrieved from databases, received in a self-defining message, or defined in a message model (using MRM data types), is mapped to one of these basic ESQL types when it is processed in ESQL expressions.

“Message mapping overview” on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

Related tasks:

Chapter 9, “Developing message flow applications,” on page 1019
Develop message flows to process your business messages and data.

“Developing ESQL” on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"ESQL INTEGER data type" on page 5037

"ESQL FLOAT data type"

"DECLARE statement" on page 5117
Use the DECLARE statement to define a variable, the data type of the variable and, optionally, its initial value.

ESQL FLOAT data type:
The FLOAT data type holds a 64-bit, base 2, fraction and exponent approximation to a real number. This gives a range of values between +-1.7E–308 and +- 1.7E+308.

Float literals consist of an unquoted string of digits and either a decimal point (for example 123.4) or an exponent (for example 123e4) or both (for example 123.4e5) . They are of type FLOAT if they are small enough to be represented as floats. Otherwise they are of type DECIMAL

Rounding:
When you CAST a FLOAT to an INTEGER, either implicitly or explicitly, the FLOAT is truncated; that is, the numbers after the decimal point are removed and no rounding occurs.

When you CAST a FLOAT to a DECIMAL or CHARACTER, either implicitly or explicitly, the FLOAT can be rounded to a maximum precision of 15 digits.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL data types" on page 2373
A data type defines the characteristics of an item of data, and determines how that data is processed. ESQL supports six data types, listed later in this section. Data that is retrieved from databases, received in a self-defining message, or defined in a message model (using MRM data types), is mapped to one of these basic ESQL types when it is processed in ESQL expressions.

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"ESQL INTEGER data type" on page 5037

"ESQL DECIMAL data type" on page 5034

"DECLARE statement" on page 5117
Use the DECLARE statement to define a variable, the data type of the variable

and, optionally, its initial value.

ESQL INTEGER data type:
The INTEGER data type holds an integer number in 64-bit two's complement form. This gives a range of values between -9223372036854775808 and +9223372036854775807.

Integer literals consist of an unquoted string of digits only; that is, they contain neither a decimal point nor an exponent; for example, 12345. They are of type INTEGER if they are small enough to be represented as integers. Otherwise they are of type DECIMAL.

In addition to this format, you can write integer literals in hexadecimal notation; for example, 0x1234abcd. You can write the hexadecimal letters A to F, and the "x" after the initial zero, in uppercase or lowercase. If you use hexadecimal format, the number must be small enough to fit into an integer. (That is, it cannot be a decimal.)

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL data types" on page 2373
A data type defines the characteristics of an item of data, and determines how that data is processed. ESQL supports six data types, listed later in this section. Data that is retrieved from databases, received in a self-defining message, or defined in a message model (using MRM data types), is mapped to one of these basic ESQL types when it is processed in ESQL expressions.

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"ESQL FLOAT data type" on page 5036

"ESQL DECIMAL data type" on page 5034

"DECLARE statement" on page 5117
Use the DECLARE statement to define a variable, the data type of the variable and, optionally, its initial value.

ESQL REFERENCE data type:

The REFERENCE data type holds the location of a field in a message. It cannot hold the location of a constant, a database table, a database column, or another reference.

For compatibility with earlier versions, reference variables can also point at scalar variables

A reference literal is an hierarchic path name, consisting of a list of path elements separated by periods. The first element in the list is known as the correlation name, and identifies a reference, row, or scalar variable. Any subsequent elements apply to references to message trees only, and identify field types, names, and indexes within the message tree relative to the field pointed to by the correlation name.

Note: If you use a REFERENCE, you are able to modify any element, even if the element is part of an input tree.

For example:

```
InputRoot.MQMD.Priority
```

is a field reference literal that refers to the Priority field contained within an MQMD structure within an input message.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL data types" on page 2373
A data type defines the characteristics of an item of data, and determines how that data is processed. ESQL supports six data types, listed later in this section. Data that is retrieved from databases, received in a self-defining message, or defined in a message model (using MRM data types), is mapped to one of these basic ESQL types when it is processed in ESQL expressions.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"ESQL field reference overview" on page 5049
You can use ESQL field references to form paths to message body elements.

"DECLARE statement" on page 5117
Use the DECLARE statement to define a variable, the data type of the variable and, optionally, its initial value.

ESQL ROW data type:

The ROW data type holds a *tree structure*. A row in a database is a particular type of tree structure, but the ROW data type is not restricted to holding data from database rows.

In a database, a row is a fixed, ordered, set of scalar values. A *scalar* is defined as a single entity value or a string.

A database table is an unordered set of rows and therefore represents a two dimensional "array" of scalar values, in which one dimension is fixed and the other is variable. In ESQL, a row is an open-ended, ordered, set of named values in

which each value can be scalar, or another row. That is, a row is an open-ended tree structure with no restrictions on dimensions or regularity. Consider the following diagram:

```
Root
  Row
    PartNumber  = 1
    Description = 'Chocolate bar'
    Price       = 0.30
  Row
    PartNumber  = 2
    Description = 'Biscuit'
    Price       = 0.35
  Row
    PartNumber  = 3
    Description = 'Fruit'
    Price       = 0.42
```

In the example, *Root* contains three elements all named "Row". Each of these elements in turn contains three elements with different names and values. This diagram equally describes an instance of an ESQL row data type (that is, a tree structure) or the contents of a database table.

ROW and LIST

The ROW data type is a normal data type. You can use the DECLARE statement to create ROW variables in the same way as you create INTEGER or CHARACTER variables. There is also a more general concept of a ROW data type. In the previous example, *Root* is the root element of a ROW variable. Each of the elements called "Row", while not the root element of ROW variables, is a root element of substructures. Many ESQL operations (and particularly the SELECT function) work with the general concept of ROW and operate equally on whole trees or parts of trees.

There is also a general concept of a LIST data type. The set of elements called "Row" can be regarded as a list. Some ESQL operations (particularly SELECT) work with the general concept of list.

InputRoot, *OutputRoot*, and so on, are examples of ROW variables that are automatically declared and present in the data structure, ready for use.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL data types" on page 2373
A data type defines the characteristics of an item of data, and determines how that data is processed. ESQL supports six data types, listed later in this section. Data that is retrieved from databases, received in a self-defining message, or defined in a message model (using MRM data types), is mapped to one of these basic ESQL types when it is processed in ESQL expressions.

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"DECLARE statement" on page 5117
Use the DECLARE statement to define a variable, the data type of the variable and, optionally, its initial value.

ESQL string data types:

ESQL supports several data types that handle string values, including BIT, BLOB, and CHARACTER.

The following data types are collectively known as **string** data types:
- "ESQL BIT data type"
- "ESQL BLOB data type" on page 5041
- "ESQL CHARACTER data type" on page 5042

For information about string functions, see "ESQL string manipulation functions" on page 5205.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL data types" on page 2373
A data type defines the characteristics of an item of data, and determines how that data is processed. ESQL supports six data types, listed later in this section. Data that is retrieved from databases, received in a self-defining message, or defined in a message model (using MRM data types), is mapped to one of these basic ESQL types when it is processed in ESQL expressions.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"ESQL BIT data type"

"ESQL BLOB data type" on page 5041

"ESQL CHARACTER data type" on page 5042

"ESQL string manipulation functions" on page 5205
A list of the ESQL string manipulation functions that you can use.

"DECLARE statement" on page 5117
Use the DECLARE statement to define a variable, the data type of the variable and, optionally, its initial value.

ESQL BIT data type:
The BIT data type holds a variable length string of binary digits. It is commonly used to represent arbitrary binary data that does not contain an exact number of bytes. A bit string literal consists of the letter B, followed by a string of binary digits enclosed in single quotation marks, as in the following example:

```
B'0100101001'
```

Any number of digits, which must be either 0 or 1, can be specified. The initial B can be specified in uppercase or lowercase.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL data types" on page 2373
A data type defines the characteristics of an item of data, and determines how that data is processed. ESQL supports six data types, listed later in this section. Data that is retrieved from databases, received in a self-defining message, or defined in a message model (using MRM data types), is mapped to one of these basic ESQL types when it is processed in ESQL expressions.

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"ESQL CHARACTER data type" on page 5042

"ESQL BLOB data type"

"DECLARE statement" on page 5117
Use the DECLARE statement to define a variable, the data type of the variable and, optionally, its initial value.

ESQL BLOB data type:
The BLOB data type holds a variable length string of 8-bit bytes. It is commonly used to represent arbitrary binary data. A BLOB literal consists of the letter X, followed by a string of hexadecimal digits enclosed in single quotation marks, as in the following example:

```
X'0123456789ABCDEF'
```

There must be an even number of digits in the string, because two digits are required to define each byte. Each digit can be one of the hexadecimal digits 0-9 and A-F. Both the initial X and the hexadecimal letters can be specified in uppercase or lowercase.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL data types" on page 2373
A data type defines the characteristics of an item of data, and determines how that data is processed. ESQL supports six data types, listed later in this section. Data that is retrieved from databases, received in a self-defining message, or defined in a message model (using MRM data types), is mapped to one of these basic ESQL types when it is processed in ESQL expressions.

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"ESQL CHARACTER data type"

"ESQL BIT data type" on page 5040

"DECLARE statement" on page 5117
Use the DECLARE statement to define a variable, the data type of the variable and, optionally, its initial value.

ESQL CHARACTER data type:
The character data type holds a variable length string of Unicode characters. A character string literal consists of any number of characters in single quotation marks. If you want to include a single quotation mark within a character string literal, use another single quotation mark as an escape character.

For example, the assignment `SET X='he''was'''` puts the value `he'was'` into X.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL data types" on page 2373
A data type defines the characteristics of an item of data, and determines how that data is processed. ESQL supports six data types, listed later in this section. Data that is retrieved from databases, received in a self-defining message, or defined in a message model (using MRM data types), is mapped to one of these basic ESQL types when it is processed in ESQL expressions.

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"ESQL BLOB data type" on page 5041

"ESQL BIT data type" on page 5040

"DECLARE statement" on page 5117
Use the DECLARE statement to define a variable, the data type of the variable and, optionally, its initial value.

ESQL-to-Java data-type mapping table:

Table summarizing the mappings from ESQL to Java.

The following table summarizes the mappings from ESQL to Java.

Notes:

- Only the Java scalar wrappers are passed to Java.
- The ESQL scalar types are mapped to Java data types as object wrappers, or object wrapper arrays, depending upon the direction of the procedure parameter. Each wrapper array contains exactly one element.
- Scalar object wrappers are used to allow NULL values to be passed to and from Java methods.

ESQL data types [1]	**Java IN data types**	**Java INOUT and OUT data types**
INTEGER, INT	java.lang.Long	java.lang.Long []
FLOAT	java.lang.Double	java.lang.Double[]
DECIMAL	java.math.BigDecimal	java.math.BigDecimal[]
CHARACTER, CHAR	java.lang.String	java.lang.String[]
BLOB	byte[]	byte[][]
BIT	java.util.BitSet	java.util.BitSet[]
DATE	com.ibm.broker.plugin.MbDate	com.ibm.broker.plugin.MbDate[]
TIME [2]	com.ibm.broker.plugin.MbTime	com.ibm.broker.plugin.MbTime[]
GMTTIME [2]	com.ibm.broker.plugin.MbTime	com.ibm.broker.plugin.MbTime[]
TIMESTAMP [2]	com.ibm.broker.plugin.MbTimestamp	com.ibm.broker.plugin.MbTimestamp[]
GMTTIMESTAMP [2]	com.ibm.broker.plugin.MbTimestamp	com.ibm.broker.plugin.MbTimestamp[]
INTERVAL	Not supported	Not supported
BOOLEAN	java.lang.Boolean	java.lang.Boolean[]
REFERENCE (to a message tree) [3] [4] [5] [6]	com.ibm.broker.plugin.MbElement	com.ibm.broker.plugin.MbElement[] **(Supported for INOUT. Not supported for OUT)**
ROW	Not supported	Not supported
LIST	Not supported	Not supported

1. Variables that are declared to be CONSTANT (or references to variables that are declared to be CONSTANT) are not allowed to have the direction INOUT or OUT.
2. The time zone set in the Java variable is not important; you obtain the required time zone in the output ESQL.
3. The reference parameter cannot be NULL when passed into a Java method.
4. The reference cannot have the direction OUT when passed into a Java method.
5. If an *MbElement* is passed back from Java to ESQL as an INOUT parameter, it must point to a location in the same message tree as that pointed to by the *MbElement* that was passed into the called Java method.

For example, if an ESQL reference to `OutputRoot.XML.Test` is passed into a Java method as an INOUT *MbElement*, but a different *MbElement* is passed back to ESQL when the call returns, the different element must also point to somewhere in the `OutputRoot` tree.

6. An *MbElement* cannot be returned from a Java method with the RETURNS clause, because no ESQL routine can return a reference. However, an *MbElement* can be returned as an INOUT direction parameter, subject to the conditions described in point 5.

A REFERENCE to a scalar variable can be used in the CALL of a Java method, provided that the data type of the variable to which the reference refers matches the corresponding data type in the Java program signature.

Related reference:

"CREATE PROCEDURE statement" on page 5103
The CREATE PROCEDURE statement defines a callable function or procedure.

"CREATE FUNCTION statement" on page 5091
The CREATE FUNCTION statement defines a callable function or procedure.

ESQL to XML Schema data type mapping:

Mapping from XML Schema simple type to ESQL message tree data type.

The following table shows the mapping that WebSphere Message Broker parsers use when converting between the simple types of XML Schema and the data types of the message tree. The mapping applies to all parsers that use an XML Schema model when parsing and serializing.

XML Schema simple type	Data type in message tree
anyURI	CHARACTER
base64Binary	BLOB
boolean	BOOLEAN
byte	INTEGER
date	DATE
dateTime	TIMESTAMP
dayTimeDuration	INTERVAL
decimal	DECIMAL
double	FLOAT
duration	INTERVAL
ENTITIES	List of CHARACTER
ENTITY	STRING
float	FLOAT
gDay	DATE
gMonth	DATE
gMonthDay	DATE
gYear	DATE
gYearMonth	DATE
hexBinary	BLOB
ID	CHARACTER

XML Schema simple type	Data type in message tree
IDREF	CHARACTER
IDREFS	List of CHARACTER
int	INTEGER
integer	DECIMAL
language	CHARACTER
long	INTEGER
Name	CHARACTER
NCName	CHARACTER
negativeInteger	DECIMAL
NMTOKEN	CHARACTER
NMTOKENS	List of CHARACTER
nonNegativeInteger	DECIMAL
nonPositiveInteger	DECIMAL
normalizedString	CHARACTER
NOTATION	CHARACTER
positiveInteger	DECIMAL
QName	CHARACTER
short	INTEGER
string	CHARACTER
time	TIME
token	CHARACTER
unsignedByte	INTEGER
unsignedInt	INTEGER
unsignedLong	DECIMAL
unsignedShort	INTEGER
yearMonthDuration	INTERVAL

Related concepts:

"XMLNSC data types" on page 1102
Mapping between XML Schema simple types and the data types that the XMLNSC parser uses in the message tree when `Build tree using XML Schema types` is specified.

Related reference:

"Data types for elements in an MRM message" on page 6254
A parser is supplied for the body of a message in the MRM domain; it associates each field with a specific data type.

"ESQL data types in message flows" on page 5020
All data that is referred to in message flows must be one of the defined types.

ESQL-to-XPath mapping table:

A table that summarizes the mappings from ESQL to XPath.

ESQL	XPath 1.0	XPath 1.0 usage notes
BOOLEAN data types		
True	True()	Equivalent to Boolean "1" or "True"
False	False()	Equivalent to Boolean "0" or "False"
Unknown	No equivalent	
Date Time data types	No equivalent	
NULL data type	No equivalent	
Numeric data types		
DECIMAL	12678967.543233 with or without quotation marks	Cannot express an exponent or a leading plus sign.
FLOAT	1.7976931348623158 with or without quotation marks	Cannot express an exponent or a leading plus sign.
INTEGER	9223372036854775807 with or without quotation marks	Cannot express an exponent or a leading plus sign.
REFERENCE data type	FilterExpression '/' RelativeLocationPath	For example, `$InputRoot/MQMD/Priority`
String data types		
BIT	No equivalent	
BLOB	No equivalent	
CHARACTER	Literal	For example `'a "b"'` or `"a 'b'"`
`NAME`	`$NAME`	Can assign such a variable any valid value, of type Boolean, number, or string.
Simple comparison operators		
>	>	
<	<	
>=	>=	
'='	'='	
<>	!=	
Complex comparison operators	No equivalent	
Logical operators		
AND	and	
OR	or	
NOT	not (operand)	The not function returns true if its argument is false, and false otherwise.
Numeric operators		
Unary -	- Unary expression	
+	+	
-	-	
*	*	Multiplication operator
/	div	
String operator	No equivalent	
Date time functions	No equivalent	

ESQL	XPath 1.0	XPath 1.0 usage notes
Numeric functions		
FLOOR	floor (number)	
CEIL and CEILING	ceiling (number)	
ROUND	No equivalent	
String manipulation functions		
SUBSTRING	substring(string, number, number)	
TRANSLATE	translate(string, string, string)	

Related tasks:

"Using XPath" on page 1506
XPath provides an alternative method to ESQL for entering expressions in the property fields of specific built-in nodes.

XPath property editors:

The XPath files are supplied in three property editors located in the `com.ibm.etools.mft.ibmnodes plugin`.

The property editors are:

Read only
Located in
`com.ibm.etools.mft.ibmnodes.editors.xpath.XPathReadOnlyPropertyEditor`

Read write
Located in
`com.ibm.etools.mft.ibmnodes.editors.xpath.XPathReadWritePropertyEditor`

Expression
Located in
`com.ibm.etools.mft.ibmnodes.editors.xpath.XPathPropertyEditor`

For information on adding a property editor to your workspace, see "Adding a property editor or compiler" on page 3091.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"XPath overview" on page 1507
The XML Path Language (XPath) is used to uniquely identify or address parts of an XML document. An XPath expression can be used to search through an XML document, and extract information from any part of the document, such as an element or attribute (referred to as a *node* in XML) in it. XPath can be used alone or in conjunction with XSLT.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

ESQL variables:

ESQL variables can be described as *external variables*, *normal variables*, or *shared variables*; their use is defined in the DECLARE statement.

Types of variable

External

External variables (defined with the EXTERNAL keyword) are also known as *user-defined properties* (see "User-defined properties in ESQL" on page 2376). They exist for the entire lifetime of a message flow and are visible to all messages that pass through the flow. You can define *external variables* only at the module and schema level. You can modify their initial values (optionally set by the DECLARE statement) by using the Message Flow editor, or at deployment time, by using the Broker Archive editor. You can query and set the values of user-defined properties at run time by using the Administration API for WebSphere Message Broker (also known as the CMP API). For more information, see "Setting message flow user-defined properties at run time in a CMP application" on page 985.

Normal

Normal variables have a lifetime of just one message passing through a node. They are visible to that message only. To define a *normal variable*, omit both the EXTERNAL and SHARED keywords.

Shared

Shared variables (defined with the SHARED keyword) can be used to implement an in-memory cache in the message flow; see "Optimizing message flow response times" on page 3264. *Shared variables* have a long lifetime and are visible to multiple messages that pass through the flow; see "Long-lived variables" on page 2378. They exist for the lifetime of the execution group process, the lifetime of the flow or node, or the lifetime of the node's SQL that declares the variable (whichever is the shortest). They are initialized when the first message passes through the flow or node after each broker startup.

See also the ATOMIC option of the "BEGIN ... END statement" on page 5070. The BEGIN ATOMIC construct is useful when a number of changes have to be made to a shared variable and when it is important to prevent other instances seeing the intermediate states of the data.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"ESQL variables" on page 2374
An ESQL variable is a data field that is used to help process a message.

"User-defined properties in ESQL" on page 2376
Access user-defined properties (UDPs) as variables in your ESQL program by specifying the EXTERNAL keyword on a DECLARE statement. For example, the ESQL statement `DECLARE today EXTERNAL CHARACTER 'monday'` defines a user-defined property called today with an initial value monday.

"User-defined properties" on page 1147
A user-defined property (UDP) is a property that is defined when you construct a message flow by using the Message Flow editor. This property can be used by the ESQL or Java program inside message flow nodes, such as a Compute node.

"Long-lived variables" on page 2378
You can use appropriate long-lived ESQL data types to cache data in memory.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Setting message flow user-defined properties at run time in a CMP application" on page 985
Use the CMP API to query, discover, and set message flow user-defined properties dynamically at run time. You can use the CMP API to set properties with a data type of character.

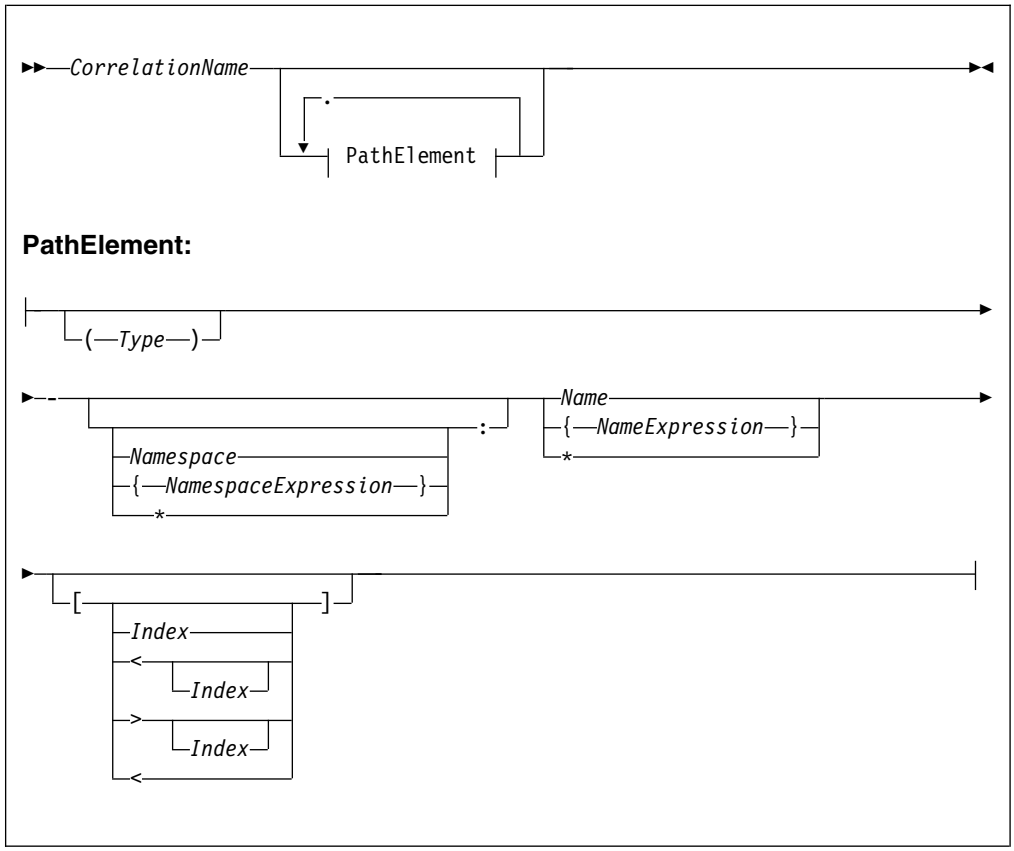
Related reference:

"DECLARE statement" on page 5117
Use the DECLARE statement to define a variable, the data type of the variable and, optionally, its initial value.

ESQL field reference overview:

You can use ESQL field references to form paths to message body elements.

The full syntax for field references is shown in the following examples:



A field reference consists of a correlation name, followed by zero or more Path Elements separated by periods (.). The correlation name identifies a well-known starting point and must be the name of a constant, a declared variable (scalar, row or reference), or one of the predefined start points; for example, InputRoot. The path Fields define a path from the start point to the desired field.

See:

- "Namespace" on page 5051 for the meaning of the different combinations of namespace and name
- "Target field references" on page 5054 for the meaning of the different combinations of field references
- "Index" on page 5051 for the meaning of the different combinations of index clauses
- "Type" on page 5052 for the meaning of the different combinations of types

For example:

```
InputRoot.XMLNS.Data.Invoice
```

starts the broker at the location InputRoot (that is, the root of the input message to a Compute node) and then performs a sequence of navigations. First, it navigates from root to the first child field called XMLNS, then to the first child field of the XMLNS field called Data. Finally, the broker navigates to the first child field of the Data field called Invoice. Whenever this field reference occurs in an ESQL program, the invoice field is accessed.

This form of field reference is simple, convenient, and is the most commonly used. However, it does have two limitations:

- Because the names used must be valid ESQL identifiers, you can use only names that conform to the rules of ESQL. That is, the names can contain only alphanumeric characters including underscore, the first character cannot be numeric, and names must be at least one character long. You can avoid these limitations by enclosing names not conforming to these rules in double quotation marks. For example:

  ```
  InputRoot.XMLNS."Customer Data".Invoice
  ```

  If you need to refer to fields that contain quotation marks, use two pairs of quotation marks around the reference. For example:

  ```
  Body.Message."""hello"""
  ```

  Some identifiers are reserved as keywords but, with the exception of the correlation name, you can use them in field references without the use of double quotation marks
- Because the names of the fields appear in the ESQL program, they must be known when the program is written. This limitation can be avoided by using the alternative syntax that uses braces ({ ... }). This syntax allows you to use any expression that returns a non-null value of type character.

  For example:

  ```
  InputRoot.XMLNS."Customer Data".{'Customer-' ||
   CurrentCustomer}.Invoice
  ```

  in which the invoices are contained in a folder with a name is formed by concatenating the character literal Customer- with the value in CurrentCustomer (which in this example must be a declared variable of type character).

You can use the asterisk (*) wildcard character in a path element to match any name. For example:

```
InputRoot.XMLNS.*.Invoice.Value
```

matches any path element in which the invoices are contained.

Note that enclosing anything in double quotation marks in ESQL makes it an identifier; enclosing anything in single quotation marks makes it a character literal. You must enclose all character strings in single quotation marks.

Namespace:
Field names can belong to namespaces. Field references provide support for namespaces as follows:

- Each field of each field reference that contains a name clause can also contain a namespace clause defining the namespace to which the specified name belongs.
- Each namespace name can be defined by either a simple identifier or by an expression (enclosed in curly braces). If an identifier is the name of a declared namespace constant, the value of the constant is used. If an expression is used, it must return a non-null value of type character.
- A namespace clause of * explicitly states that namespace information is to be ignored when locating Fields in a tree.
- A namespace clause consisting of only : explicitly targets the notarget namespace. The clause has no identifier, expression or wildcard (*).

For example:

```
DECLARE sp1 NAMESPACE 'http://www.ibm.com/space1';

/* Namespace declaration to associate prefix 'space1' with the namespace */

SET OutputRoot.XMLNS.TestCase.(XML.NamespaceDecl)xmlns:space1 = 'http://www.ibm.com/space1';
SET OutputRoot.XMLNS.TestCase.sp1:data1 = 'Hello!';
```

generates:

```
<TestCase xmlns:space1="http://www.ibm.com/space1">
<space1:data1>Hello!</space1:data1>
</TestCase>
```

Index:
Each field of a field reference can contain an index clause. This clause is denoted by brackets ([...]) and accepts any expression that returns a non-null value of type integer. This clause identifies which of several fields with the same name is to be selected. Fields are numbered, starting at one. If this clause is not present, it is assumed that the first field is required. Therefore, the following two examples have exactly the same meaning:

```
InputRoot.XMLNS.Data[1].Invoice
InputRoot.XMLNS.Data.Invoice[1]
```

This construct is most commonly used with an index variable, so that a loop steps though all such fields in sequence. For example:

```
WHILE count < 32 DO
     SET TOTAL = TOTAL + InputRoot.XMLNS.Data.Invoice[count].Amount;
     SET COUNT = COUNT + 1
END WHILE;
```

Use this kind of construct with care, because it implies that the broker must count the fields from the beginning each time round the loop. If the repeat count is large, performance will be poor. In such cases, a better alternative is to use a field reference variable.

Index expressions can optionally be preceded by a less-than sign (<), indicating that the required field is to be indexed from the last field, not the first. In this case, the index 1 refers to the last field and the index 2 refers to the penultimate field. For completeness, you can use a greater-than sign to indicate counting from the first field. The following example shows ESQL code that handles indexes where there are four fields called Invoice.

```
InputRoot.XMLNS.Data.Invoice       -- Selects the first
InputRoot.XMLNS.Data.Invoice[1]    -- Selects the first
InputRoot.XMLNS.Data.Invoice[>]    -- Selects the first
InputRoot.XMLNS.Data.Invoice[>1]   -- Selects the first
InputRoot.XMLNS.Data.Invoice[>2]   -- Selects the second
InputRoot.XMLNS.Data.Invoice[<]    -- Selects the fourth
InputRoot.XMLNS.Data.Invoice[<1]   -- Selects the fourth
InputRoot.XMLNS.Data.Invoice[<2]   -- Selects the third
InputRoot.XMLNS.Data.Invoice[<3]   -- Selects the second
```

An index clause can also consist of an empty pair of brackets ([]). This selects all fields with matching names. Use this construct with functions and statements that expect lists (for example, the SELECT, CARDINALITY, SINGULAR, and EXISTS functions, or the SET statement) .

Type:
Each field of a field reference can contain a type clause. These are denoted by parentheses (( )), and accept any expression that returns a non-null value of type integer. The presence of a type expression restricts the fields that are selected to those of the matching type. This construct is most commonly used with generic XML, where there are many field types and it is possible for one XML field to contain both attributes and further XML Fields with the same name.

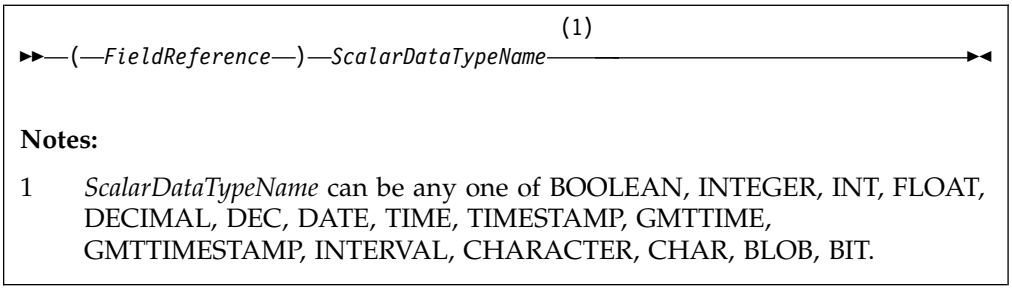
For example:

```
<Item Value = '1234'>
     <Value>5678</Value>
</Item>
```

Here, the XML field Item has two child Fields, both called “Value”. The child Fields can be distinguished by using type clauses:
`Item.(<Domain>.Attribute)Value` to select the attribute, and
`Item.(XML.Element)Value` to select the field, where `<Domain>` is one of XML, XMLNS, or XMLNSC, as determined by the message domain of the source.

Type constraints

A type constraint checks the data type returned by a field reference.



Notes:

1 *ScalarDataTypeName* can be any one of BOOLEAN, INTEGER, INT, FLOAT, DECIMAL, DEC, DATE, TIME, TIMESTAMP, GMTTIME, GMTTIMESTAMP, INTERVAL, CHARACTER, CHAR, BLOB, BIT.

Typically, a type constraint causes the scalar value of the reference to be extracted (in a similar way to the FIELDVALUE function) and an exception to be thrown if the reference is not of the correct type. By definition, an exception will be thrown for all nonexistent fields, because these evaluate to NULL. This provides a convenient and fast way of causing exceptions if essential fields are missing from messages.

However, when type constraints occur in expressions that are candidates for being passed to a database (for example, they are in a WHERE clause), the information is used to determine whether the expression can be given to the database. This can be important if a WHERE clause contains a CAST operating on a database table column. In the absence of a type constraint, such expressions cannot be given to the database because the broker cannot tell whether the database is capable of performing the required conversion. Note, however, that you should always exercise caution when using casts operating on column values, because some databases have exceedingly limited data conversion capabilities.

Field references summary:

***, *[..], (..)*, (..)*[..]**
None of these forms specifies a name or namespace. The target field can have any name, in any namespace or in no namespace. It is located solely by its type, its index, or its type and index, as appropriate.

Name, Name[..], (..)Name, (..)Name[..]
All these forms specify a name but no namespace. The target field is located by namespace and name, and also by type and index where appropriate.

The namespace is taken to be the only namespace in the namespace path containing this name. The only namespace that can be in the path is the notarget namespace.

These forms all existed before namespaces were introduced. Although their behavior has changed in that they now compare both name and namespace, existing transforms should see no change in their behavior because all existing transforms create their Fields in the notarget namespace.

:*, :*[..], (..):*, (..):*[..]
All these forms specify the notarget namespace but no name. The target field is located by its namespace and also by type and index where appropriate.

:Name, :Name[..], (..):Name, (..):Name[..]
All these forms specify a name and the notarget namespace. The target field is located by namespace and name and also by type and index where appropriate.

***:*, *:*[..], (..)*:*, (..)*:*[..]**
None of these forms specifies a name or a namespace. Note that "*:*" is equivalent to "*", and matches no namespace as well as any namespace. The target field can have any name, in any namespace or in no namespace. It is located solely by its type, its index, or its type and index, as appropriate.

***:Name, *:Name[..], (..)*:Name, (..)*:Name[..]**
All these forms specify a name but no namespace. The target field is located by name and also by type and index where appropriate.

Namespace:*, Namespace:*[..], (..)Namespace:*, (..)Namespace:*[..]
All these forms specify a namespace but no name. The target field is located by namespace and also by type and index where appropriate.

Namespace:Name, Namespace:Name[..], (..)Namespace:Name, (..)Namespace:Name[..]
All these forms specify a namespace and name. The target field is located by namespace and name and also by type and index where appropriate.

In all the preceding cases a name, or namespace, provided by an expression contained in braces ({}) is equivalent to a name provided as an identifier.

By definition, the name of the notarget namespace is the empty string. The empty string can be selected by expressions which evaluate to the empty string, the empty identifier "", or by reference to a namespace constant defined as the empty string.

Target field references:
The use of field references usually implies searching for an existing field. However, if the required field does not exist, as is usually the case for field references that are the targets of SET statements and those in the AS clauses of SELECT functions, it is created.

In these situations, there are a variety of circumstances in which the broker cannot tell what the required name or namespace is, and in these situations the following general principles apply :

- If the name clause is absent or does not specify a name, and the namespace clause is absent or does not specify or imply a namespace (that is, there is no name or namespace available), one of the following conditions applies:
  - If the assignment algorithm does **not** copy the name from some existing field, the new field has both its name and namespace set to the empty string and its name flag is **not** set automatically.

    In the absence of a type specification, the field's type is not *Name* or *NameValue*, which effectively indicates that the new field is nameless.
  - Otherwise, if the assignment algorithm copies the name from some existing field, the new field has both its name and namespace copied from the existing field and its *Name* flag is set automatically
- If the name clause is present and specifies a name, but the namespace clause is absent or does not specify or imply a namespace (that is, a name is available but a namespace is not), the new field has its:
  - *Name* set to the given value
  - *Namespace* set to the empty string
  - *Name* flag set automatically

- If the name clause is absent or does not specify a name, but the namespace clause is present and specifies or implies a namespace (that is, a namespace is available but a name is not), the new field has its:
  - *Namespace* set to the given value
  - *Name* set to the empty string
  - *Name* flag set automatically
- If the name clause is present and specifies a name, and the namespace clause is present and specifies or implies a namespace, the new field has its:
  - *Name* set to the given value
  - *Namespace* set to the given value
  - *Name* flag set automatically

There are also cases where the broker creates Fields in addition to those referenced by field references:

- Tree copy: new Fields are created by an algorithm that uses a source tree as a template. If the algorithm copies the name of a source field to a new field, its namespace is copied as well.
- Anonymous select expressions: SELECT clauses are not obliged to have AS clauses; those that do not have them, set the names of the newly created Fields to default values (see "SELECT function" on page 5260).

  These defaults can be derived from field names, column names or can simply be manufactured sequence names. If the name is an field name, this is effectively a tree copy, and the namespace name is copied as above.

  Otherwise, the namespace of the newly-created field is derived by searching the path, that is, the name is be treated as the *NameId* syntax of a field reference.

The effect of setting a field to NULL:

Take care when assigning a null value to a field. For example, the following command *deletes* the `Name` field:

```
SET OutputRoot.XMLNS.Msg.Data.Name = NULL;  -- this deletes the field
```

The correct way to assign a null value to a field is as follows:

```
SET OutputRoot.XMLNS.Msg.Data.Name VALUE = NULL;
-- this assigns a NULL value to a field without deleting it
```

Note: to users on compatibility with earlier versions

For compatibility with earlier versions, the LAST keyword is still supported, but its use is deprecated. LAST cannot be used as part of an index expression: `[LAST]` is valid, and is equivalent to `[<]`, but `[LAST3]` is not valid.

The LAST keyword has been replaced by the following arrow syntax, which allows both a direction of search and index to be specified:

```
Field [> ]                  -- The first field, equivalent to [ 1 ]
Field [> (a + b) * 2 ]
Field [ < ]                  -- The last field, equivalent to [ LAST ]
Field [ < 1 ]                -- The last field, equivalent to [ LAST ]
Field [ < 2 ]                -- The last but one field
Field [ < (a + b) / 3 ]
```

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"ESQL field references" on page 2381
An ESQL field reference is a sequence of period-separated values that identify a specific field (which might be a structure) within a message tree or a database table.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Accessing known multiple occurrences of an element" on page 2425
When you refer to or create the content of messages, it is very likely that the data contains repeating fields. If you know how many instances there are of a repeating field, and you want to access a specific instance of such a field, you can use an array index as part of a field reference.

"Manipulating message body content" on page 2418
The message body is always the last child of root, and is identified by its parser name, for example XML or MRM. You can refer to, modify, and create message body data.

ESQL operators:

A list of the various groups of operators that ESQL supports.

This section provides reference information for the following groups of operators, and for the rules for precedence:

- Simple comparison operators
- Complex comparison operators
- Logical operators
- Numeric operators
- String operator
- Rules for operator precedence

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL operators" on page 2382
An ESQL operator is a character or symbol that you can use in expressions to specify relationships between fields or values.

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your

message flows.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

ESQL simple comparison operators:

The simple comparison operators >, <, >=, <=, =, and <>.

This topic describes ESQL's simple comparison operators. For information about ESQL's complex comparison operators, see "ESQL complex comparison operators" on page 5058.

ESQL provides a full set of comparison operators (predicates). Each compares two scalar values and returns a Boolean. If either operand is null the result is null. Otherwise the result is true if the condition is satisfied and false if it is not.

Comparison operators can be applied to all scalar data types. However, if the two operands are of different types, special rules apply. These are described in "Implicit casts" on page 5282.

Some comparison operators also support the comparison of rows and lists, as follows:

Operator>
The first operand is greater than the second.

Operator <
The first operand is less than the second.

Operator>=
The first operand is greater than or equal to the second.

Operator <=
The first operand is less than or equal to the second.

Operator =
The first operand is equal to that of the second.

This operator can also compare rows and lists. See "ROW and LIST comparisons" on page 5271 for a description of list and row comparison.

Operator <>
The first operand is not equal to the second.

This operator can also compare rows and lists. See "ROW and LIST comparisons" on page 5271 for a description of list and row comparison.

The meanings of "equal", "less", and "greater" in this context are as follows:

- For the numeric types (INTEGER, FLOAT, DECIMAL) the numeric values are compared. Thus 4.2 is greater than 2.4 and -2.4 is greater than -4.2.
- For the date/time types (DATE, TIME, TIMESTAMP, GMTTIME, GMTTIMESTAMP but not INTERVAL) a later point in time is regarded as being greater than an earlier point in time. Thus the date 2004-03-31 is greater than the date 1947-10-24.
- For the INTERVAL type, a larger interval of time is regarded as being greater than a smaller interval of time.

For the string types (CHARACTER, BLOB, BIT) the comparison is lexicographic. Starting from the left, the individual elements (each character, byte or bit) are compared. If no difference is found, the strings are equal. If a difference is found, the values are greater if the first different element in the first operand is greater than the corresponding element in the second and less if they are less. In the special case where two strings are of unequal length but equal as far as they go, the longer string is regarded as being greater than the shorter. Thus:

```
'ABD' is greater than 'ABC'
'ABC' is greater than 'AB'
```

Trailing blanks are regarded as insignificant in character comparisons. Thus if you want to ensure that two strings are truly equal you need to compare both the strings themselves and their lengths. For example:

```
'ABC  ' is equal to 'ABC'
```

Note that comparing strings with a length of one is equivalent to comparing individual characters, bytes, or bits. Because ESQL has no single character, byte, or bit data types, it is standard practice to use strings of length one to compare single characters, bytes, or bits.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"ESQL complex comparison operators"
ESQL supports several operators for complex comparison tasks.

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"ROW and LIST comparisons" on page 5271
You can compare ROWs and LISTs against other ROWs and LISTs.

"Implicit casts" on page 5282

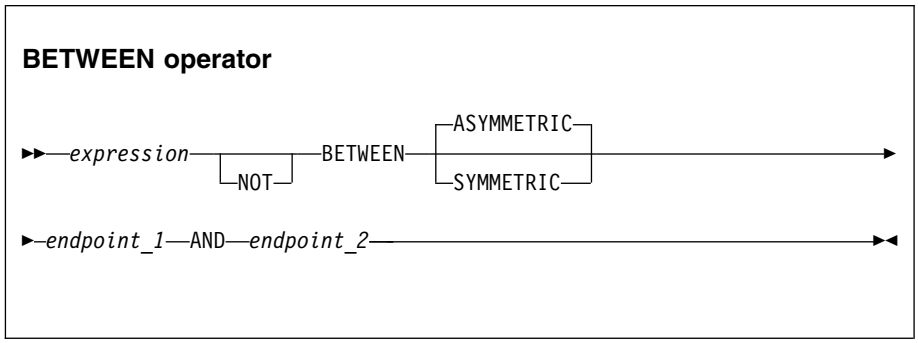
ESQL complex comparison operators:

ESQL supports several operators for complex comparison tasks.

If you want to use ESQL to perform a simple comparison, see "ESQL simple comparison operators" on page 5057.

Operator BETWEEN

Use the BETWEEN operator to test whether a value lies between two boundary values.



This operator exists in two forms, SYMMETRIC and ASYMMETRIC (which is the default if neither is specified). The SYMMETRIC form is equivalent to:

```
(source>= boundary1 AND source <= boundary2) OR
(source>= boundary2 AND source <= boundary1)
```

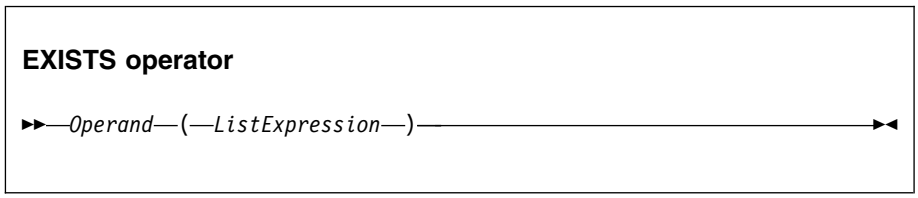
The ASYMMETRIC form is equivalent to:

```
source>= boundary1 AND source <= boundary2
```

The ASYMMETRIC form is simpler but returns only the result that you expect when the first boundary value has a smaller value than the second boundary. It is useful only when the boundary condition expressions are literals.

If the operands are of different types, special rules apply. These rules are described in "Implicit casts" on page 5282.

Operator EXISTS

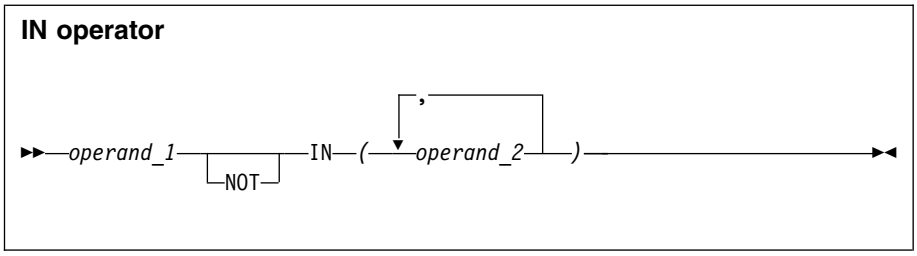


The operator EXISTS returns a Boolean value that indicates whether a SELECT function returned one or more values (TRUE) or none (FALSE).

```
EXISTS(SELECT * FROM something WHERE predicate)
```

Operator IN

Use the operator IN to test whether a value is equal to one of a list of values.



The result is TRUE if the left operand is not NULL and is equal to one of the right operands. The result is FALSE if the left operand is not NULL, and is not equal to one or more of the right operands, none of which have NULL values. Otherwise the result is UNKNOWN. If the operands are of different types, special rules apply. These rules are described in "Implicit casts" on page 5282.

operand_1 must evaluate to a scalar value. *operand_2* can be a sequence of expressions that return scalars of types comparable with *operand_1* or it can be a single expression that returns a LIST. If the LIST is one that is returned from a SELECT function, there can be only a single column in the SelectClause and each ROW in that column is compared with *operand_1* for equality.

The following examples show a sequence of expressions.

- Result1 is set to TRUE because 30 is not present in the given list:

```
SET OutputRoot.XMLNSC.Top.Result1 = 30 NOT IN(34, 42, 45)
```

- Result2 is set to TRUE if the value of var1 is found in var2, var3 or var4:

```
SET OutputRoot.XMLNSC.Top.Result2 = var1 IN(var2, var3, var4);
```

The following examples show a list containing a single column.

- Result3 is set to TRUE if 42 is found in the result set returned from the SELECT statement:

```
SET OutputRoot.XMLNSC.Top.Result3 = 42 IN(
    SELECT A FROM InputRoot.XMLNSC.Top.a[] AS A);
```

- Result4 is set to TRUE because 42 is present in the given list:

```
SET OutputRoot.XMLNSC.Top.Result4 = 42 IN(
    LIST{34,36,37,38,39,40,41,42,43,44});
```

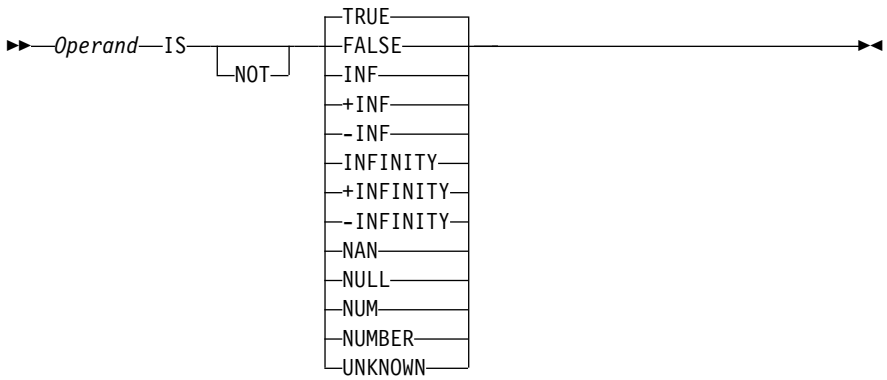
- Result5 is set to TRUE if var1 is found in one of the repeating 'test' elements or its immediate children, if present:

```
SET OutputRoot.XMLNSC.Top.Result5 = var1 IN(
    InputRoot.XMLNSC.Top.test[]);
```

Operator IS

Use the operator IS to test whether an expression has returned a special value.

IS operator



The primary purpose of the IS operator is to test whether a value is NULL. You cannot use the comparison operator (=) to test for a NULL value, because the result of comparing any value with NULL is NULL.

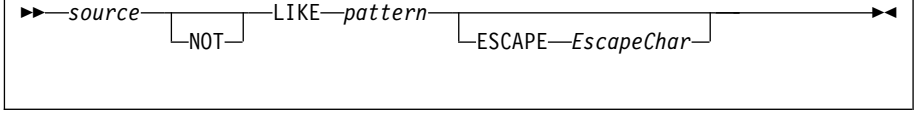
You can also use the IS operand to test for the Boolean values TRUE and FALSE, and to test decimal values for special values. These values are denoted by INF, +INF, -INF, NAN (not a number), and NUM (a valid number) in upper, lower, or mixed case. The alternative forms +INFINITY, -INFINITY, and NUMBER are also accepted.

If applied to non-numeric types, the result is FALSE.

Operator LIKE

Use the LIKE operator to search for strings that match a certain pattern.

LIKE operator



The result is TRUE if none of the operands are NULL and the *source* operand matches the pattern operand. The result is FALSE if none of the operands are NULL and the *source* operand does not match the pattern operand. Otherwise the result is UNKNOWN.

The pattern is specified by a string in which the percent (%) and underscore (_) characters have a special meaning:

- The underscore character matches a single character.

  For example, the following finds matches for IBM and for IGI, but not for International Business Machines or IBM Corp:

  ```
  Body.Trade.Company LIKE 'I__'
  ```

- The percent character % matches a string of zero or more characters.

  For example, the following phrase finds matches for IBM, IGI, International Business Machines, and IBM Corp:

  ```
  Body.Trade.Company LIKE 'I%'
  ```

To use the percent and underscore characters within the expressions that are to be matched, precede the characters with an ESCAPE character, which defaults to the backslash (\) character.

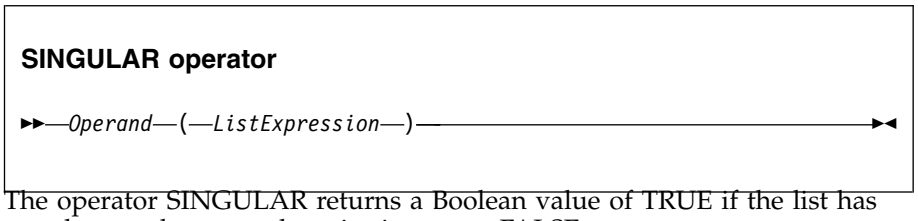
For example, the following predicate finds a match for IBM_Corp.

```
Body.Trade.Company LIKE 'IBM\_Corp'
```

You can specify a different escape character by using the ESCAPE clause. For example, you could also specify the previous example in this way:

```
Body.Trade.Company LIKE 'IBM$_Corp' ESCAPE '$'
```

Operator SINGULAR



The operator SINGULAR returns a Boolean value of TRUE if the list has exactly one element, otherwise it returns FALSE.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"ESQL simple comparison operators" on page 5057
The simple comparison operators >, <, >=, <=, =, and <>.

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"ROW and LIST comparisons" on page 5271
You can compare ROWs and LISTs against other ROWs and LISTs.

"Implicit casts" on page 5282

ESQL logical operators:

The logical operators AND, OR and NOT.

ESQL provides the following logical operators:

Operator AND
The result is the logical AND of the two operands. Both operands must be Boolean values.

Operator OR
The result is the logical OR of the two operands. Both operands must be Boolean values.

Operator NOT
The result is the logical NOT of the operand, which must be a Boolean value.

NULL and UNKNOWN values are treated as special values by these operators, according to the following rules:

- NULL and UNKNOWN are treated the same.
- If an operand is NULL, the result is NULL unless the operation result is already dictated by the other parameter.

The evaluation of the individual clauses in a statement that includes the AND or OR logical operators is stopped as soon as the overall statement can be resolved. For example, see the following statements:

- `IF A OR B THEN ...`
  - If A is false, B is evaluated.
  - If A is true, B is not evaluated because the statement is already resolved to be true.
- `IF A AND B THEN ...`
  - If A is true, B is evaluated.
  - If A is false, B is not evaluated because the statement is already resolved to be false.

The result of AND and OR operations is defined by the following table:

Value of P	Value of Q	Result of P AND Q	Result of P OR Q
TRUE	TRUE	TRUE	TRUE
TRUE	FALSE	FALSE	TRUE
TRUE	UNKNOWN	UNKNOWN	TRUE
FALSE	TRUE	FALSE	TRUE
FALSE	FALSE	FALSE	FALSE
FALSE	UNKNOWN	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	TRUE
UNKNOWN	FALSE	FALSE	UNKNOWN
UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN

The result of NOT operations is defined by the following table.

Operand	Result of NOT
TRUE	FALSE
FALSE	TRUE
UNKNOWN	UNKNOWN

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL operators" on page 2382
An ESQL operator is a character or symbol that you can use in expressions to specify relationships between fields or values.

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"ESQL simple comparison operators" on page 5057
The simple comparison operators >, <, >=, <=, =, and <>.

"ESQL numeric operators"
The numeric operators +, −, *, /, and ||.

"Rules for ESQL operator precedence" on page 5066
How ESQL calculates expressions involving more than one operator.

ESQL numeric operators:

The numeric operators +, −, *, /, and ||.

ESQL provides the following numeric operators:

Unary Operator -
: The result is the negation of the operand (that is, it has the same magnitude as the operand but the opposite sign). You can negate numeric values (INTEGER, DECIMAL and FLOAT) and intervals (INTERVAL).

Operator +
: The result is the sum of the two operands. You can add two numeric values, two intervals, and an interval to a datetime value (DATE, TIME, TIMESTAMP, GMTTIME, and GMTTIMESTAMP).

Operator -
: The result is the difference between the two operands. It is possible to:
  - Subtract one numeric value from another.
  - Subtract one date-time from another. The result is an interval.
  - Subtract one interval from another. The result is an interval.
  - Subtract an interval from a datetime value. The result is a date-time.

When subtracting one date-time from another, you must indicate the type of interval required. You do this by using a qualifier consisting of parentheses enclosing the expression, followed by an interval qualifier. For example:

```
SET OutputRoot.XMLNS.Data.Age  =
          (DATE '2005-03-31' - DATE '1947-10-24') YEAR TO MONTH;
```

Operator *

The result is the product of the two operands. You can multiply numeric values and multiply an interval by a numeric value.

Operator /

The result is the dividend of the two operands. You can divide numeric values and divide an interval by a numeric value.

Operator ||

The result is the concatenation of the two operands. You can concatenate string values (CHARACTER, BIT, and BLOB).

In all cases, if either operand is NULL, the result is NULL. If the operands are of different types, special rules apply. These are described in “Implicit casts” on page 5282.

For examples of how you can use these operators to manipulate datetime values, see “Using numeric operators with datetime values” on page 2439.

Related concepts:

“Message flows overview” on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

“ESQL operators” on page 2382
An ESQL operator is a character or symbol that you can use in expressions to specify relationships between fields or values.

“Message mapping overview” on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

Related tasks:

Chapter 9, “Developing message flow applications,” on page 1019
Develop message flows to process your business messages and data.

“Using numeric operators with datetime values” on page 2439
The following examples show the ESQL that you can code to manipulate datetime values with numeric operators.

“Developing ESQL” on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

“Built-in nodes” on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

“ESQL reference” on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

“ESQL simple comparison operators” on page 5057
The simple comparison operators >, <, >=, <=, =, and <>.

"ESQL logical operators" on page 5062
The logical operators AND, OR and NOT.

"Rules for ESQL operator precedence"
How ESQL calculates expressions involving more than one operator.

ESQL string operator:

A single string operator, concatenation, is supported in ESQL.

Operator ||
The result is the concatenation of the two operands. You can concatenate string values (CHARACTER, BIT, and BLOB).

If either operand is NULL, the result is NULL.

Related concepts:

"ESQL operators" on page 2382
An ESQL operator is a character or symbol that you can use in expressions to specify relationships between fields or values.

Related reference:

"ESQL string manipulation functions" on page 5205
A list of the ESQL string manipulation functions that you can use.

"ESQL simple comparison operators" on page 5057
The simple comparison operators >, <, >=, <=, =, and <>.

"ESQL logical operators" on page 5062
The logical operators AND, OR and NOT.

"ESQL numeric operators" on page 5064
The numeric operators +, -, *, /, and ||.

"Rules for ESQL operator precedence"
How ESQL calculates expressions involving more than one operator.

Rules for ESQL operator precedence:

How ESQL calculates expressions involving more than one operator.

When an expression involves more than one operator, the order in which the expression is evaluated might affect the result. Consider the following example:

```
SET a = b + c * d;
```

Under ESQL's precedence rules, c is multiplied by d and the result is added to b. This rule states that multiplication takes precedence over addition, so reordering the expression as follows:

```
SET a = c * d + b;
```

makes no difference. ESQL's precedence rules are set out later in this section, but it is generally considered good practice to use parentheses to make the meaning clear. The order of precedence is:

1. Parentheses
2. Unary operators including unary - and NOT
3. Multiplication and division
4. Concatenation
5. Addition and subtraction

Operations at the same level are evaluated from left to right.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message mapping overview" on page 2229
Message mappings define the blueprint for creating a message, where the created message is known as the target message.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

ESQL statements:

You can use ESQL statements to manipulate message trees, update databases, or interact with nodes.

The following table summarizes the ESQL statements and what they do.

Statement type	Description
Basic statements:	
"BEGIN ... END statement" on page 5070	Gives the statements defined within the BEGIN and END keywords the status of a single statement.
"CALL statement" on page 5077	Invokes a user-written routine that has been defined using a CREATE FUNCTION or CREATE PROCEDURE statement.
"CASE statement" on page 5081	Uses rules defined in WHEN clauses to select a block of statements to execute.
"CREATE FUNCTION statement" on page 5091	Like CREATE PROCEDURE, CREATE FUNCTION defines a user-written routine. (The few differences between CREATE FUNCTION and CREATE ROUTINE are described in the reference material.)
"CREATE MODULE statement" on page 5101	Creates a module (a named container associated with a node).
"CREATE PROCEDURE statement" on page 5103	Like CREATE FUNCTION, CREATE PROCEDURE defines a user-written routine. (The few differences between CREATE FUNCTION and CREATE ROUTINE are described in the reference material.)

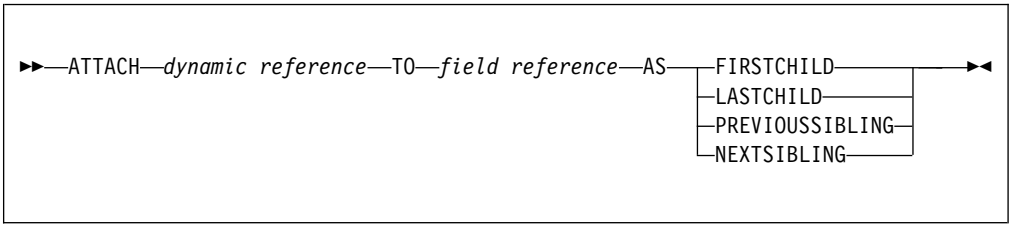
Statement type	Description
“DECLARE statement” on page 5117	Declares one or more variables that can be used to store temporary values.
“IF statement” on page 5134	Processes a set of statements based on the result of evaluating condition expressions.
“ITERATE statement” on page 5139	Abandons processing the current iteration of the containing WHILE, REPEAT, LOOP, or BEGIN statement, and might start the next iteration.
“LEAVE statement” on page 5140	Abandons processing the current iteration of the containing WHILE, REPEAT, LOOP or BEGIN statement, and stops looping.
“LOOP statement” on page 5144	Processes a sequence of statements repeatedly and unconditionally.
“REPEAT statement” on page 5154	Processes a sequence of statements, then evaluates a condition expression. If the expression evaluates to TRUE, executes the statements again.
“RETURN statement” on page 5155	Stops processing the current function or procedure and passes control back to the caller.
“SET statement” on page 5159	Evaluates a source expression, and assigns the result to the target entity.
“THROW statement” on page 5161	Generates a user exception.
“WHILE statement” on page 5167	Evaluates a condition expression, and if it is TRUE executes a sequence of statements.
Message tree manipulation statements:	
“ATTACH statement” on page 5069	Attaches a portion of a message tree into a new position in the message hierarchy.
“CREATE statement” on page 5082	Creates a new message field.
“DELETE statement” on page 5129	Detaches and destroys a portion of a message tree, allowing its memory to be reused.
“DETACH statement” on page 5130	Detaches a portion of a message tree without deleting it.
“FOR statement” on page 5133	Iterates through a list (for example, a message array).
“MOVE statement” on page 5145	Changes the field pointed to by a target reference variable.
Database update statements:	
“DELETE FROM statement” on page 5127	Deletes rows from a table in an external database based on a search condition.
“INSERT statement” on page 5135	Adds a new row to an external database.
“PASSTHRU statement” on page 5147	Takes a character value and passes it as an SQL statement to an external database.
“UPDATE statement” on page 5163	Updates the values of specified rows and columns in a table in an external database.
Node interaction statements:	

Statement type	Description
"PROPAGATE statement" on page 5150	Propagates a message to the downstream nodes within the message flow.
Other statements:	
"BROKER SCHEMA statement" on page 5073	This statement is optional and is used in an ESQL file to explicitly identify the schema that contains the file.
"DECLARE HANDLER statement" on page 5124	Declares an error handler.
"EVAL statement" on page 5131	Takes a character value, interprets it as an SQL statement, and executes it.
"LOG statement" on page 5142	Writes a record to the event or user trace log.
"RESIGNAL statement" on page 5155	Re-throws the current exception (if any). This is used by an error handler, when it cannot handle an exception, to give an error handler in higher scope the opportunity of handling the exception.

ATTACH statement:

The ATTACH statement attaches a portion of a message tree into a new position in the message hierarchy.

Syntax



The following example illustrates how to use the ATTACH statement, together with the DETACH statement described in "DETACH statement" on page 5130, to modify a message structure. The dynamic reference supplied to the DETACH statement must point to a modifiable message tree such as Environment, LocalEnvironment, OutputRoot, OutputExceptionList, or InputLocalEnvironment.

There are some limitations on the use of ATTACH. In general, elements detached from the output trees of a Compute node are not attached to the environment or to input trees.

For example, if you take the following message:

```
<Data>
  <Order>
    <Item>cheese
        <Type>stilton</Type>
    </Item>
    <Item>bread</Item>
  </Order>
  <Order>
```

```
    <Item>garlic</Item>
    <Item>wine</Item>
  </Order>
 </Data>
```

the following ESQL statements:

```
SET OutputRoot = InputRoot;
DECLARE ref1 REFERENCE TO OutputRoot.XMLNSC.Data.Order[1].Item[1];
DETACH ref1;
ATTACH ref1 TO OutputRoot.XMLNSC.Data.Order[2] AS LASTCHILD;
```

result in the following new message structure:

```
<Data>
  <Order>
     <Item>bread</Item>
  </Order>
  <Order>
     <Item>garlic</Item>
     <Item>wine</Item>
     <Item>cheese
        <Type>stilton</Type>
     </Item>
  </Order>
 </Data>
```

For information about dynamic references see “Creating dynamic field references” on page 2431.

Related concepts:

“ESQL overview” on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

“Developing ESQL” on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

“Creating dynamic field references” on page 2431
You can use a variable of type REFERENCE as a dynamic reference to navigate a message tree. This acts in a similar way to a message cursor or a variable pointer.

Related reference:

“Syntax diagrams” on page 3677

“ESQL statements” on page 5067
You can use ESQL statements to manipulate message trees, update databases, or interact with nodes.

“DETACH statement” on page 5130
The DETACH statement detaches a portion of a message tree without deleting it. This portion can be reattached using the ATTACH statement.

BEGIN ... END statement:

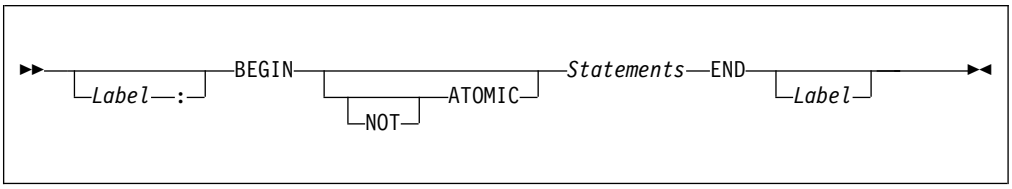
The BEGIN ... END statement gives the statements defined within the BEGIN and END keywords the status of a single statement.

This allows the contained statements to:

- Be the body of a function or a procedure
- Have their exceptions handled by a handler

- Have their execution discontinued by a LEAVE statement

Syntax



The second *Label* can be present only if the first *Label* is present. If both labels are present, they must be identical. Two or more labeled statements at the same level can have the same label, but this partly negates the advantage of the second label. The advantage is that the labels unambiguously and accurately match each END with its BEGIN. However, a labeled statement nested within *Statements* cannot have the same label, because this makes the behavior of the ITERATE and LEAVE statements ambiguous.

Scope of variables

A new local variable scope is opened immediately after the opening BEGIN and, therefore, any variables declared within this statement go out of scope when the terminating END is reached. If a local variable has the same name as an existing variable, any references to that name that occur after the declaration access the local variable. For example:

```
DECLARE Variable1 CHAR 'Existing variable';

-- A reference to Variable1 here returns 'Existing variable'

BEGIN
  -- A reference to Variable1 here returns 'Existing variable'

  DECLARE Variable1 CHAR 'Local variable';  -- Perfectly legal even though
the name is the same

  -- A reference to Variable1 here returns 'Local variable'
END;
```

ATOMIC

If ATOMIC is specified, only one instance of a message flow (that is, one thread) is allowed to execute the statements of a specific BEGIN ATOMIC... END statement (identified by its schema and label), at any one time. If no label is present, the behavior is as if a zero length label had been specified.

The BEGIN ATOMIC construct is useful when a number of changes need to be made to a shared variable and it is important to prevent other instances seeing the intermediate states of the data. Consider the following code example:

```
CREATE PROCEDURE WtiteSharedVariable1(IN NewValue CHARACTER)
SharedVariableMutex1 : BEGIN ATOMIC
  -- Set new value into shared variable
END;

CREATE FUNCTION ReadSharedVariable1() RETURNS CHARACTER
SharedVariableMutex1 : BEGIN ATOMIC
```

```
  DECLARE Value CHARACTER;
  -- Get value from shared variable
  RETURN Value;
END;
```

The last example assumes that the procedure `WriteSharedVariable1` and the function `ReadSharedVariable1` are in the same schema and are used by nodes within the same flow. However, it does not matter whether or not the procedure and function are contained within modules, or whether they are used within the same or different nodes. The broker ensures that, at any particular time, only one thread is executing any of the statements within the atomic sections. This ensures that, for example, two simultaneous writes or a simultaneous read and write are executed serially. Note that:

- The serialization is limited to the flow. Two flows which use BEGIN ATOMIC... END statements with the same schema and label can be executed simultaneously. In this respect, multiple instances within a flow and multiple copies of a flow are not equivalent.
- The serialization is limited by the schema and label. Atomic BEGIN ... END statements specified in different schemas or with different labels do not interact with each other.

Note: You can look at this in a different way, if you prefer. For each combination of message flow, schema, and label, the broker has a mutex that prevents simultaneous access to the statements associated with that mutex.

Do not nest BEGIN ATOMIC... END statements, either directly or indirectly, because this could lead to "deadly embraces". For this reason, do not use a PROPAGATE statement from within an atomic block.

It is not necessary to use the BEGIN ATOMIC construct in flows that will never be deployed with more than one instance (but it might be unwise to leave this to chance). It is also unnecessary to use the BEGIN ATOMIC construct on reads and writes to shared variables. The broker always safely writes a new value to, and safely reads the latest value from, a shared variable. ATOMIC is only required when the application is sensitive to seeing intermediate results.

Consider the following example:

```
DECLARE LastOrderDate SHARED DATE;
...
SET LastOrderDate = CURRENT_DATE;
...
SET OutputRoot.XMLNSC.Data.Orders.Order[1].Date = LastOrderDate;
```

Here we assume that one thread is periodically updating `LastOrderDate` and another is periodically reading it. There is no need to use ATOMIC, because the second SET statement always reads a valid value. If the updating and reading occur very closely in time, whether the old or new value is read is indeterminate, but it is always one or the other. The result will never be garbage.

But now consider the following example:

```
DECLARE Count SHARED INT;
...
SET Count = Count + 1;
```

Here we assume that several threads are periodically executing the SET statement. In this case you do need to use ATOMIC, because two threads might read `Count` in

almost the same instant, and get the same value. Both threads perform the addition and both store the same value back. The end result is thus N+1 and not N+2.

The broker does not automatically provide higher-level locking than this (for example, locking covering the whole SET statement), because such locking is liable to cause “deadly embraces”.

Hint

You can consider the BEGIN ... END statement to be a looping construct, which always loops just once. The effect of an ITERATE or LEAVE statement nested within a BEGIN ... END statement is then as you would expect: control is transferred to the statement following the END. Using ITERATE or LEAVE within a BEGIN ... END statement is useful in cases where there is a long series of computations that needs to be abandoned, either because a definite result has been achieved or because an error has occurred.

Related concepts:

“ESQL overview” on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

“Developing ESQL” on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

“Syntax diagrams” on page 3677

“ESQL statements” on page 5067
You can use ESQL statements to manipulate message trees, update databases, or interact with nodes.

BROKER SCHEMA statement:

The BROKER SCHEMA statement is optional; use it in an ESQL file to explicitly identify the schema that contains the file.

Syntax

An ESQL schema is a named container for functions, procedures, modules, and variables. ESQL schema is similar to the namespace concept of C++, XML, and .NET, and to the package concept of Java.

In the absence of a BROKER SCHEMA statement, all functions, procedures, modules, and constants belong to the default schema. The default schema is similar to the default namespace in C++, the no-target namespace in XML Schema, and the default package in Java.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"PATH clause" on page 5075
The PATH clause specifies a list of additional schemas to be searched when matching function and procedure calls to their implementations. The schema in which the call lies is implicitly included in the PATH clause.

"Syntax diagrams" on page 3677

"ESQL statements" on page 5067
You can use ESQL statements to manipulate message trees, update databases, or interact with nodes.

"CREATE MODULE statement" on page 5101
The CREATE MODULE statement creates a module, which is a named container associated with a node.

PATH clause:

The PATH clause specifies a list of additional schemas to be searched when matching function and procedure calls to their implementations. The schema in which the call lies is implicitly included in the PATH clause.

The PATH clause is used to resolve unqualified function and procedure names in the tools according to the following algorithm.

A single function or procedure must match the unqualified name, or the tools report an error. You can correct the error by qualifying the function or procedure name with a *schemaId*:

1. The current module (if any) is searched for a matching function or procedure. Module-scope functions or procedures are visible only within their containing module. If functions or procedures with the same name are found in the current module and schema, module-scope functions or procedures take precedence over schema scoped functions or procedures.
2. The <node schema> (but none of its contained modules) and the <SQL-broker schema> or schemas identified by the PATH clause are searched for a matching function or procedure.

Note: The *schemaId* must be a fully qualified schema name.

When you start a function or procedure, the name that you use must be qualified by the schema name. The behavior depends on the circumstances:

For a module routine:

- If the schema is specified, the named schema routine is started. The scalar built-in functions, excluding CAST, EXTRACT, and the special registers, are considered to be defined within an implicitly declared schema called SQL.
- If the schema is not specified, and the calling statement is in a module routine, and a routine of the given name exists in the local module, then that local routine is started.
- If the schema is not specified, and the calling statement is in a module routine, and a routine of the given name does not exist in the local module, then all of the schemas in the schema path are searched for a routine of the same name.

  If a matching function exists in one schema, it is used. A compile-time error occurs if a matching function exists in more than one schema. If there is no matching function, then the schema SQL is searched.

  This rule and the preceding rule imply that a local module routine takes priority over a built-in routine of the same name.

For a schema routine:

- If the schema is specified, the named schema routine is started. The scalar built-in functions, excluding CAST, EXTRACT, and the special registers, are considered to be defined within an implicitly declared schema called SQL.

- If the schema is not specified, and the caller is a schema routine, and a routine of the given name exists in the local schema, that local routine is started.
- If the schema is not specified, and the calling statement is in a schema routine, and a routine of the given name does not exist in the local schema, then all of the schemas in the schema path are searched for a routine of the same name.

  If a matching function exists in one schema, it is used. A compile-time error occurs if a matching function exists in more than one schema. If there is no matching function, the schema SQL is searched.

  This rule and the preceding rule imply that a local schema routine takes priority over a built-in routine of the same name.

The <node schema> is defined as the schema containing the node's message flow.

The <node schema> is specified in this manner to provide compatibility with earlier versions of WebSphere Message Broker.

When the <node schema> is the only schema referenced, the broker XML message does not include the extra features contained in WebSphere Message Broker V6.1.

Brokers in previous versions of WebSphere Message Broker do not support multiple schemas, for example, subroutine libraries for reuse. To deploy to a broker in a previous version of the product, put all of the ESQL subroutines into the same schema as the message flow and node that start the ESQL subroutines.

Eclipse tooling uses WebSphere Message Broker V6.1 ESQL syntax in content assist and source code validation.

The broker schema of the message flow must contain, at the schema level, any of the following in its ESQL files:
- A schema level function
- A schema level procedure
- A schema level constant
- A module level constant
- A module level variable

Without the presence of any of the preceding items, the Eclipse tooling generates broker ESQL without module and funtion Main wrappers.

Function and procedure names must be unique within their schema or module.

Examples

The following example adds a path to a schema called CommonUtils:

```
BROKER SCHEMA CommonUtils
PATH SpecialUtils;

MODULE ....
```

The next example adds a path to the default schema:

```
PATH CommonUtils, SpecialUtils;

MODULE ....
```

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"Syntax diagrams" on page 3677

"ESQL statements" on page 5067
You can use ESQL statements to manipulate message trees, update databases, or interact with nodes.

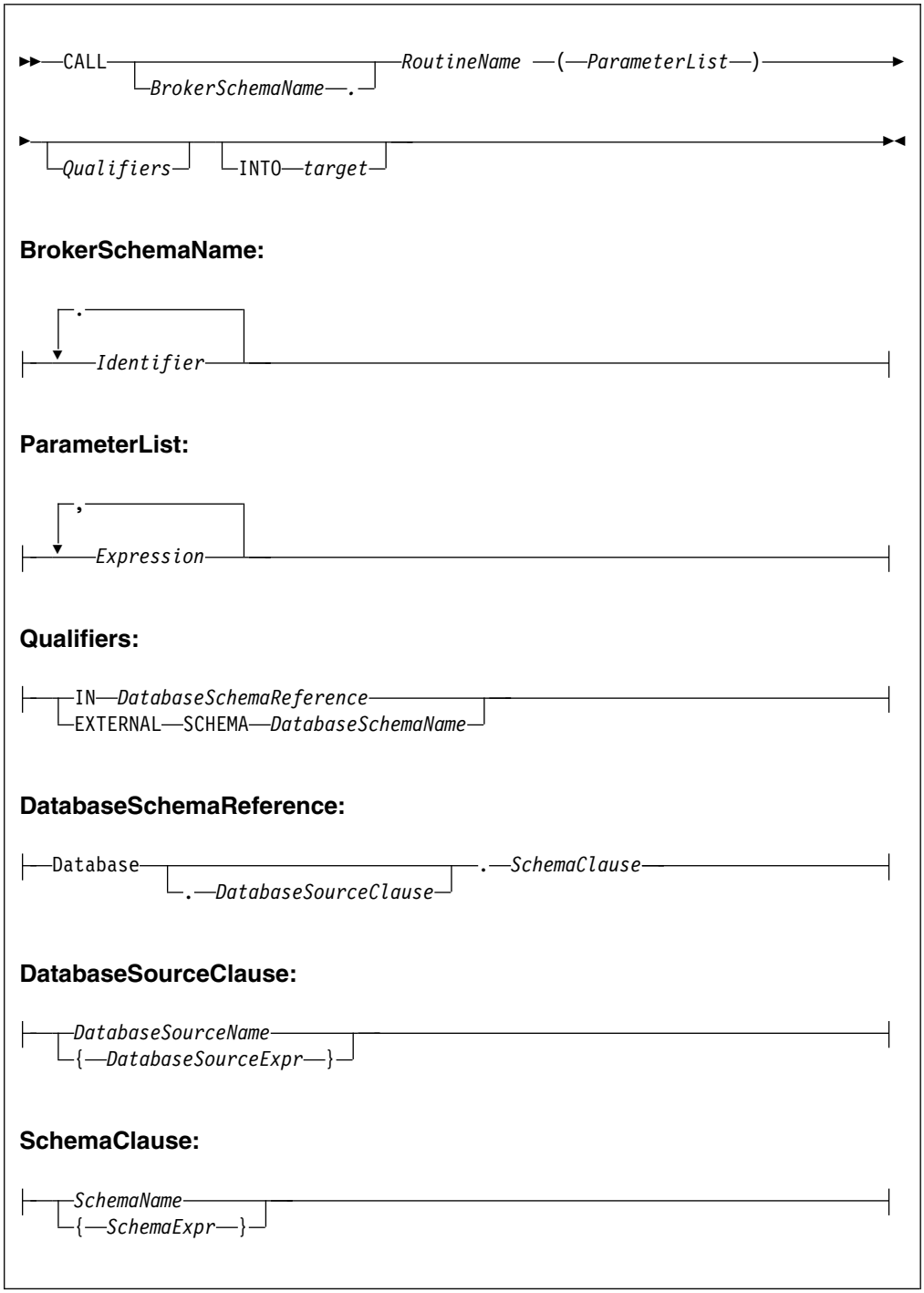
"BROKER SCHEMA statement" on page 5073
The BROKER SCHEMA statement is optional; use it in an ESQL file to explicitly identify the schema that contains the file.

"CREATE MODULE statement" on page 5101
The CREATE MODULE statement creates a module, which is a named container associated with a node.

CALL statement:

The CALL statement calls (invokes) a routine.

Syntax



Using the CALL statement

The CALL statement invokes a routine. A routine is a user-defined function or procedure that has been defined by one of the following:

- A CREATE FUNCTION statement
- A CREATE PROCEDURE statement

As well as standard user-defined functions and procedures, you can also use CALL to invoke built-in (broker-provided) functions and user-defined SQL functions. However, the usual way of invoking these types of function is simply to include their names in expressions.

The called routine must be invoked in a way that matches its definition. For example, if you have defined a routine with three parameters, the first two of type integer and the third of type character, the CALL statement must pass three variables to the routine, each of a data type that matches the definition. This technique is called *exact signature matching*, which means that the signature provided by the CALL statement must match the signature provided by the definition of the routine.

Exact signature matching also applies to the value returned to a routine. If the RETURNS clause is specified on the CREATE FUNCTION statement, or the routine is a built-in function, the INTO clause must be specified on the CALL statement. A return value from a routine cannot be ignored. Conversely, if the RETURNS clause is not specified on the CREATE FUNCTION statement, the INTO clause must not be specified, because there is no return value from the routine.

You can use the CALL statement to invoke a routine that has been implemented in all the following ways:
- ESQL.
- Java.
- As a stored procedure in a database.
- As a built-in (broker-provided) function (although as stated earlier, names are typically included in expressions).

This variety of implementation means that some of the clauses in the CALL syntax diagram are not applicable (or allowed) for all types of routine. It also allows the CALL statement to invoke any type of routine, irrespective of how the routine has been defined.

When the optional *BrokerSchemaName* parameter is not specified, the broker SQL parser searches for the named procedure using the algorithm described in the PATH statement (see the "PATH clause" on page 5075 of the BROKER SCHEMA statement).

When the *BrokerSchemaName* parameter is specified, the broker SQL parser invokes the named procedure in the specified schema without first searching the path. However, if a procedure reference is ambiguous (that is, there are two procedures with the same name in different broker schemas) and the reference is not qualified by the optional *BrokerSchemaName*, the Eclipse toolset generates an error in the Problems view that you must correct to deploy the ambiguous code.

The broker-provided built-in functions are automatically placed in a predefined broker schema called SQL. The SQL schema is always searched last for a routine that has not been matched to a user-defined routine. Therefore, a user-defined module takes precedence over a built-in routine of the same name.

Each broker schema provides a unique symbol or namespace for a routine, so a routine name is unique when it is qualified by the name of the schema to which it belongs.

The INTO clause is used to store the return value from a routine that has been defined with a RETURNS clause, or from a built-in function. The *target* can be an

ESQL variable of a data type that matches the data type on the RETURNS clause, or a dot-separated message reference. For example, both of the following ESQL statements are valid:

```
CALL myProc1() INTO cursor;
CALL myProc1() INTO OutputRoot.XMLNS.TestValue1;
```

The CALL statement passes the parameters into the procedure in the order given to it. Parameters that have been defined as IN or INOUT on the routine's definition are evaluated before the CALL is made, but parameters defined as OUT are always passed in as NULL parameters of the correct type. When the procedure has completed, any parameters declared as OUT or INOUT are updated to reflect any changes made to them during the procedure's execution. Parameters defined as IN are never changed during the cause of a procedure's execution.

Routine overloading is not supported, which means that you cannot create two routines of the same name in the same broker schema. If the broker detects that a routine has been overloaded, it raises an exception. Similarly, you cannot invoke a database stored procedure that has been overloaded. A database stored procedure is overloaded if another procedure of the same name exists in the same database schema. However, you can invoke an overloaded Java method, provided that you create a separate ESQL definition for each overloaded method you want to call, and give each ESQL definition a unique routine name.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Invoking stored procedures" on page 2503
To invoke a procedure that is stored in a database, use the ESQL CALL statement. The stored procedure must be defined by a CREATE PROCEDURE statement that has a Language clause of DATABASE and an EXTERNAL NAME clause that identifies the name of the procedure in the database and, optionally, the database schema to which it belongs.

Related reference:

"Syntax diagrams" on page 3677

"ESQL statements" on page 5067
You can use ESQL statements to manipulate message trees, update databases, or interact with nodes.

"CREATE PROCEDURE statement" on page 5103
The CREATE PROCEDURE statement defines a callable function or procedure.

"BROKER SCHEMA statement" on page 5073
The BROKER SCHEMA statement is optional; use it in an ESQL file to explicitly identify the schema that contains the file.

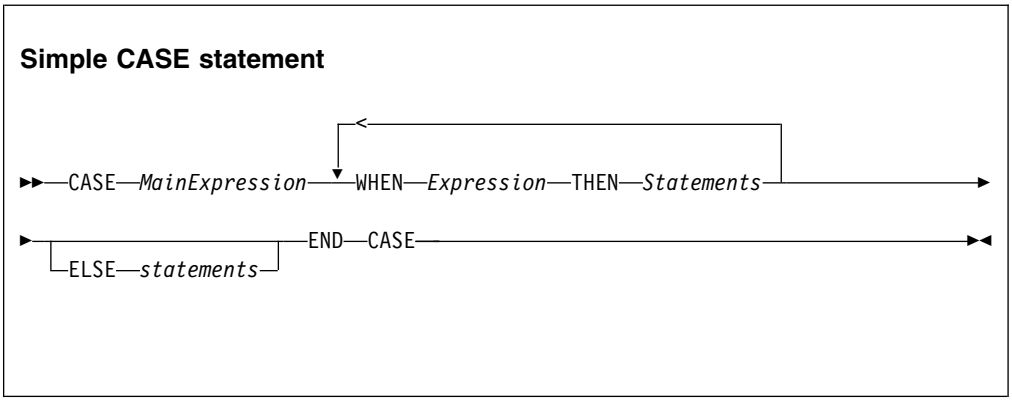
"PATH clause" on page 5075
The PATH clause specifies a list of additional schemas to be searched when matching function and procedure calls to their implementations. The schema in which the call lies is implicitly included in the PATH clause.

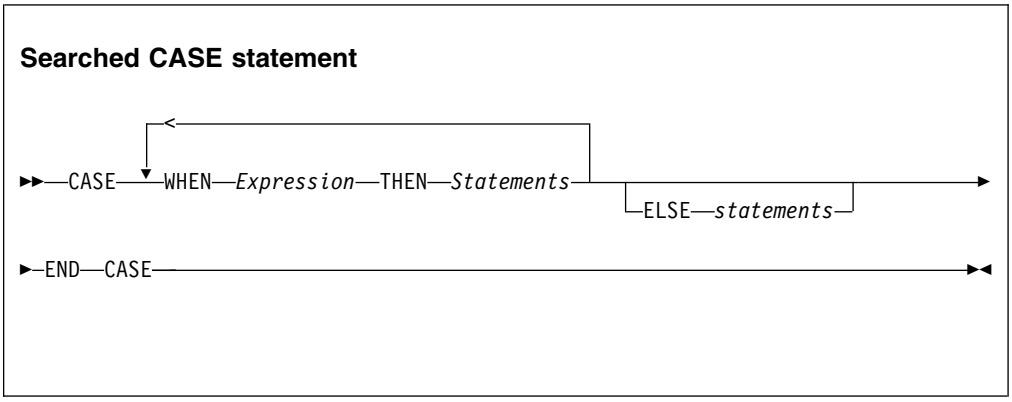
CASE statement:

The CASE statement uses rules defined in WHEN clauses to select a block of statements to process.

There are two forms of the CASE statement: the simple form and the searched form.

Syntax





In the simple form, the main expression is evaluated first. Each WHEN clause expression is evaluated in turn until the result is equal to the main expression's result. That WHEN clause's statements are then processed. If no match is found and the optional ELSE clause is present, the ELSE clause's statements are executed instead. The test values do not have to be literals. The only requirement is that the main expression and the WHEN clause expressions evaluate to types that can be compared.

In the searched form, each WHEN clause expression is evaluated in turn until one evaluates to TRUE. That WHEN clause's statements are then executed. If none of the expressions evaluates to TRUE and the optional ELSE clause is present, the ELSE clause's statements are executed. There does not have to be any similarity between the expressions in each CASE clause. The only requirement is that they all evaluate to a Boolean value.

The ESQL language has both a CASE statement and a CASE function (see "CASE function" on page 5243 for details of the CASE function). The CASE statement

chooses one of a set of statements to execute. The CASE function chooses one of a set of expressions to evaluate and returns as its value the return value of the chosen expression.

Examples

Simple CASE statement:

```
CASE size
  WHEN minimum + 0 THEN
    SET description = 'small';
  WHEN minimum + 1 THEN
    SET description = 'medium';
  WHEN minimum + 2 THEN
    SET description = 'large';
    CALL handleLargeObject();
  ELSE
    SET description = 'unknown';
    CALL handleError();
END CASE;
```

Searched CASE statement:

```
CASE
 WHEN i <> 0 THEN
    CALL handleI(i);
  WHEN j> 1 THEN
    CALL handleIZeroAndPositiveJ(j);
  ELSE
    CALL handleAllOtherCases(j);
END CASE;
```

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"Syntax diagrams" on page 3677

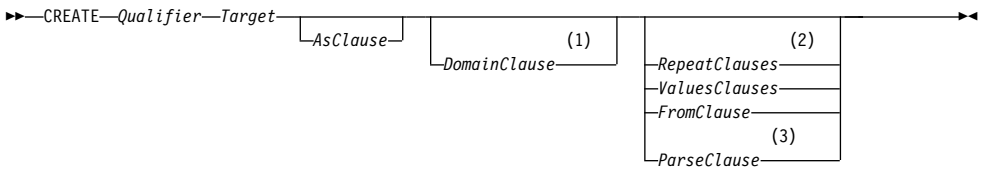
"ESQL statements" on page 5067
You can use ESQL statements to manipulate message trees, update databases, or interact with nodes.

"CASE function" on page 5243
CASE is a complex function that has two forms; the simple-when form and the searched-when form. In either form CASE returns a result, the value of which controls the path of subsequent processing.

CREATE statement:

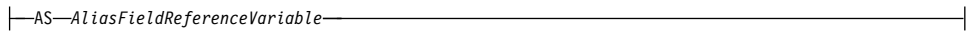
The CREATE statement creates a new message field.

Syntax

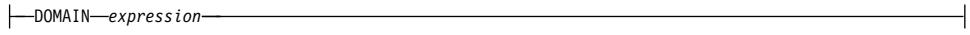


Qualifier:

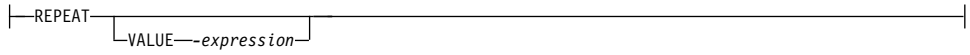
AsClause:



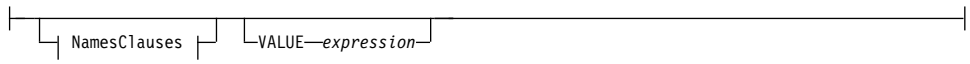
DomainClause:



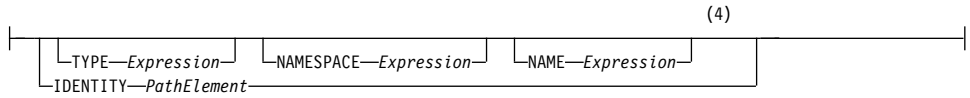
RepeatClauses:



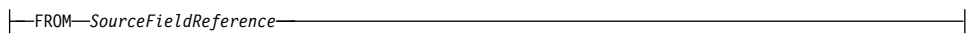
ValuesClauses:



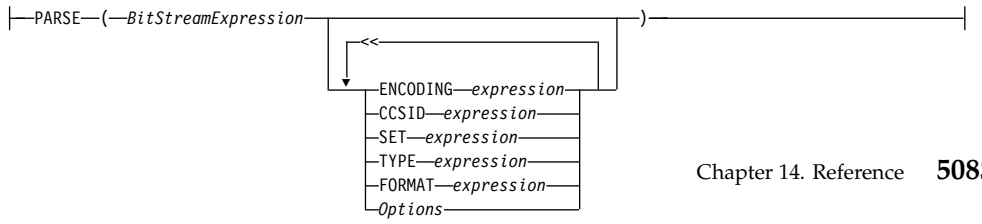
NamesClauses:



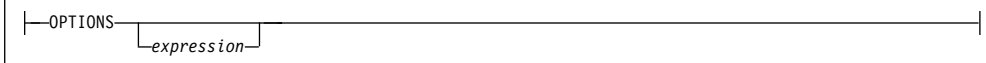
FromClause:



ParseClause:



Options:



Notes:

1 Do not use the *DomainClause* and *ParseClause* with the **FIELD** qualifier.

2 Use the *RepeatClauses* only with the **PREVIOUSSIBLING** and **NEXTSIBLING** qualifiers.

3 Each subclause within the *ParseClause* can occur once only.

4 If present, the TYPE, NAMESPACE, and NAME elements must appear in the order shown in the railroad diagram.

The new message field is positioned either at a given location (CREATE FIELD) or relative to a currently existing location (CREATE ... OF...). New fields can be created only when the target field reference points to a modifiable message; for example, Environment, InputLocalEnvironment, OutputLocalEnvironment, OutputRoot, or OutputExceptionList.

If you include a FIELD clause, the field that is specified by Target is navigated to (creating the fields, if necessary) and any values clause, or from clause, is processed. Including a FIELD clause does not necessarily create any fields; it ensures only that the given fields exist.

If you use array indexes in the target field reference, only one instance of a particular field can be created. Therefore, if you write a SET statement that starts:

```
SET OutputRoot.XMLNS.Message.Structure[2].Field = ...
```

at least one instance of Structure must already exist in the message. That is, the only fields in the tree that are created are ones on a direct path from the root to the field identified by the field reference.

If you include a PREVIOUSSIBLING, NEXTSIBLING, FIRSTCHILD, or LASTCHILD clause, the field that is specified by Target is navigated to (creating the fields if necessary) in exactly the same way as for the FIELD clause. A new field is then created and attached in the specified position (for example, as PREVIOUSSIBLING or FIRSTCHILD). A CREATE statement with one of these clauses always creates a new field, and places it in the specified position.

If you use two CREATE FIRSTCHILD OF target statements that specify the same target, the second statement creates a new field as the first child of the target, and displaces the previously created first child to the right in the message tree (so that it is no longer the first child). Similarly, CREATE LASTCHILD OF target navigates to the target field and adds a new field as its rightmost child, displacing the previous last child to the left.

CREATE PREVIOUSSIBLING OF Target creates a field to the immediate left of the field that is specified by Target (so the depth of the tree is not changed); similarly, CREATE NEXTSIBLING OF Target creates a field to the immediate right of the

field that is specified by Target. When creating PREVIOUSSIBLING or NEXTSIBLING, you can use the REPEAT keyword to copy the type and name of the new field from the current field.

AS clause:

If present, the AS clause moves the named reference variable to point at the newly-created field. Use this clause if you want to involve the new field in some further processing.

DOMAIN clause:

If present, the DOMAIN clause associates the new field with a new parser of the specified type. This clause expects a root field name (for example, XMLNS or MQRFH2). If the DOMAIN clause is present, but the value supplied is a zero-length character string, a new parser of the same type as the parser that owns the field specified by target is created. An exception is thrown if the supplied domain name is not CHARACTER data type or its value is NULL. Do not specify the DOMAIN clause with the FIELD clause; it is not certain that a new field is created.

REPEAT clause:

Use the REPEAT clause to copy the new field's type and name from the target field. Alternatively, the new field's type, name, and value can be:

- Copied from any existing field (using the FROM clause)
- Specified explicitly (using the VALUES clause)
- Defined by parsing a bit stream (using the PARSE clause)

In the case of the FROM and PARSE clauses, you can also create children of the new field.

VALUES clause:

For the VALUES clause, the type, name, and value (or any subset of these) can be specified by any expression that returns a suitable data type (INTEGER for type, CHARACTER for name, and any scalar type for value). An exception is thrown if the value supplied for a type or name is NULL.

NAMES clause:

The NAMES clause takes any expression that returns a non-null value of type character. The meaning depends on the presence of NAME and NAMESPACE clauses as follows:

NAMESPACE	NAME	Element named as follows
No	No	The element is nameless (the name flag is not automatically set).
No	Yes	The element is given the name in the default namespace.
Yes	No	The element is given the empty name in the given namespace.
Yes	Yes	The element is given the given name in the given namespace.

The IDENTITY operand takes a single path element in place of the TYPE and NAME clauses, where a path element contains (at most) a type, a namespace, a name, and an index. These elements specify the type, namespace, name, and index of the element to be created and follow all the rules described in the topic for field references (see "ESQL field reference overview" on page 5049). For example:

```
IDENTITY (XMLNS.attribute)Space1:Name1[42]
```

See the following Examples section for information about how to use the IDENTITY operand.

FROM clause:

For the FROM clause, the new field's type, name, and value are taken from the field pointed to by *SourceFieldReference*. Any existing child fields of the target are detached (the field might already exist in the case of a FIELD clause), and the new field is given copies of the source field's children.

PARSE clause:

If a PARSE clause is present, a subtree is built under the newly-created field from the supplied bit stream. The algorithm for building the subtree varies from parser to parser and according to the options specified. All parsers support the mode RootBitStream, in which the tree creation algorithm is the same as that used by an input node.

Some parsers also support a second mode, FolderBitStream, which generates a sub tree from a bit stream created by the ASBITSTREAM function (see "ASBITSTREAM function" on page 5224) that is using that mode.

When you use the PARSE clause, specify a scalar value containing the bit stream that is to be parsed for *BitStreamExpression*. If you use a message tree field reference you must ensure it contains a scalar value that contains the bit stream. An existing message body folder such as `InputRoot.XMLNSC` does not contain a bit stream and therefore cannot be used this to serialize the `XMLNS` folder. If you pass a value other than a scalar containing the bit stream to the PARSE clause for *BitStreamExpression*, the message flow produces a `BIP2906` error message. Instead, you must first call the ASBITSTREAM function to serialize the existing message tree folder. The result of the ASBITSTREAM function can then be passed as the *BitStreamExpression* to the PARSE clause.

The following example shows how to serialize the XMLNSC folder, then use the result of the ASBITSTREAM in the PARSE clause.

```
DECLARE inCCSID INT InputProperties.CodedCharSetId;
DECLARE inEncoding INT InputProperties.Encoding;
DECLARE inBitStream BLOB ASBITSTREAM(InputRoot.XMLNSC, inEncoding, inCCSID);
CREATE LASTCHILD OF OutputRoot DOMAIN('MRM')
       PARSE(inBitStream, inEncoding, inCCSID, 'DP3UK14002001',
             'TestCase', 'XML1', options);
```

When the PARSE statement is processed, any PARSE clause expressions are evaluated. An exception is thrown if any of the following expressions do not result in a non-null value of the appropriate type:

Clause	Type	Default value
OPTIONS	Integer	RootBitStream & ValidateNone
ENCODING	Integer	0

Clause	Type	Default value
CCSID	Integer	0
SET	Character	Zero length string
TYPE	Character	Zero length string
FORMAT	Character	Zero length string

For details of the syntax of the TYPE clause, see "Specifying namespaces in the Message Type property" on page 1208.

Although the OPTIONS clause accepts any expression that returns a value of type integer, it is only meaningful to generate option values from the list of supplied constants, using the BITOR function if more than one option is required.

Once generated, the value becomes an integer and you can save it in a variable or pass it as a parameter to a function, as well as using it directly with a CREATE statement. The list of globally defined constants is:

```
Validate master options...
ValidateContentAndValue
ValidateValue  -- Can be used with ValidateContent
ValidateContent  -- Can be used with ValidateValue
ValidateNone

Validate failure action options...
ValidateException
ValidateExceptionList
ValidateLocalError
ValidateUserTrace

Validate timing options...
ValidateComplete
ValidateImmediate
ValidateDeferred
```

Notes:

1. For full details of the validation options, refer to "Validation properties" on page 4169.
2. The *Validate timing options* correspond to *Parse Timing options* and, in particular, *ValidateDeferred* corresponds to *Parse Timing On Demand*.

C and Java equivalent APIs

Note that equivalent options are not available on:

- The Java plugin node API MBElement methods **createElementAsLastChildFromBitstream()** and **toBitstream()**
- The C plugin node API methods **cniCreateElementAsLastChildFromBitstream()** and **cniElementAsBitstream**.

You can specify only one option from each group, with the exception of ValidateValue and ValidateContent, which you can use together to obtain the content and value validation. If you do not specify an option within a group, the option in bold is used.

The ENCODING clause accepts any expression that returns a value of type integer. However, it is only meaningful to generate option values from the list of supplied constants:

```
MQENC_INTEGER_NORMAL
MQENC_INTEGER_REVERSED
MQENC_DECIMAL_NORMAL
MQENC_DECIMAL_REVERSED
MQENC_FLOAT_IEEE_NORMAL
MQENC_FLOAT_IEEE_REVERSED
MQENC_FLOAT_S390
```

The values used for the CCSID clause follow the normal numbering system. For example, 1200 = UCS-2, 1208 = UTF-8.

For absent clauses, the given default values are used. Use the CCSID and encoding default values because these take their values from the queue manager's encoding and CCSID settings.

Similarly, using the default values for each of the message set, type, and format options is useful, because many parsers do not require message set, type, or format information, and so any valid value is sufficient.

When any expressions have been evaluated, a bit stream is parsed using the results of the expressions.

Note: Because this function has a large number of clauses, an alternative syntax is supported, in which the parameters are supplied as a comma-separated list rather than by named clauses. In this case the expressions must be in the order:

```
ENCODING -> CCSID -> SET -> TYPE -> FORMAT -> OPTIONS
```

The list can be truncated at any point and an entirely empty expression can be used in any clauses where you do not supply a value.

Examples of how to use the CREATE statement

1. The following example creates the specified field:

   ```
   CREATE FIELD OutputRoot.XMLNS.Data;
   ```

2. The following example creates a field with no name, type, or value as the first child of `ref1`:

   ```
   CREATE FIRSTCHILD OF ref1;
   ```

3. The following example creates a field using the specified type, name, and value:

   ```
   CREATE NEXTSIBLING OF ref1 TYPE NameValue NAME 'Price' VALUE 92.3;
   ```

4. The following example creates a field with a type and name, but no value; the field is added before the sibling indicated by the dynamic reference (`ref1`):

   ```
   CREATE PREVIOUSSIBLING OF ref1 TYPE Name NAME 'Quantity';
   ```

5. The following example creates a field named Component, and moves the reference variable targetCursor to point at it:

   ```
   CREATE FIRSTCHILD OF targetCursor AS targetCursor NAME 'Component';
   ```

6. The following example creates a new field as the right sibling of the field pointed to by the reference variable targetCursor having the same type and name as that field. The statement then moves targetCursor to point at the new field:

   ```
   CREATE NEXTSIBLING OF targetCursor AS targetCursor REPEAT;
   ```

7. The following example shows how to use the PARSE clause:

```
DECLARE bodyBlob BLOB ASBITSTREAM(InputRoot.XMLNS, InputProperties.Encoding,
 InputProperties.CodedCharSetId);
DECLARE creationPtr REFERENCE TO OutputRoot;
CREATE LASTCHILD OF creationPtr DOMAIN('XMLNS') PARSE(bodyBlob,
                    InputProperties.Encoding,
 InputProperties.CodedCharSetId);
```

This example can be extended to show the serializing and parsing of a field or folder:

```
DECLARE bodyBlob BLOB ASBITSTREAM(InputRoot.XMLNS.TestCase.myFolder,
                    InputProperties.Encoding,
InputProperties.CodedCharSetId,",",",",FolderBitStream);
DECLARE creationPtr REFERENCE TO OutputRoot;
CREATE LASTCHILD OF creationPtr DOMAIN('XMLNS') PARSE(bodyBlob,
                    InputProperties.Encoding,
InputProperties.CodedCharSetId,",",",",FolderBitStream);
```

8. The following example shows how to use the IDENTITY operand:

```
CREATE FIELD OutputRoot.XMLNS.TestCase.Root IDENTITY (XML.ParserRoot)Root;
CREATE FIELD OutputRoot.XMLNS.TestCase.Root.Attribute
       IDENTITY (XML.Attribute)NSpace1:Attribute VALUE 'Attrib Value';
CREATE LASTCHILD OF OutputRoot.XMLNS.TestCase.Root
       IDENTITY (XML.Element)NSpace1:Element1[1] VALUE 'Element 1 Value';
CREATE LASTCHILD OF OutputRoot.XMLNS.TestCase.Root
       IDENTITY (XML.Element)NSpace1:Element1[2] VALUE 'Element 2 Value';
```

This produces the following output message:

```
<TestCase>
 <Root xmlns:NS1="NSpace1" NS1:Attribute="Attrib Value">
  <NS1:Element1>Element 1 Value</NS1:Element1>
  <NS1:Element1>Element 2 Value</NS1:Element1>
 </Root>
</TestCase>
```

9. The following example shows how you can use the DOMAIN clause to avoid losing information unique to the XMLNS parser when an unlike parser copy occurs:

```
DECLARE bodyBlob BLOB ASBITSTREAM(InputRoot.XMLNS, InputProperties.Encoding,
InputProperties.CodedCharSetId);
CREATE FIELD Environment.Variables.myXMLTree;
DECLARE creationPtr REFERENCE TO Environment.Variables.myXMLTree;
CREATE FIRSTCHILD OF creationPtr DOMAIN('XMLNS') PARSE(bodyBlob,
                   InputProperties.Encoding,
InputProperties.CodedCharSetId);
```

An example of a CREATE statement

This example provides sample ESQL and an input message, which together produce the output message at the end of the example.

```
CREATE COMPUTE MODULE CreateStatement_Compute
 CREATE FUNCTION Main() RETURNS BOOLEAN
 BEGIN
   CALL CopyMessageHeaders();

       CREATE FIELD OutputRoot.XMLNS.TestCase.description TYPE NameValue VALUE 'This is my TestCase' ;
       DECLARE cursor REFERENCE TO OutputRoot.XMLNS.TestCase;
       CREATE FIRSTCHILD OF cursor Domain('XMLNS')
              NAME 'Identifier' VALUE InputRoot.XMLNS.TestCase.Identifier;
       CREATE LASTCHILD  OF cursor Domain('XMLNS') NAME 'Sport' VALUE InputRoot.XMLNS.TestCase.Sport;
       CREATE LASTCHILD  OF cursor Domain('XMLNS') NAME 'Date' VALUE InputRoot.XMLNS.TestCase.Date;
       CREATE LASTCHILD  OF cursor Domain('XMLNS') NAME 'Type' VALUE InputRoot.XMLNS.TestCase.Type;
       CREATE FIELD         cursor.Division[1].Number TYPE NameValue VALUE 'Premiership';
       CREATE FIELD         cursor.Division[1].Result[1].Number TYPE NameValue VALUE  '1' ;
       CREATE FIELD         cursor.Division[1].Result[1].Home TYPE Name;
       CREATE LASTCHILD OF  cursor.Division[1].Result[1].Home NAME 'Team' VALUE 'Liverpool' ;
       CREATE LASTCHILD OF  cursor.Division[1].Result[1].Home NAME 'Score' VALUE '4';
```

```
        CREATE FIELD         cursor.Division[1].Result[1].Away TYPE Name;
        CREATE LASTCHILD OF  cursor.Division[1].Result[1].Away NAME 'Team' VALUE 'Everton';
        CREATE LASTCHILD OF  cursor.Division[1].Result[1].Away NAME 'Score' VALUE '0';
        CREATE FIELD         cursor.Division[1].Result[2].Number TYPE NameValue VALUE  '2';
        CREATE FIELD         cursor.Division[1].Result[2].Home TYPE Name;
        CREATE LASTCHILD OF  cursor.Division[1].Result[2].Home NAME 'Team' VALUE 'Manchester United';
        CREATE LASTCHILD OF  cursor.Division[1].Result[2].Home NAME 'Score' VALUE '2';
        CREATE FIELD         cursor.Division[1].Result[2].Away TYPE Name;
        CREATE LASTCHILD OF  cursor.Division[1].Result[2].Away NAME 'Team' VALUE 'Arsenal';
        CREATE LASTCHILD OF  cursor.Division[1].Result[2].Away NAME 'Score' VALUE '3';
        CREATE FIELD         cursor.Division[2].Number TYPE NameValue  VALUE '2';
        CREATE FIELD         cursor.Division[2].Result[1].Number TYPE NameValue  VALUE  '1';
        CREATE FIELD         cursor.Division[2].Result[1].Home TYPE Name;
        CREATE LASTCHILD OF  cursor.Division[2].Result[1].Home NAME 'Team' VALUE 'Port Vale';
        CREATE LASTCHILD OF  cursor.Division[2].Result[1].Home NAME 'Score' VALUE '9' ;
        CREATE FIELD         cursor.Division[2].Result[1].Away TYPE Name;
        CREATE LASTCHILD OF  cursor.Division[2].Result[1].Away NAME 'Team' VALUE 'Brentford';
        CREATE LASTCHILD OF  cursor.Division[2].Result[1].Away NAME 'Score' VALUE '5';

 END;

 CREATE PROCEDURE CopyMessageHeaders() BEGIN
  DECLARE I INTEGER 1;
  DECLARE J INTEGER CARDINALITY(InputRoot.*[]);
  WHILE I < J DO
   SET OutputRoot.*[I] = InputRoot.*[I];
   SET I = I + 1;
  END WHILE;
 END;

END MODULE;
```

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Specifying namespaces in the Message Type property" on page 1208
When using the MRM domain, the *Message Type* property is used to specify the name of the message.

"Multipart messages" on page 1191
A *multipart message* contains one or more other messages within its structure. The contained message is sometimes referred to as an *embedded message*.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"XMLNSC: Working with XML messages and bit streams" on page 2541
Use the ASBITSTREAM function and the CREATE statement to manage XML message content.

Related reference:

"Syntax diagrams" on page 3677

"ESQL statements" on page 5067
You can use ESQL statements to manipulate message trees, update databases, or interact with nodes.

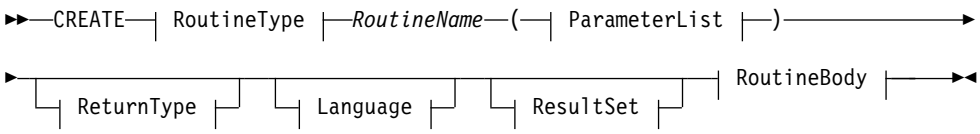
"ASBITSTREAM function" on page 5224
The ASBITSTREAM field function generates a bit stream for the subtree of a given field according to the rules of the parser that owns the field.

CREATE FUNCTION statement:

The CREATE FUNCTION statement defines a callable function or procedure.

You can also use the CREATE PROCEDURE statement to define a callable function or procedure, also known as a routine.

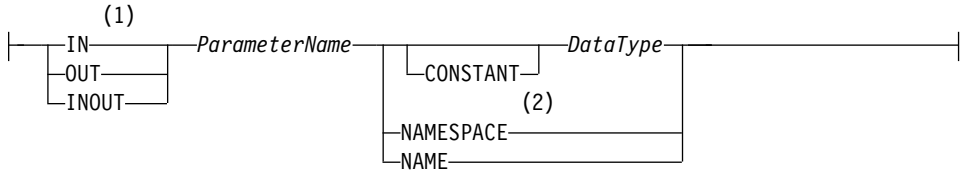
Syntax



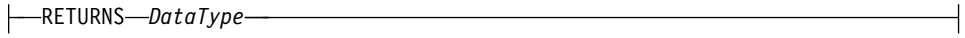
RoutineType:

ParameterList:

Parameter:

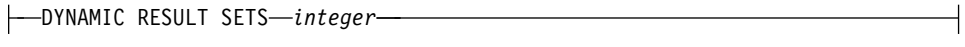


ReturnType:

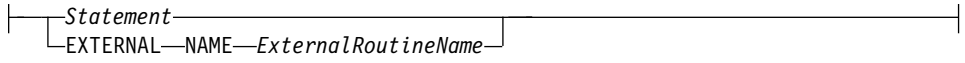


Language:

ResultSet:



RoutineBody:



Notes:

1 If the routine type is FUNCTION, the direction indicator (IN, OUT, or INOUT) is optional for each parameter. However, for documentation purposes, it is good programming practice to specify a direction indicator for all new routines; if you do not specify the direction, a default value of IN is used.

2 When the NAMESPACE or NAME clause is used, its value is implicitly CONSTANT and of type CHARACTER. For information about the use of CONSTANT variables, see the "DECLARE statement" on page 5117.

3 If the routine type is FUNCTION, you cannot specify a LANGUAGE of DATABASE.

Overview

The CREATE FUNCTION and CREATE PROCEDURE statements define a callable function or procedure, also known as a routine.

In previous versions of this product, CREATE FUNCTION and CREATE PROCEDURE had different uses and different capabilities. Subsequent enhancements have resulted in the differences listed previously in notes 1 and 3.

Routines are useful for creating reusable blocks of code that can be run independently many times. You can implement them as a series of ESQL statements, a Java method, or a database stored procedure. This flexibility means that some of the clauses in the syntax diagram are not applicable (or allowed) for all types of routine.

Each routine has a name, which must be unique within the schema to which it belongs. Routine names therefore cannot be overloaded; if the broker detects that a routine name has been overloaded, it raises an exception.

The LANGUAGE clause specifies the language the routines body is written in. The options are:

DATABASE
The procedure is called as a database stored procedure.

ESQL
The procedure is called as an ESQL routine.

JAVA
The procedure is called as a static method in a Java class.

Unspecified
If you do not specify the LANGUAGE clause, the default language is ESQL unless you specify the EXTERNAL NAME clause (in which case, the default language is DATABASE).

Restrictions on the use of the LANGUAGE clause exist. You cannot use:

- The ESQL option with an EXTERNAL NAME clause
- The DATABASE or JAVA options without an EXTERNAL NAME clause
- The DATABASE option with a routine type of FUNCTION

Specify the name of the routine by using the *RoutineName* clause, and its parameters by using the *ParameterList* clause. If the LANGUAGE clause specifies ESQL, implement the routine by using a single ESQL statement. This statement is

most useful if it is a compound statement (BEGIN ... END), because it can then contain as many ESQL statements as necessary to fulfill its function.

Alternatively, instead of providing an ESQL body for the routine, you can specify a LANGUAGE clause other than ESQL. You can then use the EXTERNAL NAME clause to provide a reference to the actual body of the routine, wherever it is located externally to the broker. For more information about using the EXTERNAL NAME clause, see "Invoking stored procedures" on page 2503 and Calling a Java routine.

Routines of any LANGUAGE type can have IN, OUT, and INOUT parameters. The caller can pass several values into the routine, and receive back several updated values. These returned parameters are in addition to any RETURNS clause that you have defined for the routine. The RETURNS clause defines the value that the routine returns to the caller.

Routines that are implemented in different languages have their own restrictions on which data types can be passed in or returned; these restrictions are documented later in this section. The data type of the returned value must match the data type of the value that is defined to be returned from the routine. Also, if a routine is defined to have a return value, the caller of the routine cannot ignore it. For more information, see "CALL statement" on page 5077.

Routines can be defined in either a module or a schema. Routines that are defined in a module are local in scope to the current node, which means that only code belonging to that same module (or node) can invoke them. Routines that are defined in a schema, however, can be invoked by using either of the following options:

- Code in the same schema
- Code in any other schema, if either of the following conditions applies:
  - The other schemas PATH clause contains the path to the called routine
  - The called routine is invoked by using its fully qualified name (which is its name, prefixed by its schema name, separated by a period)

Thus, if you need to invoke the same routine in more than one node, define it in a schema.

For any language or routine type, the method of invocation of the routine must match the manner of declaration of the routine. If the routine has a RETURNS clause, use either the FUNCTION invocation syntax or a CALL statement with an INTO clause. Conversely, if a routine has no RETURNS clause, you must use a CALL statement without an INTO clause.

Parameter directions

Parameters that are passed to routines always have a direction associated with them, which is one of the following types:

IN The value of the parameter cannot be changed by the routine. A NULL value for the parameter is allowed, and can be passed to the routine.

OUT
When it is received by the called routine, the parameter that is passed into the routine always has a NULL value of the correct data type. This value is set irrespective of its value before the routine is called. The routine is allowed to change the value of the parameter.

INOUT
INOUT is both an IN and an OUT parameter. It passes a value into the routine, and the value that is passed in can be changed by the routine. A NULL value for the parameter is allowed, and can be passed both into and out of the routine.

If the routine type is FUNCTION, the direction indicator (IN, OUT, INOUT) is optional for each parameter. However, it is good programming practice to specify a direction indicator for all new routines of any type for documentation purposes.

ESQL variables that are declared to be CONSTANT (or references to variables declared to be CONSTANT) are not allowed to have the direction OUT or INOUT.

ESQL routines

ESQL routines are written in ESQL, and have a LANGUAGE clause of ESQL. The body of an ESQL routine is typically a compound statement of the form BEGIN ... END, that contains multiple statements for processing the parameters that are passed to the routine.

ESQL example 1

The following example shows the same procedure as in "Database routine example 1" on page 5112, but is implemented as an ESQL routine and not as a stored procedure. The CALL syntax and results of this routine are the same as found in:

```
CREATE PROCEDURE swapParms (
  IN parm1 CHARACTER,
  OUT parm2  CHARACTER,
  INOUT parm3 CHARACTER )
BEGIN
   SET parm2 = parm3;
   SET parm3 = parm1;
 END;
```

ESQL example 2

This example procedure shows the recursive use of an ESQL routine. It parses a tree, visiting all places at and below the specified starting point, and reports what it has found:

```
SET OutputRoot.MQMD = InputRoot.MQMD;

  DECLARE answer CHARACTER;
  SET     answer = '';

  CALL navigate(InputRoot.XMLNS, answer);
  SET OutputRoot.XMLNS.Data.FieldNames = answer;


  CREATE PROCEDURE navigate (IN root REFERENCE, INOUT answer CHARACTER)
  BEGIN
    SET answer = answer || 'Reached Field... Type:'
    || CAST(FIELDTYPE(root) AS CHAR)||
    ': Name:' || FIELDNAME(root) || ': Value :' || root || ': ';

    DECLARE cursor REFERENCE TO root;
    MOVE cursor FIRSTCHILD;
    IF LASTMOVE(cursor) THEN
      SET answer = answer || 'Field has children... drilling down ';
    ELSE
      SET answer = answer || 'Listing siblings... ';
```

```
    END IF;

    WHILE LASTMOVE(cursor) DO
      CALL navigate(cursor, answer);
      MOVE cursor NEXTSIBLING;
    END WHILE;

    SET answer = answer || 'Finished siblings... Popping up ';
  END;
```

When given the following input message:

```
<Person>
  <Name>John Smith</Name>
  <Salary period='monthly' taxable='yes'>-1200</Salary>
</Person>
```

the procedure produces the following output, which has been manually formatted:

```
  Reached Field... Type:16777232: Name:XML: Value :: Field has children...
  drilling down
  Reached Field... Type:16777216: Name:Person: Value :: Field has children...
  drilling down
  Reached Field... Type:16777216: Name:Name:
  Value :John Smith: Field has children... drilling down
  Reached Field... Type:33554432: Name::
  Value :John Smith: Listing siblings... Finished siblings... Popping up
  Finished siblings... Popping up
  Reached Field... Type:16777216: Name:Salary:
  Value :-1200: Field has children... drilling down
  Reached Field... Type:50331648: Name:period:
  Value :monthly: Listing siblings... Finished siblings... Popping up
  Reached Field... Type:50331648: Name:taxable:
  Value :yes: Listing siblings... Finished siblings... Popping up
  Reached Field... Type:33554432: Name::
  Value :-1200: Listing siblings... Finished siblings... Popping up
  Finished siblings... Popping up
  Finished siblings... Popping up
  Finished siblings... Popping up
```

Java routines

A Java routine is implemented as a Java method, and has a LANGUAGE clause of JAVA. For Java routines, the *ExternalRoutineName* must contain the class name and method name of the Java method to be called. Specify the *ExternalRoutineName* like the following example:

```
>>--"-- className---.---methodName--"-------------><
```

where *className* identifies the class that contains the method and *methodName* identifies the method to invoke. If the class is part of a package, the class identifier part must include the complete package prefix; for example,“com.ibm.broker.test.MyClass.myMethod”

To find the Java class, the broker uses the search method that is described in “Deploying Java classes” on page 5099.

Any Java method that you want to invoke must have the following basic signature:

```
public static <return-type> <method-name> (< 0 - N parameters>)
```

Where <return-type> must be in the list of Java IN data types in the table in “ESQL to Java data type mapping” on page 5098 (excluding the REFERENCE type, which is not permitted as a return value), or the Java void data type. The

parameter data types must also be in the "ESQL to Java data type mapping" on page 5098 table. In addition, the Java method is not allowed to have `exception throws` clause in its signature.

The Java methods signature must match the ESQL routines declaration of the method. You must also observe the following rules:

- Ensure that the Java method name, including the class name and any package qualifiers, matches the procedures EXTERNAL NAME.
- If the Java return type is void, do not put a RETURNS clause on the ESQL routines definition. Conversely, if the Java return type is not void, you must put a RETURNS clause on the ESQL routines definition.
- Ensure that every parameters type and direction matches the ESQL declaration, according to the rules listed in the table in "ESQL to Java data type mapping" on page 5098.
- Ensure that the methods return type matches the data type of the RETURNS clause.
- Enclose EXTERNAL NAME in quotation marks because it must contain at least "class.method".
- If you want to invoke an overloaded Java method, you must create a separate ESQL definition for each overloaded method and give each ESQL definition a unique routine name.

You can use the Java user-defined node API in your Java method, if you observe the restrictions documented in "Restrictions on Java routines" on page 5099. For more information about using the Java API, see "Compiling a Java user-defined node" on page 3074.

Java routine example 1

This routine contains three parameters of varying directions, and returns an integer, which maps to a Java return type of `java.lang.Long`.

```
CREATE FUNCTION  myProc1( IN P1 INTEGER, OUT P2 INTEGER, INOUT P3 INTEGER )
 RETURNS INTEGER
 LANGUAGE JAVA
 EXTERNAL NAME "com.ibm.broker.test.MyClass.myMethod1";
```

You can use the following ESQL to invoke `myProc1`:

```
CALL myProc1( intVar1, intVar2, intVar3) INTO intReturnVar3;
-- or
SET intReturnVar3 = myProc1( intVar1, intVar2, intVar3);
```

Java routine example 2

This routine contains three parameters of varying directions and has a Java return type of `void`.

```
CREATE PROCEDURE myProc2( IN P1 INTEGER, OUT P2 INTEGER, INOUT P3 INTEGER )
 LANGUAGE JAVA
 EXTERNAL NAME "com.ibm.broker.test.MyClass.myMethod2";
```

You must use the following ESQL to invoke `myProc2`:

```
CALL myProc2(intVar1, intVar2, intVar3);
```

The following Java class provides a method for each of the preceding Java examples:

```
package com.ibm.broker.test;

class MyClass {
public static Long myMethod1( Long P1, Long[] P2 Long[] P3) { ... }
public static void myMethod2( Long P2, Long[] P2 Long[] P3) { ... }

 /* When either of these methods is called:
    P1 might or might not be NULL (depending on the value of intVar1).
    P2[0] is always NULL (whatever the value of intVar2).
    P3[0] might or might not be NULL (depending on the value of intVar3).
    This is the same as with LANGUAGE ESQL routines.
    When these methods return:
         intVar1 is unchanged
         intVar2 might still be NULL or might have been changed
         intVar3 might contain the same value or might have been changed.
     This is the same as with LANGUAGE ESQL routines.

    When myMethod1 returns: intReturnVar3 is either  NULL (if the
    method returns NULL) or it contains the value returned by the
    method.
 */
}
```

ESQL to Java data type mapping

The following table summarizes the mappings from ESQL to Java.

Notes:

- Only the Java scalar wrappers are passed to Java.
- The ESQL scalar types are mapped to Java data types as object wrappers, or object wrapper arrays, depending upon the direction of the procedure parameter. Each wrapper array contains exactly one element.
- Scalar object wrappers are used to allow NULL values to be passed to and from Java methods.

ESQL data types [1]	**Java IN data types**	**Java INOUT and OUT data types**
INTEGER, INT	java.lang.Long	java.lang.Long []
FLOAT	java.lang.Double	java.lang.Double[]
DECIMAL	java.math.BigDecimal	java.math.BigDecimal[]
CHARACTER, CHAR	java.lang.String	java.lang.String[]
BLOB	byte[]	byte[][]
BIT	java.util.BitSet	java.util.BitSet[]
DATE	com.ibm.broker.plugin.MbDate	com.ibm.broker.plugin.MbDate[]
TIME [2]	com.ibm.broker.plugin.MbTime	com.ibm.broker.plugin.MbTime[]
GMTTIME [2]	com.ibm.broker.plugin.MbTime	com.ibm.broker.plugin.MbTime[]
TIMESTAMP [2]	com.ibm.broker.plugin.MbTimestamp	com.ibm.broker.plugin.MbTimestamp[]
GMTTIMESTAMP [2]	com.ibm.broker.plugin.MbTimestamp	com.ibm.broker.plugin.MbTimestamp[]
INTERVAL	Not supported	Not supported
BOOLEAN	java.lang.Boolean	java.lang.Boolean[]
REFERENCE (to a message tree) [3] [4] [5] [6]	com.ibm.broker.plugin.MbElement	com.ibm.broker.plugin.MbElement[] **(Supported for INOUT. Not supported for OUT)**
ROW	Not supported	Not supported

ESQL data types [1]	Java IN data types	Java INOUT and OUT data types
LIST	Not supported	Not supported

1. Variables that are declared to be CONSTANT (or references to variables that are declared to be CONSTANT) are not allowed to have the direction INOUT or OUT.
2. The time zone set in the Java variable is not important; you obtain the required time zone in the output ESQL.
3. The reference parameter cannot be NULL when passed into a Java method.
4. The reference cannot have the direction OUT when passed into a Java method.
5. If an *MbElement* is passed back from Java to ESQL as an INOUT parameter, it must point to a location in the same message tree as that pointed to by the *MbElement* that was passed into the called Java method.

   For example, if an ESQL reference to `OutputRoot.XML.Test` is passed into a Java method as an INOUT *MbElement*, but a different *MbElement* is passed back to ESQL when the call returns, the different element must also point to somewhere in the `OutputRoot` tree.
6. An *MbElement* cannot be returned from a Java method with the RETURNS clause, because no ESQL routine can return a reference. However, an *MbElement* can be returned as an INOUT direction parameter, subject to the conditions described in point 5.

A REFERENCE to a scalar variable can be used in the CALL of a Java method, provided that the data type of the variable to which the reference refers matches the corresponding data type in the Java program signature.

Restrictions on Java routines

The following restrictions apply to Java routines that are called from ESQL:

- The Java method must be threadsafe (reentrant).
- Database connections must be JDBC type 2 or type 4. Furthermore, database operations are not part of a broker transaction and therefore cannot be controlled by an external resource coordinator (as is the case in an XA environment).
- The Java user-defined node API must be used only by the same thread that invoked the Java method.

  You can create threads inside your method. However, created threads must not use the Java APIs, and you must return control back to the broker.

  All restrictions that apply to the usage of the Java API also apply to Java methods that are called from ESQL.
- Java methods that are called from ESQL must not use the `MbNode` class. Therefore, they cannot create objects of type `MbNode`, or call any of the methods on an existing `MbNode` object.
- WebSphere MQ or JMS work done inside a Java method that is called from ESQL must be done in accordance with the guidelines for performing WebSphere MQ and JMS work in a user-defined node. See "Planning user-defined input nodes" on page 2995.

Deploying Java classes

You can deploy your Java classes to a broker within a Java Archive (JAR) file, by using one of the following two methods:

1. **Add the JAR file to the broker archive (BAR) file**

The most efficient and flexible method of deploying to the broker is to add your JAR file to the BAR file. You can do this manually or automatically using the WebSphere Message Broker Toolkit.

If the WebSphere Message Broker Toolkit finds the correct Java class inside a referenced Java project open in the workspace, it automatically compiles the Java class into a JAR file and adds it to the BAR file. This procedure is the same procedure that you follow to deploy a JavaCompute node inside a JAR, as described in "User-defined node class loading" on page 3120.

When you deploy a JAR file from the WebSphere Message Broker Toolkit, the flow that has been redeployed reloads the JAR file contained in the BAR file.

The files are also reloaded if the message flow that references a Java class is stopped and restarted. Ensure that you stop and restart (or redeploy) all flows that reference the JAR file that you want to update. This action avoids the problem of some flows running with the old version of the JAR file and other flows running with the new version.

The WebSphere Message Broker Toolkit deploys only JAR files; it does not deploy stand-alone Java class files.

2. **Store the JAR file in either of the following locations:**
   a. The *workpath*/shared-classes/ folder on the machine running the broker
   b. The CLASSPATH environment variable on the computer running the broker

   You must complete this action manually; you cannot use the WebSphere Message Broker Toolkit.

   In this method, redeploying the message flow does not reload the referenced Java classes; neither does stopping and restarting the message flow. The only way to reload the classes in this case is to stop and restart the broker itself.

To enable the broker to find a Java class, ensure that it is in one of the preceding locations. If the broker cannot find the specified class, it generates an exception.

Although you have the choices shown previously when you deploy the JAR file, by using the WebSphere Message Broker Toolkit to deploy the BAR file provides the greatest flexibility when redeploying the JAR file.

Database routines

CREATE FUNCTION does not support database routines. Use CREATE PROCEDURE to define a database routine.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Invoking stored procedures" on page 2503
To invoke a procedure that is stored in a database, use the ESQL CALL statement. The stored procedure must be defined by a CREATE PROCEDURE statement that has a Language clause of DATABASE and an EXTERNAL NAME clause that identifies the name of the procedure in the database and, optionally, the database schema to which it belongs.

"Compiling a Java user-defined node" on page 3074
When you have created the code for your Java user-defined node, you must compile it for your operating system.

"Adding files to a broker archive" on page 3223
To deploy files to an execution group, include them in a broker archive (BAR) file.

Related reference:

"Syntax diagrams" on page 3677

"ESQL statements" on page 5067
You can use ESQL statements to manipulate message trees, update databases, or interact with nodes.

"CALL statement" on page 5077
The CALL statement calls (invokes) a routine.

"CREATE PROCEDURE statement" on page 5103
The CREATE PROCEDURE statement defines a callable function or procedure.

"ESQL-to-Java data-type mapping table" on page 5043
Table summarizing the mappings from ESQL to Java.

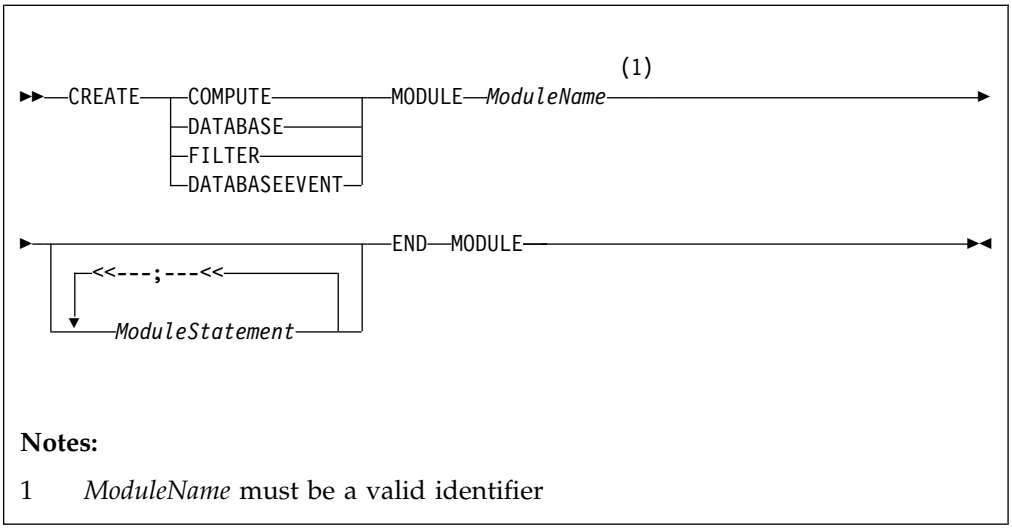
Related information:

Java user-defined extensions API

CREATE MODULE statement:

The CREATE MODULE statement creates a module, which is a named container associated with a node.

Syntax



Notes:

1 *ModuleName* must be a valid identifier

A module in the Eclipse tools is referred to from a message processing node by name. The module must be in the <node schema>.

Module names occupy the same symbol space as functions and procedures defined in the schema. That is, modules, functions, and procedures contained by a schema must all have unique names.

Note: You are warned if there is no module associated with an ESQL node. You cannot deploy a flow containing a node in which a module is missing.

The modules for the Compute node, Database node, and Filter node must all contain exactly one function called Main. This function should return a Boolean. It is the entry point used by a message flow node when processing a message.

The modules for the DatabaseInput node must contain the following entry points:

ReadEvents
This procedure gets details of events to be processed from your event store.

BuildMessage
This procedure builds the message that will be propagated to the flow.

EndEvent
This procedure updates the event table to ensure that this event is not processed again.

Correlation name	Compute module	Filter module	Database module	DatabaseEvent module
Database	x	x	x	x
Environment	x	x	x	x
Root		x	x	x
Body		x	x	
Properties		x	x	x
ExceptionList		x	x	x
LocalEnvironment		x	x	x
InputRoot	x			
InputBody	x			
InputProperties	x			
InputExceptionList	x			
InputLocalEnvironment	x			
OutputRoot	x			
OutputExceptionList	x			
OutputLocalEnvironment	x			
DestinationList	Deprecated synonym for LocalEnvironment			
InputDestinationList	Deprecated synonym for InputLocalEnvironment			
OutputDestinationList	Deprecated synonym for OutputLocalEnvironment			

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"Syntax diagrams" on page 3677

"ESQL statements" on page 5067
You can use ESQL statements to manipulate message trees, update databases, or interact with nodes.

"CALL statement" on page 5077
The CALL statement calls (invokes) a routine.

"BROKER SCHEMA statement" on page 5073
The BROKER SCHEMA statement is optional; use it in an ESQL file to explicitly identify the schema that contains the file.

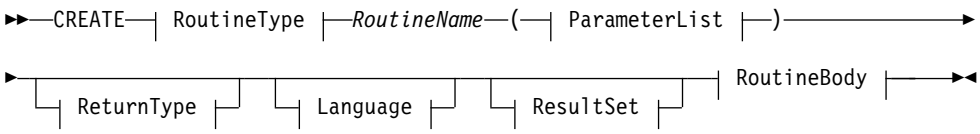
"PATH clause" on page 5075
The PATH clause specifies a list of additional schemas to be searched when matching function and procedure calls to their implementations. The schema in which the call lies is implicitly included in the PATH clause.

CREATE PROCEDURE statement:

The CREATE PROCEDURE statement defines a callable function or procedure.

You can also use the CREATE FUNCTION statement to define a callable function or procedure, also known as a routine.

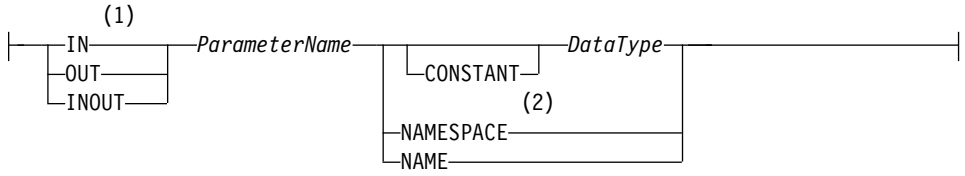
Syntax



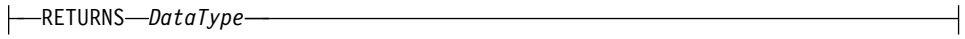
RoutineType:

ParameterList:

Parameter:



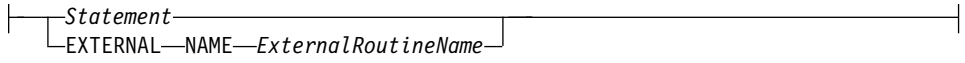
ReturnType:



Language:

ResultSet:

RoutineBody:



Notes:

1 If the routine type is FUNCTION, the direction indicator (IN, OUT, or INOUT) is optional for each parameter. However, for documentation purposes, it is good programming practice to specify a direction indicator for all new routines; if you do not specify the direction, a default value of IN is used.

2 When the NAMESPACE or NAME clause is used, its value is implicitly CONSTANT and of type CHARACTER. For information about the use of CONSTANT variables, see the "DECLARE statement" on page 5117.

3 If the routine type is FUNCTION, you cannot specify a LANGUAGE of DATABASE.

Parameter directions

Parameters that are passed to routines always have a direction associated with them, which is one of the following types:

IN The value of the parameter cannot be changed by the routine. A NULL value for the parameter is allowed, and can be passed to the routine.

OUT
When it is received by the called routine, the parameter that is passed into the routine always has a NULL value of the correct data type. This value is set irrespective of its value before the routine is called. The routine is allowed to change the value of the parameter.

INOUT
INOUT is both an IN and an OUT parameter. It passes a value into the routine, and the value that is passed in can be changed by the routine. A NULL value for the parameter is allowed, and can be passed both into and out of the routine.

If the routine type is FUNCTION, the direction indicator (IN, OUT, INOUT) is optional for each parameter. However, it is good programming practice to specify a direction indicator for all new routines of any type for documentation purposes.

ESQL variables that are declared to be CONSTANT (or references to variables declared to be CONSTANT) are not allowed to have the direction OUT or INOUT.

ESQL routines

ESQL routines are written in ESQL, and have a LANGUAGE clause of ESQL. The body of an ESQL routine is typically a compound statement of the form BEGIN ... END, that contains multiple statements for processing the parameters that are passed to the routine.

ESQL example 1

The following example shows the same procedure as in "Database routine example 1" on page 5112, but is implemented as an ESQL routine and not as a stored procedure. The CALL syntax and results of this routine are the same as found in:

```
CREATE PROCEDURE swapParms (
  IN parm1 CHARACTER,
  OUT parm2  CHARACTER,
  INOUT parm3 CHARACTER )
```

```
BEGIN
   SET parm2 = parm3;
   SET parm3 = parm1;
 END;
```

ESQL example 2

This example procedure shows the recursive use of an ESQL routine. It parses a tree, visiting all places at and below the specified starting point, and reports what it has found:

```
SET OutputRoot.MQMD = InputRoot.MQMD;

  DECLARE answer CHARACTER;
  SET     answer = '';

  CALL navigate(InputRoot.XMLNS, answer);
  SET OutputRoot.XMLNS.Data.FieldNames = answer;


  CREATE PROCEDURE navigate (IN root REFERENCE, INOUT answer CHARACTER)
  BEGIN
    SET answer = answer || 'Reached Field... Type:'
    || CAST(FIELDTYPE(root) AS CHAR)||
    ': Name:' || FIELDNAME(root) || ': Value :' || root || ': ';

    DECLARE cursor REFERENCE TO root;
    MOVE cursor FIRSTCHILD;
    IF LASTMOVE(cursor) THEN
      SET answer = answer || 'Field has children... drilling down ';
    ELSE
      SET answer = answer || 'Listing siblings... ';
    END IF;

    WHILE LASTMOVE(cursor) DO
      CALL navigate(cursor, answer);
      MOVE cursor NEXTSIBLING;
    END WHILE;

    SET answer = answer || 'Finished siblings... Popping up ';
  END;
```

When given the following input message:

```
<Person>
  <Name>John Smith</Name>
  <Salary period='monthly' taxable='yes'>-1200</Salary>
</Person>
```

the procedure produces the following output, which has been manually formatted:

```
  Reached Field... Type:16777232: Name:XML: Value :: Field has children...
  drilling down
  Reached Field... Type:16777216: Name:Person: Value :: Field has children...
  drilling down
  Reached Field... Type:16777216: Name:Name:
  Value :John Smith: Field has children... drilling down
  Reached Field... Type:33554432: Name::
  Value :John Smith: Listing siblings... Finished siblings... Popping up
  Finished siblings... Popping up
  Reached Field... Type:16777216: Name:Salary:
  Value :-1200: Field has children... drilling down
  Reached Field... Type:50331648: Name:period:
  Value :monthly: Listing siblings... Finished siblings... Popping up
  Reached Field... Type:50331648: Name:taxable:
  Value :yes: Listing siblings... Finished siblings... Popping up
  Reached Field... Type:33554432: Name::
```

```
Value :-1200: Listing siblings... Finished siblings... Popping up
Finished siblings... Popping up
Finished siblings... Popping up
Finished siblings... Popping up
```

Java routines

A Java routine is implemented as a Java method, and has a LANGUAGE clause of JAVA. For Java routines, the *ExternalRoutineName* must contain the class name and method name of the Java method to be called. Specify the *ExternalRoutineName* like the following example:

```
>>--"-- className---.---methodName--"-------------><
```

where *className* identifies the class that contains the method and *methodName* identifies the method to invoke. If the class is part of a package, the class identifier part must include the complete package prefix; for example,"`com.ibm.broker.test.MyClass.myMethod`"

To find the Java class, the broker uses the search method that is described in "Deploying Java classes" on page 5099.

Any Java method that you want to invoke must have the following basic signature:

```
public static <return-type> <method-name> (< 0 - N parameters>)
```

Where `<return-type>` must be in the list of Java IN data types in the table in "ESQL to Java data type mapping" on page 5098 (excluding the REFERENCE type, which is not permitted as a return value), or the Java void data type. The parameter data types must also be in the "ESQL to Java data type mapping" on page 5098 table. In addition, the Java method is not allowed to have `exception throws` clause in its signature.

The Java methods signature must match the ESQL routines declaration of the method. You must also observe the following rules:

- Ensure that the Java method name, including the class name and any package qualifiers, matches the procedures EXTERNAL NAME.
- If the Java return type is void, do not put a RETURNS clause on the ESQL routines definition. Conversely, if the Java return type is not void, you must put a RETURNS clause on the ESQL routines definition.
- Ensure that every parameters type and direction matches the ESQL declaration, according to the rules listed in the table in "ESQL to Java data type mapping" on page 5098.
- Ensure that the methods return type matches the data type of the RETURNS clause.
- Enclose EXTERNAL NAME in quotation marks because it must contain at least "class.method".
- If you want to invoke an overloaded Java method, you must create a separate ESQL definition for each overloaded method and give each ESQL definition a unique routine name.

You can use the Java user-defined node API in your Java method, if you observe the restrictions documented in "Restrictions on Java routines" on page 5099. For more information about using the Java API, see "Compiling a Java user-defined node" on page 3074.

Java routine example 1

This routine contains three parameters of varying directions, and returns an integer, which maps to a Java return type of `java.lang.Long`.

```
CREATE FUNCTION  myProc1( IN P1 INTEGER, OUT P2 INTEGER, INOUT P3 INTEGER )
 RETURNS INTEGER
 LANGUAGE JAVA
 EXTERNAL NAME "com.ibm.broker.test.MyClass.myMethod1";
```

You can use the following ESQL to invoke `myProc1`:

```
CALL myProc1( intVar1, intVar2, intVar3) INTO intReturnVar3;
-- or
SET intReturnVar3 = myProc1( intVar1, intVar2, intVar3);
```

Java routine example 2

This routine contains three parameters of varying directions and has a Java return type of `void`.

```
CREATE PROCEDURE myProc2( IN P1 INTEGER, OUT P2 INTEGER, INOUT P3 INTEGER )
 LANGUAGE JAVA
 EXTERNAL NAME "com.ibm.broker.test.MyClass.myMethod2";
```

You must use the following ESQL to invoke `myProc2`:

```
CALL myProc2(intVar1, intVar2, intVar3);
```

The following Java class provides a method for each of the preceding Java examples:

```
package com.ibm.broker.test;

class MyClass {
public static Long myMethod1( Long P1, Long[] P2 Long[] P3) { ... }
public static void myMethod2( Long P2, Long[] P2 Long[] P3) { ... }

 /* When either of these methods is called:
    P1 might or might not be NULL (depending on the value of intVar1).
    P2[0] is always NULL (whatever the value of intVar2).
    P3[0] might or might not be NULL (depending on the value of intVar3).
    This is the same as with LANGUAGE ESQL routines.
    When these methods return:
         intVar1 is unchanged
         intVar2 might still be NULL or might have been changed
         intVar3 might contain the same value or might have been changed.
     This is the same as with LANGUAGE ESQL routines.

    When myMethod1 returns: intReturnVar3 is either  NULL (if the
    method returns NULL) or it contains the value returned by the
    method.
 */
}
```

ESQL to Java data type mapping

The following table summarizes the mappings from ESQL to Java.

Notes:

- Only the Java scalar wrappers are passed to Java.
- The ESQL scalar types are mapped to Java data types as object wrappers, or object wrapper arrays, depending upon the direction of the procedure parameter. Each wrapper array contains exactly one element.

- Scalar object wrappers are used to allow NULL values to be passed to and from Java methods.

ESQL data types [1]	**Java IN data types**	**Java INOUT and OUT data types**
INTEGER, INT	java.lang.Long	java.lang.Long []
FLOAT	java.lang.Double	java.lang.Double[]
DECIMAL	java.math.BigDecimal	java.math.BigDecimal[]
CHARACTER, CHAR	java.lang.String	java.lang.String[]
BLOB	byte[]	byte[][]
BIT	java.util.BitSet	java.util.BitSet[]
DATE	com.ibm.broker.plugin.MbDate	com.ibm.broker.plugin.MbDate[]
TIME [2]	com.ibm.broker.plugin.MbTime	com.ibm.broker.plugin.MbTime[]
GMTTIME [2]	com.ibm.broker.plugin.MbTime	com.ibm.broker.plugin.MbTime[]
TIMESTAMP [2]	com.ibm.broker.plugin.MbTimestamp	com.ibm.broker.plugin.MbTimestamp[]
GMTTIMESTAMP [2]	com.ibm.broker.plugin.MbTimestamp	com.ibm.broker.plugin.MbTimestamp[]
INTERVAL	Not supported	Not supported
BOOLEAN	java.lang.Boolean	java.lang.Boolean[]
REFERENCE (to a message tree) [3] [4] [5] [6]	com.ibm.broker.plugin.MbElement	com.ibm.broker.plugin.MbElement[] **(Supported for INOUT. Not supported for OUT)**
ROW	Not supported	Not supported
LIST	Not supported	Not supported

1. Variables that are declared to be CONSTANT (or references to variables that are declared to be CONSTANT) are not allowed to have the direction INOUT or OUT.
2. The time zone set in the Java variable is not important; you obtain the required time zone in the output ESQL.
3. The reference parameter cannot be NULL when passed into a Java method.
4. The reference cannot have the direction OUT when passed into a Java method.
5. If an *MbElement* is passed back from Java to ESQL as an INOUT parameter, it must point to a location in the same message tree as that pointed to by the *MbElement* that was passed into the called Java method.

   For example, if an ESQL reference to `OutputRoot.XML.Test` is passed into a Java method as an INOUT *MbElement*, but a different *MbElement* is passed back to ESQL when the call returns, the different element must also point to somewhere in the `OutputRoot` tree.
6. An *MbElement* cannot be returned from a Java method with the RETURNS clause, because no ESQL routine can return a reference. However, an *MbElement* can be returned as an INOUT direction parameter, subject to the conditions described in point 5.

A REFERENCE to a scalar variable can be used in the CALL of a Java method, provided that the data type of the variable to which the reference refers matches the corresponding data type in the Java program signature.

Restrictions on Java routines

The following restrictions apply to Java routines that are called from ESQL:

- The Java method must be threadsafe (reentrant).
- Database connections must be JDBC type 2 or type 4. Furthermore, database operations are not part of a broker transaction and therefore cannot be controlled by an external resource coordinator (as is the case in an XA environment).
- The Java user-defined node API must be used only by the same thread that invoked the Java method.

  You can create threads inside your method. However, created threads must not use the Java APIs, and you must return control back to the broker.

  All restrictions that apply to the usage of the Java API also apply to Java methods that are called from ESQL.
- Java methods that are called from ESQL must not use the `MbNode` class. Therefore, they cannot create objects of type `MbNode`, or call any of the methods on an existing `MbNode` object.
- WebSphere MQ or JMS work done inside a Java method that is called from ESQL must be done in accordance with the guidelines for performing WebSphere MQ and JMS work in a user-defined node. See "Planning user-defined input nodes" on page 2995.

Deploying Java classes

You can deploy your Java classes to a broker within a Java Archive (JAR) file, by using one of the following two methods:

1. **Add the JAR file to the broker archive (BAR) file**

   The most efficient and flexible method of deploying to the broker is to add your JAR file to the BAR file. You can do this manually or automatically using the WebSphere Message Broker Toolkit.

   If the WebSphere Message Broker Toolkit finds the correct Java class inside a referenced Java project open in the workspace, it automatically compiles the Java class into a JAR file and adds it to the BAR file. This procedure is the same procedure that you follow to deploy a JavaCompute node inside a JAR, as described in "User-defined node class loading" on page 3120.

   When you deploy a JAR file from the WebSphere Message Broker Toolkit, the flow that has been redeployed reloads the JAR file contained in the BAR file.

   The files are also reloaded if the message flow that references a Java class is stopped and restarted. Ensure that you stop and restart (or redeploy) all flows that reference the JAR file that you want to update. This action avoids the problem of some flows running with the old version of the JAR file and other flows running with the new version.

   The WebSphere Message Broker Toolkit deploys only JAR files; it does not deploy stand-alone Java class files.
2. **Store the JAR file in either of the following locations:**

   a. The *workpath*`/shared-classes/` folder on the machine running the broker

   b. The CLASSPATH environment variable on the computer running the broker

   You must complete this action manually; you cannot use the WebSphere Message Broker Toolkit.

   In this method, redeploying the message flow does not reload the referenced Java classes; neither does stopping and restarting the message flow. The only way to reload the classes in this case is to stop and restart the broker itself.

To enable the broker to find a Java class, ensure that it is in one of the preceding locations. If the broker cannot find the specified class, it generates an exception.

Although you have the choices shown previously when you deploy the JAR file, by using the WebSphere Message Broker Toolkit to deploy the BAR file provides the greatest flexibility when redeploying the JAR file.

Database routines

Database routines are implemented as database stored procedures. Database routines have a LANGUAGE clause of DATABASE, and must have a routine type of PROCEDURE.

When writing stored procedures in languages like C, you must use NULL indicators to ensure that your procedure can process the data correctly.

Although the database definitions of a stored procedure vary between the databases, the ESQL used to invoke them does not. The names given to parameters in the ESQL do not have to match the names they are given on the database side. However, the external name of the routine, including any package or container specifications, must match its defined name in the database.

The DYNAMIC RESULT SETS clause is allowed only for database routines. It is required only if a stored procedure returns one or more result sets. The integer parameter to this clause must be 0 (zero) or more, and specifies the number of result sets to be returned.

The optional RETURNS clause is required if a stored procedure returns a single scalar value.

The EXTERNAL NAME clause specifies the name by which the database knows the routine. Can be either a qualified or an unqualified name, where the qualifier is the name of the database schema in which the procedure is defined. If you do not provide a schema name, the database connection user name is used as the schema in which to locate the procedure. If the required procedure does not exist in this schema, you must provide an explicit schema name, either on the routine definition or on the CALL to the routine at run time. For more information about dynamically choosing the schema that contains the routine, see the “CALL statement” on page 5077. When a qualified name is used, the name must be in quotation marks.

A fully qualified routine typically takes the form:

```
EXTERNAL NAME "mySchema.myProc";
```

However, if the procedure belongs to an Oracle package, the package is treated as part of the procedures name. Therefore you must provide a schema name and the package name, in the form:

```
EXTERNAL NAME "mySchema.myPackage.myProc";
```

This form allows the schema, but not the package name, to be chosen dynamically in the CALL statement.

If the name of the procedure contains SQL wildcards (which are the percent (%) character and the underscore (_) character), the procedure name is modified by the broker to include the database escape character immediately before each wildcard character. This technique ensures that the database receives the wildcards as literal

characters. For example, assuming that the database escape character is a backslash, the following clause is modified by the broker so that "mySchema.Proc_" is passed to the database.

```
EXTERNAL NAME "mySchema.Proc_";
```

All external procedures have the following restrictions:

- A stored procedure cannot be overloaded on the database side. A stored procedure is considered overloaded if there is more than one procedure of the same name in the same database schema. If the broker detects that a procedure has been overloaded, it raises an exception.
- Parameters cannot be of the ESQL REFERENCE, ROW, LIST, or INTERVAL data types.
- User-defined types cannot be used as parameters or as return values.

For LANGUAGE DATABASE routines, the *ExternalRoutineName* is not optional and contains the schema name, package name, and procedure name of the routine to be called. Specify the *ExternalRoutineName* as follows:

```
>>--"schemaName---.---packageName---.---procedureName--"-------------><
```

where:

- *schemaName* is optional.
- *packageName* is optional and applies only to Oracle data sources. If you supply a *packageName* you must supply a *schemaName.*
- *procedureName* is not optional.

Database routine example 1

The following example shows an ESQL definition of a stored procedure that returns a single scalar value and an OUT parameter:

```
CREATE PROCEDURE myProc1(IN P1 INT, OUT P2 INT)
RETURNS INTEGER
LANGUAGE DATABASE
EXTERNAL NAME "myschema.myproc";
```

Use this ESQL to invoke the myProc1 routine:

```
/*using CALL statement invocation syntax*/
CALL myProc1(intVar1, intVar2) INTO intReturnVar3;

/*or using function invocation syntax*/
SET intReturnVar3 = myProc1(intVar1, intVar2);
```

Database routine example 2

The following ESQL code demonstrates how to define and call DB2 stored procedures:

```
ESQL Definition:
DECLARE inputParm CHARACTER;
DECLARE outputParm CHARACTER;
DECLARE inputOutputParm CHARACTER;

SET inputParm = 'Hello';
SET inputOutputParm = 'World';
CALL swapParms( inputParm, outputParm, inputOutputParm );

CREATE PROCEDURE swapParms (
  IN parm1 CHARACTER,
  OUT parm2  CHARACTER,
```

```
  INOUT parm3 CHARACTER
)
LANGUAGE DATABASE
EXTERNAL NAME dbSwapParms;
```

To register this stored procedure with DB2, copy the following script to a file (for example, `test1.sql`)

```
-- DB2 Example Stored Procedure
DROP PROCEDURE dbSwapParms @
CREATE PROCEDURE dbSwapParms
( IN in_param CHAR(32),
  OUT out_param CHAR(32),
  INOUT inout_param CHAR(32))
LANGUAGE SQL
BEGIN
SET out_param = inout_param;
    SET inout_param = in_param;
END @
```

Now run the file from the DB2 command prompt:

```
db2 -td@ -vf test1.sql
```

Expect the following results from running this code:

- The value of the IN parameter does not (and cannot, by definition) change.
- The value of the OUT parameter becomes “World”.
- The value of the INOUT parameter changes to “Hello”.

Database routine example 3

The following ESQL code demonstrates how to define and call Oracle stored procedures:

```
ESQL Definition:
DECLARE inputParm CHARACTER;
DECLARE outputParm CHARACTER;
DECLARE inputOutputParm CHARACTER;

SET inputParm = 'Hello';
SET inputOutputParm = 'World';
CALL swapParms( inputParm, outputParm, inputOutputParm );

CREATE PROCEDURE swapParms (
  IN parm1 CHARACTER,
  OUT parm2  CHARACTER,
  INOUT parm3 CHARACTER
)
LANGUAGE DATABASE
EXTERNAL NAME dbSwapParms;
```

To register this stored procedure with Oracle, copy the following script to a file (for example, `test1.sql`)

```
CREATE OR REPLACE PROCEDURE dbSwapParms
( in_param IN VARCHAR2,
  out_param OUT VARCHAR2,
  inout_param IN OUT VARCHAR2 )
AS
BEGIN
  out_param := inout_param;
  inout_param := in_param;
END;
/
```

Now run the file:

```
sqlplus userID/password  @test1.sql
```

Expect the following results from running this code:
- The value of the IN parameter does not (and cannot, by definition) change.
- The value of the OUT parameter becomes “World”.
- The value of the INOUT parameter changes to “Hello”.

Database routine example 4

The following ESQL code demonstrates how to define and call SQL Server stored procedures:

```
ESQL Definition:
DECLARE inputParm CHARACTER;
DECLARE outputParm CHARACTER;
DECLARE inputOutputParm CHARACTER;

SET inputParm = 'Hello';
SET inputOutputParm = 'World';
CALL swapParms( inputParm, outputParm, inputOutputParm );

CREATE PROCEDURE swapParms (
  IN parm1 CHARACTER,
  INOUT parm2  CHARACTER,
  INOUT parm3 CHARACTER
)
LANGUAGE DATABASE
EXTERNAL NAME dbSwapParms;
```

To register this stored procedure with SQL Server, copy the following script to a file (for example, `test1.sql`)

```
-- SQLServer Example Stored Procedure
DROP PROCEDURE dbSwapParms
go
CREATE PROCEDURE dbSwapParms
 @in_param     CHAR(32),
 @out_param    CHAR(32) OUT,
 @inout_param  CHAR(32) OUT
AS
  SET NOCOUNT ON
  SET @out_param   = @inout_param
  SET @inout_param = @in_param
go
```

Now run file:

```
isql -UuserID -Ppassword -Sserver -ddatasource -itest1.sql
```

SQL Server considers `OUTPUT` parameters from stored procedures as `INPUT/OUTPUT` parameters. If you declare them as `OUT` parameters in your ESQL you encounter a type mismatch error at run time. To avoid that mismatch you must declare SQL Server `OUTPUT` parameters as `INOUT` in your ESQL.

Use the `SET NOCOUNT ON` option, as shown in the preceding example, with SQL stored procedures for the following reasons:

1. To limit the amount of data returned from SQL Server to the broker.
2. To allow result sets to be returned correctly.

Expect the following results from running this code:
- The value of the IN parameter does not (and cannot, by definition) change.
- The value of the OUT parameter becomes “World”.

- The value of the INOUT parameter changes to “Hello”.

Database routine example 5

The following ESQL code demonstrates how to define and call Sybase stored procedures:

```
ESQL Definition:
DECLARE inputParm CHARACTER;
DECLARE outputParm CHARACTER;
DECLARE inputOutputParm CHARACTER;

SET inputParm = 'Hello';
SET inputOutputParm = 'World';
CALL swapParms( inputParm, outputParm, inputOutputParm );

CREATE PROCEDURE swapParms (
  IN parm1 CHARACTER,
  INOUT parm2  CHARACTER,
  INOUT parm3 CHARACTER
)
LANGUAGE DATABASE
EXTERNAL NAME dbSwapParms;
```

To register this stored procedure with Sybase, copy the following script to a file (for example, `test1.sql`)

```
-- SYBASE Example Stored Procedure
DROP PROCEDURE dbSwapParms
go
CREATE PROCEDURE dbSwapParms
 @in_param     CHAR(32),
 @out_param    CHAR(32) OUT,
 @inout_param  CHAR(32) OUT
AS
  SET @out_param   = @inout_param
  SET @inout_param = @in_param
go
```

Now run file:

```
isql -U<userID> -P<password> -S<server> -D<datasource> -itest1.sql
```

Sybase considers `OUTPUT` parameters from stored procedures as `INPUT/OUTPUT` parameters. If you declare them as `OUT` parameters in your ESQL, you encounter a type mismatch error at run time. To avoid that mismatch, declare Sybase `OUTPUT` parameters as `INOUT` in your ESQL.

Expect the following results from running this code:
- The value of the IN parameter does not (and cannot, by definition) change.
- The value of the OUT parameter becomes “World”.
- The value of the INOUT parameter changes to “Hello”.

Database routine example 6

The following ESQL code demonstrates how to define and call Informix stored procedures:

```
ESQL Definition:
DECLARE inputParm CHARACTER 'Hello';
DECLARE outputParm CHARACTER;
DECLARE inputOutputParm CHARACTER 'World';
CALL swapParms( inputParm, outputParm, inputOutputParm );
```

```
CREATE PROCEDURE swapParms (
  IN parm1 CHARACTER,
  INOUT parm2  CHARACTER,
  INOUT parm3 CHARACTER
)
LANGUAGE DATABASE
EXTERNAL NAME dbSwapParms;
```

To register this stored procedure with Informix, copy the following script to a file (for example, `test1.sql`)

```
DROP SPECIFIC PROCEDURE dbSwapParms;
CREATE PROCEDURE dbSwapParms
  (       inParm     CHAR(20),
   OUT    outParm    CHAR(20),
   INOUT  inoutParm  CHAR(20))

   SPECIFIC dbSwapParms

    LET outParm   = inoutParm;
    LET inoutParm = inParm;
END PROCEDURE;
```

Now run file:

From the Informix server shell environment enter:

```
dbaccess <dataBaseName> <fully qualified path/test1.sql>
```

Expect the following results from running this code:
- The value of the IN parameter does not (and cannot, by definition) change.
- The value of the OUT parameter becomes “World”.
- The value of the INOUT parameter changes to “Hello”.

The following restrictions apply to Informix stored procedures:
- Procedures that use the Informix INTERVAL datatype cannot be invoked from the broker.
- Procedures can return only one result set.
- Procedures that return a result set must contain only IN parameters.
- Procedures cannot return CLOBs or BLOBs in result sets or as scalar return values.
- Procedures can return either a result set or a scalar value, but not both.

Database routine example 7

This example shows how to call a stored procedure that returns two result sets, in addition to an out parameter:

```
CREATE PROCEDURE myProc1 (IN P1 INT, OUT P2 INT)
  LANGUAGE DATABASE
  DYNAMIC RESULT SETS 2
  EXTERNAL NAME "myschema.myproc";
```

Use the following ESQL to invoke `myProc1`:

```
/* using a field reference */
CALL myProc1(intVar1, intVar2, Environment.RetVal[], OutputRoot.XMLNS.A[])
/* using a reference variable*/
CALL myProc1(intVar1, intVar2, myReferenceVariable.RetVal[], myRef2.B[])
```

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Invoking stored procedures" on page 2503
To invoke a procedure that is stored in a database, use the ESQL CALL statement. The stored procedure must be defined by a CREATE PROCEDURE statement that has a Language clause of DATABASE and an EXTERNAL NAME clause that identifies the name of the procedure in the database and, optionally, the database schema to which it belongs.

"Compiling a Java user-defined node" on page 3074
When you have created the code for your Java user-defined node, you must compile it for your operating system.

"Adding files to a broker archive" on page 3223
To deploy files to an execution group, include them in a broker archive (BAR) file.

Related reference:

"Syntax diagrams" on page 3677

"ESQL statements" on page 5067
You can use ESQL statements to manipulate message trees, update databases, or interact with nodes.

"CALL statement" on page 5077
The CALL statement calls (invokes) a routine.

"CREATE FUNCTION statement" on page 5091
The CREATE FUNCTION statement defines a callable function or procedure.

"ESQL-to-Java data-type mapping table" on page 5043
Table summarizing the mappings from ESQL to Java.

Related information:

Java user-defined extensions API

DECLARE statement:

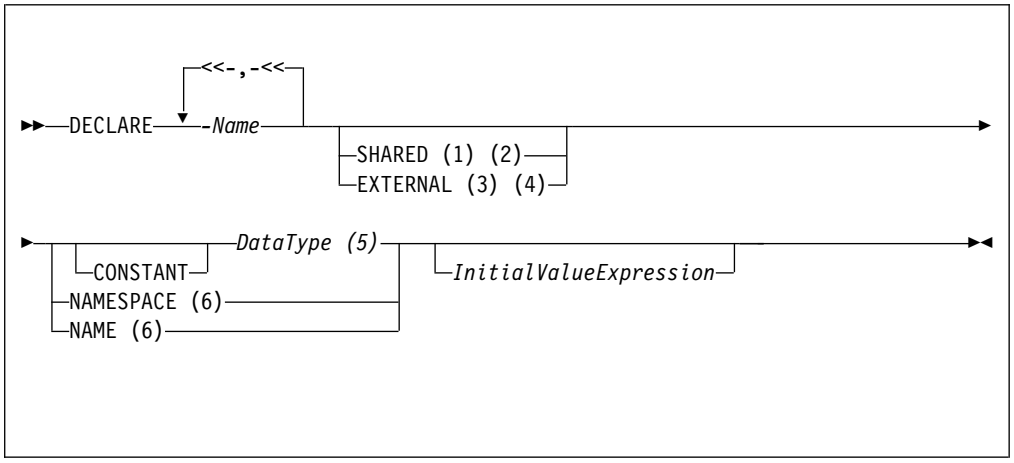
Use the DECLARE statement to define a variable, the data type of the variable and, optionally, its initial value.

You can define three types of variable with the DECLARE statement:

- External
- Normal
- Shared

For further information, see "ESQL variables" on page 5048.

Syntax



Notes:

1. The SHARED keyword is not valid within a function or procedure.
2. You cannot specify SHARED with a *DataType* of REFERENCE TO. To store a message tree in a shared variable, use the ROW data type.
3. EXTERNAL variables are implicitly constant.
4. It is good programming practice to give an EXTERNAL variable an initial value.
5. If you specify a *DataType* of REFERENCE TO, you must specify an initial value (of either a variable or a tree) in *InitialValueExpression*.
6. When you use the NAMESPACE and NAME clauses, their values are implicitly constant and of type CHARACTER.

Follow the links to see more information about all these parameters:

- "CONSTANT"
- "DataType" on page 5119
- "EXTERNAL" on page 5119
- "NAME" on page 5120
- "NAMESPACE" on page 5121
- "SHARED" on page 5121

CONSTANT

Use CONSTANT to define a constant. You can declare constants within schemas, modules, routines, or compound statements (both implicit and explicit). The behavior of these cases is shown in the following list:

- Within a compound statement, constants and variables occupy the same namespace.
- Within expressions, a constant or variable that is declared within a compound statement overlays all constants and variables of the same name that are declared in containing compound statements, modules, and schemas.
- Within field reference namespace fields, a namespace constant that is declared within a compound statement overlays all namespace constants of the same name that are declared in containing compound statements.

A constant or variable that is declared within a routine overlays any parameters of the same name, as well as all constants and variables of the same name that are declared in a containing module or schema.

DataType

The values that you can specify for *DataType* are:
- BOOLEAN
- INT
- INTEGER
- FLOAT
- DECIMAL
- DATE
- TIME
- TIMESTAMP
- GMTTIME
- GMTTIMESTAMP
- INTERVAL. This value does not apply to external variables (EXTERNAL keyword specified).
- CHAR
- CHARACTER
- BLOB
- BIT
- ROW. This value does not apply to external variables (EXTERNAL keyword specified).
- REFERENCE TO. This value does not apply to external or shared variables (EXTERNAL or SHARED keyword specified).

EXTERNAL

Use EXTERNAL to denote a user-defined property (UDP). A UDP is a user-defined constant whose initial value (optionally set by the DECLARE statement) can be modified, at design time, by the Message Flow editor (see "Message Flow editor" on page 6810) or overridden, at deployment time, by the Broker Archive editor (see "Broker Archive editor" on page 6794). The value of a UDP cannot be modified by ESQL.

When a UDP is given an initial value on the DECLARE statement, this value becomes its default. However, any value that you specify in the Message Flow editor at design time, or in the Broker Archive editor at deployment time (even a zero length string) overrides any initial value that was coded on the DECLARE statement.

For example, if you code:

```
DECLARE deployEnvironment EXTERNAL CHARACTER 'Dev';
```

you have defined a UDP variable of `deployEnvironment` with an initial value `Dev`.

Add the UDP to the message flow by using the UDP tab in the message flow editor. When you add the flow to the BAR file, the UDP is there as an attribute of the flow; you must name the attribute to be the same as the ESQL variable in the DECLARE statement (in this case, `deployEnvironment`) to ensure that the initial value that you set is unchanged.

All the UDPs in a message flow must have a value, given either on the DECLARE statement or by the Message Flow or Broker Archive editor; otherwise a

deployment-time error occurs. At run time, after the UDP has been declared, its value can be queried by subsequent ESQL statements.

You can define a UDP for a subflow. A UDP has global scope and is not specific to a particular subflow. If you reuse a subflow in a message flow, and those subflows have identical UDPs, you cannot set the UDPs to different values.

The advantage of UDPs is that their values can be changed at deployment time. For example, if you use the UDPs to hold configuration data, it means that you can configure a message flow for a particular computer, task, or environment at deployment time, without having to change the code at the node level. UDPs can also be modified at run time by using the CMP API.

You can declare UDPs only in modules or schemas; that is, you can use the DECLARE statement with the EXTERNAL keyword only at the MODULE or SCHEMA level. If you use a DECLARE statement with the EXTERNAL keyword within a PROCEDURE or FUNCTION, a `BIP2402E` exception occurs when you deploy the message flow.

The following types of broker node can access UDPs:
- Compute node
- Database node
- Filter node
- Nodes that are derived from these node types

Take care when specifying the data type of a UDP, because a CAST is used to change the value to the requested *DataType*.

For an overview of UDPs, see "User-defined properties in ESQL" on page 2376.

Example 1:

```
DECLARE mycolor EXTERNAL CHARACTER 'blue';
```

Example 2:

```
DECLARE TODAYSCOLOR EXTERNAL CHARACTER;
SET COLOR = TODAYSCOLOR;
```

where `TODAYSCOLOR` is a user-defined property that has a `TYPE` of `CHARACTER` and a `VALUE` set by the Message Flow editor.

NAME

Use NAME to define an alias (an alternative name) by which a variable can be known.

Example 1:

```
-- The following statement gives Schema1 an alias of 'Joe'.
DECLARE Schema1 NAME 'Joe';
-- The following statement produces a field called 'Joe'.
SET OutputRoot.XMLNS.Data.Schema1 = 42;

-- The following statement inserts a value into a table called Table1
-- in the schema called 'Joe'.
INSERT INTO Database.Schema1.Table1 (Answer) VALUES 42;
```

Example 2:

```
-- At Module scope define ColourElementName and set it external
-- so that its default value of 'black' can be overridden as a UDP
DECLARE ColourElementName EXTERNAL NAME 'black';


-- Use the ColourElementName in a function
CREATE FIRSTCHILD OF OutputRoot.XMLNSC.TestCase.ColourElementName
                     Domain('XMLNSC')
                     NAME 'Node1' VALUE '1';
```

If the owning message flow has been configured with a UDP named `ColourElementName` of type `String`, which has been given the value `red`, the following output message is generated:

```
<xml version="1.0"?>
<TestCase>
  <red>
    <Node1>1</Node1>
  </red>
```

NAMESPACE

Use NAMESPACE to define an alias (an alternative name) by which a namespace can be known.

Example:

This example illustrates a namespace declaration, its use as a *SpaceId* in a path, and its use as a character constant in a namespace expression:

```
DECLARE prefixOne NAMESPACE 'http://www.example.com/PO1';

-- On the right hand side of the assignment a namespace constant
-- is being used as such while, on the left hand side, one is
-- being used as an ordinary constant (that is, in an expression).

SET OutputRoot.XMLNS.{prefixOne}:{'PurchaseOrder'} =
               InputRoot.XMLNS.prefixOne:PurchaseOrder;
```

SHARED

Use SHARED to define a shared variable. Shared variables are private to the flow (if declared within a schema) or node (if declared within a module), but are shared between instances of the flow (threads). No type of variable is visible beyond the flow level; for example, you cannot share variables across execution groups.

You can use shared variables to implement an in-memory cache in the message flow; see “Optimizing message flow response times” on page 3264. Shared variables have a long lifetime and are visible to multiple messages passing through a flow; see “Long-lived variables” on page 2378.

Shared variables exist for the lifetime of the:

- Execution group process
- Flow or node, or
- Node ESQL code that declares the variable

(whichever is the shortest). They are initialized when the first message passes through the flow or node after each broker startup.

You cannot define a shared variable within a function or procedure.

The advantages of shared variables, relative to databases, are that:

- Write access is much faster.
- Read access to small data structures is faster.
- Access is direct; that is, there is no need to use a special function (SELECT) to get data, or special statements (INSERT, UPDATE, or DELETE) to modify data. You can refer to the data directly in expressions.

The advantages of databases, relative to shared variables, are that:

- The data is persistent.
- The data is changed transactionally.

These read/write variables are ideal for users who are prepared to sacrifice the persistence and transactional advantages of databases in order to obtain better performance, because they have a longer life than only one message and perform better than a database.

Because SHARED variables can be updated by multiple additional instances, you must ensure that you do not change SHARED variables that might cause unexpected results, for example, if the variable is being used as a counter.

As SHARED variables are initialized once on the first message through a node, it is possible to initialize a SHARED ROW variable once with the results from a Database query. The following code shows an example of how this is achieved:

```
 CREATE SCHEMA testSchema
    DECLARE mySharedRow SHARED ROW;
    DECLARE initialized SHARED BOOLEAN myINIT();

    CREATE COMPUTE MODULE testModule

    CREATE FUNCTION Main() RETURNS BOOLEAN
    BEGIN
    SET OutputRoot.XMLNSC.Top.TEST.Result1 VALUE = initialized;
    SET OutputRoot.XMLNSC.Top.TEST.Result2 = mySharedRow;
  END;

END MODULE;

CREATE FUNCTION myINIT( ) RETURNS BOOLEAN
BEGIN
    LOG EVENT VALUES('myINIT CALLED');
    SET mySharedRow.Top[] = SELECT A.MyCol1, A.MyCol2 from Database.Test AS A;
    RETURN TRUE;
END;
```

You can prevent other instances seeing the intermediate stages of the data by using a BEGIN ATOMIC construct; see “BEGIN ... END statement” on page 5070.

Your user program can make an efficient read, or write, copy of an input message in the input node by using shared-row variables, which simplifies the technique for handling large messages.

Restriction:

Subtrees cannot be copied directly from one shared row variable to another shared row variable. Subtrees can be copied indirectly by using a non-shared row variable. Scalar values extracted from one shared row variable (by using the FIELDVALUE function) can be copied to another shared row variable.

Sample program

The following sample shows how to use both shared and external variables:

- Message Routing

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"User-defined properties in ESQL" on page 2376
Access user-defined properties (UDPs) as variables in your ESQL program by specifying the EXTERNAL keyword on a DECLARE statement. For example, the ESQL statement `DECLARE today EXTERNAL CHARACTER 'monday'` defines a user-defined property called today with an initial value monday.

"Long-lived variables" on page 2378
You can use appropriate long-lived ESQL data types to cache data in memory.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Creating dynamic field references" on page 2431
You can use a variable of type REFERENCE as a dynamic reference to navigate a message tree. This acts in a similar way to a message cursor or a variable pointer.

"Configuring a message flow at deployment time with user-defined properties" on page 2626
Use user-defined properties (UDPs) to configure message flows at deployment and run time, without modifying program code. You can give a UDP an initial value when you declare it in your program, or when you use the Message Flow editor to create or modify a message flow.

"Setting message flow user-defined properties at run time in a CMP application" on page 985
Use the CMP API to query, discover, and set message flow user-defined properties dynamically at run time. You can use the CMP API to set properties with a data type of character.

"Optimizing message flow response times" on page 3264
You can use different solutions to improve message flow response times.

Related reference:

"Syntax diagrams" on page 3677

"ESQL statements" on page 5067
You can use ESQL statements to manipulate message trees, update databases, or interact with nodes.

"ESQL variables" on page 5048
ESQL variables can be described as *external variables*, *normal variables*, or *shared variables*; their use is defined in the DECLARE statement.

"ESQL data types in message flows" on page 5020
All data that is referred to in message flows must be one of the defined types.

"Message Flow editor" on page 6810
The Message Flow editor is the default editor provided by the Broker Application Development perspective for defining a graphical representation of a message flow in the WebSphere Message Broker Toolkit, and for setting properties for individual message flow nodes.

"FIELDVALUE function" on page 5234

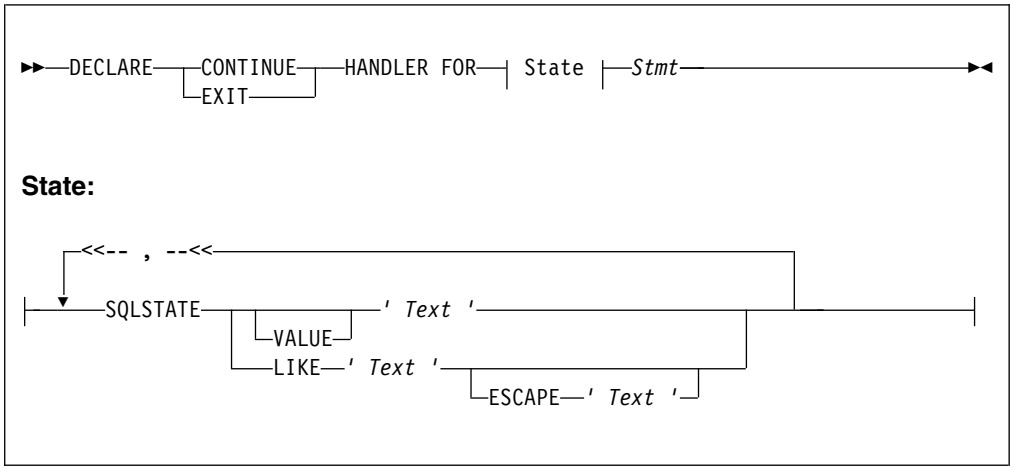
"Broker Archive editor" on page 6794
Use the Broker Archive editor to create and manage broker archive (BAR) files.

"BEGIN ... END statement" on page 5070
The BEGIN ... END statement gives the statements defined within the BEGIN and END keywords the status of a single statement.

DECLARE HANDLER statement:

The DECLARE HANDLER statement creates an error handler for handling exceptions.

Syntax



You can declare handlers in both explicitly declared (BEGIN...END) scopes and implicitly declared scopes (for example, the ELSE clause of an IF statement). However, all handler declarations must be together at the top of the scope, before any other statements.

If there are no exceptions, the presence of handlers has no effect on the behavior or performance of an SQL program. If an exception occurs, WebSphere Message Broker compares the SQL state of the exception with the SQL states associated with any relevant handlers, until either the exception leaves the node (just as it would if there were no handlers) or a matching handler is found. Within any one scope, handlers are searched in the order they are declared; that is, first to last. Scopes are searched from the innermost to outermost.

The SQL state values provided in DECLARE... HANDLER... statements can be compared directly with the SQL state of the exception or can be compared using wildcard characters. To compare the state values directly, specify either VALUE or no condition operator. To make a wildcard comparison, use the underscore and percent characters to represent single and multiple character wildcards,

respectively, and specify the LIKE operator. The wildcard method allows all exceptions of a general type to be handled without having to list them exhaustively.

If a matching handler is found, the SQLSTATE and other special registers are updated (according to the rules described later in this section) and the handler's statement is processed.

As the handler's statement must be a single statement, it is typically a compound statement (such as BEGIN...END) that contains multiple other statements. There is no special behavior associated with these inner statements and there are no special restrictions. They can, for example, include RETURN, ITERATE, or LEAVE; these affect their containing routines and looping constructs in the same way as if they were contained in the scope itself.

Handlers can contain handlers for exceptions occurring within the handler itself

If processing of the handler's code completes without throwing further unhandled exceptions, execution of the normal code is resumed as follows:

- For EXIT handlers, the next statement processed is the first statement after the handler's scope.
- For CONTINUE handlers, it is the first directly-contained statement after the one that produced the exception.

Each handler has its own SQLCODE, SQLSTATE, SQLNATIVEERROR, and SQLERRORTEXT special registers. These come into scope and their values are set just before the handler's first statement is executed. They remain valid until the handler's last statement has been executed. Because there is no carry over of SQLSTATE values from one handler to another, handlers can be written independently.

Handlers absorb exceptions, preventing their reaching the input node and thus causing the transaction to be committed rather than rolled back. A handler can use a RESIGNAL or THROW statement to prevent this.

See “SQLSTATE function” on page 5173 for a list of the valid SQLSTATES that you can use in a HANDLER.

Example 1

The following example demonstrates proper use of a USER EXCEPTION, and passing SQLCODE, SQLSTATE, SQLNATIVEERROR and SQLERRORTEXT from the exception to the handler. The example also uses the SQLSTATE to catch the specific exception:

```
DECLARE retryCount INTEGER 0;
DECLARE afterCount INTEGER 0;

WHILE retryCount <= 10 DO
  DECLARE EXIT HANDLER FOR SQLSTATE VALUE 'U11222'
  BEGIN
    /* This demonstrates how to pass data to the HANDLER in the SQL
       special registers */
    SET OutputRoot.XMLNSC.Top.WHILE.mySQLCODE = SQLCODE;
    SET OutputRoot.XMLNSC.Top.WHILE.mySQLSTATE = SQLSTATE;
    SET OutputRoot.XMLNSC.Top.WHILE.mySQLNATIVEERROR = SQLNATIVEERROR;
    SET OutputRoot.XMLNSC.Top.WHILE.mySQLERRORTEXT = SQLERRORTEXT;
```

```
    SET retryCount = retryCount + 1;

    /* If we are an EXIT HANDLER, control is now passed to back to the
       WHILE statement */
  END;

  /* In a real scenario this could be a PROPAGATE statement, and the exception
     could be thrown by a 'downstream' node. In this case the HANDLER would
     normally cope with a wider range of exception, for example, using LIKE '%' */
  THROW USER EXCEPTION VALUES( -1, 'U11222', 42, 'error text' );

  /* This is the next statement executed if it is a CONTINUE HANDLER */
  SET afterCount = afterCount + 1;

END WHILE;

SET OutputRoot.XMLNSC.Top.WHILE.retryCount = retryCount;
SET OutputRoot.XMLNSC.Top.WHILE.afterCount = afterCount;

With EXIT (as above) the output is:
<Top>
  <WHILE>
    <mySQLCODE>-1</mySQLCODE>
    <mySQLSTATE>U11222</mySQLSTATE>
    <mySQLNATIVEERROR>42</mySQLNATIVEERROR>
    <mySQLERRORTEXT>error text</mySQLERRORTEXT>
    <retryCount>11</retryCOUNT>
    <afterCount>0</afterCOUNT>
  </WHILE>
</Top>

Changing the HANDLER to be CONTINUE (DECLARE CONTINUE HANDLER FOR SQLSTATE VALUE
'U11222') then the output is:
<Top>
  <WHILE>
    <mySQLCODE>-1</mySQLCODE>
    <mySQLSTATE>U11222</mySQLSTATE>
    <mySQLNATIVEERROR>42</mySQLNATIVEERROR>
    <mySQLERRORTEXT>error text</mySQLERRORTEXT>
    <retryCount>11</retryCOUNT>
    <afterCount>11</afterCOUNT>
  </WHILE>
</Top>
```

You see the difference in `afterCount` in the output message.

Example 2

```
-- Drop the tables so that they can be re-created with the latest definition.
-- If the program has never been run before, errors will occur because you
-- can't drop tables that don't exist. We ignore these.
  BEGIN
    DECLARE CONTINUE HANDLER FOR SQLSTATE LIKE'%' BEGIN END;

    PASSTHRU 'DROP TABLE Shop.Customers' TO Database.DSN1;
    PASSTHRU 'DROP TABLE Shop.Invoices'  TO Database.DSN1;
    PASSTHRU 'DROP TABLE Shop.Sales'     TO Database.DSN1;
    PASSTHRU 'DROP TABLE Shop.Parts'     TO Database.DSN1;
  END;
```

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"RESIGNAL statement" on page 5155
The RESIGNAL statement rethrows the current exception, if one exists.

"Syntax diagrams" on page 3677

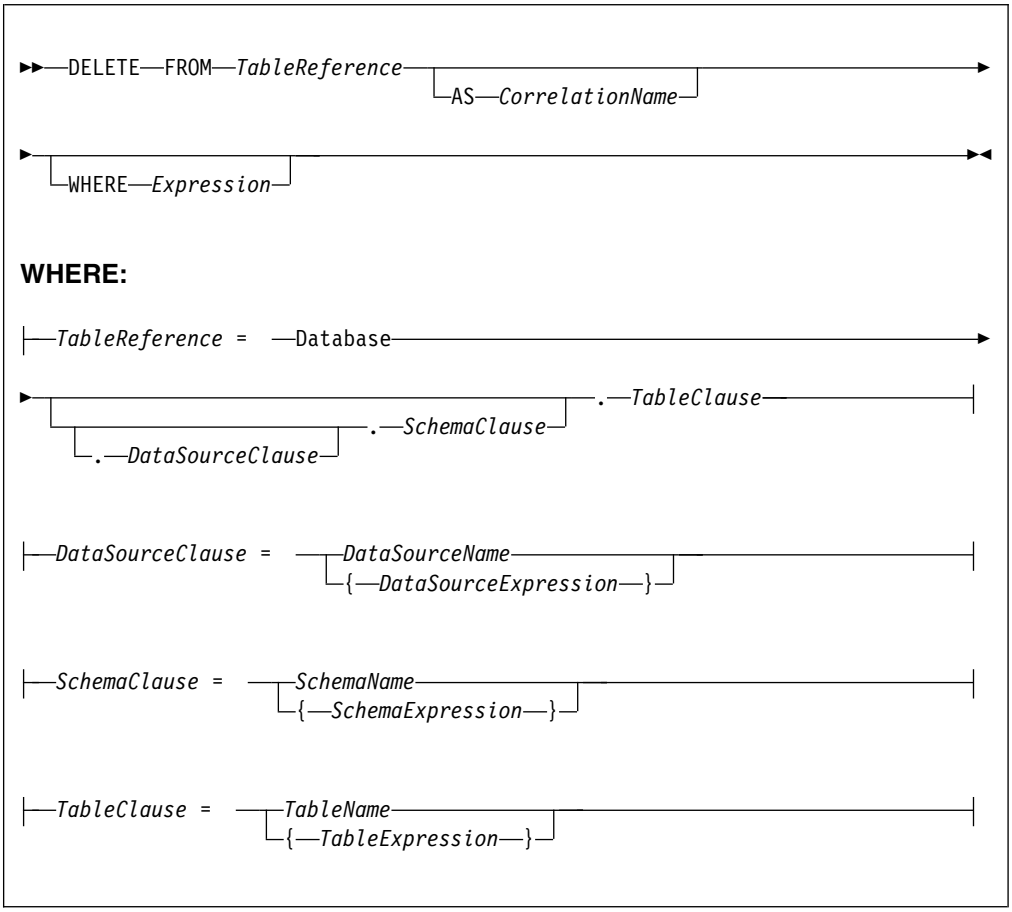
"ESQL statements" on page 5067
You can use ESQL statements to manipulate message trees, update databases, or interact with nodes.

"SQLSTATE function" on page 5173
SQLSTATE is a database state function that returns a 5 character data type of CHARACTER with a default value of '00000' (five zeros as a string).

DELETE FROM statement:

The DELETE FROM statement deletes rows from a table in an external database, based on a search condition.

Syntax



All rows for which the WHERE clause expression evaluates to TRUE are deleted from the table identified by *TableReference*.

Each row is examined in turn and a variable is set to point to the current row. Typically, the WHERE clause expression uses this variable to access column values and thus cause rows to be retained or deleted according to their contents. The variable is referred to by *CorrelationName* or, in the absence of an AS clause, by *TableName*.

Table reference

A table reference is a special case of the field references that are used to refer to message trees. It always starts with the word "Database" and can contain any of the following elements:

- A table name only
- A schema name and a table name
- A data source name (that is, the name of a database instance), a schema name, and a table name

In each case, the name can be specified directly or by an expression enclosed in braces ({...}). A directly-specified data source, schema, or table name is subject to name substitution. That is, if the name used has been declared to be a known name, the value of the declared name is used rather than the name itself (see "DECLARE statement" on page 5117).

If a schema name is not specified, the default schema for the broker's database user is used.

If a data source name is not specified, the database pointed to by the node's `data source` attribute is used.

The WHERE clause

The WHERE clause expression can use any of the broker's operators and functions in any combination. It can refer to table columns, message fields, and any declared variables or constants.

However, be aware that the broker treats the WHERE clause expression by examining the expression and deciding whether the whole expression can be evaluated by the database. If it can, it is given to the database. In order to be evaluated by the database, it must use only those functions and operators supported by the database.

The WHERE clause can, however, refer to message fields, correlation names declared by containing SELECT functions, and to any other declared variables or constants within scope.

If the whole expression cannot be evaluated by the database, the broker looks for top-level AND operators and examines each sub-expression separately. It then attempts to give the database those sub-expressions that it can evaluate, leaving the broker to evaluate the rest. You need to be aware of this situation for two reasons:

1. Apparently trivial changes to WHERE clause expressions can have large effects on performance. You can determine how much of the expression was given to the database by examining a user trace.
2. Some databases' functions exhibit subtle differences of behavior from those of the broker.

Handling errors

It is possible for errors to occur during delete operations. For example, the database might not be operational. In these cases, an exception is thrown (unless the node has its `throw exception on database error` property set to FALSE). These exceptions set appropriate SQL code, state, native error, and error text values and can be dealt with by error handlers (see the DECLARE HANDLER statement).

For further information about handling database errors, see "Capturing database state" on page 2512.

Examples

The following example assumes that the `dataSource` property has been configured and that the database it identifies has a table called SHAREHOLDINGS, with a column called ACCOUNTNO.

```
DELETE FROM Database.SHAREHOLDINGS AS S
       WHERE S.ACCOUNTNO = InputBody.AccountNumber;
```

This example removes all the rows from the SHAREHOLDINGS table where the value in the ACCOUNTNO column (in the table) is equal to that in the `AccountNumber` field in the message. This operation might delete zero, one, or more rows from the table.

The next example shows the use of calculated data source, schema, and table names:

```
-- Declare variables to hold the data source, schema, and table names and
-- set their default values
DECLARE Source CHARACTER 'Production';
DECLARE Schema CHARACTER 'db2admin';
DECLARE Table  CHARACTER 'DynamicTable1';

-- Code which calculates their actual values comes here

-- Delete rows from the table
DELETE FROM Database.{Source}.{Schema}.{Table} As R WHERE R.Name = 'Joe';
```

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

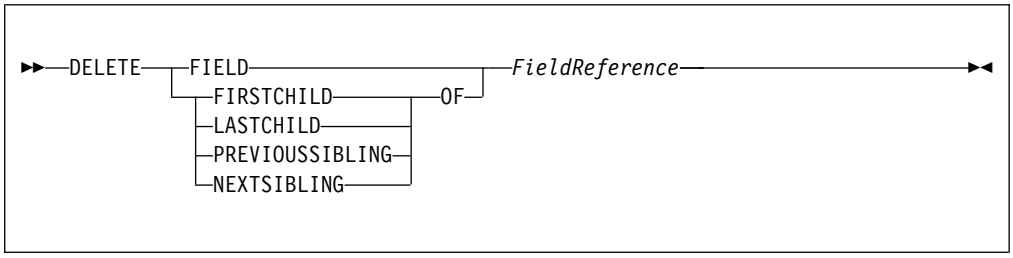
"Syntax diagrams" on page 3677

"ESQL statements" on page 5067
You can use ESQL statements to manipulate message trees, update databases, or interact with nodes.

DELETE statement:

The DELETE statement detaches and destroys a portion of a message tree, allowing its memory to be reused. This statement is particularly useful when handling very large messages.

Syntax



If the target field does not exist, the statement does nothing and normal processing continues. If any reference variables point into the deleted portion, they are disconnected from the tree so that no action involving them has any effect, and the LASTMOVE function returns FALSE. Disconnected reference variables can be reconnected by using a MOVE... TO... statement.

Example

```
DELETE FIELD OutputRoot.XMLNS.Data.Folder1.Folder12;
DELETE LASTCHILD OF Cursor;
```

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Working with large XML messages" on page 2543
The tree representation of an XML message is typically bigger than the input bit stream. Manipulating a large message tree can require much storage but you can code ESQL statements that help to reduce the storage load on the broker.

Related reference:

"Syntax diagrams" on page 3677

"ESQL statements" on page 5067
You can use ESQL statements to manipulate message trees, update databases, or interact with nodes.

"CREATE statement" on page 5082
The CREATE statement creates a new message field.

DETACH statement:

The DETACH statement detaches a portion of a message tree without deleting it. This portion can be reattached using the ATTACH statement.

Syntax

For information about dynamic references, see “Creating dynamic field references” on page 2431.

For an example of DETACH, see the example in “ATTACH statement” on page 5069.

Related concepts:

“ESQL overview” on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

“Developing ESQL” on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

“Syntax diagrams” on page 3677

“ESQL statements” on page 5067
You can use ESQL statements to manipulate message trees, update databases, or interact with nodes.

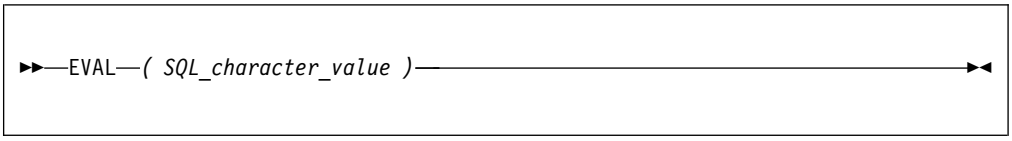
“ATTACH statement” on page 5069
The ATTACH statement attaches a portion of a message tree into a new position in the message hierarchy.

EVAL statement:

The EVAL statement takes a character value, interprets it as an SQL statement, and processes that statement.

For details of the EVAL function, see “EVAL function” on page 5294.

Syntax



EVAL takes one parameter in the form of an expression, evaluates this expression, and casts the resulting value to a character string if it is not one already. The expression that is passed to EVAL must therefore be able to be represented as a character string.

After this first stage evaluation is complete, the behavior of EVAL depends on whether it is being used as a complete ESQL statement, or in place of an expression that forms part of an ESQL statement:

- If it is a complete ESQL statement, the character string derived from the first stage evaluation is processed as if it were an ESQL statement.
- If it is an expression that forms part of an ESQL statement, the character string is evaluated as if it were an ESQL expression and EVAL returns the result.

User defined procedures cannot be defined within an EVAL statement but EVAL can be used to call a user-defined procedure that is in scope where the EVAL statement is used.

If you use the EVAL statement to call out to a user-defined procedure that is not called from anywhere else in the ESQL for a given node, you need to add the following code to your ESQL, to ensure that the user-defined procedure being called is included when the code is compiled:

```
IF (FALSE) THEN CALL procedure(<parameters>); END IF;
```

Note, that in the preceding code, you must replace `procedure()` with the named procedure in question.

In the following examples, A and B are integer scalar variables, and scalarVar1 and OperatorAsString are character string scalar variables.

The following examples are valid uses of EVAL:

- `SET OutputRoot.XMLNS.Data.Result = EVAL(A+B);`

  The expression A+B is acceptable because, although it returns an integer value, integer values are representable as character strings, and the necessary cast is performed before EVAL continues with its second stage of evaluation.
- `SET OutputRoot.XMLNS.Data.Result  = EVAL('A' || operatorAsString || 'B');`
- `EVAL('SET ' || scalarVar1 || ' = 2;');`

  The semicolon included at the end of the final string literal is necessary, because if EVAL is being used in place of an ESQL statement, its first stage evaluation must return a string that represents a valid ESQL statement, including the terminating semicolon.

Variables declared in an EVAL statement do not exist outside that EVAL statement.

The real power of EVAL is that it allows you to dynamically construct ESQL statements or expressions. In the second and third examples above, the value of scalarVar1 or operatorAsString can be set according to the value of an incoming message field, or other dynamic value, allowing you to effectively control what ESQL is processed without requiring a potentially lengthy IF-THEN ladder.

However, consider the performance implications in using EVAL. Dynamic construction and processing of statements or expressions is necessarily more time-consuming than simply processing pre-constructed ones. If performance is vital, you might prefer to write more specific, but faster, ESQL.

The following are not valid uses of EVAL:

- `SET EVAL(scalarVar1) = 2;`

  In this example, EVAL is being used to replace a field reference, not an expression.
- `SET OutputRoot.XMLNS.Data.Result[] = EVAL((SELECT T.x FROM Database.y AS T));`

  In this example, the `(SELECT T.x FROM Database.y)` passed to EVAL returns a list, which is not representable as a character string.

The following example is acceptable because `(SELECT T.x FROM Database.y AS T)` is a character string literal, not an expression in itself, and therefore is representable as a character string.

```
SET OutputRoot.XMLNS.Data.Result[]
 = EVAL('(SELECT T.x FROM Database.y AS T)');
```

Functions that are referenced only in an EVAL statement, and not in the rest of the ESQL module, might not be included in the BAR file. In the following examples, the function MyFunction must be referenced somewhere else in the ESQL module, otherwise the BAR file might fail to deploy.

```
EVAL('CALL MyFunction(parm1, parm2);');
```

```
DECLARE functionName CHARACTER 'Function';
DECLARE callStmt CHARACTER 'CALL My' || functionName || '(parm1, parm2);';
EVAL(callStmt);
```

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"Syntax diagrams" on page 3677

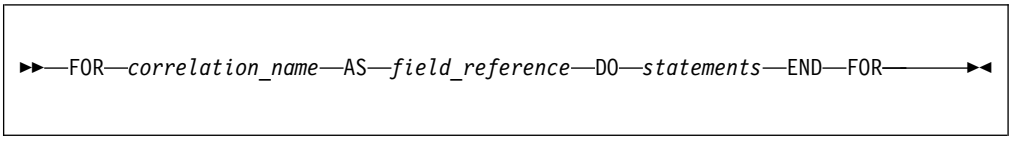
"ESQL statements" on page 5067
You can use ESQL statements to manipulate message trees, update databases, or interact with nodes.

"EVAL function" on page 5294
The EVAL function takes a character value and interprets that value as an ESQL expression that returns a value.

FOR statement:

The FOR statement iterates through a list (for example, a message array).

Syntax



For each iteration, the FOR statement makes the correlation variable (*correlation_name* in the syntax diagram) equal to the current member of the list (*field_reference*), then executes the block of statements. The advantage of the FOR statement is that it iterates through a list without your having to write any sort of loop construct (and eliminates the possibility of infinite loops).

For example the following ESQL:

```
SET OutputRoot.MQMD=InputRoot.MQMD;

SET Environment.SourceData.Folder[1].Field1 = 'Field11Value';
SET Environment.SourceData.Folder[1].Field2 = 'Field12Value';
SET Environment.SourceData.Folder[2].Field1 = 'Field21Value';
SET Environment.SourceData.Folder[2].Field2 = 'Field22Value';

DECLARE i INTEGER 1;
FOR source AS Environment.SourceData.Folder[] DO
     CREATE LASTCHILD OF OutputRoot.XMLNSC.Data.ResultData.MessageArrayTest.Folder[i]
            NAME 'FieldA' VALUE '\' || source.Field1 || '\' || CAST(i AS CHAR);
```

```
    CREATE LASTCHILD OF OutputRoot.XMLNSC.Data.ResultData.MessageArrayTest.Folder[i]
           NAME 'FieldB' VALUE '\' || source.Field2 || '\' || CAST(i AS CHAR);
    SET i = i + 1;
END FOR;
```

generates the output message:

```
<Data>
 <ResultData>
  <MessageArrayTest>
   <Folder>
    <FieldA>Field11Value\1</FieldA>
    <FieldB>Field12Value\1</FieldB>
   </Folder>
   <Folder>
    <FieldA>Field21Value\2</FieldA>
    <FieldB>Field22Value\2</FieldB>
   </Folder>
  </MessageArrayTest>
 </ResultData>
</Data>
```

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

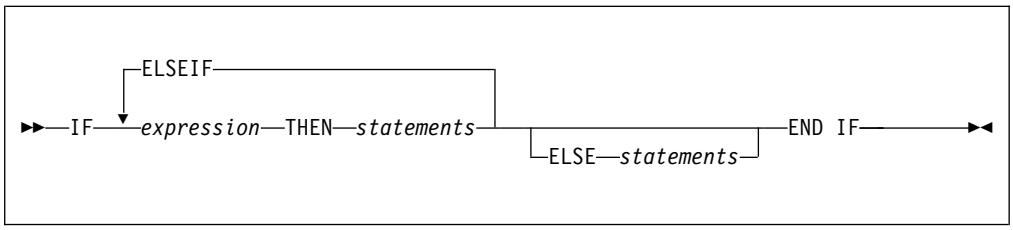
"Syntax diagrams" on page 3677

"ESQL statements" on page 5067
You can use ESQL statements to manipulate message trees, update databases, or interact with nodes.

IF statement:

The IF statement executes one set of statements based on the result of evaluating condition expressions.

Syntax



Each expression is evaluated in turn until one results in TRUE; the corresponding set of statements is then executed. If none of the expressions returns TRUE, and the optional ELSE clause is present, the ELSE clause's statements are executed.

UNKNOWN and FALSE are treated the same: the next condition expression is evaluated. ELSEIF is one word with no space between the ELSE and the IF. However, you can nest an IF statement within an ELSE clause: if you do, you can terminate both statements with END IF.

Example

```
IF i = 0 THEN
   SET size = 'small';
ELSEIF i = 1 THEN
   SET size = 'medium';
ELSEIF j = 4 THEN
   SET size = 'large';
ELSE
   SET size = 'unknown';
END IF;
```

```
IF J> MAX THEN
  SET J = MAX;
  SET Limit = TRUE;
END IF;
```

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Accessing elements in the message body" on page 2420
When you want to access the contents of a message, for reading or writing, use the structure and arrangement of the elements in the tree that is created by the parser from the input bit stream.

Related reference:

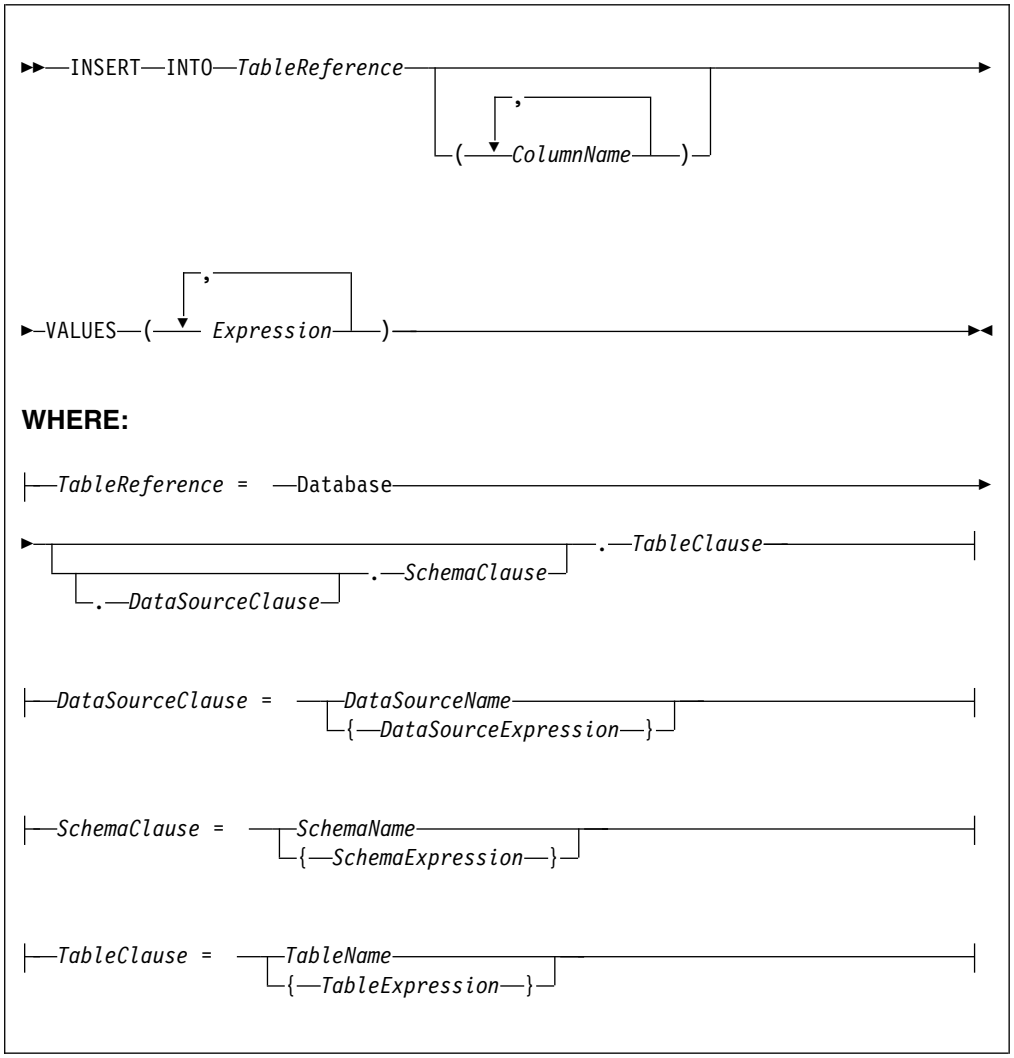
"Syntax diagrams" on page 3677

"ESQL statements" on page 5067
You can use ESQL statements to manipulate message trees, update databases, or interact with nodes.

INSERT statement:

The INSERT statement inserts a row into a database table.

Syntax



A single row is inserted into the table identified by *TableReference*. The *ColumnName* list identifies those columns in the target table that are to be given specific values. These values are determined by the expressions within the VALUES clause (the first expression gives the value of the first named column, and so on). The number of expressions in the VALUES clause must be the same as the number of named columns. Any columns present in the table but not mentioned in the list are given their default values.

Table reference

A table reference is a special case of the field references that are used to refer to message trees. It always starts with the word "Database" and can contain any of the following elements:

- A table name only
- A schema name and a table name
- A data source name (that is, the name of a database instance), a schema name, and a table name

In each case, the name can be specified directly or by an expression enclosed in braces ({...}). A directly-specified data source, schema, or table name is subject to

name substitution. That is, if the name used has been declared to be a known name, the value of the declared name is used rather than the name itself (see "DECLARE statement" on page 5117).

If a schema name is not specified, the default schema for the broker's database user is used.

If a data source name is not specified, the database pointed to by the node's `data source` attribute is used.

Handling errors

Errors can occur during insert operations. For example, the database might not be operational, or the table might have constraints defined that the new row would violate. In these cases, an exception is thrown, unless you have cleared the node property Throw Exception on Database Error. These exceptions set appropriate values for the following items, and can be dealt with by error handlers (see the DECLARE HANDLER statement):

- SQL code
- State
- Native error
- Error text

For further information about handling database errors, see "Capturing database state" on page 2512.

Examples

The following example assumes that the Data Source property of the Database node has been configured, and that the database it identifies has a table called TABLE1 with columns A, B, and C.

Given a message with the following generic XML body:

```
<A>
 <B>1</B>
 <C>2</C>
 <D>3</D>
</A>
```

The following INSERT statement inserts a new row into the table with the values 1, 2, and 3 for the columns A, B, and C:

```
INSERT INTO Database.TABLE1(A, B, C) VALUES (Body.A.B, Body.A.C, Body.A.D);
```

The next example shows the use of calculated data source, schema, and table names:

```
-- Declare variables to hold the data source, schema, and table names
-- and set their default values
DECLARE Source CHARACTER 'Production';
DECLARE Schema CHARACTER 'db2admin';
DECLARE Table  CHARACTER 'DynamicTable1';

-- Code which calculates their actual values comes here

-- Insert the data into the table
INSERT INTO Database.{Source}.{Schema}.{Table} (Name, Value) values ('Joe', 12.34);
```

Inserting a bit stream into a database

If the database column into which you want to insert data is set to a binary data type such as BLOB, the input message must be represented in bitstream form. If the input message is in the BLOB domain, use the following ESQL code:

```
DECLARE msgBitStream BLOB InputRoot.BLOB.BLOB;
INSERT INTO Database.TABLE1(MSGDATA) VALUES (msgBitStream);
```

Alternatively, if the input message is in an XML domain such as XMLNS, then the message tree must be serialized before the INSERT statement. To serialize the message tree and insert the contents into the database, use the following ESQL code:

```
DECLARE propRef REFERENCE TO InputRoot.Properties;
DECLARE msgBitStream BLOB ASBITSTREAM(InputRoot.XMLNS, propRef.Encoding, propRef.CodedCharSetId);
INSERT INTO Database.TABLE1(MSGDATA) VALUES (msgBitStream);
```

If the input messages received by your message flow come from different code pages, the CodedCharSetID and Encoding information is lost if you use the previous example. To capture CodedCharSetID and Encoding information, you can extend the table with two numeric columns to store the CodedCharSetID and Encoding data. To extend the table, modify the ESQL from the previous example to insert the CodedCharSetID and Encoding data into separate database columns:

```
DECLARE propRef REFERENCE TO InputRoot.Properties;
DECLARE inCCSID INT propRef.CodedCharSetId;
DECLARE inEncoding INT propRef.Encoding;
DECLARE msgBitStream BLOB ASBITSTREAM(InputRoot.XMLNS, inEncoding, inCCSID);
INSERT INTO Database.TABLE1(MSGDATA, MSGENCODING, MSGCCSID) VALUES
(msgBitStream, inEncoding, inCCSID);
```

As an extension to the previous example, if you require the entire message to be stored along with its MQMD header, and use it later for reconstructing the entire message in another message flow on a different platform using a different code page and encoding, the database table can be extended to hold all the numeric fields of the MQMD header.

For example, a message flow running on AIX inserts a message bit stream into the database table and another message flow running on Windows retrieves it and attempts to reconstruct the message along with the stored MQMD header.

The following set of numeric fields are contained in the MQMD header:

```
BackoutCount (MQLONG)
CodedCharSetId (MQLONG)
Encoding (MQLONG)
Expiry (MQLONG)
Feedback (MQLONG)
MsgFlags (MQLONG)
MsgSeqNumber (MQLONG)
MsgType (MQLONG)
Offset (MQLONG)
OriginalLength (MQLONG)
Persistence (MQLONG)
Priority (MQLONG)
PutApplType (MQLONG)
Report (MQLONG)
Version (MQLONG)
```

The following example uses CodedCharSetID, Encoding, Priority, and MsgSeqNumber:

```
DECLARE propRef REFERENCE TO InputRoot.Properties;
DECLARE mqmdRef REFERENCE TO InputRoot.MQMD;
DECLARE inCCSID INT propRef.CodedCharSetId;
```

```
DECLARE inEncoding INT propRef.Encoding;

DECLARE inPriority INT mqmdRef.Priority;
DECLARE inMsgSeqNumber INT mqmdRef.MsgSeqNumber;

DECLARE msgBitStream BLOB ASBITSTREAM(InputRoot, inEncoding, inCCSID);

INSERT INTO Database.TABLE1(MSGDATA, MSGENCODING, MSGCCSID, MSGPRIORITY,MSGSEQNUMBER)
VALUES (msgBitStream, inEncoding, inCCSID, inPriority, inMsgSeqNumber);
```

If you want to insert an XML message into a database column that has a CHAR or VARCHAR data type, the ESQL must be modified to convert the input message to the CHAR data type before the `INSERT` statement. In the following example, a `CAST` is used to transform the serialized message to the CHAR data type. The `CodedCharSetID` and `Encoding` data are inserted into separate database columns.

```
DECLARE propRef REFERENCE TO InputRoot.Properties;
DECLARE inCCSID INT propRef.CodedCharSetId;
DECLARE inEncoding INT propRef.Encoding;
DECLARE msgBitStream BLOB ASBITSTREAM(InputRoot.XMLNS, inEncoding, inCCSID);
DECLARE msgChar CHAR CAST(msgBitStream AS CHAR CCSID inCCSID);
INSERT INTO Database.TABLE1(MSGDATA, MSGENCODING, MSGCCSID) VALUES (msgChar, inEncoding, inCCSID);
```

For examples of how to extract a message bit stream from a database, based on the two previous examples, see “Selecting bitstream data from a database” on page 2493.

Related concepts:

“ESQL overview” on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

“Developing ESQL” on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

“Capturing database state” on page 2512
If an error occurs when the broker accesses an external database, you can either let the broker throw an exception during node processing or use ESQL statements to process the exception within the node itself.

Related reference:

“Syntax diagrams” on page 3677

“ESQL statements” on page 5067
You can use ESQL statements to manipulate message trees, update databases, or interact with nodes.

ITERATE statement:

The ITERATE statement stops the current iteration of the containing WHILE, REPEAT, LOOP, or BEGIN statement identified by Label.

The containing statement evaluates its loop condition (if any), and either starts the next iteration or stops looping, as the condition dictates.

Syntax

Example

In the following example, the loop iterates four times; that is the line identified by the comment Some statements 1 is passed through four times. However, the line identified by the comment Some statements 2 is passed through twice only because of the action of the IF and ITERATE statements. The ITERATE statement does **not** bypass testing the loop condition. Take particular care that the action of the ITERATE does not bypass the logic that makes the loop advance and eventually terminate. The loop count is incremented at the start of the loop in this example:

```
DECLARE i INTEGER;
SET i = 0;
X : REPEAT
  SET i = i + 1;

  -- Some statements 1

  IF i IN(2, 3) THEN
    ITERATE X;
  END IF;

  -- Some statements 2

UNTIL
  i>= 4
END REPEAT X;
```

ITERATE statements do not have to be directly contained by their labelled statement, making ITERATE statements particularly powerful.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"Syntax diagrams" on page 3677

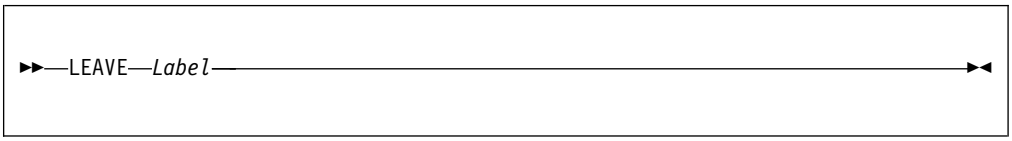
"ESQL statements" on page 5067
You can use ESQL statements to manipulate message trees, update databases, or interact with nodes.

LEAVE statement:

The LEAVE statement stops the current iteration of the containing WHILE, REPEAT, LOOP, or BEGIN statement identified by Label.

The containing statement's evaluation of its loop condition (if any) is bypassed and looping stops.

Syntax



Examples

In the following example, the loop iterates four times:

```
DECLARE i INTEGER;
SET i = 1;
X : REPEAT
  ...
  IF i>= 4 THEN
    LEAVE X;
  END IF;

  SET i = i + 1;
UNTIL
  FALSE
END REPEAT;
```

LEAVE statements do not have to be directly contained by their labelled statement, making LEAVE statements particularly powerful.

```
DECLARE i INTEGER;
SET i = 0;
X : REPEAT                    -- Outer loop
  ...
  DECLARE j INTEGER;
  SET j = 0;
  REPEAT                      -- Inner loop
    ...
    IF i>= 2 AND j = 1 THEN
      LEAVE X;                -- Outer loop left from within inner loop
    END IF;
    ...
    SET j = j + 1;
  UNTIL
    j>= 3
  END REPEAT;

  SET i = i + 1;
UNTIL
  i>= 3
END REPEAT X;
                              -- Execution resumes here after the leave
```

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

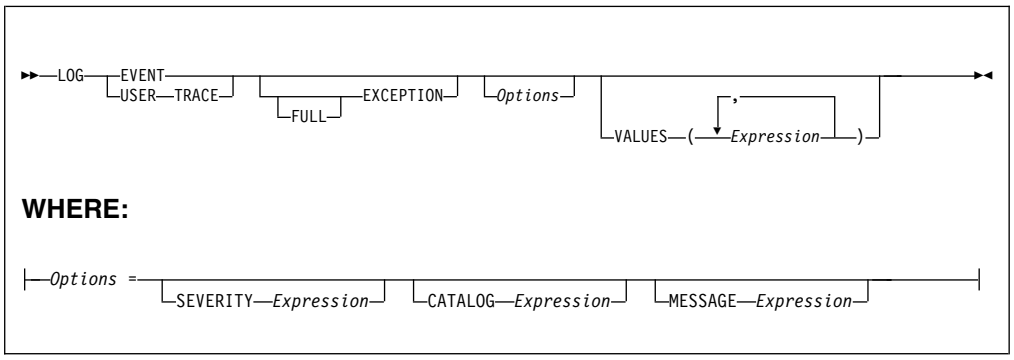
"Syntax diagrams" on page 3677

"ESQL statements" on page 5067
You can use ESQL statements to manipulate message trees, update databases, or interact with nodes.

LOG statement:

Use the LOG statement to write a record to the event log or to the user trace.

Syntax



CATALOG
CATALOG is an optional clause; if you omit it, CATALOG defaults to the WebSphere Message Broker current version catalog. To use the current version message catalog explicitly, use BIPmsgs on all operating systems.

EVENT
A record is written to the event log, and also to the user trace, if user tracing is enabled.

EXCEPTION
The current exception, if any, is logged.

For more information on exceptions, see "Errors and exception handling" on page 2973.

FULL
The complete nested exception report is logged, just as if the exception had reached the input node. If FULL is not specified, any wrapping exceptions are ignored, and only the original exception is logged. Therefore, you can have either a full report or simply the actual error report without the extra information regarding what was going on at the time. A current exception only exists within handler blocks (see "Handling errors in message flows" on page 2823).

MESSAGE
The number of the message to be used. If specified, the MESSAGE clause can contain any expression that returns a non-NULL, integer, value.

If you omit MESSAGE, its value defaults to the first message number (2951) in a block of messages that is provided for use by the LOG and THROW statements in the WebSphere Message Broker catalog. If you specify a message number, you can use message numbers 2951 through 2999. Alternatively, you can generate your own catalog.

SEVERITY
The severity associated with the message. If specified, the SEVERITY clause can contain any expression that returns a non-NULL, integer, value. If you omit the clause, its value defaults to 1.

USER TRACE
A record is written to the user trace, whether user trace is enabled or not.

VALUES
Use the optional VALUES clause to provide values for the data inserts in your message. You can insert any number of pieces of information, but the messages supplied (2951 - 2999) cater for a maximum of ten data inserts.

Note the general similarity of the LOG statement to the THROW statement.

```
-- Write a message to the event log specifying the severity, catalog and message
-- number. Four inserts are provided
LOG EVENT SEVERITY 1 CATALOG 'BIPmsgs' MESSAGE 2951 VALUES(1,2,3,4);

-- Write to the trace log whenever a divide by zero occurs
BEGIN
  DECLARE a INT 42;
  DECLARE b INT 0;
  DECLARE r INT;

  BEGIN
    DECLARE EXIT HANDLER FOR SQLSTATE LIKE 'S22012' BEGIN
      LOG USER TRACE EXCEPTION VALUES(SQLSTATE, 'DivideByZero');

      SET r = 0x7FFFFFFFFFFFFFFF;
    END;

    SET r = a / b;
  END;

  SET OutputRoot.XMLNS.Data.Result = r;
END;
```

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Creating message catalogs" on page 3138
Create your own message catalogs to write tailored entries to the local error log.

Related reference:

"Syntax diagrams" on page 3677

"ESQL statements" on page 5067
You can use ESQL statements to manipulate message trees, update databases, or interact with nodes.

"RETURN statement" on page 5155
The RETURN statement ends processing. What happens next depends on the programming context in which the RETURN statement is issued.

"THROW statement" on page 5161
Use the THROW statement to generate a user exception.

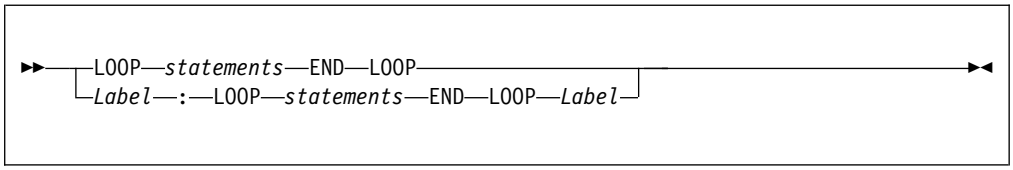
"Example message" on page 5311

LOOP statement:

The LOOP statement executes the sequence of statements repeatedly and unconditionally.

Ensure that the logic of the program provides some means of terminating the loop. You can use either LEAVE or RETURN statements.

Syntax



If present, *Label* gives the statement a name. This has no effect on the behavior of the LOOP statement, but allows *statements* to include ITERATE and LEAVE statements or other labelled statements, which in turn include ITERATE and LEAVE. The second *Label* can be present only if the first *Label* is present and, if it is, the labels must be identical.

Two or more labelled statements at the same level can have the same *Label* but this partly negates the advantage of the second *Label*. The advantage is that it unambiguously and accurately matches each END with its LOOP. However, a labelled statement within *statements* cannot have the same label, because this makes the behavior of the ITERATE and LEAVE statements ambiguous.

The LOOP statement is useful in cases where the required logic dictates that a loop is always exited part way through. This is because, in these cases, the testing of a loop condition that occurs in REPEAT or WHILE statements is both unnecessary and wasteful.

Example

```
DECLARE i INTEGER;
SET i = 1;
X : LOOP
  ...
  IF i>= 4 THEN
    LEAVE X;
  END IF;
  SET i = i + 1;
END LOOP X;
```

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

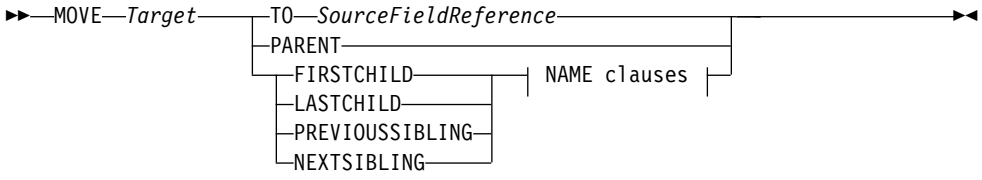
"Syntax diagrams" on page 3677

"ESQL statements" on page 5067
You can use ESQL statements to manipulate message trees, update databases, or interact with nodes.

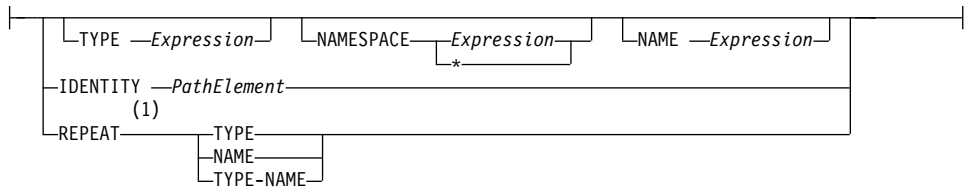
MOVE statement:

The MOVE statement changes the field to which the *Target* reference variable points.

Syntax



NAME clauses:



Notes:

1 The *RepeatClause* can be used only with the **PREVIOUSSIBLING** and **NEXTSIBLING** qualifiers.

If you include a TO clause, this clause changes the target reference to point to the same entity as that pointed to by source; this can either be a message field or a declared variable.

If you include a PARENT, PREVIOUSSIBLING, NEXTSIBLING, FIRSTCHILD, or LASTCHILD clause, the MOVE statement attempts to move the target reference variable in the direction specified relative to its current position. If any field exists in the given direction, the move succeeds. If there is no such field, the move fails; that is the reference variable continues to point to the same field or variable as before, and the LASTMOVE function returns false. You can use the LASTMOVE function to determine the success or failure of a move.

If a TYPE clause, NAME clause, or both are present, the target is again moved in the direction specified (PREVIOUSSIBLING or NEXTSIBLING, or FIRSTCHILD or LASTCHILD) but to a field with the given type, name, or both. This is particularly useful when the name or type (or both) of the target field is known, because this reduces the number of MOVE statements required to navigate to a field. This is because fields that do not match the criteria are skipped over; this can also include unexpected message tree fields, for example, those representing white space.

If the specified move cannot be made (that is, a field with the given type or name does not exist), the target remains unchanged and the LASTMOVE function returns false. The TYPE clause, NAME clause, or both clauses can contain any expression that returns a value of a suitable data type (INTEGER for type and CHARACTER for name). An exception is thrown if the value supplied is NULL.

Two further clauses, NAMESPACE and IDENTITY enhance the functionality of the NAME clause.

The NAMESPACE clause takes any expression that returns a non-null value of type character. It also takes an * indicating any namespace. Note that this cannot be confused with an expression because * is not a unary operator in ESQL.

The meaning depends on the presence of NAME and NAMESPACE clauses as follows:

NAMESPACE	NAME	Element located by...
No	No	Type, index, or both
No	Yes	Name in the default namespace
*	Yes	Name
Yes	No	Namespace
Yes	Yes	Name and namespace

The IDENTITY clause takes a single path element in place of the TYPE, NAMESPACE, and NAME clauses and follows all the rules described in the topic for field references (see "ESQL field reference overview" on page 5049).

When using MOVE with PREVIOUSSIBLING or NEXTSIBLING, you can specify REPEAT, TYPE, and NAME keywords that move the target to the previous or next field with the same type and name as the current field. The REPEAT keyword is particularly useful when moving to a sibling of the same kind, because you do not have to write expressions to define the type and name.

Example

```
MOVE cursor FIRSTCHILD TYPE Name NAME 'Field1';
```

This example moves the reference variable cursor to the first child field of the field to which the cursor is currently pointing, that has the type *Name* and the name *Field1*.

See "FIELDTYPE function" on page 5231 for a list of the types you can use.

The MOVE statement never creates new fields.

A common usage of the MOVE statement is to step from one instance of a repeating structure to the next. The fields within the structure can then be accessed by using a relative field reference. For example:

```
WHILE LASTMOVE(sourceCursor) DO
  SET targetCursor.ItemNumber  = sourceCursor.item;
  SET targetCursor.Description = sourceCursor.name;
  SET targetCursor.Price       = sourceCursor.prc;
  SET targetCursor.Tax         = sourceCursor.prc * 0.175;
```

```
  SET targetCursor.quantity    = 1;
  CREATE NEXTSIBLING OF targetCursor AS targetCursor REPEAT;
  MOVE sourceCursor NEXTSIBLING REPEAT TYPE NAME;
END WHILE;
```

For more information about reference variables, and an example of moving a reference variable, see “Creating dynamic field references” on page 2431.

Related concepts:

“ESQL overview” on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

“Developing ESQL” on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

“Creating dynamic field references” on page 2431
You can use a variable of type REFERENCE as a dynamic reference to navigate a message tree. This acts in a similar way to a message cursor or a variable pointer.

Related reference:

“Syntax diagrams” on page 3677

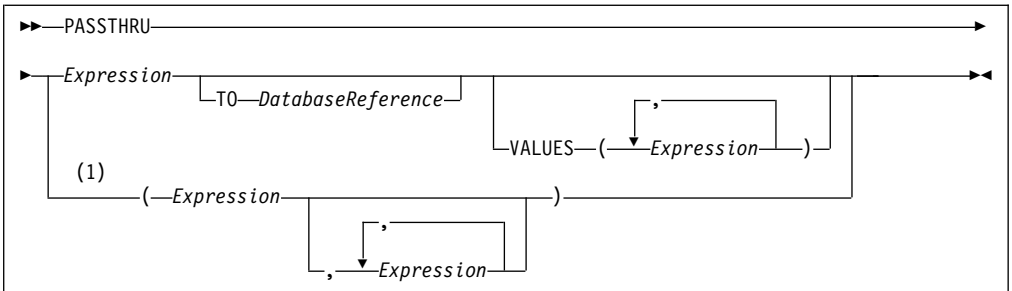
“ESQL statements” on page 5067
You can use ESQL statements to manipulate message trees, update databases, or interact with nodes.

“FIELDTYPE function” on page 5231
The FIELDTYPE field function returns the type of a given field.

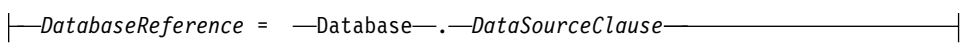
“LASTMOVE function” on page 5237

PASSTHRU statement:

The PASSTHRU statement evaluates an expression and runs the resulting character string as a database statement.



WHERE:



Notes:

1 The lower half of the main syntax diagram (the second of the two ways of coding the *Expression* to be passed to PASSTHRU) describes syntax retained for backward compatability.

Usage

The main use of the PASSTHRU statement is to issue administrative commands to databases (for example, to create a table).

Note: Do not use PASSTHRU to call stored procedures; instead, use the CALL statement because PASSTHRU imposes limitations (you cannot use output parameters, for example).

The first expression is evaluated and the resulting character string is passed to the database pointed to by *DatabaseReference* (in the TO clause) for execution. If the TO clause is not specified, the database pointed to by the node's `data source` attribute is used.

Use question marks (?) in the database string to denote parameters. The parameter values are supplied by the VALUES clause.

If the VALUES clause is specified, its expressions are evaluated and passed to the database as parameters; (that is, the expressions' values are substituted for the question marks in the database statement).

If only one VALUE expression exists, the result might or might not be a list. If it is a list, the list's scalar values are substituted sequentially for the question marks. If it is not a list, the single scalar value is substituted for the (single) question mark in the database statement. If more than one VALUE expression exists, none of the expressions evaluate to a list; their scalar values are substituted sequentially for the question marks instead.

Because the database statement is constructed by the user program, it is not essential to use parameter markers (that is, the question marks) or the VALUES clause, because the whole of the database statement could be supplied, as a literal string, by the program. However, use parameter markers whenever possible because this reduces the number of different statements that need to be prepared and stored in the database and the broker.

Database reference

A database reference is a special instance of the field references that is used to refer to message trees. It consists of the word **Database** followed by the name of a data source (that is, the name of a database instance).

You can specify the data source name directly or by an expression enclosed in braces ({...}). A directly-specified data source name is subject to name substitution. That is, if the name used has been declared to be a known name, the value of the declared name is used rather than the name itself (see "DECLARE statement" on page 5117).

If you have created a message flow that contains one of the following nodes, and the ESQL that is associated with this node includes a PASSTHRU statement and a database reference, you must specify a value for the `Data source` property of the relevant node:

- Compute
- Database
- Filter

Handling errors

It is possible for errors to occur during PASSTHRU operations. For example, the database might not be operational or the statement might be invalid. In these cases, an exception is thrown (unless the node has its `Throw exception on database error` property cleared). These exceptions set appropriate SQL code, state, native error, and error text values and can be dealt with by error handlers (see the DECLARE HANDLER statement).

For further information about handling database errors, see "Capturing database state" on page 2512.

Examples

The following example creates the table `Customers` in schema `Shop` in database DSN1:

```
PASSTHRU 'CREATE TABLE Shop.Customers (
  CustomerNumber INTEGER,
  FirstName      VARCHAR(256),
  LastName       VARCHAR(256),
  Street         VARCHAR(256),
  City           VARCHAR(256),
  Country        VARCHAR(256)
)' TO Database.DSN1;
```

If, as in the last example, the ESQL statement is specified as a string literal, you must put single quotation marks around it. If, however, it is specified as a variable, omit the quotation marks. For example:

```
SET myVar = 'SELECT * FROM user1.stocktable';
SET OutputRoot.XMLNS.Data[] = PASSTHRU(myVar);
```

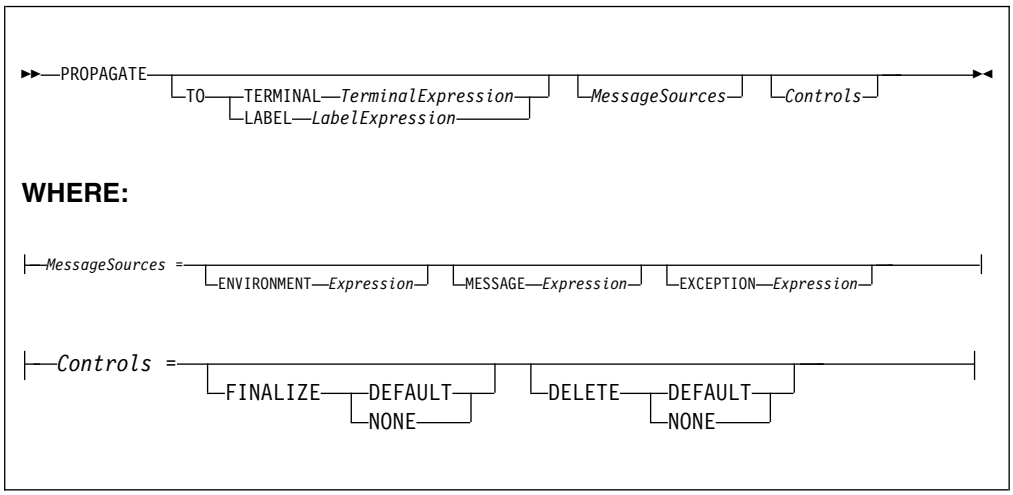
The following example "drops" (that is, deletes) the table `Customers` from schema `Shop` in database DSN1:

```
PASSTHRU 'DROP TABLE Shop.Customers' TO Database.DSN1;
```

PROPAGATE statement:

The PROPAGATE statement propagates a message to the downstream nodes.

Syntax



You can use the PROPAGATE statement in Compute and Database nodes, but not in Filter nodes. The additions to this statement assist in error handling - see "Coding ESQL to handle errors" on page 2506.

TO TERMINAL clause

If the TO TERMINAL clause is present, *TerminalExpression* is evaluated. If the result is of type CHARACTER, a message is propagated to a terminal according to the rule:

```
'nowhere' : no propagation
'failure' : Failure
 'out'    : Out
 'out1'   : Out1
 'out2'   : Out2
 'out3'   : Out3
 'out4'   : Out4
```

Tip: Terminal names are case sensitive so, for example, Out1 does not match any terminal.

If the result of *TerminalExpression* is of type INTEGER, a message is propagated to a terminal according to the rule:

```
-2 : no propagation
-1 : Failure
 0 : Out
 1 : Out1
 2 : Out2
 3 : Out3
 4 : Out4
```

If the result of *TerminalExpression* is neither a CHARACTER nor an INTEGER, the broker throws an exception.

If there is neither a TO TERMINAL nor a TO LABEL clause, the broker propagates a message to the Out terminal.

Tip: Using character values in terminal expressions leads to the most natural and readable code. Integer values, however, are easier to manipulate in loops and marginally faster.

TO LABEL clause

If the TO LABEL clause is present, *LabelExpression* is evaluated. If the result is of type CHARACTER **and** there is a Label node with a Label property that matches *LabelExpression*, in the same flow, the broker propagates a message to that node.

Tip: Labels, like terminals, are case sensitive. Also, note that, as with route to Label nodes, it is the Label Name property of the Label node that defines the target, not the node's label itself.

If the result of *LabelExpression* is NULL or not of type CHARACTER, or there is no matching Label node in the flow, the broker throws an exception.

If there is neither a TO TERMINAL nor a TO LABEL clause, the broker propagates a message to the out terminal.

MessageSources clauses

The MessageSources clauses select the message trees to be propagated. This clause is only applicable to the Compute node (it has no effect in the Database node).

The values that you can specify in MessageSources clauses are:

```
ENVIRONMENT :
  InputLocalEnvironment
  OutputLocalEnvironment

Message :
  InputRoot
  OutputRoot

ExceptionList :
  InputExceptionList
  OutputExceptionList
```

If there is no MessageSources clause, the node's Compute mode property is used to determine which messages are propagated.

FINALIZE clause

Finalization is a process that fixes header chains and makes the Properties folder match the headers. If present, the FINALIZE clause allows finalization to be controlled.

This clause is only applicable to the Compute node (it has no effect in a Database node).

The Compute node allows its output message to be changed by other nodes (by the other nodes changing their input message). However, a message created by a Compute node cannot be changed by another node after:

- It has been finalized
- It has reached any output or other node which generates a bit-stream

If FINALIZE is set to DEFAULT, or the FINALIZE clause is absent, the output message (but not the Environment, Local Environment or Exception List) is finalized before propagation.

If FINALIZE is set to NONE, no finalization takes place. This option is required if you want to preserve and allow updates of the entire output

message tree by the nodes downstream in the message flow and is used with DELETE NONE as described in the next section.

DELETE clause

The DELETE clause allows the clearing of the output local environment, message, and exception list to be controlled.

The DELETE clause is only applicable to the Compute node (it has no effect in a Database node).

If DELETE is set to DEFAULT, or the DELETE clause is absent, the output local environment, message, and exception list are all cleared and their memory recovered immediately after propagation.

If DELETE is set to NONE, nothing is cleared. Use DELETE NONE if you want the downstream nodes to be able to see a single instance of output local environment message, and exception list trees. Each propagate starts with the content of these trees as created by the previous propagate rather than starting with empty trees. If you also want these nodes to update the output tree, DELETE NONE must be used with the FINALIZE NONE option described in the previous section.

Note that the output trees that are finalized are cleared, regardless of which ones are propagated.

Propagation is a synchronous process. That is, the next statement is not executed until all the processing of the message in downstream nodes has completed. Be aware that this processing might throw exceptions and, if these exceptions are not caught, they will prevent the statement following the PROPAGATE call being reached. This behavior might be what the logic of your flow requires but, if it is not, you can use a handler to catch the exception and perform the necessary actions. Note that exceptions thrown downstream of a propagate, if not caught, will also prevent the final automatic actions of a Compute or Database node (for example, issuing a COMMIT Transaction set to Commit) from taking place.

If you are using the PROPAGATE statement in your node it is important that you use a `RETURN FALSE;` to prevent automatic propagation of the message to the next node in the message flow.

```
DECLARE i INTEGER 1;
DECLARE count INTEGER;
SET count = CARDINALITY(InputRoot.XMLNS.Invoice.Purchases."Item"[])

WHILE i <= count DO
  --use the default tooling-generated procedure for copying message headers
  CALL CopyMessageHeaders();
  SET OutputRoot.XMLNS.BookSold.Item = InputRoot.XMLNS.Invoice.Purchases.Item[i];
  PROPAGATE;
  SET i = i+1;
END WHILE;
RETURN FALSE;
```

The following messages are produced on the Out terminal by the PROPAGATE statement:

```
<BookSold>
 <Item>
  <Title Category="Computer" Form="Paperback" Edition="2">The XML Companion </Title>
  <ISBN>0201674866</ISBN>
  <Author>Neil Bradley</Author>
  <Publisher>Addison-Wesley</Publisher>
  <PublishDate>October 1999</PublishDate>
```

```
  <UnitPrice>27.95</UnitPrice>
  <Quantity>2</Quantity>
 </Item>
</BookSold>

<BookSold>
 <Item>
  <Title Category="Computer" Form="Paperback" Edition="2">A Complete Guide to
   DB2 Universal Database</Title>
  <ISBN>1558604820</ISBN>
  <Author>Don Chamberlin</Author>
  <Publisher>Morgan Kaufmann Publishers</Publisher>
  <PublishDate>April 1998</PublishDate>
  <UnitPrice>42.95</UnitPrice>
  <Quantity>1</Quantity>
 </Item>
</BookSold>

<BookSold>
 <Item>
  <Title Category="Computer" Form="Hardcover" Edition="0">JAVA 2 Developers
  Handbook</Title>
  <ISBN>0782121799</ISBN>
  <Author>Phillip Heller, Simon Roberts </Author>
  <Publisher>Sybex, Inc.</Publisher>
  <PublishDate>September 1998</PublishDate>  <UnitPrice>59.99</UnitPrice>
  <Quantity>1</Quantity>
 </Item>
</BookSold>
```

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Coding ESQL to handle errors" on page 2506
When you process messages in a message flow, errors can have a number of different causes and the message flow designer must decide how to handle those errors.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Generating multiple output messages" on page 2437
You can use the PROPAGATE statement to generate multiple output messages in the Compute node. The output messages that you generate can have the same or different content. You can also direct output messages to any of the four alternate output terminals of the Compute node, or to a Label node.

"Committing database updates" on page 2501
When you create a message flow that interacts with databases, you can choose whether the updates that you make are committed when the current node has completed processing, or when the current invocation of the message flow has terminated.

Related reference:

"Syntax diagrams" on page 3677

"ESQL statements" on page 5067
You can use ESQL statements to manipulate message trees, update databases, or interact with nodes.

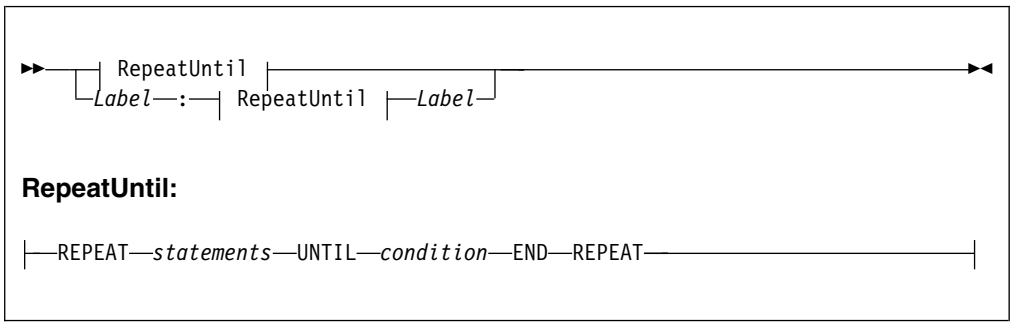
"RETURN statement" on page 5155
The RETURN statement ends processing. What happens next depends on the

programming context in which the RETURN statement is issued.
"Example message" on page 5311

REPEAT statement:

The REPEAT statement processes a sequence of statements, then evaluates the condition expression.

Syntax



The REPEAT statement repeats the steps until condition is TRUE. Ensure that the logic of the program is such that the loop terminates. If the condition evaluates to UNKNOWN, the loop does **not** terminate.

If present, the *Label* gives the statement a name. This has no effect on the behavior of the REPEAT statement, but allows statements to include ITERATE and LEAVE statements or other labelled statements, which in turn include ITERATE and LEAVE. The second *Label* can be present only if the first *Label* is present and, if it is, the labels must be identical. Two or more labelled statements at the same level can have the same label, but this partly negates the advantage of the second *Label*. The advantage is that it unambiguously and accurately matches each END with its REPEAT. However, a labelled statement within statements cannot have the same label because this makes the behavior of the ITERATE and LEAVE statements ambiguous.

Example

```
DECLARE i INTEGER;
SET i = 1;
X : REPEAT
  ...
  SET i = i + 1;
UNTIL
  i>= 3
END REPEAT X;
```

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

“Syntax diagrams” on page 3677

“ESQL statements” on page 5067
You can use ESQL statements to manipulate message trees, update databases, or interact with nodes.

RESIGNAL statement:

The RESIGNAL statement rethrows the current exception, if one exists.

Syntax

You can use RESIGNAL only in error handlers.

Typically, RESIGNAL is used when an error handler catches an exception that it cannot handle. The handler uses RESIGNAL to rethrow the original exception so that a handler in higher-level scope has the opportunity to handle it.

Because the handler throws the original exception, rather than a new (and therefore different) one:

1. The higher-level handler is not affected by the presence of the lower-level handler.
2. If no higher-level handler is present, you get a full error report in the event log.

Example

```
RESIGNAL;
```

Related concepts:

“ESQL overview” on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

“Developing ESQL” on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

“DECLARE HANDLER statement” on page 5124
The DECLARE HANDLER statement creates an error handler for handling exceptions.

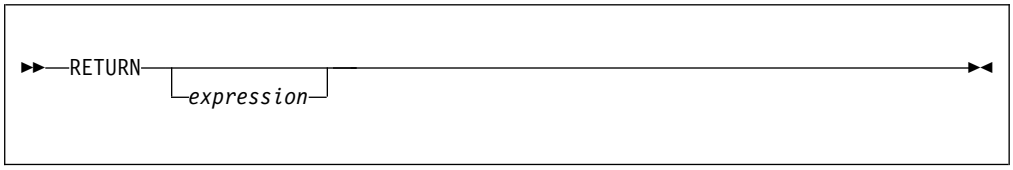
“Syntax diagrams” on page 3677

“ESQL statements” on page 5067
You can use ESQL statements to manipulate message trees, update databases, or interact with nodes.

RETURN statement:

The RETURN statement ends processing. What happens next depends on the programming context in which the RETURN statement is issued.

Syntax



Main Function

When used in the Main function, the RETURN statement stops processing of the module and returns control to the next node in a message flow. In the Main function the return statement must contain an *expression* of BOOLEAN type. The behavior of the RETURN statement in the Main function is dependant on the node. In the Compute node for example, if *expression* is anything other than TRUE, propagation of the message is stopped. In the Filter node, however, the message is propagated to the terminal matching the value of *expression*: TRUE, FALSE and UNKNOWN. The following table describes the differences between the RETURN statement when used in the Main function of the Compute, Filter, and Database nodes.

Node	RETURN TRUE;	RETURN FALSE;	RETURN UNKNOWN (if BOOLEAN type) or RETURN NULL;	RETURN;
Compute	Propagate message to Out terminal.	Stop propagation	Stop propagation	Deploy failure (BIP2912E: Type mismatch on RETURN)
Database	Propagate message to Out terminal.	Stop propagation	Stop propagation	Deploy failure (BIP2912E: Type mismatch on RETURN)
Filter	Propagate message to True terminal	Propagate message to False terminal	Propagate message to Unknown terminal	Deploy failure (BIP2912E: Type mismatch on RETURN)

User defined functions and procedures

When used in a function or a procedure, the RETURN statement stops processing of that function and returns control to the calling expression. The *expression*, which must be present if the function or procedure has been declared with a RETURNS clause, is evaluated and acts as the return value of the function. The data type of the returned value must be the same as that in the function's declaration. The following table describes the differences between the RETURN statement when used in user defined functions and procedures.

	RETURN *expression*;	**RETURN NULL; (or return** *expression* **that evaluates to NULL)**	**RETURN;**	**No RETURN statement**
User defined function or procedure with a RETURNS clause	Returns control to the calling expression with the value of *expression*	Returns control to the calling expression with NULL	Deploy failure (BIP2912E: Type mismatch on RETURN)	Returns control to the calling expression with NULL after all the statements in the function or procedure have been run
User defined function or procedure without a RETURNS clause	Deploy failure (BIP2401E: Syntax error: expected ; but found *expression*)	Deploy failure (BIP2401E: Syntax error: expected ; but found NULL)	Returns control to the calling expression	Returns control to the calling expression after all the statements in the function or procedure have been run

The RETURN statement must be used within the body of a function or procedure that has the RETURNS statement in its declaration. This function can be invoked using the CALL ... INTO statement. The RETURNS statement provides the datatype that the function or procedure returns to the "CALL statement" on page 5077. The CALL ... INTO statement specifies the variable to which the return value is assigned. The example in this topic shows an example of how a RETURNS and CALL ... INTO statement are used together to assign the return statement. If you use the CALL ... INTO statement to call a function or procedure that does not have a RETURNS statement declared, a BIP2912E error message is generated.

Example

The following example, which is based on "Example message" on page 5311, illustrates how the RETURN, RETURNS and CALL...INTO statements can be used:

```
CREATE FILTER MODULE ProcessOrder
 CREATE FUNCTION Main() RETURNS BOOLEAN
  BEGIN

      DECLARE SpecialOrder  BOOLEAN;
      SET  OutputRoot.MQMD = InputRoot.MQMD;
      CALL IsBulkOrder(InputRoot.XMLNS.Invoice.Purchases) INTO SpecialOrder;

      --
      --  more processing could be inserted here
      --  before routing the order to the appropriate terminal
      --
      RETURN SpecialOrder;

  END;

 CREATE FUNCTION IsBulkOrder (P1 REFERENCE)
        RETURNS BOOLEAN
  BEGIN
      -- Declare and initialize variables--
      DECLARE a INT 1;
      DECLARE PriceTotal FLOAT 0.0;
      DECLARE NumItems INT 0;
```

```
    DECLARE iroot REFERENCE TO P1;

    -- Calculate value of order, however if this is a bulk purchase, the          --
    -- order will need to be handled differently (discount given) so return TRUE  --
    -- or FALSE depending on the size of the order                                 --
    WHILE a <= CARDINALITY(iroot.Item[]) DO
           SET  NumItems = NumItems + iroot.Item[a].Quantity;
           SET  PriceTotal = PriceTotal + iroot.Item[a].UnitPrice;
           SET  a = a + 1;
    END WHILE;
    RETURN (PriceTotal/NumItems> 42);

END;

END MODULE;
```

In the example, if the average price of items is greater than 42, TRUE is returned; otherwise FALSE is returned. Thus, a Filter node could route messages describing expensive items down a different path from messages describing inexpensive items. From the example, the `CALL IsBulkOrder(InputRoot.XMLNS.Invoice.Purchases) INTO SpecialOrder;` statement can also be written as `SpecialOrder = IsBulkOrder(InputRoot.XMLNS.Invoice.Purchases);`

If you are using the PROPAGATE statement in your node it is important that you use a `RETURN FALSE;` to prevent automatic propagation of the message to the next node in the message flow. See "PROPAGATE statement" on page 5150 for an example of preventing the implicit propagate at the end of processing in a Compute node.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Using the CALL statement to call a user-written routine" on page 2623
The ESQL CALL statement calls routines that have been created and implemented in different ways.

Related reference:

"CREATE PROCEDURE statement" on page 5103
The CREATE PROCEDURE statement defines a callable function or procedure.

"CREATE FUNCTION statement" on page 5091
The CREATE FUNCTION statement defines a callable function or procedure.

"Compute node" on page 4340
Use the Compute node to construct one or more new output messages.

"Filter node" on page 4452
Use the Filter node to route a message according to message content.

"Syntax diagrams" on page 3677

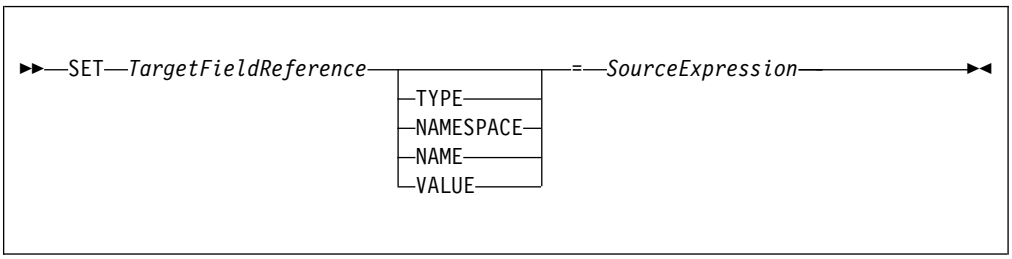
"ESQL statements" on page 5067
You can use ESQL statements to manipulate message trees, update databases, or interact with nodes.

"PROPAGATE statement" on page 5150
The PROPAGATE statement propagates a message to the downstream nodes.
"Example message" on page 5311

SET statement:

The SET statement assigns a value to a variable.

Syntax



Introduction

TargetFieldReference identifies the target of the assignment. The target can be any of the following:
- A declared scalar variable
- A declared row variable
- One of the predefined row variables (for example, *InputRoot*)
- A field within any kind of row variable (that is, a sub tree or conceptual row)
- A list of fields within any kind of row variable (that is, a conceptual list)
- A declared reference variable that points to any of the above

The target cannot be any kind of database entity.

SourceExpression is an expression that supplies the value to be assigned. It can be any kind of expression and can return a scalar, row or list value.

Assignment to scalar variables

If the target is a declared scalar variable, *SourceExpression* is evaluated and assigned to the variable. If need be, its value is converted to the data type of the variable. If this conversion is not possible, there will be either an error at deploy time or an exception at run time.

Null values are handled in exactly the same way as any other value. That is, if the expression evaluates to null, the value "null" is assigned to the variable.

For scalar variables the TYPE, NAME, NAMESPACE, and VALUE clauses are meaningless and are not allowed.

Assignment to rows, lists, and fields

If the target is a declared row variable, one of the predefined row variables, a field within any kind of row variable, a list of fields within any kind of row variable, or a declared reference variable that points to any of these things, the ultimate target is a field. In these cases, the target field is navigated to (creating the fields if necessary).

If array indices are used in *TargetFieldReference*, the navigation to the target field can only create fields on the direct path from the root to the target field. For example, the following SET statement requires that at least one instance of `Structure` already exists in the message:

```
SET OutputRoot.XMLNS.Message.Structure[2].Field = ...
```

The target field's value is set according to a set of rules, based on:

1. The presence or absence of the TYPE, NAME, NAMESPACE, or VALUE clauses
2. The data type returned by the source expression

1. If no TYPE, NAME, NAMESPACE, or VALUE clause is present (which is the most common case) the outcome depends on whether *SourceExpression* evaluates to a scalar, a row, or a list:
   - If *SourceExpression* evaluates to a scalar, the value of the target field is set to the value returned by *SourceExpression*, except that, if the result is null, the target field is discarded. Note that the new value of the field might not be of the same data type as its previous value.
   - If *SourceExpression* evaluates to a row:
     a. The target field is identified.
     b. The target field's value is set.
     c. The target field's child fields are replaced by a new set, dictated by the structure and content of the list.
   - If SourceExpression evaluates to a list:
     a. The set of target fields in the target tree are identified.
     b. If there are too few target fields, more are created; if there are too many, the extra ones are removed.
     c. The target fields' values are set.
     d. The target fields' child fields are replaced by a new set, dictated by the structure and content of the list.

   For further information on working with elements of type `list` see "Working with elements of type list" on page 2448
2. If a TYPE clause is present, the type of the target field is set to the value returned by *SourceExpression*. An exception is thrown if the returned value is not scalar, is not of type INTEGER, or is NULL.
3. If a NAMESPACE clause is present, the namespace of the target field is set to the value returned by *SourceExpression*. An exception is thrown if the returned value is not scalar, is not of type CHARACTER, or is NULL.
4. If a NAME clause is present, the name of the target field is set to the value returned by *SourceExpression*. An exception is thrown if the returned value is not scalar, is not of type CHARACTER, or is NULL.
5. If a VALUE clause is present, the value of the target field is changed to that returned by *SourceExpression*. An exception is thrown if the returned value is not scalar.

Notes

SET statements are particularly useful in Compute nodes that modify a message, either changing a field or adding a new field to the original message. SET statements are also useful in Filter and Database nodes, to set declared variables or the fields in the Environment tree or Local Environment trees. You can use statements such as the following in a Compute node that modifies a message:

```
SET OutputRoot = InputRoot;
SET OutputRoot.XMLNS.Order.Name = UPPER(InputRoot.XMLNS.Order.Name);
```

This example puts one field in the message into uppercase. The first statement constructs an output message that is a complete copy of the input message. The second statement sets the value of the `Order.Name` field to a new value, as defined by the expression on the right.

If the `Order.Name` field does not exist in the original input message, it does not exist in the output message generated by the first statement. The expression on the right of the second statement returns NULL (because the field referenced inside the UPPER function call does not exist). Assigning the NULL value to a field has the effect of deleting it if it already exists, and so the effect is that the second statement has no effect.

If you want to assign a NULL value to a field without deleting the field, use a statement like this:

```
SET OutputRoot.XMLNS.Order.Name VALUE = NULL;
```

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Accessing elements in the message body" on page 2420
When you want to access the contents of a message, for reading or writing, use the structure and arrangement of the elements in the tree that is created by the parser from the input bit stream.

Related reference:

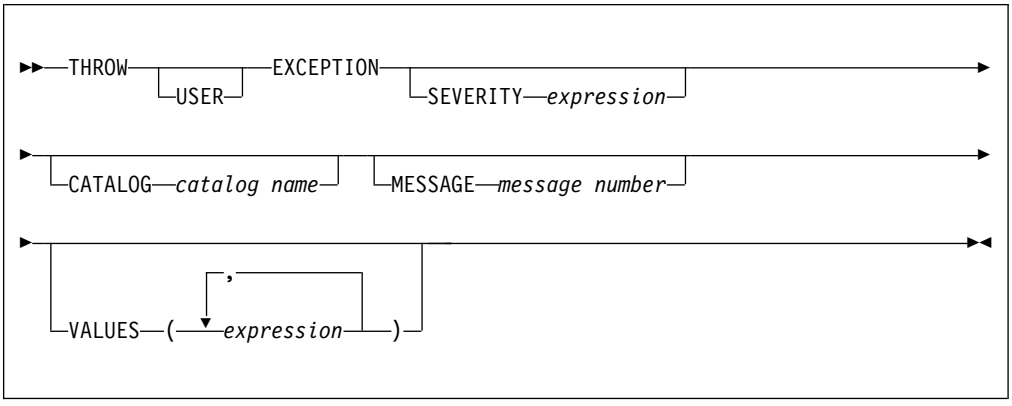
"Syntax diagrams" on page 3677

"ESQL statements" on page 5067
You can use ESQL statements to manipulate message trees, update databases, or interact with nodes.

THROW statement:

Use the THROW statement to generate a user exception.

Syntax



The USER keyword indicates the type of exception being thrown. (Currently, only USER exceptions are supported, and if you omit the USER keyword the exception defaults to a USER exception anyway.) Specify the USER keyword, even though it currently has no effect, for the following reasons:

- If future broker releases support other types of exception, and the default type changes, your code will not need to be changed.
- It makes it clear that this is a user exception.

SEVERITY is an optional clause that determines the severity associated with the exception. The clause can contain any expression that returns a non-NULL, integer value. If you omit the clause, it defaults to 1.

On Windows you must set SEVERITY to 3, so that the Windows event log reports the error correctly.

CATALOG is an optional clause; if you omit it, CATALOG defaults to the WebSphere Message Broker current version catalog. To use the current version message catalog explicitly, use BIPmsgs on all operating systems.

MESSAGE is an optional clause; if you omit it, it defaults to the first message number of the block of messages provided for using THROW statements in the default catalog (2951). If you enter a message number in the THROW statement, you can use message numbers 2951 to 2999 from the default catalog. Alternatively, you can generate your own catalog by following the instructions in "Creating message catalogs" on page 3138.

Use the optional VALUES field to insert data into your message. You can insert any number of pieces of information, but the messages supplied (2951 - 2999) cater for eight inserts only.

Examples

Here are some examples of how you might use a THROW statement:

- ```
  THROW USER EXCEPTION;
  ```
- ```
  THROW USER EXCEPTION CATALOG 'BIPmsgs' MESSAGE
  2951 VALUES(1,2,3,4,5,6,7,8) ;
  ```
- ```
  THROW USER EXCEPTION CATALOG 'BIPmsgs' MESSAGE
  2951 VALUES('The SQL State: ',
      SQLSTATE, 'The SQL Code: ', SQLCODE, 'The SQLNATIVEERROR: ', SQLNATIVEERROR,
      'The SQL Error Text: ', SQLERRORTEXT ) ;
  ```
- ```
  THROW USER EXCEPTION CATALOG 'BIPmsgs' MESSAGE
  2951 ;
  ```
- ```
  THROW USER EXCEPTION CATALOG 'MyCatalog' MESSAGE
  2951 VALUES('Hello World') ;
  ```
- ```
  THROW USER EXCEPTION MESSAGE
  2951 VALUES('Insert text 1', 'Insert text 2') ;
  ```

For more information about how to throw an exception, and details of `SQLSTATE`, `SQLCODE`, `SQLNATIVEERROR`, and `SQLERRORTEXT`, see "ESQL database state functions" on page 5168.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Throwing an exception" on page 2511
If you detect an error or other situation in your message flow in which you want message processing to be ended, you can throw an exception in a message flow in two ways.

"Creating message catalogs" on page 3138
Create your own message catalogs to write tailored entries to the local error log.

Related reference:

"Syntax diagrams" on page 3677

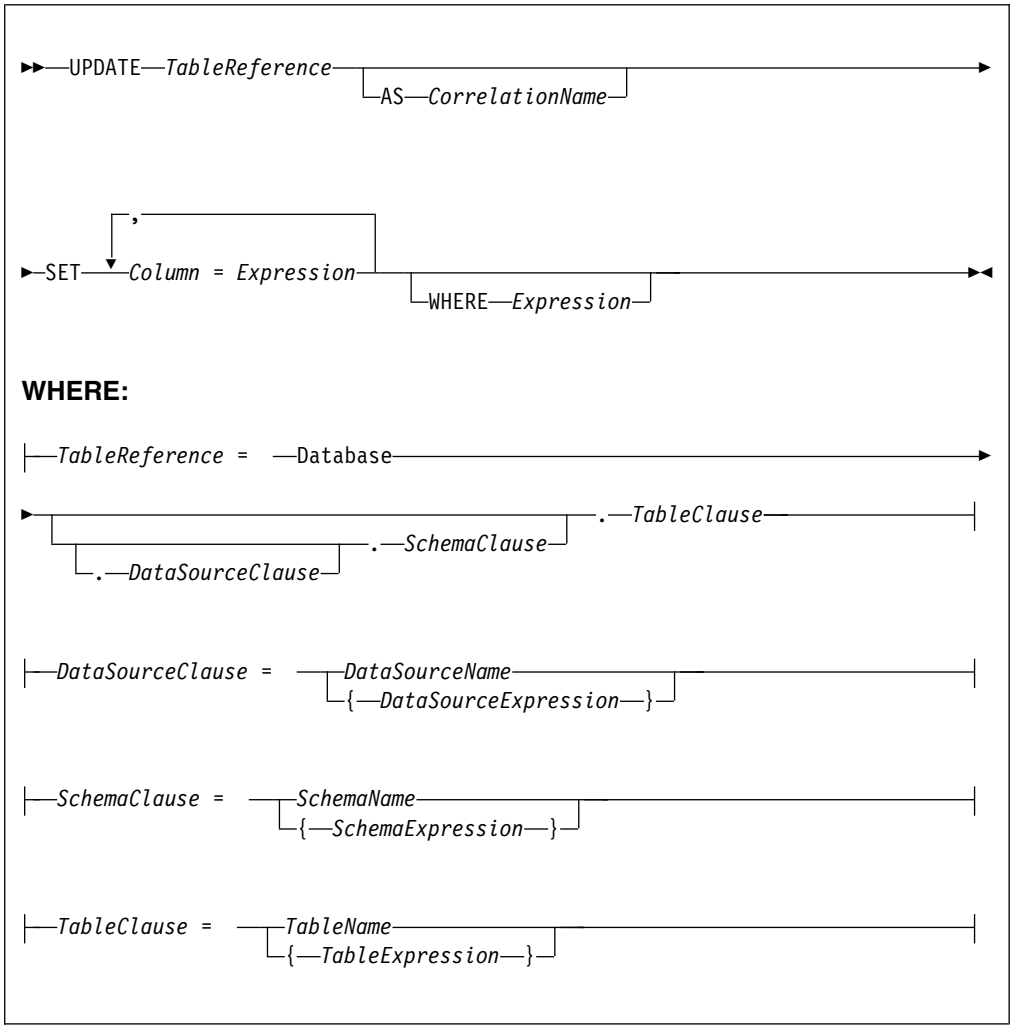
"ESQL statements" on page 5067
You can use ESQL statements to manipulate message trees, update databases, or interact with nodes.

"ESQL database state functions" on page 5168

UPDATE statement:

The UPDATE statement changes the values of specified columns, in selected rows, in a table in an external database.

Syntax



All rows for which the WHERE clause expression evaluates to TRUE are updated in the table identified by *TableReference*. Each row is examined in turn and a variable is set to point to the current row. Typically, the WHERE clause expression uses this variable to access column values and thus cause rows to be updated, or retained unchanged, according to their contents. The variable is referred to by *CorrelationName* or, in the absence of an AS clause, by *TableName*. When a row has been selected for updating, each column named in the SET clause is given a new value as determined by the corresponding expression. These expressions can, if you want, refer to the current row variable.

Table reference

A table reference is a special case of the field references that are used to refer to message trees. It always starts with the word “Database” and can contain any of the following elements:

- A table name only
- A schema name and a table name
- A data source name (that is, the name of a database instance), a schema name, and a table name

In each case, the name can be specified directly or by an expression enclosed in braces ({...}). A directly-specified data source, schema, or table name is subject to name substitution. That is, if the name used has been declared to be a known name, the value of the declared name is used rather than the name itself (see "DECLARE statement" on page 5117).

If a schema name is not specified, the default schema for the broker's database user is used.

If a data source name is not specified, the database pointed to by the node's `data source` attribute is used.

The WHERE clause

The WHERE clause expression can use any of the broker's operators and functions in any combination. It can refer to table columns, message fields, and any declared variables or constants.

However, be aware that the broker treats the WHERE clause expression by examining the expression and deciding whether the whole expression can be evaluated by the database. If it can, it is given to the database. In order to be evaluated by the database, it must use only those functions and operators supported by the database.

The WHERE clause can, however, refer to message fields, correlation names declared by containing SELECT functions, and to any other declared variables or constants within scope.

If the whole expression cannot be evaluated by the database, the broker looks for top-level AND operators and examines each sub-expression separately. It then attempts to give the database those sub-expressions that it can evaluate, leaving the broker to evaluate the rest. You need to be aware of this situation for two reasons:

1. Apparently trivial changes to WHERE clause expressions can have large effects on performance. You can determine how much of the expression was given to the database by examining a user trace.
2. Some databases' functions exhibit subtle differences of behavior from those of the broker.

Handling errors

It is possible for errors to occur during update operations. For example, the database might not be operational, or the table might have constraints defined that the new values would violate. In these cases, an exception is thrown (unless the node has its `throw exception on database error` property set to FALSE). These exceptions set appropriate SQL code, state, native error, and error text values and can be dealt with by error handlers (see the DECLARE HANDLER statement).

For further information about handling database errors, see "Capturing database state" on page 2512.

Examples

The following example assumes that the `dataSource` property of the Database node has been configured, and that the database it identifies has a table called

STOCKPRICES, with columns called COMPANY and PRICES. It updates the PRICE column of the rows in the STOCKPRICES table whose COMPANY column matches the value given in the Company field in the message.

```
UPDATE Database.StockPrices AS SP
 SET PRICE = InputBody.Message.StockPrice
 WHERE SP.COMPANY = InputBody.Message.Company
```

In the following example (which make similar assumptions), the SET clause expression refers to the existing value of a column and thus decrements the value by an amount in the message:

```
UPDATE Database.INVENTORY AS INV
 SET QUANTITY = INV.QUANTITY - InputBody.Message.QuantitySold
 WHERE INV.ITEMNUMBER = InputBody.Message.ItemNumber
```

The following example updates multiple columns:

```
UPDATE Database.table AS T
 SET column1 = T.column1+1,
     column2 = T.column2+2;
```

Note that the column names (on the left of the "=") are single identifiers. They must not be qualified with a table name or correlation name. In contrast, the references to database columns in the expressions (to the right of the "=") must be qualified with the correlation name.

The next example shows the use of calculated data source, schema, and table names:

```
-- Declare variables to hold the data source, schema and table names
-- and set their default values
DECLARE Source CHARACTER 'Production';
DECLARE Schema CHARACTER 'db2admin';
DECLARE Table  CHARACTER 'DynamicTable1';
-- Code which calculates their actual values comes here

-- Update rows in the table
UPDATE Database.{Source}.{Schema}.{Table} AS R SET Value = 0;
```

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Capturing database state" on page 2512
If an error occurs when the broker accesses an external database, you can either let the broker throw an exception during node processing or use ESQL statements to process the exception within the node itself.

Related reference:

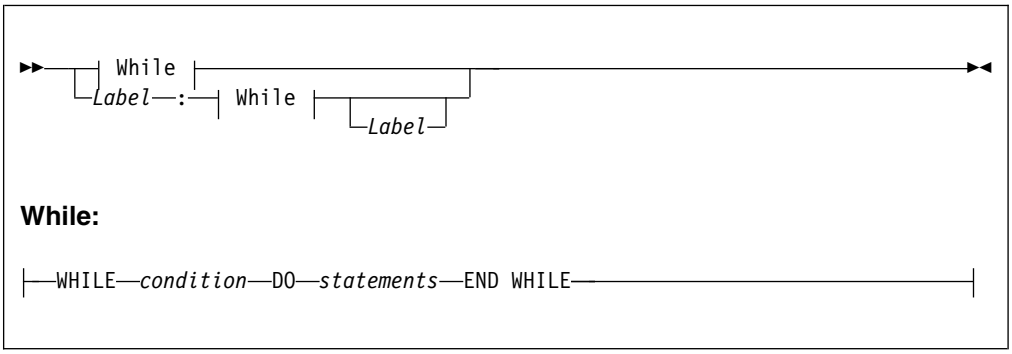
"Syntax diagrams" on page 3677

"ESQL statements" on page 5067
You can use ESQL statements to manipulate message trees, update databases, or interact with nodes.

WHILE statement:

The WHILE statement evaluates a condition expression, and if it is TRUE executes a sequence of statements.

Syntax



The WHILE statement repeats the steps specified in DO provided that *condition* is TRUE. It is your responsibility to ensure that the logic of the program is such that the loop terminates. If *condition* evaluates to UNKNOWN, the loop terminates immediately.

If present, *Label* gives the statement a name. This has no effect on the behavior of the WHILE statement itself, but allows statements to include ITERATE and LEAVE statements or other labelled statements, which in turn include them. The second *Label* can be present only if the first *Label* is present and if it is, the labels must be identical. It is not an error for two or more labelled statements at the same level to have the same *Label*, but this partly negates the advantage of the second *Label*. The advantage is that it unambiguously and accurately matches each END with its WHILE. However, it is an error for a labelled statement within statements to have the same label, because this makes the behavior of the ITERATE and LEAVE statements ambiguous.

Example

For example:

```
DECLARE i INTEGER;
SET i = 1;
X : WHILE i <= 3 DO
  ...
  SET i = i + 1;
  END WHILE X;
```

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Creating dynamic field references" on page 2431
You can use a variable of type REFERENCE as a dynamic reference to navigate a

message tree. This acts in a similar way to a message cursor or a variable pointer.

Related reference:

"Syntax diagrams" on page 3677

"ESQL statements" on page 5067
You can use ESQL statements to manipulate message trees, update databases, or interact with nodes.

ESQL functions:

The following types of function are available.

- "Calling ESQL functions"
- "ESQL database state functions"
- "ESQL datetime functions" on page 5176
- "ESQL numeric functions" on page 5183
- "ESQL string manipulation functions" on page 5205
- "ESQL field functions" on page 5224
- "ESQL list functions" on page 5238
- "Complex ESQL functions" on page 5242
- "Miscellaneous ESQL functions" on page 5290

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Calling ESQL functions:

Most ESQL functions belong to a schema called SQL and this is particularly useful if you have functions with the same name.

For example, if you have created a function called SQRT, you can code:

```
/* call my SQRT function  */

SET Variable1=SQRT (4);

/* call the SQL supplied function */

SET Variable2=SQL.SQRT (144);
```

Most of the functions described in this section impose restrictions on the data types of the arguments that can be passed to the function. If the values passed to the functions do not match the required data types, errors are generated at node configuration time whenever possible. Otherwise runtime errors are generated when the function is evaluated.

ESQL database state functions:
ESQL provides four functions to return database state. These are:

- "SQLCODE function" on page 5169
- "SQLERRORTEXT function" on page 5170

- "SQLNATIVEERROR function" on page 5171
- "SQLSTATE function" on page 5173

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

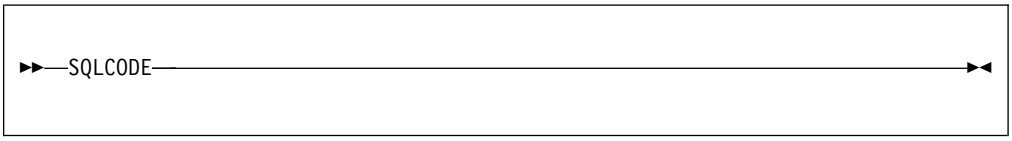
"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Capturing database state" on page 2512
If an error occurs when the broker accesses an external database, you can either let the broker throw an exception during node processing or use ESQL statements to process the exception within the node itself.

SQLCODE function:

SQLCODE is a database state function that returns an INTEGER data type with a default value of 0 (zero).

Syntax



Within a message flow, you can access and update an external database resource using the available ESQL database functions in the Filter, Database, and Compute nodes. When making calls to an external database, you might get errors, such as a table does not exist, a database is not available, or an insert for a key that already exists.

When these errors occur, the default action of the broker is to generate an exception. This behavior is determined by how you have set the property *Throw exception on database error*. If this check box is selected, the broker stops processing the node, propagates the message to the node's failure terminal, and writes the details of the error to the ExceptionList. If you want to override the default behavior and handle a database error in the ESQL in the node, clear the *Throw exception on database error* check box. The broker does not throw an exception and you must include the THROW statement to throw an exception if a certain SQL state code is not expected. See "THROW statement" on page 5161 for a description of THROW.

If you choose to handle database errors in a node, you can use the database state function SQLCODE to receive information about the status of the DBMS call made in ESQL. You can include it in conditional statements in current node's ESQL to recognize and handle possible errors.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Capturing database state" on page 2512
If an error occurs when the broker accesses an external database, you can either let the broker throw an exception during node processing or use ESQL statements to process the exception within the node itself.

Related reference:

"SQLERRORTEXT function"
SQLERRORTEXT is a database state function that returns a CHARACTER data type with a default value of '' (empty string).

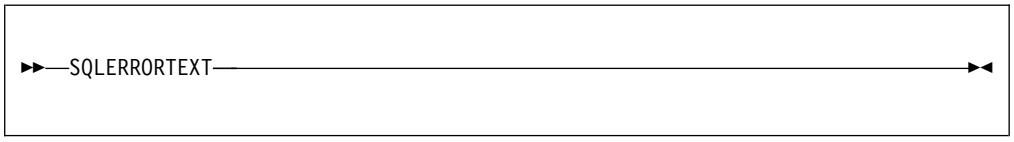
"SQLNATIVEERROR function" on page 5171
SQLNATIVEERROR is a database state function that returns an INTEGER data type with a default value of 0 (zero).

"SQLSTATE function" on page 5173
SQLSTATE is a database state function that returns a 5 character data type of CHARACTER with a default value of '00000' (five zeros as a string).

SQLERRORTEXT function:

SQLERRORTEXT is a database state function that returns a CHARACTER data type with a default value of '' (empty string).

Syntax



Within a message flow, you can access and update an external database resource using the available ESQL database functions in the Filter, Database, and Compute nodes. When making calls to an external database, you might get errors, such as a table does not exist, a database is not available, or an insert for a key that already exists.

When these errors occur, the default action of the broker is to generate an exception. This behavior is determined by how you have set the property *Throw exception on database error*. If you have selected this check box, the broker stops processing the node, propagates the message to the node's failure terminal, and writes the details of the error to the ExceptionList. If you want to override the default behavior and handle a database error in the ESQL in the node, clear the *Throw exception on database error* check box. The broker does not throw an exception and you must include the THROW statement to throw an exception if a certain SQL state code is not expected. See "THROW statement" on page 5161 for a description of THROW.

If you choose to handle database errors in a node, you can use the database state function SQLERRORTEXT to receive information about the status of the DBMS call made in ESQL. You can include it in conditional statements in current node's ESQL to recognize and handle possible errors.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Capturing database state" on page 2512
If an error occurs when the broker accesses an external database, you can either let the broker throw an exception during node processing or use ESQL statements to process the exception within the node itself.

Related reference:

"SQLCODE function" on page 5169
SQLCODE is a database state function that returns an INTEGER data type with a default value of 0 (zero).

"SQLNATIVEERROR function"
SQLNATIVEERROR is a database state function that returns an INTEGER data type with a default value of 0 (zero).

"SQLSTATE function" on page 5173
SQLSTATE is a database state function that returns a 5 character data type of CHARACTER with a default value of '00000' (five zeros as a string).

SQLNATIVEERROR function:

SQLNATIVEERROR is a database state function that returns an INTEGER data type with a default value of 0 (zero).

Syntax

Within a message flow, you can access and update an external database resource using the available ESQL database functions in the Filter, Database, and Compute nodes. When making calls to an external database, you might get errors, such as a table does not exist, a database is not available, or an insert for a key that already exists.

When these errors occur, the default action of the broker is to generate an exception. This behavior is determined by how you have set the property *Throw exception on database error*. If you have selected this check box, the broker stops processing the node, propagates the message to the node's failure terminal, and writes the details of the error to the ExceptionList. If you want to override the default behavior and handle a database error in the ESQL in the node, clear the *Throw exception on database error* check box. The broker does not throw an exception and you must include the THROW statement to throw an exception if a certain SQL state code is not expected. See "THROW statement" on page 5161 for a description of THROW.

If you choose to handle database errors in a node, you can use the database state function SQLNATIVEERROR to receive information about the status of the DBMS call made in ESQL. You can include it in conditional statements in current node's ESQL to recognize and handle possible errors.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Capturing database state" on page 2512
If an error occurs when the broker accesses an external database, you can either let the broker throw an exception during node processing or use ESQL statements to process the exception within the node itself.

Related reference:

"SQLCODE function" on page 5169
SQLCODE is a database state function that returns an INTEGER data type with a default value of 0 (zero).

"SQLERRORTEXT function" on page 5170
SQLERRORTEXT is a database state function that returns a CHARACTER data

type with a default value of '' (empty string).

“SQLSTATE function”
SQLSTATE is a database state function that returns a 5 character data type of CHARACTER with a default value of '00000' (five zeros as a string).

SQLSTATE function:

SQLSTATE is a database state function that returns a 5 character data type of CHARACTER with a default value of '00000' (five zeros as a string).

Syntax

Within a message flow, you can access and update an external database resource using the available ESQL database functions in the Compute, Database, and Filter nodes. When making calls to an external database, you might get errors, such as a table does not exist, a database is not available, or an insert for a key that already exists.

When these errors occur, the default action of the broker is to generate an exception. This behavior is determined by how you have set the property `Throw exception on database error`. If you select this property, the broker stops processing the node, propagates the message to the node's failure terminal, and writes the details of the error to the ExceptionList. If you want to override the default behavior and handle a database error in the ESQL in the node, clear `Throw exception on database error`. The broker does not throw an exception and you must include the THROW statement to throw an exception if a certain SQL state code is not expected. See “THROW statement” on page 5161 for a description of THROW.

To handle database errors in a node, you can use the database state function SQLSTATE to receive information about the status of the DBMS call made in ESQL. You can include it in conditional statements in current node's ESQL to recognize and handle possible errors.

SQL states

In ESQL, SQL states are variable length character strings. By convention, they are six characters long and contain only the characters 0-9, A-Z . The significance of the six characters is:

Char 1
The origin of the exception

Chars 2 - 3
The class of the exception

Chars 4 - 6
The subclass of the exception

The SQL state of an exception is determined by a two stage process. In the **first stage**, the exception information is examined and any wrapping exceptions (that is,

information that says what the broker was doing at the time the exception occurred) is stepped over until the exception that describes the original error is located.

The **second stage** is as follows:

1. If the selected exception is a database exception, the SQL state is that supplied by the database, but prefixed by the letter “D” to avoid any confusion with exceptions arising in the broker. The SQL code, native error, and error text are those supplied by the database.
2. If the selected exception is a user exception (that is, it originated in a THROW statement), the SQL code, state, native error, and error text are taken from the first four inserts of the exception, in order. The resulting state value is taken as is (not prefixed by a letter such as “U”). The letter “U” is not used by the broker as an origin indicator. If you want to define a unique SQL state rather than to imitate an existing one, use SQL states starting with the letter “U”. If you use SQL states that start with the letter “U”, you can write an error handler to match all user-defined and thrown exceptions with a LIKE'U%' operator.
3. If the selected exception originated in the message transport or in the ESQL implementation itself, the SQL code, state, native error, and error text are as described in the list later in this section.
4. For all other exceptions, the SQL state is '', indicating no origin, no class, and no subclass.

Some exceptions that currently give an empty SQL state might give individual states in future releases. If you want to catch unclassified exceptions, use the “all” wildcard (“%”) for the SQL state on the last handler of a scope. This wildcard will continue to catch the same set of exceptions if previously unclassified exceptions are given new unique SQL states.

The following SQL states are defined:

Dddddd
ddddd is the state returned by the database.

SqlState = 'S22003'
Arithmetic overflow. An operation whose result is a numeric type resulted in a value beyond the range supported.

SqlState = 'S22007'
Date time format not valid. A character string used in a cast from character to a datetime type had either the wrong basic format (for example, '01947-10-24') or had values outside the ranges allowed by the Gregorian calendar (for example, '1947-21-24').

SqlState = 'S22008'
Date time field overflow. An operation whose result is a datetime type resulted in a value beyond the range supported.

SqlState = 'S22012'
Divide by zero. A divide operation whose result data type has no concept of infinity had a zero right operand.

SqlState = 'S22015'
Interval field overflow. An operation whose result is of type INTERVAL resulted in a value beyond the range supported by the INTERVAL data type.

SqlState = 'S22018'
Character value for cast not valid.

SqlState = 'SFN001'
A SELECT function used in an IN predicate returned more than one column when only one column is allowed in this case.

SqlState = 'SPS001'
Target terminal not valid. A PROPAGATE to terminal statement attempted to use a terminal name that is not valid.

SqlState = 'SPS002'
Target label not valid. A PROPAGATE to label statement attempted to use a label that is not valid.

SqlState = 'SPS003'
PROPAGATE statement not valid in this context. Use the RETURN function instead.

SqlState = 'MQW001', SqlNativeError = 0
The bit stream does not meet the requirements for WebSphere MQ messages. No attempt was made to put it to a queue. Retrying and queue administration does not resolve this problem.

SqlState = 'MQW002', SqlNativeError = 0
The target queue or queue manager names were not valid (that is, they could not be converted from Unicode to the queue manager's code page). Retrying and queue emptying does not resolve this problem.

SqlState = 'MQW003', SqlNativeError = 0
Request mode was specified but the “reply to” queue or queue manager names were not valid (that is, could not be converted from Unicode to the message's code page). Retrying and queue emptying does not resolve this problem.

SqlState = 'MQW004', SqlNativeError = 0
Reply mode was specified but the queue or queue manager names taken from the message were not valid (that is, they could not be converted from the given code page to Unicode). Retrying and queue emptying does not resolve this problem.

SqlState = 'MQW005', SqlNativeError = 0
Destination list mode was specified but the destination list supplied does not meet the basic requirements for destination lists. No attempt was made to put any message to a queue. Retrying and queue administration does not resolve this problem.

SqlState = 'MQW101', SqlNativeError = returned by WebSphere MQ
The target queue manager or queue could not be opened. Queue administration might resolve this problem but retrying does not.

SqlState = 'MQW102', SqlNativeError = returned by WebSphere MQ
The target queue manager or queue could not be written to. Retrying and queue administration might resolve this problem.

SqlState = 'MQW201', SqlNativeError = number of destinations with an error
More than one error occurred while processing a destination list. The message might have been put to zero or more queues. Retrying and queue administration might resolve this problem.

Anything that the user has used in a THROW statement
Use `Uuuuuuu` for user exceptions, unless imitating one of the exceptions defined above.

Empty string
All other errors.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Capturing database state" on page 2512
If an error occurs when the broker accesses an external database, you can either let the broker throw an exception during node processing or use ESQL statements to process the exception within the node itself.

Related reference:

"SQLCODE function" on page 5169
SQLCODE is a database state function that returns an INTEGER data type with a default value of 0 (zero).

"SQLERRORTEXT function" on page 5170
SQLERRORTEXT is a database state function that returns a CHARACTER data type with a default value of '' (empty string).

"SQLNATIVEERROR function" on page 5171
SQLNATIVEERROR is a database state function that returns an INTEGER data type with a default value of 0 (zero).

ESQL datetime functions:
This topic lists the ESQL datetime functions.

In addition to the functions described here, you can use arithmetic operators to perform various calculations on datetime values. For example, you can use the - (minus) operator to calculate the difference between two dates as an interval, or you can add an interval to a timestamp.

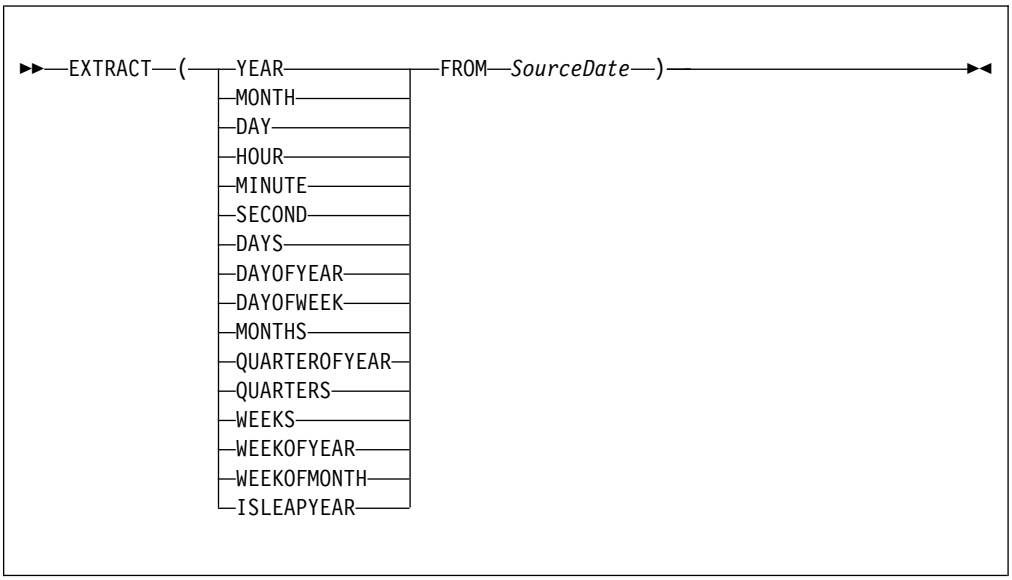
This section covers the following topics:

EXTRACT function:

The EXTRACT function extracts fields (or calculates values) from datetime values and intervals.

The result is INTEGER for YEAR, MONTH, DAY, HOUR, MINUTE, DAYS, DAYOFYEAR, DAYOFWEEK, MONTHS, QUARTEROFYEAR, QUARTERS, WEEKS, WEEKOFYEAR, and WEEKOFMONTH extracts, but FLOAT for SECOND extracts, and BOOLEAN for ISLEAPYEAR extracts. If the *SourceDate* is NULL, the result is NULL regardless of the type of extract.

Syntax



EXTRACT extracts individual fields from datetime values and intervals. You can extract a field only if it is present in the datetime value specified in the second parameter. Either a parse-time or a runtime error is generated if the requested field does not exist within the data type.

The following table describes the extracts that are supported:

Note: All new integer values start from 1.

Table 263.

Extract	Description
YEAR	Year
MONTH	Month
DAY	Day
HOUR	Hour
MINUTE	Minute
SECOND	Second
DAYS	Days encountered between 1st January 0001 and the *SourceDate*.
DAYOFYEAR	Day of year

Table 263. (continued)

Extract	Description
DAYOFWEEK	Day of the week: Sunday = 1, Monday = 2, Tuesday = 3, Wednesday = 4, Thursday = 5, Friday = 6, Saturday = 7.
MONTHS	Months encountered between 1st January 0001 and the *SourceDate*.
QUARTEROFYEAR	Quarter of year: January to March = 1, April to June = 2, July to September = 3, October to December = 4.
QUARTERS	Quarters encountered between 1st January 0001 and the *SourceDate*.
WEEKS	Weeks encountered between 1st January 0001 and the *SourceDate*.
WEEKOFYEAR	Week of year
WEEKOFMONTH	Week of month
ISLEAPYEAR	Whether this is a leap year

Notes:

1. A week is defined as Sunday to Saturday, not any seven consecutive days. You must convert to an alternative representation scheme if required.
2. The source date time epoch is 1 January 0001. Dates before the epoch are not valid for this function.
3. The Gregorian calendar is assumed for calculation.

Example

```
EXTRACT(YEAR FROM CURRENT_DATE)
```

and

```
EXTRACT(HOUR FROM LOCAL_TIMEZONE)
```

both work without error, but

```
EXTRACT(DAY FROM CURRENT_TIME)
```

fails.

```
EXTRACT (DAYS FROM DATE '2000-02-29')
```

calculates the number of days encountered since year 1 to `'2000-02-29'` and

```
EXTRACT (DAYOFYEAR FROM CURRENT_DATE)
```

calculates the number of days encountered since the beginning of the current year but

```
EXTRACT (DAYOFYEAR FROM CURRENT_TIME)
```

fails because CURRENT_TIME does not contain date information.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"Syntax diagrams" on page 3677

"ESQL datetime functions" on page 5176

CURRENT_DATE function:
The CURRENT_DATE datetime function returns the current date.

Syntax

CURRENT_DATE returns a DATE value representing the current date in local time. As with all SQL functions that take no parameters, no parentheses are required or accepted. All calls to CURRENT_DATE within the processing of one node are guaranteed to return the same value.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

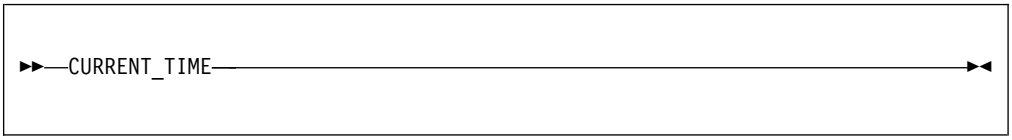
Related reference:

"Syntax diagrams" on page 3677

"ESQL datetime functions" on page 5176

CURRENT_TIME function:
The CURRENT_TIME datetime function returns the current local time.

Syntax



CURRENT_TIME returns a TIME value representing the current local time. As with all SQL functions that take no parameters, no parentheses are required or accepted. All calls to CURRENT_TIME within the processing of one node are guaranteed to return the same value.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

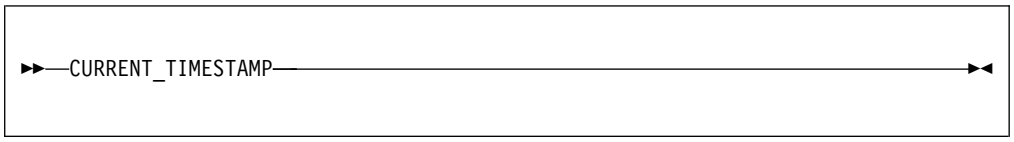
Related reference:

"Syntax diagrams" on page 3677

"ESQL datetime functions" on page 5176

CURRENT_TIMESTAMP function:
The CURRENT_TIMESTAMP datetime function returns the current date and local time.

Syntax



CURRENT_TIMESTAMP returns a TIMESTAMP value representing the current date and local time. As with all SQL functions that take no parameters, no parentheses are required or accepted. All calls to CURRENT_TIMESTAMP within the processing of one node are guaranteed to return the same value.

Example

To obtain the following XML output message:

```
<Body>
<Message>Hello World</Message>
<DateStamp>2006-02-01 13:13:56.444730</DateStamp>
</Body>
```

use the following ESQL:

```
SET OutputRoot.XMLNS.Body.Message = 'Hello World';
SET OutputRoot.XMLNS.Body.DateStamp = CURRENT_TIMESTAMP;
```

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

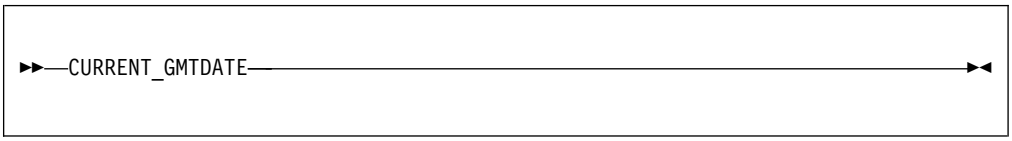
"Syntax diagrams" on page 3677

"ESQL datetime functions" on page 5176

CURRENT_GMTDATE function:

The CURRENT_GMTDATE datetime function returns the current date in the GMT time zone.

Syntax



CURRENT_GMTDATE returns a DATE value representing the current date in the GMT time zone. As with all SQL functions that take no parameters, no parentheses are required or accepted. All calls to CURRENT_GMTDATE within the processing of one node are guaranteed to return the same value.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"Syntax diagrams" on page 3677

"ESQL datetime functions" on page 5176

CURRENT_GMTTIME function:
The CURRENT_GMTTIME datetime function returns the current time in the GMT time zone.

Syntax

It returns a GMTTIME value representing the current time in the GMT time zone. As with all SQL functions that take no parameters, no parentheses are required or accepted. All calls to CURRENT_GMTTIME within the processing of one node are guaranteed to return the same value.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"Syntax diagrams" on page 3677

"ESQL datetime functions" on page 5176

CURRENT_GMTTIMESTAMP function:

The CURRENT_GMTTIMESTAMP datetime function returns the current date and time in the GMT time zone.

Syntax

CURRENT_GMTTIMESTAMP returns a GMTTIMESTAMP value representing the current date and time in the GMT time zone. As with all SQL functions that take no parameters, no parentheses are required or accepted. All calls to CURRENT_GMTTIMESTAMP within the processing of one node are guaranteed to return the same value.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

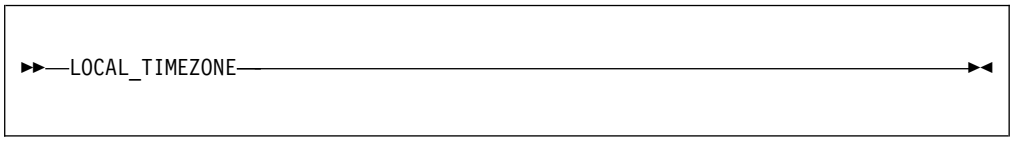
"Syntax diagrams" on page 3677

"ESQL datetime functions" on page 5176

LOCAL_TIMEZONE function:

The LOCAL_TIMEZONE datetime function returns the displacement of the local time zone from GMT.

Syntax



LOCAL_TIMEZONE returns an interval value representing the local time zone displacement from GMT. As with all SQL functions that take no parameters, no parentheses are required or accepted. The value returned is an interval in hours and minutes representing the displacement of the current time zone from Greenwich Mean Time.

The sign of the interval is such that a local time can be converted to a time in GMT by subtracting the result of the LOCAL_TIMEZONE function. However, for calculations involving GMTTIMEs and GMTTIMESTAMPs, ESQL performs this transformation automatically.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"Syntax diagrams" on page 3677

"ESQL datetime functions" on page 5176

ESQL numeric functions:

A list of the numeric functions that ESQL supports.

This topic covers the following:

“RAND function” on page 5197

“ROUND function” on page 5198

“SIGN function” on page 5201

“SIN function” on page 5202

“SINH function” on page 5202

“SQRT function” on page 5203

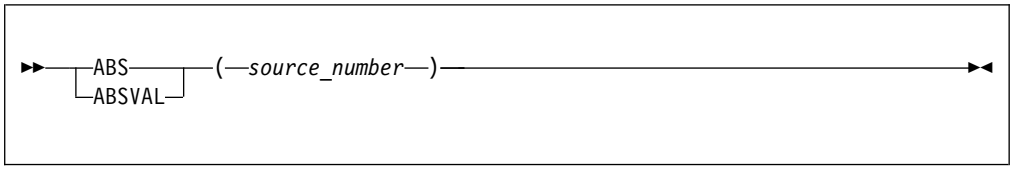
“TAN function” on page 5203

“TANH function” on page 5204

“TRUNCATE function” on page 5205

ABS and ABSVAL functions:
The ABS and ABSVAL numeric functions return the absolute value of a supplied number.

Syntax



The absolute value of the source number is a number with the same magnitude as the source but without a sign. The parameter must be a numeric value. The result is of the same type as the parameter unless it is NULL, in which case the result is NULL.

For example:

```
ABS( -3.7 )
```

returns `3.7`

```
ABS( 3.7 )
```

returns `3.7`

```
ABS( 1024 )
```

returns `1024`

Related concepts:

“ESQL overview” on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

“Developing ESQL” on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

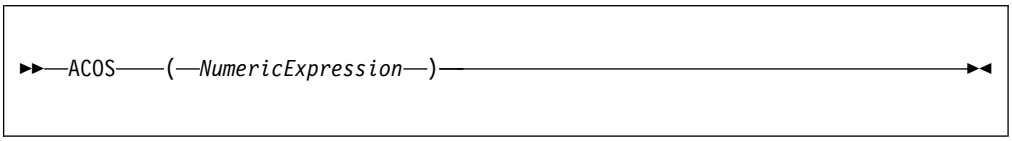
Related reference:

“Syntax diagrams” on page 3677

"ESQL numeric functions" on page 5183
A list of the numeric functions that ESQL supports.

ACOS function:
The ACOS numeric function returns the angle of a given cosine.

Syntax



The ACOS function returns the angle, in radians, whose cosine is the given *NumericExpression*. The parameter can be any built-in numeric data type. The result is FLOAT unless the parameter is NULL, in which case the result is NULL.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

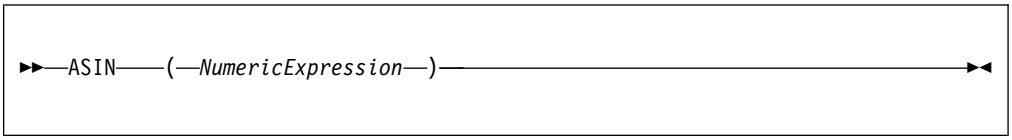
Related reference:

"Syntax diagrams" on page 3677

"ESQL numeric functions" on page 5183
A list of the numeric functions that ESQL supports.

ASIN function:
The ASIN numeric function returns the angle of the given sine.

Syntax



The ASIN function returns the angle, in radians, whose sine is the given *NumericExpression*. The parameter can be any built-in numeric data type. The result is FLOAT unless the parameter is NULL, in which case the result is NULL.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

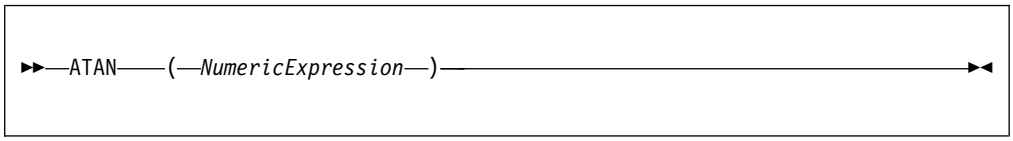
Related reference:

"Syntax diagrams" on page 3677

"ESQL numeric functions" on page 5183
A list of the numeric functions that ESQL supports.

ATAN function:
The ATAN numeric function returns the angle of the given tangent.

Syntax



The ATAN function returns the angle, in radians, whose tangent is the given *NumericExpression*. The parameter can be any built-in numeric data type. The result is FLOAT unless the parameter is NULL, in which case the result is NULL.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

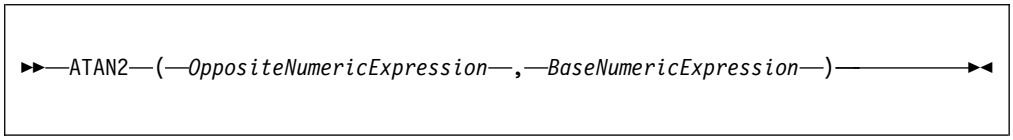
Related reference:

"Syntax diagrams" on page 3677

"ESQL numeric functions" on page 5183
A list of the numeric functions that ESQL supports.

ATAN2 function:
The ATAN2 numeric function returns the angle subtended in a right angled triangle between an opposite and the base.

Syntax



The ATAN2 function returns the angle, in radians, subtended (in a right angled triangle) by an opposite given by *OppositeNumericExpression* and the base given by *BaseNumericExpression*. The parameters can be any built-in numeric data type. The result is FLOAT unless either parameter is NULL, in which case the result is NULL

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

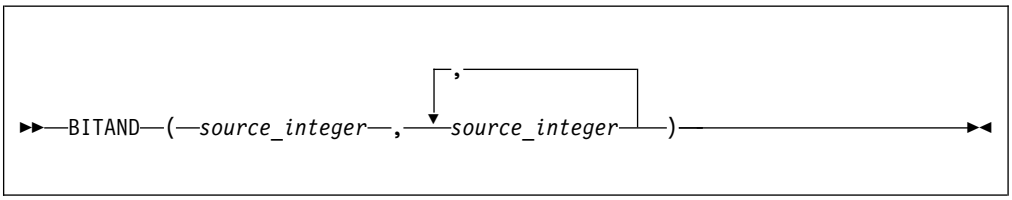
Related reference:

"Syntax diagrams" on page 3677

"ESQL numeric functions" on page 5183
A list of the numeric functions that ESQL supports.

BITAND function:
The BITAND numeric function performs a bitwise AND on the binary representation of two or more numbers.

Syntax



BITAND takes two or more integer values and returns the result of performing the bitwise AND on the binary representation of the numbers. The result is INTEGER unless either parameter is NULL, in which case the result is NULL.

For example:

```
BITAND(12, 7)
```

returns 4 as shown by this worked example:

```
   Binary Decimal
    1100    12
AND 0111     7
________
    0100     4
```

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"Syntax diagrams" on page 3677

"ESQL numeric functions" on page 5183
A list of the numeric functions that ESQL supports.

BITNOT function:
The BITNOT numeric function performs a bitwise complement on the binary representation of a number.

Syntax

BITNOT takes an integer value and returns the result of performing the bitwise complement on the binary representation of the number. The result is INTEGER unless either parameter is NULL, in which case the result is NULL.

For example:

```
BITNOT(7)
```

returns -8, as shown by this worked example:

```
   Binary Decimal
00...0111    7
NOT
________
11...1000   -8
```

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

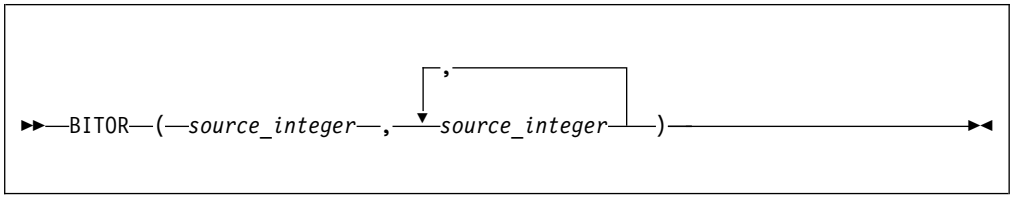
Related reference:

"Syntax diagrams" on page 3677

"ESQL numeric functions" on page 5183
A list of the numeric functions that ESQL supports.

BITOR function:
The BITOR numeric function performs a bitwise OR on the binary representation of two or more numbers.

Syntax



BITOR takes two or more integer values and returns the result of performing the bitwise OR on the binary representation of the numbers. The result is INTEGER unless either parameter is NULL, in which case the result is NULL.

For example:

```
BITOR(12, 7)
```

returns 15, as shown by this worked example:

```
   Binary Decimal
    1100    12
OR  0111     7
________
    1111    15
```

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

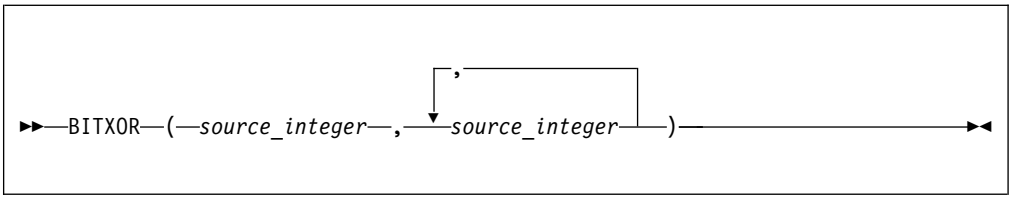
"Syntax diagrams" on page 3677

"ESQL numeric functions" on page 5183
A list of the numeric functions that ESQL supports.

BITXOR function:

The BITXOR numeric function performs a bitwise XOR on the binary representation of two or more numbers.

Syntax



BITXOR takes two or more integer values and returns the result of performing the bitwise XOR on the binary representation of the numbers. The result is INTEGER unless either parameter is NULL, in which case the result is NULL.

For example:

```
BITXOR(12, 7)
```

returns 11, as shown by this worked example:

```
   Binary Decimal
    1100    12
XOR 0111     7
________
    1011    11
```

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

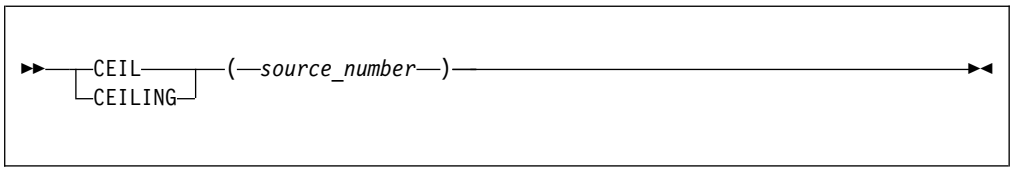
Related reference:

"Syntax diagrams" on page 3677

"ESQL numeric functions" on page 5183
A list of the numeric functions that ESQL supports.

CEIL and CEILING functions:
The CEIL and CEILING numeric functions return the smallest integer equivalent of a decimal number.

Syntax



CEIL and CEILING return the smallest integer value greater than or equal to `source_number`. The parameter can be any numeric data type. The result is of the same type as the parameter unless it is NULL, in which case the result is NULL.

For example:

```
CEIL(1)
```

returns `1`

```
CEIL(1.2)
```

returns `2.0`

```
CEIL(-1.2)
```

returns `-1.0`

If possible, the scale is changed to zero. If the result cannot be represented at that scale, it is made sufficiently large to represent the number.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

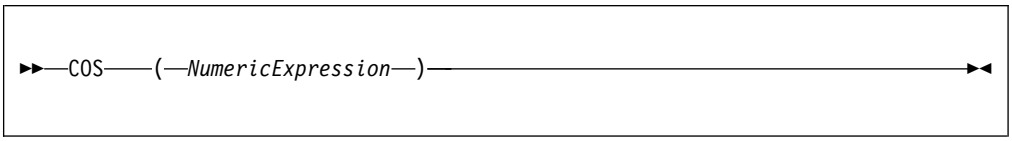
"Syntax diagrams" on page 3677

"ESQL numeric functions" on page 5183
A list of the numeric functions that ESQL supports.

COS function:

The COS numeric function returns the cosine of a given angle.

Syntax



The COS function returns the cosine of the angle, in radians, given by *NumericExpression*. The parameter can be any built-in numeric data type. The result is FLOAT unless the parameter is NULL, in which case the result is NULL.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

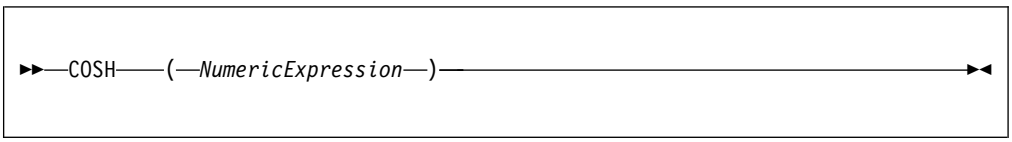
Related reference:

"Syntax diagrams" on page 3677

"ESQL numeric functions" on page 5183
A list of the numeric functions that ESQL supports.

COSH function:
The COSH numeric function returns the hyperbolic cosine of a given angle.

Syntax



The COSH function returns the hyperbolic cosine of the angle, in radians, given by *NumericExpression*. The parameter can be any built-in numeric data type. The result is FLOAT unless the parameter is NULL, in which case the result is NULL.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

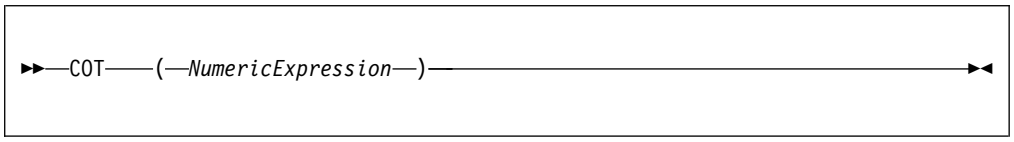
Related reference:

"Syntax diagrams" on page 3677

"ESQL numeric functions" on page 5183
A list of the numeric functions that ESQL supports.

COT function:
The COT numeric function returns the cotangent of a given angle.

Syntax



The COT function returns the cotangent of the angle, in radians, given by *NumericExpression*. The parameter can be any built-in numeric data type. The result is FLOAT unless the parameter is NULL, in which case the result is NULL.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

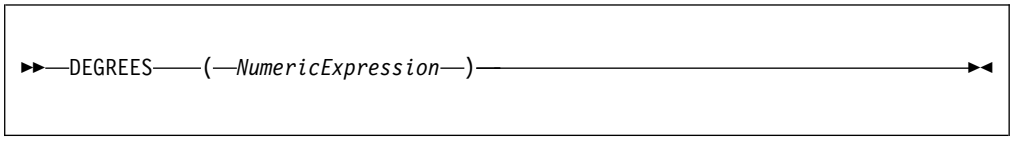
Related reference:

"Syntax diagrams" on page 3677

"ESQL numeric functions" on page 5183
A list of the numeric functions that ESQL supports.

DEGREES function:
The DEGREES numeric function returns the angle of the radians supplied.

Syntax



The DEGREES function returns the angle, in degrees, specified by *NumericExpression* in radians. The parameter can be any built-in numeric data type. The result is FLOAT unless the parameter is NULL, in which case the result is NULL.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and

Filter nodes in your message flows by coding ESQL.

Related reference:

"Syntax diagrams" on page 3677

"ESQL numeric functions" on page 5183
A list of the numeric functions that ESQL supports.

EXP function:
The EXP numeric function returns the exponential value of a given number.

Syntax

The EXP function returns the exponential of the value specified by *NumericExpression*. The parameter can be any built-in numeric data type. The result is FLOAT unless the parameter is NULL, in which case the result is NULL.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

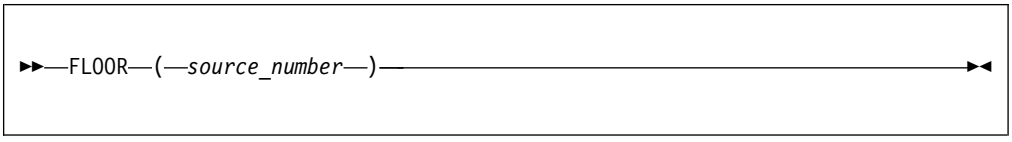
Related reference:

"Syntax diagrams" on page 3677

"ESQL numeric functions" on page 5183
A list of the numeric functions that ESQL supports.

FLOOR function:
The FLOOR numeric function returns the largest integer equivalent to a given decimal number.

Syntax



FLOOR returns the largest integer value less than or equal to `source_number`. The parameter can be any numeric data type. The result is of the same type as the parameter unless it is NULL, in which case the result is NULL.

For example:

```
FLOOR(1)
```

returns 1

```
FLOOR(1.2)
```

returns `1.0`

```
FLOOR(-1.2)
```

returns `-2.0`

If possible, the scale is changed to zero. If the result cannot be represented at that scale, it is made sufficiently large to represent the number.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

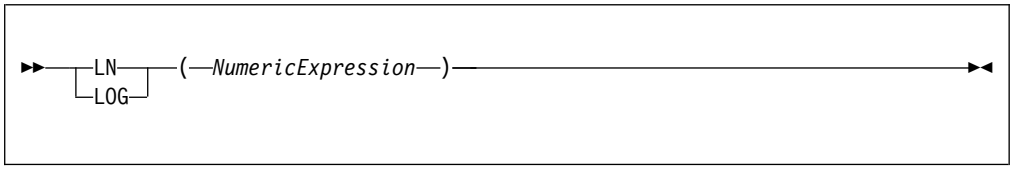
Related reference:

"Syntax diagrams" on page 3677

"ESQL numeric functions" on page 5183
A list of the numeric functions that ESQL supports.

LN and LOG functions:
The LN and LOG equivalent numeric functions return the natural logarithm of a given value.

Syntax



The LN and LOG functions return the natural logarithm of the value specified by *NumericExpression*. The parameter can be any built-in numeric data type. The result is FLOAT unless the parameter is NULL, in which case the result is NULL.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

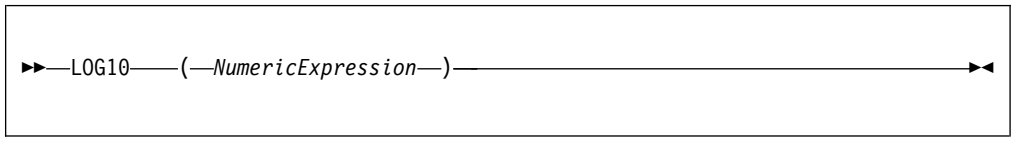
Related reference:

"Syntax diagrams" on page 3677

"ESQL numeric functions" on page 5183
A list of the numeric functions that ESQL supports.

LOG10 function:
The LOG10 numeric function returns the logarithm to base 10 of a given value.

Syntax



The LOG10 function returns the logarithm to base 10 of the value specified by *NumericExpression*. The parameter can be any built-in numeric data type. The result is FLOAT unless the parameter is NULL, in which case the result is NULL.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

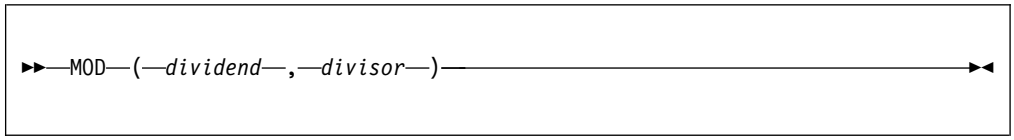
Related reference:

"Syntax diagrams" on page 3677

"ESQL numeric functions" on page 5183
A list of the numeric functions that ESQL supports.

MOD function:
The MOD numeric function returns the remainder when dividing two numbers.

Syntax



MOD returns the remainder when the first parameter is divided by the second parameter. The result is negative only if the first parameter is negative. Parameters must be integers. The function returns an integer. If any parameter is NULL, the result is NULL.

For example:

```
MOD(7, 3)
```

returns 1

```
MOD(-7, 3)
```

returns -1

```
MOD(7, -3)
```

returns 1

```
MOD(6, 3)
```

returns `0`

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

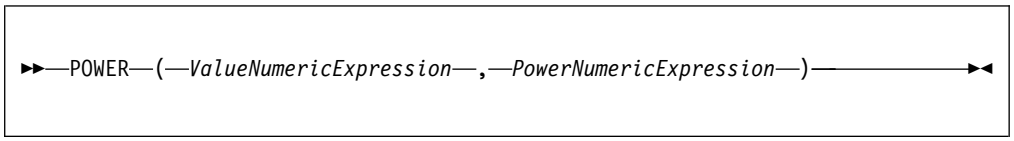
Related reference:

"Syntax diagrams" on page 3677

"ESQL numeric functions" on page 5183
A list of the numeric functions that ESQL supports.

POWER function:
The POWER numeric function raises a value to the power supplied.

Syntax



POWER returns the given value raised to the given power. The parameters can be any built-in numeric data type. The result is FLOAT unless any parameter is NULL, in which case the result is NULL

An exception occurs, if the value is either:

- Zero and the power is negative, or
- Negative and the power is not an integer

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

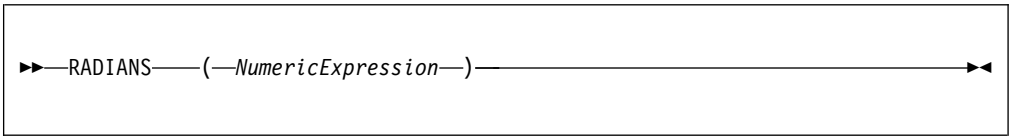
Related reference:

"Syntax diagrams" on page 3677

"ESQL numeric functions" on page 5183
A list of the numeric functions that ESQL supports.

RADIANS function:
The RADIANS numeric function returns a given radians angle in degrees.

Syntax



The RADIANS function returns the angle, in radians, specified by *NumericExpression* in degrees. The parameter can be any built-in numeric data type. The result is FLOAT unless the parameter is NULL, in which case the result is NULL.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

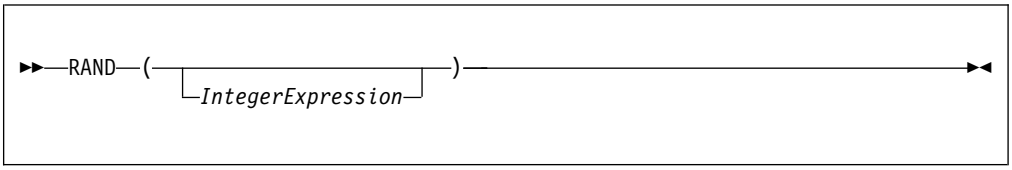
Related reference:

"Syntax diagrams" on page 3677

"ESQL numeric functions" on page 5183
A list of the numeric functions that ESQL supports.

RAND function:
The RAND numeric function returns a pseudo random number.

Syntax



The RAND function returns a pseudo random number in the range 0.0 to 1.0. If supplied, the parameter initializes the pseudo random sequence.

The parameter can be of any numeric data type, but any fractional part is ignored. The result is FLOAT unless the parameter is NULL, in which case the result is NULL.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

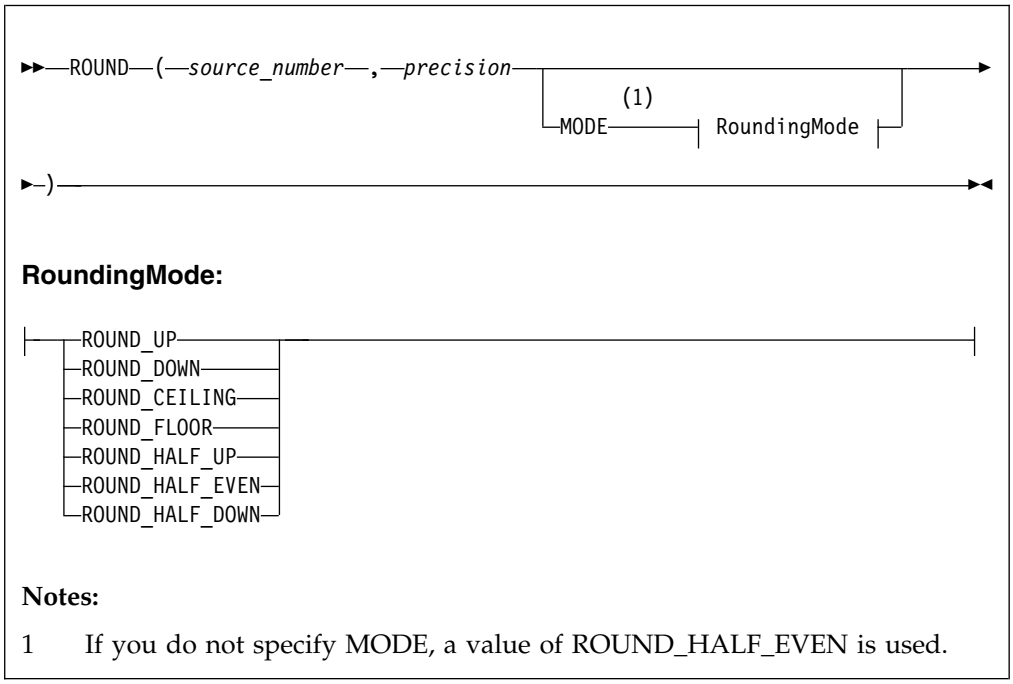
Related reference:

"Syntax diagrams" on page 3677

"ESQL numeric functions" on page 5183
A list of the numeric functions that ESQL supports.

ROUND function:
The ROUND numeric function rounds a supplied value to a given number of places.

Syntax



Notes:

1 If you do not specify MODE, a value of ROUND_HALF_EVEN is used.

If *precision* is a positive number, *source_number* is rounded to *precision* places right of the decimal point. If *precision* is negative, the result is *source_number* rounded to the absolute value of *precision* places to the left of the decimal point.

source_number can be any built-in numeric data type; *precision* must be an integer. The result is of the same data type as the *source_number* parameter unless *source_number* is NULL, in which case the result is NULL.

This means that the result of the function is:
- INTEGER if *source_number* is INTEGER
- FLOAT if *source_number* is FLOAT
- DECIMAL if *source_number* is DECIMAL

When rounding, the *banker's* or *half-even symmetric* rounding rules are used by default, unless a *RoundingMode* is specified.

RoundingMode

RoundingMode can take one of the following values:

ROUND_UP
Round away from zero. Always increments the digit prior to a nonzero discarded fraction. This rounding mode never decreases the magnitude of the calculated value.

ROUND_DOWN
Round towards zero. Never increments the digit prior to a discarded fraction, that is, truncates. This rounding mode never increases the magnitude of the calculated value.

ROUND_CEILING
Round towards positive infinity. If the decimal is positive, behaves as for ROUND_UP; if negative, behaves as for ROUND_DOWN. This rounding mode never decreases the calculated value.

ROUND_FLOOR
Round towards negative infinity. If the decimal is positive, behaves as for ROUND_DOWN; if negative, behaves as for ROUND_UP. This rounding mode never increases the calculated value.

ROUND_HALF_UP
Round towards "nearest neighbor" unless both neighbors are equidistant, in which case round up. Behaves as for ROUND_UP if the discarded fraction is greater than, or equal to, 0.5; otherwise, behaves as for ROUND_DOWN. This is the rounding mode that is typically taught in schools.

ROUND_HALF_DOWN
Round towards "nearest neighbor" unless both neighbors are equidistant, in which case round down. Behaves as for ROUND_UP if the discarded fraction is grater than 0.5; otherwise, behaves as for ROUND_DOWN.

ROUND_HALF_EVEN
Round towards the "nearest neighbor" unless both neighbors are equidistant, in which case, round towards the even neighbor. Behaves as for ROUND_HALF_UP if the digit to the left of the discarded fraction is odd; behaves as for ROUND_HALF_DOWN if it is even. This is the rounding mode that minimizes cumulative error when applied repeatedly over a sequence of calculations, and is sometimes referred to as Banker's rounding.

The following table gives a summary of rounding operations, with a precision of zero, under different rounding modes.

Input number	ROUND UP	ROUND DOWN	ROUND CEILING	ROUND FLOOR	ROUND HALF UP	ROUND HALF DOWN	ROUND HALF EVEN
5.5	6	5	6	5	6	5	6
2.5	3	2	3	2	3	2	2
1.6	2	1	2	1	2	2	2
1.1	2	1	2	1	1	1	1
1.0	1	1	1	1	1	1	1
-1.0	-1	-1	-1	-1	-1	-1	-1
-1.1	-2	-1	-1	-2	-1	-1	-1
-1.6	-2	-1	-1	-2	-2	-2	-2
-2.5	-3	-2	-2	-3	-3	-2	-2

Input number	ROUND UP	ROUND DOWN	ROUND CEILING	ROUND FLOOR	ROUND HALF UP	ROUND HALF DOWN	ROUND HALF EVEN
-5.5	-6	-5	-5	-6	-6	-5	-6

Examples using the default rounding mode (ROUND_HALF_EVEN):

```
ROUND(27.75, 2)
```

returns 27.75

```
ROUND(27.75, 1)
```

returns 27.8

```
ROUND(27.75, 0)
```

returns 28

```
ROUND(27.75, -1)
```

returns 30

Examples using a rounding mode with a precision of zero:

```
ROUND(5.5, 0 MODE ROUND_UP);
```

returns 6

```
ROUND(5.5, 0 MODE ROUND_DOWN);
```

returns 5

```
ROUND(5.5, 0 MODE ROUND_CEILING);
```

returns 6

```
ROUND(5.5, 0 MODE ROUND_FLOOR);
```

returns 5

```
ROUND(5.5, 0 MODE ROUND_HALF_UP);
```

returns 6

```
ROUND(5.5, 0 MODE ROUND_HALF_DOWN);
```

returns 5

```
ROUND(5.5, 0 MODE ROUND_HALF_EVEN);
```

returns 6

```
ROUND(2.5, 0 MODE ROUND_UP);
```

returns 3

```
ROUND(2.5, 0 MODE ROUND_DOWN);
```

returns 2

```
ROUND(2.5, 0 MODE ROUND_CEILING);
```

returns 3

```
ROUND(2.5, 0 MODE ROUND_FLOOR);
```

returns 2

```
ROUND(2.5, 0 MODE ROUND_HALF_UP);
```

returns 3

```
ROUND(2.5, 0 MODE ROUND_HALF_DOWN);
```

returns 2

```
ROUND(2.5, 0 MODE ROUND_HALF_EVEN);
```

returns 2

If possible, the scale is changed to the given value. If the result cannot be represented within the given scale, it is INFINITY.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

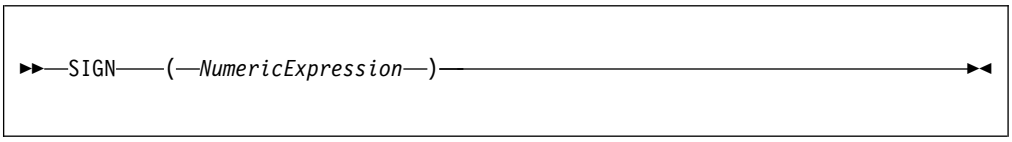
Related reference:

"Syntax diagrams" on page 3677

"ESQL numeric functions" on page 5183
A list of the numeric functions that ESQL supports.

SIGN function:
The SIGN numeric function tells you whether a given number is positive, negative, or zero.

Syntax



The SIGN function returns -1, 0, or +1 when the *NumericExpression* value is negative, zero, or positive respectively. The parameter can be any built-in numeric data type and the result is of the same type as the parameter. If the parameter is NULL, the result is NULL

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

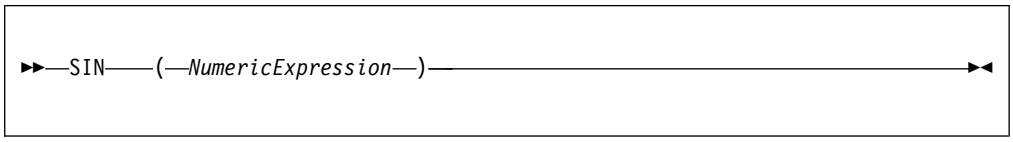
Related reference:

"Syntax diagrams" on page 3677

"ESQL numeric functions" on page 5183
A list of the numeric functions that ESQL supports.

SIN function:
The SIN numeric function returns the sine of a given angle.

Syntax



The SIN function returns the sine of the angle, in radians, given by *NumericExpression*. The parameter can be any built-in numeric data type. The result is FLOAT unless the parameter is NULL, in which case the result is NULL.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

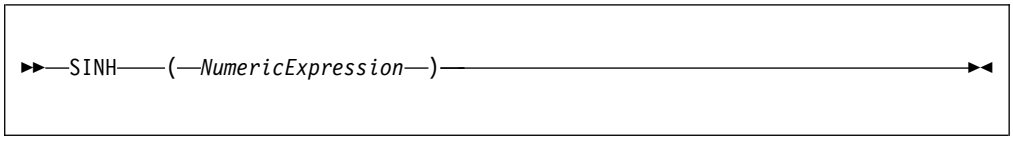
Related reference:

"Syntax diagrams" on page 3677

"ESQL numeric functions" on page 5183
A list of the numeric functions that ESQL supports.

SINH function:
The SINH numeric function returns the hyperbolic sine of a given angle.

Syntax



The SINH function returns the hyperbolic sine of the angle, in radians, given by *NumericExpression*. The parameter can be any built-in numeric data type. The result is FLOAT unless the parameter is NULL, in which case the result is NULL.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"Syntax diagrams" on page 3677

"ESQL numeric functions" on page 5183
A list of the numeric functions that ESQL supports.

SQRT function:
The SQRT numeric function returns the square root of a given number.

Syntax

SQRT returns the square root of *source_number*. The parameter can be any built-in numeric data type. The result is a FLOAT. If the parameter is NULL, the result is NULL.

For example:

```
SQRT(4)
```

returns `2E+1`

```
SQRT(2)
```

returns `1.414213562373095E+0`

```
SQRT(-1)
```

throws an exception.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"Syntax diagrams" on page 3677

"ESQL numeric functions" on page 5183
A list of the numeric functions that ESQL supports.

TAN function:
The TAN numeric function returns the tangent of a given angle.

Syntax

The TAN function returns the tangent of the angle, in radians, given by *NumericExpression*. The parameter can be any built-in numeric data type. The result is FLOAT unless the parameter is NULL, in which case the result is NULL.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

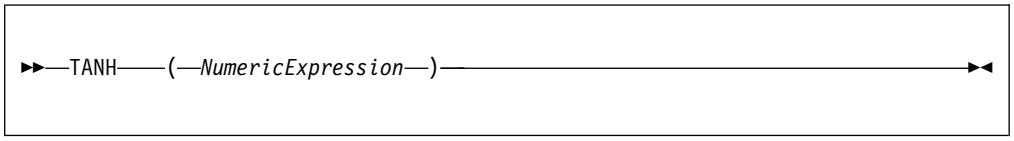
Related reference:

"Syntax diagrams" on page 3677

"ESQL numeric functions" on page 5183
A list of the numeric functions that ESQL supports.

TANH function:
The TANH numeric function returns the hyperbolic tangent of an angle.

Syntax



The TANH function returns the hyperbolic tangent of the angle, in radians, given by *NumericExpression*. The parameter can be any built-in numeric data type. The result is FLOAT unless the parameter is NULL, in which case the result is NULL.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

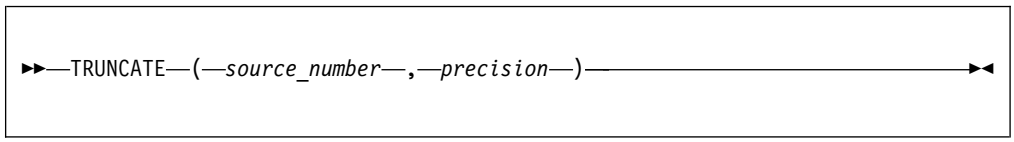
Related reference:

"Syntax diagrams" on page 3677

"ESQL numeric functions" on page 5183
A list of the numeric functions that ESQL supports.

TRUNCATE function:
The TRUNCATE numeric function truncates a supplied decimal number a specified number of places.

Syntax



If *precision* is positive, the result of the TRUNCATE function is *source_number* truncated to *precision* places right of the decimal point. If *precision* is negative, the result is *source_number* truncated to the absolute value of *precision* places to the left of the decimal point.

source_number can be any built-in numeric data type. *precision* must evaluate to an INTEGER. The result is of the same data type as *source_number*. If any parameter is NULL, the result is NULL.

For example:

```
TRUNCATE(27.75, 2)
```

returns `27.75`

```
TRUNCATE(27.75, 1)
```

returns `27.7`

```
TRUNCATE(27.75, 0)
```

returns `27.0`

```
TRUNCATE(27.75, -1)
```

returns `20.0`

If possible, the scale is changed to the given value. If the result cannot be represented within the given scale, it is INF.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"Syntax diagrams" on page 3677

"ESQL numeric functions" on page 5183
A list of the numeric functions that ESQL supports.

ESQL string manipulation functions:

A list of the ESQL string manipulation functions that you can use.

Most of the following functions manipulate all string data types (BIT, BLOB, and CHARACTER). Exceptions to this are UPPER, LOWER, LCASE, UCASE, and SPACE, which operate only on character strings.

In these descriptions, the term *singleton* refers to a single part (BIT, BLOB, or CHARACTER) within a string of that type.

In addition to the functions that are described here, you can use the logical OR operator to perform various calculations on ESQL string manipulation values.

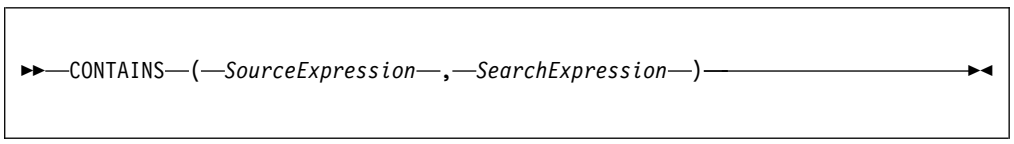
To concatenate two strings, use the "ESQL string operator" on page 5066.

This section covers the following topics:

CONTAINS function:

CONTAINS is a string manipulation function that manipulates all string data types (BIT, BLOB, and CHARACTER), and returns a Boolean value to indicate whether one string is present within another.

Syntax



CONTAINS returns TRUE if the *SearchExpression* is present within the *SourceExpression*, otherwise it returns FALSE.

The parameter strings for both *SourceExpression* and *SearchExpression* can be of the CHARACTER, BLOB, or BIT data type, but must be of the same data type.

If any parameter is NULL, the result is NULL.

Examples

```
CONTAINS('Hello World!', 'ello');
```

returns TRUE.

```
CONTAINS('Hello World!', 'daisy');
```

returns FALSE.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

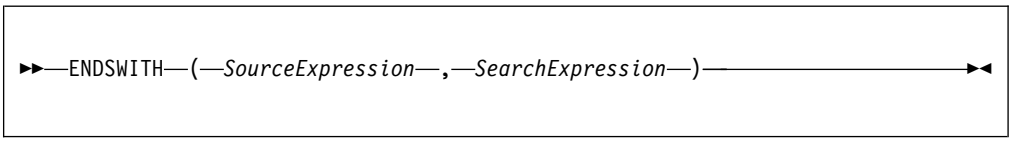
"Syntax diagrams" on page 3677

"ESQL string manipulation functions" on page 5205
A list of the ESQL string manipulation functions that you can use.

ENDSWITH function:

ENDSWITH is a string manipulation function that manipulates all string data types (BIT, BLOB, and CHARACTER), and returns a Boolean value to indicate whether one string ends with another.

Syntax



ENDSWITH returns TRUE if *SourceExpression* ends with *SearchExpression*, otherwise it returns FALSE.

The parameter strings for both *SearchExpression* and *SourceExpression* can be of the CHARACTER, BLOB, or BIT data type, but must be of the same data type.

If any parameter is NULL, the result is NULL.

Examples

```
ENDSWITH('Hello World!', 'World!');
```

returns TRUE.

```
ENDSWITH('Hello World!', 'World');
```

returns FALSE.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

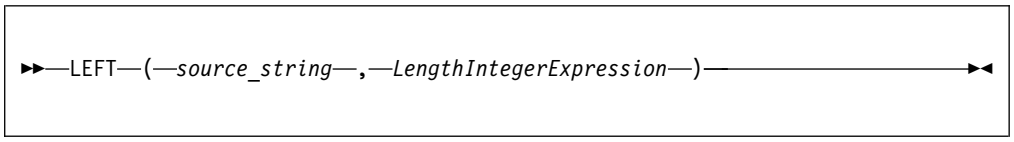
"Syntax diagrams" on page 3677

"ESQL string manipulation functions" on page 5205
A list of the ESQL string manipulation functions that you can use.

LEFT function:

LEFT is a string manipulation function that returns a string consisting of the source string truncated to the length given by the length expression.

Syntax



The source string can be of the CHARACTER, BLOB or BIT data type and the length must be of type INTEGER. The truncation discards the final characters of the *source_string*

The result is of the same type as the source string. If the length is negative or zero, a zero length string is returned. If either parameter is NULL, the result is NULL

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"Syntax diagrams" on page 3677

"ESQL string manipulation functions" on page 5205
A list of the ESQL string manipulation functions that you can use.

LENGTH function:

The LENGTH function is used for string manipulation on all string data types (BIT, BLOB, and CHARACTER) and returns an integer value giving the number of singletons in *source_string*.

Syntax

It If the *source_string* is NULL, the result is the NULL value. The term *singleton* refers to a single part (BIT, BYTE, or CHARACTER) within a string of that type.

For example:

```
LENGTH('Hello World!');
```

returns 12.

```
LENGTH('');
```

returns 0.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

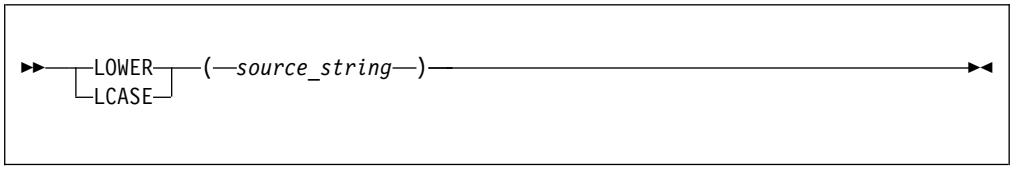
"Syntax diagrams" on page 3677

"ESQL string manipulation functions" on page 5205
A list of the ESQL string manipulation functions that you can use.

LOWER and LCASE functions:

The LOWER and LCASE functions are equivalent, and manipulate CHARACTER string data; they both return a new character string, which is identical to *source_string*, except that all uppercase letters are replaced with the corresponding lowercase letters.

Syntax



For example:

```
LOWER('Mr Smith')
```

returns `'mr smith'`.

```
LOWER('22 Railway Cuttings')
```

returns `'22 railway cuttings'`.

```
LCASE('ABCD')
```

returns `'abcd'`.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"Syntax diagrams" on page 3677

"ESQL string manipulation functions" on page 5205
A list of the ESQL string manipulation functions that you can use.

LTRIM function:

LTRIM is a string manipulation function, used for manipulating all data types (BIT, BLOB, and CHARACTER), that returns a character string value of the same data type and content as *source_string*, but with any leading default singletons removed.

Syntax

The term *singleton* is used to refer to a single part (BIT, BLOB, or CHARACTER) within a string of that type.

The LTRIM function is equivalent to TRIM(LEADING FROM *source_string*).

If the parameter is NULL, the result is NULL.

The default singleton depends on the data type of *source_string*:

Table 264.

Character	' ' (space)
BLOB	X'00'
Bit	B'0'

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

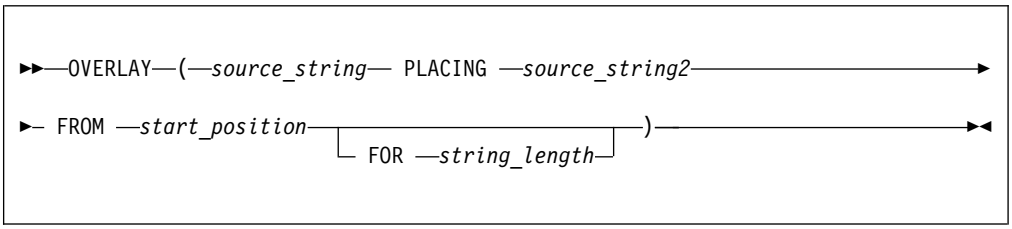
"Syntax diagrams" on page 3677

"ESQL string manipulation functions" on page 5205
A list of the ESQL string manipulation functions that you can use.

OVERLAY function:

OVERLAY is a string manipulation function that manipulates all string data types (BIT, BLOB, and CHARACTER) and replaces part of a string with a substring.

Syntax



OVERLAY returns a new string of the same type as the source and is identical to *source_string*, except that a given substring in the string, starting from the specified numeric position and of the given length, has been replaced by *source_string2*. When the length of the substring is zero, nothing is replaced.

For example:

```
OVERLAY ('ABCDEFGHIJ' PLACING '1234' FROM 4 FOR 3)
```

returns the string `'ABC1234GHIJ'`

If any parameter is NULL, the result is NULL. If *string_length* is not specified, it is assumed to be equal to `LENGTH(source_string2)`.

The result of the OVERLAY function is equivalent to:

```
SUBSTRING(source_string FROM 1 FOR start_position -1 )
  || source_string2 ||
  SUBSTRING(source_string FROM start_position + string_length)
```

where || is the concatenation operator.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

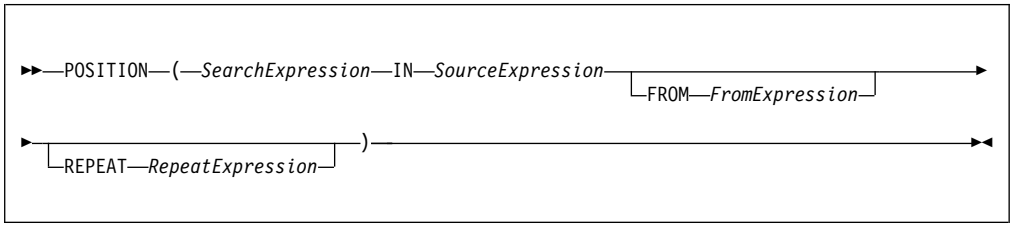
"Syntax diagrams" on page 3677

"ESQL string manipulation functions" on page 5205
A list of the ESQL string manipulation functions that you can use.

POSITION function:

POSITION is a string manipulation function that manipulates all data types (BIT, BLOB, and CHARACTER), and returns the position of one string within another.

Syntax



POSITION returns an integer giving the position of one string (*SearchExpression*) in a second string (*SourceExpression*). A position of one corresponds to the first character of the source string.

If present, the FROM clause gives a position within the search string at which the search commences. In the absence of a FROM clause, the source string is searched from the beginning.

If present, the REPEAT clause gives a repeat count, returning the position returned to be that of the nth occurrence of the search string within the source string. If the repeat count is negative, the source string is searched from the end.

In the absence of a REPEAT clause, a repeat count of +1 is assumed; that is, the position of the first occurrence, searching from the beginning is returned. If the search string has a length of zero, the result is one.

If the search string cannot be found, the result is zero: if the FROM clause is present, this applies only to the section of the source string being searched; if the REPEAT clause is present this applies only if there are insufficient occurrences of the string.

If any parameter is NULL, the result is NULL.

The search and source strings can be of the CHARACTER, BLOB, or BIT data types but they must be of the same type.

For example:

```
POSITION('Village' IN 'Hursley Village');   returns 9
POSITION('Town' IN 'Hursley Village');   returns 0

POSITION ('B' IN 'ABCABCABCABCABC'); -> returns 2
POSITION ('D' IN 'ABCABCABCABCABC'); -> returns 0

POSITION ('A' IN 'ABCABCABCABCABC' FROM 4); -> returns 4
POSITION ('C' IN 'ABCABCABCABCABC' FROM 2); -> returns 3

POSITION ('B' IN 'ABCABCABCABCABC' REPEAT 2); -> returns 5
POSITION ('C' IN 'ABCABCABCABCABC' REPEAT 4); -> returns 12

POSITION ('A' IN 'ABCABCABCABCABC' FROM 4 REPEAT 2); -> returns 7
POSITION ('AB' IN 'ABCABCABCABCABC' FROM 2 REPEAT 3); -> returns 10

POSITION ('A' IN 'ABCABCABCABCABC' REPEAT -2); -> returns 10
POSITION ('BC' IN 'ABCABCABCABCABC' FROM 2 REPEAT -3); -> returns 5
```

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Selecting a subfield from a larger field" on page 2442
You might have a message flow that processes a message containing delimited subfields. You can code ESQL to extract a subfield from the surrounding content if you know the delimiters of the subfield.

Related reference:

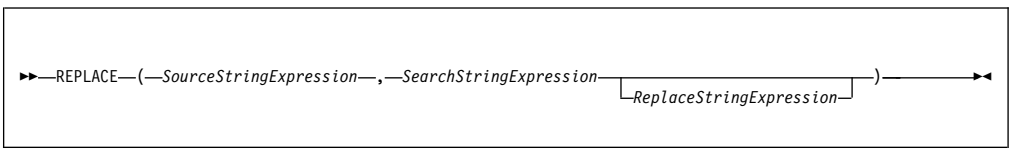
"Syntax diagrams" on page 3677

"ESQL string manipulation functions" on page 5205
A list of the ESQL string manipulation functions that you can use.

REPLACE function:

REPLACE is a string manipulation function that manipulates all string data types (BIT, BLOB, and CHARACTER), and replaces parts of a string with supplied substrings.

Syntax



REPLACE returns a string consisting of the source string, with each occurrence of the search string replaced by the replace string. The parameter strings can be of the CHARACTER, BLOB, or BIT data types, but all three must be of the same type.

If any parameter is NULL, the result is NULL.

The search process is single pass from the left and disregards characters that have already been matched.

If you do not specify the replace string expression, the replace string uses the default value of an empty string, and the behavior of the function is to delete all occurrences of the search string from the result.

The following examples give the results shown:

```
REPLACE('ABCDABCDABCDA', 'A', 'AA')
-- RESULT = AABCDAABCDAABCDAA
```

The above example shows that replacement is single pass. Each occurrence of A is replaced by AA but these are not then expanded further.

```
REPLACE('AAAABCDEFGHAAAABCDEFGH', 'AA', 'A')
-- RESULT = AABCDEFGHAABCDEFGH
```

This example shows that after characters are matched, they are not considered further. Each occurrence of AA is replaced by A. The resulting AA pairs are not matched.

```
REPLACE('AAAAABCDEFGHAAAABCDEFGH', 'AA', 'XYZ')
 -- RESULT = XYZXYZABCDEFGHXYZXYZBCDEFGH
```

This last example shows that matching is from the left. The first four As are matched as two pairs and replaced. The fifth A is not matched.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

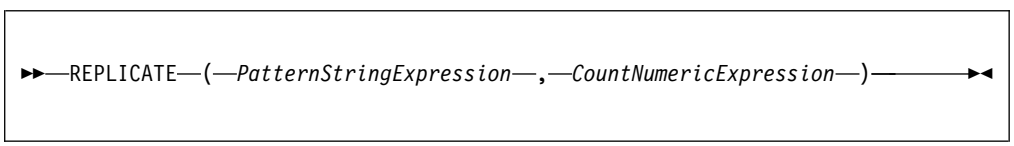
"Syntax diagrams" on page 3677

"ESQL string manipulation functions" on page 5205
A list of the ESQL string manipulation functions that you can use.

REPLICATE function:

REPLICATE is a string manipulation function that manipulates all data types (BIT, BLOB, and CHARACTER) and returns a string made up of multiple copies of a supplied string.

Syntax



REPLICATE returns a string consisting of the pattern string given by *PatternStringExpression* repeated the number of times given by *CountNumericExpression*.

The pattern string can be of the CHARACTER, BLOB, or BIT datatype and the count must be of type INTEGER. The result is of the same data type as the pattern string.

If the count is negative or zero, a zero length string is returned. If either parameter is NULL, the result is NULL.

The count is limited to 32*1024*1024 to protect the broker from erroneous programs. If this limit is exceeded, an exception condition is issued.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

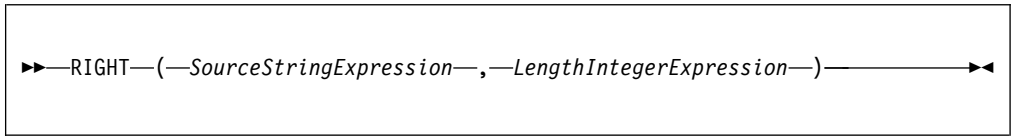
"Syntax diagrams" on page 3677

"ESQL string manipulation functions" on page 5205
A list of the ESQL string manipulation functions that you can use.

RIGHT function:

RIGHT is a string manipulation function that manipulates all data types (BIT, BLOB, and CHARACTER), and truncates a string.

Syntax



RIGHT returns a string consisting of the source string truncated to the length given by the length expression. The truncation discards the initial characters of the source string.

The source string can be of the CHARACTER, BLOB, or BIT data type and the length must be of type INTEGER.

If the length is negative or zero, a zero length string is returned. If either parameter is NULL, the result is NULL

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

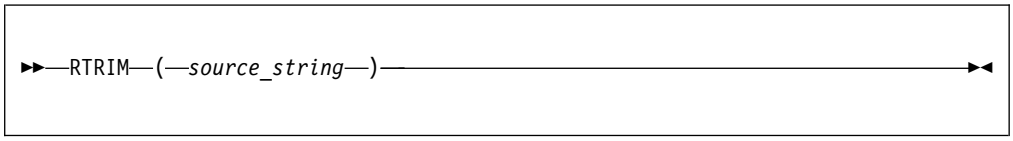
"Syntax diagrams" on page 3677

"ESQL string manipulation functions" on page 5205
A list of the ESQL string manipulation functions that you can use.

RTRIM function:

RTRIM is a string manipulation function that manipulates all data types (BIT, BLOB, and CHARACTER), and removes trailing singletons from a string.

Syntax



RTRIM returns a string value of the same data type and content as *source_string* but with any trailing default singletons removed. The term *singleton* refers to a single part (BIT, BLOB, or CHARACTER) within a string of that type.

The RTRIM function is equivalent to TRIM(TRAILING FROM *source_string*).

If the parameter is NULL, the result is NULL.

The default singleton depends on the data type of *source_string*:

Data type of source string	Default singleton
Character	' ' (space)
BLOB	X'00'
Bit	B'0'

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

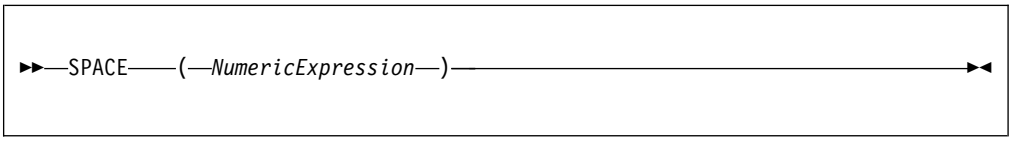
"Syntax diagrams" on page 3677

"ESQL string manipulation functions" on page 5205
A list of the ESQL string manipulation functions that you can use.

SPACE function:

SPACE is a string manipulation function that manipulates all data types (BIT, BLOB, and CHARACTER), and creates a string consisting of a defined number of blank spaces.

Syntax



SPACE returns a character string consisting of the number of blank spaces given by *NumericExpression*. The parameter must be of type INTEGER; the result is of type CHARACTER.

If the parameter is negative or zero, a zero length character string is returned. If the parameter is NULL, the result is NULL.

The string is limited to 32*1024*1024 to protect the broker from erroneous programs. If this limit is exceeded, an exception condition is issued.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

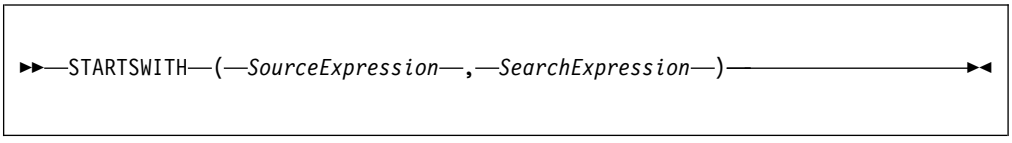
"Syntax diagrams" on page 3677

"ESQL string manipulation functions" on page 5205
A list of the ESQL string manipulation functions that you can use.

STARTSWITH function:

STARTSWITH is a string manipulation function that manipulates all string data types (BIT, BLOB, and CHARACTER), and returns a Boolean value to indicate whether one string begins with another.

Syntax



STARTSWITH returns TRUE if *SourceExpression* begins with *SearchExpression*, otherwise it returns FALSE.

The parameter strings for both *SearchExpression* and *SourceExpression* can be of the CHARACTER, BLOB, or BIT data type, but must be of the same data type.

If any parameter is NULL, the result is NULL.

Examples

```
STARTSWITH('Hello World!', 'Hello');
```

returns TRUE.

```
STARTSWITH('Hello World!', 'World');
```

returns FALSE.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

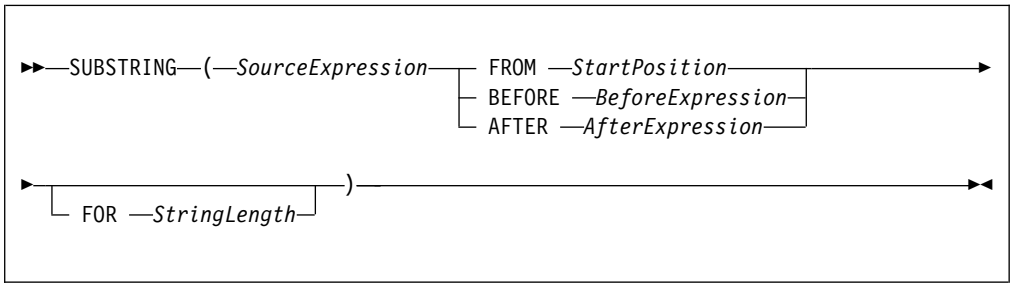
"Syntax diagrams" on page 3677

"ESQL string manipulation functions" on page 5205
A list of the ESQL string manipulation functions that you can use.

SUBSTRING function:

SUBSTRING is a string manipulation function that manipulates all string data types (BIT, BLOB, and CHARACTER), and extracts characters from a string to create another string.

Syntax



Parameters must be of the following types:

- *SourceExpression*, *BeforeExpression*, and *AfterExpression* can be BIT, BLOB, or CHARACTER.
- *StartPosition* and *StringLength* can be INTEGER only.

StartPosition

If you specify *StartPosition*, SUBSTRING returns a new string of the same type as *SourceExpression* containing one contiguous sequence of characters that are

extracted from *SourceExpression*, as specified by *StartPosition* and *StringLength*. If you do not specify *StringLength*, the sequence runs from *StartPosition* until the end of *SourceExpression*. The *StartPosition* can be negative, and together, the *StartPosition* and *StringLength* define a range. The result is the overlap between this range and the *SourceExpression*; the *StringLength* cannot be less than the *StartPosition*.

BeforeExpression

If you specify *BeforeExpression*, SUBSTRING returns a new string of the same type as *SourceExpression* containing one contiguous sequence of characters that are extracted from *StringLength* characters before the first occurrence of *BeforeExpression* within *SourceExpression*, up to (but not including) the first character of the first occurrence of *BeforeExpression*. If you do not specify *StringLength*, the sequence of characters is taken from the beginning of *SourceExpression* up to (but not including) the first character of the first occurrence of *BeforeExpression*. If the *BeforeExpression* string does not occur in *SourceExpression*, an empty (zero length) string is returned.

The *BeforeExpression* string must be of the same data type as *SourceExpression*.

AfterExpression

If you specify *AfterExpression*, SUBSTRING returns a new string of the same type as *SourceExpression*, containing one contiguous sequence of characters that are extracted from *SourceExpression*, beginning with the first character after the end of the first occurrence of *AfterExpression* until the end of *SourceExpression* (or *StringLength* characters, if specified). If the *AfterExpression* string does not occur in *SourceExpression*, an empty (zero length) string is returned.

The *AfterExpression* string must be of the same data type as *SourceExpression*.

If any parameter is NULL, the result is NULL. This is not a zero length string.

Examples:

```
SUBSTRING('Hello World!' FROM 7 FOR 4)
```

returns `'Worl'`.

```
SUBSTRING('Hello World!' BEFORE 'World');
```

returns `'Hello '`.

```
SUBSTRING('Hello World!' BEFORE 'World' FOR 3);
```

returns `'lo '`.

```
SUBSTRING('Hello World!' BEFORE 'e');
```

returns `'H'`.

```
SUBSTRING('Hello World!' AFTER 'World');
```

returns `'!'`.

```
SUBSTRING('Hello World!' AFTER 'W' FOR 2);
```

returns `'or'`.

```
SUBSTRING('Hello World!' AFTER 'P');
```

returns `''`.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Selecting a subfield from a larger field" on page 2442
You might have a message flow that processes a message containing delimited subfields. You can code ESQL to extract a subfield from the surrounding content if you know the delimiters of the subfield.

"Manipulating messages in the BLOB domain" on page 2615
How to deal with messages that belong to the BLOB domain, and that are parsed by the BLOB parser.

Related reference:

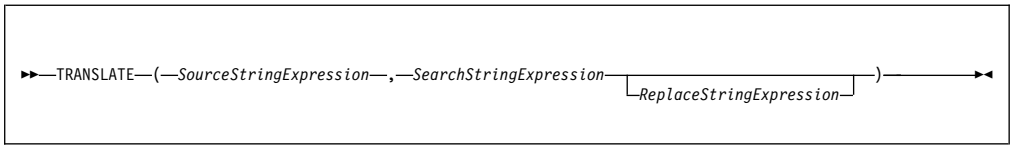
"Syntax diagrams" on page 3677

"ESQL string manipulation functions" on page 5205
A list of the ESQL string manipulation functions that you can use.

TRANSLATE function:

TRANSLATE is a string manipulation function that manipulates all string data types (BIT, BLOB, and CHARACTER), and replaces specified characters in a string.

Syntax



TRANSLATE returns a string consisting of the source string, with each occurrence of any character that occurs in the search string being replaced by the corresponding character from the replace string.

The parameter strings can be of the CHARACTER, BLOB, or BIT data type but all three must be of the same type. If any parameter is NULL, the result is NULL.

If the replace string is shorter than the search string, there are characters in the search string for which there is no corresponding character in the replace string. This is treated as an instruction to delete these characters and any occurrences of these characters in the source string are absent from the returned string

If the replace string expression is not specified, the replace string is assumed to be an empty string, and the function deletes all occurrences of any characters in the search string from the result.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

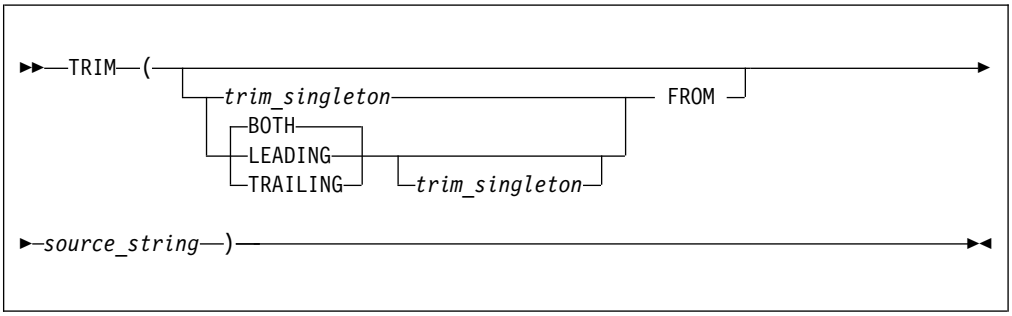
"Syntax diagrams" on page 3677

"ESQL string manipulation functions" on page 5205
A list of the ESQL string manipulation functions that you can use.

TRIM function:

TRIM is a string manipulation function that manipulates all string data types (BIT, BLOB, and CHARACTER), and removes trailing and leading singletons from a string.

Syntax



TRIM returns a new string of the same type as *source_string*, in which the leading, trailing, or both leading and trailing singletons have been removed. The term *singleton* refers to a single part (BIT, BYTE, or CHARACTER) within a string of that type.

The singleton can contain a list of multiple characters to be trimmed from the source string.

If *trim_singleton* is not specified, a default singleton is assumed. The default singleton depends on the data type of *source_string*:

Character	**' ' (space)**
BLOB	X'00'
Bit	B'0'

If any parameter is NULL, the result is NULL.

It is often unnecessary to strip trailing blanks from character strings before comparison, because the rules of character string comparison mean that trailing blanks are not significant.

The following examples illustrate the behavior of the TRIM function:

```
TRIM(TRAILING 'b' FROM 'aaabBb')
```

returns `'aaabB'`.

```
TRIM('  a  ')
```

returns 'a'.

```
TRIM(LEADING FROM '  a  ')
```

returns 'a '.

```
TRIM('b' FROM 'bbbaaabbb')
```

returns 'aaa'.

An example of using a multiple character singleton is:

```
DECLARE input1 CHARACTER 'testmgktest';
SET OutputRoot.XMLNSC.Top.Out1 = TRIM( 'ste' FROM input1);
```

The preceding code produces the output message:

```
<Top><Out1>mgk</Out1></Top>
```

An example of using a multiple character singleton to remove leading and trailing white space characters is:

```
DECLARE whiteSpace CONSTANT CHARACTER CAST( X'090D0A20' AS CHAR CCSID 1208);
 /* tab, cr, lf, space */
DECLARE input2 CHARACTER 'smith';

SET input2 = whiteSpace || input2 || whiteSpace;
SET OutputRoot.XMLNSC.Top.Out2 = TRIM( whiteSpace FROM input2);
```

The preceding code produces the output message:

```
<Top><Out2>smith</Out2></Top>
```

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Selecting a subfield from a larger field" on page 2442
You might have a message flow that processes a message containing delimited subfields. You can code ESQL to extract a subfield from the surrounding content if you know the delimiters of the subfield.

Related reference:

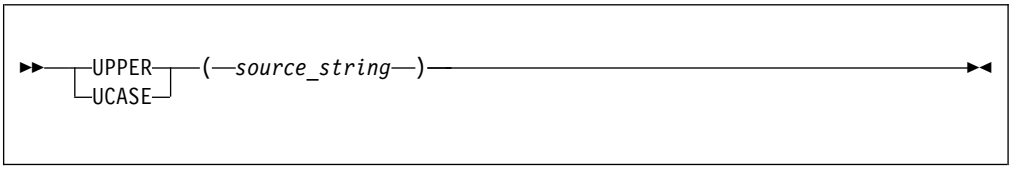
"Syntax diagrams" on page 3677

"ESQL string manipulation functions" on page 5205
A list of the ESQL string manipulation functions that you can use.

UPPER and UCASE functions:

UPPER and UCASE are equivalent string manipulation functions that manipulate CHARACTER string data and convert lowercase characters in a string to uppercase.

Syntax



UPPER and UCASE both return a new character string, which is identical to *source_string*, except that all lowercase letters are replaced with the corresponding uppercase letters.

For example:

```
UPPER('ABCD')
```

returns `'ABCD'`.

```
UCASE('abc123')
```

returns `'ABC123'`.

Converting characters from different code pages to uppercase

If you are using certain code pages, characters with no uppercase equivalent in your code page might be converted when you use the UPPER or UCASE function. This conversion happens because the bit stream is converted to a Unicode message tree by the message parser. Even though characters might have no uppercase equivalent in the source code page, they can still have an uppercase equivalent in the Unicode code page, and are converted by the UPPER or UCASE function. When the bit stream is converted back to the original code page, these characters cannot be converted back, and a substitution character is inserted into the output message for each character. The substitution character inserted depends on the original code page. For example, conversion to an EBCDIC code page inserts an X'3F' byte and conversion to a Japanese code page inserts an X'7F' byte.

A solution to this problem is to use the TRANSLATE function to convert selected characters to uppercase, instead of using the UPPER or UCASE function. Any characters that have no uppercase equivalent in the code page are excluded from the conversion.

In the following example, the input message is in code page 284, and the `InputRoot.XML.MSG.APPDATA` element contains characters that do not have an uppercase equivalent in code page 284, but do have uppercase equivalents in the Unicode code page. The TRANSLATE function is used to convert only the lowercase characters 'a' to 'z' to their equivalent uppercase characters. Any other characters in `InputRoot.XML.MSG.APPDATA` are excluded from the conversion.

```
DECLARE char1 CHAR;
SET char1 = TRANSLATE(InputRoot.XML.MSG.APPDATA,'abcdefghijklmnopqrstuvwxyz','ABCDEFGHIJKLMNOPQRSTUVWXYZ');
SET OutputRoot.MQMD.CodedCharSetId = 284;
SET OutputRoot.XML.TEST.translated = char1;
```

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"TRANSLATE function" on page 5220
TRANSLATE is a string manipulation function that manipulates all string data types (BIT, BLOB, and CHARACTER), and replaces specified characters in a string.

"Syntax diagrams" on page 3677

"ESQL string manipulation functions" on page 5205
A list of the ESQL string manipulation functions that you can use.

ESQL field functions:

ESQL provides functions that support field operations.

"ASBITSTREAM function"

"BITSTREAM function (deprecated)" on page 5228

"FIELDNAME function" on page 5229

"FIELDNAMESPACE function" on page 5230

"FIELDTYPE function" on page 5231

"FIELDVALUE function" on page 5234

"FOR function" on page 5235

"LASTMOVE function" on page 5237

"SAMEFIELD function" on page 5237

Related tasks:

"Writing ESQL" on page 2413
How you can use ESQL to customize nodes.

Related reference:

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"ESQL operators" on page 5056
A list of the various groups of operators that ESQL supports.

"ESQL statements" on page 5067
You can use ESQL statements to manipulate message trees, update databases, or interact with nodes.

"ESQL functions" on page 5168
The following types of function are available.

ASBITSTREAM function:

The ASBITSTREAM field function generates a bit stream for the subtree of a given field according to the rules of the parser that owns the field.

The ASBITSTREAM field function uses parameters supplied by the caller for:

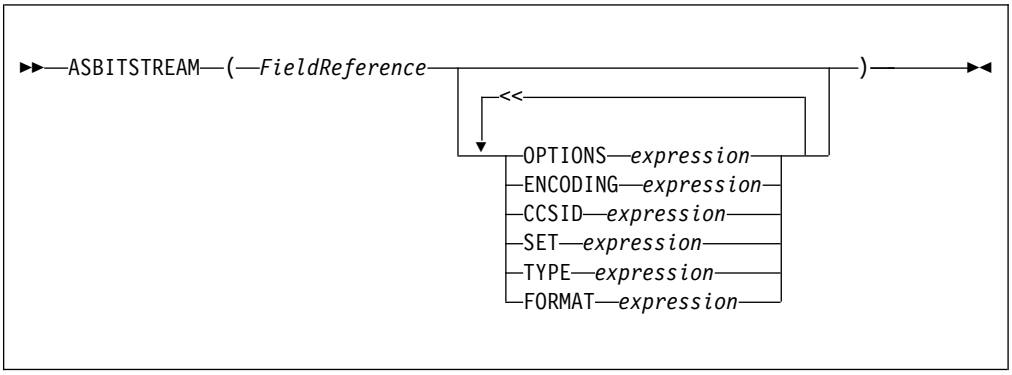
- Encoding

- CCSID
- Message set
- Message type
- Message format
- Options

The ASBITSTREAM function removes the limitation of the existing BITSTREAM function, which can be used only on a tree produced by a parser that belongs to an input node.

The BITSTREAM function is retained only for compatibility with earlier versions.

Syntax



Note that each clause can occur once only.

ASBITSTREAM returns a value of type BLOB that contains a bitstream representation of the field that is pointed to by *FieldReference* and its children.

The algorithm for doing this varies from parser to parser, and according to the options specified. All parsers support the following modes:

- RootBitStream, in which the algorithm that generates the bit stream is the same as the algorithm that is used by an output node. In this mode, a meaningful result is obtained only if the field pointed to is at the head of a subtree with an appropriate structure.
- EmbeddedBitStream, in which not only is the algorithm that generates the bit stream the same as the algorithm used by an output node, but also the
  - Encoding
  - CCSID
  - Message set
  - Message type
  - Message format

  are determined, if not explicitly specified, in the same way as the output node. That is, they are determined by searching the previous siblings of *FieldReference* on the assumption that they represent headers.

  In this way, the algorithm for determining these properties is essentially the same as that used for the BITSTREAM function.

Some parsers also support another mode, FolderBitStream, which generates a meaningful bit stream for any subtree, provided that the field that is pointed to represents a folder.

In all cases, the bit stream obtained can be given to a CREATE statement with a PARSE clause, using the same DOMAIN and OPTIONS to reproduce the original subtree.

When the function is called, any clause expressions are evaluated. An exception is thrown if any of the expressions do not result in a value of the appropriate type.

If any parameter is NULL the result is NULL.

Clause	Type	Default value
OPTIONS	Integer	RootBitStream & ValidateNone
ENCODING	Integer	0
CCSID	Integer	0
SET	Character	Zero length string
TYPE	Character	Zero length string
FORMAT	Character	Zero length string

For details of the syntax of the TYPE clause, refer to "Specifying namespaces in the Message Type property" on page 1208.

Although the OPTIONS clause accepts any expression that returns a value of type integer, it is only meaningful to generate option values from the list of supplied constants, using the BITOR function if more than one option is required.

The generated value becomes an integer and can be saved in a variable, passed as a parameter to a function, or used directly in an ASBITSTREAM call. The list of globally-defined constants is:

```
Validate master options...
ValidateContentAndValue
ValidateValue    -- Can be used with ValidateContent
ValidateContent  -- Can be used with ValidateValue
ValidateNone

Validate failure action options...
ValidateException
ValidateExceptionList
ValidateLocalError
ValidateUserTrace
```

Note:

1. For full details of the validation options, refer to "Validation properties" on page 4169.

C and Java equivalent APIs

Note that equivalent options are not available on:

- The Java plugin node API MBElement methods **createElementAsLastChildFromBitstream()** and **toBitstream()**
- The C plugin node API methods **cniCreateElementAsLastChildFromBitstream()** and **cniElementAsBitstream**.

Only one option from each group can be specified, with the exception of ValidateValue and ValidateContent, which can be used together to obtain the content and value validation. If you do not specify an option within a group, the option in bold is used.

The ENCODING clause accepts any expression that returns a value of type integer. However, it is only meaningful to generate encoding values from the list of supplied constants:

```
0
MQENC_INTEGER_NORMAL
MQENC_INTEGER_REVERSED
MQENC_DECIMAL_NORMAL
MQENC_DECIMAL_REVERSED
MQENC_FLOAT_IEEE_NORMAL
MQENC_FLOAT_IEEE_REVERSED
MQENC_FLOAT_S390
```

`0` uses the queue manager's encoding.

The values that are used for the CCSID clause follow the normal numbering system. For example, 1200 = UCS-2, 1208 = UTF-8.

In addition the special values 0 and -1 are supported:

- `0` uses the queue manager's CCSID
- `-1` uses the CCSID's as determined by the parser itself. This value is reserved.

For absent clauses, the given default values are used. Use the CCSID and encoding default values, because they take their values from the queue manager's encoding and CCSID settings.

Similarly, use the default values for each of the message set, type, and format options, because many parsers do not require message set, type, or format information; any valid value is sufficient.

When any expressions have been evaluated, the appropriate bit stream is generated.

Note: Because this function has a large number of clauses, an alternative syntax is supported in which the parameters are supplied as a comma-separated list rather than by named clauses. In this case, the expressions must be in the following order:

```
ENCODING -> CCSID -> SET -> TYPE -> FORMAT -> OPTIONS
```

The list can be truncated at any point and you can use an empty expression for any clauses for which you do not supply a value.

Examples

```
DECLARE options INTEGER BITOR(FolderBitStream, ValidateContent,
                              ValidateValue);
SET result = ASBITSTREAM(cursor OPTIONS options CCSID 1208);
SET Result = ASBITSTREAM(Environment.Variables.MQRFH2.Data,,1208
                         ,,,,options);
```

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message

flow.

"Specifying namespaces in the Message Type property" on page 1208
When using the MRM domain, the *Message Type* property is used to specify the name of the message.

"Multipart messages" on page 1191
A *multipart message* contains one or more other messages within its structure. The contained message is sometimes referred to as an *embedded message*.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Working with large XML messages" on page 2543
The tree representation of an XML message is typically bigger than the input bit stream. Manipulating a large message tree can require much storage but you can code ESQL statements that help to reduce the storage load on the broker.

"XMLNSC: Working with XML messages and bit streams" on page 2541
Use the ASBITSTREAM function and the CREATE statement to manage XML message content.

Related reference:

"Syntax diagrams" on page 3677

"ESQL field functions" on page 5224
ESQL provides functions that support field operations.

"BITSTREAM function (deprecated)"
The BITSTREAM field function returns a value that represents the bit stream that is described by the given field and its children. Its use is deprecated; use the newer ASBITSTREAM function instead.

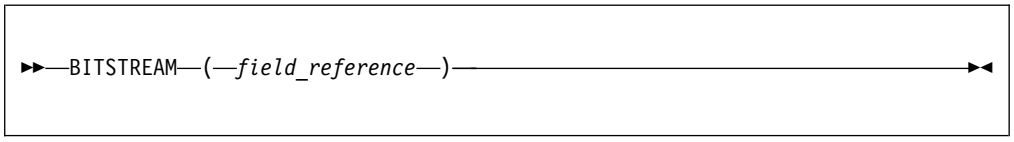
"Supported code pages" on page 4176
Application messages must conform to supported code pages.

BITSTREAM function (deprecated):

The BITSTREAM field function returns a value that represents the bit stream that is described by the given field and its children. Its use is deprecated; use the newer ASBITSTREAM function instead.

The BITSTREAM function can be used only on a tree produced by a parser belonging to an input node. The ASBITSTREAM function does not suffer from this limitation.

Syntax



The BITSTREAM function returns a value of type BLOB that represents the bit stream that is described by the given field and its children. For incoming messages, the appropriate portion of the incoming bit stream is used. For messages that are constructed by Compute nodes, the following algorithm is used to establish the ENCODING, CCSID, message set, message type, and message format:

- If the addressed field has a previous sibling, and this sibling is the root of a subtree that belongs to a parser capable of providing an ENCODING and CCSID, these values are obtained and used to generate the requested bit stream. Otherwise, the broker's default ENCODING and CCSID (that is, those of its queue manager) are used.
- Similarly, if the addressed field has a previous sibling, and this sibling is the root of a subtree that belongs to a parser capable of providing a message set, message type, and message format, these values are obtained and used to generate the requested bit stream. Otherwise, zero length strings are used.

This function is typically used for message warehouse scenarios, where the bit stream of a message needs to be stored in a database. The function returns the bit stream of the physical portion of the incoming message, identified by the parameter. In some cases, it does not return the bit stream that represents the actual field identified. For example, the following two calls return the same value:

```
BITSTREAM(Root.MQMD);
BITSTREAM(Root.MQMD.UserIdentifier);
```

because they lie in the same portion of the message.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

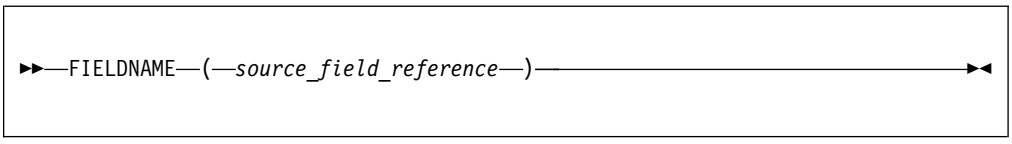
"Syntax diagrams" on page 3677

"ESQL field functions" on page 5224
ESQL provides functions that support field operations.

"ASBITSTREAM function" on page 5224
The ASBITSTREAM field function generates a bit stream for the subtree of a given field according to the rules of the parser that owns the field.

FIELDNAME function:
The FIELDNAME field function returns the name of a given field.

Syntax



FIELDNAME returns the name of the field identified by *source_field_reference* as a character value. If the parameter identifies a nonexistent field, NULL is returned.

For example:

- `FIELDNAME(InputRoot.XMLNS)` returns XMLNS.

- `FIELDNAME(InputBody)` returns the name of the last child of InputRoot, which could be XMLNS.
- `FIELDNAME(InputRoot.*[<])` returns the name of the last child of InputRoot, which could be XMLNS.

This function does not show any namespace information; this must be obtained by a separate call to the "FIELDNAMESPACE function."

Whereas the following ESQL sets X to "F1":

```
SET X=FIELDNAME(InputBody.*[<]);
```

The following ESQL sets Y to null:

```
SET Y=FIELDNAME(InputBody.F1.*[<]);
```

However, the following ESQL sets Z to the (expected) child of F1:

```
SET Z=FIELDNAME(InputBody.*[<].*[<]);
```

This is because F1 belongs to a namespace and needs to be explicitly referenced by, for example:

```
DECLARE ns NAMESPACE 'urn:nid:xxxxxx';

SET Y=FIELDNAME(InputBody.ns:F1.*[<]);
```

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

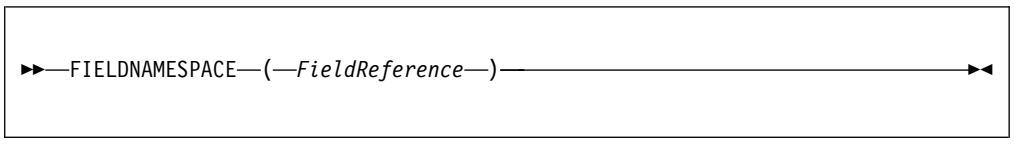
Related reference:

"Syntax diagrams" on page 3677

"ESQL field functions" on page 5224
ESQL provides functions that support field operations.

FIELDNAMESPACE function:
The FIELDNAMESPACE field function returns the namespace of a given field.

Syntax



FIELDNAMESPACE takes a field reference as a parameter and returns a value of type CHARACTER containing the namespace of the addressed field. If the parameter identifies a nonexistent field, NULL is returned.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

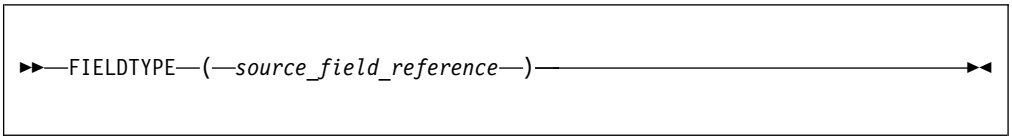
"Syntax diagrams" on page 3677

"ESQL field functions" on page 5224
ESQL provides functions that support field operations.

FIELDTYPE function:

The FIELDTYPE field function returns the type of a given field.

Syntax



FIELDTYPE returns an integer representing the type of the field identified by *source_field_reference*; this is the type of the field, not the data type of the field that the parameter identifies. If the parameter identifies a nonexistent entity, NULL is returned.

The mapping of integer values to field types is not published, and might change from release to release. Compare the results of the FIELDTYPE function with named field types.

For example:

```
IF FIELDTYPE(source_field_reference) = NameValue
 THEN ...
```

The named field types that you can use in this context are listed in this section. You must use these types with the capitalization shown.

The following types are domain-independent:

- *Name*
- *Value*
- *NameValue*
- *MQRFH2.BitStream*
- *MQRFH2.Field*
- *MQRFH2C.Field*

The *XML.** types are applicable to the XML, XMLNS, JMSMap, and JMSStream domains, except for *XML.NamespaceDecl*, which is specific to the XML domain.

- *XML.AsisElementContent*
- *XML.Attribute*

- *XML.AttributeDef*
- *XML.AttributeDefDefaultType*
- *XML.AttributeDefType*
- *XML.AttributeDefValue*
- *XML.AttributeList*
- *XML.BitStream*
- *XML.CDataSection*
- *XML.Comment*
- *XML.Content*
- *XML.DocTypeCommentt*
- *XML.DocTypeDecl*
- *XML.DocTypePI*
- *XML.DocTypeWhiteSpace*
- *XML.Element*
- *XML.ElementDef*
- *XML.Encoding*
- *XML.EntityDecl*
- *XML.EntityDeclValue*
- *XML.EntityReferenceEnd*
- *XML.EntityReferenceStart*
- *XML.EntityValue*
- *XML.ExternalEntityDecl*
- *XML.ExternalParameterEntityDecl*
- *XML.ExtSubset*
- *XML.IntSubset*
- *XML.NamespaceDecl*
- *XML.NotationDecl*
- *XML.NotationReference*
- *XML.Opaque*
- *XML.ParameterEntityDecl*
- *XML.ParserRoot*
- *XML.ProcessingInstruction*
- *XML.PublicId*
- *XML.RequestedDomain*
- *XML.Standalone*
- *XML.SystemId*
- *XML.UnparsedEntityDecl*
- *XML.Version*
- *XML.WhiteSpace*
- *XML.XmlDecl*

The *XMLNSC.** types are applicable to the XMLNSC domain. The same constants can also be prefixed with *SOAP.** for use in the SOAP domain.

- *XMLNSC.AnyCData*
- *XMLNSC.AnyHybrid*

- *XMLNSC.AnyPCData*
- *XMLNSC.AnyValue*
- *XMLNSC.AsisElementContent*
- *XMLNSC.Attribute*
- *XMLNSC.base64Binary*
- *XMLNSC.BitStream*
- *XMLNSC.CDataField*
- *XMLNSC.CDataValue*
- *XMLNSC.Comment*
- *XMLNSC.DocumentType*
- *XMLNSC.DoubleAttribute*
- *XMLNSC.DoubleEntityDefinition*
- *XMLNSC.DoubleNamespaceDecl*
- *XMLNSC.Element*
- *XMLNSC.EntityDefinition*
- *XMLNSC.EntityReference*
- *XMLNSC.Field*
- *XMLNSC.Folder*
- *XMLNSC.gDay*
- *XMLNSC.gMonth*
- *XMLNSC.gMonthDay*
- *XMLNSC.gYear*
- *XMLNSC.gYearMonth*
- *XMLNSC.HybridField*
- *XMLNSC.HybridValue*
- *XMLNSC.NamespaceDecl*
- *XMLNSC.Opaque*
- *XMLNSC.PCDataField*
- *XMLNSC.PCDataValue*
- *XMLNSC.ProcessingInstruction*
- *XMLNSC.SingleAttribute*
- *XMLNSC.SingleEntityDefinition*
- *XMLNSC.SingleNamespaceDecl*
- *XMLNSC.Value*
- *XMLNSC.XmlDeclaration*

You can also use this function to determine whether a field in a message exists. To do this, use the form:

```
FIELDTYPE(SomeFieldReference) IS NULL
```

If the field exists, an integer value is returned to the function that indicates the field type (for example, string). When this is compared to NULL, the result is FALSE. If the field does not exist, NULL is returned and therefore the result is TRUE. For example:

```
IF FIELDTYPE(InputRoot.XMLNS.Message1.Name)
    IS NULL THEN
// Name field does not exist, take error
action....
```

```
... more ESQL ...
ELSE
// Name field does exist, continue....
... more ESQL ...
END IF
```

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"Syntax diagrams" on page 3677

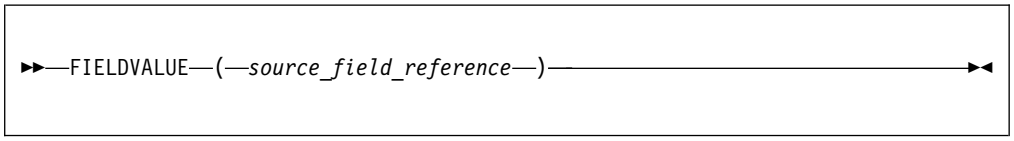
"ESQL field functions" on page 5224
ESQL provides functions that support field operations.

"XML, MRM, and XMLNSC parser constants" on page 6691
The names of the XML and MRM parser constants, together with their corresponding values, and a link to the XMLNSC constants.

"XMLNSC: Using field types" on page 1094
The XMLNSC parser sets the field type on every syntax element that it creates.

FIELDVALUE function:
The FIELDVALUE field function returns the scalar value of a given field.

Syntax



FIELDVALUE returns the scalar value of the field identified by *source_field_reference*. If it identifies a non-existent field, NULL is returned.

For example, consider the following XML input message:

```
<Data>
   <Qty Unit="Gallons">1234</Qty>
</Data>
```

The ESQL statement

```
SET OutputRoot.XML.Data.Quantity =
    FIELDVALUE(InputRoot.XML.Data.Qty);
```

gives the result:

```
<Data><Quantity>1234</Quantity></Data>
```

whereas this ESQL statement (without the FIELDVALUE function):

```
SET OutputRoot.XML.Data.Quantity =
    InputRoot.XML.Data.Qty;
```

causes a tree copy, with the result:

```
<Data><Quantity Unit="Gallons">1234</Quantity></Data>
```

because the field `Qty` is not a leaf field.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"XMLNSC: Attributes and elements" on page 2552
The XMLNSC parser uses field types to represent attributes and elements.

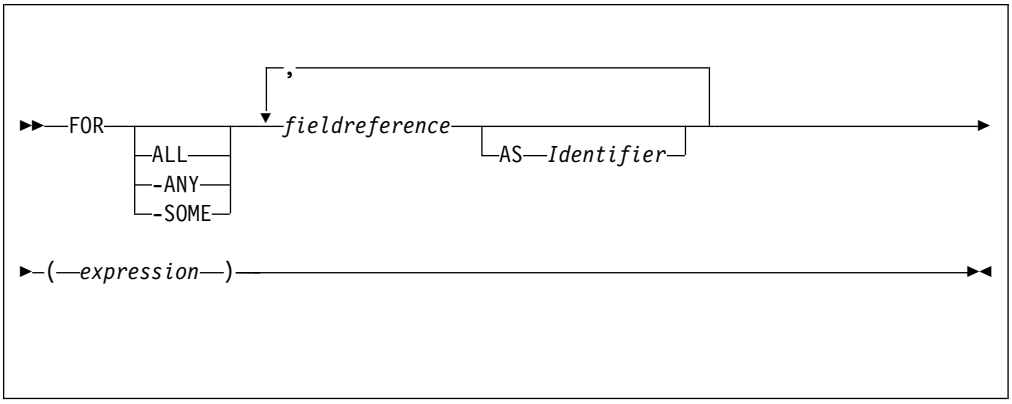
"Syntax diagrams" on page 3677

"ESQL field functions" on page 5224
ESQL provides functions that support field operations.

FOR function:

The FOR field function evaluates an expression and assigns a resulting value of TRUE, FALSE, or UNKNOWN.

Syntax



FOR enables you to write an expression that iterates over all instances of a repeating field. For each instance it processes a boolean expression and collates the results.

For example:

```
FOR ALL Body.Invoice.Purchases."Item"[] AS I (I.Quantity <= 50)
```

Note:

1. With the quantified predicate , the first thing to note is the [] on the end of the field reference after the FOR ALL. The square brackets define iteration over all instances of the `Item` field.

In some cases, this syntax appears unnecessary, because you can get that information from the context, but it is done for consistency with other pieces of syntax.

2. The ASclause associates the name `I` in the field reference with the current instance of the repeating field. This is similar to the concept of iterator classes used in some object oriented languages such as C++. The expression in parentheses is a predicate that is evaluated for each instance of the `Item` field.

If you specify the **ALL** keyword, the function iterates over all instances of the field Item inside `Body.Invoice.Purchases` and evaluates the predicate `I.Quantity <= 50`. If the predicate evaluates to:

- TRUE (if the field is empty, or for all instances of `Item`) return TRUE.
- FALSE (for any instance of `Item`) return FALSE.
- Anything else, return UNKNOWN.

The **ANY** and **SOME** keywords are equivalent. If you use either, the function iterates over all instances of the field Item inside `Body.Invoice.Purchases` and evaluates the predicate `I.Quantity <= 50`. If the predicate evaluates to:

- FALSE (if the field is empty, or for all instances of `Item`) return FALSE.
- TRUE (for any instance of `Item`) return TRUE.
- Anything else, return UNKNOWN.

To further illustrate this, the following examples are based on the message described in "Example message" on page 5311. In the following filter expression:

```
FOR ANY Body.Invoice.Purchases."Item"[] AS I (I.Title = 'The XML Companion')
```

the sub-predicate evaluates to TRUE. However, this next expression returns FALSE:

```
FOR ANY Body.Invoice.Purchases."Item"[] AS I (I.Title = 'C Primer')
```

because the C Primer is not included on this invoice. If in this instance some of the items in the invoice do not include a book title field, the sub-predicate returns UNKNOWN, and the quantified predicate returns the value UNKNOWN.

Take great care to deal with the possibility of null values appearing. Write this filter with an explicit check on the existence of the field, as follows:

```
FOR ANY Body.Invoice.Purchases."Item"[] AS I (I.Book IS NOT NULL AND
I.Book.Title = 'C Primer')
```

The IS NOT NULL predicate ensures that, if an `Item` field does not contain a `Book`, a FALSE value is returned from the sub-predicate.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Accessing unknown multiple occurrences of an element" on page 2427
To access repeating fields in a message, you must use a construct that can iterate

over all instances of a repeating field.

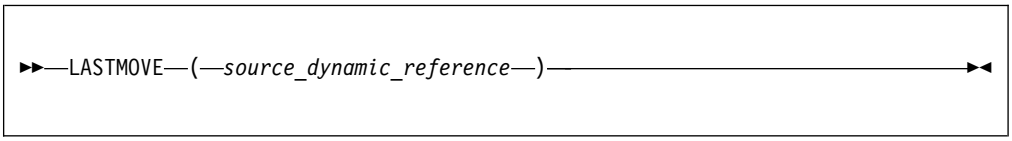
Related reference:

"Syntax diagrams" on page 3677

"Example message" on page 5311

LASTMOVE function:
The LASTMOVE field function tells you whether the last MOVE function succeeded.

Syntax



LASTMOVE returns a Boolean value indicating whether the last MOVE function applied to *source_dynamic_reference* was successful (TRUE) or not (FALSE).

See "MOVE statement" on page 5145 for an example of using the MOVE statement, and the LASTMOVE function to check its success.

See "Creating dynamic field references" on page 2431 for information about dynamic references.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

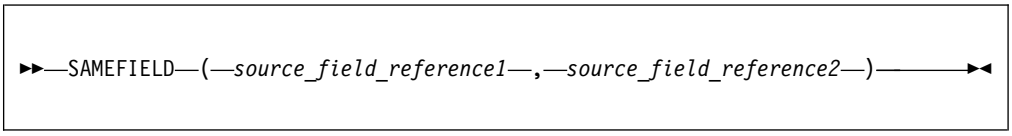
"Syntax diagrams" on page 3677

"ESQL field functions" on page 5224
ESQL provides functions that support field operations.

"MOVE statement" on page 5145
The MOVE statement changes the field to which the *Target* reference variable points.

SAMEFIELD function:
The SAMEFIELD field function tells you whether two field references point to the same target.

Syntax



SAMEFIELD returns a BOOLEAN value indicating whether two field references point to the same target. If they do, SAMEFIELD returns TRUE; otherwise SAMEFIELD returns FALSE.

For example:

```
DECLARE ref1 REFERENCE TO OutputRoot.XMLNS.Invoice.Purchases.Item[1];
MOVE ref1 NEXTSIBLING;
SET Result = SAMEFIELD(ref1,OutputRoot.XMLNS.Invoice.Purchases.Item[2]);
```

`Result` is `TRUE`.

See "Creating dynamic field references" on page 2431 for information about dynamic references.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"Syntax diagrams" on page 3677

"ESQL field functions" on page 5224
ESQL provides functions that support field operations.

"MOVE statement" on page 5145
The MOVE statement changes the field to which the *Target* reference variable points.

ESQL list functions:

ESQL provides several functions to work with lists.

"CARDINALITY function"

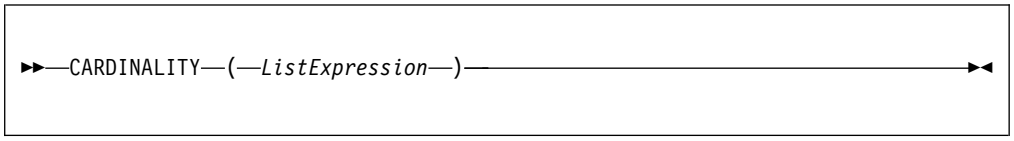
"EXISTS function" on page 5239

"SINGULAR function" on page 5241

"THE function" on page 5242

CARDINALITY function:
The CARDINALITY function returns the number of elements in a list.

Syntax



CARDINALITY returns an integer value giving the number of elements in the list specified by *ListExpression*.

ListExpression is any expression that returns a list. All the following, for example, return a list:
- A LIST constructor
- A field reference with the [] array indicator
- Some SELECT expressions (not all return a list)

A common use of this function is to determine the number of fields in a list before iterating over them.

Examples

```
-- Determine the number of F1 fields in the message.
-- Note that the [ ] are required
DECLARE CountF1 INT CARDINALITY(OutputRoot.XMLNS.Data.Source.F1[]);

-- Determine the number of fields called F1 with the value 'F12' in the message.
-- Again note that the [ ] are required
DECLARE CountF1F12 INT
  CARDINALITY(SELECT F.* FROM OutputRoot.XMLNS.Data.Source.F1[] AS F
              where F = 'F12');

-- Use the value returned by CARDINALITY to refer to a specific element
-- in a list or array:
-- Array indices start at 1, so this example refers to the third-from-last
-- instance of the Item field
Body.Invoice.Item[CARDINALITY(Body.Invoice.Item[]) - 2].Quantity
```

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"EXISTS function"
The EXISTS function returns a Boolean value to indicate whether a list contains at least one element (that is, whether the list exists).

"SINGULAR function" on page 5241
The SINGULAR function returns a Boolean value that indicates whether a list contains exactly one element.

"THE function" on page 5242
The THE function returns the first element of a list.

"Syntax diagrams" on page 3677

"ESQL list functions" on page 5238
ESQL provides several functions to work with lists.

EXISTS function:

The EXISTS function returns a Boolean value to indicate whether a list contains at least one element (that is, whether the list exists).

Syntax

If the list specified by *ListExpression* contains one or more elements, EXISTS returns TRUE. If the list contains no elements, EXISTS returns FALSE.

ListExpression is any expression that returns a list. All the following expressions, for example, return a list:
- A LIST constructor
- A field reference with the [] array indicator
- Some SELECT expressions (not all return a list)

If you want to know only whether a list contains at least one elements or none, EXISTS executes more quickly than an expression involving the CARDINALITY function (for example, `CARDINALITY(ListExpression ) <> 0`).

A typical use of this function is to determine whether a field exists.

Examples

```
-- Determine whether the F1 array exists in the message. Note that the [ ]
-- are required
DECLARE Field1Exists BOOLEAN EXISTS(OutputRoot.XMLNS.Data.Source.F1[]);
-- Determine whether the F1 array contains an element with the value 'F12'.
-- Again note that the [ ] are required
DECLARE Field1F12Exists BOOLEAN
  EXISTS(SELECT F.* FROM OutputRoot.XMLNS.Data.Source.F1[] AS F where F = 'F12');
```

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"CARDINALITY function" on page 5238

"SINGULAR function" on page 5241
The SINGULAR function returns a Boolean value that indicates whether a list contains exactly one element.

"THE function" on page 5242
The THE function returns the first element of a list.

"Syntax diagrams" on page 3677

"ESQL list functions" on page 5238
ESQL provides several functions to work with lists.

SINGULAR function:

The SINGULAR function returns a Boolean value that indicates whether a list contains exactly one element.

Syntax

If the list specified by *ListExpression* contains exactly one element, SINGULAR returns TRUE. If the list contains more or fewer elements, SINGULAR returns FALSE.

ListExpression is an expression that returns a list. All the following expressions, for example, return a list:
- A LIST constructor
- A field reference with the [] array indicator
- Some SELECT expressions (not all return a list)

If you want to know only whether a list contains just one element or some other number, SINGULAR executes more quickly than an expression involving the CARDINALITY function (for example, `CARDINALITY(ListExpression ) = 1`).

A typical use of this function is to determine whether a field is unique.

Examples

```
-- Determine whether there is just one F1 field in the message.
-- Note that the [ ] are required
DECLARE Field1Unique BOOLEAN SINGULAR(OutputRoot.XMLNS.Data.Source.F1[]);

-- Determine whether there is just one field called F1 with the value 'F12'
-- in the message. Again note that the [ ] are required
DECLARE Field1F12Unique BOOLEAN
  SINGULAR(SELECT F.* FROM OutputRoot.XMLNS.Data.Source.F1[] AS F where F = 'F12');
```

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"CARDINALITY function" on page 5238

"EXISTS function" on page 5239
The EXISTS function returns a Boolean value to indicate whether a list contains at least one element (that is, whether the list exists).

"THE function" on page 5242
The THE function returns the first element of a list.

"Syntax diagrams" on page 3677

"ESQL list functions" on page 5238
ESQL provides several functions to work with lists.

THE function:

The THE function returns the first element of a list.

Syntax

If *ListExpression* contains one or more elements; THE returns the first element of the list. In all other cases, it returns an empty list.

Restrictions

ListExpression must be a SELECT expression.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"CARDINALITY function" on page 5238

"EXISTS function" on page 5239
The EXISTS function returns a Boolean value to indicate whether a list contains at least one element (that is, whether the list exists).

"SINGULAR function" on page 5241
The SINGULAR function returns a Boolean value that indicates whether a list contains exactly one element.

"Syntax diagrams" on page 3677

"ESQL list functions" on page 5238
ESQL provides several functions to work with lists.

Complex ESQL functions:
This topic lists the complex ESQL functions and covers the following:

"CASE function" on page 5243

"CAST function" on page 5245

"SELECT function" on page 5260

"ROW constructor function" on page 5267

"LIST constructor function" on page 5269

"ROW and LIST combined" on page 5270

"ROW and LIST comparisons" on page 5271

"Supported casts" on page 5273

"Implicit casts" on page 5282

"Implicit CASTs for comparisons" on page 5282

"Implicit CASTs for arithmetic operations" on page 5285

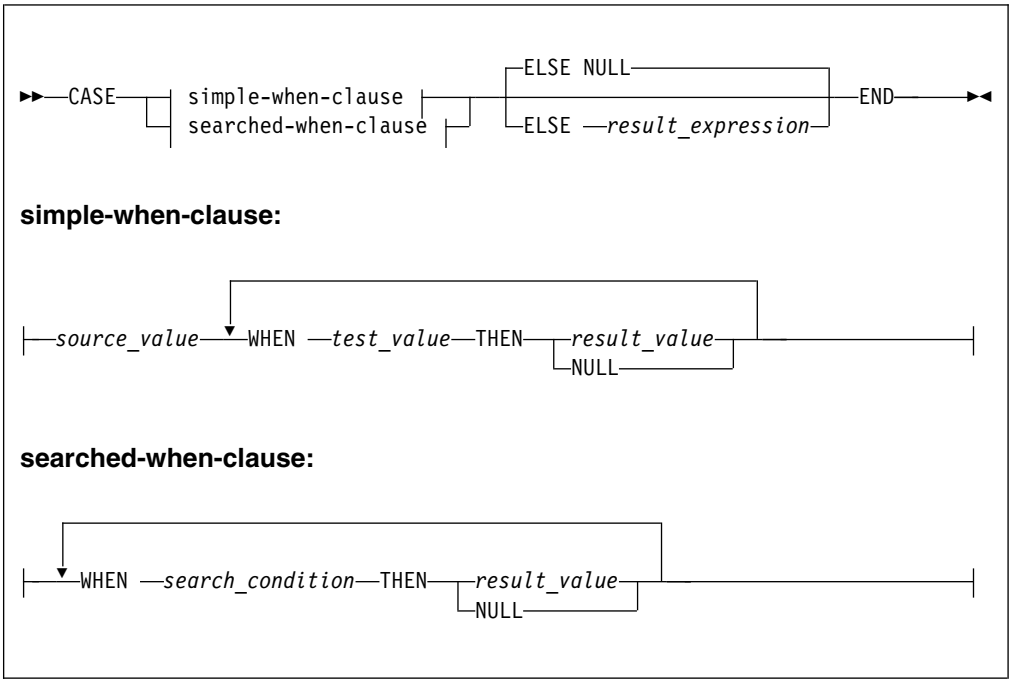
"Implicit CASTs for assignment" on page 5287

"Data types of values from external databases" on page 5288

CASE function:

CASE is a complex function that has two forms; the simple-when form and the searched-when form. In either form CASE returns a result, the value of which controls the path of subsequent processing.

Syntax



In the simple-when form, *source_value* is compared with each *test_value* until a match is found. The result of the CASE function is the value of the corresponding *result_value*. The data type of *source_value* must therefore be comparable to the data type of each *test_value*.

The CASE function must have at least one WHEN clause. The ELSE expression is optional. The default ELSE expression is NULL. A CASE expression is delimited by END. The test values do not have to be literal values.

The searched-when form is similar, but has the additional flexibility of allowing a number of different values to be tested.

The following example shows a CASE function with a simple WHEN clause. In this example, the CASE can be determined only by one variable that is specified next to the CASE keyword.

```
DECLARE CurrentMonth CHAR;
 DECLARE MonthText CHAR;
 SET CurrentMonth = SUBSTRING(InputBody.Invoice.InvoiceDate FROM 6 FOR 2);

 SET MonthText =
  CASE CurrentMonth
     WHEN '01' THEN 'January'
     WHEN '02' THEN 'February'
     WHEN '03' THEN 'March'
     WHEN '04' THEN 'April'
     WHEN '05' THEN 'May'
     WHEN '06' THEN 'June'
     ELSE 'Second half of year'
  END;
```

The following example shows a CASE function with a searched-when-clause. This example is still determined by one variable CurrentMonth:

```
DECLARE CurrentMonth CHAR;
 DECLARE MonthText CHAR;
 SET CurrentMonth = SUBSTRING(InputBody.Invoice.InvoiceDate FROM 6 FOR 2);

 SET MonthText =
  CASE
     WHEN Month = '01' THEN 'January'
     WHEN Month = '02' THEN 'February'
     WHEN Month = '03' THEN 'March'
     WHEN Month = '04' THEN 'April'
     WHEN Month = '05' THEN 'May'
     WHEN Month = '06' THEN 'June'
     ELSE 'Second half of year'
  END;
```

In a searched-when-clause, different variables can be used in the WHEN clauses to determine the result. This is demonstrated in the following example of the searched-when-clause:

```
DECLARE CurrentMonth CHAR;
 DECLARE CurrentYear CHAR;
 DECLARE MonthText CHAR;
 SET CurrentMonth = SUBSTRING(InputBody.Invoice.InvoiceDate FROM 6 FOR 2);
 SET CurrentYear = SUBSTRING(InputBody.Invoice.InvoiceDate FROM 1 FOR 4);

 SET MonthText =
   CASE
     WHEN CurrentMonth = '01' THEN 'January'
     WHEN CurrentMonth = '02' THEN 'February'
     WHEN CurrentMonth = '03' THEN 'March'
     WHEN CurrentYear = '2000' THEN 'A month in the Year 2000'
     WHEN CurrentYear = '2001' THEN 'A month in the Year 2001'
     ELSE 'Not first three months of any year or a month in the Year 2000 or 2001'
   END;
```

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and

Filter nodes in your message flows by coding ESQL.

Related reference:

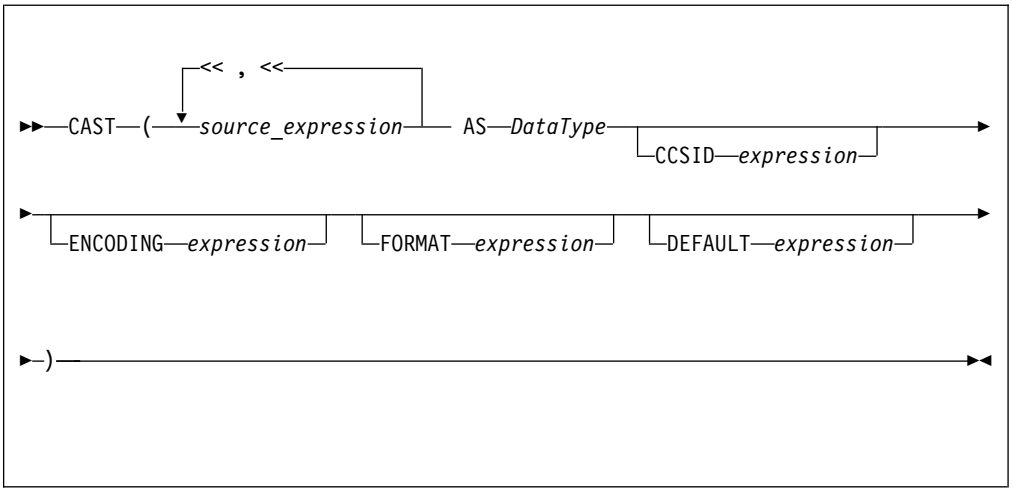
"Syntax diagrams" on page 3677

"Complex ESQL functions" on page 5242

"MOVE statement" on page 5145
The MOVE statement changes the field to which the *Target* reference variable points.

CAST function:
CAST is a complex function that transforms one or more values from one data type into another.

Syntax



In practice, you cannot specify all of the above parameters at the same time. For example, **CCSID** and **ENCODING** parameters take effect only on string-to-string conversions, while **FORMAT** applies only to string-numeric and string-datetime conversions (in either direction).

Not all conversions are supported; see "Supported casts" on page 5273 for a list of supported conversions.

Parameters:
Source expression

CAST returns its first parameter (**source_expression**), which can contain more than one value, as the data type that is specified by its second parameter (**DataType**). In all cases, if the source expression is NULL, the result is NULL. If the evaluated source expression is not compatible with the target data type, or if the source expression is of the wrong format, a runtime error is generated.

CCSID

The **CCSID** parameter is used only for conversions to or from one of the string data types. Use the **CCSID** parameter to specify the code page of the source or target string.

The **CCSID** parameter can be any expression that evaluates to a value of type INT. The expression is interpreted according to normal WebSphere Message Broker rules for CCSIDs. See "Supported code pages" on page 4176 for a list of valid values.

DataType

The **DataType** parameter is the data type into which the source value is transformed. The possible values are:

- String types:
  - BIT
  - BLOB
  - CHARACTER
- Numeric types:
  - DECIMAL
  - FLOAT
  - INTEGER
- Date/Time types:
  - DATE
  - GMTTIME
  - GMTTIMESTAMP
  - INTERVAL
  - TIME
  - TIMESTAMP
- Boolean:
  - BOOLEAN

Ensure that you specify a valid ESQL interval subtype after a Date/Time type of INTERVAL. For valid ESQL interval subtypes, see "ESQL INTERVAL data type" on page 5027. For example commands that show how to specify a valid ESQL interval subtype, see examples 12, 13, and 14 later in this section.

DEFAULT

The **DEFAULT** parameter provides a method of avoiding exceptions being thrown from CAST statements by providing a last-resort value to return.

The **DEFAULT** parameter must be a valid ESQL expression that returns the same data type as that specified on the **DataType** parameter, otherwise an exception is thrown.

The **CCSID**, **ENCODING**, and **FORMAT** parameters are not applied to the result of the **DEFAULT** parameter; the expression must, therefore, be of the correct **CCSID**, **ENCODING**, and **FORMAT**.

ENCODING

Use the **ENCODING** parameter to specify the encoding for certain conversions. The **ENCODING** value can be any expression that evaluates to a value of type INT, and is interpreted according to normal WebSphere Message Broker rules for encoding. Valid values are:

- MQENC_NATIVE (0x00000222L)
- MQENC_INTEGER_NORMAL (0x00000001L)
- MQENC_INTEGER_REVERSED (0x00000002L)
- MQENC_DECIMAL_NORMAL (0x00000010L)

- MQENC_DECIMAL_REVERSED (0x00000020L)
- MQENC_FLOAT_IEEE_NORMAL (0x00000100L)
- MQENC_FLOAT_IEEE_REVERSED (0x00000200L)
- MQENC_FLOAT_S390 (0x00000300L)

FORMAT

Use the **FORMAT** parameter for conversions between string data types and numeric or date/time data types. For conversions *from* string types, **FORMAT** defines how the source string should be parsed to fill the target data type. For conversions *to* string types, it defines how the data in the source expression is formatted in the target string.

FORMAT takes different types of expression for date/time and numeric conversions. However, the same **FORMAT** expression can be used irrespective of whether the conversion is to a string or from a string.

You can specify a **FORMAT** parameter when casting:

- From any of the string data types (BIT, BLOB, or CHARACTER) to:
  - DECIMAL
  - FLOAT
  - INTEGER
  - DATE
  - GMTTIMESTAMP
  - TIMESTAMP
  - GMTTIME
  - TIME
- To any of the string data types (BIT, BLOB, or CHARACTER) from any of the numeric and date/time data types in the previous list.

Specifying **FORMAT** for an unsupported combination of source and target data types causes error message BIP3205 to be issued.

For more information about conversion to and from numeric data types, see "Formatting and parsing numbers as strings" on page 5250. For more information about conversion to and from date/time data types, see "Formatting and parsing dateTimes as strings" on page 5253.

The **FORMAT** parameter is equivalent to those used in many other products, such as ICU and Microsoft Excel.

Examples:

Example 1. Formatted CAST from DECIMAL to CHARACTER

```
DECLARE source DECIMAL 31415.92653589;
DECLARE target CHARACTER;
DECLARE pattern CHARACTER '#,##0.00';
SET target = CAST(source AS CHARACTER FORMAT pattern);
-- target is now '31,415.93'
```

Example 2. Formatted CAST from DATE to CHARACTER

```
DECLARE now CHARACTER;
SET now = CAST(CURRENT_TIMESTAMP AS CHARACTER FORMAT 'yyyyMMdd-HHmmss');
-- target is now '20041007-111656' (in this instance at least)
```

Example 3. Formatted CAST from CHARACTER to DATE

```
DECLARE source CHARACTER '01-02-03';
DECLARE target DATE;
DECLARE pattern CHARACTER 'dd-MM-yy';
SET target = CAST(source AS DATE FORMAT pattern);
-- target now contains Year=2003, Month=02, Day=01
```

Example 4. Formatted CAST from CHARACTER to TIMESTAMP

```
DECLARE source CHARACTER '12 Jan 03, 3:45pm';
DECLARE target TIMESTAMP;
DECLARE pattern CHARACTER 'dd MMM yy, h:mma';
SET target = CAST(source AS TIMESTAMP FORMAT pattern);
-- target now contains Year=2003, Month=01, Day=03, Hour=15, Minute=45,
                       Seconds=58
-- (seconds taken from CURRENT_TIME since not present in input)
```

Example 5. Formatted CAST from DECIMAL to CHARACTER, with negative pattern

```
DECLARE source DECIMAL -54231.122;
DECLARE target CHARACTER;
DECLARE pattern CHARACTER '#,##0.00;(#,##0.00)';
SET target = CAST(source AS CHARACTER FORMAT pattern);
-- target is now '£(54,231.12)'
```

Example 6. Formatted CAST from CHARACTER to TIME

```
DECLARE source CHARACTER '16:18:30';
DECLARE target TIME;
DECLARE pattern CHARACTER 'hh:mm:ss';
SET target = CAST(source AS TIME FORMAT pattern);
-- target now contains  Hour=16, Minute=18, Seconds=30
```

Example 7. CASTs from the numeric types to DATE

```
CAST(7, 6, 5 AS DATE);
CAST(7.4e0, 6.5e0, 5.6e0 AS DATE);
CAST(7.6, 6.51, 5.4 AS DATE);
```

Example 8. CASTs from the numeric types to TIME

```
CAST(9, 8, 7 AS TIME);
CAST(9.4e0, 8.6e0, 7.1234567e0 AS TIME);
CAST(9.6, 8.4, 7.7654321 AS TIME);
```

Example 9. CASTs to TIMESTAMP

```
CAST(DATE '0001-02-03', TIME '04:05:06' AS TIMESTAMP);
CAST(2, 3, 4, 5, 6, 7.8 AS TIMESTAMP);
```

Example 10. CASTs to GMTTIMESTAMP

```
CAST(DATE '0002-03-04', GMTTIME '05:06:07' AS GMTTIMESTAMP);
CAST(3, 4, 5, 6, 7, 8 AS GMTTIMESTAMP);
CAST(3.1e0, 4.2e0, 5.3e0, 6.4e0, 7.5e0, 8.6789012e0 AS GMTTIMESTAMP);
CAST(3.2, 4.3, 5.4, 6.5, 7.6, 8.7890135 AS GMTTIMESTAMP);
```

Example 11. CASTs to INTERVAL from INTEGER

```
CAST(1234 AS INTERVAL YEAR);
CAST(32, 10 AS INTERVAL YEAR   TO MONTH );
CAST(33, 11 AS INTERVAL DAY    TO HOUR  );
CAST(34, 12 AS INTERVAL HOUR   TO MINUTE);
CAST(35, 13 AS INTERVAL MINUTE TO SECOND);
CAST(36, 14, 10  AS INTERVAL DAY  TO MINUTE);
CAST(37, 15, 11  AS INTERVAL HOUR TO SECOND);
CAST(38, 16, 12, 10 AS INTERVAL DAY TO SECOND);
```

Example 12. CASTs to INTERVAL from FLOAT

```
CAST(2345.67e0   AS INTERVAL YEAR  );
CAST(3456.78e1   AS INTERVAL MONTH );
CAST(4567.89e2   AS INTERVAL DAY   );
CAST(5678.90e3   AS INTERVAL HOUR  );
CAST(6789.01e4   AS INTERVAL MINUTE);
CAST(7890.12e5   AS INTERVAL SECOND);
CAST(7890.1234e0 AS INTERVAL SECOND);
```

Example 13. CASTs to INTERVAL from DECIMAL

```
CAST(2345.67   AS INTERVAL YEAR  );
CAST(34567.8   AS INTERVAL MONTH );
CAST(456789    AS INTERVAL DAY   );
CAST(5678900   AS INTERVAL HOUR  );
CAST(67890100  AS INTERVAL MINUTE);
CAST(789012000 AS INTERVAL SECOND);
CAST(7890.1234 AS INTERVAL SECOND);
```

Example 14. CASTs to FLOAT from INTERVAL

```
CAST(INTERVAL '1234' YEAR   AS FLOAT);
CAST(INTERVAL '2345' MONTH  AS FLOAT);
CAST(INTERVAL '3456' DAY    AS FLOAT);
CAST(INTERVAL '4567' HOUR   AS FLOAT);
CAST(INTERVAL '5678' MINUTE AS FLOAT);
CAST(INTERVAL '6789.01' SECOND AS FLOAT);
```

Example 15. CASTs DECIMAL from INTERVAL

```
CAST(INTERVAL '1234' YEAR   AS DECIMAL);
CAST(INTERVAL '2345' MONTH  AS DECIMAL);
CAST(INTERVAL '3456' DAY    AS DECIMAL);
CAST(INTERVAL '4567' HOUR   AS DECIMAL);
CAST(INTERVAL '5678' MINUTE AS DECIMAL);
CAST(INTERVAL '6789.01' SECOND AS DECIMAL);
```

Example 16. A ternary cast that fails and results in the substitution of a default value

```
CAST(7, 6, 32 AS DATE DEFAULT DATE '1947-10-24');
```

Example 17. A sexternary cast that fails and results in the substitution of a default value

```
CAST(2, 3, 4, 24, 6, 7.8 AS TIMESTAMP DEFAULT TIMESTAMP '1947-10-24 07:08:09');
```

Example 18. A ternary cast that fails and throws an exception

```
BEGIN
  DECLARE EXIT HANDLER FOR SQLSTATE LIKE '%' BEGIN
    SET OutputRoot.XMLNS.Data.Date.FromIntegersInvalidCast = 'Exception thrown';
    END;

  DECLARE Dummy CHARACTER CAST(7, 6, 32 AS DATE);
  END;
```

Example 19. A sexternary cast that fails and throws an exception

```
BEGIN
  DECLARE EXIT HANDLER FOR SQLSTATE LIKE '%' BEGIN
    SET OutputRoot.XMLNS.Data.Timestamp.FromIntegersInvalidCast = 'Exception thrown';
    END;

  DECLARE Dummy CHARACTER CAST(2, 3, 4, 24, 6, 7.8 AS TIMESTAMP);
  END;
```

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Casting data from message fields" on page 2474
You can use the CAST function to transform the data type of one value to match the data type of the other. For example, you can use the CAST function when you process generic XML messages. All fields in an XML message have character values, so if you want to perform arithmetic calculations or datetime comparisons, for example, you must convert the string value of the field into a value of the appropriate type using CAST.

Related reference:

"Syntax diagrams" on page 3677

"Complex ESQL functions" on page 5242

"Supported casts" on page 5273

"Implicit casts" on page 5282

"Formatting and parsing numbers as strings"
For conversions between string data types and numeric data types, you can supply, on the FORMAT parameter of the CAST function, an optional formatting expression.

"Formatting and parsing dateTimes as strings" on page 5253
This section gives information on how you can specify the dateTime format using a string of pattern letters.

"Data types of values from external databases" on page 5288
How database data types are implicitly cast to ESQL data types.

"Supported code pages" on page 4176
Application messages must conform to supported code pages.

Formatting and parsing numbers as strings:

For conversions between string data types and numeric data types, you can supply, on the FORMAT parameter of the CAST function, an optional formatting expression.

For conversions *from* string types, the formatting expression defines how the source string should be parsed to fill the target data type.

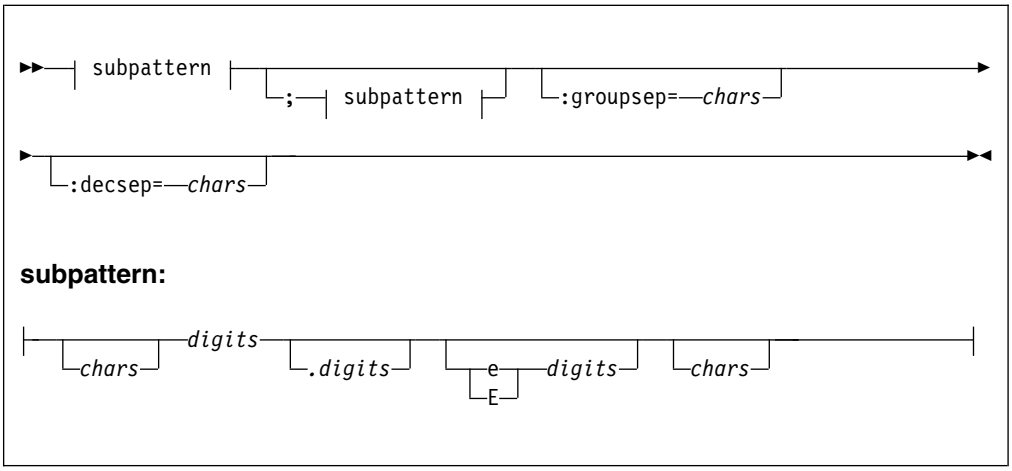
For conversions *to* string types, the formatting expression defines how the data in the source expression is to be formatted in the target string.

You can specify a FORMAT expression for the following numeric conversions (Specifying a FORMAT expression for date/time conversions is described in "Formatting and parsing dateTimes as strings" on page 5253).

- From any of the string data types (BIT, BLOB, or CHARACTER) to:
  - DECIMAL
  - FLOAT
  - INTEGER
- To any of the string data types (BIT, BLOB, or CHARACTER) from any of the numeric data types that are in the previous list.

The formatting expression consists of three parts:
1. A `subpattern` that defines positive numbers.
2. An optional `subpattern` that defines negative numbers. (If only one `subpattern` is defined, negative numbers use the positive pattern, prefixed with a minus sign.)
3. The optional parameters `groupsep` and `decsep`.

Syntax



Parameters:
chars

A sequence of zero or more characters. All characters can be used, *except* the special characters that are listed under "subpattern" on page 5252.

decsep

One or more characters to be used as the separator between the whole and decimal parts of a number (the decimal separator). The default decimal separator is a period (.).

digits

A sequence of one or more of the numeric tokens (0 # - + , .) that are listed under "subpattern" on page 5252.

groupsep

One or more characters to be used as the separator between clusters of integers, to make large numbers more readable (the grouping separator). The default grouping separator is nothing (that is, there is no grouping of digits or separation of groups).

Grouping is commonly done in thousands, but it can be redefined by either the pattern or the locale. There are two grouping sizes:

The primary grouping size
Used for the least significant integer digits.

The secondary grouping size
Used for all other integer digits.

In most cases, the primary and secondary grouping sizes are the same, but they can be different. For example, if the pattern used is `#,##,##0`, the primary grouping size is 3 and the secondary is 2. The number 123456789 would become the string “12,34,56,789”.

If multiple grouping separators are used (as in the previous example), the rightmost separator defines the primary size, and the penultimate rightmost separator defines the secondary size.

subpattern

The subpattern consists of:
1. An optional prefix (*chars*)
2. A mandatory pattern representing a whole number
3. An optional pattern representing decimal places
4. An optional pattern representing an exponent (the power by which the preceding number is raised)
5. An optional suffix (*chars*)

Parts 2, 3, and 4 of the subpattern are defined by the tokens in the following table.

Token	Represents
0	Any digit, including a leading zero.
#	Any digit, excluding a leading zero. (See the explanation of the difference between 0 and # that follows this table.)
.	Decimal separator.
+	Prefix of positive numbers.
-	Prefix of negative numbers.
,	Grouping separator.
E/e	Separates the number from the exponent.
;	Subpattern boundary.
'	Quotation mark, applied to special characters. If a quotation mark is needed in output, it must be doubled (').
*	Padding specifier. The character following the asterisk is used to pad the number to fit the length of the format pattern.

The # and 0 characters are used for digit substitution, the difference between them being that a # character is removed if there is no number to replace it with. For example, 10 formatted by the pattern `#,##0.00` gives “10.00”, but formatted by `0,000.00` gives “0,010.00”.

To specify padding characters, use an asterisk. When an asterisk is placed in either of the two *chars* regions (the prefix and suffix), the character immediately following it is used to pad the output. Padding can be specified only once. For example, a pattern of `*x#,###,##0.00` applied to 1234 gives “xxx1,234.00”. But applied to 1234567, it gives “1,234,567.00”.

Examples of formatting patterns:
The following table shows formatting patterns and the strings that are generated from sample numeric input.

Pattern	Input number	Output string
`+###,##0.00;-###,###,##0.00:groupsep='':decsep=,`	123456789.123	“+123'456'789,12”
`##0.00`	1000000	“1000000.00”
`##0.00`	3.14159265	“3.14”

Related reference:

“CAST function” on page 5245

“Formatting and parsing dateTimes as strings”
This section gives information on how you can specify the dateTime format using a string of pattern letters.

“Supported casts” on page 5273

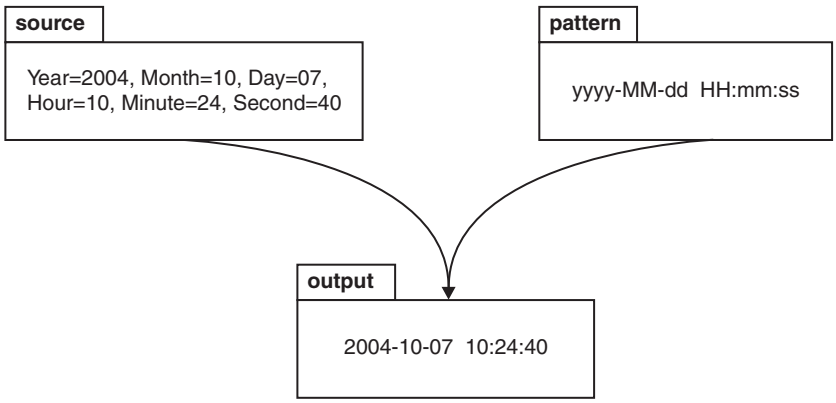
“Implicit casts” on page 5282

Formatting and parsing dateTimes as strings:

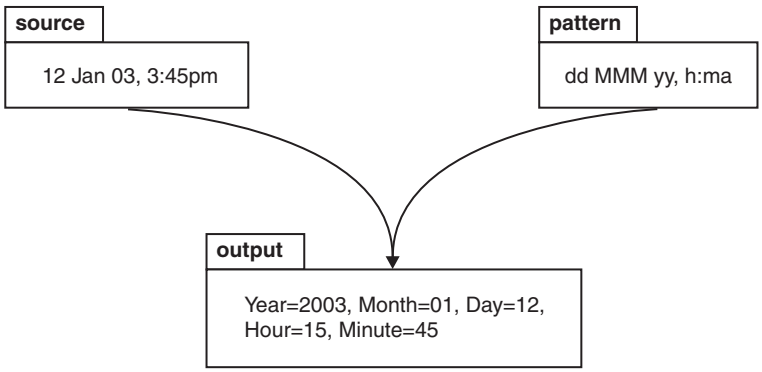
This section gives information on how you can specify the dateTime format using a string of pattern letters.

When you convert a date or time into a string, a format pattern must be applied that directs the conversion. Apply the format pattern to convert a date or time into a string, or to parse a string into a date or time.

During the conversion (for example, of a dateTime into a string), a pattern or a set of tokens is replaced with the equivalent source. The following diagram shows how a pattern is used to format a dateTime source to produce a character string output.



When a string is parsed (for example, when converting the string to a dateTime), the pattern or set of tokens is used to determine which part of the target dateTime is represented by which part of the string. The following diagram shows how this is done.



Syntax

The expression pattern is defined by:

Where:

symbol
is a character in the set **adDeEFGhHIkKmMsSTUwWyYzZ**.

string is a sequence of characters enclosed in single quotation marks. If a single quotation mark is required within the string, use two single quotation marks ('').

Characters for formatting a dateTime as a string

The following table lists the characters that you can use in a pattern for formatting or parsing strings in relation to a dateTime. The table is followed by some notes that explain more about some of the examples in the table.

Symbol	Meaning	Presentation	Examples
a	am or pm marker	Text	Input am, AM, pm, PM. Output AM or PM
d	day in month (1-31)	Number	1, 20
dd	day in month (01-31)	Number	01, 31
D	day in year (1-366)	Number	3, 80, 100
DD	day in year (01-366)	Number	03, 80, 366
DDD	day in year (001-366)	Number	003
e	day in week (1-7)[1]	Number	2
EEE	day in week[1]	Text	Tue
EEEE	day in week[1]	Text	Tuesday

Symbol	Meaning	Presentation	Examples
F	day of week in month (1-5)[2]	Number	2
G	Era	Text	BC or AD
h	hour in am or pm (1-12)	Number	6
hh	hour in am or pm (01-12)	Number	06
H	hour of day in 24 hour form (0-23)[3]	Number	7
HH	hour of day in 24 hour form (00-23)[3]	Number	07
I	ISO8601 Date/Time (up to yyyy-MM-dd'T'HH:mm:ss.SSSZZZ)[4]	Text	2006-10-07T12:06:56.568+01:00
IU	ISO8601 Date/Time (similar to I, but ZZZ with output "Z" if the time zone is +00:00)[4]	Text	2006-10-07T12:06:56.568+01:00, 2003-12 -15T15:42:12.000Z
k	hour of day in 24 hour form (1-24)[3]	Number	8
kk	hour of day in 24 hour form (01-24)[3]	Number	08
K	hour in am or pm (0-11)	Number	9
KK	hour in am or pm (00-11)	Number	09
m	minute	Number	4
mm	minute	Number	04
M	numeric month	Number	5, 12
MM	numeric month	Number	05, 12
MMM	named month	Text	Jan, Feb
MMMM	named month	Text	January, February
s	seconds[10]	Number	5
ss	seconds[10]	Number	05
S	decisecond[5]	Number	7
SS	centisecond[5]	Number	70
SSS	millisecond[5]	Number	700
SSSS	0.0001 second[5]	Number	7000
SSSSS	0.00001 second[5]	Number	70000
SSSSSS	0.000001 second[5]	Number	700000
T	ISO8601 Time (up to HH:mm:ss.SSSZZZ)[4]	Text	12:06:56.568+01:00
TU	ISO8601 Time (similar to T, but a time zone of +00:00 is replaced with 'Z')[4]	Text	12:06:56.568+01:00, 15:42:12.000Z
w	week in year[6]	Number	7, 53
ww	week in year[6]	Number	07, 53
W	week in month[7]	Number	2
yy	year[8]	Number	06

Symbol	Meaning	Presentation	Examples
yyyy	year[8]	Number	2006
YY	year: use with week in year only[6]	Number	06
YYYY	year: use with week in year only[6]	Number	2006
zzz	time zone (abbreviated name)[9]	Text	EST
zzzz	time zone (full name)	Text	Eastern Standard Time
Z	time zone (+/-n)	Text	+3
ZZ	time zone (+/-nn)	Text	+03
ZZZ	time zone (+/-nn:nn)	Text	+03:00
ZZZU	time zone (as ZZZ, "+00:00" is replaced by "Z")	Text	+03:00, Z
ZZZZ	time zone (GMT+/-nn:nn)	Text	GMT+03:00
ZZZZZ	time zone (as ZZZ, but no colon) (+/-nnnn)	Text	+0300
'	escape for text		'User text'
''	(two single quotation marks) single quotation mark within escaped text		'o''clock'

The presentation of the dateTime object depends on what symbols you specify.

- **Text**. If you specify four or more of the symbols, the full form is presented. If you specify less than four symbols, the short or abbreviated form, if it exists, is presented. For example, `EEEE` produces `Monday`, `EEE` produces `Mon`.
- **Number**. The number of characters for a numeric dateTime component must be within the bounds of the corresponding formatting symbols. Repeat the symbol to specify the minimum number of digits that are required. The maximum number of digits allowed is the upper bound for a particular symbol. For example, `day in month` has an upper bound of 31; therefore, a format string of `d` allows the values 2 or 21 to be parsed but disallows the values 32 and 210. On output, numbers are padded with zeros to the specified length. A year is a special case; see note 8 in the following list. Fractional seconds are also special case; see note 5 in the following list.
- Any characters in the pattern that are not in the ranges of ['a'..'z'] and ['A'..'Z'] are treated as quoted text. For example, characters like colon (:), comma (,), period (.), the number sign (hash or pound, #), the at sign (@), and space appear in the resulting time text even if they are not enclosed within single quotes.
- You can create formatting strings that produce unpredictable results; therefore, you must use these symbols with care. For example, if you specify `dMyyyy`, you cannot distinguish between day, month, and year. `dMyyyy` tells the broker that a minimum of one character represents the day, a minimum of one character represents the month, and four characters represent the year. Therefore, `3111999` can be interpreted as either 3/11/1999 or 31/1/1999.

Notes: The following notes apply to the table above.

1. You can specify the following values in the `day in week` field:
   - 1 - Sunday

- 2 - Monday
- 3 - Tuesday
- 4 - Wednesday
- 5 - Thursday
- 6 - Friday
- 7 - Saturday

2. 12th July 2006 is the second Wednesday in July and can be expressed as `2006 July Wednesday 2` using the format string `yyyy MMMM EEEE F`. Note that this format does not represent the Wednesday in week 2 of July 2006, which is 5th July 2006; the format string for this is `yyyy MMMM EEEE W`.
3. 24-hour fields might result in an ambiguous time, if specified with a conflicting am/pm field.
4. See "ISO8601, I and T DateTime tokens" on page 5258.
5. Fractional seconds are represented by uppercase S. The length must implicitly match the number of format symbols on input. The format string `ss SSS` or `ss.SSS`, for example, represents seconds and milliseconds. However, the format string `ss.sss` represents a repeated field (of seconds); the value after the period (.) is taken as a seconds field, not as fractional seconds. The output is truncated to the specified length.
6. In ESQL, the first day of the year is assumed to be in the first week; therefore, January 1 is always in week 1. As a result, dates that are specified relative to one year might be in a different year. For example, "Monday week 1 2005" parsed using `"EEEE' week 'w' 'YYYY"` gives a date of 2004-12-27, because the Monday of the first week in 2005 is a date in 2004.

   If you use the `y` symbol, the adjustment is not done and unpredictable results might occur for dates around the end of the year. For example, if the string "2005 01 Monday" is formatted:

   - Monday of week 1 in 2005 using format string `"YYYY ww EEEE"` is correctly interpreted as 27th December 2004
   - Monday of week 1 in 2005 using format string `"yyyy ww EEEE"` is incorrectly interpreted as 27th December 2005
7. The first and last week in a month might include days from neighboring months. For example, Monday 31st July 2006 can be expressed as *Monday in week one of August 2006*, which is `2006 08 1 Monday` using format string `yyyy MM W EEEE`.
8. Year is handled as a special case.
   - On output, if the count of `y` is 2, the year is truncated to 2 digits. For example, if `yyyy` produces `1997`, `yy` produces `97`.
   - On input, for 2 digit years the `century window` is fixed to 53. For example, an input date of 52 results in a year value of 2052, whereas an input date of 53 gives an output year of 1953, and 97 gives 1997.
9. Using the `zzz` option can have ambiguous results. For example, `BST` can be interpreted as `Bangladesh Standard Time` or `British Summer Time`. For compatibility reasons, WebSphere Message Broker uses the former interpretation.

   To avoid these problems, use the `zzzz` option with a well-defined name; for example, Europe/London, Asia/Dhaka, or America/Los_Angeles.
10. Seconds `s` & `ss`, must be in the range 0-59. If you need to construct a TIMESTAMP representing the time during a leap second, where the value being created or cast uses the value 60 for seconds, you must handle this case

within your ESQL code. The CURRENT_ datetime functions (for example, CURRENT_TIME) within the product never produce a time where the seconds value falls outside of the 0-59 range.

ISO8601, I and T DateTime tokens

If your dateTime values comply with the ISO8601:2000 'Representation of dates and times' standard, consider using the formatting symbols I and T, which match the following subset of the ISO8601 standard.

- The restricted profile as proposed by the W3C at http://www.w3.org/TR/NOTE-datetime
- Truncated representations of calendar dates, as specified in section 5.2.1.3 of ISO8601:2000
  - Basic format (subsections c, e, and f)
  - Extended format (subsections a, b, and d)

Use the formatting symbols I and T only on their own:

- The I formatting symbol matches any dateTime string that conforms to the supported subset.
- The T formatting symbol matches any dateTime string that conforms to the supported subset that consists of a time portion only.

The following table shows how the output form relates to the logical data type.

Logical model data type	ESQL data type	Output form
xsd:dateTime	TIMESTAMP or GMTTIMESTAMP	yyyy-MM-dd'T'HH:mm:ss.SSSZZZ
xsd:date	DATE	yyyy-MM-dd
xsd:gYear	INTERVAL	yyyy
xsd:gYearMonth	INTERVAL	yyyy-MM
xsd:gMonth	INTERVAL	--MM
xsd:gmonthDay	INTERVAL	--MM-dd
xsd:gDay	INTERVAL	---dd
xsd:time	TIME / GMTTIME	'T'HH:mm:ss.SSSZZZ

Note:

- On input, both I and T accept both '+00:00' and 'Z' to indicate a zero time difference from Coordinated Universal Time (UTC), but on output they always generate '+00:00'. If you want 'Z' to always be generated on output, use the IU or TU formatting symbols instead.
- ZZZ always writes '+00:00' to indicate a zero time difference from Coordinated Universal Time (UTC). If you want 'Z' to always be generated on output, use ZZZU instead.

Using the input UTC format on output

An element or attribute of logical type xsd:dateTime or xsd:time that contains a dateTime as a string can specify Coordinated Universal Time (UTC) by using either the Z symbol or time zone +00:00. On input, the MRM parser remembers the UTC format of such elements and attributes. On output, you can specify whether Z or +00:00 is displayed by using the `Default DateTime Format` property of the element or attribute. Alternatively, you can preserve the input UTC format by selecting the

message set property `Use input UTC format on output`. If this property is selected, the UTC format is preserved in the output message and overrides the format that is implied by the dateTime format property.

Understanding daylight saving time and the CAST function

When the broker is running in a time zone other than GMT, it calculates the daylight saving time (DST) offset on times that are supplied to it by the CAST function. For CAST to calculate the offset correctly, the time passed into CAST must have a time zone associated with it, as a Z parameter. If no time zone is associated with the value passed, the time is converted into GMT time; it is not treated as a local time stamp.

Also, when you use CAST to cast a string to a time value, the DST offset is calculated using the current system date. To cast a string to a time variable and calculate DST for a specific date, you must also specify the date.

For example, if timeValue='10:00:00', the following code, run on a broker that is in the Central Daylight Time zone, converts the time to GMT, because no time zone identifier is specified:

```
DECLARE castTime TIME;
SET castTime = CAST (timeValue AS TIME FORMAT timePattern)
```

The time is not converted into GMT again if the castTime variable is used in any subsequent code, for example

```
CAST(castDate, castTime AS GMTTIMESTAMP);
```

Examples

The following table shows a few examples of dateTime formats.

Format pattern	Result
"yyyy.MM.dd 'at' HH:mm:ss ZZZ"	2006.07.10 at 15:08:56 -05:00
"EEE, MMM d, "yy"	Wed, July 10, '06
"h:mm a"	8:08 PM
"hh o"clock a, ZZZZ"	09 o'clock AM, GMT+09:00
"K:mm a, ZZZ"	9:34 AM, -05:00
"yyyy.MMMMM.dd hh:mm aaa"	1996.July.10 12:08 PM

Use within an MRM domain

In MRM it is possible to define an element that has the logical type of dateTime.

When a dateTime element is parsed, a field is created in the message tree that has the ESQL datatype of CURRENT_TIME or CURRENT_TIMESTAMP. However, the CURRENT_TIME and CURRENT_TIMESTAMP data types do not have the functionality to store timezone information, and the MRM does not adjust the time according to the input timezone and the timezone of the broker.

Although the CURRENT_TIME and CURRENT_TIMESTAMP data types cannot store timezone information, the MRM stores this information as part of the

underlying field. This means that if the field is copied between message trees, the timezone information is copied with it, allowing this information to be preserved on output.

Note that the information is preserved only if the field is copied to a field of the same name.

However, if any new field is derived from the original field, the new field does not have the timezone information. This means that if such a field is cast as a character, the new field assumes the timezone of the broker, but its value is not adjusted for any difference between the input timezone and the timezone of the broker.

For example, an input dateTime element containing `2009-02-20T06:08:07-08:00` could be copied from the input message tree to the output message tree and appear in an output message in exactly the same format. However, if the element is cast as character, using format `IU`, by a broker running GMT the result would be `2009-02-20T06:08:07.000Z`.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with a message definition file" on page 2863
Create, open, and delete a message definition file.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"CAST function" on page 5245

"CURRENT_TIME function" on page 5179

"CURRENT_TIMESTAMP function" on page 5180

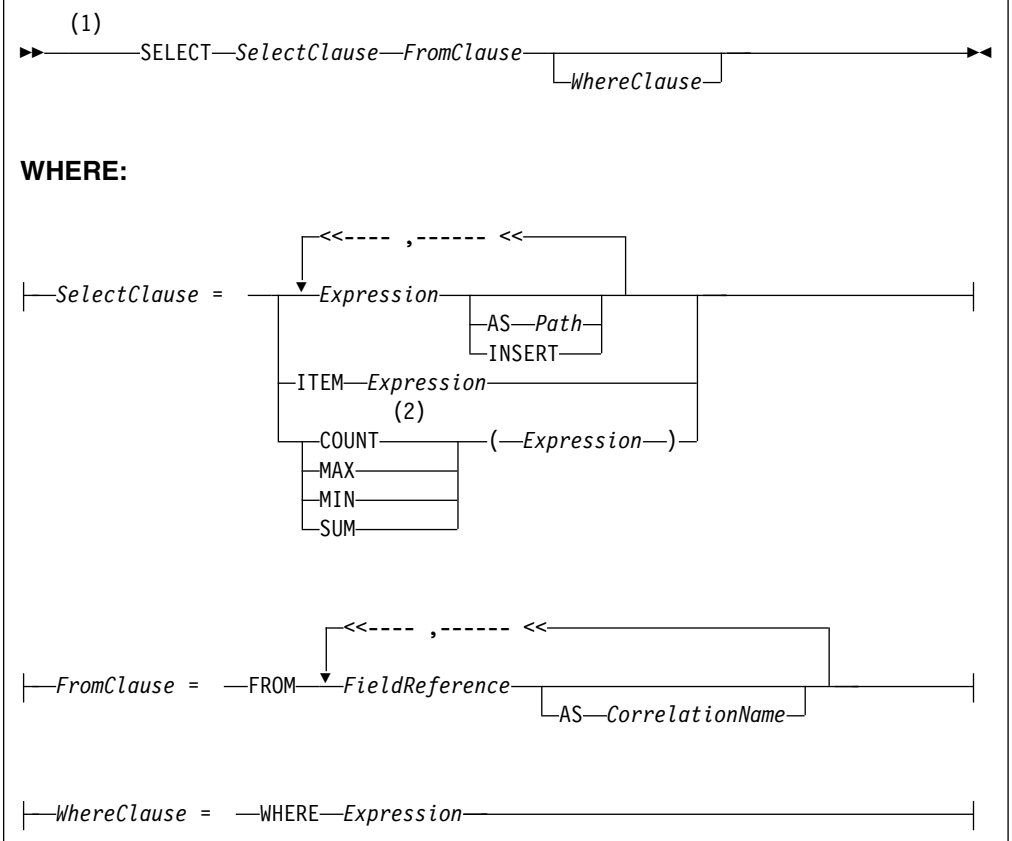
"Supported casts" on page 5273

"Implicit casts" on page 5282

SELECT function:

The SELECT function combines, filters, and transforms complex message and database data.

Syntax



Notes:

1 You no longer require the enclosing parentheses in SELECT expressions. This does not prevent you using parentheses but, if they are present, they are merely normal, expression-scoping, parentheses.

2 For the COUNT parameter only, you can specify the value of the following *Expression* as a single star (*).

Usage

The SELECT function is the usual and most efficient way of transforming messages. You can use SELECT to:

- Comprehensively reformat messages
- Access database tables
- Make an output array that is a subset of an input array
- Make an output array that contains only the values of an input array
- Count the number of entries in an array
- Select the minimum or maximum value from a number of entries in an array
- Sum the values in an array

Introduction to SELECT

The SELECT function considers a message tree (or sub-tree) to consist of a number of rows and columns, rather like a database table. A *FieldReference* in a FROM clause identifies a field in a message tree. The identified field is regarded in the following ways:
- The identified field is regarded as a row in a table.
- The field's siblings are regarded as other rows of the same table.
- The field's children are regarded as the table's columns.

Note: The *FieldReference* in a FROM clause can also be a table reference that refers directly to a real database table.

The return value of the SELECT function is typically another message tree that contains rows whose structure and content is determined by the *SelectClause*. The number of rows in the result is the sum of all the rows pointed to by all the field references and table references in the FROM clause, filtered by the WHERE clause; only those fields for which the WHERE clause evaluates to TRUE are included.

The return value of the SELECT function can also be scalar (see "ITEM selections" on page 5264).

You can specify the *SelectClause* in several ways; see:
- "Simple selections"
- "INSERT selections" on page 5264
- "ITEM selections" on page 5264
- "Column function selections" on page 5264

If you have created a message flow that contains one of the following nodes, and the ESQL that is associated with this node includes a SELECT function and a database reference, you must specify a value for the `Data source` property of the relevant node:
- Compute
- Database
- Filter

Simple selections

To understand the SELECT function in more detail, first consider the following simple case:
- The *SelectClause* consists of a number of expressions, each with an AS *Path* clause.
- The FROM clause contains a single *FieldReference* and an AS *CorrelationName* clause.

The SELECT function creates a local, reference, correlation variable, whose name is given by the AS *CorrelationName* clause, and then steps, in turn, through each row of the list of rows derived from the FROM clause. For each row:
1. The correlation variable is set to point to the current row.
2. The WHERE clause (if present) is evaluated. If it evaluates to FALSE or unknown (null), nothing is added to the result tree and processing proceeds to the next row of the input. Otherwise processing proceeds to the next step.
3. A new member is added to the result list.

4. The SELECT clause expressions are evaluated and assigned to fields named as dictated by the AS *Path* clause. These fields are child fields of the new member of the result list.

Typically, both the *SelectClause* and the WHERE clause expressions use the correlation variable to access column values (that is, fields in the input message tree) and thus to build a new message tree containing data from the input message. The correlation variable is referred to by the name specified in the AS *CorrelationName* clause or, if an AS clause is not specified, by the final name in the FROM *FieldReference* (that is, the name after the last dot).

Note that:

- Despite the analogy with a table, you are not restricted to accessing or creating messages with a flat, table-like, structure; you can access and build trees with arbitrarily deep folder structures.
- You are not restricted to a column being a single value; a column can be a repeating list value or a structure.

These concepts are best understood by reference to the examples.

If the field reference is a *TableReference*, the operation is very similar. In this case, the input is a real database table and is thus restricted to the flat structures supported by databases. The result tree is still not so restricted, however.

If the FROM clause contains more than one field reference, the rightmost reference steps through each of its rows for each row in the next-to-rightmost reference, and so on. The total number of rows in the result is thus the product of the number of rows in each table. Such selects are known as *joins* and commonly use a WHERE clause that excludes most of these rows from the result. Joins are commonly used to add database data to messages.

The AS *Path* clause is optional. If it is unspecified, the broker generates a default name according to the following rules:

1. If the *SelectClause* expression is a reference to a field or a cast of a reference to a field, the name of the field is used.
2. Otherwise the broker uses the default names Column1, Column2, and so on.

Examples

The following example performs a SELECT on the table Parts in the schema Shop in the database DSN1. Because no WHERE clause exists, all rows are selected. Because the select clause expressions (for example, P.PartNumber) contain no AS clauses, the fields in the result adopt the same names:

```
SET PartsTable.Part[] = SELECT
  P.PartNumber,
  P.Description,
  P.Price
 FROM Database.DSN1.Shop.Parts AS P;
```

If the target of the SET statement (PartsTable) is a variable of type ROW, after the statement is executed PartsTable will have, as children of its root element, a field called Part for each row in the table. Each of the Part fields will have child fields called PartNumber, Description, and Price. The child fields will have values dictated by the contents of the table. (PartsTable could also be a reference into a message tree).

The next example performs a similar SELECT. This case differs from the last in that the SELECT is performed on the message tree produced by the first example (rather than on a real database table). The result is assigned into a subfolder of OutputRoot:

```
SET OutputRoot.XMLNS.Data.TableData.Part[] = SELECT
  P.PartNumber,
  P.Description,
  P.Price
 FROM PartsTable.Part[] AS P;
```

INSERT selections

The INSERT clause is an alternative to the AS clause. It assigns the result of the *SelectClause* expression (which must be a row) to the current new row itself, rather than to a child of it. The effect of this is to merge the row result of the expression into the row being generated by the SELECT. This differs from the AS clause, in that the AS clause always generates at least one child element before adding a result, whereas INSERT generates none. INSERT is useful when inserting data from other SELECT operations, because it allows the data to be merged without extra folders.

ITEM selections

The *SelectClause* can consist of the keyword ITEM and a single expression. The effect of this is to make the results nameless. That is, the result is a list of values of the type returned by the expression, rather than a row. This option has several uses:

- In conjunction with a scalar expression and the THE function, it can be used to create a SELECT query that returns a single scalar value (for example, the price of a particular item from a table).
- In conjunction with a CASE expression and ROW constructors, it can be used to create a SELECT query that creates or handles messages in which the structure of some rows (that is, repeats in the message) is different from others. This is useful for handling messages that have a repeating structure but in which the repeats do not all have the same structure.
- In conjunction with a ROW constructor, it can be used to create a SELECT query that collapses levels of repetition in the input message.

Column function selections

The *SelectClause* can consist of one of the functions COUNT, MAX, MIN, and SUM operating on an expression. These functions are known as column functions. They return a single scalar value (not a list) giving the count, maximum, minimum, or sum of the values that *Expression* evaluated to in stepping through the rows of the FROM clause. If *Expression* evaluates to NULL for a particular row, the value is ignored, so that the function returns the count, maximum, minimum, or sum of the remaining rows.

For the COUNT function only, *Expression* can consist of a single star (*). This form counts the rows regardless of null values.

To make the result a useful reflection of the input message, *Expression* typically includes the correlation variable.

Typically, *Expression* evaluates to the same data type for each row. In these cases, the result of the MAX, MIN, and SUM functions are of the same data type as the

operands. The returned values are not required to be all of the same type however, and if they are not, the normal rules of arithmetic apply. For example, if a field in a repeated message structure contains integer values for some rows and float values for others, the sum follows the normal rules for addition. The sum is of type float because the operation is equivalent to adding a number of integer and float values.

The result of the COUNT function is always an integer.

Differences between message and database selections

FROM expressions in which a correlation variable represents a row in a message behave slightly differently from those in which the correlation variable represents a row in a real database table.

In the message case, a path involving a star (*) has the normal meaning; it ignores the field's name and finds the first field that matches the other criteria (if any).

In the database case a star (*) has, for historical reasons, the special meaning of all fields. This special meaning requires advance knowledge of the definition of the database table and is supported only when querying the default database (that is, the database pointed to by the node's `data source` attribute). For example, the following queries return column name and value pairs only when querying the default database:

```
SELECT * FROM Database.Datasource.SchemaName.Table As A
SELECT A.* FROM Database.Datasource.SchemaName.Table As A
SELECT A FROM Database.Datasource.SchemaName.Table AS A
```

Specifying the SELECT expressions

SelectClause
SelectClause expressions can use any of the broker's operators and functions in any combination. They can refer to the tables' columns, message fields, correlation names declared by containing SELECTs, and to any other declared variables or constants that are in scope.

AS *Path*
An AS *Path* expression is a relative path (that is, there is no correlation name) but is otherwise unrestricted in any way. For example, it can contain:

- Indexes (for example, A.B.C[i])
- Field-type specifiers (for example, A.B.(XML.Attribute)C)
- Multipart paths (for example, A.B.C)
- Name expressions (for example, A.B.{var})

Any expressions in these paths can also use any of the broker's operators and functions in any combination. The expressions can refer to the tables' columns, message fields, correlation names declared by containing SELECTs, and any declared variables or constants.

FROM clause
FROM clause expressions can contain multiple database references, multiple message references, or a mixture of the two. You can join tables with tables, messages with messages, or tables with messages.

FROM clause *FieldReference*s can contain expressions of any kind (for example, `Database.{DataSource}.{Schema}.Table1`).

You can calculate a field, data source, schema, or table name at run time.

WHERE clause

The WHERE clause expression can use any of the broker's operators and functions in any combination. It can refer to table columns, message fields, and any declared variables or constants.

However, be aware that the broker treats the WHERE clause expression by examining the expression and deciding whether the whole expression can be evaluated by the database. If it can, it is given to the database. In order to be evaluated by the database, it must use only those functions and operators supported by the database.

The WHERE clause can, however, refer to message fields, correlation names declared by containing SELECT functions, and to any other declared variables or constants within scope.

If the whole expression cannot be evaluated by the database, the broker looks for top-level AND operators and examines each sub-expression separately. It then attempts to give the database those sub-expressions that it can evaluate, leaving the broker to evaluate the rest. You need to be aware of this situation for two reasons:

1. Apparently trivial changes to WHERE clause expressions can have large effects on performance. You can determine how much of the expression was given to the database by examining a user trace.
2. Some databases' functions exhibit subtle differences of behavior from those of the broker.

Relation to the THE function

You can use the function THE (which returns the first element of a list) in conjunction with SELECT to produce a non-list result. This is useful, for example, when a SELECT query is required to return no more than one item. It is particularly useful in conjunction with ITEM (see "ITEM selections" on page 5264).

Differences from the SQL standard

ESQL SELECT differs from database SQL SELECT in the following ways:

- ESQL can produce tree-structured result data
- ESQL can accept arrays in SELECT clauses
- ESQL has the THE function and the ITEM and INSERT parameters
- ESQL has no SELECT ALL function in this release
- ESQL has no ORDER BY function in this release
- ESQL has no SELECT DISTINCT function in this release
- ESQL has no GROUP BY or HAVING parameters in this release
- ESQL has no AVG column function in this release

Restrictions

The following restrictions apply to the current release:

- When a SELECT command operates on more than one database table, all the tables must be in the same database instance. (That is, the *TableReference*s must not specify different data source names.)
- If the FROM clause refers to both messages and tables, the tables must precede the messages in the list.

- Using dynamic DSN, SCHEMA and TABLE names with 'SELECT *' statements is not supported. If you use a schema, table or datasource name as a variable (dynamic variables) in 'SELECT *' queries, the variables are not resolved to the correct set of schema or table names.
- The WHERE clause of a SELECT statement cannot itself contain a SELECT statement that relies on results returned from the original SELECT if either SELECT statement is from database tables.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Selecting data from database columns" on page 2491
You can configure a Compute, Filter, or Database node to select data from database columns and include it in an output message.

"Transforming a simple message" on page 2516
When you code the ESQL for a Compute node, use the SELECT function to transform simple messages.

"Transforming a complex message" on page 2520
When you code the ESQL for a Compute node, use the SELECT function for complex message transformation.

"Returning a scalar value in a message" on page 2523
Use a SELECT statement to return a scalar value by including both the **THE** and **ITEM** keywords.

"Interaction with databases using ESQL" on page 2487
Use ESQL statements and functions to read from, write to, and modify databases from your message flows.

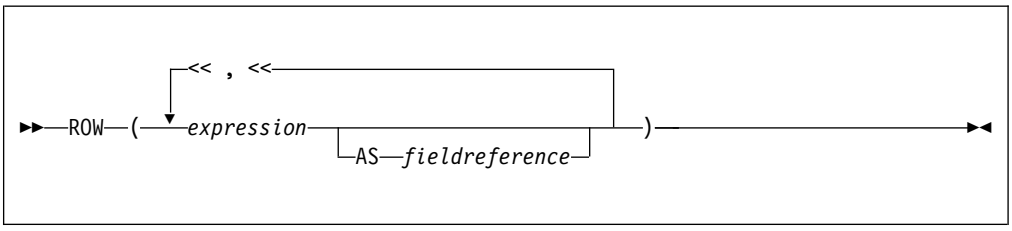
Related reference:

"Syntax diagrams" on page 3677

"Complex ESQL functions" on page 5242

ROW constructor function:
ROW constructor is a complex function used to explicitly generate rows of values that can be assigned to fields in an output message.

Syntax



A ROW consists of a sequence of named values. When assigned to a field reference it creates that sequence of named values as child fields of the referenced field. A ROW cannot be assigned to an array field reference.

Examples:

Example 1

```
SET OutputRoot.XMLNS.Data = ROW('granary' AS bread,
                        'riesling' AS wine,
                        'stilton' AS cheese);
```

produces:

```
<Data>
   <bread>granary</bread>
   <wine>riesling</wine>
   <cheese>stilton</cheese>
</Data>
```

Example 2

Given the following XML input message body:

```
<Proof>
   <beer>5</beer>
   <wine>12</wine>
   <gin>40</gin>
</Proof>
```

the following ESQL:

```
SET OutputRoot.XMLNS.Data = ROW(InputBody.Proof.beer,
                      InputBody.Proof.wine AS vin,
                        (InputBody.Proof.gin * 2) AS special);
```

produces the following result:

```
<Data>
   <beer>5</beer>
   <vin>12</vin>
   <special>80</special>
</Data>
```

Because the values in this case are derived from field references that already have names, it is not necessary to explicitly provide a name for each element of the row, but you might choose to do so.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Accessing the environment tree" on page 2469
The environment tree has its own correlation name, Environment, and you must use this name in all ESQL statements that refer to, or set, the content of this tree.

Related reference:

"Complex ESQL functions" on page 5242

"LIST constructor function" on page 5269

"ROW and LIST combined" on page 5270

"ROW and LIST comparisons" on page 5271
You can compare ROWs and LISTs against other ROWs and LISTs.

LIST constructor function:
The LIST constructor complex function is used to explicitly generate lists of values that can be assigned to fields in an output message.

Syntax

A LIST consists of a sequence of unnamed values. When assigned to an array field reference (indicated by [] suffixed to the last element of the reference), each value is assigned in sequence to an element of the array. A LIST cannot be assigned to a non-array field reference.

Examples:
Example 1

Given the following XML message input body:

```
<Car>
   <size>big</size>
   <color>red</color>
</Car>
```

The following ESQL:

```
SET OutputRoot.XMLNS.Data.Result[] = LIST{InputBody.Car.colour,
                                                    'green',
                                                    'blue'};
```

produces the following results:

```
<Data>
   <Result>red</Result>
   <Result>green</Result>
   <Result>blue</Result>
</Data>
```

In the case of a LIST, there is no explicit name associated with each value. The values are assigned in sequence to elements of the message field array specified as the target of the assignment. Curly braces rather than parentheses are used to surround the LIST items.

Example 2

Given the following XML input message body:

```
<Data>
   <Field>Keats</Field>
   <Field>Shelley</Field>
   <Field>Wordsworth</Field>
   <Field>Tennyson</Field>
   <Field>Byron</Field>
</Data>
```

the following ESQL:

```
-- Copy the entire input message to the output message,
-- including the XML message field array as above
SET OutputRoot = InputRoot;
SET OutputRoot.XMLNS.Data.Field[] = LIST{'Henri','McGough','Patten'};
```

Produces the following output:

```
<Data>
   <Field>Henri</Field>
   <Field>McGough</Field>
   <Field>Patten</Field>
</Data>
```

The previous members of the `Data.Field[]` array have been discarded. Assigning a new list of values to an already existing message field array removes all the elements in the existing array before the new ones are assigned.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Accessing the environment tree" on page 2469
The environment tree has its own correlation name, Environment, and you must use this name in all ESQL statements that refer to, or set, the content of this tree.

Related reference:

"Complex ESQL functions" on page 5242

"ROW constructor function" on page 5267

"ROW and LIST combined"

"ROW and LIST comparisons" on page 5271
You can compare ROWs and LISTs against other ROWs and LISTs.

ROW and LIST combined:
ROW and LIST combined form a complex function.

A ROW might validly be an element in a LIST. For example:

```
SET OutputRoot.XMLNS.Data.Country[] =
         LIST{ROW('UK' AS name,'pound' AS currency),
              ROW('US' AS name, 'dollar' AS currency),
                                'default'};
```

produces the following result:

```
<Data>
   <Country>
      <name>UK</name>
      <currency>pound</currency>
   </Country>
   <Country>
      <name>US</name>
      <currency>dollar</currency>
   </Country>
   <Country>default</Country>
</Data>
```

ROW and non-ROW values can be freely mixed within a LIST.

A LIST cannot be a member of a ROW. Only named scalar values can be members of a ROW.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Accessing the environment tree" on page 2469
The environment tree has its own correlation name, Environment, and you must use this name in all ESQL statements that refer to, or set, the content of this tree.

Related reference:

"Complex ESQL functions" on page 5242

"ROW constructor function" on page 5267

"LIST constructor function" on page 5269

"ROW and LIST comparisons"
You can compare ROWs and LISTs against other ROWs and LISTs.

ROW and LIST comparisons:

You can compare ROWs and LISTs against other ROWs and LISTs.

Examples:
Example 1

```
IF ROW(InputBody.Data.*[1],InputBody.Data.*[2]) =
                ROW('Raf' AS Name,'25' AS Age) THEN ...
IF LIST{InputBody.Data.Name, InputBody.Data.Age} = LIST{'Raf','25'} THEN ...
```

With the following XML input message body both the IF expressions in both the above statements evaluate to TRUE:

```
<Data>
   <Name>Raf</Name>
   <Age>25</Age>
</Data>
```

In the comparison between ROWs, both the name and the value of each element are compared; in the comparison between LISTs only the value of each element is compared. In both cases, the cardinality and sequential order of the LIST or ROW operands being compared must be equal in order for the two operands to be equal. Therefore, the following examples are false because either the sequential order or the cardinality of the operands being compared do not match:

```
ROW('alpha' AS A, 'beta' AS B) =
             ROW('alpha' AS A, 'beta' AS B, 'delta' AS D)
ROW('alpha' AS A, 'beta' AS B) =
             ROW('beta' AS B,'alpha' AS A)
LIST{1,2,3} = LIST{1,2,3,4}
LIST{3,2,1} = LIST{1,2,3}
```

Example 2

Consider the following ESQL:

```
IF InputBody.Places =
   ROW('Ken' AS first, 'Bob' AS second, 'Kate' AS third) THEN ...
```

With the following XML input message body, the above IF expression evaluates to TRUE:

```
<Places>
   <first>Ken</first>
   <second>Bob</second>
   <third>Kate</third>
</Places>
```

The presence of an explicitly-constructed ROW as one of the operands to the comparison operator results in the other operand also being treated as a ROW.

Contrast this with a comparison such as:

```
IF InputBody.Lottery.FirstDraw = InputBody.Lottery.SecondDraw THEN ...
```

which compares the value of the FirstDraw and SecondDraw fields, not the names and values of each of FirstDraw and SecondDraw's child fields constructed as a ROW. Thus an XML input message body such as:

```
<Lottery>
   <FirstDraw>wednesday
      <ball1>32</ball1>
      <ball2>12</ball2>
   </FirstDraw>
   <SecondDraw>saturday
      <ball1>32</ball1>
      <ball2>12</ball2>
   </SecondDraw>
</Lottery>
```

would not result in the above IF expression being evaluated as TRUE, because the values wednesday and saturday are being compared, not the names and values of the ball fields.

Example 3

Consider the following ESQL:

```
IF InputBody.Cities.City[] = LIST{'Athens','Sparta','Thebes'} THEN ...
```

With the following XML input message body, the IF expression evaluates to TRUE:

```
<Cities>
 <City>Athens</City>
 <City>Sparta</City>
 <City>Thebes</City>
</Cities>
```

Two message field arrays can be compared together in this way, for example:

```
IF InputBody.Cities.Mediaeval.City[] =
                    InputBody.Cities.Modern.City[] THEN ...

IF InputBody.Cities.Mediaeval.*[] = InputBody.Cities.Modern.*[] THEN ...

IF InputBody.Cities.Mediaeval.(XML.Element)[] =
                    InputBody.Cities.Modern.(XML.Element)[] THEN ...
```

With the following XML input message body, the IF expression of the first and third of the statements above evaluates to TRUE:

```
<Cities>
   <Mediaeval>1350
      <City>London</City>
      <City>Paris</City>
   </Mediaeval>
   <Modern>1990
      <City>London</City>
      <City>Paris</City>
   </Modern>
</Cities>
```

However the IF expression of the second statement evaluates to FALSE, because the *[] indicates that all the children of Mediaeval and Modern are to be compared, not just the (XML.Element)s. In this case the values 1350 and 1990, which form nameless children of Mediaeval and Modern, are compared as well as the values of the City tags.

The IF expression of the third statement above evaluates to TRUE with an XML input message body such as:

```
<Cities>
   <Mediaeval>1350
      <Location>London</Location>
      <Location>Paris</Location>
   </Mediaeval>
   <Modern>1990
      <City>London</City>
      <City>Paris</City>
   </Modern>
</Cities>
```

LISTs are composed of unnamed values. It is the values of the child fields of Mediaeval and Modern that are compared, not their names.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"Complex ESQL functions" on page 5242

"ROW constructor function" on page 5267

"LIST constructor function" on page 5269

"ROW and LIST combined" on page 5270

Supported casts:
This topic lists the CASTs that are supported between combinations of data-types.

A CAST is not supported between every combination of data-types. Those that are supported are listed in the following table, along with the effect of the CAST.

When casting, there can be a one-to-one or a many-to-one mapping between the source data-type and the target data-type. An example of a one-to-one mapping is where the source data-type is a single integer and the target data-type a single float. An example of a many-to-one mapping is where the source data consists of three integers that are converted to a single date. Table 265 lists the supported one-to-one casts. Table 266 on page 5280 lists the supported many-to-one casts.

See "ESQL data types" on page 2373 for information about `precision`, `scale`, and `interval qualifier`.

Table 265. Supported casts: one-to-one mappings of source to target data-type

Source data-type	Target data-type	Effect
BIT	BIT	The result is the same as the input.
BIT	BLOB	The bit array is converted to a byte array with a maximum of 2^{63} elements. An error is reported if the source is not of a suitable length to produce a BLOB (that is a multiple of 8).
BIT	CHARACTER	The result is a string conforming to the definition of a bit string literal whose interpreted value is the same as the source value. The resulting string has the form B'bbbbbb' (where b is either 0 or 1). If you specify either a CCSID or ENCODING clause, the bit array is assumed to be characters in the specified CCSID and encoding, and is code-page converted into the character return value. If you specify only a CCSID, big endian encoding is assumed. If you specify only an encoding, a CCSID of 1208 is assumed. This function reports conversion errors if the code page or encoding are unknown, the data supplied is not an integral number of characters of the code page, or the data contains characters that are not valid in the given code page.
BIT	INTEGER	The bit array has a maximum of 2^{63} elements and is converted to an integer. An error is reported if the source is not of the correct length to match an integer.
BLOB	BIT	The given byte array is converted to a bit array with a maximum of 2^{63} elements.
BLOB	BLOB	The result is the same as the input.
BLOB	CHARACTER	The result is a string conforming to the definition of a binary string literal whose interpreted value is the same as the source value. The resulting string has the form X'hhhh' (where h is any hexadecimal character). If you specify either a CCSID or ENCODING clause, the byte array is assumed to be characters in the specified CCSID and encoding, and is code-page converted into the character return value. If you specify only a CCSID, big endian encoding is assumed. If you specify only an encoding, a CCSID of 1208 is assumed. This function reports conversion errors if the code page or encoding are unknown, the data supplied is not an integral number of characters of the code page, or the data contains characters that are not valid in the given code page.

Table 265. Supported casts: one-to-one mappings of source to target data-type (continued)

Source data-type	Target data-type	Effect
BLOB	INTEGER	The byte array has a maximum of 2^{63} elements and is converted to an integer. An error is reported if the source is not of the correct length to match an integer.
BOOLEAN	BOOLEAN	The result is the same as the input.
BOOLEAN	CHARACTER	If the source value is TRUE, the result is the character string TRUE. If the source value is FALSE, the result is the character string FALSE. Because the UNKNOWN Boolean value is the same as the NULL value for Booleans, the result is NULL if the source value is UNKNOWN.
CHARACTER	BIT	The character string must conform to the rules for a bit string literal or for the contents of the bit string literal. That is, the character string can be of the form B'bbbbbbb' or bbbbbb (where b' can be either 0 or 1). If you specify either a CCSID or ENCODING clause, the character string is converted into the specified CCSID and encoding and placed without further conversion into the bit array return value. If you specify only a CCSID, big endian encoding is assumed. If you specify only an encoding, a CCSID of 1208 is assumed. This function reports conversion errors if the code page or encoding are unknown or the data contains Unicode characters that cannot be converted to the given code page.
CHARACTER	BLOB	This cast can work in two ways: 1. If you specify either a CCSID or ENCODING clause, the whole string is written out in the code page or encoding that you requested. For example, the string "Cat" in CCSID 850 becomes the three-byte array in hexadecimal, 43,61,74. 2. If you specify neither the CCSID nor ENCODING clause, the string must itself contain two-character hexadecimal digits of the form X'hhhhhh' or hhhhhh (where h can be any hexadecimal characters). In this case, the input string "436174" becomes the same three-byte binary array (43,61,74). Note that an error is generated if the input string is not of the correct format. If you specify only a CCSID, big endian encoding is assumed. If you specify only an encoding, a CCSID of 1208 is assumed. This function reports conversion errors if the code page or encoding are unknown or the data contains Unicode characters that cannot be converted to the given code page.
CHARACTER	BOOLEAN	The character string is interpreted in the same way as a Boolean literal. That is, the character string must be one of the strings TRUE, FALSE, or UNKNOWN (in any case combination).
CHARACTER	CHARACTER	The result is the same as the input.
CHARACTER	DATE	If a FORMAT clause is not specified, the character string must conform to the rules for a date literal or the date string. That is, the character string can be either DATE '2002-10-05' or 2002-10-05. See also “Formatting and parsing dateTimes as strings” on page 5253.

Table 265. Supported casts: one-to-one mappings of source to target data-type (continued)

Source data-type	Target data-type	Effect
CHARACTER	DECIMAL	The character string is interpreted in the same way as an exact numeric literal to form a temporary decimal result with a `scale` and `precision` defined by the format of the string. This is converted into a decimal of the specified `precision` and `scale`, with a runtime error being generated if the conversion results in loss of significant digits. If you do not specify the `precision` and `scale`, the `precision` and `scale` of the result are the minimum necessary to hold the given value. The behavior changes if the FORMAT clause is specified. See also "Formatting and parsing numbers as strings" on page 5250.
CHARACTER	FLOAT	The character string is interpreted in the same way as a floating point literal. The behavior changes if the FORMAT clause is specified. See also "Formatting and parsing numbers as strings" on page 5250.
CHARACTER	GMTTIME	The character string must conform to the rules for a GMT time literal or the time string. That is, the character string can be either GMTTIME '09:24:15' or 09:24:15. The behavior changes if the FORMAT clause is specified. See also "Formatting and parsing dateTimes as strings" on page 5253.
CHARACTER	GMTTIMESTAMP	The character string must conform to the rules for a GMT timestamp literal or the timestamp string. That is, the character string can be either GMTTIMESTAMP '2002-10-05 09:24:15' or 2002-10-05 09:24:15. The behavior changes if the FORMAT clause is specified. See also "Formatting and parsing dateTimes as strings" on page 5253.
CHARACTER	INTEGER	The character string is interpreted in the same way as an integer literal. The behavior changes if the FORMAT clause is specified. See also "Formatting and parsing numbers as strings" on page 5250.
CHARACTER	INTERVAL	The character string must conform to the rules for an interval literal with the same `interval qualifier` as specified in the CAST function, or it must conform to the rules for an interval string that apply for the specified `interval qualifier`.
CHARACTER	TIME	The character string must conform to the rules for a time literal or for the time string. That is, the character string can be either TIME '09:24:15' or 09:24:15. The behavior changes if the FORMAT clause is specified. See also "Formatting and parsing dateTimes as strings" on page 5253.
CHARACTER	TIMESTAMP	The character string must conform to the rules for a timestamp literal or for the timestamp string. That is, the character string can be either TIMESTAMP '2002-10-05 09:24:15' or 2002-10-05 09:24:15. The behavior changes if the FORMAT clause is specified. See also "Formatting and parsing dateTimes as strings" on page 5253.

Table 265. Supported casts: one-to-one mappings of source to target data-type (continued)

Source data-type	Target data-type	Effect

DATE	CHARACTER	The result is a string conforming to the definition of a date literal, whose interpreted value is the same as the source date value.
		
		For example:
		`CAST(DATE '2002-10-05' AS CHARACTER)`
		
		returns
		`DATE '2002-10-05'`
		
		The behavior changes if the FORMAT clause is specified. See also "Formatting and parsing dateTimes as strings" on page 5253.

DATE	DATE	The result is the same as the input.
DATE	GMTTIMESTAMP	The result is a value whose date fields are taken from the source date value, and whose time fields are taken from the current GMT time.
DATE	TIMESTAMP	The result is a value whose date fields are taken from the source date value, and whose time fields are taken from the current time.
DECIMAL	CHARACTER	The result is the shortest character string that conforms to the definition of an exact numeric literal and whose interpreted value is the value of the decimal. The behavior changes if the FORMAT clause is specified. See also "Formatting and parsing numbers as strings" on page 5250.
DECIMAL	DECIMAL	The value is converted to the specified `precision` and `scale`, with a runtime error being generated if the conversion results in loss of significant digits. If you do not specify the `precision` and `scale`, the value, `precision` and `scale` are preserved; it is a NOOP (no operation).
DECIMAL	FLOAT	The number is converted, with rounding if necessary.
DECIMAL	INTEGER	The value is rounded and converted into an integer, with a runtime error being generated if the conversion results in loss of significant digits.
DECIMAL	INTERVAL	If the `interval qualifier` specified has only one field, the result is an interval with that qualifier with the field equal to the value of the exact numeric. Otherwise a runtime error is generated.
FLOAT	CHARACTER	The result is the shortest character string that conforms to the definition of an approximate numeric literal and whose mantissa consists of a single digit that is not 0, followed by a period and an unsigned integer, and whose interpreted value is the value of the float. The behavior changes if the FORMAT clause is specified. See also "Formatting and parsing numbers as strings" on page 5250. When you CAST a FLOAT to a DECIMAL or CHARACTER, either implicitly or explicitly, the FLOAT can be rounded to a maximum precision of 15 digits.
FLOAT	FLOAT	The result is the same as the input.

Table 265. Supported casts: one-to-one mappings of source to target data-type (continued)

Source data-type	Target data-type	Effect
FLOAT	DECIMAL	The value is rounded and converted into a decimal of the specified `precision` and `scale`, with a runtime error being generated if the conversion results in loss of significant digits. If you do not specify the `precision` and `scale`, the `precision` and `scale` of the result are the minimum necessary to hold the given value. When you CAST a FLOAT to a DECIMAL or CHARACTER, either implicitly or explicitly, the FLOAT can be rounded to a maximum precision of 15 digits.
FLOAT	INTEGER	The value is rounded and converted into an integer, with a runtime error being generated if the conversion results in loss of significant digits.
FLOAT	INTERVAL	If the specified `interval qualifier` has only one field, the result is an interval with that qualifier with the field equal to the value of the numeric. Otherwise a runtime error is generated.
GMTTIME	CHARACTER	The result is a string conforming to the definition of a GMTTIME literal whose interpreted value is the same as the source value. The resulting string has the form GMTTIME 'hh:mm:ss'. The behavior changes if the FORMAT clause is specified. See also “Formatting and parsing dateTimes as strings” on page 5253.
GMTTIME	GMTTIME	The result is the same as the input.
GMTTIME	TIME	The resulting value is the source value plus the local time zone displacement (as returned by LOCAL_TIMEZONE). The hours field is calculated modulo 24.
GMTTIME	GMTTIMESTAMP	The result is a value whose date fields are taken from the current date, and whose time fields are taken from the source GMT time.
GMTTIME	TIMESTAMP	The result is a value whose date fields are taken from the current date, and whose time fields are taken from the source GMT time, plus the local time zone displacement (as returned by LOCAL_TIMEZONE).
GMTTIMESTAMP	CHARACTER	The result is a string conforming to the definition of a GMTTIMESTAMP literal whose interpreted value is the same as the source value. The resulting string has the form GMTTIMESTAMP 'yyyy-mm-dd hh:mm:ss'. The behavior changes if the FORMAT clause is specified. See also “Formatting and parsing dateTimes as strings” on page 5253.
GMTTIMESTAMP	DATE	The result is a value whose fields consist of the date fields of the source GMTTIMESTAMP value.
GMTTIMESTAMP	GMTTIME	The result is a value whose fields consist of the time fields of the source GMTTIMESTAMP value.
GMTTIMESTAMP	TIME	The result is a value whose time fields are taken from the source GMTTIMESTAMP value, plus the local time zone displacement (as returned by LOCAL_TIMEZONE). The hours field is calculated modulo 24.
GMTTIMESTAMP	GMTTIMESTAMP	The result is the same as the input.
GMTTIMESTAMP	TIMESTAMP	The resulting value is source value plus the local time zone displacement (as returned by LOCAL_TIMEZONE).
INTEGER	BIT	The given integer is converted to a bit array with a maximum of 2^{63} elements.

Table 265. Supported casts: one-to-one mappings of source to target data-type (continued)

Source data-type	Target data-type	Effect
INTEGER	BLOB	The given integer is converted to a byte array with a maximum of 2^{63} elements.
INTEGER	CHARACTER	The result is the shortest character string that conforms to the definition of an exact numeric literal and whose interpreted value is the value of the integer. The behavior changes if the FORMAT clause is specified. See also "Formatting and parsing numbers as strings" on page 5250.
INTEGER	FLOAT	The number is converted, with rounding if necessary.
INTEGER	INTEGER	The result is the same as the input.
INTEGER	DECIMAL	The value is converted into a decimal of the specified `precision` and `scale`, with a runtime error being generated if the conversion results in loss of significant digits. If you do not specify the `precision` and `scale`, the `precision` and `scale` of the result are the minimum necessary to hold the given value.
INTEGER	INTERVAL	If the `interval qualifier` specified has only one field, the result is an interval with that qualifier with the field equal to the value of the exact numeric. Otherwise a runtime error is generated.

INTERVAL	CHARACTER	The result is a string conforming to the definition of an INTERVAL literal, whose interpreted value is the same as the source interval value.
		
		For example:
		`CAST(INTERVAL '4' YEARS AS CHARACTER)`
		
		returns
		`INTERVAL '4' YEARS`

INTERVAL	DECIMAL	If the interval value has a qualifier that has only one field, the result is a decimal of the specified `precision` and `scale` with that value, with a runtime error being generated if the conversion results in loss of significant digits. If the interval has a qualifier with more than one field, such as YEAR TO MONTH, a runtime error is generated. If you do not specify the `precision` and `scale`, the `precision` and `scale` of the result are the minimum necessary to hold the given value.
INTERVAL	FLOAT	If the interval value has a qualifier that has only one field, the result is a float with that value. If the interval has a qualifier with more than one field, such as YEAR TO MONTH, a runtime error is generated.
INTERVAL	INTEGER	If the interval value has a qualifier that has only one field, the result is an integer with that value. If the interval has a qualifier with more than one field, such as YEAR TO MONTH, a runtime error is generated.
INTERVAL	INTERVAL	The result is the same as the input. Year-month intervals can be converted only to year-month intervals, and day-second intervals only to day-second intervals. The source interval is converted into a scalar in units of the least significant field of the target `interval qualifier`. This value is normalized into an interval with the target `interval qualifier`. For example, to convert an interval that has the qualifier MINUTE TO SECOND into an interval with the qualifier DAY TO HOUR, the source value is converted into a scalar in units of hours, and this value is normalized into an interval with qualifier DAY TO HOUR.

Table 265. Supported casts: one-to-one mappings of source to target data-type (continued)

Source data-type	Target data-type	Effect

TIME	CHARACTER	The result is a string conforming to the definition of a TIME literal, whose interpreted value is the same as the source time value.
		For example:
		`CAST(TIME '09:24:15' AS CHARACTER)`
		returns
		`TIME '09:24:15'`
		The behavior changes if the FORMAT clause is specified. See also "Formatting and parsing dateTimes as strings" on page 5253.

TIME	GMTTIME	The result value is the source value minus the local time zone displacement (as returned by LOCAL_TIMEZONE). The hours field is calculated modulo 24.
TIME	GMTTIMESTAMP	The result is a value whose date fields are taken from the current date, and whose time fields are taken from the source GMT time, minus the local time zone displacement (as returned by LOCAL_TIMEZONE).
TIME	TIME	The result is the same as the input.
TIME	TIMESTAMP	The result is a value whose date fields are taken from the current date, and whose time fields are taken from the source time value.

TIMESTAMP	CHARACTER	The result is a string conforming to the definition of a TIMESTAMP literal, whose interpreted value is the same as the source timestamp value.
		For example:
		`CAST(TIMESTAMP '2002-10-05 09:24:15' AS CHARACTER)`
		returns
		`TIMESTAMP '2002-10-05 09:24:15'`
		The behavior changes if the FORMAT clause is specified. See also "Formatting and parsing dateTimes as strings" on page 5253.

TIMESTAMP	DATE	The result is a value whose fields consist of the date fields of the source timestamp value.
TIMESTAMP	GMTTIME	The result is a value whose time fields are taken from the source TIMESTAMP value, minus the local time zone displacement (as returned by LOCAL_TIMEZONE). The hours field is calculated modulo 24.
TIMESTAMP	GMTTIMESTAMP	The resulting value is the source value minus the local time zone displacement (as returned by LOCAL_TIMEZONE).
TIMESTAMP	TIME	The result is a value whose fields consist of the time fields of the source timestamp value.
TIMESTAMP	TIMESTAMP	The result is the same as the input.

Table 266. Supported casts: many-to-one mappings of source to target data-type

Source data-type	Target data-type	Effect
Numeric, Numeric, Numeric	DATE	Creates a DATE value from the numerics in the order year, month, and day. Non-integer values are rounded.
Numeric, Numeric, Numeric	TIME	Creates a TIME value from the numerics in the order hours, minutes, and seconds. Non-integer values for hours and minutes are rounded.

Table 266. Supported casts: many-to-one mappings of source to target data-type (continued)

Source data-type	Target data-type	Effect
Numeric, Numeric, Numeric	GMTIME	Creates a GMTTIME value from the numerics in the order of hours, minutes, and seconds. Non-integer values for hours and minutes are rounded.
Numeric, Numeric, Numeric, Numeric, Numeric, Numeric	TIMESTAMP	Creates a TIMESTAMP value from the numerics in the order years, months, days, hours, minutes, and seconds. Non-integer values for years, months, days, hours, and minutes are rounded.
Numeric, Numeric, Numeric, Numeric, Numeric, Numeric	GMTTIMESTAMP	Creates a GMTIMESTAMP value from the numerics in the order years, months, days, hours, minutes, and seconds. Non-integer values for years, months, days, hours, and minutes are rounded.
DATE, TIME	TIMESTAMP	The result is a TIMESTAMP value with the given DATE and TIME.
DATE, GMTTIME	GMTIMESTAMP	The result is a GMTTIMESTAMP value with the given DATE and GMTTIME.
Numeric, Numeric	INTERVAL YEAR TO MONTH	The result is an INTERVAL with the first source as years and the second as months. Non-integer values are rounded.
Numeric, Numeric	INTERVAL HOUR TO MINUTE	The result is an INTERVAL with the first source as hours and the second as minutes. Non-integer values are rounded.
Numeric, Numeric, Numeric	INTERVAL HOUR TO SECOND	The result is an INTERVAL with the sources as hours, minutes, and seconds, respectively. Non-integer values for hours and minutes are rounded.
Numeric, Numeric	INTERVAL MINUTE TO SECOND	The result is an INTERVAL with the sources as minutes and seconds, respectively. Non-integer values for minutes are rounded.
Numeric, Numeric	INTERVAL DAY TO HOUR	The result is an INTERVAL with the sources as days and hours, respectively. Non-integer values are rounded.
Numeric, Numeric, Numeric	INTERVAL DAY TO MINUTE	The result is an INTERVAL with the sources as days, hours, and minutes, respectively. Non-integer values are rounded.
Numeric, Numeric, Numeric, Numeric	INTERVAL DAY TO SECOND	The result is an INTERVAL with the sources as days, hours, minutes, and seconds, respectively. Non-integer values for days, hours, and minutes are rounded.
Numeric	INTERVAL YEAR	The result is an INTERVAL with the source as years, rounded if necessary.
Numeric	INTERVAL MONTH	The result is an INTERVAL with the source as months, rounded if necessary.
Numeric	INTERVAL DAY	The result is an INTERVAL with the source as days, rounded if necessary.
Numeric	INTERVAL HOUR	The result is an INTERVAL with the source as hours, rounded if necessary.
Numeric	INTERVAL MINUTE	The result is an INTERVAL with the source as minutes, rounded if necessary.
Numeric	INTERVAL SECOND	The result is an INTERVAL with the source as seconds.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"ESQL data types" on page 2373
A data type defines the characteristics of an item of data, and determines how that data is processed. ESQL supports six data types, listed later in this section. Data that is retrieved from databases, received in a self-defining message, or defined in a message model (using MRM data types), is mapped to one of these basic ESQL types when it is processed in ESQL expressions.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"Complex ESQL functions" on page 5242

"CAST function" on page 5245

"Implicit casts"

"Data types of values from external databases" on page 5288
How database data types are implicitly cast to ESQL data types.

"Supported code pages" on page 4176
Application messages must conform to supported code pages.

Implicit casts:
This topic discusses implicit casts.

It is not always necessary to cast values between types. Some casts are done implicitly. For example, numbers are implicitly cast between the three numeric types for the purposes of comparison and arithmetic. Character strings are also implicitly cast to other data types for the purposes of comparison.

There are three situations in which a data value of one type is cast to another type implicitly. The behavior and restrictions of the implicit cast are the same as described for the explicit cast function, except where noted in the following topics.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"Complex ESQL functions" on page 5242

"Implicit CASTs for comparisons"
The standard SQL comparison operators>, <,>=, <=, =, <> are supported for comparing two values in ESQL.

"Implicit CASTs for arithmetic operations" on page 5285

"Implicit CASTs for assignment" on page 5287

Implicit CASTs for comparisons:

The standard SQL comparison operators>, <,>=, <=, =, <> are supported for comparing two values in ESQL.

When the data types of the two values are not the same, one of them can be implicitly cast to the type of the other to allow the comparison to proceed. In the following table, the vertical axis represents the left hand operand, the horizontal axis represents the right hand operand.

L means that the right hand operand is cast to the type of the left hand operand before comparison; R means the opposite; X means that no implicit casting takes place; a blank means that comparison between the values of the two data types is not supported.

	ukn	**bln**	**int**	**float**	**dec**	**char**	**time**	**gtm**	**date**	**ts**	**gts**	**ivl**	**blob**	**bit**
ukn														
bln		X				L								
int			X	R	R	L								
float			L	X	L	L								
dec			L	R	X	L								
chr		R	R	R	R	X	R	R	R	R	R	R[1]	R	R
tm						L	X	L						
gtm						L	R	X						
dt						L			X	R[2]	R[2]			
ts						L			L[2]	X	L			
gts						L			L[2]	R	X			
ivl						L[1]						X		
blb						L							X	
bit						L								X

Notes:

1. When casting from a character string to an interval, the character string must be of the format INTERVAL '<values>' <qualifier>. The format <values>, which is allowede for an explicit CAST, is not allowed here because no qualifier external to the string is supplied.
2. When casting from a DATE to a TIMESTAMP or GMTTIMESTAMP, the time portion of the TIMESTAMP is set to all zero values (00:00:00). This is different from the behavior of the explicit cast, which sets the time portion to the current time.

Numeric types:
The comparison operators operate on all three numeric types.

Character strings:
You cannot define an alternative collation order that, for example, collates upper and lowercase characters equally.

When comparing character strings, trailing blanks are not significant, so the comparison `'hello' = 'hello '` returns true.

Datetime values:
Datetime values are compared in accordance with the natural rules of the Gregorian calendar and clock.

You can compare the time zone you are working in with the GMT time zone. The GMT time zone is converted into a local time zone based on the difference between your local time zone and the GMT time specified. When you compare your local time with the GMT time, the comparison is based on the difference at a given time on a given date.

Conversion is always based on the value of LOCAL_TIMEZONE. This is because GMT timestamps are converted to local timestamps only if it can be done unambiguously. Converting a local timestamp to a GMT timestamp has difficulties around the daylight saving cut-over time, and converting between times and GMT times (without date information) has to be done based on the LOCAL_TIMEZONE value, because you cannot specify which time zone difference to use otherwise.

Booleans:
Boolean values can be compared using all the normal comparison operators. The TRUE value is defined to be greater than the FALSE value. Comparing either value to the UNKNOWN Boolean value (which is equivalent to NULL) returns an UNKNOWN result.

Intervals:
Intervals are compared by converting the two interval values into intermediate representations, so that both intervals have the same interval qualifier. Year-month intervals can be compared only with other year-month intervals, and day-second intervals can be compared only with other day-second intervals.

For example, if an interval in minutes, such as `INTERVAL '120' MINUTE` is compared with an interval in days to seconds, such as `INTERVAL '0 02:01:00'`, the two intervals are first converted into values that have consistent interval qualifiers, which can be compared. So, in this example, the first value is converted into an interval in days to seconds, which gives `INTERVAL '0 02:00:00'`, which can be compared with the second value.

Comparing character strings with other types:
If a character string is compared with a value of another type, the broker attempts to cast the character string into a value of the same data type as the other value.

For example, you can write an expression:

```
'1234'> 4567
```

The character string on the left is converted into an integer before the comparison takes place. This behavior reduces some of the need for explicit CAST operators when comparing values derived from a generic XML message with literal values. (For details of explicit casts that are supported, see “Supported casts” on page 5273.) It is this facility that allows you to write the following expression:

```
Body.Trade.Quantity> 5000
```

In this example, the field reference on the left evaluates to the character string '1000' and, because this is being compared to an integer, that character string is converted into an integer before the comparison takes place.

You must still check whether the price field that you want interpreted as a decimal is greater than a given threshold. Make sure that the literal you compare it to is a decimal value and not an integer.

Consider the following example:

```
Body.Trade.Price> 100
```

This comparison does not return the required or expected result, because the `Price` field is converted into an integer, and that conversion fails because the character string contains a decimal point. However, the following expression succeeds:

```
Body.Trade.Price> 100.00
```

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"Complex ESQL functions" on page 5242

"Supported casts" on page 5273

"Implicit casts" on page 5282

"Implicit CASTs for arithmetic operations"

"Implicit CASTs for assignment" on page 5287

Implicit CASTs for arithmetic operations:
This topic lists the implicit CASTs available for arithmetic operations.

Normally the arithmetic operators (+, -, *, and /) operate on operands of the same data type, and return a value of the same data type as the operands. Cases where it is acceptable for the operands to be of different data types, or where the data type of the resulting value is different from the type of the operands, are shown in the following table.

The following table lists the implicit CASTs for arithmetic operation.

Left operand data type	Right operand data type	Supported operators	Result data type
INTEGER	FLOAT	+, -, *, /	FLOAT[1]
INTEGER	DECIMAL	+, -, *, /	DECIMAL[1]
INTEGER	INTERVAL	*	INTERVAL[4]
FLOAT	INTEGER	+, -, *, /	FLOAT[1]
FLOAT	DECIMAL	+, -, *, /	FLOAT[1]
FLOAT	INTERVAL	*	INTERVAL[4]
DECIMAL	INTEGER	+, -, *, /	DECIMAL[1]
DECIMAL	FLOAT	+, -, *, /	FLOAT[1]
DECIMAL	INTERVAL	*	INTERVAL[4]
TIME	TIME	-	INTERVAL[2]
TIME	GMTTIME	-	INTERVAL[2]
TIME	INTERVAL	+, -	TIME[3]
GMTTIME	TIME	-	INTERVAL[2]
GMTTIME	GMTTIME	-	INTERVAL[2]
GMTTIME	INTERVAL	+, -	GMTTIME[3]

Left operand data type	Right operand data type	Supported operators	Result data type
DATE	DATE	-	INTERVAL[2]
DATE	INTERVAL	+, -	DATE[3]
TIMESTAMP	TIMESTAMP	-	INTERVAL[2]
TIMESTAMP	GMTTIMESTAMP	-	INTERVAL[2]
TIMESTAMP	INTERVAL	+, -	TIMESTAMP[3]
GMTTIMESTAMP	TIMESTAMP	-	INTERVAL[2]
GMTTIMESTAMP	GMTTIMESTAMP	-	INTERVAL[2]
GMTTIMESTAMP	INTERVAL	+, -	GMTTIMESTAMP[3]
INTERVAL	INTEGER	*, /	INTERVAL[4]
INTERVAL	FLOAT	*, /	INTERVAL[4]
INTERVAL	DECIMAL	*, /	INTERVAL[4]
INTERVAL	TIME	+	TIME[3]
INTERVAL	GMTTIME	+	GMTTIME[3]
INTERVAL	DATE	+	DATE[3]
INTERVAL	TIMESTAMP	+	TIMESTAMP[3]
INTERVAL	GMTTIMESTAMP	+	GMTTIMESTAMP[3]

Notes:

1. The operand that does not match the data type of the result is cast to the data type of the result before the operation proceeds. For example, if the left operand to an addition operator is an INTEGER, and the right operand is a FLOAT, the left operand is cast to a FLOAT before the addition operation is performed.
2. Subtracting a (GMT)TIME value from a (GMT)TIME value, a DATE value from a DATE value, or a (GMT)TIMESTAMP value from a (GMT)TIMESTAMP value, results in an INTERVAL value representing the time interval between the two operands.
3. Adding or subtracting an INTERVAL from a (GMT)TIME, DATE or (GMT)TIMESTAMP value results in a new value of the data type of the non-INTERVAL operand, representing the point in time represented by the original non-INTERVAL, plus or minus the length of time represented by the INTERVAL.
4. Multiplying or dividing an INTERVAL by an INTEGER, FLOAT, or DECIMAL value results in a new INTERVAL representing the length of time represented by the original, multiplied or divided by the factor represented by the non-INTERVAL operand. For example, an INTERVAL value of 2 hours 16 minutes multiplied by a FLOAT value of 2.5 results in a new INTERVAL value of 5 hours 40 minutes. The intermediate calculations involved in multiplying or dividing the original INTERVAL are carried out in the data type of the non-INTERVAL, but the individual fields of the INTERVAL (such as HOUR, YEAR, and so on) are always integral, so some rounding errors might occur.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"Complex ESQL functions" on page 5242

"Implicit casts" on page 5282

"Implicit CASTs for comparisons" on page 5282
The standard SQL comparison operators>, <,>=, <=, =, <> are supported for comparing two values in ESQL.

"Implicit CASTs for assignment"

Implicit CASTs for assignment:
Values can be assigned to one of three entities.

A message field (or equivalent in an exception or destination list)
Support for implicit conversion between the WebSphere Message Broker data types and the message (in its bitstream form) depends on the appropriate parser. For example, the XML parser casts everything as character strings before inserting them into the WebSphere MQ message.

A field in a database table

WebSphere Message Broker converts each of its data types into a suitable standard SQL C data type, as detailed in the following table. Conversion between this standard SQL C data type, and the data types supported by each DBMS, depends on the DBMS. Consult your DBMS documentation for more details.

The following table lists the available conversions from WebSphere Message Broker to SQL data types

WebSphere Message Broker data type	SQL data type
NULL, or unknown or invalid value	SQL_NULL_DATA
BOOLEAN	SQL_C_BIT
INTEGER	SQL_C_LONG
FLOAT	SQL_C_DOUBLE
DECIMAL	SQL_C_CHAR[1]
CHARACTER	SQL_C_CHAR
TIME	SQL_C_TIME
GMTTIME	SQL_C_TIME
DATE	SQL_C_DATE
TIMESTAMP	SQL_C_TIMESTAMP
GMTTIMESTAMP	SQL_C_DATE
INTERVAL	Not supported[2]
BLOB	SQL_C_BINARY
BIT	Not supported[2]

Notes:

1. For convenience, DECIMAL values are passed to the DBMS in character form.
2. There is no suitable standard SQL C data type for INTERVAL or BIT. Cast these to another data type, such as CHARACTER, if you need to assign them to a database field.

A scalar variable
When assigning to a scalar variable, if the data type of the value being assigned and that of the target variable data type are different, an implicit cast is attempted with the same restrictions and behavior as specified for the explicit CAST function. The only exception is when the data type of the variable is INTERVAL or DECIMAL.

In both these cases, the value being assigned is first cast to a CHARACTER value, and an attempt is made to cast the CHARACTER value to an INTERVAL or DECIMAL. This is because INTERVAL requires a qualifier and DECIMAL requires a precision and scale. These must be specified in the explicit cast, but must be obtained from the character string when implicitly casting. Therefore, a further restriction is that when implicitly casting to an INTERVAL variable, the character string must be of the form INTERVAL '<values>' <qualifier>. The shortened <values> form that is acceptable for the explicit cast is not acceptable here.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"Complex ESQL functions" on page 5242

"Implicit casts" on page 5282

"Implicit CASTs for comparisons" on page 5282
The standard SQL comparison operators>, <,>=, <=, =, <> are supported for comparing two values in ESQL.

"Implicit CASTs for arithmetic operations" on page 5285

Data types of values from external databases:

How database data types are implicitly cast to ESQL data types.

The ESQL data type of message fields depends on the type of the message (for example, XML), and the parser that is used to parse it. The ESQL data type of the value returned by a database column reference depends on the data type of the column in the database.

The following table shows how the various built-in database data types are cast to ESQL data types, when they are accessed by message flows that are running in a broker.

The versions that are supported for the database products shown in this table are listed in "Supported databases" on page 3591.

DB2	SQL Server and Sybase	Oracle	Informix	ESQL data type
N/A	BIT	N/A	N/A	BOOLEAN
SMALLINT, INTEGER, BIGINT	INT, SMALLINT, TINYINT	N/A	INT, SMALLINT	INTEGER
REAL, DOUBLE	FLOAT, REAL	NUMBER()[1]	FLOAT, SMALLFLOAT, DOUBLE	FLOAT
DECIMAL	DECIMAL, NUMERIC, MONEY, SMALLMONEY	NUMBER(P)[1], NUMBER(P,S)[1]	DECIMAL, MONEY	DECIMAL

DB2	SQL Server and Sybase	Oracle	Informix	ESQL data type
CHAR, VARCHAR, CLOB, GRAPHIC, VARGRAPHIC[2]	CHAR, VARCHAR, TEXT	CHAR, VARCHAR2, ROWID, UROWID, LONG, CLOB	CHAR, VARCHAR, CHAR VARYING	CHARACTER
TIME	N/A	N/A	N/A	TIME
N/A	N/A	N/A	N/A	GMTTIME
DATE	N/A	N/A	DATE	DATE
TIMESTAMP	DATETIME, SMALLDATETIME, TIMESTAMP	DATE	DATETIME	TIMESTAMP
N/A	N/A	N/A	N/A	GMTTIMESTAMP
N/A	N/A	N/A	INTERVAL	INTERVAL
BLOB	BINARY, VARBINARY, IMAGE, UNIQUEIDENTIFIER	RAW LONG, RAW BLOB	N/A	BLOB
N/A	N/A	N/A	N/A	BIT
XML	N/A	• Up to Version 7.0.0.2: CHAR • At Version 7.0.0.3 onwards: XML	N/A	BLOB

The table shows all data types that are supported for each database.

If an Oracle database column with NUMBER data type is defined with an explicit precision (P) and scale (S), it is cast to an ESQL DECIMAL value; otherwise it is cast to a FLOAT. For example, an ESQL statement like this:

```
SET OutputRoot.xxx[]
 = (SELECT T.department FROM Database.personnel AS T);
```

where `Database.personnel` resolves to a TINYINT column in an SQL Server database table, results in a list of ESQL INTEGER values being assigned to `OutputRoot.xxx`.

By contrast, an identical query, where `Database.personnel` resolves to a NUMBER() column in an Oracle database, results in a list of ESQL FLOAT values being assigned to `OutputRoot.xxx`.

Note, that data types of stored procedure parameters are cast using the definition of that external procedure in the “CREATE PROCEDURE statement” on page 5103.

Related concepts:

“ESQL overview” on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

“Developing ESQL” on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"Support for Unicode and DBCS data in databases" on page 3668
You can manipulate Unicode Standard version 3.0 data, in suitably configured databases, using ESQL, in nodes that access databases by ODBC. The broker does not support DBCS-only columns in tables that are defined in databases.

"Complex ESQL functions" on page 5242

"CAST function" on page 5245

"CREATE PROCEDURE statement" on page 5103
The CREATE PROCEDURE statement defines a callable function or procedure.

"Supported casts" on page 5273

"Implicit casts" on page 5282

"Supported databases" on page 3591
You can optionally configure databases to contain data that is accessed by your message flows. Databases from IBM and other suppliers are supported at specific versions on supported operating systems.

Miscellaneous ESQL functions:

ESQL provides additional functions that support miscellaneous operations.

- "BASE64DECODE function"
- "BASE64ENCODE function" on page 5291
- "CHANGEIDENTIFIERTIMEOUT function" on page 5292
- "COALESCE function" on page 5294
- "EVAL function" on page 5294
- "NULLIF function" on page 5296
- "PASSTHRU function" on page 5297
- "SLEEP function" on page 5299
- "UUIDASBLOB function" on page 5300
- "UUIDASCHAR function" on page 5301

Related tasks:

"Writing ESQL" on page 2413
How you can use ESQL to customize nodes.

Related reference:

"ESQL reference" on page 5019
SQL is the industry standard language for accessing and updating database data and ESQL is a language derived from SQL Version 3, particularly suited to manipulating both database and message data.

"ESQL operators" on page 5056
A list of the various groups of operators that ESQL supports.

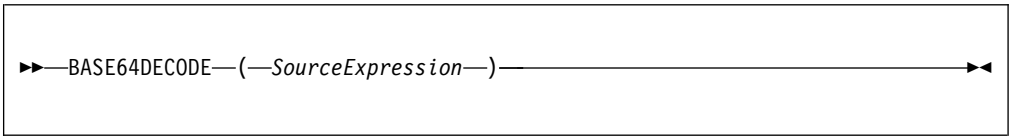
"ESQL statements" on page 5067
You can use ESQL statements to manipulate message trees, update databases, or interact with nodes.

"ESQL functions" on page 5168
The following types of function are available.

BASE64DECODE function:

BASE64DECODE is a function that manipulates CHARACTER strings that are base64-encoded, and returns a BLOB string that contains the base64-decoded version of the source string.

Syntax



BASE64DECODE returns a BLOB string containing a base64-decoded representation of the source string. The source string can only be of the CHARACTER data type. If *SourceExpression* is NULL, the result is NULL. If the source string does not contain valid base64 data, SQSTATE 'S22018' exception is produced.

Examples

For examples of usage see the BASE64ENCODE function.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"Syntax diagrams" on page 3677

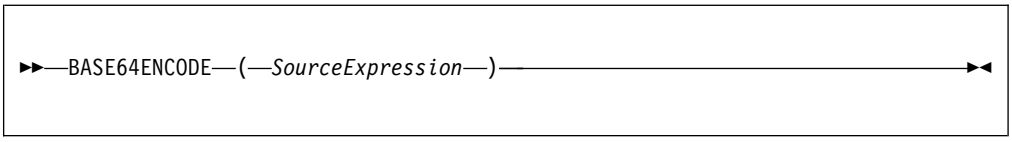
"Miscellaneous ESQL functions" on page 5290
ESQL provides additional functions that support miscellaneous operations.

BASE64ENCODE function:

BASE64ENCODE is a function that manipulates all string data types (BIT, BLOB, and CHARACTER).

BASE64ENCODE returns a CHARACTER string that contains the base64-encoded version of the source string.

Syntax



BASE64ENCODE returns a CHARACTER string containing a base64 representation of the source string. The source string can be a string of the CHARACTER, BLOB, or BIT data type. If *SourceExpression* is NULL, the result is NULL.

If *SourceExpression* is of CHARACTER type, it is first converted to the UTF-8 code page before encoding as base64.
If *SourceExpression* is of BLOB type, it is encoded as base64 directly, without any prior changes.

If *SourceExpression* is of BIT type, it is first CAST to BLOB before encoding as base64 and so its length must be a multiple of 8.

Examples

The base64 encoding of a BLOB source string and subsequent decoding back to BLOB is shown by the following example:

```
DECLARE original BLOB X'48656c6c6f';
DECLARE encoded CHARACTER BASE64ENCODE(original);
DECLARE decoded BLOB BASE64DECODE(encoded);
```

The base64 encoding of a CHARACTER source string that is first automatically converted to UTF-8 and later decoded is shown by the following example:

```
DECLARE original CHARACTER 'Hello World!';
DECLARE encoded CHARACTER BASE64ENCODE(original);
DECLARE decoded BLOB BASE64DECODE(encoded);
DECLARE decoded2 CHARACTER CAST(decoded AS CHARACTER CCSID 1208);
```

The base64 encoding of a BIT source string that is first automatically converted to a BLOB and later decoded is shown by the following example:

```
DECLARE original BIT B'0010001001000001';
DECLARE encoded CHARACTER BASE64ENCODE(original);
DECLARE decoded BLOB BASE64DECODE(encoded);
DECLARE decoded2 BIT CAST(decoded AS BIT);
```

Encoding from a CHARACTER source string to a BLOB and back to CHARACTER again in a code page other than UTF-8 is shown by the following example:

```
DECLARE original CHARACTER 'Hello World!';
DECLARE originalBlob BLOB CAST(original AS BLOB CCSID 819);
DECLARE encoded CHARACTER BASE64ENCODE(originalBlob);
DECLARE decoded BLOB BASE64DECODE(encoded);
DECLARE decoded2 CHARACTER CAST(decoded AS CHARACTER CCSID 819);
```

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"Syntax diagrams" on page 3677

"Miscellaneous ESQL functions" on page 5290
ESQL provides additional functions that support miscellaneous operations.

CHANGEIDENTIFIERTIMEOUT function:

The CHANGEIDENTIFIERTIMEOUT function changes the timeout value associated with the reply identifier of a SOAPInput node or the request identifier of an HTTPInput node. The function returns a Boolean value to indicate the success or otherwise of the change.

Syntax

The function returns TRUE if the given identifier is valid and the timeout was adjusted to the new value without the new value causing the identifier to timeout. It returns FALSE if the identifier passed to the function is invalid or, after the timeout adjustment, the identifier has expired. Note that if the function is passed an identifier that has already expired, the function always return FALSE. Note that for HTTPInput nodes, the function only processes identifiers that are associated with the embedded execution group listener.

The *IdentifierExpression* parameter is a BLOB expression that must resolve to a valid reply identifier (for a SOAPInput node) or request identifier (for an HTTPInput node). An identifier is unique to an individual message, therefore only the message associated with the identifier will be affected by this function.

The *TimeoutExpression* parameter is an INTEGER expression that represents a certain number of seconds to be added to or subtracted from the identifier's timeout value. Specifying a positive value will cause the timeout to be increased by the chosen amount, whilst specifying a negative value will cause the timeout to be decreased by the chosen amount. Passing a value of 0 allows an identifier to be tested for validity without changing it, because FALSE is returned if the identifier is invalid or has expired, whereas TRUE is returned if the identifier is valid and has not timed out.

If either parameter is NULL, the result is NULL.

Examples

The following example shows how to reduce the timeout of the specified SOAP reply identifier by 5 seconds. It returns TRUE if the identifier is valid and has not expired after changing the timeout duration:

```
ChangeIdentifierTimeout(
  LocalEnvironment.Destination.SOAP.Reply.ReplyIdentifier, -5);
```

The following example shows how to increase the timeout on the specified identifier by 30 seconds. It returns TRUE if the identifier is valid and has not expired after changing the timeout duration:

```
ChangeIdentifierTimeout(myReplyIdentifier, 30);
```

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

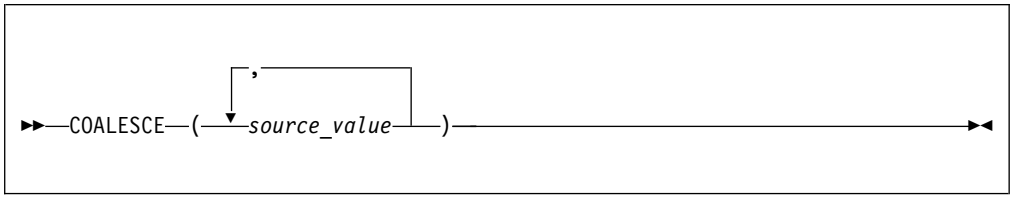
Related reference:

"Syntax diagrams" on page 3677

"Miscellaneous ESQL functions" on page 5290
ESQL provides additional functions that support miscellaneous operations.

COALESCE function:
COALESCE is a miscellaneous function that lets you provide default values for fields.

Syntax



The COALESCE function evaluates its parameters in order and returns the first one that is not NULL. The result is NULL if, and only if, all the arguments are NULL. The parameters can be of any scalar type, but they need not all be of the same type.

Use the COALESCE function to provide a default value for a field, which might not exist in a message. For example, the expression:

```
COALESCE(Body.Salary, 0)
```

returns the value of the Salary field in the message if it exists, or 0 (zero) if that field does not exist.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Transforming a complex message" on page 2520
When you code the ESQL for a Compute node, use the SELECT function for complex message transformation.

Related reference:

"Syntax diagrams" on page 3677

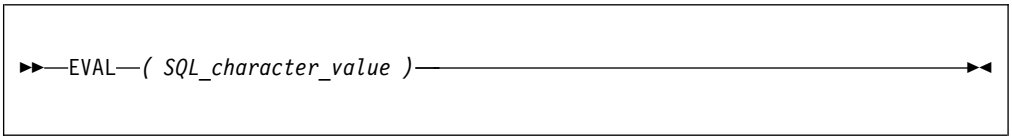
"Miscellaneous ESQL functions" on page 5290
ESQL provides additional functions that support miscellaneous operations.

EVAL function:

The EVAL function takes a character value and interprets that value as an ESQL expression that returns a value.

For details of the EVAL statement, see "EVAL statement" on page 5131.

Syntax



EVAL takes one parameter in the form of an expression, evaluates this expression, and casts the resulting value to a character string if it is not one already. The expression that is passed to EVAL must therefore be able to be represented as a character string.

User defined functions cannot be defined within an EVAL function but EVAL can be used to call a user-defined function that is in scope where the EVAL function is used.

If you use the EVAL function to call out to a user-defined function that is not called from anywhere else in the ESQL for a given node, you need to add the following code to your ESQL, to ensure that the user-defined function being called is included when the code is compiled:

```
IF (FALSE) THEN CALL function(<parameters>) INTO Environment.temp; END IF;
```

Note, that in the preceding example, you must replace `function()` with the name of the function in question.

In the following examples, A and B are integer scalar variables, and scalarVar1 and OperatorAsString are character string scalar variables.

The following examples are valid uses of EVAL:

- `SET OutputRoot.XMLNS.Data.Result = EVAL(A+B);`

  The expression A+B is acceptable because, although it returns an integer value, integer values are representable as character strings, and the necessary cast is performed before EVAL continues with its second stage of evaluation.
- `SET OutputRoot.XMLNS.Data.Result  = EVAL('A' ||  operatorAsString || 'B');`
- `EVAL('SET ' || scalarVar1 || ' = 2;');`

  The semicolon included at the end of the final string literal is necessary, because if EVAL is being used in place of an ESQL statement, its first stage evaluation must return a string that represents a valid ESQL statement, including the terminating semicolon.

The real power of EVAL is that it allows you to dynamically construct ESQL statements or expressions. In the second and third examples above, the value of scalarVar1 or operatorAsString can be set according to the value of an incoming message field, or other dynamic value, allowing you to effectively control what ESQL is executed without requiring a potentially lengthy IF-THEN ladder.

However, consider the performance implications in using EVAL. Dynamic construction and execution of statements or expressions is necessarily more time-consuming than simply executing pre-constructed ones. If performance is vital, you might prefer to write more specific, but faster, ESQL.

The following are not valid uses of EVAL:

- `SET EVAL(scalarVar1) = 2;`

  In this example, EVAL is being used to replace a field reference, not an expression.
- `SET OutputRoot.XMLNS.Data.Result[] = EVAL((SELECT T.x FROM Database.y AS T));`

  In this example, the `(SELECT T.x FROM Database.y)` passed to EVAL returns a list, which is not representable as a character string.

The following example is acceptable because `(SELECT T.x FROM Database.y AS T)` is a character string literal, not an expression in itself, and therefore is representable as a character string.

```
SET OutputRoot.XMLNS.Data.Result[]
 = EVAL('(SELECT T.x FROM Database.y AS T)');
```

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

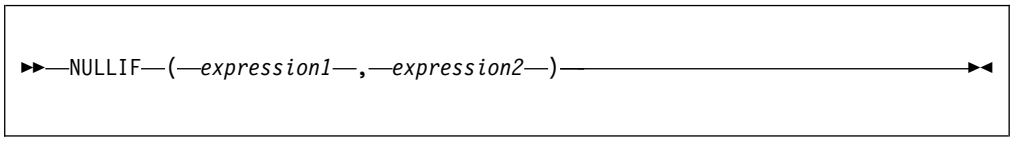
"Syntax diagrams" on page 3677

"Miscellaneous ESQL functions" on page 5290
ESQL provides additional functions that support miscellaneous operations.

"EVAL statement" on page 5131
The EVAL statement takes a character value, interprets it as an SQL statement, and processes that statement.

NULLIF function:
NULLIF is a miscellaneous function that returns a NULL value if the arguments are equal.

Syntax



The NULLIF function returns a NULL value if the arguments are equal; otherwise, it returns the value of the first argument. The arguments must be comparable. The result of using `NULLIF(e1,e2)` is the same as using the expression:

```
CASE WHEN e1=e2 THEN NULL ELSE e1 END
```

When `e1=e2` evaluates to unknown (because one or both of the arguments is NULL), NULLIF returns the value of the first argument.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message

flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

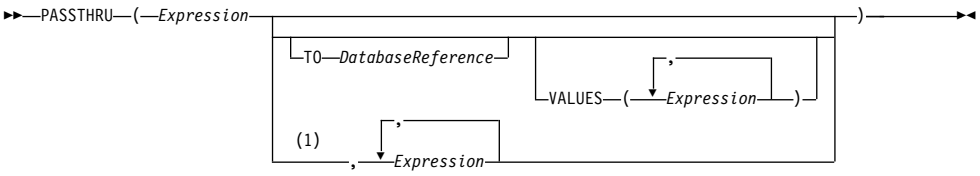
"Syntax diagrams" on page 3677

"Miscellaneous ESQL functions" on page 5290
ESQL provides additional functions that support miscellaneous operations.

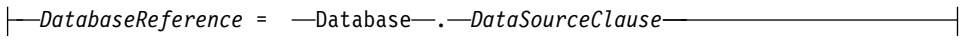
PASSTHRU function:

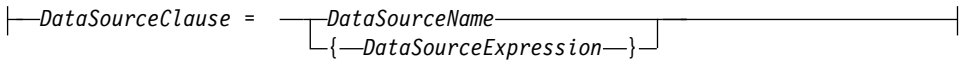
The PASSTHRU function evaluates an expression and executes the resulting character string as a database statement, returning a result set.

Syntax



WHERE:





Notes:

1 The lower half of the main syntax diagram describes syntax retained for backward compatability.

The PASSTHRU function is similar to the PASSTHRU statement, which is described in "PASSTHRU statement" on page 5147.

Usage

The main use of the PASSTHRU function is to issue complex SELECTs, not currently supported by the broker, to databases. (Examples of complex SELECTs not currently supported by the broker are those containing GROUP BY or HAVING clauses.)

The first expression is evaluated and the resulting character string is passed to the database pointed to by *DatabaseReference* (in the TO clause) for execution. If the TO clause is not specified, the database pointed to by the node's `data source` attribute is used.

Use question marks (?) in the database string to denote parameters. The parameter values are supplied by the VALUES clause.

If the VALUES clause is specified, its expressions are evaluated and passed to the database as parameters; (that is, the expressions' values are substituted for the question marks in the database statement).

If only one VALUE expression exists, the result might or might not be a list. If it is a list, the list's scalar values are substituted sequentially for the question marks. If it is not a list, the single scalar value is substituted for the (single) question mark in the database statement. If more than one VALUE expression exists, none of the expressions evaluate to a list; their scalar values are substituted sequentially for the question marks instead.

Because the database statement is constructed by the user program, it is not essential to use parameter markers (that is, the question marks) or the VALUES clause, because the whole of the database statement could be supplied, as a literal string, by the program. However, use parameter markers whenever possible because this reduces the number of different statements that need to be prepared and stored in the database and the broker.

Database reference

A database reference is a special instance of the field references that is used to refer to message trees. It consists of the word **Database** followed by the name of a data source (that is, the name of a database instance).

You can specify the data source name directly or by an expression enclosed in braces ({...}). A directly-specified data source name is subject to name substitution. That is, if the name used has been declared to be a known name, the value of the declared name is used rather than the name itself (see "DECLARE statement" on page 5117).

If you have created a message flow that contains one of the following nodes, and the ESQL that is associated with this node includes a PASSTHRU statement and a database reference, you must specify a value for the `Data source` property of the relevant node:
- Compute
- Database
- Filter

Handling errors

It is possible for errors to occur during PASSTHRU operations. For example, the database might not be operational or the statement might be invalid. In these cases, an exception is thrown (unless the node has its `Throw exception on database error` property cleared). These exceptions set appropriate SQL code, state, native error, and error text values and can be dealt with by error handlers (see the DECLARE HANDLER statement).

For further information about handling database errors, see "Capturing database state" on page 2512.

Example

The following example performs a SELECT on table `Table1` in schema `Schema1` in database DSN1, passing two parameters to the WHERE clause and asking for the result set to be ordered in ascending name order. The result set is assigned to the `SelectResult` folder:

```
SET OutputRoot.XML.Data.SelectResult.Row[] =
  PASSTHRU('SELECT R.* FROM Schema1.Table1 AS R WHERE R.Name = ? OR R.Name =
           ? ORDER BY Name'
   TO Database.DSN1
   VALUES ('Name1', 'Name4'));
```

The above example assigns the result set to the `OutputRoot` message body tree that is owned by the Generic XML parser, which allows self-defining messages.

If assigning the result set into a message tree owned by one of the MRM parsers, and the result set structure exactly matches the MRM message definition, the result set can be assigned directly into the `OutputRoot` message body tree.

If the result set structure does not exactly match the MRM message definition, you must first assign the result set into a ROW data type, or an Environment tree that does not have any parsers associated with it. The required data can then be assigned to `OutputRoot` to build a message tree that conforms to the message definition.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Selecting data from database columns" on page 2491
You can configure a Compute, Filter, or Database node to select data from database columns and include it in an output message.

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

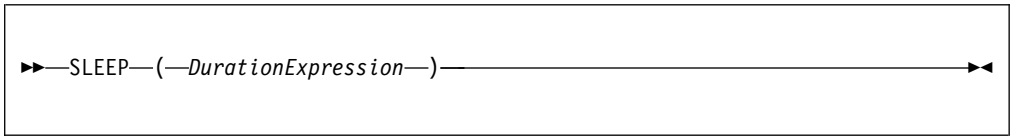
"PASSTHRU statement" on page 5147
The PASSTHRU statement evaluates an expression and runs the resulting character string as a database statement.

"Miscellaneous ESQL functions" on page 5290
ESQL provides additional functions that support miscellaneous operations.

SLEEP function:

The SLEEP function delays the execution of a message flow instance for a defined period of time, and returns a Boolean value to indicate whether the sleep completed without interruption.

Syntax



The SLEEP function returns TRUE if the sleep is completed for the specified duration without interruption, otherwise it returns FALSE.

The *DurationExpression* parameter specifies the number of milliseconds to sleep. It must be of INTEGER data type. If *DurationExpression* is NULL, the function returns NULL immediately without sleeping.

SLEEP cannot be called from inside an ATOMIC block because this would block other instances from having access to the shared resource. If such a call is attempted, FALSE is returned immediately.

The SLEEP function is automatically interrupted and returns FALSE, if a configuration or redeploy message for the flow arrives while the flow is sleeping.

Example

In the following example, SLEEP is called for 1000, if it is not interrupted for reconfiguration:

```
DECLARE returnValue BOOLEAN
SET returnValue = SLEEP(1000); /* attempt to sleep for one second */
```

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

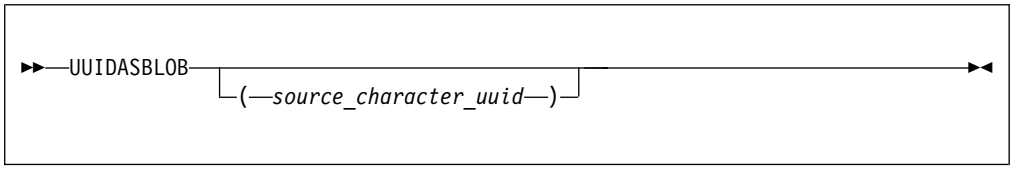
Related reference:

"Syntax diagrams" on page 3677

"Miscellaneous ESQL functions" on page 5290
ESQL provides additional functions that support miscellaneous operations.

UUIDASBLOB function:
UUIDASBLOB is a miscellaneous function that returns universally unique identifiers (UUIDs) as BLOBs.

Syntax



If (*source_character_uuid*) is not specified, UUIDASBLOB creates a new UUID and returns it as a BLOB.

If (*source_character_uuid*) is specified, UUIDASBLOB converts an existing character UUID in the form `ddddddddd_dddd_dddd_dddd_dddddddddddd` to the BLOB form. An exception is thrown if the parameter is not of the expected form.

The result is NULL if a NULL parameter is supplied.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

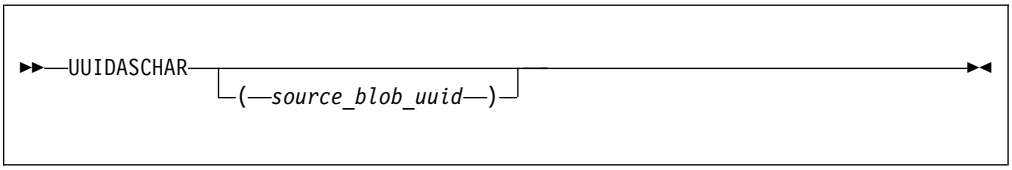
Related reference:

"Syntax diagrams" on page 3677

"Miscellaneous ESQL functions" on page 5290
ESQL provides additional functions that support miscellaneous operations.

UUIDASCHAR function:
UUIDASCHAR is a miscellaneous function that returns universally unique identifiers (UUIDs) as CHARACTER values.

Syntax



If (*source_blob_uuid*) is not specified, UUIDASCHAR creates a new UUID and returns it as a CHARACTER value.

If (*source_blob_uuid*) is specified, UUIDASCHAR converts an existing BLOB UUID to the character form.

The result is NULL if a NULL parameter is supplied.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"Syntax diagrams" on page 3677

"Miscellaneous ESQL functions" on page 5290
ESQL provides additional functions that support miscellaneous operations.

ESQL constants:

Use these constants to make or parse a bit stream.

ESQL constants in the Mapping node	Corresponding ESQL constants
$esql:RootBitStream	RootBitStream
$esql:FolderBitStream	FolderBitStream
$esql:ValidateContentAndValue	ValidateContentAndValue
$esql:ValidateValue	ValidateValue
$esql:ValidateContent	ValidateContent
$esql:ValidateNone	ValidateNone
$esql:ValidateException	ValidateException
$esql:ValidateExceptionList	ValidateExceptionList
$esql:ValidateLocalError	ValidateLocalError
$esql:ValidateUserTrace	ValidateUserTrace
$esql:ParseComplete	ParseComplete
$esql:ParseImmediate	ParseImmediate
$esql:ParseOnDemand	ParseOnDemand

Related tasks:
"Setting the value of a target element to an ESQL constant" on page 2260
There are two ways to set the value of a target element to an ESQL constant, depending on whether the target element has an entry in the Map Script column of the Message Mapping editor Spreadsheet pane.

Broker properties that are accessible from ESQL and Java:

You can access broker, message flow, and node properties from ESQL and Java.

The following table shows the properties that are available to ESQL and Java code by using the CMP interface.

The Java code, where applicable, is shown in the third column of the table.

Note that the `BrokerProxy`, `ExecutionGroupProxy`, `MessageFlowProxy`, and `LocalBrokerUtilities` classes are all part of the CMP interface (`ConfigManagerProxy.jar`). This JAR needs to be referenced from your Java project in your WebSphere Message Broker Toolkit, but it does not need to be deployed to the broker.

For a complete overview of broker properties, see "Broker properties" on page 1144.

Table 267. General broker properties.

Note: The only broker-defined properties that can be used in a Trace node are those in the "General broker properties" group. For example, you could specify the `Pattern` setting of a Trace node as:

```
#### Start Trace Input Message
  Time: ${CURRENT_TIMESTAMP}
  Broker: ${BrokerName}  Version: ${BrokerVersion}  Platform: ${Family}
  ProcessID: ${ProcessId}  BrokerUserId: ${BrokerUserId}
  ExecutionGroupLabel: ${ExecutionGroupLabel}
  Transaction: ${Transaction}
  Root Tree: ${Root}
#### End Trace Input Message
```

ESQL Property name	Java access method	Description
BrokerName (Character)	Accessible through: 1. `MbNode.getBroker()` 2. `MbBroker.getName()`	The name of the broker.
BrokerUserId (Character)	Use `System.getProperty("user.name");` to get the name of the user ID under which the broker was started.	The user ID under which the broker is running (that is, the user ID specified by the **-i** flag on the **mqsicreatebroker** command on Windows, or the user ID that started the broker by using the **mqsistart** command on Linux and UNIX systems).

BrokerVersion (Character)	Use
	
	`BrokerProxy b = BrokerProxy.getLocalInstance();`
	`int v = b.getBrokerVersion();`
		The 4-character version number of the broker (see "BrokerVersion" on page 5305).

ExecutionGroupLabel (Character)	Use
	
	`ExecutionGroupProxy eg = ExecutionGroupProxy.getLocalInstance();`
	`eg.getName();`
		The label of the execution group (a human-readable name).

ExecutionGroupName (Character)	Use
	
	`ExecutionGroupProxy eg = ExecutionGroupProxy.getLocalInstance();`
	`eg.getUUID();`
		The name of the execution group (typically a UUID identifier).

Family (Character)	Use `System.getProperty("os.name")` to return the operating system name from Java.	The generic name of the software platform that the broker is running on (`'WINDOWS'`, `'UNIX'`, or `'ZOS'`).

ProcessId (Integer)	Use
	
	`ExecutionGroupProxy eg = ExecutionGroupProxy.getLocalInstance();`
	`String processId =`
	`eg.getRuntimeProperty(AttributeConstants.EG_THIS_PROCESSID_PROPERTY);`
		The process identifier (PID) of the execution group.

QueueManagerName (Character)		The name of the WebSphere MQ queue manager to which the broker is connected.
WorkPath (Character)	To return any non-default workpath in Java, use: `String wp = LocalBrokerUtilities.getLocalBrokerWorkpath(brokerName);`	(Optional) The directory in which working files for this broker are stored.

Table 268. Flow properties

ESQL Property name	Java access method	Description

AdditionalInstances (Integer)	Use
	
	`ExecutionGroupProxy eg = ExecutionGroupProxy.getLocalInstance();`
	`MessageFlowProxy mf = eg.getMessageFlowByName("mf1");`
	`int i = mf.getAdditionalInstances();`
		The number of additional threads that the broker can use to service the message flow.

CommitCount (Integer)	Use
	
	`ExecutionGroupProxy eg = ExecutionGroupProxy.getLocalInstance();`
	`MessageFlowProxy mf = eg.getMessageFlowByName("mf1");`
	`int i = mf.getCommitCount();`
		The number of input messages that are processed by the message flow before a syncpoint is taken.

CommitInterval (Integer)	Use
	
	`ExecutionGroupProxy eg = ExecutionGroupProxy.getLocalInstance();`
	`MessageFlowProxy mf = eg.getMessageFlowByName("mf1");`
	`int i = mf.getCommitInterval();`
		The time interval at which a commit is taken when the `CommitCount` property is greater than 1 (that is, when the message flow is batching messages), but the number of messages processed has not reached the value of the `CommitCount` property.

CoordinatedTransaction (Boolean)	Not possible	Whether the message flow is processed as a global transaction, coordinated by WebSphere MQ.
MessageFlowLabel (Character)	Not possible	The name of the flow.

Table 269. Node properies

ESQL Property name	Java access method	Description
DataSource (Character)	Not possible	The ODBC Data Source Name (DSN) of the database in which the user tables are created.
DataSourceUserId (Character)	Not possible	The user ID that the broker uses to access the database user tables.
MessageOptions (Integer 64-bit)[1]	Not possible	The bit stream and validation options in force.
NodeLabel (Character)	Not applicable	The name of the node.
NodeType (Character)	Not applicable	The type of node (Compute, Database, or Filter).
ThrowExceptionOnDatabaseError (Boolean)[1]	Not possible	Whether the broker generates an exception when a database error is detected.
TransactionType (Character)[1]	Not possible	The type of transaction (Automatic or Commit) used to access a database from this node.
TreatWarningsAsErrors (Boolean)[1]	Not possible	Whether database warning messages are treated as errors, and cause the output message to be propagated to the failure terminal.

Notes:
1. Not applicable for the DatabaseInput node.

BrokerVersion

The BrokerVersion property contains a 4-character code that indicates the version of the broker. The code is based on the IBM Version/Release/Modification/Fix pack (VRMF) product-numbering system. The VRMF code works like this:

V The Version number. A Version is a separate IBM licensed program that usually has significant new code or new function. Each version has its own license, terms, and conditions.

R The Release number. A Release is a distribution of new function and authorized program analysis report (APAR) fixes for an existing product.

M The Modification number. A Modification is new function added to an existing product, and is delivered separately from an announced Version or Release.

F The Fix pack number. Fix packs contain defect and APAR fixes. They do not contain new function.

A fix pack is cumulative: that is, it contains all the fixes shipped in previous maintenance to the release, including previous fix packs. It can be applied on top of any previously-shipped maintenance to bring the system up to the current fix pack level.

Related concepts:

"Broker properties" on page 1144
For each broker, WebSphere Message Broker maintains a set of properties. You can access some of these properties from your ESQL programs. A subset of the properties is also accessible from Java programs. It can be useful, at run time, to have real-time access to details of a specific node, flow, or broker.

Related tasks:

"Accessing broker properties from ESQL" on page 2625
You can access broker properties, at run time, from the ESQL modules in your message flow nodes.

"Accessing broker properties from the JavaCompute node" on page 2658
For each broker, WebSphere Message Broker maintains a set of properties. You can access some of these properties from your Java programs. It can be useful, during the run time of your code, to have real-time access to details of a specific node, flow, or broker.

"Creating a user-defined extension in Java" on page 3054
You must complete a series of tasks to create user-defined nodes that use the Java language.

Related reference:

"ESQL-to-Java data-type mapping table" on page 5043
Table summarizing the mappings from ESQL to Java.

Related information:

Java user-defined extensions API

Special characters, case sensitivity, and comments in ESQL:

When you work with ESQL, it is useful to know about the special characters that are available, whether ESQL syntax is case sensitive, and how to handle comments.

- "Special characters" on page 5306
- "Case sensitivity of ESQL syntax" on page 5306
- "Comments" on page 5306

Special characters

Symbol	Name	Usage
;	semicolon	End of ESQL statement
.	period	Field reference separator or decimal point
=	equals	Comparison or assignment
>	greater than	Comparison
<	less than	Comparison
[]	square brackets	Array subscript
'	single quotation mark	Delimit string, date-time, and decimal literals Note, that to escape a single quotation mark inside a string literal, you must use two single quotation marks.
||	double vertical bar	Concatenation
()	parentheses	Expression delimiter
"	quotation mark	Identifier delimiter
*	asterisk	Any name or multiply
+	plus	Arithmetic add
-	minus	Arithmetic subtract, date separator, or negation
/	forward slash	Arithmetic divide
_	underscore	LIKE single wildcard
%	percent	LIKE multiple wildcard
\	backslash	LIKE escape character
:	colon	Name space and Time literal separator
,	comma	List separator
<>	less than greater than	Not equals
--	double minus	ESQL single line comment
/* */	slash asterisk asterisk slash	ESQL multiline comment
?	question mark	Substitution variable in PASSTHRU
<=	less than or equal	Comparison
>=	greater than or equal	Comparison
/*!{ }!*/	executable comment	Bypass tools check

Case sensitivity of ESQL syntax

The case of ESQL statements is:

- Case sensitive in field reference literals
- Not case sensitive in ESQL language words

Comments

ESQL has two types of comment: single line and multiple line. A single line comment starts with the characters -- and ends at the end of the line.

In arithmetic expressions you must take care not to initiate a line comment accidentally. For example, consider the expression:

```
1 - -2
```

Removing all white space from the expression results in:

```
1--2
```

which is interpreted as the number 1, followed by a line comment.

A multiple line comment starts with /* anywhere in ESQL and ends with */.

ESQL reserved keywords:

The keywords listed are reserved in uppercase, lowercase, or mixed case.

You cannot use these keywords for variable names. However, you *can* use reserved keywords as names in a field reference.

The following list shows all ESQL reserved keywords:

- ALL
- ASYMMETRIC
- BOTH
- CASE
- DISTINCT
- FROM
- ITEM
- LEADING
- NOT
- SYMMETRIC
- TRAILING
- WHEN

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"ESQL non-reserved keywords"
Some keywords are used in the ESQL language, but are not reserved. Do not use them for variable, function, or procedure names (in any combination of uppercase and lowercase) because your code can become difficult to understand.

ESQL non-reserved keywords:

Some keywords are used in the ESQL language, but are not reserved. Do not use them for variable, function, or procedure names (in any combination of uppercase and lowercase) because your code can become difficult to understand.

The following keywords are used in the ESQL language, but are not reserved.

- AND

- ANY
- AS
- ATOMIC
- ATTACH
- BEGIN
- BETWEEN
- BIT
- BLOB
- BOOLEAN
- BY
- CALL
- CATALOG
- CCSID
- CHAR
- CHARACTER
- COMPUTE
- CONDITION
- CONSTANT
- CONTINUE
- COORDINATED
- COUNT
- CREATE
- CURRENT_DATE
- CURRENT_GMTDATE
- CURRENT_GMTTIME
- CURRENT_GMTTIMESTAMP
- CURRENT_TIME
- CURRENT_TIMESTAMP
- DATA
- DATABASE
- DATE
- DAY
- DAYOFWEEK
- DAYOFYEAR
- DAYS
- DECIMAL
- DECLARE
- DEFAULT
- DELETE
- DETACH
- DO
- DOMAIN
- DYNAMIC
- ELSE
- ELSEIF
- ENCODING
- END
- ENVIRONMENT
- ESCAPE
- ESQL
- EVAL
- EVENT
- EXCEPTION
- EXISTS
- EXIT
- EXTERNAL

- FALSE
- FIELD
- FILTER
- FINALIZE
- FIRSTCHILD
- FLOAT
- FOR
- FORMAT
- FOUND
- FULL
- FUNCTION
- GMTTIME
- GMTTIMESTAMP
- GROUP
- HANDLER
- HAVING
- HOUR
- IDENTITY
- IF
- IN
- INF
- INFINITY
- INOUT
- INSERT
- INT
- INTEGER
- INTERVAL
- INTO
- IS
- ISLEAPYEAR
- ITERATE
- JAVA
- LABEL
- LANGUAGE
- LAST
- LASTCHILD
- LEAVE
- LIKE
- LIST
- LOCALTIMEZONE
- LOG
- LOOP
- MAX
- MESSAGE
- MIN
- MINUTE
- MODIFIES
- MODULE
- MONTH
- MONTHS
- MOVE
- NAME
- NAMESPACE
- NAN
- NEXTSIBLING
- NONE

- NULL
- NUM
- NUMBER
- OF
- OPTIONS
- OR
- ORDER
- OUT
- PARSE
- PASSTHRU
- PATH
- PLACING
- PREVIOUSSIBLING
- PROCEDURE
- PROPAGATE
- QUARTEROFYEAR
- QUARTERS
- READS
- REFERENCE
- REPEAT
- RESIGNAL
- RESULT
- RETURN
- RETURNS
- ROW
- SAMEFIELD
- SCHEMA
- SECOND
- SELECT
- SET
- SETS
- SEVERITY
- SHARED
- SHORT
- SOME
- SQL
- SQLCODE
- SQLERRORTEXT
- SQLEXCEPTION
- SQLNATIVEERROR
- SQLSTATE
- SQLWARNING
- SUM
- TERMINAL
- THE
- THEN
- THROW
- TIME
- TIMESTAMP
- TO
- TRACE
- TRUE
- TYPE
- UNCOORDINATED
- UNKNOWN
- UNTIL

- UPDATE
- USER
- UUIDASBLOB
- UUIDASCHAR
- VALUE
- VALUES
- WEEKOFMONTH
- WEEKOFYEAR
- WEEKS
- WHERE
- WHILE
- YEAR

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

Related reference:

"ESQL reserved keywords" on page 5307
The keywords listed are reserved in uppercase, lowercase, or mixed case.

Example message:
This topic defines the example message that is used in many of the examples throughout the information center.

The example message is:

```
<Invoice>
  <InvoiceNo>300524</InvoiceNo>
  <InvoiceDate>2000-12-07</InvoiceDate>
  <InvoiceTime>12:40:00</InvoiceTime>
  <TillNumber>3</TillNumber>
  <Cashier StaffNo='089'>Mary</Cashier>
  <Customer>
    <FirstName>Andrew</FirstName>
    <LastName>Smith</LastName>
    <Title>Mr</Title>
    <DOB>20-01-70</DOB>
    <PhoneHome>01962818000</PhoneHome>
    <PhoneWork />
    <Billing>
      <Address>14 High Street</Address>
      <Address>Hursley Village</Address>
      <Address>Hampshire</Address>
      <PostCode>SO213JR</PostCode>
    </Billing>
  </Customer>
  <Payment>
    <CardType>Visa</CardType>
    <CardNo>4921682832258418</CardNo>
    <CardName>Mr Andrew J. Smith</CardName>
    <Valid>1200</Valid>
    <Expires>1101</Expires>
  </Payment>
  <Purchases>
    <Item>
```

```
      <Title Category='Computer' Form='Paperback' Edition='2'>The XML Companion</Title>
      <ISBN>0201674866</ISBN>
      <Author>Neil Bradley</Author>
      <Publisher>Addison-Wesley</Publisher>
      <PublishDate>October 1999</PublishDate>
      <UnitPrice>27.95</UnitPrice>
      <Quantity>2</Quantity>
    </Item>
    <Item>
      <Title Category='Computer' Form='Paperback' Edition='2'>A Complete Guide to DB2 Universal Datal
      <ISBN>1558604820</ISBN>
      <Author>Don Chamberlin</Author>
      <Publisher>Morgan Kaufmann Publishers</Publisher>
      <PublishDate>April 1998</PublishDate>
      <UnitPrice>42.95</UnitPrice>
      <Quantity>1</Quantity>
    </Item>
    <Item>
      <Title Category='Computer' Form='Hardcover' Edition='0'>JAVA 2 Developers Handbook</Title>
      <ISBN>0782121799</ISBN>
      <Author>Philip Heller, Simon Roberts </Author>
      <Publisher>Sybex, Inc.</Publisher>
      <PublishDate>September 1998</PublishDate>
      <UnitPrice>59.99</UnitPrice>
      <Quantity>1</Quantity>
    </Item>
  </Purchases>
  <StoreRecords/>
  <DirectMail/>
  <Error/>
</Invoice>
```

For a diagrammatic representation of this message, and for examples of how this message can be manipulated with ESQL statements and functions, refer to "Writing ESQL" on page 2413.

Java reference

You can use Java language from within JavaCompute nodes and from user-defined nodes.

To see the Java classes and methods see "Java user-defined node API"

Related reference:

"JavaCompute node" on page 4514
Use the JavaCompute node to work with messages by using the Java language.

Java user-defined node API:

Use the API described here for both user-defined nodes, and for Java code that is called by JavaCompute nodes.

The Java classes and methods are described in the Java user-defined extensions API.

Related concepts:

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

"Developing user-defined extensions" on page 2970
A user-defined extension is a component that you design and implement to extend the function of WebSphere Message Broker.

Related reference:

"JavaCompute node" on page 4514
Use the JavaCompute node to work with messages by using the Java language.

PHP API

The WebSphere Message Broker PHP API provides classes to represent the message, the message assembly, and the element tree.

The following five classes have been defined:

- "MbsElement"
- "MbsElementIterator" on page 5316
- "MbsMessage" on page 5317
- "MbsMessageAssembly" on page 5319
- "MbsBlob" on page 5322

See "PHP data types" on page 5323 for information about the mappings between PHP and ESQL data types.

See "PHP extensions" on page 5324 for information about the PHP extensions supported by WebSphere Message Broker.

Related tasks:

"Using PHP" on page 2670
You can use the PHP scripting language for message routing and transformation.

"Creating PHP code for a PHPCompute node" on page 2672
Use these instructions to create your PHP code and associate it with your PHPCompute node.

Related reference:

"PHPCompute node" on page 4639
Use the PHPCompute node to route and transform an incoming message, using the PHP scripting language.

"PHP data types" on page 5323
PHP data types are supported by WebSphere Message Broker.

MbsElement:

The MbsElement class represents a single parsed element in a message (or other logical tree).

The MbsElement class provides the API methods shown in the following table (parameters in square brackets are optional):

Method	Description
Constructor()	Instantiates a new MbsElement object.
object **getValue**()	Returns the value of the current element.
void **setValue**(*object $value*)	Sets the value of the current element.
int **getType**()	Returns the specific type of the current element. For a full list of type values, see "PHP constants for type values" on page 5343.
void **setType**(*int $type*)	Sets the specific type of the current element. For a full list of type values, see "PHP constants for type values" on page 5343.

Method	Description
object **xpath**(*string $expression [, array $namespace]*)	Evaluates the XPath expression with the current element as the context node. It returns the result as either a string, double, boolean, or a nodeset as an array of MbsElement objects. The optional **namespace** parameter is an associative array with namespace prefixes as the keys and namespace URIs as the values.
string **getName**()	Returns the name of the current element.
void **setName**(*string $name*)	Sets the name of the current element.
string **getNamespace**()	Returns the namespace URI of the current element.
void **setNamespace**(*$namespace*)	Sets the namespace URI of the current element.
MbsElement **getParent**()	Returns the parent of the current element.
MbsElement **getChild**(*string $name [, int $occurrence]*)	Returns the first child of the current element whose name is given by the first parameter. The n^{th} occurrence of that child can be returned by specifying the second optional parameter.
array **getChildren**(*[string $namespace]*)	Returns all the child elements n of the current element as an array of MbsElements. If the **namespace** parameter is specified, the array contains only the child elements with that namespace URI.
MbsElement **getFirstChild**()	Returns the first child of the current element.
MbsElement **getLastChild**()	Returns the last child of the current element.
MbsElement **getNextSibling**()	Returns the next sibling of the current element.
MbsElement **getPreviousSibling**()	Returns the previous sibling of the current element.
MbsElement **getAttribute**(*string $name*)	Returns the attribute of the current element given by the **name** parameter.
MbsElement **addAttribute**(*string $name, object $value [, string $namespace]*)	Adds an attribute to the current element.
MbsElement **addElement**(*string $name, object $value [, string $namespace [, int $type [, int $position]]]*)	Creates an element as last child (by default) of the current element. The optional **type** parameter is the parser-specific type of the new node (which defaults to XML element type for XML parsers). The optional **position** parameter can be one of the following values: • *MB_FIRST_CHILD* • *MB_LAST_CHILD* • *MB_NEXT_SIBLING* • *MB_PREVIOUS_SIBLING*
MbsElement **detach**()	Detaches the current element from the tree.
void **detachAllChildren**()	Detaches all child elements of the current element.

Method	Description
string **asBitstream**(*[array $options]*)	Serializes the element tree to produce a bit stream. When using the MRM parser (and other parsers), the *options* array must be populated with the following key/value pairs: `array ('set' => '<MessageSet>', 'type' => '<MessageType>', 'format' => '<MessageFormat>', 'encoding' => '<encoding>', 'ccsid' => '<ccsid>')`
void **addElementFromBitstream**(*string $bitstream [, array $options]*)	Creates an element tree from the supplied bit stream. Without *options* supplied, the bit stream is parsed by using the parser of the current element and attached as its last child (no domain element is created in this case). If *options* is supplied, it must contain the key/value pair `'domain' => '<parserDomain>'` plus any of the additional options for the MRM parser (see **asBitstream**()). In this case, a parser element is created at the head of the new subtree and attached as the last child of the current element.

You can create elements in PHP to build subtrees. The following example shows a PHP function that generates and returns part of an output tree:

```
function transformItem($input) {
      $item = new MbsElement;
      $item->Desc = $input->Description;
      $item->Cost = $input->Price->getValue() * 1.6;
      $item->Qty = $input->Quantity;

      return $item;
   }
```

You can copy this subtree into an output message by using the following code:

```
$output_assembly->XMLNSC->doc->item = transformItem($input_sub_tree);
```

where *$input_sub_tree* is a reference to a part of the input message.

Related concepts:

"Accessing elements in the message tree from a PHPCompute node" on page 2692
Access the contents of a message, for reading or writing, by using the structure and arrangement of the elements in the tree that the parser creates from the input bit stream.

Related tasks:

"Using PHP" on page 2670
You can use the PHP scripting language for message routing and transformation.

"Creating PHP code for a PHPCompute node" on page 2672
Use these instructions to create your PHP code and associate it with your PHPCompute node.

Related reference:

"PHP API" on page 5313
The WebSphere Message Broker PHP API provides classes to represent the message, the message assembly, and the element tree.

"MbsElementIterator"
The MbsElementIterator implements the SPL RecursiveIterator.

"MbsMessage" on page 5317
The MbsMessage class represents one of the logical trees that make up the message assembly.

"MbsMessageAssembly" on page 5319
The MbsMessageAssembly class represents the message assembly that is propagated between nodes in a message flow.

"MbsBlob" on page 5322
The MbsBlob class supports the ESQL BLOB type.

"PHPCompute node" on page 4639
Use the PHPCompute node to route and transform an incoming message, using the PHP scripting language.

"PHP constants for type values" on page 5343
You can use type values to create syntax elements in your message tree.

MbsElementIterator:

The MbsElementIterator implements the SPL RecursiveIterator.

The MbsElementIterator class provides the API methods shown in the following table:

Method	Description
mixed current()	Returns the current element as an MbsElementIterator object, or NULL on failure.
object getChildren()	Returns an MbsElementIterator object containing child elements of the current MbsElementIterator element.
bool hasChildren()	Returns TRUE if the current MbsElementIterator element has child elements; otherwise returns FALSE.
mixed key()	Returns the name of the current element, or FALSE on failure.
void next()	Moves the MbsElementIterator to the next element.
void rewind()	Rewinds the MbsElementIterator to the first element.
bool valid()	Checks whether the current element is valid after calls to rewind() or next(). It returns TRUE if the current element is valid; otherwise returns FALSE.

Related concepts:

"Using SPL iterators with PHP" on page 2695
The PHPCompute node provides support for the Standard PHP Library (SPL) syntax, which provides iterators that can be used in PHP code.

"Iterating over elements" on page 2696
Use the MbsElementIterator class to provide sequential iteration over elements in the message tree.

"Recursive iterators" on page 2697
You can use a RecursiveIteratorIterator to iterate over a whole message tree, by

using it to wrap around an MbsElementIterator.

"Iterating with a filter" on page 2698
Use a FilterIterator to filter elements in the message tree.

Related tasks:

"Using PHP" on page 2670
You can use the PHP scripting language for message routing and transformation.

"Creating PHP code for a PHPCompute node" on page 2672
Use these instructions to create your PHP code and associate it with your PHPCompute node.

Related reference:

"PHP API" on page 5313
The WebSphere Message Broker PHP API provides classes to represent the message, the message assembly, and the element tree.

"MbsBlob" on page 5322
The MbsBlob class supports the ESQL BLOB type.

"MbsElement" on page 5313
The MbsElement class represents a single parsed element in a message (or other logical tree).

"MbsMessage"
The MbsMessage class represents one of the logical trees that make up the message assembly.

"MbsMessageAssembly" on page 5319
The MbsMessageAssembly class represents the message assembly that is propagated between nodes in a message flow.

"PHPCompute node" on page 4639
Use the PHPCompute node to route and transform an incoming message, using the PHP scripting language.

MbsMessage:

The MbsMessage class represents one of the logical trees that make up the message assembly.

The MbsMessage class provides the API methods shown in the following table. The parameters in square brackets are optional:

Method	Description
Constructor([*MbsMessage $msg*])	Instantiates a new message object, which is a copy of the **msg** parameter (optional). If no parameter is supplied, a new empty message is created.
int getType()	Returns the specific type of the root element.
array xpath(*string $expression [, array $namespace]*)	Evaluates the XPath expression by using the last child of root (body element) as the context node. It returns the result as either a string, double, boolean, or a nodeset as an array of MbsElement objects. The optional **namespace** parameter is an associative array with namespace prefixes as the keys and namespace URIs as the values.

Method	Description
MbsElement getChild(*string $name [, int $occurrence]*)	Returns the first child of the root element whose name is given by the first parameter. The nth occurrence of that child can be returned by specifying the second optional parameter.
array getChildren(*[string $namespace]*)	Returns all the child elements of the root element as an array of MbsElements. If the **namespace** parameter is specified, the array contains only the child elements with that namespace URI.
MbsElement getFirstChild()	Returns the first child of the root element.
MbsElement getLastChild()	Returns the last child of the root element.
MbsElement addElement(*string $name, object $value [, string $namespace [, int $type [, int $position]]]*)	Creates an element as last child (by default) of the root element. The optional **type** parameter is the parser-specific type of the new node, which defaults to XML element type for XML parsers. The optional **position** parameter can be one of the following values: • *MB_FIRST_CHILD* • *MB_LAST_CHILD* • *MB_NEXT_SIBLING* • *MB_PREVIOUS_SIBLING*
MbsElement addDomainElement(*string $domainName*)	Creates a domain element.
void detachAllChildren()	Detaches all child elements of the root element.
string asBitstream(*[array $options]*)	Serializes the element tree to produce a bit stream. When using the MRM parser (and other parsers), the *options* array must be populated with the following key/value pairs: `array ('set' => '<MessageSet>', 'type' => '<MessageType>', 'format' => '<MessageFormat>', 'encoding' => '<encoding>', 'ccsid' => '<ccsid>')`
void addElementFromBitstream(*string $bitstream [, array $options]*)	Creates an element tree from the supplied bit stream. Without *options* supplied, the bit stream is parsed by the parser of the root element and attached as its last child (no domain element is created in this case). If *options* is supplied, it must contain the key/value pair `'domain' => '<parserDomain>'` plus any of the additional options for the MRM parser (see **asBitstream**()). In this case, a parser element is created at the head of the new subtree and attached as the last child of the root element.

Related concepts:

"Accessing elements in the message tree from a PHPCompute node" on page 2692
Access the contents of a message, for reading or writing, by using the structure and arrangement of the elements in the tree that the parser creates from the input

bit stream.

Related tasks:

"Using PHP" on page 2670
You can use the PHP scripting language for message routing and transformation.

"Creating PHP code for a PHPCompute node" on page 2672
Use these instructions to create your PHP code and associate it with your PHPCompute node.

Related reference:

"PHP API" on page 5313
The WebSphere Message Broker PHP API provides classes to represent the message, the message assembly, and the element tree.

"MbsElement" on page 5313
The MbsElement class represents a single parsed element in a message (or other logical tree).

"MbsElementIterator" on page 5316
The MbsElementIterator implements the SPL RecursiveIterator.

"MbsMessageAssembly"
The MbsMessageAssembly class represents the message assembly that is propagated between nodes in a message flow.

"MbsBlob" on page 5322
The MbsBlob class supports the ESQL BLOB type.

"PHPCompute node" on page 4639
Use the PHPCompute node to route and transform an incoming message, using the PHP scripting language.

MbsMessageAssembly:

The MbsMessageAssembly class represents the message assembly that is propagated between nodes in a message flow.

The message assembly comprises four individual trees:

- Message
- LocalEnvironment
- GlobalEnvironment
- ExceptionList

If *assembly* is an instance of MbsMessageAssembly, you can access the four component trees by using the following code:

```
MbsMessage message = $assembly[MB_MESSAGE];
MbsMessage local_env = $assembly[MB_LOCAL_ENVIRONMENT];
MbsMessage global_env = $assembly[MB_GLOBAL_ENVIRONMENT];
MbsMessage ex_list = $assembly[MB_EXCEPTION_LIST];
```

The MbsMessageAssembly class provides the API methods shown in the following table. Parameters in square brackets are optional:

Method	Description
Constructor(*MbsMessageAssembly $assembly, MbsMessage $msg [, MbsMessage $local_env [, MbsMessage $exception_list]]*)	Instantiates a new message assembly object based on the specified template assembly. The message for the assembly is replaced by the message passed in on the **msg** parameter. The other logical trees can also be replaced if their associated parameter is provided. If the associated parameter is not provided, the tree from the template assembly is used.
int getType()	Returns the specific type of the root element of the message tree.
array xpath(*string $expression [, array $namespace]*)	Evaluates the XPath expression with the last child of root (body element) as the context node. It returns the result as either a string, double, boolean, or a nodeset as an array of MbsElement objects. The optional **namespace** parameter is an associative array with namespace prefixes as the keys and namespace URIs as the values.
MbsElement getChild(*string $name [, int $occurrence]*)	Returns the first child of the root element of the message tree whose name is given by the first parameter. The n^{th} occurrence of that child can be returned by specifying the second optional parameter.
array getChildren(*[string $namespace]*)	Returns all the child elements of the root element of the message tree as an array of MbsElements. If the **namespace** parameter is specified, the array contains only the child elements with that namespace URI.
MbsElement getFirstChild()	Returns the first child of the root element of the message tree.
MbsElement getLastChild()	Returns the last child of the root element of the message tree.
MbsElement addElement(*string $name, object $value [, string $namespace [, int $type [, int $position]]]*)	Creates an element as last child (by default) of the root element of the message tree. The optional **type** parameter is the parser-specific type of the new node (defaults to XML element type for XML parsers). The optional **position** parameter can be one of the following values: • *MB_FIRST_CHILD* • *MB_LAST_CHILD* • *MB_NEXT_SIBLING* • *MB_PREVIOUS_SIBLING*
MbsElement addDomainElement(*string $domainName*)	Creates a domain element.
void detachAllChildren()	Detaches all child elements of the root element of the message tree.

Method	Description
string asBitstream(*[array $options]*)	Serializes the element tree to produce a bit stream. When using the MRM parser (and other parsers), the *options* array must be populated with the following key/value pairs: `array ('set' => '<MessageSet>', 'type' => '<MessageType>', 'format' => '<MessageFormat>', 'encoding' => '<encoding>', 'ccsid' => '<ccsid>')`
void addElementFromBitstream(*string $bitstream [, array $options]*)	Creates an element tree from the supplied bit stream. Without *options* supplied, the bit stream is parsed by the parser of the root element of the message tree and attached as its last child (no domain element is created in this case). If *options* is supplied, it must contain the key/value pair `'domain' => '<parserDomain>'` plus any of the additional options for the MRM parser (see **asBitstream**()). In this case, a parser element is created at the head of the new subtree and attached as the last child of the root element of the message tree.
void propagate(*string $terminalName*)	Propagates the message assembly to the output terminal given by the parameter **terminalName**.
void routeToLabel(*string $labelName*)	Routes the message assembly to the Label node given by the parameter **labelName**.

Related concepts:

"Accessing elements in the message tree from a PHPCompute node" on page 2692
Access the contents of a message, for reading or writing, by using the structure and arrangement of the elements in the tree that the parser creates from the input bit stream.

Related tasks:

"Using PHP" on page 2670
You can use the PHP scripting language for message routing and transformation.

"Creating PHP code for a PHPCompute node" on page 2672
Use these instructions to create your PHP code and associate it with your PHPCompute node.

Related reference:

"PHPCompute node" on page 4639
Use the PHPCompute node to route and transform an incoming message, using the PHP scripting language.

"PHP API" on page 5313
The WebSphere Message Broker PHP API provides classes to represent the message, the message assembly, and the element tree.

"MbsElement" on page 5313
The MbsElement class represents a single parsed element in a message (or other logical tree).

"MbsElementIterator" on page 5316
The MbsElementIterator implements the SPL RecursiveIterator.

"MbsMessage" on page 5317
The MbsMessage class represents one of the logical trees that make up the message assembly.

"MbsBlob"
The MbsBlob class supports the ESQL BLOB type.

MbsBlob:

The MbsBlob class supports the ESQL BLOB type.

The MbsBlob class provides the API methods shown in the following table:

Method	Description
Constructor(*[string $blob]*)	Instantiates a new BLOB object from the string parameter (or an empty BLOB if no parameter is supplied).
string getValue()	Returns the value of the current element.
void setValue(*[string $value]*)	Sets the value of the current element.

Related concepts:

"Accessing elements in the message tree from a PHPCompute node" on page 2692
Access the contents of a message, for reading or writing, by using the structure and arrangement of the elements in the tree that the parser creates from the input bit stream.

Related tasks:

"Using PHP" on page 2670
You can use the PHP scripting language for message routing and transformation.

"Creating PHP code for a PHPCompute node" on page 2672
Use these instructions to create your PHP code and associate it with your PHPCompute node.

Related reference:

"PHP API" on page 5313
The WebSphere Message Broker PHP API provides classes to represent the message, the message assembly, and the element tree.

"MbsElement" on page 5313
The MbsElement class represents a single parsed element in a message (or other logical tree).

"MbsElementIterator" on page 5316
The MbsElementIterator implements the SPL RecursiveIterator.

"MbsMessage" on page 5317
The MbsMessage class represents one of the logical trees that make up the message assembly.

"MbsMessageAssembly" on page 5319
The MbsMessageAssembly class represents the message assembly that is propagated between nodes in a message flow.

"PHPCompute node" on page 4639
Use the PHPCompute node to route and transform an incoming message, using the PHP scripting language.

PHP data types:

PHP data types are supported by WebSphere Message Broker.

The following table shows the mappings between the ESQL and PHP data types:

ESQL type	PHP type
INTEGER, INT	• int - for integers in the range -2147483648 - 2147483647 • float - for integers outside the range -2147483648 - 2147483647
FLOAT	float
DECIMAL	Java BigDecimal
CHARACTER, CHAR	string
BLOB	MbsBlob
DATE	Java com.ibm.broker.plugin.MbDate
TIME, GMTIME	Java com.ibm.broker.plugin.MbTime
TIMESTAMP, GMTIMESTAMP	Java com.ibm.broker.plugin.MbTimestamp
INTERVAL	Not supported
BOOLEAN	Boolean
REFERENCE	MbsElement

Some of the PHP types are mapped onto Java data types; for example, BigDecimal and MbTimestamp. You can manipulate these values by using the Java bridge. For more information, see “Calling Java from PHP” on page 2716.

No 64-bit integer type exists in PHP, therefore large values are represented as a float. You can use the Java bridge to manipulate Java types. The INTERVAL ESQL type is not represented in the Java API for the broker.

PHP processes the values stored in a string data type as single-byte strings. However, PHP has a multibyte string extension that enables the manipulation of multibyte strings in a PHP string variable. This PHP extension is included with the PHPCompute node.

For more information about PHP extensions, see the PHP: Hypertext Preprocessor website.

The following multibyte functions are provided with the PHPCompute node:

Function	Description
mb_convert_encoding	Converts the character encoding of a string.
mb_decode_mimeheader	Decodes the encoded string in a MIME header.
mb_encode_mimeheader	Encodes a string with MIME header encoding.
mb_ereg	Runs the regular expression match with multibyte support.
mb_ereg_replace	Replaces the regular expression with multibyte support.
mb_internal_encoding	Sets or gets the internal character encoding.
mb_regex_encoding	Returns the current encoding for a multibyte regex as a string.

Function	Description
mb_regex_set_options	Sets the default options (specified by the **options** parameter) for multibyte regex functions.
mb_split	Splits a multibyte string and returns the result as an array.
mb_stripos	Finds the position of the first occurrence of a string within another. This function is not case sensitive.
mb_stristr	Finds the first occurrence of a string within another. This function is not case sensitive.
mb_strlen	Gets the length of a string.
mb_strpos	Finds the position of the first occurrence of a string in a string.
mb_strrchr	Finds the last occurrence of a character in a string within another string.
mb_strrichr	Finds the last occurrence of a character in a string within another string. This function is not case sensitive.
mb_strripos	Finds the position of the last occurrence of a string within another string. This function is not case sensitive.
mb_strrpos	Finds the position of the last occurrence of a string within a string.
mb_strstr	Finds the first occurrence of a string within another.
mb_strtolower	Makes a string lowercase.
mb_strtoupper	Makes a string uppercase.
mb_substitute_character	Sets or gets a substitution character.
mb_substr	Gets part of a string.
mb_substr_count	Counts the number of substring occurrences.

Related concepts:

"Calling Java from PHP" on page 2716
The IBM sMash Runtime for PHP provides access to Java classes and functionality from PHP. This Java Bridge can instantiate Java classes and call their methods.

Related tasks:

"Using PHP" on page 2670
You can use the PHP scripting language for message routing and transformation.

"Creating PHP code for a PHPCompute node" on page 2672
Use these instructions to create your PHP code and associate it with your PHPCompute node.

Related reference:

"PHP API" on page 5313
The WebSphere Message Broker PHP API provides classes to represent the message, the message assembly, and the element tree.

"PHPCompute node" on page 4639
Use the PHPCompute node to route and transform an incoming message, using the PHP scripting language.

PHP extensions:

WebSphere Message Broker supports a set of PHP extensions.

The following PHP functions (listed by extension) are provided by WebSphere Message Broker. However, some differences exist between PHP.net and the

implementation of PHP that is provided by WebSphere Message Broker. For more information about these differences, see “Differences between WebSphere Message Broker PHP and PHP.net” on page 5339.

Array

- array
- array_change_key_case
- array_chunk
- array_combine
- array_count_values
- array_diff
- array_diff_assoc
- array_diff_key
- array_diff_uassoc
- array_diff_ukey
- array_fill
- array_fill_keys
- array_filter
- array_flip
- array_intersect
- array_intersect_assoc
- array_intersect_key
- array_intersect_uassoc
- array_intersect_ukey
- array_key_exists
- array_keys
- array_map
- array_merge
- array_merge_recursive
- array_multisort
- array_pad
- array_pop
- array_product
- array_push
- array_rand
- array_reduce
- array_reverse
- array_search
- array_shift
- array_slice
- array_splice
- array_sum
- array_udiff
- array_udiff_assoc
- array_udiff_uassoc
- array_uintersect
- array_uintersect_assoc
- array_uintersect_uassoc
- array_unique
- array_unshift
- array_values
- array_walk
- array_walk_recursive
- arsort
- asort
- compact

- count
- current
- each
- end
- extract
- in_array
- key
- key_exists
- krsort
- ksort
- natcasesort
- natsort
- next
- pos
- prev
- range
- reset
- rsort
- shuffle
- sizeof
- sort
- uasort
- uksort
- usort

BC Math

- bcadd
- bccomp
- bcdiv
- bcmod
- bcmul
- bcpow
- bcpowmod
- bcscale
- bcsqrt
- bcsub

Class-object

- call_user_method
- call_user_method_array
- class_exists
- get_class
- get_class_methods
- get_class_vars
- get_declared_classes
- get_declared_interfaces
- get_object_vars
- get_parent_class
- interface_exists
- is_a
- is_subclass_of
- method_exists
- property_exists

Date and time

- checkdate
- date

- date_create
- date_date_set
- date_default_timezone_get
- date_default_timezone_set
- date_format
- date_isodate_set
- date_modify
- date_offset_get
- date_parse
- date_sun_info
- date_sunrise
- date_sunset
- date_time_set
- date_timezone_get
- date_timezone_set
- DateTime_construct
- DateTime_format
- DateTime_getOffset
- DateTime_getTimeZone
- DateTime_modify
- DateTime_setDate
- DateTime_setISODate
- DateTime_setTime
- DateTime_setTimeZone
- DateTimeZone_construct
- DateTimeZone_getName
- DateTimeZone_getOffset
- DateTimeZone_getTransitions
- DateTimeZone_listAbbreviations
- DateTimeZone_listIdentifiers
- getdate
- gettimeofday
- gmdate
- gmmktime
- gmstrftime
- idate
- localtime
- microtime
- mktime
- strftime
- strptime
- strtotime
- time
- timezone_abbreviations_list
- timezone_identifiers_list
- timezone_name_from_abbr
- timezone_name_get
- timezone_offset_get
- timezone_open
- timezone_transitions_get

Directory

- chdir
- closedir
- dir
- Directory.close

- Directory.read
- getcwd
- opendir
- readdir
- rewinddir
- scandir

Error handling and logging
- debug_backtrace
- debug_print_backtrace
- error_get_last
- error_log
- error_reporting
- restore_error_handler
- restore_exception_handler
- set_error_handler
- set_exception_handler
- trigger_error
- user_error

File system
- basename
- chgrp
- chmod
- chown
- clearstatcache
- copy
- dirname
- disk_free_space
- diskfreespace
- fclose
- feof
- fflush
- fgetc
- fgetcsv
- fgets
- file
- file_exists
- file_get_contents
- file_put_contents
- fileatime
- filectime
- filegroup
- fileinode
- filemtime
- fileowner
- fileperms
- filesize
- filetype
- flock
- fopen
- fpassthru
- fputcsv
- fputs
- fread
- fseek
- fstat

- ftell
- ftruncate
- fwrite
- glob
- is_dir
- is_executable
- is_file
- is_link
- is_readable
- is_uploaded_file
- is_writable
- is_writeable
- link
- linkinfo
- lstat
- mkdir
- move_uploaded_file
- parse_ini_file
- pathinfo
- pclose
- popen
- readfile
- readlink
- realpath
- rename
- rewind
- rmdir
- stat
- symlink
- tempnam
- touch
- umask
- unlink

Function handling

- call_user_func
- call_user_func_array
- create_function
- func_get_arg
- func_get_args
- func_num_args
- function_exists
- get_defined_functions
- register_shutdown_function
- register_tick_function
- unregister_tick_function

Java

- java_import

Mail

- ezmlm_hash
- mail

Math

- abs
- acos
- acosh

- asin
- asinh
- atan
- atan2
- atanh
- base_convert
- bindec
- ceil
- cos
- cosh
- decbin
- dechex
- decoct
- deg2rad
- exp
- expm1
- floor
- fmod
- getrandmax
- hexdec
- hypot
- is_finite
- is_infinite
- is_nan
- lcg_value
- log
- log10
- log1p
- max
- min
- mt_getrandmax
- mt_rand
- mt_srand
- octdec
- pi
- pow
- rad2deg
- rand
- round
- sin
- sinh
- sqrt
- srand
- tan
- tanh

Message Broker

- mb_get_user_defined_property

Miscellaneous

- constant
- define
- defined
- eval
- highlight_string
- ignore_user_abort
- pack

- sleep
- uniqid
- unpack
- usleep

Multibyte string

- mb_convert_encoding
- mb_decode_mimeheader
- mb_detect_encoding
- mb_detect_order
- mb_encode_mimeheader
- mb_ereg
- mb_eregi
- mb_eregi_replace
- mb_ereg_replace
- mb_internal_encoding
- mb_language
- mb_regex_encoding
- mb_regex_set_options
- mb_split
- mb_stripos
- mb_stristr
- mb_strlen
- mb_strpos
- mb_strrchr
- mb_strrichr
- mb_strripos
- mb_strrpos
- mb_strstr
- mb_strtolower
- mb_strtoupper
- mb_substitute_character
- mb_substr
- mb_substr_count

MySQL

- mysql_affected_rows
- mysql_client_encoding
- mysql_close
- mysql_connect
- mysql_create_db
- mysql_data_seek
- mysql_db_query
- mysql_drop_db
- mysql_errno
- mysql_error
- mysql_escape_string
- mysql_fetch_array
- mysql_fetch_assoc
- mysql_fetch_field
- mysql_fetch_lengths
- mysql_fetch_object
- mysql_fetch_row
- mysql_field_flags
- mysql_field_len
- mysql_field_name
- mqsql_field_seek

- mysql_field_table
- mysql_field_type
- mysql_free_result
- mysql_get_client_info
- mysql_get_host_info
- mqsql_get_proto_info
- mysql_get_server_info
- mqsql_info
- mysql_insert_id
- mysql_list_dbs
- mysql_list_fields
- mysql_list_processes
- mysql_list_tables
- mysql_num_fields
- mysql_num_rows
- mysql_pconnect
- mysql_ping
- mysql_query
- mysql_real_escape_string
- mysql_result
- mysql_select_db
- mysql_set_charset
- mysql_stat
- mysql_tablename
- mysql_thread_id
- mysql_unbuffered_query

Network

- checkdnsrr
- closelog
- define_syslog_variables
- fsockopen
- gethostbyaddr
- gethostbyname
- gethostbynamel
- getmxrr
- inet_ntop
- inet_pton
- ip2long
- long2ip
- pfsockopen
- socket_get_status
- socket_set_blocking
- socket_set_timeout
- syslog

Output control

- flush
- ob_clean
- ob_end_clean
- ob_end_flush
- ob_flush
- ob_get_clean
- ob_get_contents
- ob_get_flush
- ob_get_length
- ob_get_level

- ob_get_status
- ob_implicit_flush
- ob_list_handlers
- ob_start

PHP options and information

- assert
- assert_options
- extension_loaded
- get_cfg_var
- get_current_user
- get_defined_constants
- get_extension_funcs
- get_include_path
- get_included_files
- get_loaded_extensions
- get_magic_quotes_gpc
- get_magic_quotes_runtime
- getenv
- getmypid
- getrusage
- ini_alter
- ini_get
- ini_get_all
- ini_restore
- ini_set
- magic_quotes_runtime
- memory_get_usage
- php_sapi_name
- php_uname
- phpinfo
- phpversion
- putenv
- restore_include_path
- set_include_path
- set_magic_quotes_runtime
- set_time_limit
- sys_get_temp_dir
- version_compare
- zend_version

POSIX Regex

- ereg
- ereg_replace
- eregi
- eregi_replace
- split
- spliti
- sql_regcase

Program execution

- escapeshellarg
- escapeshellcmd
- exec
- passthru
- proc_close
- proc_open

- proc_terminate
- shell_exec
- system

PCRE

- preg_grep
- preg_last_error
- preg_match
- preg_match_all
- preg_quote
- preg_replace
- preg_replace_callback
- preg_split

Session

- session_cache_expire
- session_cache_limiter
- session_commit
- session_decode
- session_destroy
- session_encode
- session_get_cookie_params
- session_id
- session_is_registered
- session_module_name
- session_name
- session_regenerate_id
- session_register
- session_save_path
- session_set_cookie_params
- session_set_save_handler
- session_start
- session_unregister
- session_unset
- session_write_close

SNMP

- snmp2_get
- snmp2_getnext
- snmp2_real_walk
- snmp2_set
- snmp2_walk
- snmp3_get
- snmp3_getnext
- snmp3_real_walk
- snmp3_set
- snmp3_walk
- snmp_get_quick_print
- snmp_get_valueretrieval
- snmp_read_mib
- snmp_set_enum_print
- snmp_set_oid_numeric_print
- snmp_set_oid_output_format
- snmp_set_quick_print
- snmp_set_valueretrieval
- snmpget
- snmpgetnext

- snmprealwalk
- snmpset
- snmpwalk
- snmpwalkoid

SPL

Data structures
- SplObjectStorage

Iterators
- AppendIterator
- ArrayIterator
- CachingIterator
- DirectoryIterator
- EmptyIterator
- FilterIterator
- InfiniteIterator
- IteratorIterator
- LimitIterator
- NoRewindIterator
- RecursiveDirectoryIterator
- RecursiveFilterIterator
- RecursiveIteratorIterator

Interfaces
- Countable
- OuterIterator
- RecursiveIterator
- SeekableIterator

Exceptions
- BadFunctionCallException
- BadMethodCallException
- DomainException
- InvalidArgumentException
- LengthException
- LogicException
- OutOfBoundsException
- OutOfRangeException
- OverflowException
- RangeException
- RuntimeException
- UnderflowException
- UnexpectedValueException

SPL functions
- class_implements
- spl_autoload_call
- spl_autoload_extensions
- spl_autoload_functions
- spl_autoload_register
- spl_autoload_unregister
- spl_autoload

File handling
- SplFileInfo

Streams
- stream_bucket_append

- stream_bucket_make_writeable
- stream_bucket_new
- stream_bucket_prepend
- stream_context_create
- stream_context_get_default
- stream_context_get_options
- stream_context_get_params
- stream_context_set_default
- stream_context_set_option
- stream_context_set_params
- stream_copy_to_stream
- stream_filter_append
- stream_filter_prepend
- stream_filter_register
- stream_filter_remove
- stream_get_contents
- stream_get_filters
- stream_get_line
- stream_get_meta_data
- stream_get_transports
- stream_get_wrappers
- stream_select
- stream_set_blocking
- stream_set_timeout
- stream_socket_accept
- stream_socket_client
- stream_socket_get_name
- stream_socket_pair
- stream_socket_recvfrom
- stream_socket_sendto
- stream_socket_server
- stream_socket_shutdown
- stream_wrapper_register
- stream_wrapper_restore
- stream_wrapper_unregister

String

- addcslashes
- addslashes
- bin2hex
- chop
- chr
- chunk_split
- convert_cyr_string
- convert_uudecode
- convert_uuencode
- count_chars
- crc32
- echo
- explode
- fprintf
- get_html_translation_table
- hebrev
- hebrevc
- html_entity_decode
- htmlentities

- htmlspecialchars
- htmlspecialchars_decode
- implode
- join
- levenshtein
- localeconv
- ltrim
- md5
- md5_file
- metaphone
- money_format
- nl2br
- nl_langinfo
- number_format
- ord
- parse_str
- print
- printf
- quoted_printable_decode
- quotemeta
- rtrim
- setlocale
- sha1
- sha1_file
- similar_text
- soundex
- sprintf
- sscanf
- str_ireplace
- str_pad
- str_repeat
- str_replace
- str_rot13
- str_shuffle
- str_split
- str_word_count
- strcasecmp
- strchr
- strcmp
- strcoll
- strcspn
- strip_tags
- stripcslashes
- stripos
- stripslashes
- stristr
- strlen
- strnatcasecmp
- strnatcmp
- strncasecmp
- strncmp
- strpbrk
- strpos
- strrchr
- strrev
- strripos

- strrpos
- strspn
- strstr
- strtok
- strtolower
- strtoupper
- strtr
- substr
- substr_compare
- substr_count
- substr_replace
- trim
- ucfirst
- ucwords
- vfprintf
- vprintf
- vsprintf
- wordwrap

Tokenizer

- token_get_all
- token_name

URL

- base64_decode
- base64_encode
- http_build_query
- parse_url
- rawurldecode
- rawurlencode
- urldecode
- urlencode

Variable handling

- debug_zval_dump
- doubleval
- empty
- floatval
- get_defined_vars
- get_resource_type
- gettype
- intval
- is_array
- is_bool
- is_callable
- is_double
- is_float
- is_int
- is_integer
- is_long
- is_null
- is_numeric
- is_object
- is_real
- is_resource
- is_scalar
- is_string

- isset
- print_r
- serialize
- settype
- strval
- unserialize
- unset
- var_dump
- var_export

For more information about the PHP extension functions, see the PHP: Hypertext Preprocessor website.

Related concepts:

"Accessing elements in the message tree from a PHPCompute node" on page 2692
Access the contents of a message, for reading or writing, by using the structure and arrangement of the elements in the tree that the parser creates from the input bit stream.

Related tasks:

"Using PHP" on page 2670
You can use the PHP scripting language for message routing and transformation.

"Creating PHP code for a PHPCompute node" on page 2672
Use these instructions to create your PHP code and associate it with your PHPCompute node.

"Accessing user-defined properties from a PHPCompute node" on page 2716
Customize a PHPCompute node to access properties that you have associated with the message flow in which the node is included.

Related reference:

"Differences between WebSphere Message Broker PHP and PHP.net"
Some differences exist between PHP.net and the implementation of PHP that is provided by WebSphere Message Broker.

"PHPCompute node" on page 4639
Use the PHPCompute node to route and transform an incoming message, using the PHP scripting language.

"PHP API" on page 5313
The WebSphere Message Broker PHP API provides classes to represent the message, the message assembly, and the element tree.

"PHP data types" on page 5323
PHP data types are supported by WebSphere Message Broker.

Differences between WebSphere Message Broker PHP and PHP.net:

Some differences exist between PHP.net and the implementation of PHP that is provided by WebSphere Message Broker.

The differences for each function are shown in the following tables.

Table 270. MySQL functions

Function	Differences
mysql_fetch_field	Checks for a distinction between types that are returned from TINYTEXT and VARCHAR. One is a blob, the other is a string. This implementation returns both as VARCHAR, therefore it is not possible to distinguish between them. The call returns the maximum length that is defined in the database, not the maximum length that is used. For example, (`VARCHAR[50] xyz = abc`) gives *3* on PHP.net, and *50* on WebSphere Message Broker PHP.

Table 271. String functions

Function	Differences
crypt	Not available on Windows.
html_entity_decode	Results in a warning if the specified character set hint is not equal to UTF8.
htmlentities	Results in a warning if the specified character set hint is not equal to UTF8.
htmlspecialchars	Results in a warning if the specified character set hint is not equal to UTF8.

Table 272. Session handling functions

Function	Differences
Session handling functions	The INI option support session.use_trans_sid is not supported.

Table 273. File system functions

Function	Differences
All file system functions	Safe mode is not supported.
clearstatcache	No statcache exists, therefore this function has no effect.
flock	You cannot use flock(, LOCK_EX) to obtain an exclusive lock on a file that is opened as read-only, unless you also have write permissions for the file; this does not mean that you must open the file in write mode, just that you have permission to do so. If you do not have write permissions and False is returned indicating that no lock has been obtained, a warning is generated.
fopen	The PHP.net runtime issues a "No such file or directory" message when an invalid mode is used within fopen, whereas WebSphere Message Broker PHP issues a message saying that it is an invalid mode.
fseek	When you use fopen and file modes with the "t" option under Windows (Windows translation), fseek does not operate in the same way as the runtime PHP.net runtime.

Table 273. File system functions (continued)

Function	Differences
realpath	The realpath() function does not detect changes to file names that were previously symbolic links because of the canonical file name cache in Java. To replicate the behavior in PHP, you can start the JVM with `Dsun.io.useCanonCaches=false`

Table 274. BCMath Arbitrary Precision Mathematics functions

Function	Differences
All BCMath Arbitrary Precision Mathematics functions	WebSphere Message Broker PHP raises an E_WARNING, whereas PHP.net writes a warning to STDERR.
bcmul	WebSphere Message Broker PHP does not truncate trailing zeros.
bcpow	WebSphere Message Broker PHP does not truncate trailing zeros.
bcpowmod	WebSphere Message Broker PHP does not support trailing zeros when a scale other than 0 is used.

Table 275. Network functions

Function	Differences
fsockopen	The errno output argument is not supported. The errstr argument is not supported. Encrypted streams (SSL and TLS) are not available.

Table 276. Array functions

Function	Differences
All array functions	When sorting array entries that have equal weight under the sorting algorithm in use, the resulting sorted array might have entries in a different order to that produced by the sorting algorithms used by PHP.net. For example, a SORT_NUMERIC sort of the values "a","b","c".

Table 277. Multibyte string functions

Function	Differences
All multibyte string functions	If you specify the correct encoding for character data and if the character data is correct, the mb_* functions operate in the same way as MBString. No guarantee exists that the output is the same as MBString if the character data is not valid or if you do not specify the correct encoding. For example, mb_substr, if you pass in str and define it as UTF-32 when it is really ASCII or UTF-8, you do not get the same result as PHP.net.

Table 277. Multibyte string functions (continued)

Function	Differences
mb_convert_encoding	Does not support an array or string containing multiple encodings, or *auto* being passed as the from_encoding value. A PHP warning is generated in these cases.

Table 278. Program execution functions

Function	Differences
All program execution functions	WebSphere Message Broker PHP does not support Safe mode.
proc_open	WebSphere Message Broker PHP does not support the additional options that are specified by the optional sixth parameter.
proc_terminate	WebSphere Message Broker PHP does not support the sending of signals to the child process, as specified by the optional second parameter.
proc_get_status	This function is not supported by WebSphere Message Broker PHP.

Table 279. Error handling and logging functions

Function	Differences
error_log	Does not support message_type of 1 (email). A warning is issued and function returns *false*.

Table 280. Stream functions

Function	Differences
stream_get_meta_data	The unread bytes field (used buffer size) in return is not always the same for file streams, because the file stream implementation does not use buffers.
stream_context_create	Support for stream contexts is limited; the HTTP stream type supports only header, method, timeout, and content options.

Related concepts:

"Accessing elements in the message tree from a PHPCompute node" on page 2692
Access the contents of a message, for reading or writing, by using the structure and arrangement of the elements in the tree that the parser creates from the input bit stream.

Related tasks:

"Using PHP" on page 2670
You can use the PHP scripting language for message routing and transformation.

"Creating PHP code for a PHPCompute node" on page 2672
Use these instructions to create your PHP code and associate it with your PHPCompute node.

"Accessing user-defined properties from a PHPCompute node" on page 2716
Customize a PHPCompute node to access properties that you have associated with the message flow in which the node is included.

Related reference:

"PHP extensions" on page 5324
WebSphere Message Broker supports a set of PHP extensions.

"PHPCompute node" on page 4639
Use the PHPCompute node to route and transform an incoming message, using the PHP scripting language.

"PHP API" on page 5313
The WebSphere Message Broker PHP API provides classes to represent the message, the message assembly, and the element tree.

"PHP data types" on page 5323
PHP data types are supported by WebSphere Message Broker.

PHP constants for type values:

You can use type values to create syntax elements in your message tree.

The following tables detail PHP constants for type variables and their corresponding XMLNSC constants:
- "Common field types"
- "XMLNSC field types"
- "JSON field types" on page 5344

Common field types

Use the following common field type constants to create syntax elements in the message tree.

Construct	PHP Field Type constant	Field Type constant
A name element	MB_TYPE_NAME	Name
A name value element	MB_TYPE_NAME_VALUE	NameValue
A value element	MB_TYPE_VALUE	Value

XMLNSC field types

Use the following XMLNSC field type constants to create syntax elements in the message tree.

Construct	PHP Field Type constant	Field Type constant
Simple element	MB_XMLNSC_FIELD MB_XMLNSC_CDATA_FIELD	XMLNSC.Field XMLNSC.CDataField
Attribute	MB_XMLNSC_SINGLE_ATTRIBUTE MB_XMLNSC_ATTRIBUTE	XMLNSC.SingleAttribute XMLNSC.Attribute
Mixed content	MB_XMLNSC_VALUE MB_XMLNSC_CDATA_VALUE	XMLNSC.Value XMLNSC.CDataValue
Namespace declaration	MB_XMLNSC_NAMESPACE_DECL	XMLNSC.NamespaceDecl
Complex element	MB_XMLNSC_FOLDER	XMLNSC.Folder
Inline DTD	MB_XMLNSC_DOCUMENT_TYPE	XMLNSC.DocumentType
XML declaration	MB_XMLNSC_XML_DECLARATION	XMLNSC.XmlDeclaration
Entity reference	MB_XMLNSC_ENTITY_REFERENCE	XMLNSC.EntityReference
Entity definition	MB_XMLNSC_SINGLE_ENTITY MB_XMLNSC_ENTITY	XMLNSC.SingleEntityDefinition XMLNSC.EntityDefinition

Construct	PHP Field Type constant	Field Type constant
Comment	MB_XMLNSC_COMMENT	XMLNSC.Comment
Processing instruction	MB_XMLNSC_PROCESSING_INSTRUCTION	XMLNSC.ProcessingInstruction
BLOB type element to be written directly to the bit stream	MB_XMLNSC_BITSTREAM	XMLNSC.BitStream

JSON field types

Use the following JSON field type constants to create syntax elements in the message tree.

Construct	PHP Field Type constant	Field Type constant
JSON array	MB_JSON_ARRAY	JSON.ARRAY

Related concepts:

"PHP overview" on page 2671
WebSphere Message Broker provides support for the PHP scripting language.

Related tasks:

"Using PHP" on page 2670
You can use the PHP scripting language for message routing and transformation.

Related reference:

"PHPCompute node" on page 4639
Use the PHPCompute node to route and transform an incoming message, using the PHP scripting language.

"Broker Archive editor" on page 6794
Use the Broker Archive editor to create and manage broker archive (BAR) files.

User-defined patterns

View the reference information in this section to modify user-defined patterns by using the Java API and the PHP API. Use these APIs to write code that modifies a pattern instance when a pattern instance is generated by a pattern user.

APIs

To find information about APIs, see the following topics:

- "Java API for user-defined patterns" on page 5345
- "PHP API for user-defined patterns" on page 5345

Related concepts:

"Patterns" on page 1310
A *pattern* is a reusable solution that encapsulates a tested approach to solving a common architecture, design, or deployment task in a particular context.

"User-defined patterns" on page 1334
A *user-defined pattern* extends the function of WebSphere Message Broker so that you are able to create patterns that you can reuse within your organization.

"Modifying pattern instances by using Java or PHP" on page 1364
Use Java or PHP code to modify pattern instances when the pattern user generates an instance of a user-defined pattern. For example, you can use Java or PHP code to modify the structure of a message flow based on the values of pattern parameters.

Related tasks:

"Extending a user-defined pattern" on page 1340
Extend a user-defined pattern to customize its behavior so that it is easier to use or to provide guidance to pattern users.

Java API for user-defined patterns

Use the Java API to write Java code to modify user-defined pattern instances.

To view the Javadoc for the Java API, see Java API.

For the Java user-defined node API, which applies to user-defined nodes and to Java code that is called by JavaCompute nodes, see Java user-defined extensions API.

Related concepts:

"Modifying pattern instances by using Java or PHP" on page 1364
Use Java or PHP code to modify pattern instances when the pattern user generates an instance of a user-defined pattern. For example, you can use Java or PHP code to modify the structure of a message flow based on the values of pattern parameters.

PHP API for user-defined patterns

Use the PHP API to write PHP code to develop user-defined pattern applications.

PHP API

For information about the mb_pattern_run_template function, see "mb_pattern_run_template."

For information about the variables available in the _MB superglobal, see "_MB superglobal variables" on page 5347.

For information about the PHP extensions supported by WebSphere Message Broker, see "PHP extensions" on page 5348.

Related concepts:

"Patterns" on page 1310
A *pattern* is a reusable solution that encapsulates a tested approach to solving a common architecture, design, or deployment task in a particular context.

"User-defined patterns" on page 1334
A *user-defined pattern* extends the function of WebSphere Message Broker so that you are able to create patterns that you can reuse within your organization.

"Modifying pattern instances by using Java or PHP" on page 1364
Use Java or PHP code to modify pattern instances when the pattern user generates an instance of a user-defined pattern. For example, you can use Java or PHP code to modify the structure of a message flow based on the values of pattern parameters.

Related tasks:

"Extending a user-defined pattern" on page 1340
Extend a user-defined pattern to customize its behavior so that it is easier to use or to provide guidance to pattern users.

mb_pattern_run_template:

Use the mb_pattern_run_template function to run a PHP script from within another script in user-defined patterns.

The following example shows how the mb_pattern_run_template function is called:

```
mb_pattern_run_template(string $project_name, string template_path, string
$output_path);
```

The arguments taken by mb_pattern_run_template are shown in the following table.

Argument	Description
project_name	The name of the pattern instance project to which the output from the template is written. You can include the pattern instance name if you are not using an exemplar project.For example, if the pattern includes an exemplar project called *project1*, and the pattern user creates an instance called *instance1*, the *project_name* is *project1* (although the actual project name created is *instance1_project1*). However, if the pattern is not an exemplar project, you can specify the prefix for that project: *instance1_project1*.
template_path	The relative path from this PHP script to the template PHP script that you want to run. For example, if the template script is called `example.esql.php` and it is in the same directory, *template_path* is `example.esql.php`. If the template script is in a subdirectory, `testdirectory`, of the directory where the current script is located, *template_path* is `testdirectory/example.esql.php`.
output_path	The relative path in the project to where the output from the template script is written. This path is relative to the root directory of the pattern instance project specified by *project_name*. The path must exist. For example, if the output file is called `example.esql` and is written to the root directory of the pattern instance project, *output_path* is `example.esql`. If the file is written to a subdirectory, `outputdirectory`, of the pattern instance project, *output_path* is `outputdirectory/example.esql`

Related tasks:

"Modifying pattern instances by using PHP" on page 1389
Add PHP code to a code plug-in project to modify a pattern instance when the pattern instance is generated by a pattern user.

"Examples of PHP API code" on page 1390
Use the following examples of PHP API code for common tasks to help you write your own PHP code to modify pattern instances.

"Testing PHP API code" on page 1394
After writing PHP code to modify pattern instances, test the code to check that it works correctly.

Related reference:

"_MB superglobal variables"
The PHP API for user-defined patterns contains four variables within the _MB superglobal. You can use these variables in PHP scripts that are used to modify instances of user-defined patterns.

"PHP extensions" on page 5348
WebSphere Message Broker supports a set of PHP extensions.

"Differences between WebSphere Message Broker PHP and PHP.net" on page 5363
Some differences exist between PHP.net and the implementation of PHP that is provided by WebSphere Message Broker.

_MB superglobal variables:

The PHP API for user-defined patterns contains four variables within the _MB superglobal. You can use these variables in PHP scripts that are used to modify instances of user-defined patterns.

The _MB superglobal for the PHP API for user-defined patterns contains the variables shown in the following table.

Variable	Description

PATTERN_INSTANCE_MANAGER	Includes methods to log messages and get access to message flows. For more information about the methods available in PATTERN_INSTANCE_MANAGER, see "Java API for user-defined patterns" on page 5345. The following example shows you how to access PATTERN_INSTANCE_MANAGER methods:
	`<?php`
	`  $pim = $_MB["PATTERN_INSTANCE_MANAGER"];`
	`  $flow = $pim->getMessageFlow("Transform", "mqsi/Tra`
	`  $node = $flow->getNodeByName("MQInput");`
	`?>`

PP	Returns an array that is populated with the pattern parameters. The following example shows you how to print the value of the parameter *pp2*:
	`<?php`
	`  var_dump($_MB['PP']['pp2']);`
	`?>`

PATTERN_INSTANCE_NAME	Returns the name of the current pattern instance that is being generated.
WORKSPACE_ROOT	Returns the full path to the current workspace location.

Related tasks:

"Modifying pattern instances by using PHP" on page 1389
Add PHP code to a code plug-in project to modify a pattern instance when the pattern instance is generated by a pattern user.

"Examples of PHP API code" on page 1390
Use the following examples of PHP API code for common tasks to help you write your own PHP code to modify pattern instances.

"Testing PHP API code" on page 1394
After writing PHP code to modify pattern instances, test the code to check that it

works correctly.

Related reference:

"mb_pattern_run_template" on page 5345
Use the mb_pattern_run_template function to run a PHP script from within another script in user-defined patterns.

"PHP extensions"
WebSphere Message Broker supports a set of PHP extensions.

"Differences between WebSphere Message Broker PHP and PHP.net" on page 5363
Some differences exist between PHP.net and the implementation of PHP that is provided by WebSphere Message Broker.

PHP extensions:

WebSphere Message Broker supports a set of PHP extensions.

The following PHP functions (listed by extension) are provided by WebSphere Message Broker. However, some differences exist between PHP.net and the implementation of PHP that is provided by WebSphere Message Broker. For more information about these differences, see "Differences between WebSphere Message Broker PHP and PHP.net" on page 5339.

Array

- array
- array_change_key_case
- array_chunk
- array_combine
- array_count_values
- array_diff
- array_diff_assoc
- array_diff_key
- array_diff_uassoc
- array_diff_ukey
- array_fill
- array_fill_keys
- array_filter
- array_flip
- array_intersect
- array_intersect_assoc
- array_intersect_key
- array_intersect_uassoc
- array_intersect_ukey
- array_key_exists
- array_keys
- array_map
- array_merge
- array_merge_recursive
- array_multisort
- array_pad
- array_pop
- array_product
- array_push
- array_rand
- array_reduce
- array_reverse
- array_search
- array_shift

- array_slice
- array_splice
- array_sum
- array_udiff
- array_udiff_assoc
- array_udiff_uassoc
- array_uintersect
- array_uintersect_assoc
- array_uintersect_uassoc
- array_unique
- array_unshift
- array_values
- array_walk
- array_walk_recursive
- arsort
- asort
- compact
- count
- current
- each
- end
- extract
- in_array
- key
- key_exists
- krsort
- ksort
- natcasesort
- natsort
- next
- pos
- prev
- range
- reset
- rsort
- shuffle
- sizeof
- sort
- uasort
- uksort
- usort

BC Math

- bcadd
- bccomp
- bcdiv
- bcmod
- bcmul
- bcpow
- bcpowmod
- bcscale
- bcsqrt
- bcsub

Class-object

- call_user_method
- call_user_method_array

- class_exists
- get_class
- get_class_methods
- get_class_vars
- get_declared_classes
- get_declared_interfaces
- get_object_vars
- get_parent_class
- interface_exists
- is_a
- is_subclass_of
- method_exists
- property_exists

Date and time

- checkdate
- date
- date_create
- date_date_set
- date_default_timezone_get
- date_default_timezone_set
- date_format
- date_isodate_set
- date_modify
- date_offset_get
- date_parse
- date_sun_info
- date_sunrise
- date_sunset
- date_time_set
- date_timezone_get
- date_timezone_set
- DateTime_construct
- DateTime_format
- DateTime_getOffset
- DateTime_getTimeZone
- DateTime_modify
- DateTime_setDate
- DateTime_setISODate
- DateTime_setTime
- DateTime_setTimeZone
- DateTimeZone_construct
- DateTimeZone_getName
- DateTimeZone_getOffset
- DateTimeZone_getTransitions
- DateTimeZone_listAbbreviations
- DateTimeZone_listIdentifiers
- getdate
- gettimeofday
- gmdate
- gmmktime
- gmstrftime
- idate
- localtime
- microtime
- mktime

- strftime
- strptime
- strtotime
- time
- timezone_abbreviations_list
- timezone_identifiers_list
- timezone_name_from_abbr
- timezone_name_get
- timezone_offset_get
- timezone_open
- timezone_transitions_get

Directory

- chdir
- closedir
- dir
- Directory.close
- Directory.read
- getcwd
- opendir
- readdir
- rewinddir
- scandir

Error handling and logging

- debug_backtrace
- debug_print_backtrace
- error_get_last
- error_log
- error_reporting
- restore_error_handler
- restore_exception_handler
- set_error_handler
- set_exception_handler
- trigger_error
- user_error

File system

- basename
- chgrp
- chmod
- chown
- clearstatcache
- copy
- dirname
- disk_free_space
- diskfreespace
- fclose
- feof
- fflush
- fgetc
- fgetcsv
- fgets
- file
- file_exists
- file_get_contents
- file_put_contents

- fileatime
- filectime
- filegroup
- fileinode
- filemtime
- fileowner
- fileperms
- filesize
- filetype
- flock
- fopen
- fpassthru
- fputcsv
- fputs
- fread
- fseek
- fstat
- ftell
- ftruncate
- fwrite
- glob
- is_dir
- is_executable
- is_file
- is_link
- is_readable
- is_uploaded_file
- is_writable
- is_writeable
- link
- linkinfo
- lstat
- mkdir
- move_uploaded_file
- parse_ini_file
- pathinfo
- pclose
- popen
- readfile
- readlink
- realpath
- rename
- rewind
- rmdir
- stat
- symlink
- tempnam
- touch
- umask
- unlink

Function handling

- call_user_func
- call_user_func_array
- create_function
- func_get_arg

- func_get_args
- func_num_args
- function_exists
- get_defined_functions
- register_shutdown_function
- register_tick_function
- unregister_tick_function

Java

- java_import

Mail

- ezmlm_hash
- mail

Math

- abs
- acos
- acosh
- asin
- asinh
- atan
- atan2
- atanh
- base_convert
- bindec
- ceil
- cos
- cosh
- decbin
- dechex
- decoct
- deg2rad
- exp
- expm1
- floor
- fmod
- getrandmax
- hexdec
- hypot
- is_finite
- is_infinite
- is_nan
- lcg_value
- log
- log10
- log1p
- max
- min
- mt_getrandmax
- mt_rand
- mt_srand
- octdec
- pi
- pow
- rad2deg
- rand

- round
- sin
- sinh
- sqrt
- srand
- tan
- tanh

Message Broker
- mb_get_user_defined_property

Miscellaneous
- constant
- define
- defined
- eval
- highlight_string
- ignore_user_abort
- pack
- sleep
- uniqid
- unpack
- usleep

Multibyte string
- mb_convert_encoding
- mb_decode_mimeheader
- mb_detect_encoding
- mb_detect_order
- mb_encode_mimeheader
- mb_ereg
- mb_eregi
- mb_eregi_replace
- mb_ereg_replace
- mb_internal_encoding
- mb_language
- mb_regex_encoding
- mb_regex_set_options
- mb_split
- mb_stripos
- mb_stristr
- mb_strlen
- mb_strpos
- mb_strrchr
- mb_strrichr
- mb_strripos
- mb_strrpos
- mb_strstr
- mb_strtolower
- mb_strtoupper
- mb_substitute_character
- mb_substr
- mb_substr_count

MySQL
- mysql_affected_rows
- mysql_client_encoding
- mysql_close

- mysql_connect
- mysql_create_db
- mysql_data_seek
- mysql_db_query
- mysql_drop_db
- mysql_errno
- mysql_error
- mysql_escape_string
- mysql_fetch_array
- mysql_fetch_assoc
- mysql_fetch_field
- mysql_fetch_lengths
- mysql_fetch_object
- mysql_fetch_row
- mysql_field_flags
- mysql_field_len
- mysql_field_name
- mqsql_field_seek
- mysql_field_table
- mysql_field_type
- mysql_free_result
- mysql_get_client_info
- mysql_get_host_info
- mqsql_get_proto_info
- mysql_get_server_info
- mqsql_info
- mysql_insert_id
- mysql_list_dbs
- mysql_list_fields
- mysql_list_processes
- mysql_list_tables
- mysql_num_fields
- mysql_num_rows
- mysql_pconnect
- mysql_ping
- mysql_query
- mysql_real_escape_string
- mysql_result
- mysql_select_db
- mysql_set_charset
- mysql_stat
- mysql_tablename
- mysql_thread_id
- mysql_unbuffered_query

Network

- checkdnsrr
- closelog
- define_syslog_variables
- fsockopen
- gethostbyaddr
- gethostbyname
- gethostbynamel
- getmxrr
- inet_ntop
- inet_pton

- ip2long
- long2ip
- pfsockopen
- socket_get_status
- socket_set_blocking
- socket_set_timeout
- syslog

Output control

- flush
- ob_clean
- ob_end_clean
- ob_end_flush
- ob_flush
- ob_get_clean
- ob_get_contents
- ob_get_flush
- ob_get_length
- ob_get_level
- ob_get_status
- ob_implicit_flush
- ob_list_handlers
- ob_start

PHP options and information

- assert
- assert_options
- extension_loaded
- get_cfg_var
- get_current_user
- get_defined_constants
- get_extension_funcs
- get_include_path
- get_included_files
- get_loaded_extensions
- get_magic_quotes_gpc
- get_magic_quotes_runtime
- getenv
- getmypid
- getrusage
- ini_alter
- ini_get
- ini_get_all
- ini_restore
- ini_set
- magic_quotes_runtime
- memory_get_usage
- php_sapi_name
- php_uname
- phpinfo
- phpversion
- putenv
- restore_include_path
- set_include_path
- set_magic_quotes_runtime
- set_time_limit
- sys_get_temp_dir

- version_compare
- zend_version

POSIX Regex

- ereg
- ereg_replace
- eregi
- eregi_replace
- split
- spliti
- sql_regcase

Program execution

- escapeshellarg
- escapeshellcmd
- exec
- passthru
- proc_close
- proc_open
- proc_terminate
- shell_exec
- system

PCRE

- preg_grep
- preg_last_error
- preg_match
- preg_match_all
- preg_quote
- preg_replace
- preg_replace_callback
- preg_split

Session

- session_cache_expire
- session_cache_limiter
- session_commit
- session_decode
- session_destroy
- session_encode
- session_get_cookie_params
- session_id
- session_is_registered
- session_module_name
- session_name
- session_regenerate_id
- session_register
- session_save_path
- session_set_cookie_params
- session_set_save_handler
- session_start
- session_unregister
- session_unset
- session_write_close

SNMP

- snmp2_get
- snmp2_getnext

- snmp2_real_walk
- snmp2_set
- snmp2_walk
- snmp3_get
- snmp3_getnext
- snmp3_real_walk
- snmp3_set
- snmp3_walk
- snmp_get_quick_print
- snmp_get_valueretrieval
- snmp_read_mib
- snmp_set_enum_print
- snmp_set_oid_numeric_print
- snmp_set_oid_output_format
- snmp_set_quick_print
- snmp_set_valueretrieval
- snmpget
- snmpgetnext
- snmprealwalk
- snmpset
- snmpwalk
- snmpwalkoid

SPL

Data structures
- SplObjectStorage

Iterators
- AppendIterator
- ArrayIterator
- CachingIterator
- DirectoryIterator
- EmptyIterator
- FilterIterator
- InfiniteIterator
- IteratorIterator
- LimitIterator
- NoRewindIterator
- RecursiveDirectoryIterator
- RecursiveFilterIterator
- RecursiveIteratorIterator

Interfaces
- Countable
- OuterIterator
- RecursiveIterator
- SeekableIterator

Exceptions
- BadFunctionCallException
- BadMethodCallException
- DomainException
- InvalidArgumentException
- LengthException
- LogicException
- OutOfBoundsException
- OutOfRangeException

- OverflowException
- RangeException
- RuntimeException
- UnderflowException
- UnexpectedValueException

SPL functions

- class_implements
- spl_autoload_call
- spl_autoload_extensions
- spl_autoload_functions
- spl_autoload_register
- spl_autoload_unregister
- spl_autoload

File handling

- SplFileInfo

Streams

- stream_bucket_append
- stream_bucket_make_writeable
- stream_bucket_new
- stream_bucket_prepend
- stream_context_create
- stream_context_get_default
- stream_context_get_options
- stream_context_get_params
- stream_context_set_default
- stream_context_set_option
- stream_context_set_params
- stream_copy_to_stream
- stream_filter_append
- stream_filter_prepend
- stream_filter_register
- stream_filter_remove
- stream_get_contents
- stream_get_filters
- stream_get_line
- stream_get_meta_data
- stream_get_transports
- stream_get_wrappers
- stream_select
- stream_set_blocking
- stream_set_timeout
- stream_socket_accept
- stream_socket_client
- stream_socket_get_name
- stream_socket_pair
- stream_socket_recvfrom
- stream_socket_sendto
- stream_socket_server
- stream_socket_shutdown
- stream_wrapper_register
- stream_wrapper_restore
- stream_wrapper_unregister

String

- addcslashes

- addslashes
- bin2hex
- chop
- chr
- chunk_split
- convert_cyr_string
- convert_uudecode
- convert_uuencode
- count_chars
- crc32
- echo
- explode
- fprintf
- get_html_translation_table
- hebrev
- hebrevc
- html_entity_decode
- htmlentities
- htmlspecialchars
- htmlspecialchars_decode
- implode
- join
- levenshtein
- localeconv
- ltrim
- md5
- md5_file
- metaphone
- money_format
- nl2br
- nl_langinfo
- number_format
- ord
- parse_str
- print
- printf
- quoted_printable_decode
- quotemeta
- rtrim
- setlocale
- sha1
- sha1_file
- similar_text
- soundex
- sprintf
- sscanf
- str_ireplace
- str_pad
- str_repeat
- str_replace
- str_rot13
- str_shuffle
- str_split
- str_word_count
- strcasecmp
- strchr

- strcmp
- strcoll
- strcspn
- strip_tags
- stripcslashes
- stripos
- stripslashes
- stristr
- strlen
- strnatcasecmp
- strnatcmp
- strncasecmp
- strncmp
- strpbrk
- strpos
- strrchr
- strrev
- strripos
- strrpos
- strspn
- strstr
- strtok
- strtolower
- strtoupper
- strtr
- substr
- substr_compare
- substr_count
- substr_replace
- trim
- ucfirst
- ucwords
- vfprintf
- vprintf
- vsprintf
- wordwrap

Tokenizer

- token_get_all
- token_name

URL

- base64_decode
- base64_encode
- http_build_query
- parse_url
- rawurldecode
- rawurlencode
- urldecode
- urlencode

Variable handling

- debug_zval_dump
- doubleval
- empty
- floatval
- get_defined_vars

- get_resource_type
- gettype
- intval
- is_array
- is_bool
- is_callable
- is_double
- is_float
- is_int
- is_integer
- is_long
- is_null
- is_numeric
- is_object
- is_real
- is_resource
- is_scalar
- is_string
- isset
- print_r
- serialize
- settype
- strval
- unserialize
- unset
- var_dump
- var_export

For more information about the PHP extension functions, see the PHP: Hypertext Preprocessor website.

Related tasks:

"Modifying pattern instances by using PHP" on page 1389
Add PHP code to a code plug-in project to modify a pattern instance when the pattern instance is generated by a pattern user.

"Examples of PHP API code" on page 1390
Use the following examples of PHP API code for common tasks to help you write your own PHP code to modify pattern instances.

"Testing PHP API code" on page 1394
After writing PHP code to modify pattern instances, test the code to check that it works correctly.

Related reference:

"mb_pattern_run_template" on page 5345
Use the mb_pattern_run_template function to run a PHP script from within another script in user-defined patterns.

"_MB superglobal variables" on page 5347
The PHP API for user-defined patterns contains four variables within the _MB superglobal. You can use these variables in PHP scripts that are used to modify instances of user-defined patterns.

"Differences between WebSphere Message Broker PHP and PHP.net" on page 5363
Some differences exist between PHP.net and the implementation of PHP that is provided by WebSphere Message Broker.

Differences between WebSphere Message Broker PHP and PHP.net:

Some differences exist between PHP.net and the implementation of PHP that is provided by WebSphere Message Broker.

The differences for each function are shown in the following tables.

Table 281. MySQL functions

Function	Differences
mysql_fetch_field	Checks for a distinction between types that are returned from TINYTEXT and VARCHAR. One is a blob, the other is a string. This implementation returns both as VARCHAR, therefore it is not possible to distinguish between them. The call returns the maximum length that is defined in the database, not the maximum length that is used. For example, (`VARCHAR[50] xyz = abc`) gives *3* on PHP.net, and *50* on WebSphere Message Broker PHP.

Table 282. String functions

Function	Differences
crypt	Not available on Windows.
html_entity_decode	Results in a warning if the specified character set hint is not equal to UTF8.
htmlentities	Results in a warning if the specified character set hint is not equal to UTF8.
htmlspecialchars	Results in a warning if the specified character set hint is not equal to UTF8.

Table 283. Session handling functions

Function	Differences
Session handling functions	The INI option support session.use_trans_sid is not supported.

Table 284. File system functions

Function	Differences
All file system functions	Safe mode is not supported.
clearstatcache	No statcache exists, therefore this function has no effect.
flock	You cannot use flock(, LOCK_EX) to obtain an exclusive lock on a file that is opened as read-only, unless you also have write permissions for the file; this does not mean that you must open the file in write mode, just that you have permission to do so. If you do not have write permissions and False is returned indicating that no lock has been obtained, a warning is generated.

Table 284. File system functions (continued)

Function	Differences
fopen	The PHP.net runtime issues a "No such file or directory" message when an invalid mode is used within fopen, whereas WebSphere Message Broker PHP issues a message saying that it is an invalid mode.
fseek	When you use fopen and file modes with the "t" option under Windows (Windows translation), fseek does not operate in the same way as the runtime PHP.net runtime.
realpath	The realpath() function does not detect changes to file names that were previously symbolic links because of the canonical file name cache in Java. To replicate the behavior in PHP, you can start the JVM with `Dsun.io.useCanonCaches=false`

Table 285. BCMath Arbitrary Precision Mathematics functions

Function	Differences
All BCMath Arbitrary Precision Mathematics functions	WebSphere Message Broker PHP raises an E_WARNING, whereas PHP.net writes a warning to STDERR.
bcmul	WebSphere Message Broker PHP does not truncate trailing zeros.
bcpow	WebSphere Message Broker PHP does not truncate trailing zeros.
bcpowmod	WebSphere Message Broker PHP does not support trailing zeros when a scale other than 0 is used.

Table 286. Network functions

Function	Differences
fsockopen	The errno output argument is not supported. The errstr argument is not supported. Encrypted streams (SSL and TLS) are not available.

Table 287. Array functions

Function	Differences
All array functions	When sorting array entries that have equal weight under the sorting algorithm in use, the resulting sorted array might have entries in a different order to that produced by the sorting algorithms used by PHP.net. For example, a SORT_NUMERIC sort of the values "a","b","c".

Table 288. Multibyte string functions

Function	Differences
All multibyte string functions	If you specify the correct encoding for character data and if the character data is correct, the mb_* functions operate in the same way as MBString. No guarantee exists that the output is the same as MBString if the character data is not valid or if you do not specify the correct encoding. For example, mb_substr, if you pass in str and define it as UTF-32 when it is really ASCII or UTF-8, you do not get the same result as PHP.net.
mb_convert_encoding	Does not support an array or string containing multiple encodings, or *auto* being passed as the from_encoding value. A PHP warning is generated in these cases.

Table 289. Program execution functions

Function	Differences
All program execution functions	WebSphere Message Broker PHP does not support Safe mode.
proc_open	WebSphere Message Broker PHP does not support the additional options that are specified by the optional sixth parameter.
proc_terminate	WebSphere Message Broker PHP does not support the sending of signals to the child process, as specified by the optional second parameter.
proc_get_status	This function is not supported by WebSphere Message Broker PHP.

Table 290. Error handling and logging functions

Function	Differences
error_log	Does not support message_type of 1 (email). A warning is issued and function returns *false*.

Table 291. Stream functions

Function	Differences
stream_get_meta_data	The unread bytes field (used buffer size) in return is not always the same for file streams, because the file stream implementation does not use buffers.
stream_context_create	Support for stream contexts is limited; the HTTP stream type supports only header, method, timeout, and content options.

Related tasks:

"Modifying pattern instances by using PHP" on page 1389
Add PHP code to a code plug-in project to modify a pattern instance when the pattern instance is generated by a pattern user.

"Examples of PHP API code" on page 1390
Use the following examples of PHP API code for common tasks to help you write your own PHP code to modify pattern instances.

"Testing PHP API code" on page 1394
After writing PHP code to modify pattern instances, test the code to check that it works correctly.

Related reference:

"mb_pattern_run_template" on page 5345
Use the mb_pattern_run_template function to run a PHP script from within another script in user-defined patterns.

"_MB superglobal variables" on page 5347
The PHP API for user-defined patterns contains four variables within the _MB superglobal. You can use these variables in PHP scripts that are used to modify instances of user-defined patterns.

"PHP extensions" on page 5348
WebSphere Message Broker supports a set of PHP extensions.

Message model reference information

Reference information in this section can help you develop and configure message models.

Message model reference information is available for:

- "Message set preferences"
- "Message set properties" on page 5371
- "Message definition file properties" on page 5409
- "Message category properties" on page 5413
- "Message model object properties" on page 5416
- "Deprecated message model object properties" on page 6069
- "Additional MRM domain information" on page 6251
- "Additional MIME domain information" on page 6322
- "Import formats" on page 6346
- "Message model wizards" on page 6360

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

Message set preferences

Preferences for message sets.

Property	Type	Meaning
Default version tag	String	Provide the default version information you would like to be set in the message set `Version` property when you create a new message set.

You can alter a number of the preferences that affect the way certain areas of message set processing are handled. The areas are:

- "Message Set Editor and Message Definition Editor preferences"
- "Validation of the message model" on page 5369
- "XML Schema Importer" on page 5370

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"Version and keyword information for deployable objects" on page 1443
Use the Broker Archive editor to view the version and keyword information of deployable objects.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with a message set" on page 2840
Complete a variety of tasks that are involved in working with a message set.

"Configuring message set preferences" on page 2840
This task topic explains how to make changes to preferences that relate to message set processing. These preferences are for message set editors, message set model validation, and importing XML Schema.

"Configuring CVS to run with the WebSphere Message Broker Toolkit" on page 573
Install CVS as a normal program by following the usual prompts. Not all versions of *CVSNT* are supported by Eclipse.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message Set Editor and Message Definition Editor preferences"
While looking at a large message set that contains a number of message definition files that have different namespaces, or multiple message definition files that have the same namespace, you might want to view the information in alternative ways to make it easier for you to visualize the structure of the message set. If you double-click the global construct, you open the message definition file in which the global construct is defined.

"Validation of the message model" on page 5369
You can customize some of the warning messages that are generated by message set validation. Use the Message Set Validation Preference page to do this.

"XML Schema Importer" on page 5370
Preferences for the message set XML Schema Importer.

"Message set properties" on page 5371
Message sets have properties that you can set to define their characteristics and the way in which they are processed.

Message Set Editor and Message Definition Editor preferences:

While looking at a large message set that contains a number of message definition files that have different namespaces, or multiple message definition files that have the same namespace, you might want to view the information in alternative ways

to make it easier for you to visualize the structure of the message set. If you double-click the global construct, you open the message definition file in which the global construct is defined.

Message set editor settings

Property	Type	Meaning
Group by namespace and then by collections	Button	Selecting this view groups the global constructs by namespace then by collection (for example, Messages, Types, Groups, or Elements and Attributes). Using this view you can visualize all of the constructs that belong to each of the defined namespaces.
Group by collections and then by namespace	Button	Selecting this view groups the global constructs by collection (for example, Messages, Types, Groups, or Elements and Attributes) then by namespace. Using this view you can visualize which global construct in the message set is defined in which namespace.

Message definition editor settings

Property	Type	Meaning
Show base complex types	Check box	Where your complex type is based on another complex type that is derived by an extension, selecting this will display the base complex type in the outline view.
Prefix for created messages	String	This property allows you to specify a prefix to precede the name of the initial complex type in the name of the created message. This prefix applies only to messages created from C or COBOL files. The default value is `msg_`. Note, however, that no prefix is applied when a message is created from a C file, and the selected preprocessing option is SAP ALE IDoc or SAP File IDoc.

Tab Extensions

Click **Tab Extensions** to display check boxes that allow you to determine what tabs are enabled for the Message Set Editor, the Message Definition Editor, and the Message Category Editor. All these check boxes are always selected and cannot be cleared.

Editor	Tab Extensions
Message Set Editor	Properties
Message Definition Editor	Overview Properties
Message Category Editor	Properties

A control is provided that allows you to choose the order in which the tab extensions for each of the editors are displayed.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with a message set" on page 2840
Complete a variety of tasks that are involved in working with a message set.

"Configuring message set preferences" on page 2840
This task topic explains how to make changes to preferences that relate to message set processing. These preferences are for message set editors, message set model validation, and importing XML Schema.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Validation of the message model"
You can customize some of the warning messages that are generated by message set validation. Use the Message Set Validation Preference page to do this.

"XML Schema Importer" on page 5370
Preferences for the message set XML Schema Importer.

Validation of the message model:

You can customize some of the warning messages that are generated by message set validation. Use the Message Set Validation Preference page to do this.

Any warning or error that falls into a category from the following list can be customized according to the relevant category. The customization can affect both severity and priority.

The severity can be one of the following values:

- Error
- Warning
- Info
- Ignore

If the severity is not *Ignore*, the priority can be one of the following values:

- High
- Normal
- Low

If the severity is *Ignore*, you cannot change the priority.

Message set validation settings

The categories that you can customize are described in the following list:

- Use of deprecated constructs
- Messages with abstract global elements
- Facet runtime validation differences
- Type/Element substitution runtime validation differences
- Mixed content runtime validation differences
- Wildcard runtime validation differences
- Unique Particle Attribution checks

- Tagged/Delimited String group content
- Zero Custom Wire Format length count
- Zero Tagged/Delimited String Format length count
- Empty Tagged/Delimited String Format tag
- List or Union with Custom Wire Format
- List or Union with Tagged/Delimited String Format
- Unbounded max occurs with Custom Wire Format
- Unbounded max occurs with Tagged/Delimited String Format

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with a message set" on page 2840
Complete a variety of tasks that are involved in working with a message set.

"Configuring message set preferences" on page 2840
This task topic explains how to make changes to preferences that relate to message set processing. These preferences are for message set editors, message set model validation, and importing XML Schema.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message Set Editor and Message Definition Editor preferences" on page 5367
While looking at a large message set that contains a number of message definition files that have different namespaces, or multiple message definition files that have the same namespace, you might want to view the information in alternative ways to make it easier for you to visualize the structure of the message set. If you double-click the global construct, you open the message definition file in which the global construct is defined.

"XML Schema Importer"
Preferences for the message set XML Schema Importer.

XML Schema Importer:

Preferences for the message set XML Schema Importer.

You can customize the following categories that affect the way in which an XML Schema is imported into a message set that does not support namespaces.

Category	Modify	Reject	Accept
Import	Converts Import to Include	Import fails if it sees an Import	Not applicable
Redefine	Removes the Redefine statements	Import fails if it sees a Redefine	Redefine imported (gives task list error)
List	Changes type base to xsd:string	Import fails if it sees a List	List imported

Category	Modify	Reject	Accept
Union	Changes type base to xsd:string	Import fails if it sees a Union	Union imported
Abstract Complex Type	Sets abstract to false	Import fails if it sees an Abstract Complex Type	Abstract Complex Type imported
Abstract Element	Sets abstract to false	Import fails if it sees an Abstract Element	Abstract Element imported

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with a message set" on page 2840
Complete a variety of tasks that are involved in working with a message set.

"Configuring message set preferences" on page 2840
This task topic explains how to make changes to preferences that relate to message set processing. These preferences are for message set editors, message set model validation, and importing XML Schema.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message Set Editor and Message Definition Editor preferences" on page 5367
While looking at a large message set that contains a number of message definition files that have different namespaces, or multiple message definition files that have the same namespace, you might want to view the information in alternative ways to make it easier for you to visualize the structure of the message set. If you double-click the global construct, you open the message definition file in which the global construct is defined.

"Validation of the message model" on page 5369
You can customize some of the warning messages that are generated by message set validation. Use the Message Set Validation Preference page to do this.

Message set properties

Message sets have properties that you can set to define their characteristics and the way in which they are processed.

General message set properties

The following table defines the properties that you can set to customize the message set.

Property	Type	Meaning
Default message domain and Supported message domains	String and check boxes	The message parser name must match the *Message Domain* property of any input node that processes messages from the message set, or the <Msd> element value of any MQRFH2 header that precedes a message from the message set. Choose a value from the list offered for the *Default Message Domain* property, and select check boxes (from *Supported Message Domains*) to choose other domains. You can select as many of these check boxes as you want. Use the message parser name when you write ESQL field references for messages in the message set; for example, `InputRoot.MRM.Document`. The Mapping editor and the content assist feature of the ESQL editor use the message parser name when they generate ESQL field references. You can choose from the following names: • XMLNSC (the default if you select **Finish** from page two of the New Message Set wizard). Choose this domain if you want to model XML messages. You can deploy the message set to brokers if you want, because the XMLNSC parser optionally uses the message set at run time. • MRM. Choose this domain for binary or text messages. You can also use this domain for XML messages. You must deploy the message set to the brokers that receive these messages. The deploy action creates a runtime dictionary against which the MRM parser checks the received message. • SOAP. Choose this domain for SOAP Web Services. • DataObject. Choose this domain for data from WebSphere Adapters. • XMLNS. You might need to choose this domain for some kinds of XML messages. You do not have to deploy the message set to brokers, because the XMLNS parser does not use the message set at run time. • JMSMap. Choose this domain if you want to model a JMS MapMessage message. You do not have to deploy the message set to brokers, because this parser does not use the message set at run time. • JMSStream. Choose this domain if you want to model a JMS StreamMessage message. You do not have to deploy the message set to brokers, because this parser does not use the message set at run time. • MIME. Choose this domain if you want to model a MIME message. You do not have to deploy the message set to brokers, because the MIME parser does not use the message set at run time. • JSON. Choose this domain if you want to model a JSON message. You do not have to deploy the message set to brokers, because the JSON parser does not use the message set at run time. • XML. This domain is deprecated. Use the XMLNSC domain instead. • IDOC. This domain is deprecated. Use the MRM domain instead.
Use namespaces	Check box	Select this property if you want to use namespaces within the message set. Namespaces provide a method of avoiding naming conflicts where different document definitions have elements of the same name. For further information see Namespaces. By default, this check box is selected. Using namespaces affects how elements are created in the logical message tree. Each element in the message tree has both a name and a namespace, so an ESQL or Java reference to one of these elements has to specify both name and namespace. Therefore, using namespaces has an effect on the ESQL or the Java that you write. Always select this property if you want to use the message set to model XML messages.

MRM domain

Property	Type	Meaning
Default wire format	String	(Optional) Specify the default wire format used, only if you select MRM as the default message domain, or MRM is selected in the list of supported message domains. The default value is <no default specified>. If you do not select MRM, either as the default message domain or as one of the supported message domains, the Default Wire Format property is unavailable.
Message set ID	String	This property is a unique identifier that is automatically generated for you when you create the message set. You cannot change this property.
Message set alias	String	Specify an alternative unique value that identifies the message set. This property is only required if you are using the Message Identity technique to identify embedded messages. Using this technique, the embedded messages are defined in this message set but the parent message is defined in a different message set, and the bit stream does not contain the actual message set name or identifier.
Message type prefix	String	This property is used when you define multipart messages, specifically when using the Message Path technique to identify embedded messages. The value that you specify is used as an absolute or relative path to the innermost message from the outermost, and is used as a prefix to the value of the *Message Type* property that is specified for the outermost message (specified either in the MQRFH2 header of the message, or in the input node of the message flow). If you set a value, it must be in the form id1/id2/.../idnu where id1 is the identifier of the outermost message, id2 is the identifier of the next element or message, and idn is the identifier of the innermost message. The default value is blank (not set). The following table, describing the use of the message set property *Message Type Prefix*, shows how this value is combined with the *Message Type* property of an input message.
Broker will treat Length facet as MaxLength	Check box	Select this property if you want the COBOL importer to create a maxLength facet, rather than a length facet, for a fixed length string element. By default, this check box is selected.

Use of the *Message type prefix* property

The following table shows the implications of using the property *Message type prefix*. The message type or message prefix can describe either elements or messages.

Message Type property example	Message type prefix not set	Message type prefix set
Simple Message Type:`msg_type`	Results in the simple Message Type:`msg_type`	Results in the path Message Type: `/msg_prefix_1/.../msg_prefix_n/msg_type`
Path Message Type:`msg_type_1/.../msg_type_m`	Results in the path Message Type:`/msg_type_1/.../msg_type_m`	Results in the combined path Message Type: `/msg_prefix_1.../msg_prefix_n /msg_type_1/.../msg_type_m`

Message Type property example	Message type prefix not set	Message type prefix set
Simple absolute Message Type:/msg_type	Results in the simple Message Type:msg_type	Results in the simple Message Type:msg_type An error is raised if Message Type Prefix is set to any value other than msg_type.
Path absolute Message Type:/msg_type_1/.../msg_type_m	Results in the path Message Type:/msg_type_1/.../msg_type_m	Results in the path Message Type:/msg_type_1/.../msg_type_m An error is raised if all identifiers in Message Type Prefix do not match the corresponding identifiers in the resulting path.

If you are using MRM or IDOC domains, in addition to the main message set properties, you can update message set properties that are specific to each of the physical formats. The following reference topics describe these properties.

Related concepts:

"Parsers" on page 1072
A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from the internal message tree representation.

"XMLNSC parser" on page 1090
The XMLNSC parser is a flexible, general-purpose XML parser that offers high performance XML parsing and optional XML Schema validation.

"XMLNS parser" on page 1104
The XMLNS parser is a flexible, general-purpose XML parser.

"XML parser" on page 1110
The XML domain is very similar to the XMLNS domain, but the XML domain has no support for XML namespaces or opaque parsing.

"MRM parser and domain" on page 1111
You can use the MRM domain to parse and write a wide range of message formats.

"Which XML parser should you use?" on page 1080
If your messages are general-purpose XML documents, you can use one of the dedicated XML domains (XMLNSC or XMLNS) to parse the message, or you can use the MRM domain to parse the message.

"JMS parsers and domains" on page 1116
The JMSMap and JMSStream domains can be used for modeling messages that are produced by the implementations of the Java Messaging Service standard.

"SOAP parser and domain" on page 1082
You can use the SOAP parser to create a common WSDL-based logical tree format for working with Web services, independent of the physical bitstream format.

"Multipart messages" on page 1191
A *multipart message* contains one or more other messages within its structure. The contained message is sometimes referred to as an *embedded message*.

"Identifying an embedded message by using a Message Identity" on page 1193
You can identify an embedded message by using the Message Identity. This technique is used by the MRM domain, and replaces the use of the Message Key.

"Identifying an embedded message by using a Message Path" on page 1196
The Message Path technique for identifying embedded messages is useful when the multipart message contains no information about the identity of an embedded message.

"Message set projects" on page 1161
A *message set project* is a specialized container in which you create and maintain all the resources associated with one message set.

"Message sets overview" on page 1162
A message set is a container for grouping messages and associated message resources (elements, types, groups).

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"Namespaces in the message model" on page 1201
Use namespaces to qualify message model object names.

Related tasks:

"Accessing embedded messages in the MRM domain" on page 2594
If you have defined a multipart message, you have at least one message embedded within another. Within the overall complex type that represents the outer messages, you can model the inner message in two ways.

"Working with a message set project" on page 2838
Creating and deleting a message set.

"Working with a message set" on page 2840
Complete a variety of tasks that are involved in working with a message set.

Related reference:

"Custom Wire Format message set properties"
The tables define the properties that you can set for a Custom Wire Format message set.

"XML Wire Format message set properties" on page 5400
The following tables define the properties for the XML Wire Format for the message set.

"TDS Format message set properties" on page 5381
The following tables show the properties that you can set for a TDS format message set.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

Custom Wire Format message set properties:

The tables define the properties that you can set for a Custom Wire Format message set.

Some of the message set properties (marked with an asterisk (*)) are relevant only if the message being processed is ***not*** using WebSphere MQ as the transport protocol.

If the transport protocol is WebSphere MQ, values are derived from the message headers (for example, MQMD), and the message set properties, if set, are ignored.

Binary representation of boolean values

Property	Type	Meaning
Boolean True Value	String	Enter up to eight hexadecimal digits. Do not include the hexadecimal indicator (0x) preceding this number. Each digit is a half byte. The maximum length is 4 bytes. You must enter an even number of digits (a whole number of bytes). This value must be different from, but the same length as, the `Boolean False Value`. The default value is 00000001.
Boolean False Value	String	Enter up to eight hexadecimal digits. Do not include the hexadecimal indicator (0x) preceding this number. Each digit is a half byte. The maximum length is 4 bytes. You must enter an even number of digits (a whole number of bytes). This value must be different from, but the same length as, the `Boolean True Value`. The default value is 00000000.
Boolean Null Value	String	Enter up to eight hexadecimal digits. Do not include the hexadecimal indicator (0x) preceding this number. Each digit is a half byte. The maximum length is 4 bytes. You must enter an even number of digits (a whole number of bytes). This value can be the same as either `Boolean True Value` or `Boolean False Value`, or different. The default value is 00000000.

Output settings

Use these settings when messages are being produced.

Property	Type	Meaning
Byte Alignment Pad	String	If the xsd:element Custom Wire Format properties `Byte Alignment`, `Leading Skip Count`, and `Trailing Skip Count` cause bytes to be skipped in the bit stream when the message is serialized, this property supplies the character to be used in the skipped positions. Set this character in one of the following ways: • Select SPACE, NUL, or 0 (the default) from the list of values shown. • Enter a character between quotation marks, for example "c" or 'c', where c is any alphanumeric character. • Enter a decimal character code in the form YY where YY is a decimal value. • Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value. • Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal format.
Policy for Missing Elements	Enumerated	This property determines the action that is taken by the broker when fields are missing from the message tree when the message is serialized (for output): • Use *Default Value* (the default). If a *Default Value* exists for the element, write it; otherwise, throw an exception. • Use *Null Value*. If the *Nillable* property of the element is selected, and an *Encoding Null Value* is specified for the element, write the *Encoding Null Value* according to the rules that are defined by the *Encoding Null* property; otherwise, throw an exception.

Property	Type	Meaning
Truncate fixed length strings	Check box	This property applies only to output strings. If this check box is selected, and the element or attribute is a fixed length string (that is, the logical type is xsd:string and the physical type is Fixed Length String) that is longer than either the length that is specified in the model or the length reference, the string is truncated to this length. No exception is raised on output, unless validation (see "Validating messages" on page 1478) is active. The end from which data is truncated is determined by the value of the *Justification* property. If the value of the *Justification* property is Left justify, data is truncated from the right; if the value of the *Justification* property is Right justify, data is truncated from the left. However, if the value of the *Justification* property is Not applicable, truncation does not occur and an exception occurs if the string is too long. If this check box is cleared, an exception occurs if the element or attribute is a fixed length string (that is, the logical type is xsd:string and the physical type is Fixed Length String) that is longer than either the length that is specified in the model, or the length reference. This behavior occurs in releases of the WebSphere Message Broker earlier than Version 6.1. By default, this check box is cleared.

Binary representation of decimal values

Property	Type	Meaning
Packed Decimal Positive Code	Enumerated	Select, from the list, the positive sign that is used for packed decimal numbers. The default value is `C`, which indicates that `0x0C` is used as the positive sign; this value is used in most systems. You can also select `F`, which indicates that `0x0F` is used as the positive sign; this value is used in some systems.

Datetime settings

Property	Type	Meaning
Derive default dateTime format from logical type	Button	Select this option if you want the default dateTime format to be determined by the logical type of the element or attribute. You can override this property for an element or attribute within a complex type.
Use default dateTime format	Button and String	Select this option if you want to specify a default dateTime format that is fixed for all elements or attributes of logical type dateTime, date, time, gYear, gYearMonth, gMonth, gMonthDay, and gDay. You can override this property for an element or attribute within a complex type. For more information, see "DateTime formats" on page 6310.
Start of century for 2-digit years	Integer	This property determines how 2-digit years are interpreted. Specify the two digits that start a 100-year window that contains the current year. For example, if you specify 89, and the current year is 2002, all 2-digit dates are interpreted as being in the range 1989 - 2088.

Property	Type	Meaning
Days in First Week of Year	Enumerated	Specify the number of days of the new year that must fall within the first week. The start of a year typically falls in the middle of a week. If the number of days in that week is less than the value specified here, the week is considered to be the last week of the previous year; therefore, week 1 starts some days into the new year. Otherwise, it is considered to be the first week of the new year; in this case, week 1 starts some days before the new year. Select Use Broker Locale, which causes the broker to get the information from the underlying platform, or select a number from the list that is displayed.
First Day Of Week	Enumerated	Specify the day on which each new week starts. Select Use Broker Locale, which causes the broker to get the information from the underlying platform, or select a value from the list that is displayed.

Property	Type	Meaning
Strict DateTime Checking	Check box	Select this option if you want to restrict dateTimes to a valid dateTime format. If `Strict DateTime Checking` is selected, receiving an incorrect dateTime causes an error. **Strict dateTime checking** Examples of strict dateTime checking are: • DateTimes are restricted to valid dateTimes only. When you use this option, a date such as the 35th March is not processed as 4th April, and 10:79 is not processed as 11:19. Receiving an out-of-band dateTime, such as these examples, causes an error to occur. • The number of characters for a numeric dateTime component must be within the bounds of the corresponding formatting symbols. Repeat the symbol to specify the minimum number of digits that you require. The maximum number of digits that is permitted becomes the upper bound for a particular symbol. For example, day in month has an upper bound of 31; therefore, a format string of 'd' allows the values 2 and 21 to be parsed, but does not allow the values 32 and 210. On output, numbers are padded with zeros to the specified length. A year is a special case; see the message set property `Start of century for 2 digit years`. For fractional seconds, the length must implicitly match the number of format symbols on input. The output is rounded to the specified length. • White space is not skipped over. The white space in the input string must correspond with the same number and position of white space in the formatting string. • If data remains that is not parsed in the input string after all the symbols in the formatting string have been matched, an error occurs. **Lenient dateTime checking** Examples of lenient dateTime checking are: • The parser converts out-of-band dateTime values to the appropriate in-band value. For example, a date of 2005-05-32 is converted to 2005-06-01. • Output of dateTimes always adheres to the symbol count. For example, a formatting string of yyyy-MM-dd (where '-' is the field separator) allows one or more characters to be parsed against MM and dd. Therefore, dates that are not valid - for example, 2005-1-123 and 2005-011-12 - can be entered. The first value of 2005-1-123 is output as the date 2005-05-03, and the second value of 2005-011-12 is output as the date 2005-11-12. • The number of the timezone formatting symbol Z is applicable only to the output dateTime format. • White space is skipped over.
Time Zone	Enumerated	The value that you set for this property is used if the value that you specified for the `Default DateTime Format` property does not include Time Zone information. The initial value is Use Broker Locale, which causes the broker to get the information from the underlying platform. You can change this property by selecting from the list of values.
Daylight Saving Time	Check box	Select this option if the area in the `Time Zone` property observes daylight saving time. If it does not observe daylight saving time, do not select this option. For example, if an area is selected in `Time Zone` and this option is not selected, the value passed represents the time zone without the daylight saving time.

Property	Type	Meaning
Use input UTC format on output	Check box	This property applies to elements and attributes of logical type xsd:dateTime or xsd:time that contain a dateTime as a string and that have a dateTime format of I, IU, T, or TU, or that include ZZZ or ZZZU. Such elements and attributes can specify Coordinated Universal Time (UTC) by using either the Z character or timezone +00:00 in the value. On input, the MRM parser remembers the way that UTC was specified. If this property is selected, and the element or attribute is copied to an output message, the UTC format is preserved into the output message and overrides the format that is implied by the dateTime format property. If this property is cleared, or if the element or attribute was not copied from an input message, the UTC format in the output message is controlled solely by the dateTime format property.

Character and numeric encoding for non-WebSphere MQ messages

Use these settings only for messages with no MQMD.

Property	Type	Meaning
Default CCSID*	Integer	Enter a numeric value for the default Coded Character Set Identifier. The default is 500. If the input message is a WebSphere MQ message, the equivalent attribute that is set for the queue manager is used, and this property is ignored.
Default Byte Order*	Enumerated	Select either Big Endian (the default) or Little Endian from the list to specify the byte order of numbers that are represented as binary integers. In C, this is equivalent to data type short or long. In COBOL, this is equivalent to a PIC 9, COMP, COMP-4, COMP-5, or BINARY data type. Your choice must match the encoding with which messages are created. Typically, Big Endian is the correct option for messages that are created on UNIX or z/OS; Little Endian is the correct option for messages that are created on Windows. Do not use this property if the message is received across the WebSphere MQ transport protocol; in this case, the property is deduced from the MQMD of the message, or from the encoding of the broker queue manager.
Default Packed Decimal Byte Order*	Enumerated	Select Big Endian (the default) or Little Endian from the displayed list to specify the byte order of numbers that are represented as packed decimal. In COBOL, this is equivalent to PIC 9 COMP-3 data type. There is no equivalent data type in C. Your choice must match the encoding with which messages are created. Typically, Big Endian is the correct option for messages that are created on UNIX or z/OS; Little Endian is the correct option for messages that are created on Windows.
Default Float Format*	Enumerated	Select one of S390 (the default), IEEE, or Reverse IEEE from the displayed list to specify the byte order of numbers in the message that are represented as floating point.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise

appearance of your message bit stream during transmission.

"MRM Custom Wire Format" on page 1214
Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not separated by delimiters.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message set properties" on page 5371
Message sets have properties that you can set to define their characteristics and the way in which they are processed.

"XML Wire Format message set properties" on page 5400
The following tables define the properties for the XML Wire Format for the message set.

"TDS Format message set properties"
The following tables show the properties that you can set for a TDS format message set.

TDS Format message set properties:

The following tables show the properties that you can set for a TDS format message set.

See "Default TDS message set properties" on page 5394 for the default settings of these properties for each of the industry standards.

Messaging Standard

Property	Type	Meaning
Messaging Standard	Enumerated	Specify the standard to be used for this wire format. Select one of the following values: • User Defined Text - for text data not based on a standard • SWIFT • ACORD AL3 • EDIFACT • X12 • TLOG • HL7 • CSV - Comma Separated Values • User Defined Mixed - for mixed text and binary data If you are defining your own tagged/delimited messages, or are using a standard that is not included in the list of values shown, select either User Defined Text, if all your data is text, or User Defined Mixed, if not all your data is text. The value that you select for this property determines the default values of some of the other properties. The default is User Defined Text.

Data element separation settings

Property	Type	Meaning
Group Indicator	String	Specify the default value of a special character or string that precedes the data that belongs to a group or complex type within the bit stream.
Group Terminator	String	Specify the default value of a special character or string that terminates data that belongs to a group or a complex type within the bit stream.
Delimiter	String	Specify the default value of a special character or string that specifies the delimiter that is used between data elements. This property applies only to the delimited `Data Element Separation` methods (Tagged Delimited, All Elements Delimited, and Variable Length Elements Delimited).
Suppress Absent Element Delimiters	Enumerated	Use this property to select whether you want delimiters to be suppressed for elements that are missing within a message. Select from: • End Of Type. Use this option to suppress the delimiter when an element is missing. For example, if the model has been defined to have up to three elements and only two are present, the last delimiter can be omitted from the message. • Never. Use this option to ensure that even if optional elements are not present, all delimiters are written out. This option must be used when the same delimiter is used to delimit parent objects and child objects. For example, if an optional child element is missing and all the delimiters are the same, message processing applications cannot tell where the child elements in a message ends and where the next parent element starts.

Property	Type	Meaning
Tag Data Separator	String	Specify the default value of a special character or string that separates the tag from the data. If you set the property `Tag Data Separator`, the `Length of Tag` property is ignored. This property applies only to the tagged `Data Element Separation` methods (Tagged Delimited, Tagged Fixed Length, and Tagged Encoded Length).
Length of Tag	Integer	Specify the default length of a tag value. When the message is parsed, this property allows tags to be extracted from the bit stream. The `Tag Data Separator` and `Length of Tag` properties are mutually exclusive. If you set the property `Tag Data Separator`, the `Length of Tag` property is ignored. This property applies only to the tagged `Data Element Separation` methods (Tagged Delimited, Tagged Fixed Length, and Tagged Encoded Length).

Note: Any value that you set for a group or complex type property overrides the value that you set for the corresponding message set property.

Character data settings

Property	Type	Meaning
Default CCSID	Integer	CCSID (Coded Character Set Identification) specifies the mapping between character codes and symbols. You must specify a code set that is supported by WebSphere Message Broker. This property stores the default CCSID for the message bit stream, but this value can be overridden when the message is processed (for example, by the CCSID in the header of a WebSphere MQ input message).

Property	Type	Meaning
Trim on input	Enumerated	This property applies only to elements and attributes with a physical type of Text. This property specifies whether a simple element or attribute value is to be trimmed when it is parsed. The property does not apply to a simple element, or attribute, with a logical type of Boolean or Binary. All trimming is applied to element or attribute values before the conversion of the value to its logical type. This property does not apply when writing elements or attributes. This property only applies to a simple element, or attribute, that is contained within a complex type or group that has the `Justification` property set to Left Justify or Right Justify, and that satisfies one of the following conditions: • The `Data Element Separation` property is set to Fixed Length, Fixed Length AL3, Tagged Fixed Length, Use Data Pattern, or Tagged Encoded Length. • The `Data Element Separation` property is set to Variable Length Elements Delimited, and the element or attribute has a value set for its model length or length reference. • The `Data Element Separation` property is set to Tagged Delimited or All Elements Delimited, and the `Observe Element Length` property is set. The element or attribute has a model length or length reference value set. This property can be set to one of the following values: • No Trim. No characters are trimmed from the element or attribute value. • Leading White Spaces. White space characters are trimmed from the left of the element or attribute value. • Trailing White Spaces. White space characters are trimmed from the right of the element or attribute value. • Trim Both. White space characters are trimmed from both the left and the right of the element or attribute value. • Trim Padding Chars. Padding characters are trimmed from the element or attribute value. The padding character is set by the `Padding Character` property of the element or attribute. If the `Justification` property of the element or attribute is set to Left Justify, the padding characters are trimmed from the right. If the `Justification` property of the element or attribute is set to Right Justify, the padding characters are trimmed from the left. If the `Justification` property of the element or attribute is set to Not Applicable, no trimming takes place. White space characters include control characters that are in the range from U+0000 to U+001f and from U+007f to U+009f. You might need to use this property if you have data input that is mapped to a numeric simple type. For example, if the input data has leading spaces, you can set this property to Leading White Spaces to avoid data conversion problems when you process these fields.

Property	Type	Meaning
Truncate on output	Check box	This property applies only to output strings that have a physical type of Text. The property applies to elements or attributes that have a logical type of xsd:string and that are contained within a structure with a `Data Element Separation` of Fixed Length, Fixed Length AL3, Tagged Fixed Length, Use Data Pattern, or Variable Length Elements Delimited where a length has been specified. If this check box is selected, and the element or attribute has a length that is longer than the length that is specified in the model or the length reference, the string is truncated to this length. No exception is raised on output, unless validation (see "Validating messages" on page 1478) is active. The end from which data is truncated is determined by the value of the Justification property. If the value of the Justification property is Left justify, data is truncated from the right; if the value of the Justification property is Right justify, data is truncated from the left. However, if the value of the Justification property is Not applicable, truncation does not occur and an exception occurs if the string is too long. If this check box is cleared, an exception occurs if the element or attribute is a fixed length string (that is, the physical type is Text and a length has been specified) that is longer than either the length that is specified in the model or the length reference. This behavior occurs in releases of the WebSphere Message Broker earlier than Version 6.1.
Escape Character	Button and String	Specify the escape character that is used to allow special reserved characters (such as delimiters) to be included as part of data. You must specify a single character only, or a mnemonic that represents a single character. Escape characters apply only in variable length fields. Escape characters, on parsing, always escape the next character, and are always removed. Escape characters, on writing, are inserted in front of all the characters that are listed in `Reserved Characters`. You can specify either an escape character or a quote character, but not both, for a given message set.
Quote Character	Button and String	Specify the quote character that is used to allow special reserved characters (such as delimiters) to be included as part of data. You must specify a single character only, or a mnemonic that represents a single character. Quote characters apply only to variable length fields. Quote characters, on parsing, must be present at both the start and the end of the data, and are always removed. Quote characters, on writing, are added to both the start and end of the data, if the data contains any character that is listed in the `Reserved Characters` property. You can specify either an `Escape Character` or a `Quote Character`, but not both, for a given message set.

Property	Type	Meaning
Reserved Characters	String	Specify any special reserved characters. Either these reserve characters must be preceded by the `Escape Character`, or the data field that contains them must be delimited by a pair of `Quote Characters`, if they are to be included as part of the data. The `Escape Character`, `Quote Character`, delimiters, and group indicators must be included in this list. If the set of reserved characters is to be updated dynamically (in the case of EDIFACT and X12 when reserved characters, such as delimiters, are specified in service strings), you must use the supplied mnemonics to specify characters in this list. If you have specified `Reserved Characters`, an `Escape Character` or a `Quote Character` must also be specified. Reserved characters apply only in variable length fields. Reserved characters are not used when parsing.

Numeric settings

Property	Type	Meaning
Decimal Point	String	Specify the character that is used to separate the whole part of a number from its fraction.
Packed decimal positive code	String	Controls the positive sign that is used for packed decimal fields. Valid values are `C` or `F`.Specify the character that is used to separate the whole part of a number from its fraction.
Strict Numeric Checking	Check box	Use this property in conjunction with the `Messaging Standard` property, the `Virtual Decimal Point` property and the `Precision` property of an element. Using this property allows you to apply stricter rules for the checking of numbers. The rules for `Strict Numeric Checking` are: • If the `Precision` property of an element is set to All Significant Digits , a decimal separator is present only if the value has a fractional part. • If the `Precision` property of an element is set to Explicit Decimal Point, the decimal separator must always be present, even if the fractional part is missing. • If the `Precision` property of an element is set to Exponential Notation, the incoming value must be in exponential notation. Exponential notation is only allowed for floating numbers. • If the `Precision` property of an element is set to a specific value, the specific number of digits after the decimal separator must be present. • All values must contain at least one digit in the integer part of the number. • If a `Virtual Decimal Point` of an element has been set, the number must not have a decimal point. • Except for EDIFACT, the decimal separator can be only the specified value, and '.' is not permitted. For EDIFACT, both '.' and the specified separator are permitted. In this case, the decimal separator must be specified as ',' and the code permits '.' to be used. • Except for exponential functions, only digits 0-9, the decimal separator, the positive sign, and the negative sign are permitted. For exponential functions the characters 'e' and 'E' are also permitted. Padding characters are permitted only if they are in a position to be stripped from the number.

Property	Type	Meaning
Derive sign from logical type	Check box	If this property is selected, an unset TDS `Signed` property attempts to derive its value from the simple type of the element (integer and decimal simple logical types only). For these logical types it applies only to the Integer, External Decimal, and Packed Decimal physical types.
Default byte order	Enumerated	Controls the byte order of numbers that are represented as binary integers for messages with no MQMD. Valid values are Big Endian or Little Endian. This property stores the default byte order for numbers that are represented as binary integers for messages with no MQMD, but this value can be overridden when the message is processed.
Default packed decimal byte order	Enumerated	Controls the byte order of numbers that are represented as packed decimal for messages with no MQMD. Valid values are Big Endian or Little Endian. This property stores the default byte order of numbers that are represented as packed decimal for messages with no MQMD, but this value can be overridden when the message is processed.
Default float format	Enumerated	Controls the format of numbers that are represented as float for messages with no MQMD. Valid values are S390, IEEE, or Reverse IEEE. This property stores the default format of numbers that are represented as float for messages with no MQMD, but this value can be overridden when the message is processed.

Representation of boolean values

Property	Type	Meaning
Text boolean true value	String	Specifies the character that represents the text Boolean true value.
Text boolean false value	String	Specifies the character that represents the text Boolean false value.
Text boolean null value	String	Specifies the character that represents the text Boolean null value.
Binary boolean true value	String	Specifies a hexadecimal value that represents the binary Boolean true value.
Binary boolean false value	String	Specifies a hexadecimal value that represents the binary Boolean false value.
Binary boolean null value	String	Specifies a hexadecimal value that represents the binary Boolean null value.

Datetime settings

Property	Type	Meaning
Derive default dateTime format from logical type	Button	Select this option if you want the default dateTime format to be determined by the logical type of the element or attribute. You can override this property for an element or attribute within a complex type.

Property	Type	Meaning
Use default dateTime format	Button and String	Select this option if you want to specify a default dateTime format that is fixed for all elements or attributes of logical type dateTime, date, time, gYear, gYearMonth, gMonth, gMonthDay, and gDay. You can override this property for an element or attribute within a complex type. For more information, see "DateTime formats" on page 6310.
Start of century for 2-digit years	Integer	This property determines how 2-digit years are interpreted. Specify the two digits that start a 100-year window that contains the current year. For example, if you specify 89, and the current year is 2002, all 2-digit dates are interpreted as being in the range 1989 - 2088.
Days in First Week of Year	Enumerated	Specify the number of days of the new year that must fall within the first week. The start of a year typically falls in the middle of a week. If the number of days in that week is less than the value specified here, the week is considered to be the last week of the previous year; therefore, week 1 starts some days into the new year. Otherwise, it is considered to be the first week of the new year; in this case, week 1 starts some days before the new year. Select Use Broker Locale, which causes the broker to get the information from the underlying platform, or select a number from the list that is displayed.
First Day Of Week	Enumerated	Specify the day on which each new week starts. Select Use Broker Locale, which causes the broker to get the information from the underlying platform, or select a value from the list that is displayed.

Property	Type	Meaning
Strict DateTime Checking	Check box	Select this option if you want to restrict dateTimes to a valid dateTime format. If `Strict DateTime Checking` is selected, receiving an incorrect dateTime causes an error. **Strict dateTime checking** Examples of strict dateTime checking are: • DateTimes are restricted to valid dateTimes only. When you use this option, a date such as the 35th March is not processed as 4th April, and 10:79 is not processed as 11:19. Receiving an out-of-band dateTime, such as these examples, causes an error to occur. • The number of characters for a numeric dateTime component must be within the bounds of the corresponding formatting symbols. Repeat the symbol to specify the minimum number of digits that you require. The maximum number of digits that is permitted becomes the upper bound for a particular symbol. For example, day in month has an upper bound of 31; therefore, a format string of 'd' allows the values 2 and 21 to be parsed, but does not allow the values 32 and 210. On output, numbers are padded with zeros to the specified length. A year is a special case; see the message set property `Start of century for 2 digit years`. For fractional seconds, the length must implicitly match the number of format symbols on input. The output is rounded to the specified length. • White space is not skipped over. The white space in the input string must correspond with the same number and position of white space in the formatting string. • If data remains that is not parsed in the input string after all the symbols in the formatting string have been matched, an error occurs. **Lenient dateTime checking** Examples of lenient dateTime checking are: • The parser converts out-of-band dateTime values to the appropriate in-band value. For example, a date of 2005-05-32 is converted to 2005-06-01. • Output of dateTimes always adheres to the symbol count. For example, a formatting string of yyyy-MM-dd (where '-' is the field separator) allows one or more characters to be parsed against MM and dd. Therefore, dates that are not valid - for example, 2005-1-123 and 2005-011-12 - can be entered. The first value of 2005-1-123 is output as the date 2005-05-03, and the second value of 2005-011-12 is output as the date 2005-11-12. • The number of the timezone formatting symbol Z is applicable only to the output dateTime format. • White space is skipped over.
Time Zone	Enumerated	The value that you set for this property is used if the value that you specified for the `Default DateTime Format` property does not include Time Zone information. The initial value is Use Broker Locale, which causes the broker to get the information from the underlying platform. You can change this property by selecting from the list of values.
Daylight Saving Time	Check box	Select this option if the area in the `Time Zone` property observes daylight saving time. If it does not observe daylight saving time, do not select this option. For example, if an area is selected in `Time Zone` and this option is not selected, the value passed represents the time zone without the daylight saving time.

Property	Type	Meaning
Use input UTC format on output	Check box	This property applies to elements and attributes of logical type xsd:dateTime or xsd:time that have a dateTime format of I, IU, T, or TU, or that include ZZZ or ZZZU. Such elements and attributes can specify Coordinated Universal Time (UTC) by using either the Z character or timezone +00:00 in the value. On input, the MRM parser remembers the way that UTC was specified. If this property is selected, and the element or attribute is copied to an output message, the UTC format is preserved into the output message and overrides the format that is implied by the dateTime format property. If this property is cleared, or the element or attribute was not copied from an input message, the UTC format in the output message is controlled solely by the dateTime format property.

General settings

Property	Type	Meaning
Output policy for missing elements	Enumerated	Controls whether the default value or null value is used on output for missing elements. Valid values are UseDefaultValue or UseNullValue.
Derive default length from logical type	Check box	If this property is selected, an unset TDS `Length` property attempts to derive its default value from the simple type of the element (string, binary, integer, and decimal simple logical types only). For these logical types, it applies only to the Binary, Text, Integer, External Decimal, and Packed Decimal physical types.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM TDS format" on page 1221
The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

Related reference:

"Message set properties" on page 5371
Message sets have properties that you can set to define their characteristics and the way in which they are processed.

"Custom Wire Format message set properties" on page 5375
The tables define the properties that you can set for a Custom Wire Format message set.

"XML Wire Format message set properties" on page 5400
The following tables define the properties for the XML Wire Format for the message set.

"TDS Mnemonics"
The Tagged/Delimited String Format (TDS) uses mnemonics for a number of properties for a message set, complex type, or both.

"White space characters in TDS" on page 5533
White space characters are defined as ASCII characters (hexadecimal) 'X'09 to 'X'0D and EBCDIC characters 'X'05, 'X'0B, 'X'0C, 'X'0D, 'X'25, and 'X'40.

"Default TDS message set properties" on page 5394
The following tables define the defaults for the message set properties for the TDS Format for each of the industry standard messages that you can define.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

TDS Mnemonics:

The Tagged/Delimited String Format (TDS) uses mnemonics for a number of properties for a message set, complex type, or both.

These TDS mnemonics and their associated properties are listed in the following table.

Mnemonic string	Meaning	Default value	Associated property
<EDIFACT_CS>	Component separator in EDIFACT	:	Message set and complex type or group, `Delimiter`
<EDIFACT_DEC_NOTATION>	Decimal notation in EDIFACT	.	Message set, `Decimal Point`
<EDIFACT_DS>	Data element separator in EDIFACT	+	Message set and complex type or group, `Delimiter`
<EDIFACT_ESC_CHAR>	Escape character in EDIFACT	?	Message set, `Escape Character`
<EDIFACT_GROUP_TERM>	Tag terminator in EDIFACT	'	Message set, `Group Terminator`
<EDIFACT_TAGDATA_SEP>	Tag data separator in EDIFACT This is overridden with the same value as that which overrides <EDIFACT_DS>	+	Message set and complex type or group, `Tag Data Separator`
<HL7_CS>	Component separator in HL7	^	Message set and complex type or group, `Delimiter`
<HL7_FS>	Data element separator in HL7	|	Message set and complex type or group, `Delimiter`
<HL7_RS>	Repeating element delimiter in HL7	~	Local element and element reference, `Repeating Element Delimiter`
<HL7_SCS>	Sub-component separator in HL7	&	Message set and complex type or group, `Delimiter`
<X12_CS>	Component separator for X12	:	Message set and complex type or group, `Delimiter`
<X12_DS>	Data element separator for X12	*	Message set and complex type or group, `Delimiter`

Mnemonic string	Meaning	Default value	Associated property
<X12_ERS>	Element repetition separator for X12	{	Local element and element reference, `Repeating Element Delimiter`
<X12_GROUP_TERM>	Tag terminator in X12	!	Message set level, `Group Terminator`

Mnemonics for control characters are shown in the following table.

Mnemonic	Hex value	Unicode	Description
<ACK>	X'06'	<U+0006>	Acknowledge
<BEL>	X'07'	<U+0007>	Bell
<BS>	X'08'	<U+0008>	Backspace
<CAN>	X'18'	<U+0018>	Cancel
<CR>	X'0D'	<U+000D>	Carriage Return
<DC1>	X'11'	<U+0011>	Device Control One
<DC2>	X'12'	<U+0012>	Device Control Two
<DC3>	X'13'	<U+0013>	Device Control Three
<DC4>	X'14'	<U+0014>	Device Control Four
<DLE>	X'10'	<U+0010>	Data Link Escape
<EM>	X'19'	<U+0019>	End of Medium
<ENQ>	X'05'	<U+0005>	Inquiry
<EOT>	X'04'	<U+0004>	End of Transmission
<ESC>	X'1B'	<U+001B>	Escape
<ETB>	X'17'	<U+0017>	End of Transmission Block
<ETX>	X'03'	<U+0003>	End of Text
<FF>	X'0C'	<U+000C>	Form Feed
<FS>	X'1C'	<U+001C>	File Separator
<GS>	X'1D'	<U+001D>	Group Separator
<GT>	X'3E'	<U+003E>	Greater Than
<HT>	X'09'	<U+0009>	Horizontal Tabulation
<LF>	X'0A'	<U+000A>	Line Feed
<LT>	X'3C'	<U+003C>	Less Than
<NAK>	X'15'	<U+0015>	Negative Acknowledge
<NUL>	X'00'	<U+0000>	Null-
<RS>	X'1E'	<U+001E>	Record Separator
<SI>	X'0F'	<U+000F>	Locking Shift Zero (Shift In)
<SO>	X'0E'	<U+000E>	Locking Shift One (Shift Out)
<SOH>	X'01'	<U+0001>	Start of Heading
<SP>	X'20'	<U+0020>	Space
<STX>	X'02'	<U+0002>	Start of Text
<SUB>	X'1A'	<U+001A>	Substitute

Mnemonic	Hex value	Unicode	Description
<SYN>	X'16'	<U+0016>	Synchronous Idle
<US>	X'1F'	<U+001F>	Unit Separator
<VT>	X'0B'	<U+000B>	Vertical Tabulation

These mnemonics were created for characters that cannot be entered into the message editor.

You can enter a mnemonic in the form `<U+NNNN>`, where `NNNN` are hexadecimal digits. None of the characters in this structure are case-sensitive. Do not enclose spaces inside the angle brackets. These numbers represent a Unicode character, not a character in the code page of the input message.

You can enter a mnemonic in the form `<0xNN>`, where `NN` are hexadecimal digits. None of the characters in this structure are case-sensitive. Do not enclose spaces inside the angle brackets. These numbers represent a raw hexadecimal byte value, not a character in the code page of the input message.

If a mnemonic is of the form `<0xNN>`, it is applied directly to the input data, and no code page conversion takes place. Otherwise, a mnemonic is applied to the data after the data has been converted into Unicode from the code page of the input data.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM TDS format" on page 1221
The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

Related reference:

"Message set properties" on page 5371
Message sets have properties that you can set to define their characteristics and the way in which they are processed.

"TDS Format message set properties" on page 5381
The following tables show the properties that you can set for a TDS format message set.

"Default TDS message set properties" on page 5394
The following tables define the defaults for the message set properties for the TDS Format for each of the industry standard messages that you can define.

Default TDS message set properties:

The following tables define the defaults for the message set properties for the TDS Format for each of the industry standard messages that you can define.

For more information about the TDS Format, see "TDS Format message set properties" on page 5381 and "TDS Mnemonics" on page 5391.

Default message set property values for TDS (part 1 of 3)

Property	Messaging standard = User Defined Text	Messaging Standard = SWIFT	Messaging standard = ACORD AL3
Group Indicator	Empty	<CR><LF>:	Empty
Group Terminator	Empty	<CR><LF>-	Empty
Delimiter	Empty	<CR><LF>:	Empty
Suppress Absent Element Delimiters	End of Type	End of Type	End of Type
Tag Data Separator	Empty	:	Empty
Length of Tag	Empty	Empty	Empty
Default CCSID	367	37	367
Trim on input	No Trim	Trim Both	No Trim
Truncate on output	Cleared	Cleared	Cleared
Escape Character	Chosen - empty	Chosen - empty	Chosen - empty
Quote character	Not chosen	Not chosen	Not chosen
Reserved Characters	Empty	Empty	Empty
Decimal Point	.	,	.
Packed decimal positive code	C	Not applicable	Not applicable
Strict Numeric Checking	Cleared	Selected	Selected
Derive sign from logical type	Selected	Not applicable	Not applicable
Default byte order	Big Endian	Not applicable	Not applicable
Default packed decimal byte order	Big Endian	Not applicable	Not applicable
Default float format	S390	Not applicable	Not applicable
Text boolean true value	1	1	Y
Text boolean false value	0	0	N
Text boolean null value	0	0	N
Binary boolean true value	00000001	Not applicable	Not applicable
Binary boolean false value	00000000	Not applicable	Not applicable
Binary boolean null value	00000000	Not applicable	Not applicable

Property	Messaging standard = User Defined Text	Messaging Standard = SWIFT	Messaging standard = ACORD AL3
Derive default dateTime format from logical type	Chosen	Chosen	Chosen
Use default DateTime Format[1]	Not chosen, but yyyy-MM-dd'T'HH:mm:ssZZZ if chosen	Not chosen, but yyyy-MM-dd'T'HH:mm:ssZZZ if chosen	Not chosen, but yyyy-MM-dd'T'HH:mm:ssZZZ if chosen
Start of century for 2 digit years	53	80	53
Days in First Week of Year	Use Broker Locale	Use Broker Locale	Use Broker Locale
First Day of Week	Use Broker Locale	Use Broker Locale	Use Broker Locale
Strict Datetime Checking	Selected	Selected	Selected
Time Zone	Use Broker Locale	Use Broker Locale	Use Broker Locale
Daylight Saving Time	Cleared	Cleared	Cleared
Use input UTC format on output	Cleared	Cleared	Cleared
Output policy for missing elements	UseDefaultValue	UseDefaultValue	UseDefaultValue
Derive default length from logical type	Selected	Selected	Selected

Default message set property values for TDS (part 2 of 3)

Property	Messaging standard = EDIFACT	Messaging Standard = X12	Messaging standard = TLOG
Group Indicator	Empty	Empty	Empty
Group Terminator	<EDIFACT_GROUP_TERM>	<X12_GROUP_TERM>	Empty
Delimiter	<EDIFACT_CS>	<X12_CS>	:
Suppress Absent Element Delimiters	End of Type	End of Type	End of Type
Tag Data Separator	<EDIFACT_TAGDATA_SEP>	Empty	Empty
Length of Tag	Empty	Empty	Empty
Default CCSID	367	367	367
Trim on input	Trim Both	Trim Both	No Trim
Truncate on output	Cleared	Cleared	Cleared
Escape Character	Chosen - <EDIFACT_ESC_CHAR>	Chosen - empty	Chosen - empty
Quote character	Not chosen	Not chosen	Not chosen
Reserved Characters	<EDIFACT_ESC_CHAR> <EDIFACT_TAGDATA_SEP> <EDIFACT_GROUP_TERM> <EDIFACT_CS>	Empty	Empty
Decimal Point	<EDIFACT_DEC_NOTATION>	.	.

Property	Messaging standard = EDIFACT	Messaging Standard = X12	Messaging standard = TLOG
Packed decimal positive code	Not applicable	Not applicable	Not applicable
Strict Numeric Checking	Selected	Selected	Cleared
Derive sign from logical type	Not applicable	Not applicable	Not applicable
Default byte order	Not applicable	Not applicable	Not applicable
Default packed decimal byte order	Not applicable	Not applicable	Not applicable
Default float format	Not applicable	Not applicable	Not applicable
Text boolean true value	1	1	1
Text boolean false value	0	0	0
Text boolean null value	0	0	0
Binary boolean true value	Not applicable	Not applicable	Not applicable
Binary boolean false value	Not applicable	Not applicable	Not applicable
Binary boolean null value	Not applicable	Not applicable	Not applicable
Derive default dateTime format from logical type	Chosen	Chosen	Chosen
Use default DateTime Format[1]	Not chosen, but yyyy-MM-dd'T'HH:mm:ssZZZ if chosen	Not chosen, but yyyy-MM-dd'T'HH:mm:ssZZZ if chosen	Not chosen, but yyyy-MM-dd'T'HH:mm:ssZZZ if chosen
Start of century for 2 digit years	53	53	53
Days in First Week of Year	Use Broker Locale	Use Broker Locale	Use Broker Locale
First Day of Week	Use Broker Locale	Use Broker Locale	Use Broker Locale
Strict Datetime Checking	Selected	Selected	Selected
Time Zone	Use Broker Locale	Use Broker Locale	Use Broker Locale
Daylight Saving Time	Cleared	Cleared	Cleared
Use input UTC format on output	Cleared	Cleared	Cleared
Output policy for missing elements	UseDefaultValue	UseDefaultValue	UseDefaultValue
Derive default length from logical type	Selected	Selected	Selected

Default message set property values for TDS (part 3 of 3)

Property	Messaging standard = HL7	Messaging Standard = CSV	Messaging standard = User Defined Mixed
Group Indicator	Empty	Empty	Empty
Group Terminator	<CR>	Empty	Empty
Delimiter	<HL7_FS>	,	Empty
Suppress Absent Element Delimiters	End of Type	Never	End of Type
Tag Data Separator	<HL7_FS>	Empty	Empty
Length of Tag	Empty	Empty	Empty
Default CCSID	367	367	850
Trim on input	No Trim	No Trim	Trim Padding Chars
Truncate on output	Cleared	Cleared	Cleared
Escape Character	Chosen - empty	Not chosen	Chosen - empty
Quote character	Not chosen	Chosen - "	Not chosen
Reserved Characters	Empty	, <CR> <LF> "	Empty
Decimal Point	.	.	.
Packed decimal positive code	Not applicable	C	C
Strict Numeric Checking	Cleared	Cleared	Cleared
Derive sign from logical type	Not applicable	Selected	Selected
Default byte order	Not applicable	Big Endian	Big Endian
Default packed decimal byte order	Not applicable	Big Endian	Big Endian
Default float format	Not applicable	S390	S390
Text boolean true value	1	1	1
Text boolean false value	0	0	0
Text boolean null value	0	0	0
Binary boolean true value	Not applicable	00000001	00000001
Binary boolean false value	Not applicable	00000000	00000000
Binary boolean null value	Not applicable	00000000	00000000
Derive default dateTime format from logical type	Not chosen	Chosen	Chosen

Property	Messaging standard = HL7	Messaging Standard = CSV	Messaging standard = User Defined Mixed
Use default DateTime Format[1]	Chosen - yyyy-MM-dd'T'HH:mm:ssZZZ	Not chosen - but yyyy-MM-dd'T'HH:mm:ssZZZ if chosen	Not chosen - but yyyy-MM-dd'T'HH:mm:ssZZZ if chosen
Start of century for 2 digit years	53	53	53
Days in First Week of Year	Use Broker Locale	Use Broker Locale	Use Broker Locale
First Day of Week	Use Broker Locale	Use Broker Locale	Use Broker Locale
Strict Datetime Checking	Selected	Selected	Selected
Time Zone	Use Broker Locale	Use Broker Locale	Use Broker Locale
Daylight Saving Time	Cleared	Cleared	Cleared
Use input UTC format on output	Cleared	Cleared	Cleared
Output policy for missing elements	UseDefaultValue	UseDefaultValue	UseDefaultValue
Derive default length from logical type	Selected	Selected	Selected

Default complex type/group property values for TDS (part 1 of 3)

Property	Messaging standard = User Defined Text	Messaging standard = SWIFT	Messaging standard = ACORD AL3
Data Element Separation	Fixed Length	Tagged Delimited	Fixed Length AL3
Group Indicator	Empty	<CR><LF>:	Empty
Group Terminator	Empty	<CR><LF>-	Empty
Delimiter	Empty	<CR><LF>:	not applicable
Suppress Absent Element Delimiters	End of Type	End of Type	End of Type
Observe Element Length	Selected	Cleared	Selected
Tag Data Separator	Empty	:	Empty
Length of Tag	Empty	Empty	Empty
Length of Encoded Length	not applicable	not applicable	not applicable
Extra Chars in Encoded Length	not applicable	not applicable	not applicable

Default complex type/group property values for TDS (part 2 of 3)

Property	Messaging standard = EDIFACT	Messaging standard = X12	Messaging standard = TLOG
Data Element Separation	All Elements Delimited	All Elements Delimited	Fixed length

Property	Messaging standard = EDIFACT	Messaging standard = X12	Messaging standard = TLOG
Group Indicator	Empty	Empty	Empty
Group Terminator	<EDIFACT_GROUP_TERM>	<X12_GROUP_TERM>	Empty
Delimiter	<EDIFACT_CS>	<X12_CS>	:
Suppress Absent Element Delimiters	End of Type	End of Type	End of Type
Observe Element Length	Cleared	Cleared	Cleared
Tag Data Separator	<EDIFACT_TAGDATA_SEP>	Empty	Empty
Length of Tag	Empty	Empty	Empty
Length of Encoded Length	not applicable	not applicable	not applicable
Extra Chars in Encoded Length	not applicable	not applicable	not applicable

Default complex type/group property values for TDS (part 3 of 3)

Property	Messaging standard = HL7	Messaging standard = CSV	Messaging standard = User Defined Mixed
Data Element Separation	All Elements Delimited	All Elements Delimited	Fixed Length
Group Indicator	Empty	Empty	Empty
Group Terminator	<CR>	Empty	Empty
Delimiter	<HL7_FS>	,	Empty
Suppress Absent Element Delimiters	End of Type	Never	End of Type
Observe Element Length	Cleared	Cleared	Selected
Tag Data Separator	<HL7_FS>	not applicable	Empty
Length of Tag	Empty	not applicable	Empty
Length of Encoded Length	not applicable	not applicable	not applicable
Extra Chars in Encoded Length	not applicable	not applicable	not applicable

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM TDS format" on page 1221
The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

Related reference:

"Message set properties" on page 5371
Message sets have properties that you can set to define their characteristics and the way in which they are processed.

"TDS Format message set properties" on page 5381
The following tables show the properties that you can set for a TDS format message set.

"TDS Mnemonics" on page 5391
The Tagged/Delimited String Format (TDS) uses mnemonics for a number of properties for a message set, complex type, or both.

XML Wire Format message set properties:

The following tables define the properties for the XML Wire Format for the message set.

Namespace settings

Property	Type	Meaning
Namespace URI	String	Enter the name of the namespace that you are using for the associated prefix.
Prefix	String	Enter the prefix to associate the element and attribute names that you use it with to the namespace name.

Namespace schema locations

Property	Type	Meaning
Namespace URI	String	Enter the namespace name that identifies which namespace you are using.
Schema location	String	Enter the location of the schema for the associated namespace name that is used to validate objects within the namespace.

XML declaration

Property	Type	Meaning
Suppress XML Declaration	Check box	Select the check box to suppress the XML declaration. If selected, the declaration (for example, `<?xml version='1.0'>`) is suppressed. By default, the check box is cleared.
XML Version	Enumerated type	This controls the value of the version in the generated XML declaration. The default is 1.0. If you set `Suppress XML Declaration` to Yes, this property is ignored.

Property	Type	Meaning
XML Encoding	Enumerated type	This controls whether an encoding attribute is written in the generated XML declaration. If Null is selected, no encoding attribute is written in the XML declaration of the output XML document. If As document text is selected, an encoding attribute is generated that is consistent with the text in the XML document. The default is Null. If the `Suppress XML Declaration` check box is selected, this property is ignored.
Standalone Document	Enumerated type	Select `Yes`, `No`, or `Null` from the list of values. If you select `Null`, no standalone declaration is present in the XML declaration. If you select `Yes` or `No`, the declaration `standalone = "yes"` or `standalone = "no"` is added to the XML declaration when the output message is written. The default value is `Null`. The setting of this property does not determine whether an external DTD subset is loaded; external DTD subsets are never loaded in this release. If the `Suppress XML Declaration` check box is selected, this property is ignored.
Output Namespace Declaration	Enumerated type	The *`Output Namespace Declaration`* property controls where the namespace declarations are placed in the output XML document. Select from: • `At start of document`. Declarations for all of the entries in the *`Namespace schema locations`* table above are output as attributes of the message in the output XML document. The disadvantage of this option is that, in some cases, unnecessary declarations might be output. • `As required`. Declarations are output only when required by an element or attribute that is in that namespace. The disadvantage of this option is that the same namespace declaration might need to be output more than once in the output XML document. The default option is `At start of document`. This property is active only if namespaces are enabled for this message set.

XML document type settings

Property	Type	Meaning
Suppress DOCTYPE	Check box	If you select the check box, the DOCTYPE (DTD) declaration is suppressed. By default, the check box is selected.
DOCTYPE System ID	String	Specify the System ID for DOCTYPE external DTD subset (if DOCTYPE is present). This is typically set to the name of the generated (or imported) DTD for a message set. If *`Suppress DOCTYPE`* is set, this property is ignored and cannot be changed (the field is disabled). The default value is `www.mrmnames.net/`, followed by the message set identifier.
DOCTYPE Public ID	String	Specify the Public ID for DOCTYPE external DTD subset (if DOCTYPE is present, and System ID is specified). If *`Suppress DOCTYPE`* is set, this property is ignored and cannot be changed (the field is disabled). The default value is the message set identifier.

Property	Type	Meaning
DOCTYPE Text	String	Use this property to add additional DTD declarations. It is not parsed by the XML parser and, therefore, it might not be valid XML. You can include ENTITY definitions or internal DTD declarations. It is a string (up to 32 KB) in which new line and tab characters are replaced by `\n` and `\t` respectively. The content is *not* parsed, and appears in the output message. If there is an in-line DTD, the content of this property takes precedence. If you have set *`Suppress DOCTYPE`*, this property is ignored and cannot be changed (the field is disabled). For more information, see "MRM XML: In-line DTDs and the DOCTYPE text property" on page 5407. The default value is empty (not set).

XML representation of Boolean values

Property	Type	Meaning
Boolean True Value	String	Specify the string that is used to encode and recognize BOOLEAN true values. When an XML document is parsed, the string `1` is always accepted as true for a BOOLEAN element. Enter a string of up to 254 characters. The default is `true`. `1` is also valid.
Boolean False Value	String	Specify the string that is used to encode and recognize BOOLEAN false values. When an XML document is parsed, the string `0` is always accepted as false for a BOOLEAN element. Enter a string of up to 254 characters. The default is `false`. `0` is also valid.

XML representation of null values

Property	Type	Meaning
Encoding Numeric Null	Enumerated type	Specify the null encoding for numeric XML elements. This provides a method of assigning a logical null meaning to such elements. You must select one of the following values from the list shown: • `NULLEmpty`. If the element value is the empty string, the element is null. This is the default value. • `NULLValue`. If the element value matches that provided by associated property *`Encoding Numeric Null Value`*, the element is null. • `NULLXMLSchema`. If the element contains an *`xsi:nil`* attribute that evaluates to true, the element is null. • `NULLValueAttribute`. This option is valid only for elements that have XML Wire Format property *`Render`* set to either `XMLElementAttrVal` or `XMLElementAttrIDVal`. See "XML Null handling options" on page 6258 for details. • `NULLAttribute` (deprecated). If the element contains an attribute with a name that matches that provided by associated property *`Encoding Numeric Null Value`*, and the attribute evaluates to true, the element is null. • `NULLElement` (deprecated). If the element contains a child element with a name that matches that provided by associated property *`Encoding Numeric Null Value`*, the element is null. See "XML Null handling options" on page 6258 for full details.

Property	Type	Meaning
Encoding Numeric Null Value	String	Specify the value to qualify the *Encoding Numeric Null* property, if you have set that to NULLValue, NULLAttribute, or NULLElement. Refer to "XML Null handling options" on page 6258 for further information.
Encoding Non-Numeric Null	Enumerated type	Specify the null encoding for non-numeric XML elements. This provides a method of assigning a logical null meaning to such elements. The options are identical to those available for property *Encoding Numeric Null*.
Encoding Non-Numeric Null Value	String	Specify the value to qualify the *Encoding Non-Numeric Null* property. Refer to "XML Null handling options" on page 6258 for further information.

DateTime settings

Property	Type	Meaning
Derive default dateTime format from logical type	Button	Select this option if you want the default dateTime format to be determined by the logical type of the element or attribute. You can override this property for an element or attribute within a complex type.
Use default dateTime format	Button and String	Select this option if you want to specify a default dateTime format that is fixed for all elements or attributes of logical type dateTime, date, time, gYear, gYearMonth, gMonth, gMonthDay, and gDay. You can override this property for an element or attribute within a complex type. For more information, see "DateTime formats" on page 6310.
Start of century for 2-digit years	Integer	This property determines how 2-digit years are interpreted. Specify the two digits that start a 100-year window that contains the current year. For example, if you specify 89, and the current year is 2002, all 2-digit dates are interpreted as being in the range 1989 - 2088.
Days in First Week of Year	Enumerated	Specify the number of days of the new year that must fall within the first week. The start of a year typically falls in the middle of a week. If the number of days in that week is less than the value specified here, the week is considered to be the last week of the previous year; therefore, week 1 starts some days into the new year. Otherwise, it is considered to be the first week of the new year; in this case, week 1 starts some days before the new year. Select Use Broker Locale, which causes the broker to get the information from the underlying platform, or select a number from the list that is displayed.
First Day Of Week	Enumerated	Specify the day on which each new week starts. Select Use Broker Locale, which causes the broker to get the information from the underlying platform, or select a value from the list that is displayed.

Property	Type	Meaning
Strict DateTime Checking	Check box	Select this option if you want to restrict dateTimes to a valid dateTime format. If `Strict DateTime Checking` is selected, receiving an incorrect dateTime causes an error. **Strict dateTime checking** Examples of strict dateTime checking are: • DateTimes are restricted to valid dateTimes only. When you use this option, a date such as the 35th March is not processed as 4th April, and 10:79 is not processed as 11:19. Receiving an out-of-band dateTime, such as these examples, causes an error to occur. • The number of characters for a numeric dateTime component must be within the bounds of the corresponding formatting symbols. Repeat the symbol to specify the minimum number of digits that you require. The maximum number of digits that is permitted becomes the upper bound for a particular symbol. For example, day in month has an upper bound of 31; therefore, a format string of 'd' allows the values 2 and 21 to be parsed, but does not allow the values 32 and 210. On output, numbers are padded with zeros to the specified length. A year is a special case; see the message set property `Start of century for 2 digit years`. For fractional seconds, the length must implicitly match the number of format symbols on input. The output is rounded to the specified length. • White space is not skipped over. The white space in the input string must correspond with the same number and position of white space in the formatting string. • If data remains that is not parsed in the input string after all the symbols in the formatting string have been matched, an error occurs. **Lenient dateTime checking** Examples of lenient dateTime checking are: • The parser converts out-of-band dateTime values to the appropriate in-band value. For example, a date of 2005-05-32 is converted to 2005-06-01. • Output of dateTimes always adheres to the symbol count. For example, a formatting string of yyyy-MM-dd (where '-' is the field separator) allows one or more characters to be parsed against MM and dd. Therefore, dates that are not valid - for example, 2005-1-123 and 2005-011-12 - can be entered. The first value of 2005-1-123 is output as the date 2005-05-03, and the second value of 2005-011-12 is output as the date 2005-11-12. • The number of the timezone formatting symbol Z is applicable only to the output dateTime format. • White space is skipped over.
Time Zone	Enumerated	The value that you set for this property is used if the value that you specified for the `Default DateTime Format` property does not include Time Zone information. The initial value is Use Broker Locale, which causes the broker to get the information from the underlying platform. You can change this property by selecting from the list of values.
Daylight Saving Time	Check box	Select this option if the area in the `Time Zone` property observes daylight saving time. If it does not observe daylight saving time, do not select this option. For example, if an area is selected in `Time Zone` and this option is not selected, the value passed represents the time zone without the daylight saving time.

Property	Type	Meaning
Use input UTC format on output	Check box	This property applies to elements and attributes of logical type xsd:dateTime or xsd:time that have a dateTime format of I, IU, T, or TU, or that include ZZZ or ZZZU. Such elements and attributes can specify Coordinated Universal Time (UTC) by using either the Z character or timezone +00:00 in the value. On input, the MRM parser remembers the way that UTC was specified. If this property is selected, and the element or attribute is copied to an output message, the UTC format is preserved into the output message and overrides the format that is implied by the dateTime format property. If this property is cleared, or the element or attribute was not copied from an input message, the UTC format in the output message is controlled solely by the dateTime format property.

xsi:type settings

Property	Type	Meaning
Output policy for xsi:type attributes	Enumerated type	When writing XML documents, use this property to specify the circumstances under which the xsi:type attribute of elements is produced as output. **Never** Do not produce xsi:type attributes for elements, even if xsi:type attributes appear in the message tree. **When present** Produce xsi:type attributes for elements only when xsi:type attributes appear in the message tree. This value is the default value. **Always (Simple elements only)** Ensure that all simple elements are produced with an xsi:type attribute. If a simple element already has an xsi:type attribute in the message tree, it is used; otherwise, an xsi:type attribute is generated by using the rules in the following table. **Always (All elements)** Ensure that all elements are produced with an xsi:type attribute if possible to do so. If an element already has an xsi:type attribute in the message tree, it is used; otherwise, an xsi:type attribute is generated by using the rules in the following table. **Follow SOAP Encoding rules** Follow the same behavior as for Always (Simple elements only). Additionally, produce a SOAP encoding-style attribute in the root tag of all messages.

If an xsi:type attribute needs to be produced as output, but does not appear in the message tree, the value is generated as described in the following table.

Element type	Value generated when element is defined in model	Value generated when element is self-defining
Simple type	If the type is global or is a built-in type, use it. If the type is local, use the global or built-in type from which it is derived.	Use the built-in type which best matches the data type of the element in the message tree.

Element type	Value generated when element is defined in model	Value generated when element is self-defining
Complex type with simple content	If the type is global use it. If the type is local, use the global or built-in type from which it is derived.	Use the built-in type which best matches the data type of the element in the message tree.
Complex type with complex content	If the type is global use it. If the type is local, no xsi:type attribute is produced.	No xsi:type attribute is produced.

Deprecated

Note: The following properties are used to control behavior of the MRM parser; they should not be changed from their default settings. These properties will be withdrawn in a future release.

Property	Type	Meaning
Root Tag Name	String	Specify the name of the message set root tag. You can leave this property blank, in which case no wrapper tags are used for messages (that is, the message tag is the root of the document). The name can be followed by a space and additional text for attribute/value pairs to appear with the root tag. The default value is blank.
Suppress Timestamp Comment	Check box	If selected, the timestamp comment string in the XML output is suppressed. If not selected, the comment is *not* suppressed, and a comment of the form `<!--MRM Generated XML Output on: Tue Apr 23 09:34:42 2002-->` is included in the output message. The default is for the check box to be selected.
Enable Versioning Support	Check box	If this is selected, versioning support is enabled. This property specifies whether XML namespace definitions are coded for the root tag in the message, together with namespace qualifiers for any elements that do not belong to the default namespace. These namespace definitions are used to represent the message set dependency information, which is used to support the exchange of messages between applications that are based on different customizations of the same message set. The default is for the check box to be selected, for compatibility with MRM XML messages in earlier releases. If you did *not* use MRM XML messages in earlier releases, you should ensure that this check box is not selected.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"MRM XML physical format: Handling xsi:type attributes" on page 1252
The prefix "xsi" is the namespace prefix used by convention for the XML Schema instance namespace. XML documents can contain elements that have an xsi:type attribute. This behavior provides an explicit data type for the element.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

Related reference:

"Message set properties" on page 5371
Message sets have properties that you can set to define their characteristics and the way in which they are processed.

"Custom Wire Format message set properties" on page 5375
The tables define the properties that you can set for a Custom Wire Format message set.

"MRM XML: In-line DTDs and the DOCTYPE text property"
You can include in-line DTDs in your messages, and you can specify additional information by setting the property *DOCTYPE Text*. The parser takes certain actions when constructing an output message.

"TDS Format message set properties" on page 5381
The following tables show the properties that you can set for a TDS format message set.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

MRM XML: In-line DTDs and the DOCTYPE text property:

You can include in-line DTDs in your messages, and you can specify additional information by setting the property *DOCTYPE Text*. The parser takes certain actions when constructing an output message.

1. If the output message has to be regenerated, for example if you configure a Compute node to create an output message by coding ESQL statements like `SET OutputRoot.MRM.Field1 = xxx`:
   - If you have set the property *Suppress DOCTYPE* for the message set in which you have defined this message to `Yes`, both DOCTYPE information (specified in the *DOCTYPE Text* property for the message set or message) and in-line DTD are excluded from the output message.
   - If you have set the property *Suppress DOCTYPE* for the message set in which you have defined this message to `No`.
     - The in-line DTD is preserved if possible.
     - Otherwise, if the message is self-defining, the message set *DOCTYPE Text* property information is included in the output message.
     - Otherwise (the message is *not* self-defining), the message level *DOCTYPE Text* property information is included in the output message.
2. If the output message does not have to be regenerated, the parser generates an output message that is a direct copy of the input message. This occurs if you have configured a Compute node in the message flow to copy the message using `SET OutputRoot = InputRoot` (explicitly, or by checking the *Copy entire message* check box), and you do not modify the message in any way in this or

any other node. In this case the in-line DTD is retained in the output message but any information that you specify in the *DOCTYPE Text* property for the message set or message is not included.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

Related reference:

"Message set properties" on page 5371
Message sets have properties that you can set to define their characteristics and the way in which they are processed.

"XML Wire Format message set properties" on page 5400
The following tables define the properties for the XML Wire Format for the message set.

Documentation properties for a message set:

Use the documentation property of a message set to add information that enhances the understanding of the function of the message set.

Property	Type	Meaning
Version	String	Use this field to enter a version for the message set. The version of the message set is then displayed in the Eclipse properties view. You can set a default value for this field in the message set preferences.
Documentation	String	You use the documentation property of a message set to add information that enhances the understanding of the function of the message set. This field requires a string value ; you can use any standard alphanumeric characters. You can also use this field to define a keyword and its value that is displayed for the deployed message set in the properties view of Eclipse. An example is: `$MQSI Author=Fred MQSI$` When the properties of the deployed message set are displayed, a row is added to the display showing "Author" as the property name and "Fred" as its value.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

"Message sets overview" on page 1162
A message set is a container for grouping messages and associated message resources (elements, types, groups).

"Version and keyword information for deployable objects" on page 1443
Use the Broker Archive editor to view the version and keyword information of deployable objects.

"Message flow version and keywords" on page 1445
When you are developing a message flow, you can define the version of the message flow as well as other key information that you want to be associated with it.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with a message set" on page 2840
Complete a variety of tasks that are involved in working with a message set.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message set properties" on page 5371
Message sets have properties that you can set to define their characteristics and the way in which they are processed.

"Message set preferences" on page 5366
Preferences for message sets.

"Guidance for defining keywords" on page 4017
You can add extra information to an object in the form of one or more keywords.

Message definition file properties

The properties of a message definition file.

Namespace

Property	Type	Meaning
Prefix	String	The namespace prefix for the target namespace of this file. This field cannot be changed after the message definition file has been created.
Target Namespace	String	The target namespace for the message definition file. All global objects created within the file will have this namespace by default. This field cannot be changed after the message definition file has been created.

Default namespaces for local objects

Property	Type	Meaning
Elements	String	The default namespace for all local elements within this message definition file.
Attributes	String	The default namespace for all local attributes within this message definition file.

Property	Type	Meaning
Default block	String and Enumerated type	The default value for the block attribute for all complex types and elements within this message definition file.

Property	Type	Meaning
Default final	String and Enumerated type	The default value for the final attribute for all complex types and elements within this message definition file.

Property	Type	Meaning
Use xml.xsd schema	Check box	Select this check box if you need to use the xml.xsd schema. When you select this check box, the http://www.w3.org/2001/xml.xsd schema is imported and you can use any of the constructs in that schema.

Note: The full text that describes this check box is *Use http://www.w3.org/2001/xml.xsd schema*.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"Namespaces in the message model" on page 1201
Use namespaces to qualify message model object names.

"Message definition files" on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

"Substitution groups in the message model" on page 1199
Substitution groups are an XML Schema feature that provides a way of substituting one element for another in an XML message.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with a message definition file" on page 2863
Create, open, and delete a message definition file.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

Message definition file includes properties:

The location of each message definition file that has been included in this message definition file is displayed.

Property	Type	Meaning
Schema Location	String	For each message definition file that has been included in this message definition file, this field displays its location. The location is displayed as a relative path from the message definition file to the included file.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"Namespaces in the message model" on page 1201
Use namespaces to qualify message model object names.

"Message definition files" on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

"Reusing message model files" on page 1209
One message definition file can reuse message model objects defined in another file.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with a message definition file" on page 2863
Create, open, and delete a message definition file.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Message definition file properties" on page 5409
The properties of a message definition file.

Message definition file imports properties:

The file imports properties of a message definition.

Property	Type	Meaning
Schema Location	String	For each message definition file that has been imported into this message definition file, this field displays its location. The location is displayed as a relative path from the message definition file to the imported file.
Prefix	String	Displays the namespace prefix for each imported message definition file.
Namespace	String	Displays the namespace URI for each imported message definition file.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"Namespaces in the message model" on page 1201
Use namespaces to qualify message model object names.

"Message definition files" on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

"Reusing message model files" on page 1209
One message definition file can reuse message model objects defined in another file.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with a message definition file" on page 2863
Create, open, and delete a message definition file.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Message definition file properties" on page 5409
The properties of a message definition file.

Message definition file redefines properties:

This provides details of the properties associated with message definition redefines.

Property	Type	Meaning
Schema Location	String	For each message definition file that has been redefined in this message definition file, this field displays its location. The location is displayed as a relative path from the message definition file to the included file.

Redefines are not supported, and result in a validation error. If you right-click the error message and select **Quick Fix**, you can convert the redefines construct into an include construct, which also removes the error message.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"Namespaces in the message model" on page 1201
Use namespaces to qualify message model object names.

"Message definition files" on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

"Reusing message model files" on page 1209
One message definition file can reuse message model objects defined in another file.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with a message definition file" on page 2863
Create, open, and delete a message definition file.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Message definition file properties" on page 5409
The properties of a message definition file.

Documentation properties for all message set objects:

Use the documentation property of an object to add information that describes the function of the object.

The documentation property is available on all objects except Key, Keyref, and Unique objects.

The property is a string field, and you can use any standard alphanumeric characters.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

Message category properties

A message category provides a way of grouping your messages.

The following table describes the properties that are associated with a message category:

Property	Type	Meaning
Category Kind	Enumerated type	This property describes the purpose of the message category. Choose from the following values: • `wsdl`. This value is the default. Choose this value if the message category is to participate in the generation of WSDL documents. When the WSDL document is generated, the name of the message category provides the name for the <wsdl:operation> element that is generated for eligible messages in the message category. **Note:** Message categories are no longer necessary for the generation of WSDL documents; they were necessary in Version 6.0. • `other`. This value ndicates that the category represents a generic grouping of messages as an aid to organizing them in your workspace.

Property	Type	Meaning
Category Usage	Enumerated type	Use this property to describe the operation type for a WSDL operation. Choose from the following values: • `wsdl:request-response`. This is the default if Category Kind is `wsdl`. • `wsdl:solicit-response`. • `wsdl:one-way`. • `wsdl:notification`. • `empty string`. This is the default if Category Kind is `other`.
Documentation	String	Use this property to add information to enhance the understanding of an object's function. This property is a string field; any standard alphanumeric characters can be used. If Category Kind is `wsdl`, the value of the field is included in any generated WSDL as the wsdl:documentation child of the operation element in the WSDL portType. If Category Kind is `other`, the value of the field merely documents the Message Category within your workspace.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

"Message categories" on page 1200
Message category files have the suffix `.category`. These files are optional. You can have many message category files in a message set.

Related tasks:

"Working with a message category file" on page 2923
This topic area lists the tasks that are involved when working with a message category file.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message category member properties"
This describes the properties that are associated with a message category member.

Message category member properties:

This describes the properties that are associated with a message category member.

Property	Type	Meaning
Role Name	String	If Category Kind is wsdl, the value of the property becomes the WSDL message part name and must be unique within the category. It always defaults to the message name. If Category Kind is other, the value of the property has no particular significance.

Property	Type	Meaning
Role Type	Enumerated type	This property determines the role that the message plays in the message category. Select from: • `wsdl:input` • `wsdl:output` • `wsdl:return` • `wsdl:fault` • `empty string` If Category Kind is wsdl, the default value is wsdl:input. This property dictates the role within a WSDL operation. The value wsdl:return implies wsdl:output, but for rpc-style WSDL generation it also identifies the message part that is used as the return value and in this instance can be omitted from the parameterOrder attribute. No more than one message can have Role Type of wsdl:return. If Category Kind is other, the value defaults to an empty string and this property has no role in the message category.
Role Usage	Enumerated type	This property determines the role that the message plays in the SOAP binding. Select from: • `soap:body` • `soap:header` • `soap:fault` • `soap:headerfault` • `empty string` If Category Kind is wsdl, this property defaults to soap:body and dictates the SOAP-binding child of the WSDL input, output, or fault element. If Category Kind is other, this property is deactivated.
Documentation	String	This is a string property; any standard alphanumeric characters can be used. If Category Kind is wsdl, the value of the property is included in any generated WSDL as the wsdl:documentation child of the WSDL input, output, or fault element under the WSDL portType. If Category Kind is other, the value merely documents the Message Category within your workspace.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

"Message categories" on page 1200
Message category files have the suffix `.category`. These files are optional. You can have many message category files in a message set.

Related tasks:

"Working with a message category file" on page 2923
This topic area lists the tasks that are involved when working with a message category file.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message category properties" on page 5413
A message category provides a way of grouping your messages.

Message model object properties

Access property information by property kind, or by object.

There are two ways of accessing the reference information for the properties of message model objects. The following topics allow you to access the property information by property kind:

- "Logical properties for message model objects"
- "Physical properties for message model objects" on page 5455
- "Documentation properties for all message set objects" on page 5413

Alternatively, you can access the property information by object, starting from:

- "Message model object properties by object" on page 5536

Deprecated objects are dealt with separately. For further information, see "Deprecated message model object properties" on page 6069

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Logical properties for message model objects"
Logical property information is available for certain objects.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Logical properties for message model objects:

Logical property information is available for certain objects.

- "Attribute group reference logical properties" on page 5417
- "Attribute reference logical properties" on page 5418
- "Complex type logical properties" on page 5419

- "Element reference logical properties" on page 5423
- "Global attribute logical properties" on page 5425
- "Global attribute group logical properties" on page 5429
- "Global element logical properties" on page 5430
- "Global group logical properties" on page 5433
- "Group reference logical properties" on page 5436
- "Key logical properties" on page 5437
- "Keyref logical properties" on page 5437
- "Local element logical properties" on page 5442
- "Local group logical properties" on page 5446
- "Message logical properties" on page 5449
- "Simple type logical properties" on page 5450
- "Unique logical properties" on page 5452
- "Wildcard attribute logical properties" on page 5453
- "Wildcard element logical properties" on page 5453

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

"Message definition files" on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Attribute group reference logical properties:

The logical properties of an attribute group reference.

Property	Type	Meaning
Reference Name	Enumerated type	The `Reference Name` is the name of the object that this object is referring to. The objects available to reference can be selected from the list.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Attribute reference logical properties:

The logical properties of an attribute reference.

Property	Type	Meaning
Reference Name	Enumerated type	The `Reference Name` is the name of the object that this object is referring to. The objects available to reference can be selected from the list.

Property	Type	Meaning
Usage	Enumerated type	Use this property with the `Value` property found in an attribute object. The default value for the `Usage` property is optional. Select from the following options: • optional. – If the `Value` property is set to default, and no data has been entered in the `Value` property, the attribute can appear once and can have any value. – If the `Value` property is set to default, the attribute can appear once. If it does not appear, its value is the data that has been entered in the `Value` property. If it does appear, it is the value given. – Where the `Value` property is set to fixed, the attribute can appear once. If it does appear, its value must match the data that has been entered in the `Value` property. If it does not appear, its value is the data that has been entered in the `Value` property. • prohibited. The attribute must not appear. • required. – If the `Value` property is set to default, and no data has been entered in the `Value` property, the attribute must appear once and can have any value. – If the `Value` property is set to fixed, the attribute must appear once, and it must match the data that has been entered in the `Value` property.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Complex type logical properties:

The logical properties of a complex type include properties that describe content and substitution settings.

Property	Type	Meaning
Name	String	Specify a name for the object when you create it. Names can consist of alphanumeric characters, including the letters *A* through *Z*, *a* through *z*, and the digits *0* through *9*. They might also include the following punctuation characters; • - the hyphen • _ the underscore • . the period Names can start only with a letter or the underscore character, and not with a number, hyphen, or period. Names that begin with `xml`, or any variant of these characters (for example `XmL`), are reserved by the XML standards specification. Further details of naming conventions and allowable characters can be found in the Extensible Markup Language (XML) specification that can be found on the World Wide Web Consortium (W3C) Web site.
Base Type	Enumerated type	You can use this property to select a type (simple or complex) that is used as the starting point to define a new complex type that is derived by restriction or extension.
Derived By	Enumerated type	If this property is active, select one of the following options: • restriction. If a complex type is derived by restriction, the content model of the complex type is a subset of the base type. • extension. If the complex type is derived by extension, the content model of the complex type is the content model of the base type plus the content model specified in the type derivation. Derivation by list or union is not supported.

Content

The following table shows the valid settings for `Composition` and `Content Validation`. These properties are located on the group which defines the content of this type. They can be edited only if the `Local group` button is selected. If the `Global group` button is selected, these properties are taken from the global group identified by the `Group name` field.

Valid children in a complex type that depend on both `Composition` and `Content Validation` are shown in "MRM content validation" on page 5422.

Property	Type	Meaning
Local Group	Button	Select this property if the content of your complex type is a local group.

Property	Type	Meaning
Composition	Enumerated type	Define the order, and the number of occurrences, of the elements and groups in your messages. Composition does not affect the attributes in a complex type. Select one of the following options: • Empty • sequence. If you select this option, you can define members that are elements or groups. These members, if present, must appear in the specified order in the message. They can repeat, and the same element or group can appear more than once. • choice. If you select this option, you can define members that are elements or groups. Exactly one of the defined members must be present in the message, and can repeat. Use this option if you want to model C unions and COBOL REDEFINES in a Custom Wire Format, or an XML DTD element that uses choice in an XML Wire Format, or a SWIFT field that has more than one option. • all. If you select this option, you can define members that are elements; groups are not allowed. The elements in an all group can appear in any order. Each element can appear once, or not at all. An all group can be used only at the top level of a complex type; it cannot be a member of another group within a type. • unorderedSet. This option is supported only by the MRM domain.If you select this option, you can define members that are elements. The elements can repeat, but the same element cannot appear twice in the list of members. The elements can appear in any order in the message. • orderedSet. This option is supported only by the MRM domain.If you select this option, you can define members that are elements. The elements can repeat, but the same element cannot appear twice in the list of members. The elements must appear in the specified order in the message. • message. This option is supported only by the MRM domain.If you select this option, you can define only messages as members. Each member can repeat, but the same message cannot appear twice in the list of members. Like choice, only one of the defined members can be present in a message. Unlike choice, when writing a message, if the complex type or group has more than one member, the bit stream is not padded to the length of the longest member. Use this option to model multipart messages, which are used in some industry standards; for example, SWIFT. For more information, see the section on multipart messages in “Multipart messages” on page 1191.

Property	Type	Meaning
Content Validation	Enumerated type	`Content Validation` is used only by the MRM domain. If validation is enabled in your message flow, `Content Validation` specifies the strictness of the MRM validation for members of a complex type or group. See "MRM content validation" on page 5422 for further details. Select from the following options: • Closed. The complex type can only contain the child elements that you have added to it. • Open Defined. The complex type can contain any valid element defined within the message set. • Open. The complex type can contain any valid element, not just those that you have added to this complex type. See "Combinations of Composition and Content Validation" on page 5612 for further details of these options.
Group Reference	Button	Select this option if the content of your complex type is a reference to a group object
Group Name	Enumerated type	The `Group Name` is the name of the group that this complex type refers to. The groups available to be referenced can be selected from the drop down list.
Min Occurs	Integer	Specify the minimum number of times that the object can repeat. The default value is 1. If the value is set to 0, the object is optional. With the exception of `Max Occurs` being set to -1, if a value is set for `Min Occurs`, it must be less than or equal to the value in `Max Occurs`.
Max Occurs	Integer	Specify the maximum number of times that the object can repeat. The default value is 1. If this property is not set, the object cannot occur more than once. If this property is set to 0, it is interpreted as if the object does not exist in the message. It can also be set to -1, to indicate that the limit is unbounded and there is no maximum to the number of occurrences.
Mixed	Check box	Select this option when the complex type has mixed content, and contains character data and sub-elements.

Substitution settings

Property	Type	Meaning
Final	Multiple selection enumerated type	The final attribute on a complex type controls whether other types can be derived from it. Valid values are extension/restriction/all. You can select from one or more of the following: • Empty • `restriction`. Prohibit type substitution by elements whose types are restrictions of the head element's type. • `extension`. Prohibit type substitution by elements whose types are extensions of the head element's type. • `#all`. Prohibit substitution by any method. To select more than one, you must type the selection into the property field.

Property	Type	Meaning
Block	Multiple selection enumerated type	The block attribute on a complex type restricts the types of substitutions which are allowed for elements based on that type. In the WebSphere Message Broker its effect is the same as if the block attribute were copied from the complex type onto every element based on the complex type. You can select from one or more of the following: • Empty • `restriction`. Prohibit type substitution by elements whose types are restrictions of the head element's type. • `extension`. Prohibit type substitution by elements whose types are extensions of the head element's type. • `#all`. Prohibit substitution by any method. To select more than one, you must type the selection into the property field.
Abstract	Check box	If selected, no elements based on this type can appear in the message.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

MRM content validation:

`Content Validation` is applied when the domain is MRM and validation is enabled. The `Content Validation` property specifies how strictly the MRM parser validates the members of a complex type or group.

The first of the following two tables shows the valid settings for `Content Validation` if `Composition` is set to `Message`. The second table shows the valid settings for `Content Validation` if `Composition` is not set to `Message`.

Content Validation options if Composition is set to Message

Option	Meaning
Open	When a message is parsed, this complex type or group can contain any message, not just those that you have defined in this message set. You can use this option for sparse messages (see "Self-defining elements and messages" on page 1198 for a definition of sparse messages).
Closed	When a message is parsed, this complex type or group can only contain the messages that are members of this complex type or group. This is always the case for messages represented in CWF format.
Open Defined	When a message is parsed, this complex type or group can contain any message defined within the message set.

Content Validation options if Composition is not set to Message

Option	Meaning
Open	When a message is parsed, this complex type or group can contain any elements and not just those that you have defined in this message set (see "Self-defining elements and messages" on page 1198 for a definition of sparse messages).
Closed	When a message is parsed, this complex type or group can only contain the elements that are members of this complex type or group.
Open Defined	When a message is parsed, this complex type or group can contain any element that you have defined within the message set.

When you are using `Content Validation` set to `open` or `open defined`, you cannot specify the precise position where the content that is not modeled is permitted to occur. If you want to do this, you should consider using a wildcard element as an alternative. Wildcard elements can appear only within a complex type or group with `Composition` of `sequence` and `Content Validation` of `closed`.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Complex type logical properties" on page 5419
The logical properties of a complex type include properties that describe content and substitution settings.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Element reference logical properties:

The logical properties of an element reference include properties that specify the number of occurrences of the element reference.

Property	Type	Meaning
Reference Name	Enumerated type	The `Reference Name` is the name of the object that this object is referring to. The objects available to reference can be selected from the list.

Occurrences

Property	Type	Meaning
Min Occurs	Integer	Specify the minimum number of times that the object can repeat. The default value is 1. If the value is set to 0, the object is optional. With the exception of `Max Occurs` being set to -1, if a value is set for `Min Occurs`, it must be less than or equal to the value in `Max Occurs`.

Property	Type	Meaning
Max Occurs	Integer	Specify the maximum number of times that the object can repeat. The default value is 1. If this property is not set, the object cannot occur more than once. If this property is set to 0, it is interpreted as if the object does not exist in the message. It can also be set to -1, to indicate that the limit is unbounded and there is no maximum to the number of occurrences.

The Min Occurs and Max Occurs properties are used in conjunction with an element's Value properties. The following table summarizes how an element reference can be constrained.

Min Occurs	Max Occurs	Fixed	Default	Notes
1	1			The element must appear once, and can have any value.
1	1	Delta		The element must appear once, and it must match the data that has been entered in the `Value` property. In this example, the element must contain the text Delta.
2	-1	Delta		The element must appear twice or more, and it must match the data that has been entered in the `Value` property. In this example, at least two elements must contain the text Delta.
0	1			The element is optional, can appear once, and can have any value.
0	1	Delta		The element is optional, and can appear once. If it does appear, its value must match the data that has been entered in the `Value` property. If it does not appear, its value is the data that has been entered in the `Value` property.
0	1		Delta	The element is optional, and can appear once. If it does not appear, its value is the data that has been entered in the `Value` property. If it does appear, it must be the value given in the element.
0	2		Delta	The element is optional and can appear once, twice, or not at all. If the element does not appear, it is not provided. If the element appears and it is empty, it set to the data held in the `Value` property, else it is the value given in the element.
0	0			The element is prohibited, and must not appear.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Global attribute logical properties:

The logical properties of a global attribute.

Property	Type	Meaning
Name	String	Specify a name for the object when you create it. Names can consist of alphanumeric characters, including the letters *A* through *Z*, *a* through *z*, and the digits *0* through *9*. They might also include the following punctuation characters; • - the hyphen • _ the underscore • . the period Names can start only with a letter or the underscore character, and not with a number, hyphen, or period. Names that begin with `xml`, or any variant of these characters (for example `XmL`), are reserved by the XML standards specification. Further details of naming conventions and allowable characters can be found in the Extensible Markup Language (XML) specification that can be found on the World Wide Web Consortium (W3C) Web site.

Property	Type	Meaning
Type	Enumerated type	The Type property constrains the type of data that can be present in the object. Select from the following options: • `int` • `string` • `Boolean` • `hexBinary` • `dateTime` • `date` • `time` • `decimal` • `float` • `(More...)` • `(New Simple Type)` • `(New Complex Type)` If you select (More...), the Type Selection wizard starts. In this wizard, you can select any of the available types. If you select (New Simple Type), the New Simple Type wizard starts. In this wizard, you can create an Anonymous simple type that is based on an existing type. This type can be created locally or globally. If you select (New Complex Type), the New Complex Type wizard starts. In this wizard, you can create an Anonymous complex type, which can be derived from an existing base type. This type can be created locally or globally. For further information about these types, and examples of their use, see the XML Schema Part 0: Primer. This document is available on the World Wide Web Consortium (W3C) Web site.
Namespace	Enumerated type	Namespaces are a simple method for qualifying element and attribute names by associating them with namespaces identified by URI references. If <no target namespace> is displayed, a namespace has not been set for this object. If the property is inactive, the message set has not been configured to support namespaces. Where the property is active, namespaces that are available for selection are displayed in the drop-down list.

Value

The *Value* properties are used with the *Usage* property in an Attribute Reference or a Local Attribute.

Property	Type	Meaning
Default	Button and String	This property provides the default value for an element or attribute. **XMLNSC domain** When parsing with validation enabled, default values are applied to missing attributes and empty elements as required by the XML Schema 1.0 specification. When writing, elements or attributes that are missing from the message tree are not automatically added to the output XML bit stream, even if they have default values. If missing elements or attributes are required, the message tree can be serialized and then re-parsed with validation enabled. **MRM (CWF and TDS physical formats)** When writing a fixed-length portion of a message (CWF or fixed-length TDS), if an attribute or element is missing from the message tree, the default value is inserted into the bit stream so that the message structure is preserved. **MRM (XML physical format)** No support for default values **Other domains** No support for default values.
Fixed	Button and String	This property provides the fixed value for an element or attribute. **XMLNSC domain** When parsing with validation enabled, if an attribute or element is present, the value is validated against the fixed value. If the values are not equal, a validation error is signalled. Also, when parsing with validation enabled, fixed values are applied to missing attributes and empty elements as required by the XML Schema 1.0 specification. When writing, elements or attributes that are missing from the message tree are not automatically added to the output XML bit stream, even if a fixed value has been specified. If missing elements or attributes are required, the message tree can be serialized and then re-parsed with validation enabled. **MRM (CWF and TDS physical formats)** When writing a fixed-length portion of a message (CWF or fixed-length TDS), if an attribute or element is missing from the message tree, the fixed value is inserted into the bit stream so that the message structure is preserved. **MRM (XML physical format)** No support for fixed values **Other domains** No support for fixed values.

Property	Type	Meaning
Interpret Value As	Enumerated type	Specify if values stored within this object must be interpreted as having significance for the parser and, if so, the type of interpretation that must occur. Select one of the following options: • None This value is the default value, and indicates that the element or attribute does not have a key value associated with it. • MessageSetIdentity. Specifies that the value of the element or attribute corresponds to the identifier, name, or alias (in that priority order) that is associated with the message set where all subsequent embedded messages that are descendents of the enclosing message are defined. This value remains in force unless a new element or attribute MessageSetIdentity field is encountered which resets the MessageSetIdentity value. • MessageIdentity. Specifies that the value of the element or attribute corresponds to the name or alias (in that priority order) that is associated with a message, and acts as an identifier for subsequent embedded messages which are the immediate children of the enclosing message. This identity applies until a new element or attribute MessageIdentity field is encountered at the same level in the tree. The embedded message can be defined in either the current message set, or in a message set identified by using a MessageSetIdentity. **Note:** This property is applicable only when the type of the object is derived from xsd:string.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Global attribute group logical properties:

The logical properties of an attribute group.

Property	Type	Meaning
Name	String	Specify a name for the object when you create it. Names can consist of alphanumeric characters, including the letters *A* through *Z*, *a* through *z*, and the digits *0* through *9*. They might also include the following punctuation characters; • - the hyphen • _ the underscore • . the period Names can start only with a letter or the underscore character, and not with a number, hyphen, or period. Names that begin with `xml`, or any variant of these characters (for example `XmL`), are reserved by the XML standards specification. Further details of naming conventions and allowable characters can be found in the Extensible Markup Language (XML) specification that can be found on the World Wide Web Consortium (W3C) Web site.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Global element logical properties:

The logical properties of a global element.

Property	Type	Meaning
Name	String	Specify a name for the object when you create it. Names can consist of alphanumeric characters, including the letters *A* through *Z*, *a* through *z*, and the digits *0* through *9*. They might also include the following punctuation characters; • `-` the hyphen • `_` the underscore • `.` the period Names can start only with a letter or the underscore character, and not with a number, hyphen, or period. Names that begin with `xml`, or any variant of these characters (for example `XmL`), are reserved by the XML standards specification. Further details of naming conventions and allowable characters can be found in the Extensible Markup Language (XML) specification that can be found on the World Wide Web Consortium (W3C) Web site.
Type	Enumerated type	The Type property constrains the type of data that can be present in the object. Select from the following options: • `int` • `string` • `Boolean` • `hexBinary` • `dateTime` • `date` • `time` • `decimal` • `float` • `(More...)` • `(New Simple Type)` • `(New Complex Type)` If you select (More...), the Type Selection wizard starts. In this wizard, you can select any of the available types. If you select (New Simple Type), the New Simple Type wizard starts. In this wizard, you can create an Anonymous simple type that is based on an existing type. This type can be created locally or globally. If you select (New Complex Type), the New Complex Type wizard starts. In this wizard, you can create an Anonymous complex type, which can be derived from an existing base type. This type can be created locally or globally. For further information about these types, and examples of their use, see the XML Schema Part 0: Primer. This document is available on the World Wide Web Consortium (W3C) Web site.

Property	Type	Meaning
Namespace	Enumerated type	Namespaces are a simple method for qualifying element and attribute names by associating them with namespaces identified by URI references. If <no target namespace> is displayed, a namespace has not been set for this object. If the property is inactive, the message set has not been configured to support namespaces. Where the property is active, namespaces that are available for selection are displayed in the drop-down list.

Value

Property	Type	Meaning
Default	Button and String	This property provides the default value for an element or attribute. **XMLNSC domain** When parsing with validation enabled, default values are applied to missing attributes and empty elements as required by the XML Schema 1.0 specification. When writing, elements or attributes that are missing from the message tree are not automatically added to the output XML bit stream, even if they have default values. If missing elements or attributes are required, the message tree can be serialized and then re-parsed with validation enabled. **MRM (CWF and TDS physical formats)** When writing a fixed-length portion of a message (CWF or fixed-length TDS), if an attribute or element is missing from the message tree, the default value is inserted into the bit stream so that the message structure is preserved. **MRM (XML physical format)** No support for default values **Other domains** No support for default values.

Property	Type	Meaning
Fixed	Button and String	This property provides the fixed value for an element or attribute. **XMLNSC domain** When parsing with validation enabled, if an attribute or element is present, the value is validated against the fixed value. If the values are not equal, a validation error is signalled. Also, when parsing with validation enabled, fixed values are applied to missing attributes and empty elements as required by the XML Schema 1.0 specification. When writing, elements or attributes that are missing from the message tree are not automatically added to the output XML bit stream, even if a fixed value has been specified. If missing elements or attributes are required, the message tree can be serialized and then re-parsed with validation enabled. **MRM (CWF and TDS physical formats)** When writing a fixed-length portion of a message (CWF or fixed-length TDS), if an attribute or element is missing from the message tree, the fixed value is inserted into the bit stream so that the message structure is preserved. **MRM (XML physical format)** No support for fixed values **Other domains** No support for fixed values.
Nillable	Check box	Select this option if you want the element to be able to be defined as null. A null value is distinct from being empty, when the element contains no data.
Interpret Value As	Enumerated type	Specify if values stored within this object must be interpreted as having significance for the parser and, if so, the type of interpretation that must occur. Select one of the following options: • None This value is the default value, and indicates that the element or attribute does not have a key value associated with it. • MessageSetIdentity. Specifies that the value of the element or attribute corresponds to the identifier, name, or alias (in that priority order) that is associated with the message set where all subsequent embedded messages that are descendents of the enclosing message are defined. This value remains in force unless a new element or attribute MessageSetIdentity field is encountered which resets the MessageSetIdentity value. • MessageIdentity. Specifies that the value of the element or attribute corresponds to the name or alias (in that priority order) that is associated with a message, and acts as an identifier for subsequent embedded messages which are the immediate children of the enclosing message. This identity applies until a new element or attribute MessageIdentity field is encountered at the same level in the tree. The embedded message can be defined in either the current message set, or in a message set identified by using a MessageSetIdentity. **Note:** This property is applicable only when the type of the object is derived from xsd:string.

Substitution settings

Substitution Groups provide a means by which one element may be substituted for another in a message. The element which can be substituted is called the 'head' element, and the substitution group is the list of elements that may be used in its place. An element can be in at most one substitution group.

Property	Type	Meaning
Final	Enumerated type	Limit the set of elements that can belong to its substitution group. • Empty • restriction. Prohibit element substitution by elements whose types are restrictions of the type of the head element. • extension. Prohibit element substitution by elements whose types are extensions of the type of the head element. • #all. Prohibit substitution by all methods.
Block	Enumerated type	Limit the set of elements that can be substituted for this element in a message. • Empty • restriction. Prohibit element substitution by elements whose types are restrictions of the type of the head element • extension. Prohibit element substitution by elements whose types are extensions of the type of the head element • substitution. Prohibit element substitution by members of the element's substitution group. • #all. Prohibit substitution by all methods.
Substitution Group	Enumerated type	Specify the name of a 'head' element. Setting this property indicates that this element is a member of the substitution group for the head element.
Abstract	Check box	Select this option if you do not want the element to appear in the message, but require one of the members of its substitution group to appear in its place.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Global group logical properties:

The logical properties of a global group.

Valid children in a global group that depend on both **Composition** and **Content Validation** are shown in "MRM content validation" on page 5422.

Property	Type	Meaning
Name	String	Specify a name for the object when you create it. Names can consist of alphanumeric characters, including the letters *A* through *Z*, *a* through *z*, and the digits *0* through *9*. They might also include the following punctuation characters; • - the hyphen • _ the underscore • . the period Names can start only with a letter or the underscore character, and not with a number, hyphen, or period. Names that begin with `xml`, or any variant of these characters (for example `XmL`), are reserved by the XML standards specification. Further details of naming conventions and allowable characters can be found in the Extensible Markup Language (XML) specification that can be found on the World Wide Web Consortium (W3C) Web site.

Property	Type	Meaning
Composition	Enumerated type	Define the order, and the number of occurrences, of the elements and groups in your messages. Composition does not affect the attributes in a complex type. Select from the following options: • Empty • sequence. If you select this option, you can define members that are elements or groups. These members, if present, must appear in the specified order in the message. They can repeat, and the same element or group can appear more than once. • choice. If you select this option, you can define members that are elements or groups. Exactly one of the defined members must be present in the message, and can repeat. Use this option if you want to model C unions and COBOL REDEFINES in a Custom Wire Format, or an XML DTD element that uses choice in an XML Wire Format, or a SWIFT field that has more than one option. • all. If you select this option, you can define members that are elements; groups are not allowed. The elements in an all group can appear in any order. Each element can appear once, or not at all. An all group can be used only at the top level of a complex type; it cannot be a member of another group within a type. • unorderedSet. This option is supported only by the MRM domain.If you select this option, you can define members that are elements. The elements can repeat, but the same element cannot appear twice in the list of members. The elements can appear in any order in the message. • orderedSet. This option is supported only by the MRM domain.If you select this option, you can define members that are elements. The elements can repeat, but the same element cannot appear twice in the list of members. The elements must appear in the specified order in the message. • message. This option is supported only by the MRM domain. If you select this option, you can define only messages as members. Each member can repeat, but the same message cannot appear twice in the list of members. Like choice, only one of the defined members can be present in a message. Unlike choice, when writing a message, if the complex type or group has more than one member, the bit stream is not padded to the length of the longest member. Use this option to model multipart messages, which are used in some industry standards; for example, SWIFT. For more information, see the section on multipart messages in “Multipart messages” on page 1191.

Property	Type	Meaning
Content Validation	Enumerated type	`Content Validation` is used only by the MRM domain. If validation is enabled in your message flow, `Content Validation` specifies the strictness of the MRM validation for members of a complex type or group. See "MRM content validation" on page 5422 for further details. Select from the following options: • Closed. The complex type can only contain the child elements that you have added to it. • Open Defined. The complex type can contain any valid element defined within the message set. • Open. The complex type can contain any valid element, not just those that you have added to this complex type. See "Combinations of Composition and Content Validation" on page 5612 for further details of these options.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Group reference logical properties:

The logical properties of a group reference include properties that specify the number of occurrences of the group reference.

Property	Type	Meaning
Reference Name	Enumerated type	The `Reference Name` is the name of the object that this object is referring to. The objects available to reference can be selected from the list.

Occurrence properties

Property	Type	Meaning
Min Occurs	Integer	Specify the minimum number of times that the object can repeat. The default value is 1. If the value is set to 0, the object is optional. With the exception of `Max Occurs` being set to -1, if a value is set for `Min Occurs`, it must be less than or equal to the value in `Max Occurs`.

Property	Type	Meaning
Max Occurs	Integer	Specify the maximum number of times that the object can repeat. The default value is 1. If this property is not set, the object cannot occur more than once. If this property is set to 0, it is interpreted as if the object does not exist in the message. It can also be set to -1, to indicate that the limit is unbounded and there is no maximum to the number of occurrences.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Key logical properties:

There are no properties to show.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Keyref logical properties:

This describes the logical properties of a keyref.

There are no properties to show.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Local attribute logical properties:

The logical properties of a local attribute.

Property	Type	Meaning
Name	String	Specify a name for the object when you create it. Names can consist of alphanumeric characters, including the letters *A* through *Z*, *a* through *z*, and the digits *0* through *9*. They might also include the following punctuation characters; • - the hyphen • _ the underscore • . the period Names can start only with a letter or the underscore character, and not with a number, hyphen, or period. Names that begin with `xml`, or any variant of these characters (for example `XmL`), are reserved by the XML standards specification. Further details of naming conventions and allowable characters can be found in the Extensible Markup Language (XML) specification that can be found on the World Wide Web Consortium (W3C) Web site.

Property	Type	Meaning
Type	Enumerated type	The Type property constrains the type of data that can be present in the object. Select from the following options: • `int` • `string` • `Boolean` • `hexBinary` • `dateTime` • `date` • `time` • `decimal` • `float` • `(More...)` • `(New Simple Type)` • `(New Complex Type)` If you select (More...), the Type Selection wizard starts. In this wizard, you can select any of the available types. If you select (New Simple Type), the New Simple Type wizard starts. In this wizard, you can create an Anonymous simple type that is based on an existing type. This type can be created locally or globally. If you select (New Complex Type), the New Complex Type wizard starts. In this wizard, you can create an Anonymous complex type, which can be derived from an existing base type. This type can be created locally or globally. For further information about these types, and examples of their use, see the XML Schema Part 0: Primer. This document is available on the World Wide Web Consortium (W3C) Web site.
Namespace	Enumerated type	Namespaces are a simple method for qualifying element and attribute names by associating them with namespaces identified by URI references. If <no target namespace> is displayed, a namespace has not been set for this object. If the property is inactive, the message set has not been configured to support namespaces. Where the property is active, namespaces that are available for selection are displayed in the drop-down list.

Value

The *Value* properties are used in conjunction with the *Usage* property in an Attribute Reference or a Local Attribute.

Property	Type	Meaning
Default	Button and String	This property provides the default value for an element or attribute. **XMLNSC domain** When parsing with validation enabled, default values are applied to missing attributes and empty elements as required by the XML Schema 1.0 specification. When writing, elements or attributes that are missing from the message tree are not automatically added to the output XML bit stream, even if they have default values. If missing elements or attributes are required, the message tree can be serialized and then re-parsed with validation enabled. **MRM (CWF and TDS physical formats)** When writing a fixed-length portion of a message (CWF or fixed-length TDS), if an attribute or element is missing from the message tree, the default value is inserted into the bit stream so that the message structure is preserved. **MRM (XML physical format)** No support for default values **Other domains** No support for default values.
Fixed	Button and String	This property provides the fixed value for an element or attribute. **XMLNSC domain** When parsing with validation enabled, if an attribute or element is present, the value is validated against the fixed value. If the values are not equal, a validation error is signalled. Also, when parsing with validation enabled, fixed values are applied to missing attributes and empty elements as required by the XML Schema 1.0 specification. When writing, elements or attributes that are missing from the message tree are not automatically added to the output XML bit stream, even if a fixed value has been specified. If missing elements or attributes are required, the message tree can be serialized and then re-parsed with validation enabled. **MRM (CWF and TDS physical formats)** When writing a fixed-length portion of a message (CWF or fixed-length TDS), if an attribute or element is missing from the message tree, the fixed value is inserted into the bit stream so that the message structure is preserved. **MRM (XML physical format)** No support for fixed values **Other domains** No support for fixed values.

Property	Type	Meaning
Interpret Value As	Enumerated type	Specify if values stored within this object must be interpreted as having significance for the parser and, if so, the type of interpretation that must occur. Select one of the following options: • None This value is the default value, and indicates that the element or attribute does not have a key value associated with it. • MessageSetIdentity. Specifies that the value of the element or attribute corresponds to the identifier, name, or alias (in that priority order) that is associated with the message set where all subsequent embedded messages that are descendents of the enclosing message are defined. This value remains in force unless a new element or attribute MessageSetIdentity field is encountered which resets the MessageSetIdentity value. • MessageIdentity. Specifies that the value of the element or attribute corresponds to the name or alias (in that priority order) that is associated with a message, and acts as an identifier for subsequent embedded messages which are the immediate children of the enclosing message. This identity applies until a new element or attribute MessageIdentity field is encountered at the same level in the tree. The embedded message can be defined in either the current message set, or in a message set identified by using a MessageSetIdentity. **Note:** This property is applicable only when the type of the object is derived from xsd:string.

Usage properties

Property	Type	Meaning
Usage	Enumerated type	Use this property with the `Value` property found in an attribute object. The default value for the `Usage` property is optional. Select from the following options: • optional. – If the `Value` property is set to default, and no data has been entered in the `Value` property, the attribute can appear once and can have any value. – If the `Value` property is set to default, the attribute can appear once. If it does not appear, its value is the data that has been entered in the `Value` property. If it does appear, it is the value given. – Where the `Value` property is set to fixed, the attribute can appear once. If it does appear, its value must match the data that has been entered in the `Value` property. If it does not appear, its value is the data that has been entered in the `Value` property. • prohibited. The attribute must not appear. • required. – If the `Value` property is set to default, and no data has been entered in the `Value` property, the attribute must appear once and can have any value. – If the `Value` property is set to fixed, the attribute must appear once, and it must match the data that has been entered in the `Value` property.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Local element logical properties:

The logical properties of a local element include properties that specify the number of occurrences and value of the local element.

Property	Type	Meaning
Name	String	Specify a name for the object when you create it. Names can consist of alphanumeric characters, including the letters *A* through *Z*, *a* through *z*, and the digits *0* through *9*. They might also include the following punctuation characters; • - the hyphen • _ the underscore • . the period Names can start only with a letter or the underscore character, and not with a number, hyphen, or period. Names that begin with `xml`, or any variant of these characters (for example `XmL`), are reserved by the XML standards specification. Further details of naming conventions and allowable characters can be found in the Extensible Markup Language (XML) specification that can be found on the World Wide Web Consortium (W3C) Web site.

Property	Type	Meaning
Type	Enumerated type	The Type property constrains the type of data that can be present in the object. Select from the following options: • `int` • `string` • `Boolean` • `hexBinary` • `dateTime` • `date` • `time` • `decimal` • `float` • `(More...)` • `(New Simple Type)` • `(New Complex Type)` If you select (More...), the Type Selection wizard starts. In this wizard, you can select any of the available types. If you select (New Simple Type), the New Simple Type wizard starts. In this wizard, you can create an Anonymous simple type that is based on an existing type. This type can be created locally or globally. If you select (New Complex Type), the New Complex Type wizard starts. In this wizard, you can create an Anonymous complex type, which can be derived from an existing base type. This type can be created locally or globally. For further information about these types, and examples of their use, see the XML Schema Part 0: Primer. This document is available on the World Wide Web Consortium (W3C) Web site.
Namespace	Enumerated type	Namespaces are a simple method for qualifying element and attribute names by associating them with namespaces identified by URI references. If <no target namespace> is displayed, a namespace has not been set for this object. If the property is inactive, the message set has not been configured to support namespaces. Where the property is active, namespaces that are available for selection are displayed in the drop-down list.

Occurrences

Property	Type	Meaning
Min Occurs	Integer	Specify the minimum number of times that the object can repeat. The default value is 1. If the value is set to 0, the object is optional. With the exception of `Max Occurs` being set to -1, if a value is set for `Min Occurs`, it must be less than or equal to the value in `Max Occurs`.

Property	Type	Meaning
Max Occurs	Integer	Specify the maximum number of times that the object can repeat. The default value is 1. If this property is not set, the object cannot occur more than once. If this property is set to 0, it is interpreted as if the object does not exist in the message. It can also be set to -1, to indicate that the limit is unbounded and there is no maximum to the number of occurrences.

Value

Property	Type	Meaning
Default	Button and String	This property provides the default value for an element or attribute. **XMLNSC domain** When parsing with validation enabled, default values are applied to missing attributes and empty elements as required by the XML Schema 1.0 specification. When writing, elements or attributes that are missing from the message tree are not automatically added to the output XML bit stream, even if they have default values. If missing elements or attributes are required, the message tree can be serialized and then re-parsed with validation enabled. **MRM (CWF and TDS physical formats)** When writing a fixed-length portion of a message (CWF or fixed-length TDS), if an attribute or element is missing from the message tree, the default value is inserted into the bit stream so that the message structure is preserved. **MRM (XML physical format)** No support for default values **Other domains** No support for default values.
Fixed	Button and String	This property provides the fixed value for an element or attribute. **XMLNSC domain** When parsing with validation enabled, if an attribute or element is present, the value is validated against the fixed value. If the values are not equal, a validation error is signalled. Also, when parsing with validation enabled, fixed values are applied to missing attributes and empty elements as required by the XML Schema 1.0 specification. When writing, elements or attributes that are missing from the message tree are not automatically added to the output XML bit stream, even if a fixed value has been specified. If missing elements or attributes are required, the message tree can be serialized and then re-parsed with validation enabled. **MRM (CWF and TDS physical formats)** When writing a fixed-length portion of a message (CWF or fixed-length TDS), if an attribute or element is missing from the message tree, the fixed value is inserted into the bit stream so that the message structure is preserved. **MRM (XML physical format)** No support for fixed values **Other domains** No support for fixed values.

Property	Type	Meaning
Nillable	Check box	Select this option if you want the element to be able to be defined as null. A null value is distinct from being empty, when the element contains no data.
Interpret Value As	Enumerated type	Specify if values stored within this object must be interpreted as having significance for the parser and, if so, the type of interpretation that must occur. Select one of the following options: • None This value is the default value, and indicates that the element or attribute does not have a key value associated with it. • MessageSetIdentity. Specifies that the value of the element or attribute corresponds to the identifier, name, or alias (in that priority order) that is associated with the message set where all subsequent embedded messages that are descendents of the enclosing message are defined. This value remains in force unless a new element or attribute MessageSetIdentity field is encountered which resets the MessageSetIdentity value. • MessageIdentity. Specifies that the value of the element or attribute corresponds to the name or alias (in that priority order) that is associated with a message, and acts as an identifier for subsequent embedded messages which are the immediate children of the enclosing message. This identity applies until a new element or attribute MessageIdentity field is encountered at the same level in the tree. The embedded message can be defined in either the current message set, or in a message set identified by using a MessageSetIdentity. **Note:** This property is applicable only when the type of the object is derived from xsd:string.

Substitution settings

Substitution Groups provide a means by which one element may be substituted for another in a message. The element which can be substituted is called the 'head' element, and the substitution group is the list of elements that may be used in its place. An element can be in at most one substitution group.

Property	Type	Meaning
Final	Enumerated type	Limit the set of elements that can belong to its substitution group. • Empty • restriction. Prohibit element substitution by elements whose types are restrictions of the type of the head element. • extension. Prohibit element substitution by elements whose types are extensions of the type of the head element. • #all. Prohibit substitution by all methods.
Block	Enumerated type	Limit the set of elements that can be substituted for this element in a message. • Empty • restriction. Prohibit element substitution by elements whose types are restrictions of the type of the head element • extension. Prohibit element substitution by elements whose types are extensions of the type of the head element • substitution. Prohibit element substitution by members of the element's substitution group. • #all. Prohibit substitution by all methods.
Substitution Group	Enumerated type	Specify the name of a 'head' element. Setting this property indicates that this element is a member of the substitution group for the head element.

Property	Type	Meaning
Abstract	Check box	Select this option if you do not want the element to appear in the message, but require one of the members of its substitution group to appear in its place.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Local group logical properties:

The logical properties of a local group include properties that specify the number of occurrences of the local group.

Valid children in a local group that depend on both **Composition** and **Content Validation** are shown in "MRM content validation" on page 5422.

Property	Type	Meaning
Composition	Enumerated type	Define the order, and the number of occurrences, of the elements and groups in your messages. Composition does not affect the attributes in a complex type. Select from the following options: • Empty • sequence. If you select this option, you can define members that are elements or groups. These members, if present, must appear in the specified order in the message. They can repeat, and the same element or group can appear more than once. • choice. If you select this option, you can define members that are elements or groups. Exactly one of the defined members must be present in the message, and can repeat. Use this option if you want to model C unions and COBOL REDEFINES in a Custom Wire Format, or an XML DTD element that uses choice in an XML Wire Format, or a SWIFT field that has more than one option. • all. If you select this option, you can define members that are elements; groups are not allowed. The elements in an all group can appear in any order. Each element can appear once, or not at all. An all group can be used only at the top level of a complex type; it cannot be a member of another group within a type. • unorderedSet. This option is supported only by the MRM domain.If you select this option, you can define members that are elements. The elements can repeat, but the same element cannot appear twice in the list of members. The elements can appear in any order in the message. • orderedSet. This option is supported only by the MRM domain.If you select this option, you can define members that are elements. The elements can repeat, but the same element cannot appear twice in the list of members. The elements must appear in the specified order in the message. • message. This option is supported only by the MRM domain. If you select this option, you can define only messages as members. Each member can repeat, but the same message cannot appear twice in the list of members. Like choice, only one of the defined members can be present in a message. Unlike choice, when writing a message, if the complex type or group has more than one member, the bit stream is not padded to the length of the longest member. Use this option to model multipart messages, which are used in some industry standards; for example, SWIFT. For more information, see the section on multipart messages in “Multipart messages” on page 1191.

Property	Type	Meaning
Content Validation	Enumerated type	`Content Validation` is used only by the MRM domain. If validation is enabled in your message flow, `Content Validation` specifies the strictness of the MRM validation for members of a complex type or group. See "MRM content validation" on page 5422 for further details. Select from the following options: • Closed. The complex type can only contain the child elements that you have added to it. • Open Defined. The complex type can contain any valid element defined within the message set. • Open. The complex type can contain any valid element, not just those that you have added to this complex type. See "Combinations of Composition and Content Validation" on page 5612 for further details of these options.

Occurrences

Property	Type	Meaning
Min Occurs	Integer	Specify the minimum number of times that the object can repeat. The default value is 1. If the value is set to 0, the object is optional. With the exception of `Max Occurs` being set to -1, if a value is set for `Min Occurs`, it must be less than or equal to the value in `Max Occurs`.
Max Occurs	Integer	Specify the maximum number of times that the object can repeat. The default value is 1. If this property is not set, the object cannot occur more than once. If this property is set to 0, it is interpreted as if the object does not exist in the message. It can also be set to -1, to indicate that the limit is unbounded and there is no maximum to the number of occurrences.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Message logical properties:

This section describes the logical properties of a message.

Property	Type	Meaning
Name	String	Specify a name for the object when you create it. Names can consist of alphanumeric characters, including the letters *A* through *Z*, *a* through *z*, and the digits *0* through *9*. They might also include the following punctuation characters; • - the hyphen • _ the underscore • . the period Names can start only with a letter or the underscore character, and not with a number, hyphen, or period. Names that begin with `xml`, or any variant of these characters (for example `XmL`), are reserved by the XML standards specification. Further details of naming conventions and allowable characters can be found in the Extensible Markup Language (XML) specification that can be found on the World Wide Web Consortium (W3C) Web site.
Message Alias	String	Specify an alternative unique value that identifies the message. This property is only required if you are using the MRM domain and the Message Identity technique to identify embedded messages, and the bit stream does not contain the actual message name.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Simple type logical properties:

The logical properties of a simple type.

Property	Type	Meaning
Name	String	Specify a name for the object when you create it. Names can consist of alphanumeric characters, including the letters *A* through *Z*, *a* through *z*, and the digits *0* through *9*. They might also include the following punctuation characters; • - the hyphen • _ the underscore • . the period Names can start only with a letter or the underscore character, and not with a number, hyphen, or period. Names that begin with `xml`, or any variant of these characters (for example `XmL`), are reserved by the XML standards specification. Further details of naming conventions and allowable characters can be found in the Extensible Markup Language (XML) specification that can be found on the World Wide Web Consortium (W3C) Web site.
Base Type	Enumerated type	This property only applies to a simple type restriction. You can use this property to select a base type that is used as the starting point to define a new simple type that is derived by setting additional value constraints.
Item Type	Enumerated type	This property only applies to a simple type list. You can use this property to select the type that is used as the item type of the list.
Variety	Enumerated type	This property displays the variety of the simple type you have selected, either atomic, list, or union.

A simple type can also have "Simple type logical value constraints."

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Simple type logical value constraints:

The properties, and their permissible values, vary according to the object type.

The properties that are displayed on the object page, and the values that those properties can take, can vary according to the type of the object. For example, the properties for type *string* are different from those of type *Boolean*. Select the link for the object type from the following table.

Type of object	**Properties**
Binary types	• base64Binary • hexBinary
Boolean types	• Boolean
DateTime types	• date • dateTime • gDay • gMonth • gMonthDay • gYear • gYearMonth • time
Decimal types	• decimal • integer • negativeInteger • nonNegativeInteger • nonPositiveInteger • positiveInteger • unsignedLong
Float types	• double • float
Integer types	• byte • int • long • short • unsignedByte • unsignedInt • unsignedShort
Interval types	• duration

Type of object	Properties
String types	• anyURI • ENTITIES • ENTITY • ID • IDREF • IDREFS • language • Name • NCName • NMTOKEN • NMTOKENS • normalizedString • NOTATION • QName • string • token

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Unique logical properties:

There are no properties to show.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Wildcard attribute logical properties:

The logical properties of a wildcard attribute.

Property	Type	Meaning
Namespace	String	Namespaces are a simple method for qualifying element and attribute names by associating them with namespaces identified by URI references. This field is initially blank.

Property	Type	Meaning
Process Content	Enumerated type	If a message contains an attribute that corresponds to a wildcard in the message model, `Process Content` defines how the attribute is validated. Select one of the following options: • strict. The parser can match only against attributes declared in the specified namespace. • lax. The parser attempts to match against attributes declared in all accessible namespaces. If the specified namespace cannot be found, an error is not generated. • skip. If you select skip, the parser does not perform validation on the attribute.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Wildcard element logical properties:

The logical properties of a wildcard element include properties that specify the number of occurrences of the wildcard element.

Property	Type	Meaning
Namespace	String	Namespaces are a simple method for qualifying element and attribute names by associating them with namespaces identified by URI references. This field is initially blank.

Property	Type	Meaning
Process Content	Enumerated type	If a message contains an element that corresponds to a wildcard in the message model, `Process Content` defines how the element is validated. Select one of the following options: • strict. The parser can match only against elements declared in the specified namespace. • lax. The parser attempts to match against elements declared in any accessible namespace. If the specified namespace cannot be found, an error is not generated. • skip. If you select skip the parser does not perform validation on the element.

Occurrences

Property	Type	Meaning
Min Occurs	Integer	Specify the minimum number of times that the object can repeat. The default value is 1. If the value is set to 0, the object is optional. With the exception of `Max Occurs` being set to -1, if a value is set for `Min Occurs`, it must be less than or equal to the value in `Max Occurs`. This property is ignored by brokers that are earlier than WebSphere Message Broker Version 6.0.
Max Occurs	Integer	Specify the maximum number of times that the object can repeat. The default value is 1. If this property is not set, the object cannot occur more than once. If this property is set to 0, it is interpreted as if the object does not exist in the message. It can also be set to -1, to indicate that the limit is unbounded and there is no maximum to the number of occurrences. This property is ignored by brokers that are earlier than WebSphere Message Broker Version 6.0.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Physical properties for message model objects:

CWF, XML, and TDS format physical properties for message model objects.

Property information is available for objects within:

- "Custom Wire Format physical properties for message model objects"
- "XML wire format physical properties for message model objects" on page 5476
- "TDS format physical properties for message model objects" on page 5501

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with a message set" on page 2840
Complete a variety of tasks that are involved in working with a message set.

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Physical properties for message model objects"
CWF, XML, and TDS format physical properties for message model objects.

Custom Wire Format physical properties for message model objects:

Custom wire format physical property information is available for some objects.

- "Attribute group reference CWF properties" on page 5456
- "Attribute reference CWF properties" on page 5457
- "Complex type CWF properties" on page 5459
- "Element reference CWF properties" on page 5460
- "Global attribute CWF properties" on page 5462
- "Global attribute group CWF properties" on page 5463
- "Global element CWF properties" on page 5463
- "Global group CWF properties" on page 5464
- "Group reference CWF properties" on page 5465
- "Key CWF properties" on page 5466
- "Keyref CWF properties" on page 5467
- "Local element CWF properties" on page 5469
- "Local group CWF properties" on page 5471

- "Message CWF properties" on page 5473
- "Simple type CWF properties" on page 5474
- "Unique CWF properties" on page 5474
- "Wildcard attribute CWF properties" on page 5475
- "Wildcard element CWF properties" on page 5475

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM Custom Wire Format" on page 1214
Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not separated by delimiters.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Attribute group reference CWF properties:

There are no properties to show.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM Custom Wire Format" on page 1214
Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not

separated by delimiters.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Attribute reference CWF properties:

The properties, and their permissible values, vary according to the object type.

The properties that are displayed on the object page, and the values that those properties can take, can vary according to the type of the object. For example, the properties for type *string* are different from those of type *Boolean*. Select the link for the object type from the following table.

Type of object	Properties
Binary types	• base64Binary • hexBinary
Boolean types	• Boolean
DateTime types	• date • dateTime • gDay • gMonth • gMonthDay • gYear • gYearMonth • time

Type of object	Properties
Decimal types	• decimal • integer • negativeInteger • nonNegativeInteger • nonPositiveInteger • positiveInteger • unsignedLong
Float types	• double • float
Integer types	• byte • int • long • short • unsignedByte • unsignedInt • unsignedShort
Interval types	• duration
String types	• anyURI • ENTITIES • ENTITY • ID • IDREF • IDREFS • language • Name • NCName • NMTOKEN • NMTOKENS • normalizedString • NOTATION • QName • string • token

Note: The physical format properties for simple type *duration* are the same as the physical properties of the string logical types.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM Custom Wire Format" on page 1214
Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not separated by delimiters.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Complex type CWF properties:

There are no properties to show.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM Custom Wire Format" on page 1214
Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not separated by delimiters.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Element reference CWF properties:

The properties, and permissible values, vary according to the type of object.

The properties that are displayed on the object page, and the values that those properties can take, can vary according to the type of the object. For example, the properties for type *string* are different from those of type *Boolean*. Select the link for the object type from the following table.

Type of object	**Properties**
Binary types	• base64Binary • hexBinary
Boolean types	• Boolean
DateTime types	• date • dateTime • gDay • gMonth • gMonthDay • gYear • gYearMonth • time
Decimal types	• decimal • integer • negativeInteger • nonNegativeInteger • nonPositiveInteger • positiveInteger • unsignedLong
Float types	• double • float

Type of object	Properties
Integer types	• byte • int • long • short • unsignedByte • unsignedInt • unsignedShort
Interval types	• duration
String types	• anyURI • ENTITIES • ENTITY • ID • IDREF • IDREFS • language • Name • NCName • NMTOKEN • NMTOKENS • normalizedString • NOTATION • QName • string • token

Note: The physical format properties for simple type *duration* are the same as the physical properties of the string logical types.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM Custom Wire Format" on page 1214
Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not separated by delimiters.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Global attribute CWF properties:

There are no properties to show.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM Custom Wire Format" on page 1214
Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not separated by delimiters.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Global attribute group CWF properties:

There are no properties to show.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM Custom Wire Format" on page 1214
Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not separated by delimiters.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Global element CWF properties:

There are no CWF properties to show for a global element.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM Custom Wire Format" on page 1214
Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not

separated by delimiters.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Global group CWF properties:

There are no properties to show.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM Custom Wire Format" on page 1214
Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not separated by delimiters.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Group reference CWF properties:

The CWF properties of a group reference.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to all instances.

Occurrences

Property	Type	Meaning
Repeat Reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the displayed list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* logical property when parsing and writing the message, but not for validation of the message.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM Custom Wire Format" on page 1214
Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not separated by delimiters.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Key CWF properties:

There are no properties to show.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Keyref CWF properties:

This describes the CWF properties of a keyref.

There are no properties to show.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Local attribute CWF properties:

The properties, and their permissible values, vary according to the object type.

The properties that are displayed on the object page, and the values that those properties can take, can vary according to the type of the object. For example, the properties for type *string* are different from those of type *Boolean*. Select the link for the object type from the following table.

Type of object	Properties
Binary types	• base64Binary • hexBinary
Boolean types	• Boolean

Type of object	Properties
DateTime types	• date • dateTime • gDay • gMonth • gMonthDay • gYear • gYearMonth • time
Decimal types	• decimal • integer • negativeInteger • nonNegativeInteger • nonPositiveInteger • positiveInteger • unsignedLong
Float types	• double • float
Integer types	• byte • int • long • short • unsignedByte • unsignedInt • unsignedShort
Interval types	• duration
String types	• anyURI • ENTITIES • ENTITY • ID • IDREF • IDREFS • language • Name • NCName • NMTOKEN • NMTOKENS • normalizedString • NOTATION • QName • string • token

Note: The physical format properties for simple type *duration* are the same as the physical properties of the string logical types.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM Custom Wire Format" on page 1214
Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not separated by delimiters.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Local element CWF properties:

The properties, and their permissible values, vary according to the object type.

The properties that are displayed on the object page, and the values that those properties can take, can vary according to the type of the object. For example, the properties for type *string* are different from those of type *Boolean*. Select the link for the object type from the following table.

Type of object	Properties
Binary types	• base64Binary • hexBinary
Boolean types	• Boolean

Type of object	Properties
DateTime types	• date • dateTime • gDay • gMonth • gMonthDay • gYear • gYearMonth • time
Decimal types	• decimal • integer • negativeInteger • nonNegativeInteger • nonPositiveInteger • positiveInteger • unsignedLong
Float types	• double • float
Integer types	• byte • int • long • short • unsignedByte • unsignedInt • unsignedShort
Interval types	• duration
String types	• anyURI • ENTITIES • ENTITY • ID • IDREF • IDREFS • language • Name • NCName • NMTOKEN • NMTOKENS • normalizedString • NOTATION • QName • string • token

Note: The physical format properties for simple type *duration* are the same as the physical properties of the string logical types.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM Custom Wire Format" on page 1214
Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not separated by delimiters.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Local group CWF properties:

The CWF properties of a local group are described in the following tables.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`

Property	Type	Meaning
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to all instances.

Occurrences

Property	Type	Meaning
Repeat Reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the displayed list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, there are no integer objects before this one in the message structure. If a *`Repeat Reference`* is specified, it overrides any setting for the *`Max Occurs`* logical property when parsing and writing the message, but not for validation of the message.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM Custom Wire Format" on page 1214
Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not separated by delimiters.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Message CWF properties:

There are no properties to show.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM Custom Wire Format" on page 1214
Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not separated by delimiters.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Simple type CWF properties:

There are no properties to show.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM Custom Wire Format" on page 1214
Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not separated by delimiters.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Unique CWF properties:

There are no properties to show.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Wildcard attribute CWF properties:

There are no properties to show.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM Custom Wire Format" on page 1214
Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not separated by delimiters.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Wildcard element CWF properties:

There are no properties to show.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise

appearance of your message bit stream during transmission.

"MRM Custom Wire Format" on page 1214
Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not separated by delimiters.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"XML wire format physical properties for message model objects"
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

XML wire format physical properties for message model objects:

XML wire format physical property information is available for some objects.

- "Attribute group reference XML properties" on page 5477
- "Attribute reference XML properties" on page 5478
- "Complex type XML properties" on page 5480
- "Element reference XML properties" on page 5481
- "Global attribute XML properties" on page 5483
- "Global attribute group XML properties" on page 5485
- "Global element XML properties" on page 5486
- "Global group XML properties" on page 5488
- "Group reference XML properties" on page 5488
- "Key XML properties" on page 5489
- "Keyref XML properties" on page 5490
- "Local attribute XML properties" on page 5490
- "Local element XML properties" on page 5492
- "Local group XML properties" on page 5494
- "Message XML properties" on page 5495

- "Simple type XML properties" on page 5498
- "Unique XML properties" on page 5499
- "Wildcard attribute XML properties" on page 5499
- "Wildcard element XML properties" on page 5500

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Attribute group reference XML properties:

The XML properties of an attribute group reference.

There are no properties to show.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Attribute reference XML properties:

The properties, and their permissible values, vary according to the object type.

The properties that are displayed on the object page, and the values that those properties can take, can vary according to the type of the object. For example, the properties for type *string* are different from those of type *Boolean*. Select the link for the object type from the following table.

Type of object	Properties
Binary types	• base64Binary • hexBinary
Boolean types	• Boolean
DateTime types	• date • dateTime • gDay • gMonth • gMonthDay • gYear • gYearMonth • time

Type of object	Properties
Decimal types	• decimal • integer • negativeInteger • nonNegativeInteger • nonPositiveInteger • positiveInteger • unsignedLong
Float types	• double • float
Integer types	• byte • int • long • short • unsignedByte • unsignedInt • unsignedShort
Interval types	• duration
String types	• anyURI • ENTITIES • ENTITY • ID • IDREF • IDREFS • language • Name • NCName • NMTOKEN • NMTOKENS • normalizedString • NOTATION • QName • string • token

Note: The physical format properties for simple type *duration* are the same as the physical properties of the string logical types.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Complex type XML properties:

There are no properties to show.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Element reference XML properties:

The properties, and their permissible values, vary according to the type of object.

The properties that are displayed on the object page, and the values that those properties can take, can vary according to the type of the object. For example, the properties for type *string* are different from those of type *Boolean*. Select the link for the object type from the following table.

Type of object	**Properties**
Binary types	• base64Binary • hexBinary
Boolean types	• Boolean
DateTime types	• date • dateTime • gDay • gMonth • gMonthDay • gYear • gYearMonth • time
Decimal types	• decimal • integer • negativeInteger • nonNegativeInteger • nonPositiveInteger • positiveInteger • unsignedLong
Float types	• double • float

Type of object	Properties
Integer types	• byte • int • long • short • unsignedByte • unsignedInt • unsignedShort
Interval types	• duration
String types	• anyURI • ENTITIES • ENTITY • ID • IDREF • IDREFS • language • Name • NCName • NMTOKEN • NMTOKENS • normalizedString • NOTATION • QName • string • token

Note: The physical format properties for simple type *duration* are the same as the physical properties of the string logical types.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Global attribute XML properties:

The properties, and their permissible values, vary according to the object type.

The properties that are displayed on the object page, and the values that those properties can take, can vary according to the type of the object. For example, the properties for type *string* are different from those of type *Boolean*. Select the link for the object type from the following table.

Type of object	**Properties**
Binary types	• base64Binary • hexBinary
Boolean types	• Boolean
DateTime types	• date • dateTime • gDay • gMonth • gMonthDay • gYear • gYearMonth • time
Decimal types	• decimal • integer • negativeInteger • nonNegativeInteger • nonPositiveInteger • positiveInteger • unsignedLong
Float types	• double • float

Type of object	Properties
Integer types	• byte • int • long • short • unsignedByte • unsignedInt • unsignedShort
Interval types	• duration
String types	• anyURI • ENTITIES • ENTITY • ID • IDREF • IDREFS • language • Name • NCName • NMTOKEN • NMTOKENS • normalizedString • NOTATION • QName • string • token

Note: The physical format properties for simple type *duration* are the same as the physical properties of the string logical types.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Global attribute group XML properties:

There are no properties to show.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Global element XML properties:

The properties, and their permissible values, vary according to the type of object.

The properties that are displayed on the object page, and the values that those properties can take, can vary according to the type of the object. For example, the properties for type *string* are different from those of type *Boolean*. Select the link for the object type from the following table.

Type of object	**Properties**
Binary types	• base64Binary • hexBinary
Boolean types	• Boolean
DateTime types	• date • dateTime • gDay • gMonth • gMonthDay • gYear • gYearMonth • time
Decimal types	• decimal • integer • negativeInteger • nonNegativeInteger • nonPositiveInteger • positiveInteger • unsignedLong
Float types	• double • float
Integer types	• byte • int • long • short • unsignedByte • unsignedInt • unsignedShort
Interval types	• duration

Type of object	Properties
String types	• anyURI • ENTITIES • ENTITY • ID • IDREF • IDREFS • language • Name • NCName • NMTOKEN • NMTOKENS • normalizedString • NOTATION • QName • string • token

Note: The physical format properties for simple type *duration* are the same as the physical properties of the string logical types.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Global group XML properties:

There are no properties to show.

There are no properties to show.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Group reference XML properties:

The XML properties of a group reference.

There are no properties to show.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Key XML properties:

There are no properties to show.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Keyref XML properties:

There are no properties to show.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Local attribute XML properties:

The properties, and their permissible values, vary according to the object type.

The properties that are displayed on the object page, and the values that those properties can take, can vary according to the type of the object. For example, the properties for type *string* are different from those of type *Boolean*. Select the link for the object type from the following table.

Type of object	Properties
Binary types	• base64Binary • hexBinary
Boolean types	• Boolean
DateTime types	• date • dateTime • gDay • gMonth • gMonthDay • gYear • gYearMonth • time

Type of object	Properties
Decimal types	• decimal • integer • negativeInteger • nonNegativeInteger • nonPositiveInteger • positiveInteger • unsignedLong
Float types	• double • float
Integer types	• byte • int • long • short • unsignedByte • unsignedInt • unsignedShort
Interval types	• duration
String types	• anyURI • ENTITIES • ENTITY • ID • IDREF • IDREFS • language • Name • NCName • NMTOKEN • NMTOKENS • normalizedString • NOTATION • QName • string • token

Note: The physical format properties for simple type *duration* are the same as the physical properties of the string logical types.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Local element XML properties:

The properties, and their permissible values, vary according to the object type.

The properties that are displayed on the object page, and the values that those properties can take, can vary according to the type of the object. For example, the properties for type *string* are different from those of type *Boolean*. Select the link for the object type from the following table.

Type of object	Properties
Binary types	• base64Binary • hexBinary
Boolean types	• Boolean
DateTime types	• date • dateTime • gDay • gMonth • gMonthDay • gYear • gYearMonth • time

Type of object	Properties
Decimal types	• decimal • integer • negativeInteger • nonNegativeInteger • nonPositiveInteger • positiveInteger • unsignedLong
Float types	• double • float
Integer types	• byte • int • long • short • unsignedByte • unsignedInt • unsignedShort
Interval types	• duration
String types	• anyURI • ENTITIES • ENTITY • ID • IDREF • IDREFS • language • Name • NCName • NMTOKEN • NMTOKENS • normalizedString • NOTATION • QName • string • token

Note: The physical format properties for simple type *duration* are the same as the physical properties of the string logical types.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Local group XML properties:

There are no properties to show.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Message XML properties:

The following tables describe the XML properties of a message.

Namespace schema locations

This property is only active if namespaces have been enabled.

Property	Type	Meaning
Namespace URI	String	A unique string, usually in the form of a URL that identifies the schema. If namespaces have not been enabled, this property displays `<no target namespace>`. This property overrides the same property at the message set level.
Schema location	String	Enter the location of the schema for the associated namespace name to be used to validate objects in the namespace.

XML declarations

Property	Type	Meaning
Output Namespace Declaration	Enumerated type	The `Output Namespace Declaration` property controls where the namespace declarations are placed in the output XML document. Select from: • `At start of document`. Declarations for all of the entries in the `Namespace schema locations` table above are produced as attributes of the message in the output XML document. The disadvantage of this option is that in some cases unnecessary declarations might be produced. • `As required`. Declarations are produced only when required by an element or attribute that is in that namespace. The disadvantage of this option is that the same namespace declaration might need to be produced more than once in the output XML document. The default option is `At start of document`. This property is active only if namespaces are enabled for this message set.

XML document type settings

Property	Type	Meaning
DOCTYPE System ID	String	Specify the System ID for DOCTYPE external DTD subset. It overrides the equivalent message set property setting for this particular message. If the message set property `Suppress DOCTYPE` is set to `Yes`, this parameter is ignored and cannot be changed (the field is disabled). The default value is the value that you specified for the `DOCTYPE System ID` property for the message set.
DOCTYPE Public ID	String	Specify the Public ID for DOCTYPE external DTD subset. It overrides the equivalent message set property setting for this particular message. If the message set property `Suppress DOCTYPE` is set to `Yes`, this parameter is ignored and cannot be changed (the field is disabled) . The default value is the value that you specified for the `DOCTYPE Public ID` property for the message set.
DOCTYPE Text	String	Enter optional additional text to include within the DOCTYPE. It overrides the message set property for this particular message. If the message set property `Suppress DOCTYPE` is set to `Yes`, this parameter is ignored and cannot be changed (the field is disabled). For more information, see "MRM XML: In-line DTDs and the DOCTYPE text property" on page 5407. The default value is the value that you specified for the `DOCTYPE Text` property for the message set.

Property	Type	Meaning
Root Tag Name	String	Specify the name of the root tag for a message bit stream XML document. It overrides the message set property set for this message. The default value is the value that you specified for the `Root Tag Name` property for the message set. **Note:** This property is deprecated. Do not change its value from its default setting.

Field identification

A number of the following properties will only become active depending on the value that `Render` property is set to.

Property	Type	Meaning
Render	Enumerated type	Specify how the instantiated object or type is rendered (for output) in the resulting XML document. Select one of the following values from the drop-down list: • `XMLElement`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their `XML Name` property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLElementAttrID`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in `ID Attribute Name` and a value as specified in `ID Attribute Value`. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set `XML Name`, `ID Attribute Name`, `ID Attribute Value` to the same values: – You must also set `Value Attribute Name` to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 6262 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.
XML Name	String	Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message). This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name. You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name. If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).
ID Attribute Name	String	Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if *`Render`* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *`ID Attribute`*.
ID Attribute Value	String	Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if *`Render`* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise

appearance of your message bit stream during transmission.

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Simple type XML properties:

There are no properties to show.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Unique XML properties:

There are no properties to show.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Wildcard attribute XML properties:

There are no properties to show.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Wildcard element XML properties:

There are no properties to show.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"TDS format physical properties for message model objects"
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

TDS format physical properties for message model objects:

Some objects have TDS wire format properties.

TDS format physical property information is available for the following objects:

- "Attribute group reference TDS properties" on page 5502
- "Attribute reference TDS properties" on page 5503
- "Complex type TDS properties" on page 5505
- "Element reference TDS properties" on page 5509
- "Global attribute TDS properties" on page 5511
- "Global attribute group TDS properties" on page 5513
- "Global element TDS properties" on page 5513
- "Global group TDS properties" on page 5516
- "Group reference TDS properties" on page 5520
- "Key TDS properties" on page 5521
- "Keyref TDS properties" on page 5521
- "Local attribute TDS properties" on page 5522
- "Local element TDS properties" on page 5524
- "Local group TDS properties" on page 5526
- "Message TDS properties" on page 5531
- "Simple type TDS properties" on page 5532
- "Unique TDS properties" on page 5532
- "White space characters in TDS" on page 5533
- "Wildcard attribute TDS properties" on page 5534
- "Wildcard element TDS properties" on page 5534

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM TDS format" on page 1221
The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Attribute group reference TDS properties:

There are no properties to show.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM TDS format" on page 1221
The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.
"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.
"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Attribute reference TDS properties:

The properties, and their permissible values, vary according to the object type.

The properties that are displayed on the object page, and the values that those properties can take, can vary according to the type of the object. For example, the properties for type *string* are different from those of type *Boolean*. Select the link for the object type from the following table.

Type of object	**Properties**
Binary types	• base64Binary • hexBinary
Boolean types	• Boolean
DateTime types	• date • dateTime • gDay • gMonth • gMonthDay • gYear • gYearMonth • time
Decimal types	• decimal • integer • negativeInteger • nonNegativeInteger • nonPositiveInteger • positiveInteger • unsignedLong
Float types	• double • float
Integer types	• byte • int • long • short • unsignedByte • unsignedInt • unsignedShort
Interval types	• duration

Type of object	Properties
String types	• anyURI • ENTITIES • ENTITY • ID • IDREF • IDREFS • language • Name • NCName • NMTOKEN • NMTOKENS • normalizedString • NOTATION • QName • string • token

Note: The physical format properties for simple type *duration* are the same as the physical properties of the string logical types.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM TDS format" on page 1221
The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Complex type TDS properties:

The TDS properties of a complex type.

Field Identification

If the complex type is based on a global group, the TDS properties listed are located in the global group. If so, any changes to these properties are applied to the global group, and affect all references to the group (including any other complex types which are based on it).

Property	Type	Meaning
Data Element Separation	Enumerated type	Select one of the following values to specify the method that is used to separate the data elements within the type. • `Tagged Delimited`. This value indicates that all elements within the complex type are identified by a tag, and, if a value is specified in the optional `Delimiter` property, are separated by that value. You must set the `Tag` property for all child elements of simple type, and you can set the `Delimiter` property to a non-empty value. See "Global element TDS properties" on page 5513. You must also set the `Tag Data Separator` or the `Length of Tag` property. • `Tagged Fixed Length`. This value indicates that each element is identified by a tag, and the data has a fixed length. There are no delimiters. You must set the `Tag` property for each of the child elements of this complex type, and each child element must have a `Length` or `Length Reference` property assigned to it. You must also set the `Tag Data Separator` or the `Length of Tag` property. • `Tagged Encoded Length`. This value indicates that all elements within the complex type are separated by a tag, and a length field follows each tag. There are no delimiters. The tag can be fixed length, as set by the `Length of Tag` property, or variable length delimited by the `Tag Data Separator` property. You must also set the `Length Of Encoded Length` property so that the parser knows the size of the length field, and set the `Extra Chars in Encoded Length` property. This property tells the parser what to subtract from the value in the `Length Of Encoded Length` property to get the actual length of the data that follows the length field. This method provides a more flexible way of handling ACORD AL3 standard messages than using the `Fixed Length AL3` value, by allowing different parts of the messages to be at different versions of the ACORD AL3 standard. • `All Elements Delimited`. This value indicates that all elements within the complex type are separated by a delimiter. You must set a value in the `Delimiter` property. • `Variable Length Elements Delimited`. This value indicates that some of the elements within the complex type might be of variable length. Variable length elements must be delimited by the value specified in the `Delimiter` property. • `Use Data Pattern`. This value indicates that the parser determines the elements by matching the data with the regular expression that is set in the `Data Pattern` property of the element or type member. See "Message definition file properties" on page 5409. • `Fixed Length`. This value indicates that all elements within the complex type are fixed length. The next data element is accessed by adding the value of the `Length` property to the offset. See "Global element TDS properties" on page 5513. If you set the `Data Element Separation` property of a complex type to `Fixed Length`, you must also set the `Data Element Separation` property of all complex children of this type to `Fixed Length`. Each child element must have a `Length` or `Length Reference` property assigned to it. • `Fixed Length AL3`. This value has a similar meaning to the separation type `Fixed Length`, but also indicates to the parser that a number of predefined rules regarding missing optional elements, encoded lengths, and versioning, must be applied. If you set the `Data Element Separation` property of a complex type to `Fixed Length AL3`, you must also set the `Data Element Separation` property of all complex children of this type to `Fixed Length AL3`. • `Undefined`. This value is set automatically if you set the `Type Composition` property of a complex type to `Message`, and you cannot change it to any other value. Do not set the `Type Composition` property to `Empty`, `Choice`, `Unordered Set`, `Ordered Set`, `Sequence`, or `Simple Unordered Set`. If you do, you cannot check in the type.

Property	Type	Meaning
Group Indicator	String	Specify the value of a special character, or string, that precedes the data that belongs to a group, or a complex type, within the bit stream.
Group Terminator	String	Specify the value of a special character, or string, that terminates the data that belongs to a group, or a complex type, within the bit stream.
Delimiter	String	Specify the value of a special character, or string, that specifies the delimiter that is used between data elements. This property applies only to the delimited `Data Element Separation` methods (`Tagged Delimited`, `All Elements Delimited`, and `Variable Length Elements Delimited`).
Suppress Absent Element Delimiters	Enumerated type	Use this property to select whether you want delimiters to be suppressed for elements that are missing within a message. Select from: • `End Of Type`. Use this option to suppress the delimiter when an element is missing. For example, if the model has been defined to have up to three elements and only two are present, the last delimiter can be omitted from the message. • `Never`. Use this option to ensure that even if optional elements are not present, all delimiters are written out. Use this option when the same delimiter is used to delimit parent and child objects. For example, if an optional child element is missing, message processing applications cannot tell where the child elements in a message end and the next parent element starts, if the delimiters are all the same.
Observe Element Length	Check box	This property is applicable when `Data Element Separation` is `All Elements Delimited` or `Tagged Delimited`. Select this check box if the `Length` property of child simple elements is significant when parsing and writing. • During parsing, an exception is thrown if the length of the extracted data exceeds the specified length. Otherwise, the data is trimmed according to the `Justification` and `Padding Character` properties of the child element. • During writing, an exception is thrown if the data to write exceeds the specified length. Otherwise, the data is padded according to the `Justification` and `Padding Character` properties of the child element. Clear this check box to ignore the `Length` property when parsing and writing. The default value depends on the setting of the `Messaging Standard` property (at the message set level) and the `Data Element Separation` property. • If `Data Element Separation` is `All Elements Delimited` and the `Messaging Standard` is `TLOG`, the check box is selected. • If `Data Element Separation` is `All Elements Delimited` and the `Messaging Standard` is other than `TLOG`, the check box is cleared. • If `Data Element Separation` is `Tagged Delimited`, the check box is cleared. For all other data element separation methods, the check box is disabled and does not influence the behavior of the TDS parser.
Tag Data Separator	Button and String	Specify the value of a special character or string that separates the Tag from the data. The `Tag Data Separator` and `Length of Tag` properties are mutually exclusive. If you set the property `Tag Data Separator`, it overrides `Length of Tag`. This property applies only to the tagged `Data Element Separation` methods (`Tagged Delimited`, `Tagged Fixed Length`, and `Tagged Encoded Length`).

Property	Type	Meaning
Length of Tag	Button and Integer	Specify the length of a tag value. When the message is parsed, this property allows tags to be extracted from the bit stream if the `Tag Data Separator` property is not set. The `Tag Data Separator` and `Length of Tag` properties are mutually exclusive. If you set the property `Tag Data Separator`, it overrides this value. This property applies only to the tagged `Data Element Separation` methods (`Tagged Delimited`, `Tagged Fixed Length`, and `Tagged Encoded Length`).
Length of Encoded Length	Integer	Specifies the number of characters (not bytes) after a tag that are used for the length field. Enter a value from 0 to 2147483647. You must set this property if you have set the `Data Element Separation` property to `Tagged Encoded Length`; it is not valid otherwise. The actual number of data characters that are parsed depends on the value of the `Extra Chars in Encoded Length` property.
Extra Chars in Encoded Length	Integer	(Only valid if the `Data Element Separation` method is set to `Tagged Encoded Length`.) Specifies the number of extra characters included in the value found in the length field. (For example, the value in the length might include the size of the length field itself as well as the size of the data field, or it might be the total size of the tag, length, and data fields.) Enter a value from 0 to 2147483647. The parser subtracts this number from the number found in the length field to get the number of data characters that follow the length field. You must set this property if you have set the `Data Element Separation` property to `Tagged Encoded Length`, and the actual number of data characters is less than the value found in the length field.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM TDS format" on page 1221
The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.
"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.
"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.
"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Element reference TDS properties:

The properties, and their permissible values, vary according to the object type.

The properties that are displayed on the object page, and the values that those properties can take, can vary according to the type of the object. For example, the properties for type *string* are different from those of type *Boolean*. Select the link for the object type from the following table.

Type of object	**Properties**
Binary types	• base64Binary • hexBinary
Boolean types	• Boolean
Complex types	
DateTime types	• date • dateTime • gDay • gMonth • gMonthDay • gYear • gYearMonth • time
Decimal types	• decimal • integer • negativeInteger • nonNegativeInteger • nonPositiveInteger • positiveInteger • unsignedLong
Float types	• double • float
Integer types	• byte • int • long • short • unsignedByte • unsignedInt • unsignedShort

Type of object	Properties
Interval types	• duration
String types	• anyURI • ENTITIES • ENTITY • ID • IDREF • IDREFS • language • Name • NCName • NMTOKEN • NMTOKENS • normalizedString • NOTATION • QName • string • token

Note: The physical format properties for simple type *duration* are the same as the physical properties of the string logical types.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM TDS format" on page 1221
The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.
"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.
"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Global attribute TDS properties:

The properties, and their permissible values, vary according to the object type.

The properties that are displayed on the object page, and the values that those properties can take, can vary according to the type of the object. For example, the properties for type *string* are different from those of type *Boolean*. Select the link for the object type from the following table.

Type of object	**Properties**
Binary types	• base64Binary • hexBinary
Boolean types	• Boolean
DateTime types	• date • dateTime • gDay • gMonth • gMonthDay • gYear • gYearMonth • time
Decimal types	• decimal • integer • negativeInteger • nonNegativeInteger • nonPositiveInteger • positiveInteger • unsignedLong
Float types	• double • float
Integer types	• byte • int • long • short • unsignedByte • unsignedInt • unsignedShort
Interval types	• duration

Type of object	Properties
String types	• anyURI • ENTITIES • ENTITY • ID • IDREF • IDREFS • language • Name • NCName • NMTOKEN • NMTOKENS • normalizedString • NOTATION • QName • string • token

Note: The physical format properties for simple type *duration* are the same as the physical properties of the string logical types.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM TDS format" on page 1221
The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Global attribute group TDS properties:

The TDS properties of a global attribute group.

There are no properties to show.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM TDS format" on page 1221
The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Global element TDS properties:

The properties, and their permissible values, vary according to the object type.

The properties that are displayed on the object page, and the values that those properties can take, can vary according to the type of the object. For example, the properties for type *string* are different from those of type *Boolean*. Select the link for the object type from the following table.

Type of object	**Properties**
Binary types	• base64Binary • hexBinary
Boolean types	• Boolean
Complex types	
DateTime types	• date • dateTime • gDay • gMonth • gMonthDay • gYear • gYearMonth • time
Decimal types	• decimal • integer • negativeInteger • nonNegativeInteger • nonPositiveInteger • positiveInteger • unsignedLong
Float types	• double • float
Integer types	• byte • int • long • short • unsignedByte • unsignedInt • unsignedShort
Interval types	• duration

Type of object	Properties
String types	• anyURI • ENTITIES • ENTITY • ID • IDREF • IDREFS • language • Name • NCName • NMTOKEN • NMTOKENS • normalizedString • NOTATION • QName • string • token

Note: The physical format properties for simple type *duration* are the same as the physical properties of the string logical types.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM TDS format" on page 1221
The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Global group TDS properties:

The TDS properties of a global group.

Field Identification

Property	Type	Meaning
Data Element Separation	Enumerated type	Select one of the following values to specify the method that is used to separate the data elements within the type. • `Tagged Delimited`. This value indicates that all elements within the complex type are identified by a tag, and, if a value is specified in the optional `Delimiter` property, are separated by that value. You must set the `Tag` property for all child elements of simple type, and you can set the `Delimiter` property to a non-empty value. See "Global element TDS properties" on page 5513. You must also set the `Tag Data Separator` or the `Length of Tag` property. • `Tagged Fixed Length`. This value indicates that each element is identified by a tag, and the data has a fixed length. There are no delimiters. You must set the `Tag` property for each of the child elements of this complex type, and each child element must have a `Length` or `Length Reference` property assigned to it. You must also set the `Tag Data Separator` or the `Length of Tag` property. • `Tagged Encoded Length`. This value indicates that all elements within the complex type are separated by a tag, and a length field follows each tag. There are no delimiters. The tag can be fixed length, as set by the `Length of Tag` property, or variable length delimited by the `Tag Data Separator` property. You must also set the `Length Of Encoded Length` property so that the parser knows the size of the length field, and set the `Extra Chars in Encoded Length` property. This property tells the parser what to subtract from the value in the `Length Of Encoded Length` property to get the actual length of the data that follows the length field. This method provides a more flexible way of handling ACORD AL3 standard messages than using the `Fixed Length AL3` value, by allowing different parts of the messages to be at different versions of the ACORD AL3 standard. • `All Elements Delimited`. This value indicates that all elements within the complex type are separated by a delimiter. You must set a value in the `Delimiter` property. • `Variable Length Elements Delimited`. This value indicates that some of the elements within the complex type might be of variable length. Variable length elements must be delimited by the value specified in the `Delimiter` property. • `Use Data Pattern`. This value indicates that the parser determines the elements by matching the data with the regular expression that is set in the `Data Pattern` property of the element or type member. See "Message definition file properties" on page 5409. • `Fixed Length`. This value indicates that all elements within the complex type are fixed length. The next data element is accessed by adding the value of the `Length` property to the offset. See "Global element TDS properties" on page 5513. If you set the `Data Element Separation` property of a complex type to `Fixed Length`, you must also set the `Data Element Separation` property of all complex children of this type to `Fixed Length`. Each child element must have a `Length` or `Length Reference` property assigned to it. • `Fixed Length AL3`. This value has a similar meaning to the separation type `Fixed Length`, but also indicates to the parser that a number of predefined rules regarding missing optional elements, encoded lengths, and versioning, must be applied. If you set the `Data Element Separation` property of a complex type to `Fixed Length AL3`, you must also set the `Data Element Separation` property of all complex children of this type to `Fixed Length AL3`. • `Undefined`. This value is set automatically if you set the `Type Composition` property of a complex type to `Message`, and you cannot change it to any other value. Do not set the `Type Composition` property to `Empty`, `Choice`, `Unordered Set`, `Ordered Set`, `Sequence`, or `Simple Unordered Set`. If you do, you cannot check in the type.

Property	Type	Meaning
Group Indicator	String	Specify the value of a special character, or string, that precedes the data that belongs to a group, or a complex type, within the bit stream.
Group Terminator	String	Specify the value of a special character, or string, that terminates the data that belongs to a group, or a complex type, within the bit stream.
Delimiter	String	Specify the value of a special character, or string, that specifies the delimiter that is used between data elements. This property applies only to the delimited `Data Element Separation` methods (`Tagged Delimited`, `All Elements Delimited`, and `Variable Length Elements Delimited`).
Suppress Absent Element Delimiters	Enumerated type	Use this property to select whether you want delimiters to be suppressed for elements that are missing within a message. Select from: • `End Of Type`. Use this option to suppress the delimiter when an element is missing. For example, if the model has been defined to have up to three elements and only two are present, the last delimiter can be omitted from the message. • `Never`. Use this option to ensure that even if optional elements are not present, all delimiters are written out. Use this option when the same delimiter is used to delimit parent and child objects. For example, if an optional child element is missing, message processing applications cannot tell where the child elements in a message end and the next parent element starts, if the delimiters are all the same.
Observe Element Length	Check box	This property is applicable when `Data Element Separation` is `All Elements Delimited` or `Tagged Delimited`. Select this check box if the `Length` property of child simple elements is significant when parsing and writing. • During parsing, an exception is thrown if the length of the extracted data exceeds the specified length. Otherwise, the data is trimmed according to the `Justification` and `Padding Character` properties of the child element. • During writing, an exception is thrown if the data to write exceeds the specified length. Otherwise, the data is padded according to the `Justification` and `Padding Character` properties of the child element. Clear this check box to ignore the `Length` property when parsing and writing. The default value depends on the setting of the `Messaging Standard` property (at the message set level) and the `Data Element Separation` property. • If `Data Element Separation` is `All Elements Delimited` and the `Messaging Standard` is `TLOG`, the check box is selected. • If `Data Element Separation` is `All Elements Delimited` and the `Messaging Standard` is other than `TLOG`, the check box is cleared. • If `Data Element Separation` is `Tagged Delimited`, the check box is cleared. For all other data element separation methods, the check box is disabled and does not influence the behavior of the TDS parser.
Tag Data Separator	Button and String	Specify the value of a special character or string that separates the Tag from the data. The `Tag Data Separator` and `Length of Tag` properties are mutually exclusive. If you set the property `Tag Data Separator`, it overrides `Length of Tag`. This property applies only to the tagged `Data Element Separation` methods (`Tagged Delimited`, `Tagged Fixed Length`, and `Tagged Encoded Length`).

Property	Type	Meaning
Length of Tag	Button and Integer	Specify the length of a tag value. When the message is parsed, this property allows tags to be extracted from the bit stream if the `Tag Data Separator` property is not set. The `Tag Data Separator` and `Length of Tag` properties are mutually exclusive. If you set the property `Tag Data Separator`, it overrides this value. This property applies only to the tagged `Data Element Separation` methods (`Tagged Delimited`, `Tagged Fixed Length`, and `Tagged Encoded Length`).
Length of Encoded Length	Integer	Specifies the number of characters (not bytes) after a tag that are used for the length field. Enter a value from 0 to 2147483647. You must set this property if you have set the `Data Element Separation` property to `Tagged Encoded Length`; it is not valid otherwise. The actual number of data characters that are parsed depends on the value of the `Extra Chars in Encoded Length` property.
Extra Chars in Encoded Length	Integer	(Only valid if the `Data Element Separation` method is set to `Tagged Encoded Length`.) Specifies the number of extra characters included in the value found in the length field. (For example, the value in the length might include the size of the length field itself as well as the size of the data field, or it might be the total size of the tag, length, and data fields.) Enter a value from 0 to 2147483647. The parser subtracts this number from the number found in the length field to get the number of data characters that follow the length field. You must set this property if you have set the `Data Element Separation` property to `Tagged Encoded Length`, and the actual number of data characters is less than the value found in the length field.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM TDS format" on page 1221
The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Group reference TDS properties:

The following tables describe the TDS properties of a group reference.

Field identification

Property	Type	Meaning
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the `Data Element Separation` method is set to `Use Data Pattern` in the complex type. For more details, see "Regular expression syntax" on page 6304.

Occurrences

Property	Type	Meaning
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the `Data Element Separation` property of the parent group or complex type is set to `All Elements Delimited` or `Variable Length Elements Delimited`. A default value is provided if the previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; if the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used. If none of the previous conditions are true, a default is not applied.
Repeat reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, no integer objects exist before this one in the message structure. If a `Repeat Reference` is specified, it overrides any setting for the `Max Occurs` logical property when parsing and writing the message, but not for validating the message.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM TDS format" on page 1221
The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Key TDS properties:

There are no properties to show.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Keyref TDS properties:

There are no properties to show.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Local attribute TDS properties:

The properties, and their permissible values, vary according to the object type.

The properties that are displayed on the object page, and the values that those properties can take, can vary according to the type of the object. For example, the properties for type *string* are different from those of type *Boolean*. Select the link for the object type from the following table.

Binary types	Boolean types	DateTime types	Decimal types
• base64Binary • hexBinary	• Boolean	• date • dateTime • gDay • gMonth • gMonthDay • gYear • gYearMonth • time	• decimal • integer • negativeInteger • nonNegativeInteger • nonPositiveInteger • positiveInteger • unsignedLong

Float types	Integer types	Interval types	String types
• double • float	• byte • int • long • short • unsignedByte • unsignedInt • unsignedShort	• duration	• anyURI • ENTITIES • ENTITY • ID • IDREF • IDREFS • language • Name • NCName • NMTOKEN • NMTOKENS • normalizedString • NOTATION • QName • string • token

Note: The physical format properties for simple type *duration* are the same as the physical properties of the string logical types.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM TDS format" on page 1221
The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Local element TDS properties:

The properties, and their permissible values, vary according to the object type.

The properties that are displayed on the object page, and the values that those properties can take, can vary according to the type of the object. For example, the properties for type *string* are different from those of type *Boolean*. Select the link for the object type from the following table.

Type of object	**Properties**
Binary types	• base64Binary • hexBinary
Boolean types	• Boolean
Complex types	
DateTime types	• date • dateTime • gDay • gMonth • gMonthDay • gYear • gYearMonth • time
Decimal types	• decimal • integer • negativeInteger • nonNegativeInteger • nonPositiveInteger • positiveInteger • unsignedLong
Float types	• double • float
Integer types	• byte • int • long • short • unsignedByte • unsignedInt • unsignedShort
Interval types	• duration

Type of object	Properties
String types	• anyURI • ENTITIES • ENTITY • ID • IDREF • IDREFS • language • Name • NCName • NMTOKEN • NMTOKENS • normalizedString • NOTATION • QName • string • token

Note: The physical format properties for simple type *duration* are the same as the physical properties of the string logical types.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM TDS format" on page 1221
The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Local group TDS properties:

TDS properties of a local group.

Field Identification

Property	Type	Meaning
Data Element Separation	Enumerated type	Select one of the following values to specify the method that is used to separate the data elements within the type. • `Tagged Delimited`. This value indicates that all elements within the complex type are identified by a tag, and, if a value is specified in the optional `Delimiter` property, are separated by that value. You must set the `Tag` property for all child elements of simple type, and you can set the `Delimiter` property to a non-empty value. See "Global element TDS properties" on page 5513. You must also set the `Tag Data Separator` or the `Length of Tag` property. • `Tagged Fixed Length`. This value indicates that each element is identified by a tag, and the data has a fixed length. There are no delimiters. You must set the `Tag` property for each of the child elements of this complex type, and each child element must have a `Length` or `Length Reference` property assigned to it. You must also set the `Tag Data Separator` or the `Length of Tag` property. • `Tagged Encoded Length`. This value indicates that all elements within the complex type are separated by a tag, and a length field follows each tag. There are no delimiters. The tag can be fixed length, as set by the `Length of Tag` property, or variable length delimited by the `Tag Data Separator` property. You must also set the `Length Of Encoded Length` property so that the parser knows the size of the length field, and set the `Extra Chars in Encoded Length` property. This property tells the parser what to subtract from the value in the `Length Of Encoded Length` property to get the actual length of the data that follows the length field. This method provides a more flexible way of handling ACORD AL3 standard messages than using the `Fixed Length AL3` value, by allowing different parts of the messages to be at different versions of the ACORD AL3 standard. • `All Elements Delimited`. This value indicates that all elements within the complex type are separated by a delimiter. You must set a value in the `Delimiter` property. • `Variable Length Elements Delimited`. This value indicates that some of the elements within the complex type might be of variable length. Variable length elements must be delimited by the value specified in the `Delimiter` property. • `Use Data Pattern`. This value indicates that the parser determines the elements by matching the data with the regular expression that is set in the `Data Pattern` property of the element or type member. See "Message definition file properties" on page 5409. • `Fixed Length`. This value indicates that all elements within the complex type are fixed length. The next data element is accessed by adding the value of the `Length` property to the offset. See "Global element TDS properties" on page 5513. If you set the `Data Element Separation` property of a complex type to `Fixed Length`, you must also set the `Data Element Separation` property of all complex children of this type to `Fixed Length`. Each child element must have a `Length` or `Length Reference` property assigned to it. • `Fixed Length AL3`. This value has a similar meaning to the separation type `Fixed Length`, but also indicates to the parser that a number of predefined rules regarding missing optional elements, encoded lengths, and versioning, must be applied. If you set the `Data Element Separation` property of a complex type to `Fixed Length AL3`, you must also set the `Data Element Separation` property of all complex children of this type to `Fixed Length AL3`. • `Undefined`. This value is set automatically if you set the `Type Composition` property of a complex type to `Message`, and you cannot change it to any other value. Do not set the `Type Composition` property to `Empty`, `Choice`, `Unordered Set`, `Ordered Set`, `Sequence`, or `Simple Unordered Set`. If you do, you cannot check in the type.

Property	Type	Meaning
Group Indicator	String	Specify the value of a special character, or string, that precedes the data that belongs to a group, or a complex type, within the bit stream.
Group Terminator	String	Specify the value of a special character, or string, that terminates the data that belongs to a group, or a complex type, within the bit stream.
Delimiter	String	Specify the value of a special character, or string, that specifies the delimiter that is used between data elements. This property applies only to the delimited `Data Element Separation` methods (`Tagged Delimited`, `All Elements Delimited`, and `Variable Length Elements Delimited`).
Suppress Absent Element Delimiters	Enumerated type	Use this property to select whether you want delimiters to be suppressed for elements that are missing within a message. Select from: • `End Of Type`. Use this option to suppress the delimiter when an element is missing. For example, if the model has been defined to have up to three elements and only two are present, the last delimiter can be omitted from the message. • `Never`. Use this option to ensure that even if optional elements are not present, all delimiters are written out. Use this option when the same delimiter is used to delimit parent and child objects. For example, if an optional child element is missing, message processing applications cannot tell where the child elements in a message end and the next parent element starts, if the delimiters are all the same.
Observe Element Length	Check box	This property is applicable when `Data Element Separation` is `All Elements Delimited` or `Tagged Delimited`. Select this check box if the `Length` property of child simple elements is significant when parsing and writing. • During parsing, an exception is thrown if the length of the extracted data exceeds the specified length. Otherwise, the data is trimmed according to the `Justification` and `Padding Character` properties of the child element. • During writing, an exception is thrown if the data to write exceeds the specified length. Otherwise, the data is padded according to the `Justification` and `Padding Character` properties of the child element. Clear this check box to ignore the `Length` property when parsing and writing. The default value depends on the setting of the `Messaging Standard` property (at the message set level) and the `Data Element Separation` property. • If `Data Element Separation` is `All Elements Delimited` and the `Messaging Standard` is `TLOG`, the check box is selected. • If `Data Element Separation` is `All Elements Delimited` and the `Messaging Standard` is other than `TLOG`, the check box is cleared. • If `Data Element Separation` is `Tagged Delimited`, the check box is cleared. For all other data element separation methods, the check box is disabled and does not influence the behavior of the TDS parser.
Tag Data Separator	Button and String	Specify the value of a special character or string that separates the Tag from the data. The `Tag Data Separator` and `Length of Tag` properties are mutually exclusive. If you set the property `Tag Data Separator`, it overrides `Length of Tag`. This property applies only to the tagged `Data Element Separation` methods (`Tagged Delimited`, `Tagged Fixed Length`, and `Tagged Encoded Length`).

Property	Type	Meaning
Length of Tag	Button and Integer	Specify the length of a tag value. When the message is parsed, this property allows tags to be extracted from the bit stream if the `Tag Data Separator` property is not set. The `Tag Data Separator` and `Length of Tag` properties are mutually exclusive. If you set the property `Tag Data Separator`, it overrides this value. This property applies only to the tagged `Data Element Separation` methods (`Tagged Delimited`, `Tagged Fixed Length`, and `Tagged Encoded Length`).
Length of Encoded Length	Integer	Specifies the number of characters (not bytes) after a tag that are used for the length field. Enter a value from 0 to 2147483647. You must set this property if you have set the `Data Element Separation` property to `Tagged Encoded Length`; it is not valid otherwise. The actual number of data characters that are parsed depends on the value of the `Extra Chars in Encoded Length` property.
Extra Chars in Encoded Length	Integer	(Only valid if the `Data Element Separation` method is set to `Tagged Encoded Length`.) Specifies the number of extra characters included in the value found in the length field. (For example, the value in the length might include the size of the length field itself as well as the size of the data field, or it might be the total size of the tag, length, and data fields.) Enter a value from 0 to 2147483647. The parser subtracts this number from the number found in the length field to get the number of data characters that follow the length field. You must set this property if you have set the `Data Element Separation` property to `Tagged Encoded Length`, and the actual number of data characters is less than the value found in the length field.

Field Identification

Property	Type	Meaning
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the `Data Element Separation` method is set to `Use Data Pattern` in the complex type. For more details, see “Regular expression syntax” on page 6304.

Occurrences

Property	Type	Meaning
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the `Data Element Separation` property of the parent group or complex type is set to `All Elements Delimited` or `Variable Length Elements Delimited`. A default value is provided if the previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; if the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used. If none of the previous conditions are true, a default is not applied.

Property	Type	Meaning
Repeat reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, no integer objects exist before this one in the message structure. If a `Repeat Reference` is specified, it overrides any setting for the `Max Occurs` logical property when parsing and writing the message, but not for validating the message.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM TDS format" on page 1221
The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Message TDS properties:

Message TDS properties.

Property	Type	Meaning
Message Key	String	Specify an alternative unique value that identifies the message in the bit stream. This property is required if the message is embedded within another message. **Note:** From Version 6.0 onwards, the use of `Message Key` has been deprecated for identifying an embedded message. You now have the option of identifying an embedded message by Message Identity, using the `Message Alias` logical property.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM TDS format" on page 1221
The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

"Multipart messages" on page 1191
A *multipart message* contains one or more other messages within its structure. The contained message is sometimes referred to as an *embedded message*.

"Identifying an embedded message by using a Message Identity" on page 1193
You can identify an embedded message by using the Message Identity. This technique is used by the MRM domain, and replaces the use of the Message Key.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message logical properties" on page 5449
This section describes the logical properties of a message.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Simple type TDS properties:

There are no properties to show.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM TDS format" on page 1221
The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Unique TDS properties:

There are no properties to show.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

“Working with MRM message model objects” on page 2870
Add, configure, and delete objects.

Related reference:

“Message model object properties” on page 5416
Access property information by property kind, or by object.

“Logical properties for message model objects” on page 5416
Logical property information is available for certain objects.

“Message model object properties by object” on page 5536
The following objects have properties that can be viewed or set.

White space characters in TDS:

White space characters are defined as ASCII characters (hexadecimal) 'X'09 to 'X'0D and EBCDIC characters 'X'05, 'X'0B, 'X'0C, 'X'0D, 'X'25, and 'X'40.

You can specify these characters in your message model using TDS mnemonics if they are important to your processing, for example, to use as group terminators or delimiting characters. See “TDS Mnemonics” on page 5391 for further information.

If both the following conditions are met, white space after the end of a group and preceding the tag of the next element is ignored:

- TDS messaging standard property is "X12" or "EDIFACT"
- TDS data element separation in force for the structure is one of the following types:
  - Tagged delimiter
  - Tagged fixed length
  - Tagged encoded length

The following bit stream is accepted:

```
Tag<data>!<Any white space character>Tag
```

where:

- `!` is the group terminator
- `<Any white space character>` is one of the ASCII or EBCDIC characters listed previously

The following X12 ASCII message successfully parses:

```
ST*856*777777%<SPC><SPC><SPC><HEX 09>BSN*00*7654321*940920*10000%
```

The sequence

```
<SPC><SPC><SPC><HEX 09>
```

is ignored by the parser.

Related concepts:

“The message model” on page 1160
The message model consists of the following components.

Related tasks:

“Working with MRM message model objects” on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Wildcard attribute TDS properties:

There are no properties to show.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM TDS format" on page 1221
The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Wildcard element TDS properties:

There are no properties to show.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM TDS format" on page 1221
The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Documentation properties for all message set objects:

Use the documentation property of an object to add information that describes the function of the object.

The documentation property is available on all objects except Key, Keyref, and Unique objects.

The property is a string field, and you can use any standard alphanumeric characters.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

Message model object properties by object:

The following objects have properties that can be viewed or set.

- "Attribute group reference properties" on page 5537
- "Attribute reference properties" on page 5541
- "Complex type properties" on page 5607
- "Element reference properties" on page 5620
- "Global attribute properties" on page 5697
- "Global attribute group properties" on page 5741
- "Global element properties" on page 5746
- "Global group properties" on page 5800
- "Group reference properties" on page 5810
- "Key properties" on page 5816
- "Keyref properties" on page 5819
- "Local attribute properties" on page 5822
- "Local element properties" on page 5909
- "Local group properties" on page 6014
- "Message properties" on page 6025
- "Simple type properties" on page 6032
- "Unique properties" on page 6057
- "Wildcard attribute properties" on page 6060
- "Wildcard element properties" on page 6064

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

Attribute group reference properties:

Different types of properties are available for an attribute group reference.

An attribute group reference can have the following properties;

- "Attribute group reference logical properties" on page 5417
- "Attribute group reference CWF properties" on page 5456
- "Attribute group reference XML properties" on page 5477
- "Attribute group reference TDS properties" on page 5502
- "Documentation properties for all message set objects" on page 5413

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

Attribute group reference logical properties:

The logical properties of an attribute group reference.

Property	Type	Meaning
Reference Name	Enumerated type	The `Reference Name` is the name of the object that this object is referring to. The objects available to reference can be selected from the list.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Attribute group reference CWF properties:

There are no properties to show.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM Custom Wire Format" on page 1214
Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not separated by delimiters.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Attribute group reference XML properties:

The XML properties of an attribute group reference.

There are no properties to show.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Attribute group reference TDS properties:

There are no properties to show.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM TDS format" on page 1221
The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Documentation properties for all message set objects:

Use the documentation property of an object to add information that describes the function of the object.

The documentation property is available on all objects except Key, Keyref, and Unique objects.

The property is a string field, and you can use any standard alphanumeric characters.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

Attribute reference properties:

Different types of properties are available for an attribute reference.

An attribute reference can have the following properties;

- "Attribute reference logical properties" on page 5418
- "Attribute reference CWF properties" on page 5457
- "Attribute reference XML properties" on page 5478
- "Attribute reference TDS properties" on page 5503
- "Documentation properties for all message set objects" on page 5413

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

Attribute reference logical properties:

The logical properties of an attribute reference.

Property	Type	Meaning
Reference Name	Enumerated type	The `Reference Name` is the name of the object that this object is referring to. The objects available to reference can be selected from the list.

Property	Type	Meaning
Usage	Enumerated type	Use this property with the `Value` property found in an attribute object. The default value for the `Usage` property is optional. Select from the following options: • optional. – If the `Value` property is set to default, and no data has been entered in the `Value` property, the attribute can appear once and can have any value. – If the `Value` property is set to default, the attribute can appear once. If it does not appear, its value is the data that has been entered in the `Value` property. If it does appear, it is the value given. – Where the `Value` property is set to fixed, the attribute can appear once. If it does appear, its value must match the data that has been entered in the `Value` property. If it does not appear, its value is the data that has been entered in the `Value` property. • prohibited. The attribute must not appear. • required. – If the `Value` property is set to default, and no data has been entered in the `Value` property, the attribute must appear once and can have any value. – If the `Value` property is set to fixed, the attribute must appear once, and it must match the data that has been entered in the `Value` property.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Attribute reference CWF properties:

The properties, and their permissible values, vary according to the object type.

The properties that are displayed on the object page, and the values that those properties can take, can vary according to the type of the object. For example, the properties for type *string* are different from those of type *Boolean*. Select the link for the object type from the following table.

Type of object	**Properties**
Binary types	• base64Binary • hexBinary
Boolean types	• Boolean
DateTime types	• date • dateTime • gDay • gMonth • gMonthDay • gYear • gYearMonth • time
Decimal types	• decimal • integer • negativeInteger • nonNegativeInteger • nonPositiveInteger • positiveInteger • unsignedLong
Float types	• double • float
Integer types	• byte • int • long • short • unsignedByte • unsignedInt • unsignedShort
Interval types	• duration

Type of object	Properties
String types	• anyURI • ENTITIES • ENTITY • ID • IDREF • IDREFS • language • Name • NCName • NMTOKEN • NMTOKENS • normalizedString • NOTATION • QName • string • token

Note: The physical format properties for simple type *duration* are the same as the physical properties of the string logical types.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM Custom Wire Format" on page 1214
Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not separated by delimiters.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

CWF properties for attribute reference and local attribute binary types:

CWF properties for attribute reference and local attribute binary types.

The Custom Wire Format properties described here apply to:

- Objects: Attribute Reference, Local Attribute
- Binary schema types: base64Binary, hexBinary

Physical representation

Property	Type	Meaning
Length	Button and Integer	If you have selected the length to be defined by *Length*, enter the number of length units for the element. The minimum value that you can specify is 1. The maximum value that you can specify is 2147483647. The default value is empty (not set).
Length Reference	Button and Enumerated type	If you have selected the length to be defined by *Length Reference*, select the name of the integer object that specifies the length of this object. Make your selection from the displayed list of integer objects that are defined as siblings of the current object, and occur before it in the structure of the message. For information about reordering elements, see "Reordering objects" on page 2900.
Inclusive Length Reference	Check box	This property is applicable only if *Length Reference* is set. If the check box is selected, the value of the sibling integer object that is identified by *Length Reference* is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by *Length Reference* is the length of the current object only. If the check box is selected, the *Length Units* property of the sibling integer object must be the same as that of the current object.

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. The length is given in bytes. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value for the *`Length Count`* or *`Length Reference`* property. The default is `Bytes`.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to the first instance only.

Property	Type	Meaning
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to all instances.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

CWF properties for attribute reference and local attribute Boolean types:

CWF properties for attribute reference and local attribute Boolean types.

The Custom Wire Format properties described here apply to:

- Objects: Attribute Reference, Local Attribute
- Boolean schema types: Boolean

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to all instances.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

CWF properties for attribute reference and local attribute dateTime types:

CWF properties for attribute reference and local attribute dateTime types.

The Custom Wire Format properties described here apply to:

- Objects: Attribute Reference, Local Attribute
- DateTime schema types: date, dateTime, gDay, gMonth, gMonthDay, gYear, gYearMonth, time

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select one from the displayed list: • `Fixed Length String`. The element's length is determined by other length properties as follows. • `Length Encoded String 1`. The first byte of the element contains the length of the string following the length byte in length units. The maximum length of a `Length Encoded String 1` element is 255 length units. • `Length Encoded String 2`. The element's first 2 bytes contain the length of the string following the two length bytes in length units. The maximum length of a `Length Encoded String 2` element is 65535 length units. The two length bytes are in the format of the WebSphere MQ queue manager Encoding. • `Null Terminated String`. The string ends with the hexadecimal NULL character, `X'00'`. • `Packed Decimal`. The dateTime is coded as a Packed Decimal number. It is valid only if the *`DateTime Format`* property represents numeric-only data. • `Binary`. The dateTime is encoded as a binary sequence of bytes. If you select this option, the range of symbols that you can specify for the Format String property is less than the range of symbols you can specify if you select a string option (see "DateTime formats" on page 6310 for details). • `Time Seconds`. This value supports C time_t and Java Date and Time objects. It is valid only if the *`DateTime Format`* property represents numeric-only data. • `Time Milliseconds`. This value supports C time_t and Java Date and Time objects. It is valid only if the *`DateTime Format`* property represents numeric-only data. The default value is `fixed length string`.
DateTime Format	String	Specify a template for date and time. The default dateTime format is dependent on the logical type of the object. For information about the defaults for the dateTime format according to the logical type, see "DateTime defaults by logical type" on page 6320. See "DateTime formats" on page 6310 for details of date and time formats.
Length	Button and Integer	If you have selected a *`Physical Type`* of `Fixed Length String`, `Packed Decimal`, or `Binary`, and have selected the length to be defined by *`Length`*, enter the number of length units for the element. The minimum value that you can specify is 1 for all three physical types. The maximum value that you can specify is `256` for `Fixed Length String`, `10` for `Packed Decimal`, and `2147483647` for `Binary`. The default value is 0 (zero).
Length Reference	Button and Enumerated type	If you have selected the length to be defined by *`Length Reference`*, select the name of the integer object that specifies the length of this object. Make your selection from the displayed list of integer objects that are defined as siblings of the current object, and occur before it in the structure of the message. For information about reordering elements, see "Reordering objects" on page 2900.

Property	Type	Meaning
Inclusive Length Reference	Check box	This property is applicable only if *Length Reference* is set. If the check box is selected, the value of the sibling integer object that is identified by *Length Reference* is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by *Length Reference* is the length of the current object only. If the check box is selected, the *Length Units* property of the sibling integer object must be the same as that of the current object.
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • Bytes. The length is given in bytes. • Characters. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • Character Units. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to Characters. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the Length property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • End of Bitstream. All data up to the end of the bitstream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value for the *Length Count* or *Length Reference* property. The default is Bytes.
Justification	Enumerated type	If you have set the *Physical Type* property to Fixed Length String, select Left Justify (the default value) or Right Justify from the list. If you have selected another value for *Physical Type*, this property is inactive.

Property	Type	Meaning
Padding Character	String	If you have set the *Physical Type* property to `Fixed Length String`, and the *Justification* property is either `Left Justify` or `Right Justify`, this property is applicable. When writing an output message, use the padding character to fill out the remaining character positions when the string length is less than the length implied by the *Length* or *Length Reference* property. Whether the string is padded from the left or the right is governed by the *Justification* property. When parsing an input message, the padding character is trimmed from the end of the string. Whether the string is trimmed from the left or the right is governed by the *Justification* property. Specify this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the displayed list. • Enter a character between quotation marks; for example, `"c"` or `'c'`, where c is any alphanumeric character. • Enter a Unicode value in the form `U+xxxx` where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form `0xYY`, where `YY` is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, it is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Numeric representation

Property	Type	Meaning
Signed	Check box	Specify whether the value is signed. This property is applicable only if the *Physical type* property is `Packed Decimal`. By default, this check box is cleared, which indicates that the value is not signed.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to all instances.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message

models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

CWF properties for attribute reference and local attribute decimal types:

CWF properties for attribute reference and local attribute decimal types.

The Custom Wire Format properties described here apply to:

- Objects: Attribute Reference, Local Attribute
- Decimal schema types: decimal, integer, negativeInteger, nonNegativeInteger, nonPositiveInteger, positiveInteger, unsignedLong

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select one from the displayed list: • `Integer`. This equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • `Packed Decimal`. Equates to the COMP-3 data type in COBOL. • `External Decimal`. Equates to the data type PIC 9 USAGE DISPLAY in COBOL. The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager: • Elements that have *`Physical Type`* set to `Integer` and `Packed Decimal` are represented in the appropriate WebSphere MQ Encoding value. • Elements that have *`Physical Type`* set to `External Decimal` are represented in the WebSphere MQ CodedCharSetId value.
Length	Integer	Enter the number of bytes to specify the element length: • If you set the *`Physical Type`* to `Integer`, select `1`, `2`, or `4` (the default) from the displayed list. • If you set the *`Physical Type`* to `Packed Decimal`, enter a value between 1 and 10. • If you set the *`Physical Type`* to `Extended Decimal`, enter a value between 1 and 256. (Numbers greater than the maximum COBOL PICTURE clause of 18 are assumed to be 18.)

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value for the *`Length Count`*. The default is `Bytes`.
Justification	Enumerated type	If you have set the *`Physical Type`* property to `External Decimal`, select `Left Justify` or `Right Justify` (the default value) from the list. If you have selected another value for *`Physical Type`*, this property is inactive.

Property	Type	Meaning
Padding Character	String	The padding character is used to fill out the remaining character positions when the string length is less than the specified string size. If you have set the *Physical Type* property to Extended Decimal, and the *Justification* property is either Left Justify or Right Justify, specify this character in one of the following ways: • Select NUL, '0', or SPACE from the displayed list. • Enter a character between quotation marks; for example, "c" or 'c', where c is any alphanumeric character. • Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form 0xYY, where YY is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Numeric representation

Property	Type	Meaning
Signed	Check box	Select (the default) or clear this property. This property is used with *Sign Orientation*.

Property	Type	Meaning
Sign EBCDIC Custom Overpunched	Check box	If the *Physical Type* is set to External Decimal and the *Signed EBCDIC Custom* property is set, this indicates that the Sign EBCDIC Custom Overpunched representation is to be used within an ASCII environment. If this check box is not selected (the default), the Sign ASCII representation is used. The setting of the *Sign EBCDIC Custom Overpunched* check box is appropriate only if the *Sign Orientation* property is set to Leading or Trailing (indicating that the element/attribute has an embedded sign representation). The check box is not available if the element/attribute is unsigned (for example, if the *Signed* check box is not set).
Sign Orientation	Enumerated type	If you have set *Physical Type* to External Decimal and you have selected *Signed*, choose from the following options that represent the COBOL options for displaying numeric data: • Leading Overpunched. This option sets a bit in the first byte if the number is negative. No setting is made if the number is positive. For example, the ASCII hexadecimal representation of the number 22 is x'3232'. Using this option, the number +22 would be x'3232' and the number -22 would be x'7232'. The default value. • Leading Separate. This option sets the first byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte. • Trailing Overpunched. This option sets a bit in the last byte if the number is negative. No setting is made if the number is positive. Using this option, the number +22 would be x'3232' and the number -22 would be x'3272'. • Trailing Separate. This option sets the last byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte. If you have set *Physical Type* to any other value, the value Not Applicable is set for you.
Virtual Decimal Point	Integer	Specify the number of places to the left (for a positive value) or right (for a negative value) that a decimal point is to be moved from its assumed position. For example, a decimal element containing 1234 with a Virtual Decimal value of 3 is 1.234, equivalent to 'V' or 'P' in a COBOL picture clause. There is no C equivalent

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • 1 Bytes. The default value. • 2 Bytes • 4 Bytes • 8 Bytes • 16 Bytes
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to the first instance only.

Property	Type	Meaning
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to all instances.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

CWF properties for attribute reference and local attribute float types:

CWF properties for attribute reference and local attribute float types.

The Custom Wire Format properties described here apply to:

- Objects: Attribute Reference, Local Attribute
- Float schema types: double, float

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select one from the displayed list: • `Integer`. This equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • `Float`. This equates to the data type FLOAT or DOUBLE in C or the COMP-1 or COMP-2 data type in COBOL and is the default value. • `Packed Decimal`. Equates to the COMP-3 data type in COBOL. • `External Decimal`. Equates to the data type PIC 9 USAGE DISPLAY in COBOL. The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager: • Elements that have *`Physical Type`* set to `Integer`, `Packed Decimal`, and `Float` are represented in the appropriate WebSphere MQ Encoding value. • Elements that have *`Physical Type`* set to `External Decimal` are represented in the WebSphere MQ CodedCharSetId value.
Length	Integer	Enter the number of bytes to specify the element length: • If you set the *`Physical Type`* to `Float`, select a value from the displayed list. The default value is `8`. • If you set the *`Physical Type`* to `Integer`, select `1`, `2`, or `4` (the default) from the displayed list. • If you set the *`Physical Type`* to `Packed Decimal`, enter a value between 1 and 10. • If you set the *`Physical Type`* to `Extended Decimal`, enter a value between 1 and 256. (Numbers greater than the maximum COBOL PICTURE clause of 18 are assumed to be 18.)

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value for the *`Length Count`*. The default is `Bytes`.
Justification	Enumerated type	If you have set the *`Physical Type`* property to `External Decimal`, select `Left Justify` or `Right Justify` (the default value) from the list. If you have selected another value for *`Physical Type`*, this property is inactive.

Property	Type	Meaning
Padding Character	String	The padding character is used to fill out the remaining character positions when the string length is less than the specified string size. If you have set the *Physical Type* property to Extended Decimal, and the *Justification* property is either Left Justify or Right Justify, specify this character in one of the following ways: • Select NUL, '0', or SPACE from the displayed list. • Enter a character between quotation marks; for example, "c" or 'c', where c is any alphanumeric character. • Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form 0xYY, where YY is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Numeric representation

Property	Type	Meaning
Signed	Check box	Select or clear (unsigned, the default) this property. If you have set *Physical Type* to Float, this is selected. This property is used with *Sign Orientation*.

Property	Type	Meaning
Sign EBCDIC Custom Overpunched	Check box	If the *Physical Type* is set to External Decimal and the *Signed EBCDIC Custom* property is set, this indicates that the Sign EBCDIC Custom Overpunched representation is to be used within an ASCII environment. If this check box is not selected (the default), the Sign ASCII representation is used. The setting of the *Sign EBCDIC Custom Overpunched* check box is appropriate only if the *Sign Orientation* property is set to Leading or Trailing (indicating that the element/attribute has an embedded sign representation). The check box is not available if the element/attribute is unsigned (for example, if the *Signed* check box is not set).
Sign Orientation	Enumerated type	If you have set *Physical Type* to External Decimal and you have selected *Signed*, choose from the following options that represent the COBOL options for displaying numeric data: • Leading Overpunched. This option sets a bit in the first byte if the number is negative. No setting is made if the number is positive. For example, the ASCII hexadecimal representation of the number 22 is x'3232'. Using this option, the number +22 would be x'3232' and the number -22 would be x'7232'. The default value. • Leading Separate. This option sets the first byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte. • Trailing Overpunched. This option sets a bit in the last byte if the number is negative. No setting is made if the number is positive. Using this option, the number +22 would be x'3232' and the number -22 would be x'3272'. • Trailing Separate. This option sets the last byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte. If you have set *Physical Type* to any other value, the value Not Applicable is set for you.
Virtual Decimal Point	Integer	Specify the number of places to the left (for a positive value) or right (for a negative value) that a decimal point is to be moved from its assumed position. For example, a float element containing 1234 with a Virtual Decimal value of 3 is 1.234. This property is not applicable if you have set *Physical Type* to Float.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • 1 Bytes. The default value. • 2 Bytes • 4 Bytes • 8 Bytes • 16 Bytes
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to the first instance only.

Property	Type	Meaning
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to all instances.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

CWF properties for attribute reference and local attribute integer types:

CWF properties for attribute reference and local attribute integer types.

The Custom Wire Format properties described here apply to:

- Objects: Attribute Reference, Local Attribute
- Integer schema types: byte, int, long, short, unsignedByte, unsignedInt, unsignedShort

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select one from the displayed list: • `Integer`. This equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • `Packed Decimal`. Equates to the COMP-3 data type in COBOL. • `External Decimal`. Equates to the data type PIC 9 USAGE DISPLAY in COBOL. The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager: • Elements that have *`Physical Type`* set to `Integer` and `Packed Decimal` are represented in the appropriate WebSphere MQ Encoding value. • Elements that have *`Physical Type`* set to `External Decimal` are represented in the WebSphere MQ CodedCharSetId value.
Length	Integer	Enter the number of bytes to specify the element length: • If you set the *`Physical Type`* to `Integer`, select `1`, `2`, or `4` (the default) from the displayed list. • If you have set *`Physical Type`* to `Packed Decimal`, enter a value between 1 and 6. • If you have set *`Physical Type`* to `Extended Decimal`, enter a value between 1 and 11.

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value for the *`Length Count`*. The default is `Bytes`.
Justification	Enumerated type	If you have set the *`Physical Type`* property to `External Decimal`, select `Left Justify` or `Right Justify` (the default value) from the list. If you have selected another value for *`Physical Type`*, this property is inactive.

Property	Type	Meaning
Padding Character	String	The padding character is used to fill out the remaining character positions when the string length is less than the specified string size. If you have set the *Physical Type* property to Extended Decimal, and the *Justification* property is either Left Justify or Right Justify, specify this character in one of the following ways: • Select NUL, '0', or SPACE from the displayed list. • Enter a character between quotation marks; for example, "c" or 'c', where c is any alphanumeric character. • Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form 0xYY, where YY is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Numeric representation

Property	Type	Meaning
Signed	Check box	Select (the default) or clear this property. This property is used with *Sign Orientation*.

Property	Type	Meaning
Sign EBCDIC Custom Overpunched	Check box	If the *Physical Type* is set to External Decimal and the *Signed EBCDIC Custom* property is set, this indicates that the Sign EBCDIC Custom Overpunched representation is to be used within an ASCII environment. If this check box is not selected (the default), the Sign ASCII representation is used. The setting of the *Sign EBCDIC Custom Overpunched* check box is appropriate only if the *Sign Orientation* property is set to Leading or Trailing (indicating that the element/attribute has an embedded sign representation). The check box is not available if the element/attribute is unsigned (for example, if the *Signed* check box is not set).
Sign Orientation	Enumerated type	If you have set *Physical Type* to External Decimal and you have selected *Signed*, choose from the following options that represent the COBOL options for displaying numeric data: • Leading Overpunched. This option sets a bit in the first byte if the number is negative. No setting is made if the number is positive. For example, the ASCII hexadecimal representation of the number 22 is x'3232'. Using this option, the number +22 would be x'3232' and the number -22 would be x'7232'. The default value. • Leading Separate. This option sets the first byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte. • Trailing Overpunched. This option sets a bit in the last byte if the number is negative. No setting is made if the number is positive. Using this option, the number +22 would be x'3232' and the number -22 would be x'3272'. • Trailing Separate. This option sets the last byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte. If you have set *Physical Type* to any other value, the value Not Applicable is set for you.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • 1 Bytes. The default value. • 2 Bytes • 4 Bytes • 8 Bytes • 16 Bytes
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to the first instance only.

Property	Type	Meaning
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to all instances.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

CWF properties for attribute reference and local attribute string types:

CWF properties for attribute reference and local attribute string types.

The Custom Wire Format properties described here apply to:

- Objects: Attribute Reference, Local Attribute
- String schema types: anyURI, ENTITIES, ENTITY, ID, IDREF, IDREFS, language, Name, NCName, NMTOKEN, NMTOKENS, normalizedString, NOTATION, QName, string, token

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select one from the displayed list: • `Fixed Length String`. The element's length is determined by other length properties as follows. • `Length Encoded String 1`. The first byte of the element contains the length of the string following the length byte in length units. The maximum length of a `Length Encoded String 1` element is 255 length units. • `Length Encoded String 2`. The element's first 2 bytes contain the length of the string following the two length bytes in length units. The maximum length of a `Length Encoded String 2` element is 65535 length units. The two length bytes are in the format of the WebSphere MQ queue manager Encoding. • `Null Terminated String`. The string ends with the hexadecimal NULL character, `X'00'`. The default is `Fixed Length String`.
Length	Button and Integer	If you have selected a *`Physical Type`* of `Fixed Length String` or `Binary`, and have selected the length to be defined by *`Length`*, enter the number of length units for the element. The minimum value that you can specify is `0` (zero), the maximum value that you can specify is `2147483647` The default value is `0` (zero).
Length Reference	Button and Enumerated type	If you have selected the length to be defined by *`Length Reference`*, select the name of the integer object that specifies the length of this object. Make your selection from the displayed list of integer objects that are defined as siblings of the current object, and occur before it in the structure of the message. For information about reordering elements, see “Reordering objects” on page 2900.
Inclusive Length Reference	Check box	This property is applicable only if *`Length Reference`* is set. If the check box is selected, the value of the sibling integer object that is identified by *`Length Reference`* is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by *`Length Reference`* is the length of the current object only. If the check box is selected, the *`Length Units`* property of the sibling integer object must be the same as that of the current object.

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. The length is given in bytes. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value for the *`Length Count`* or *`Length Reference`* property. The default is `Bytes`.
Justification	Enumerated type	If you have set the *`Physical Type`* property to `Fixed Length String`, select `Left Justify` (the default value) or `Right Justify` from the list. If you have selected another value for *`Physical Type`*, this property is inactive.

Property	Type	Meaning
Padding Character	String	If you have set the *Physical Type* property to `Fixed Length String`, and the *Justification* property is either `Left Justify` or `Right Justify`, this property is applicable. When writing an output message, use the padding character to fill out the remaining character positions when the string length is less than the length implied by the *Length* or *Length Reference* property. Whether the string is padded from the left or the right is governed by the *Justification* property. When parsing an input message, the padding character is trimmed from the end of the string. Whether the string is trimmed from the left or the right is governed by the *Justification* property. Specify this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the displayed list. • Enter a character between quotation marks; for example, `"c"` or `'c'`, where c is any alphanumeric character. • Enter a Unicode value in the form `U+xxxx` where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form `0xYY`, where `YY` is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, it is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to all instances.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

Attribute reference XML properties:

The properties, and their permissible values, vary according to the object type.

The properties that are displayed on the object page, and the values that those properties can take, can vary according to the type of the object. For example, the properties for type *string* are different from those of type *Boolean*. Select the link for the object type from the following table.

Type of object	**Properties**
Binary types	• base64Binary • hexBinary
Boolean types	• Boolean
DateTime types	• date • dateTime • gDay • gMonth • gMonthDay • gYear • gYearMonth • time
Decimal types	• decimal • integer • negativeInteger • nonNegativeInteger • nonPositiveInteger • positiveInteger • unsignedLong
Float types	• double • float

Type of object	Properties
Integer types	• byte • int • long • short • unsignedByte • unsignedInt • unsignedShort
Interval types	• duration
String types	• anyURI • ENTITIES • ENTITY • ID • IDREF • IDREFS • language • Name • NCName • NMTOKEN • NMTOKENS • normalizedString • NOTATION • QName • string • token

Note: The physical format properties for simple type *duration* are the same as the physical properties of the string logical types.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

XML properties for attribute reference, element reference, local attribute, local element binary types:

XML wire format properties for attribute reference, element reference, local attribute, and local element binary types.

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- Binary schema types: base64Binary, hexBinary

Field identification

A number of the following properties only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated type	Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list: • `XMLElement`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their *`XML Name`*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object , you must set their *`XML Name`* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in *`ID Attribute Name`* and a value as specified in *`ID Attribute Value`*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *`XML Name`*, *`ID Attribute Name`*, *`ID Attribute Value`* to the same values: – You must also set *`Value Attribute Name`* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in *`Value Attribute Name`*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *`ID Attribute Name`*. The value is the value of *`ID Attribute Value`*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *`XML Name`*, *`ID Attribute Name`*, *`ID Attribute Value`* to the same values: – You must also set *`Value Attribute Name`* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 6262 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
XML Name	String	Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message). This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name. You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name. If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).
ID Attribute Name	String	Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
ID Attribute Value	String	Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

Physical representation

Property	Type	Meaning
Encoding	String	Select one of the following values from the drop-down list: : • `CDatahex` (the default). Hexadecimal values in this field are specified with the CDATA qualifier, for example <e1><![CDATA[62]]></e1> • `hex`. Hexadecimal values in this field are specified as digits only, for example <e1>62</e1>. • `base64`. Values in this field are specified as digits only, coded in base 64.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

XML properties for attribute reference, element reference, local attribute, local element Boolean types:

XML wire format properties for attribute reference, element reference, local attribute and local element Boolean types.

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- Boolean schema types: Boolean

Field identification

A number of the following properties only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated type	Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list: • `XMLElement`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their *XML Name*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object , you must set their *XML Name* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in *ID Attribute Name* and a value as specified in *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in *Value Attribute Name*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *ID Attribute Name*. The value is the value of *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 6262 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
XML Name	String	Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message). This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name. You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name. If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).
ID Attribute Name	String	Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
ID Attribute Value	String	Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

XML properties for attribute reference, element reference, local attribute, local element dateTime types:

XML wire format properties for attribute reference, element reference, local attribute, and local element dateTime types.

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- DateTime schema types: date, dateTime, gDay, gMonth, gMonthDay, gYear, gYearMonth, time

Field identification

A number of the following properties only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated type	Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list: • `XMLElement`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their *`XML Name`*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object , you must set their *`XML Name`* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in *`ID Attribute Name`* and a value as specified in *`ID Attribute Value`*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *`XML Name`*, *`ID Attribute Name`*, *`ID Attribute Value`* to the same values: – You must also set *`Value Attribute Name`* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in *`Value Attribute Name`*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *`ID Attribute Name`*. The value is the value of *`ID Attribute Value`*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *`XML Name`*, *`ID Attribute Name`*, *`ID Attribute Value`* to the same values: – You must also set *`Value Attribute Name`* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 6262 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
XML Name	String	Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message). This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name. You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name. If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).
ID Attribute Name	String	Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
ID Attribute Value	String	Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

Physical representation

Property	Type	Meaning
DateTime Format	String	Specify a format string that specifies the rendering of the value for dateTime elements. The default dateTime format is dependent on the logical type of the object. For information about the defaults for the dateTime format according to the logical type see "DateTime defaults by logical type" on page 6320. See "DateTime formats" on page 6310 for details of dateTime formats.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical

model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

XML properties for attribute reference, element reference, local attribute, local element decimal types:

XML wire format properties for attribute reference, element reference, local attribute, and local element decimal types.

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- Decimal schema types: decimal, integer, negativeInteger, nonNegativeInteger, nonPositiveInteger, positiveInteger, unsignedLong

Field identification

A number of the following properties only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated type	Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list: • `XMLElement`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their *XML Name*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object , you must set their *XML Name* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in *ID Attribute Name* and a value as specified in *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in *Value Attribute Name*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *ID Attribute Name*. The value is the value of *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 6262 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
XML Name	String	Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message). This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name. You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name. If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).
ID Attribute Name	String	Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
ID Attribute Value	String	Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

XML properties for attribute reference, element reference, local attribute, local element float types:

XML wire format properties for attribute reference, element reference, local attribute, and local element float types.

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- Float schema types: double, float

Field identification

A number of the following properties only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated type	Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list: • `XMLElement`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their *`XML Name`*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object , you must set their *`XML Name`* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in *`ID Attribute Name`* and a value as specified in *`ID Attribute Value`*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *`XML Name`*, *`ID Attribute Name`*, *`ID Attribute Value`* to the same values: – You must also set *`Value Attribute Name`* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in *`Value Attribute Name`*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *`ID Attribute Name`*. The value is the value of *`ID Attribute Value`*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *`XML Name`*, *`ID Attribute Name`*, *`ID Attribute Value`* to the same values: – You must also set *`Value Attribute Name`* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 6262 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
XML Name	String	Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message). This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name. You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name. If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).
ID Attribute Name	String	Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
ID Attribute Value	String	Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

XML properties for attribute reference, element reference, local attribute, local element integer types:

XML wire format properties for attribute reference, element reference, local attribute, and local element integer types.

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- Integer schema types: byte, int, long, short, unsignedByte, unsignedInt, unsignedShort

Field identification

A number of the following properties only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated type	Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list: • `XMLElement`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their *XML Name*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object , you must set their *XML Name* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in *ID Attribute Name* and a value as specified in *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in *Value Attribute Name*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *ID Attribute Name*. The value is the value of *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 6262 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
XML Name	String	Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message). This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name. You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name. If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).
ID Attribute Name	String	Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
ID Attribute Value	String	Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

XML properties for attribute reference, element reference, local attribute, local element string types:

XML wire format properties for attribute reference, element reference, local attribute, and local element string types.

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- String schema types: anyURI, ENTITIES, ENTITY, ID, IDREF, IDREFS, language, Name, NCName, NMTOKEN, NMTOKENS, normalizedString, NOTATION, QName, string, token

Field identification

A number of the following properties only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated type	Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list: • `XMLElement`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their *`XML Name`*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object , you must set their *`XML Name`* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in *`ID Attribute Name`* and a value as specified in *`ID Attribute Value`*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *`XML Name`*, *`ID Attribute Name`*, *`ID Attribute Value`* to the same values: – You must also set *`Value Attribute Name`* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in *`Value Attribute Name`*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *`ID Attribute Name`*. The value is the value of *`ID Attribute Value`*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *`XML Name`*, *`ID Attribute Name`*, *`ID Attribute Value`* to the same values: – You must also set *`Value Attribute Name`* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 6262 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
XML Name	String	Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message). This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name. You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name. If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).
ID Attribute Name	String	Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
ID Attribute Value	String	Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

Attribute reference TDS properties:

The properties, and their permissible values, vary according to the object type.

The properties that are displayed on the object page, and the values that those properties can take, can vary according to the type of the object. For example, the properties for type *string* are different from those of type *Boolean*. Select the link for the object type from the following table.

Type of object	**Properties**
Binary types	• base64Binary • hexBinary
Boolean types	• Boolean
DateTime types	• date • dateTime • gDay • gMonth • gMonthDay • gYear • gYearMonth • time
Decimal types	• decimal • integer • negativeInteger • nonNegativeInteger • nonPositiveInteger • positiveInteger • unsignedLong
Float types	• double • float

Type of object	Properties
Integer types	• byte • int • long • short • unsignedByte • unsignedInt • unsignedShort
Interval types	• duration
String types	• anyURI • ENTITIES • ENTITY • ID • IDREF • IDREFS • language • Name • NCName • NMTOKEN • NMTOKENS • normalizedString • NOTATION • QName • string • token

Note: The physical format properties for simple type *duration* are the same as the physical properties of the string logical types.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM TDS format" on page 1221
The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

TDS properties for attribute reference binary types:

The TDS wire format properties for attribute reference binary types.

The TDS Format properties described here apply to:

- Objects: Attribute Reference
- Binary schema types: base64Binary, hexBinary

Property	Type	Meaning
Length Reference	Enumerated type	This property is applicable only if `Physical Type` is `Text`, `Binary`, or `TLOG Specific`. If set, this property takes precedence over any value in the `Length Units` property. Specify the identifier of a sibling integer object, the value of which determines the length of the object in question. The sibling object must be defined before the current object within the message structure. For information about reordering elements, see "Reordering objects" on page 2900.
Inclusive Length Reference	Check box	This property is applicable only if `Length Reference` is set. If the check box is selected, the value of the sibling integer object that is identified by `Length Reference` is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by `Length Reference` is the length of the current object only. If the check box is selected, the `Length Units` property of the sibling integer object must be the same as that of the current object.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

TDS properties for attribute reference Boolean types:

There are no properties to show.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

TDS properties for attribute reference dateTime types:

The TDS wire format properties for attribute reference dateTime types.

The TDS Format properties described here apply to:

- Objects: Attribute Reference
- DateTime schema types: date, dateTime, gDay, gMonth, gMonthDay, gYear, gYearMonth, time

Property	Type	Meaning
Length Reference	Enumerated type	This property is applicable only if `Physical Type` is `Text`, `Binary`, or `TLOG Specific`. If set, this property takes precedence over any value in the `Length Units` property. Specify the identifier of a sibling integer object, the value of which determines the length of the object in question. The sibling object must be defined before the current object within the message structure. For information about reordering elements, see "Reordering objects" on page 2900.
Inclusive Length Reference	Check box	This property is applicable only if `Length Reference` is set. If the check box is selected, the value of the sibling integer object that is identified by `Length Reference` is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by `Length Reference` is the length of the current object only. If the check box is selected, the `Length Units` property of the sibling integer object must be the same as that of the current object.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

TDS properties for attribute reference decimal types:

The TDS wire format properties for attribute reference reference decimal types.

The TDS Format properties described here apply to:

- Objects: Attribute Reference
- Decimal schema types: decimal, integer, negativeInteger, nonNegativeInteger, nonPositiveInteger, positiveInteger, unsignedLong

Property	Type	Meaning
Length Reference	Enumerated type	This property is applicable only if `Physical Type` is `Text`, `Binary`, or `TLOG Specific`. If set, this property takes precedence over any value in the `Length Units` property. Specify the identifier of a sibling integer object, the value of which determines the length of the object in question. The sibling object must be defined before the current object within the message structure. For information about reordering elements, see "Reordering objects" on page 2900.

Property	Type	Meaning
Inclusive Length Reference	Check box	This property is applicable only if `Length Reference` is set. If the check box is selected, the value of the sibling integer object that is identified by `Length Reference` is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by `Length Reference` is the length of the current object only. If the check box is selected, the `Length Units` property of the sibling integer object must be the same as that of the current object.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

TDS properties for attribute reference float types:

The TDS wire format properties for attribute reference float types.

The TDS Format properties described here apply to:

- Objects: Attribute Reference
- Float schema types: double, float

Property	Type	Meaning
Length Reference	Enumerated type	This property is applicable only if `Physical Type` is `Text`, `Binary`, or `TLOG Specific`. If set, this property takes precedence over any value in the `Length Units` property. Specify the identifier of a sibling integer object, the value of which determines the length of the object in question. The sibling object must be defined before the current object within the message structure. For information about reordering elements, see "Reordering objects" on page 2900.
Inclusive Length Reference	Check box	This property is applicable only if `Length Reference` is set. If the check box is selected, the value of the sibling integer object that is identified by `Length Reference` is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by `Length Reference` is the length of the current object only. If the check box is selected, the `Length Units` property of the sibling integer object must be the same as that of the current object.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

TDS properties for attribute reference integer types:

The TDS wire format properties for attribute reference integer types.

The TDS Format properties described here apply to:

- Objects: Attribute Reference
- Integer schema types: byte, int, long, short, unsignedByte, unsignedInt, unsignedShort

Property	Type	Meaning
Length Reference	Enumerated type	This property is applicable only if `Physical Type` is `Text`, `Binary`, or `TLOG Specific`. If set, this property takes precedence over any value in the `Length Units` property. Specify the identifier of a sibling integer object, the value of which determines the length of the object in question. The sibling object must be defined before the current object within the message structure. For information about reordering elements, see "Reordering objects" on page 2900.
Inclusive Length Reference	Check box	This property is applicable only if `Length Reference` is set. If the check box is selected, the value of the sibling integer object that is identified by `Length Reference` is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by `Length Reference` is the length of the current object only. If the check box is selected, the `Length Units` property of the sibling integer object must be the same as that of the current object.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as

group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

TDS properties for attribute reference string types:

The TDS wire format properties for attribute reference string types.

The TDS Format properties described here apply to:

- Objects: Attribute Reference
- String schema types: anyURI, ENTITIES, ENTITY, ID, IDREF, IDREFS, language, Name, NCName, NMTOKEN, NMTOKENS, normalizedString, NOTATION, QName, string, token

Property	Type	Meaning
Length Reference	Enumerated type	This property is applicable only if `Physical Type` is `Text`, `Binary`, or `TLOG Specific`. If set, this property takes precedence over any value in the `Length Units` property. Specify the identifier of a sibling integer object, the value of which determines the length of the object in question. The sibling object must be defined before the current object within the message structure. For information about reordering elements, see "Reordering objects" on page 2900.

Property	Type	Meaning
Inclusive Length Reference	Check box	This property is applicable only if `Length Reference` is set. If the check box is selected, the value of the sibling integer object that is identified by `Length Reference` is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by `Length Reference` is the length of the current object only. If the check box is selected, the `Length Units` property of the sibling integer object must be the same as that of the current object.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

Documentation properties for all message set objects:

Use the documentation property of an object to add information that describes the function of the object.

The documentation property is available on all objects except Key, Keyref, and Unique objects.

The property is a string field, and you can use any standard alphanumeric characters.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

Complex type properties:

Different types of properties are available for a complex type.

A complex type can have the following properties;

- "Complex type logical properties" on page 5419
- "Complex type CWF properties" on page 5459
- "Complex type XML properties" on page 5480
- "Complex type TDS properties" on page 5505
- "Documentation properties for all message set objects" on page 5413

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

Complex type logical properties:

The logical properties of a complex type include properties that describe content and substitution settings.

Property	Type	Meaning
Name	String	Specify a name for the object when you create it. Names can consist of alphanumeric characters, including the letters *A* through *Z*, *a* through *z*, and the digits *0* through *9*. They might also include the following punctuation characters; • - the hyphen • _ the underscore • . the period Names can start only with a letter or the underscore character, and not with a number, hyphen, or period. Names that begin with `xml`, or any variant of these characters (for example `XmL`), are reserved by the XML standards specification. Further details of naming conventions and allowable characters can be found in the Extensible Markup Language (XML) specification that can be found on the World Wide Web Consortium (W3C) Web site.
Base Type	Enumerated type	You can use this property to select a type (simple or complex) that is used as the starting point to define a new complex type that is derived by restriction or extension.
Derived By	Enumerated type	If this property is active, select one of the following options: • restriction. If a complex type is derived by restriction, the content model of the complex type is a subset of the base type. • extension. If the complex type is derived by extension, the content model of the complex type is the content model of the base type plus the content model specified in the type derivation. Derivation by list or union is not supported.

Content

The following table shows the valid settings for `Composition` and `Content Validation`. These properties are located on the group which defines the content of this type. They can be edited only if the `Local group` button is selected. If the `Global group` button is selected, these properties are taken from the global group identified by the `Group name` field.

Valid children in a complex type that depend on both `Composition` and `Content Validation` are shown in "MRM content validation" on page 5422.

Property	Type	Meaning
Local Group	Button	Select this property if the content of your complex type is a local group.
Composition	Enumerated type	Define the order, and the number of occurrences, of the elements and groups in your messages. Composition does not affect the attributes in a complex type. Select one of the following options: • Empty • sequence. If you select this option, you can define members that are elements or groups. These members, if present, must appear in the specified order in the message. They can repeat, and the same element or group can appear more than once. • choice. If you select this option, you can define members that are elements or groups. Exactly one of the defined members must be present in the message, and can repeat. Use this option if you want to model C unions and COBOL REDEFINES in a Custom Wire Format, or an XML DTD element that uses choice in an XML Wire Format, or a SWIFT field that has more than one option. • all. If you select this option, you can define members that are elements; groups are not allowed. The elements in an all group can appear in any order. Each element can appear once, or not at all. An all group can be used only at the top level of a complex type; it cannot be a member of another group within a type. • unorderedSet. This option is supported only by the MRM domain.If you select this option, you can define members that are elements. The elements can repeat, but the same element cannot appear twice in the list of members. The elements can appear in any order in the message. • orderedSet. This option is supported only by the MRM domain.If you select this option, you can define members that are elements. The elements can repeat, but the same element cannot appear twice in the list of members. The elements must appear in the specified order in the message. • message. This option is supported only by the MRM domain.If you select this option, you can define only messages as members. Each member can repeat, but the same message cannot appear twice in the list of members. Like choice, only one of the defined members can be present in a message. Unlike choice, when writing a message, if the complex type or group has more than one member, the bit stream is not padded to the length of the longest member. Use this option to model multipart messages, which are used in some industry standards; for example, SWIFT. For more information, see the section on multipart messages in "Multipart messages" on page 1191.

Property	Type	Meaning
Content Validation	Enumerated type	`Content Validation` is used only by the MRM domain. If validation is enabled in your message flow, `Content Validation` specifies the strictness of the MRM validation for members of a complex type or group. See "MRM content validation" on page 5422 for further details. Select from the following options: • Closed. The complex type can only contain the child elements that you have added to it. • Open Defined. The complex type can contain any valid element defined within the message set. • Open. The complex type can contain any valid element, not just those that you have added to this complex type. See "Combinations of Composition and Content Validation" on page 5612 for further details of these options.
Group Reference	Button	Select this option if the content of your complex type is a reference to a group object
Group Name	Enumerated type	The `Group Name` is the name of the group that this complex type refers to. The groups available to be referenced can be selected from the drop down list.
Min Occurs	Integer	Specify the minimum number of times that the object can repeat. The default value is 1. If the value is set to 0, the object is optional. With the exception of `Max Occurs` being set to -1, if a value is set for `Min Occurs`, it must be less than or equal to the value in `Max Occurs`.
Max Occurs	Integer	Specify the maximum number of times that the object can repeat. The default value is 1. If this property is not set, the object cannot occur more than once. If this property is set to 0, it is interpreted as if the object does not exist in the message. It can also be set to -1, to indicate that the limit is unbounded and there is no maximum to the number of occurrences.
Mixed	Check box	Select this option when the complex type has mixed content, and contains character data and sub-elements.

Substitution settings

Property	Type	Meaning
Final	Multiple selection enumerated type	The final attribute on a complex type controls whether other types can be derived from it. Valid values are extension/restriction/all. You can select from one or more of the following: • Empty • `restriction`. Prohibit type substitution by elements whose types are restrictions of the head element's type. • `extension`. Prohibit type substitution by elements whose types are extensions of the head element's type. • `#all`. Prohibit substitution by any method. To select more than one, you must type the selection into the property field.

Property	Type	Meaning
Block	Multiple selection enumerated type	The block attribute on a complex type restricts the types of substitutions which are allowed for elements based on that type. In the WebSphere Message Broker its effect is the same as if the block attribute were copied from the complex type onto every element based on the complex type. You can select from one or more of the following: • Empty • `restriction`. Prohibit type substitution by elements whose types are restrictions of the head element's type. • `extension`. Prohibit type substitution by elements whose types are extensions of the head element's type. • `#all`. Prohibit substitution by any method. To select more than one, you must type the selection into the property field.
Abstract	Check box	If selected, no elements based on this type can appear in the message.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

MRM content validation:

`Content Validation` is applied when the domain is MRM and validation is enabled. The `Content Validation` property specifies how strictly the MRM parser validates the members of a complex type or group.

The first of the following two tables shows the valid settings for `Content Validation` if `Composition` is set to `Message`. The second table shows the valid settings for `Content Validation` if `Composition` is not set to `Message`.

Content Validation options if Composition is set to Message

Option	Meaning
Open	When a message is parsed, this complex type or group can contain any message, not just those that you have defined in this message set. You can use this option for sparse messages (see "Self-defining elements and messages" on page 1198 for a definition of sparse messages).
Closed	When a message is parsed, this complex type or group can only contain the messages that are members of this complex type or group. This is always the case for messages represented in CWF format.
Open Defined	When a message is parsed, this complex type or group can contain any message defined within the message set.

Content Validation options if Composition is not set to Message

Option	Meaning
Open	When a message is parsed, this complex type or group can contain any elements and not just those that you have defined in this message set (see "Self-defining elements and messages" on page 1198 for a definition of sparse messages).
Closed	When a message is parsed, this complex type or group can only contain the elements that are members of this complex type or group.
Open Defined	When a message is parsed, this complex type or group can contain any element that you have defined within the message set.

When you are using `Content Validation` set to `open` or `open defined`, you cannot specify the precise position where the content that is not modeled is permitted to occur. If you want to do this, you should consider using a wildcard element as an alternative. Wildcard elements can appear only within a complex type or group with `Composition` of `sequence` and `Content Validation` of `closed`.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Complex type logical properties" on page 5419
The logical properties of a complex type include properties that describe content and substitution settings.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Combinations of Composition and Content Validation:

If your message is in the MRM domain, and validation is enabled, the members of each complex type or group are validated. The MRM validation logic is controlled by the `Composition` and `Content Validation` properties.

Content validation applies also to the IDOC domain because the IDoc parser uses the MRM parser internally. Content Validation does not affect validation in the XMLNSC domain.

Valid children in complex types dependent on Composition and Content Validation

Composition	Content Validation	Valid children
Empty	Closed	None
Empty	Open	None
Empty	Open Defined	None
Sequence	Open	Elements, group references, embedded simple types

Composition	Content Validation	Valid children
Sequence	Closed	Elements, group references, embedded simple types
Sequence	Open Defined	Elements, group references, embedded simple types
Choice	Closed	Elements, group references, embedded simple types
All	Closed	Elements
All	Open	Elements
All	Open Defined	Elements
Unordered Set	Open	Elements
Unordered Set	Closed	Elements
Unordered Set	Open Defined	Elements
Ordered Set	Open	Elements
Ordered Set	Closed	Elements
Ordered Set	Open Defined	Elements
Message	Open	Messages
Message	Closed	Messages
Message	Open Defined	Messages

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Complex type logical properties" on page 5419
The logical properties of a complex type include properties that describe content and substitution settings.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Valid combinations of repeat and duplicate elements in complex types:

Valid combinations of repeated and duplicate elements within a complex type depend on the `Composition` property value.

- A repeated element is an element that is included once within the complex type, and is defined with the property `Min Occurs` set to greater than 1. Repeated elements are therefore always contiguous and are always specified in the form A[n].
- A duplicate element is an element included more than once anywhere within the complex type. Duplicate elements do not have to be contiguous.

Repeated and duplicate elements in a complex type

Elements in type	Example	Unordered Set	Ordered Set	Sequence
No repeats, no duplicates	A, B, C	Yes	Yes	Yes
Repeated element (contiguous)	A[n], B, C	Yes	Yes	Yes
Duplicate element A (contiguous)	A, A, B, C	No	No	Yes
Duplicate element A (non-contiguous)	A, B, C, A	No	No	Yes

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Complex type logical properties" on page 5419
The logical properties of a complex type include properties that describe content and substitution settings.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Complex type CWF properties:

There are no properties to show.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM Custom Wire Format" on page 1214
Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not separated by delimiters.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Complex type XML properties:

There are no properties to show.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Complex type TDS properties:

The TDS properties of a complex type.

Field Identification

If the complex type is based on a global group, the TDS properties listed are located in the global group. If so, any changes to these properties are applied to the global group, and affect all references to the group (including any other complex types which are based on it).

Property	Type	Meaning
Data Element Separation	Enumerated type	Select one of the following values to specify the method that is used to separate the data elements within the type. • `Tagged Delimited`. This value indicates that all elements within the complex type are identified by a tag, and, if a value is specified in the optional `Delimiter` property, are separated by that value. You must set the `Tag` property for all child elements of simple type, and you can set the `Delimiter` property to a non-empty value. See "Global element TDS properties" on page 5513. You must also set the `Tag Data Separator` or the `Length of Tag` property. • `Tagged Fixed Length`. This value indicates that each element is identified by a tag, and the data has a fixed length. There are no delimiters. You must set the `Tag` property for each of the child elements of this complex type, and each child element must have a `Length` or `Length Reference` property assigned to it. You must also set the `Tag Data Separator` or the `Length of Tag` property. • `Tagged Encoded Length`. This value indicates that all elements within the complex type are separated by a tag, and a length field follows each tag. There are no delimiters. The tag can be fixed length, as set by the `Length of Tag` property, or variable length delimited by the `Tag Data Separator` property. You must also set the `Length Of Encoded Length` property so that the parser knows the size of the length field, and set the `Extra Chars in Encoded Length` property. This property tells the parser what to subtract from the value in the `Length Of Encoded Length` property to get the actual length of the data that follows the length field. This method provides a more flexible way of handling ACORD AL3 standard messages than using the `Fixed Length AL3` value, by allowing different parts of the messages to be at different versions of the ACORD AL3 standard. • `All Elements Delimited`. This value indicates that all elements within the complex type are separated by a delimiter. You must set a value in the `Delimiter` property. • `Variable Length Elements Delimited`. This value indicates that some of the elements within the complex type might be of variable length. Variable length elements must be delimited by the value specified in the `Delimiter` property. • `Use Data Pattern`. This value indicates that the parser determines the elements by matching the data with the regular expression that is set in the `Data Pattern` property of the element or type member. See "Message definition file properties" on page 5409. • `Fixed Length`. This value indicates that all elements within the complex type are fixed length. The next data element is accessed by adding the value of the `Length` property to the offset. See "Global element TDS properties" on page 5513. If you set the `Data Element Separation` property of a complex type to `Fixed Length`, you must also set the `Data Element Separation` property of all complex children of this type to `Fixed Length`. Each child element must have a `Length` or `Length Reference` property assigned to it. • `Fixed Length AL3`. This value has a similar meaning to the separation type `Fixed Length`, but also indicates to the parser that a number of predefined rules regarding missing optional elements, encoded lengths, and versioning, must be applied. If you set the `Data Element Separation` property of a complex type to `Fixed Length AL3`, you must also set the `Data Element Separation` property of all complex children of this type to `Fixed Length AL3`. • `Undefined`. This value is set automatically if you set the `Type Composition` property of a complex type to `Message`, and you cannot change it to any other value. Do not set the `Type Composition` property to `Empty`, `Choice`, `Unordered Set`, `Ordered Set`, `Sequence`, or `Simple Unordered Set`. If you do, you cannot check in the type.

Property	Type	Meaning
Group Indicator	String	Specify the value of a special character, or string, that precedes the data that belongs to a group, or a complex type, within the bit stream.
Group Terminator	String	Specify the value of a special character, or string, that terminates the data that belongs to a group, or a complex type, within the bit stream.
Delimiter	String	Specify the value of a special character, or string, that specifies the delimiter that is used between data elements. This property applies only to the delimited `Data Element Separation` methods (`Tagged Delimited`, `All Elements Delimited`, and `Variable Length Elements Delimited`).
Suppress Absent Element Delimiters	Enumerated type	Use this property to select whether you want delimiters to be suppressed for elements that are missing within a message. Select from: • `End Of Type`. Use this option to suppress the delimiter when an element is missing. For example, if the model has been defined to have up to three elements and only two are present, the last delimiter can be omitted from the message. • `Never`. Use this option to ensure that even if optional elements are not present, all delimiters are written out. Use this option when the same delimiter is used to delimit parent and child objects. For example, if an optional child element is missing, message processing applications cannot tell where the child elements in a message end and the next parent element starts, if the delimiters are all the same.
Observe Element Length	Check box	This property is applicable when `Data Element Separation` is `All Elements Delimited` or `Tagged Delimited`. Select this check box if the `Length` property of child simple elements is significant when parsing and writing. • During parsing, an exception is thrown if the length of the extracted data exceeds the specified length. Otherwise, the data is trimmed according to the `Justification` and `Padding Character` properties of the child element. • During writing, an exception is thrown if the data to write exceeds the specified length. Otherwise, the data is padded according to the `Justification` and `Padding Character` properties of the child element. Clear this check box to ignore the `Length` property when parsing and writing. The default value depends on the setting of the `Messaging Standard` property (at the message set level) and the `Data Element Separation` property. • If `Data Element Separation` is `All Elements Delimited` and the `Messaging Standard` is `TLOG`, the check box is selected. • If `Data Element Separation` is `All Elements Delimited` and the `Messaging Standard` is other than `TLOG`, the check box is cleared. • If `Data Element Separation` is `Tagged Delimited`, the check box is cleared. For all other data element separation methods, the check box is disabled and does not influence the behavior of the TDS parser.
Tag Data Separator	Button and String	Specify the value of a special character or string that separates the Tag from the data. The `Tag Data Separator` and `Length of Tag` properties are mutually exclusive. If you set the property `Tag Data Separator`, it overrides `Length of Tag`. This property applies only to the tagged `Data Element Separation` methods (`Tagged Delimited`, `Tagged Fixed Length`, and `Tagged Encoded Length`).

Property	Type	Meaning
Length of Tag	Button and Integer	Specify the length of a tag value. When the message is parsed, this property allows tags to be extracted from the bit stream if the `Tag Data Separator` property is not set. The `Tag Data Separator` and `Length of Tag` properties are mutually exclusive. If you set the property `Tag Data Separator`, it overrides this value. This property applies only to the tagged `Data Element Separation` methods (`Tagged Delimited`, `Tagged Fixed Length`, and `Tagged Encoded Length`).
Length of Encoded Length	Integer	Specifies the number of characters (not bytes) after a tag that are used for the length field. Enter a value from 0 to 2147483647. You must set this property if you have set the `Data Element Separation` property to `Tagged Encoded Length`; it is not valid otherwise. The actual number of data characters that are parsed depends on the value of the `Extra Chars in Encoded Length` property.
Extra Chars in Encoded Length	Integer	(Only valid if the `Data Element Separation` method is set to `Tagged Encoded Length`.) Specifies the number of extra characters included in the value found in the length field. (For example, the value in the length might include the size of the length field itself as well as the size of the data field, or it might be the total size of the tag, length, and data fields.) Enter a value from 0 to 2147483647. The parser subtracts this number from the number found in the length field to get the number of data characters that follow the length field. You must set this property if you have set the `Data Element Separation` property to `Tagged Encoded Length`, and the actual number of data characters is less than the value found in the length field.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM TDS format" on page 1221
The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Documentation properties for all message set objects:

Use the documentation property of an object to add information that describes the function of the object.

The documentation property is available on all objects except Key, Keyref, and Unique objects.

The property is a string field, and you can use any standard alphanumeric characters.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

Element reference properties:

Different types of properties are available for an element reference.

An element reference can have the following properties:

- "Element reference logical properties" on page 5423
- "Element reference CWF properties" on page 5460
- "Element reference XML properties" on page 5481
- "Element reference TDS properties" on page 5509
- "Documentation properties for all message set objects" on page 5413

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

Element reference logical properties:

The logical properties of an element reference include properties that specify the number of occurrences of the element reference.

Property	Type	Meaning
Reference Name	Enumerated type	The `Reference Name` is the name of the object that this object is referring to. The objects available to reference can be selected from the list.

Occurrences

Property	Type	Meaning
Min Occurs	Integer	Specify the minimum number of times that the object can repeat. The default value is 1. If the value is set to 0, the object is optional. With the exception of `Max Occurs` being set to -1, if a value is set for `Min Occurs`, it must be less than or equal to the value in `Max Occurs`.

Property	Type	Meaning
Max Occurs	Integer	Specify the maximum number of times that the object can repeat. The default value is 1. If this property is not set, the object cannot occur more than once. If this property is set to 0, it is interpreted as if the object does not exist in the message. It can also be set to -1, to indicate that the limit is unbounded and there is no maximum to the number of occurrences.

The Min Occurs and Max Occurs properties are used in conjunction with an element's Value properties. The following table summarizes how an element reference can be constrained.

Min Occurs	Max Occurs	Fixed	Default	Notes
1	1			The element must appear once, and can have any value.
1	1	Delta		The element must appear once, and it must match the data that has been entered in the `Value` property. In this example, the element must contain the text Delta.
2	-1	Delta		The element must appear twice or more, and it must match the data that has been entered in the `Value` property. In this example, at least two elements must contain the text Delta.
0	1			The element is optional, can appear once, and can have any value.
0	1	Delta		The element is optional, and can appear once. If it does appear, its value must match the data that has been entered in the `Value` property. If it does not appear, its value is the data that has been entered in the `Value` property.
0	1		Delta	The element is optional, and can appear once. If it does not appear, its value is the data that has been entered in the `Value` property. If it does appear, it must be the value given in the element.
0	2		Delta	The element is optional and can appear once, twice, or not at all. If the element does not appear, it is not provided. If the element appears and it is empty, it set to the data held in the `Value` property, else it is the value given in the element.
0	0			The element is prohibited, and must not appear.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Element reference CWF properties:

The properties, and permissible values, vary according to the type of object.

The properties that are displayed on the object page, and the values that those properties can take, can vary according to the type of the object. For example, the properties for type *string* are different from those of type *Boolean*. Select the link for the object type from the following table.

Type of object	Properties
Binary types	• base64Binary • hexBinary
Boolean types	• Boolean
DateTime types	• date • dateTime • gDay • gMonth • gMonthDay • gYear • gYearMonth • time
Decimal types	• decimal • integer • negativeInteger • nonNegativeInteger • nonPositiveInteger • positiveInteger • unsignedLong
Float types	• double • float
Integer types	• byte • int • long • short • unsignedByte • unsignedInt • unsignedShort
Interval types	• duration

Type of object	Properties
String types	• anyURI • ENTITIES • ENTITY • ID • IDREF • IDREFS • language • Name • NCName • NMTOKEN • NMTOKENS • normalizedString • NOTATION • QName • string • token

Note: The physical format properties for simple type *duration* are the same as the physical properties of the string logical types.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM Custom Wire Format" on page 1214
Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not separated by delimiters.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

CWF properties for element reference and local element binary types:

CWF wire format properties for element reference and local element binary types.

The Custom Wire Format properties described here apply to:

- Objects: Element Reference, Local Element
- Binary schema types: base64Binary, hexBinary

Physical representation

Property	Type	Meaning
Length	Button and Integer	If you have selected the length to be defined by *Length*, enter the number of length units for the element. The minimum value that you can specify is 1. The maximum value that you can specify is 2147483647. The default value is empty (not set).
Length Reference	Button and Enumerated type	If you have selected the length to be defined by *Length Reference*, select the name of the integer object that specifies the length of this object. Make your selection from the displayed list of integer objects that are defined as siblings of the current object, and occur before it in the structure of the message. For information about reordering elements, see "Reordering objects" on page 2900.
Inclusive Length Reference	Check box	This property is applicable only if *Length Reference* is set. If the check box is selected, the value of the sibling integer object that is identified by *Length Reference* is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by *Length Reference* is the length of the current object only. If the check box is selected, the *Length Units* property of the sibling integer object must be the same as that of the current object.

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. The length is given in bytes. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value for the *`Length Count`* or *`Length Reference`* property. The default is `Bytes`.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to the first instance only.

Property	Type	Meaning
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to all instances.

Occurrences

Property	Type	Meaning
Repeat Reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the displayed list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, there are no integer objects before this one in the message structure. If a *`Repeat Reference`* is specified, it overrides any setting for the *`Max Occurs`* logical property when parsing and writing the message, but not for validation of the message.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

CWF properties for element reference and local element Boolean types:

The CWF wire format properties for element reference and local element Boolean types.

The Custom Wire Format properties described here apply to:

- Objects: Element Reference, Local Element
- Boolean schema types: Boolean

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to all instances.

Occurrences

Property	Type	Meaning
Repeat Reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the displayed list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* logical property when parsing and writing the message, but not for validation of the message.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

CWF properties for element reference and local element dateTime types:

The CWF wire format properties for element reference and local element dateTime types.

The Custom Wire Format properties described here apply to:

- Objects: Element Reference, Local Element
- DateTime schema types: date, dateTime, gDay, gMonth, gMonthDay, gYear, gYearMonth, time

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select one from the displayed list: • `Fixed Length String`. The element's length is determined by other length properties as follows. • `Length Encoded String 1`. The first byte of the element contains the length of the string following the length byte in length units. The maximum length of a `Length Encoded String 1` element is 255 length units. • `Length Encoded String 2`. The element's first 2 bytes contain the length of the string following the two length bytes in length units. The maximum length of a `Length Encoded String 2` element is 65535 length units. The two length bytes are in the format of the WebSphere MQ queue manager Encoding. • `Null Terminated String`. The string ends with the hexadecimal NULL character, `X'00'`. • `Packed Decimal`. The dateTime is coded as a Packed Decimal number. It is valid only if the *`DateTime Format`* property represents numeric-only data. • `Binary`. The dateTime is encoded as a binary sequence of bytes. If you select this option, the range of symbols that you can specify for the Format String property is less than the range of symbols you can specify if you select a string option (see "DateTime formats" on page 6310 for details). • `Time Seconds`. This value supports C time_t and Java Date and Time objects. It is valid only if the *`DateTime Format`* property represents numeric-only data. • `Time Milliseconds`. This value supports C time_t and Java Date and Time objects. It is valid only if the *`DateTime Format`* property represents numeric-only data. The default value is `fixed length string`.
Length	Button and Integer	If you have selected a *`Physical Type`* of `Fixed Length String`, `Packed Decimal`, or `Binary`, and have selected the length to be defined by *`Length`*, enter the number of length units for the element. The minimum value that you can specify is `1` for all three physical types. The maximum value that you can specify is `256` for `Fixed Length String`, `10` for `Packed Decimal`, and `2147483647` for `Binary`. The default value is 0 (zero).
Length Reference	Button and Enumerated type	If you have selected the length to be defined by *`Length Reference`*, select the name of the integer object that specifies the length of this object. Make your selection from the displayed list of integer objects that are defined as siblings of the current object, and occur before it in the structure of the message. For information about reordering elements, see "Reordering objects" on page 2900.

Property	Type	Meaning
Inclusive Length Reference	Check box	This property is applicable only if *Length Reference* is set. If the check box is selected, the value of the sibling integer object that is identified by *Length Reference* is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by *Length Reference* is the length of the current object only. If the check box is selected, the *Length Units* property of the sibling integer object must be the same as that of the current object.
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. The length is given in bytes. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value for the *Length Count* or *Length Reference* property. The default is `Bytes`.
Justification	Enumerated type	If you have set the *Physical Type* property to `Fixed Length String`, select `Left Justify` (the default value) or `Right Justify` from the list. If you have selected another value for *Physical Type*, this property is inactive.

Property	Type	Meaning
Padding Character	String	If you have set the *Physical Type* property to `Fixed Length String`, and the *Justification* property is either `Left Justify` or `Right Justify`, this property is applicable. When writing an output message, use the padding character to fill out the remaining character positions when the string length is less than the length implied by the *Length* or *Length Reference* property. Whether the string is padded from the left or the right is governed by the *Justification* property. When parsing an input message, the padding character is trimmed from the end of the string. Whether the string is trimmed from the left or the right is governed by the *Justification* property. Specify this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the displayed list. • Enter a character between quotation marks; for example, `"c"` or `'c'`, where c is any alphanumeric character. • Enter a Unicode value in the form `U+xxxx` where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form `0xYY`, where `YY` is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, it is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.
DateTime Format	String	Specify a template for date and time. The default dateTime format is dependent on the logical type of the object. For information about the defaults for the dateTime format according to the logical type, see “DateTime defaults by logical type” on page 6320. See “DateTime formats” on page 6310 for details of date and time formats.

Numeric representation

Property	Type	Meaning
Signed	Check box	Specify whether the value is signed. This property is applicable only if the *Physical type* property is Packed Decimal. By default, this check box is cleared, which indicates that the value is not signed.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the displayed list. The option that you select determines the value that you must set for the property *Encoding Null Value*: • NULLPadFill. This option is valid only if *Physical Type* is Fixed Length String. The field is filled with the value specified by the *Padding Character*. The default value. • NULLLogicalValue. The *Encoding Null Value* property is first converted to an actual value, and rendered in the way specified for the field. • NULLLiteralValue. This specifies that *Encoding Null Value* contains a value that is directly substituted as if it is a string. Use this option when the value you have set for *Encoding Null Value* to specify a null date is not a dateTime value, or does not conform to the standard dateTime format yyyy-MM-dd 'T'HH:mm:ss. • NULLLiteralFill. The field is filled with the value specified by the *Encoding Null Value*. *Encoding Null Value* must resolve to a single character.
Encoding Null Value	String	If you set the *Encoding Null* property to NULLPadFill, this property is disabled. If you set the *Encoding Null* property to NULLLogicalValue, you must set this property to an ISO8601 dateTime format. These formats are described in "DateTime as string data" on page 6311. For example, specify a value conforming to yyyy-MM-dd'T'HH:mm:ss such as 1970-12-01. If you set the *Encoding Null* property to NULLLiteralValue, you can enter any value that is the same length as the field. If you set the *Encoding Null* property to NULLLiteralFill, the value must resolve to a single character. Set the character in one of the following ways: • Select SPACE, NUL, 0x00 or 0xFF from the displayed list • Enter a character between quotation marks, for example 'c' or "c", where c is any alphanumeric character. • Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value. • Enter a decimal character code in the form YY where YY is a decimal value. • Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal format.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to all instances.

Occurrences

Property	Type	Meaning
Repeat Reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the displayed list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, there are no integer objects before this one in the message structure. If a *`Repeat Reference`* is specified, it overrides any setting for the *`Max Occurs`* logical property when parsing and writing the message, but not for validation of the message.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

CWF properties for element reference and local element decimal types:

The CWF wire format properties for element reference and local element decimal types.

The Custom Wire Format properties described here apply to:

- Objects: Element Reference, Local Element
- Decimal schema types: decimal, integer, negativeInteger, nonNegativeInteger, nonPositiveInteger, positiveInteger, unsignedLong

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select one from the displayed list: • `Integer`. This equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • `Packed Decimal`. Equates to the COMP-3 data type in COBOL. • `External Decimal`. Equates to the data type PIC 9 USAGE DISPLAY in COBOL. The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager: • Elements that have *`Physical Type`* set to `Integer` and `Packed Decimal` are represented in the appropriate WebSphere MQ Encoding value. • Elements that have *`Physical Type`* set to `External Decimal` are represented in the WebSphere MQ CodedCharSetId value.

Property	Type	Meaning
Length	Integer	Enter the number of bytes to specify the element length: • If you set the *Physical Type* to Integer, select 1, 2, or 4 (the default) from the displayed list. • If you set the *Physical Type* to Packed Decimal, enter a value between 1 and 10. • If you set the *Physical Type* to Extended Decimal, enter a value between 1 and 256. (Numbers greater than the maximum COBOL PICTURE clause of 18 are assumed to be 18.)
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • Bytes. • Characters. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • Character Units. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to Characters. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the Length property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • End of Bitstream. All data up to the end of the bitstream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value for the *Length Count*. The default is Bytes.
Justification	Enumerated type	If you have set the *Physical Type* property to External Decimal, select Left Justify or Right Justify (the default value) from the list. If you have selected another value for *Physical Type*, this property is inactive.

Property	Type	Meaning
Padding Character	String	The padding character is used to fill out the remaining character positions when the string length is less than the specified string size. If you have set the *Physical Type* property to `Extended Decimal`, and the *Justification* property is either `Left Justify` or `Right Justify`, specify this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the displayed list. • Enter a character between quotation marks; for example, `"c"` or `'c'`, where c is any alphanumeric character. • Enter a Unicode value in the form `U+xxxx` where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form `0xYY`, where `YY` is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Numeric representation

Property	Type	Meaning
Signed	Check box	Select (the default) or clear this property. This property is used with *Sign Orientation*.

Property	Type	Meaning
Sign EBCDIC Custom Overpunched	Check box	If the *Physical Type* is set to External Decimal and the *Signed EBCDIC Custom* property is set, this indicates that the Sign EBCDIC Custom Overpunched representation is to be used within an ASCII environment. If this check box is not selected (the default), the Sign ASCII representation is used. The setting of the *Sign EBCDIC Custom Overpunched* check box is appropriate only if the *Sign Orientation* property is set to Leading or Trailing (indicating that the element/attribute has an embedded sign representation). The check box is not available if the element/attribute is unsigned (for example, if the *Signed* check box is not set).
Sign Orientation	Enumerated type	If you have set *Physical Type* to External Decimal and you have selected *Signed*, choose from the following options that represent the COBOL options for displaying numeric data: • Leading Overpunched. This option sets a bit in the first byte if the number is negative. No setting is made if the number is positive. For example, the ASCII hexadecimal representation of the number 22 is x'3232'. Using this option, the number +22 would be x'3232' and the number -22 would be x'7232'. The default value. • Leading Separate. This option sets the first byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte. • Trailing Overpunched. This option sets a bit in the last byte if the number is negative. No setting is made if the number is positive. Using this option, the number +22 would be x'3232' and the number -22 would be x'3272'. • Trailing Separate. This option sets the last byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte. If you have set *Physical Type* to any other value, the value Not Applicable is set for you.
Virtual Decimal Point	Integer	Specify the number of places to the left (for a positive value) or right (for a negative value) that a decimal point is to be moved from its assumed position. For example, a decimal element containing 1234 with a Virtual Decimal value of 3 is 1.234, equivalent to 'V' or 'P' in a COBOL picture clause. There is no C equivalent

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the displayed list: • NULLPadFill. This option is valid only if *Physical Type* is External Decimal. The field is filled with the value specified by the *Padding Character*. *Encoding Null Value* must be set to an empty string. • NULLLogicalValue. The *Encoding Null Value* is transformed to match the required format for the field. The default value. • NULLLiteralValue. The *Encoding Null Value* is directly substituted as if it is a string. You can specify a nonnumeric value for *Encoding Null Value*. • NULLLiteralFill. The field is filled with the value specified by the *Encoding Null Value*. *Encoding Null Value* must resolve to a single character.

Property	Type	Meaning
Encoding Null Value	String	The use of this property depends on the *Encoding Null* property, except for NULLLiteralFill. The default value is zero. If you set the *Encoding Null* property to NULLLiteralFill, the value must resolve to a single character. Set the character in one of the following ways: • Select SPACE, NUL, 0x00 or 0xFF from the displayed list • Enter a character between quotation marks, for example 'c' or "c", where c is any alphanumeric character. • Enter a hexadecimal character in the form 0xYY where YY is a hexadecimal value. • Enter a decimal character code in the form YY where YY is a decimal value. • Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal format.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • 1 Bytes. The default value. • 2 Bytes • 4 Bytes • 8 Bytes • 16 Bytes
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to all instances.

Occurrences

Property	Type	Meaning
Repeat Reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the displayed list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* logical property when parsing and writing the message, but not for validation of the message.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

CWF properties for element reference and local element float types:

The CWF wire format properties for element reference and local element float types.

The Custom Wire Format properties described here apply to:

- Objects: Element Reference, Local Element
- Float schema types: double, float

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select one from the displayed list: • `Integer`. This equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • `Float`. This equates to the data type FLOAT or DOUBLE in C or the COMP-1 or COMP-2 data type in COBOL and is the default value. • `Packed Decimal`. Equates to the COMP-3 data type in COBOL. • `External Decimal`. Equates to the data type PIC 9 USAGE DISPLAY in COBOL. The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager: • Elements that have *`Physical Type`* set to `Integer`, `Packed Decimal`, and `Float` are represented in the appropriate WebSphere MQ Encoding value. • Elements that have *`Physical Type`* set to `External Decimal` are represented in the WebSphere MQ CodedCharSetId value.
Length	Integer	Enter the number of bytes to specify the element length: • If you set the *`Physical Type`* to `Float`, select a value from the displayed list. The default value is 8. • If you set the *`Physical Type`* to `Integer`, select `1`, `2`, or `4` (the default) from the displayed list. • If you set the *`Physical Type`* to `Packed Decimal`, enter a value between 1 and 10. • If you set the *`Physical Type`* to `Extended Decimal`, enter a value between 1 and 256. (Numbers greater than the maximum COBOL PICTURE clause of 18 are assumed to be 18.)

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value for the *`Length Count`*. The default is `Bytes`.
Justification	Enumerated type	If you have set the *`Physical Type`* property to `External Decimal`, select `Left Justify` or `Right Justify` (the default value) from the list. If you have selected another value for *`Physical Type`*, this property is inactive.

Property	Type	Meaning
Padding Character	String	The padding character is used to fill out the remaining character positions when the string length is less than the specified string size. If you have set the *Physical Type* property to Extended Decimal, and the *Justification* property is either Left Justify or Right Justify, specify this character in one of the following ways: • Select NUL, '0', or SPACE from the displayed list. • Enter a character between quotation marks; for example, "c" or 'c', where c is any alphanumeric character. • Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form 0xYY, where YY is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Numeric representation

Property	Type	Meaning
Signed	Check box	Select or clear (unsigned, the default) this property. If you have set *Physical Type* to Float, this is selected. This property is used with *Sign Orientation*.

Property	Type	Meaning
Sign EBCDIC Custom Overpunched	Check box	If the *Physical Type* is set to External Decimal and the *Signed EBCDIC Custom* property is set, this indicates that the Sign EBCDIC Custom Overpunched representation is to be used within an ASCII environment. If this check box is not selected (the default), the Sign ASCII representation is used. The setting of the *Sign EBCDIC Custom Overpunched* check box is appropriate only if the *Sign Orientation* property is set to Leading or Trailing (indicating that the element/attribute has an embedded sign representation). The check box is not available if the element/attribute is unsigned (for example, if the *Signed* check box is not set).
Sign Orientation	Enumerated type	If you have set *Physical Type* to External Decimal and you have selected *Signed*, choose from the following options that represent the COBOL options for displaying numeric data: • Leading Overpunched. This option sets a bit in the first byte if the number is negative. No setting is made if the number is positive. For example, the ASCII hexadecimal representation of the number 22 is x'3232'. Using this option, the number +22 would be x'3232' and the number -22 would be x'7232'. The default value. • Leading Separate. This option sets the first byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte. • Trailing Overpunched. This option sets a bit in the last byte if the number is negative. No setting is made if the number is positive. Using this option, the number +22 would be x'3232' and the number -22 would be x'3272'. • Trailing Separate. This option sets the last byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte. If you have set *Physical Type* to any other value, the value Not Applicable is set for you.
Virtual Decimal Point	Integer	Specify the number of places to the left (for a positive value) or right (for a negative value) that a decimal point is to be moved from its assumed position. For example, a float element containing 1234 with a Virtual Decimal value of 3 is 1.234. This property is not applicable if you have set *Physical Type* to Float.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the displayed list: • NULLPadFill. This option is valid only if *Physical Type* is External Decimal. The field is filled with the value specified by the *Padding Character*. *Encoding Null Value* must be set to an empty string. • NULLLogicalValue. The *Encoding Null Value* is transformed to match the required format for the field. The default value. • NULLLiteralValue. The *Encoding Null Value* is directly substituted as if it is a string. You can specify a nonnumeric value for *Encoding Null Value*. • NULLLiteralFill. The field is filled with the value specified by the *Encoding Null Value*. *Encoding Null Value* must resolve to a single character.

Property	Type	Meaning
Encoding Null Value	String	The use of this property depends on the *Encoding Null* property, except for NULLLiteralFill. The default value is zero. If you set the *Encoding Null* property to NULLLiteralFill, the value must resolve to a single character. Set the character in one of the following ways: • Select SPACE, NUL, 0x00 or 0xFF from the displayed list • Enter a character between quotation marks, for example 'c' or "c", where c is any alphanumeric character. • Enter a hexadecimal character in the form 0xYY where YY is a hexadecimal value. • Enter a decimal character code in the form YY where YY is a decimal value. • Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal format.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • 1 Bytes. The default value. • 2 Bytes • 4 Bytes • 8 Bytes • 16 Bytes
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to all instances.

Occurrences

Property	Type	Meaning
Repeat Reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the displayed list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* logical property when parsing and writing the message, but not for validation of the message.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

CWF properties for element reference and local element integer types:

The CWF wire format properties for element reference and local element integer types.

The Custom Wire Format properties described here apply to:

- Objects: Element Reference, Local Element
- Integer schema types: byte, int, long, short, unsignedByte, unsignedInt, unsignedShort

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select one from the displayed list: • `Integer`. This equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • `Packed Decimal`. Equates to the COMP-3 data type in COBOL. • `External Decimal`. Equates to the data type PIC 9 USAGE DISPLAY in COBOL. The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager: • Elements that have *`Physical Type`* set to `Integer` and `Packed Decimal` are represented in the appropriate WebSphere MQ Encoding value. • Elements that have *`Physical Type`* set to `External Decimal` are represented in the WebSphere MQ CodedCharSetId value.
Length	Integer	Enter the number of bytes to specify the element length: • If you set the *`Physical Type`* to `Integer`, select `1`, `2`, or `4` (the default) from the displayed list. • If you have set *`Physical Type`* to `Packed Decimal`, enter a value between 1 and 6. • If you have set *`Physical Type`* to `Extended Decimal`, enter a value between 1 and 11.
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value for the *`Length Count`*. The default is `Bytes`.

Property	Type	Meaning
Justification	Enumerated type	If you have set the *Physical Type* property to `External Decimal`, select `Left Justify` or `Right Justify` (the default value) from the list. If you have selected another value for *Physical Type*, this property is inactive.
Padding Character	String	The padding character is used to fill out the remaining character positions when the string length is less than the specified string size. If you have set the *Physical Type* property to `Extended Decimal`, and the *Justification* property is either `Left Justify` or `Right Justify`, specify this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the displayed list. • Enter a character between quotation marks; for example, `"c"` or `'c'`, where `c` is any alphanumeric character. • Enter a Unicode value in the form `U+xxxx` where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form `0xYY`, where `YY` is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Numeric representation

Property	Type	Meaning
Signed	Check box	Select (the default) or clear this property. This property is used with *Sign Orientation*.

Property	Type	Meaning
Sign EBCDIC Custom Overpunched	Check box	If the *Physical Type* is set to External Decimal and the *Signed EBCDIC Custom* property is set, this indicates that the Sign EBCDIC Custom Overpunched representation is to be used within an ASCII environment. If this check box is not selected (the default), the Sign ASCII representation is used. The setting of the *Sign EBCDIC Custom Overpunched* check box is appropriate only if the *Sign Orientation* property is set to Leading or Trailing (indicating that the element/attribute has an embedded sign representation). The check box is not available if the element/attribute is unsigned (for example, if the *Signed* check box is not set).
Sign Orientation	Enumerated type	If you have set *Physical Type* to External Decimal and you have selected *Signed*, choose from the following options that represent the COBOL options for displaying numeric data: • Leading Overpunched. This option sets a bit in the first byte if the number is negative. No setting is made if the number is positive. For example, the ASCII hexadecimal representation of the number 22 is x'3232'. Using this option, the number +22 would be x'3232' and the number -22 would be x'7232'. The default value. • Leading Separate. This option sets the first byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte. • Trailing Overpunched. This option sets a bit in the last byte if the number is negative. No setting is made if the number is positive. Using this option, the number +22 would be x'3232' and the number -22 would be x'3272'. • Trailing Separate. This option sets the last byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte. If you have set *Physical Type* to any other value, the value Not Applicable is set for you.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the displayed list: • NULLPadFill. This option is valid only if *Physical Type* is External Decimal. The field is filled with the value specified by the *Padding Character*. *Encoding Null Value* must be set to an empty string. • NULLLogicalValue. The *Encoding Null Value* is transformed to match the required format for the field. The default value. • NULLLiteralValue. The *Encoding Null Value* is directly substituted as if it is a string. You can specify a nonnumeric value for *Encoding Null Value*. • NULLLiteralFill. The field is filled with the value specified by the *Encoding Null Value*. *Encoding Null Value* must resolve to a single character.

Property	Type	Meaning
Encoding Null Value	String	The use of this property depends on the *Encoding Null* property, except for NULLLiteralFill. The default value is zero. If you set the *Encoding Null* property to NULLLiteralFill, the value must resolve to a single character. Set the character in one of the following ways: • Select SPACE, NUL, 0x00 or 0xFF from the displayed list • Enter a character between quotation marks, for example 'c' or "c", where c is any alphanumeric character. • Enter a hexadecimal character in the form 0xYY where YY is a hexadecimal value. • Enter a decimal character code in the form YY where YY is a decimal value. • Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal format.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • 1 Bytes. The default value. • 2 Bytes • 4 Bytes • 8 Bytes • 16 Bytes
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to all instances.

Occurrences

Property	Type	Meaning
Repeat Reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the displayed list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* logical property when parsing and writing the message, but not for validation of the message.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

CWF properties for element reference and local element string types:

The CWF wire format properties for element reference and local element string types.

The Custom Wire Format properties described here apply to:

- Objects: Attribute Reference, Local Attribute
- String schema types: anyURI, ENTITIES, ENTITY, ID, IDREF, IDREFS, language, Name, NCName, NMTOKEN, NMTOKENS, normalizedString, NOTATION, QName, string, token

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select one from the displayed list: • `Fixed Length String`. The element's length is determined by other length properties as follows. • `Length Encoded String 1`. The first byte of the element contains the length of the string following the length byte in length units. The maximum length of a `Length Encoded String 1` element is 255 length units. • `Length Encoded String 2`. The element's first 2 bytes contain the length of the string following the two length bytes in length units. The maximum length of a `Length Encoded String 2` element is 65535 length units. The two length bytes are in the format of the WebSphere MQ queue manager Encoding. • `Null Terminated String`. The string ends with the hexadecimal NULL character, `X'00'`. The default is `Fixed Length String`.
Length	Button and Integer	If you have selected a *`Physical Type`* of `Fixed Length String` or `Binary`, and have selected the length to be defined by *`Length`*, enter the number of length units for the element. The minimum value that you can specify is `0` (zero), the maximum value that you can specify is `2147483647` The default value is `0` (zero).
Length Reference	Button and Enumerated type	If you have selected the length to be defined by *`Length Reference`*, select the name of the integer object that specifies the length of this object. Make your selection from the displayed list of integer objects that are defined as siblings of the current object, and occur before it in the structure of the message. For information about reordering elements, see “Reordering objects” on page 2900.
Inclusive Length Reference	Check box	This property is applicable only if *`Length Reference`* is set. If the check box is selected, the value of the sibling integer object that is identified by *`Length Reference`* is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by *`Length Reference`* is the length of the current object only. If the check box is selected, the *`Length Units`* property of the sibling integer object must be the same as that of the current object.

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. The length is given in bytes. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value for the *`Length Count`* or *`Length Reference`* property. The default is `Bytes`.
Justification	Enumerated type	If you have set the *`Physical Type`* property to `Fixed Length String`, select `Left Justify` (the default value) or `Right Justify` from the list. If you have selected another value for *`Physical Type`*, this property is inactive.

Property	Type	Meaning
Padding Character	String	If you have set the *Physical Type* property to `Fixed Length String`, and the *Justification* property is either `Left Justify` or `Right Justify`, this property is applicable. When writing an output message, use the padding character to fill out the remaining character positions when the string length is less than the length implied by the *Length* or *Length Reference* property. Whether the string is padded from the left or the right is governed by the *Justification* property. When parsing an input message, the padding character is trimmed from the end of the string. Whether the string is trimmed from the left or the right is governed by the *Justification* property. Specify this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the displayed list. • Enter a character between quotation marks; for example, `"c"` or `'c'`, where c is any alphanumeric character. • Enter a Unicode value in the form `U+xxxx` where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form `0xYY`, where `YY` is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, it is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the displayed list: • `NULLPadFill`. This is valid only if *`Physical Type`* is `Fixed Length String`. The field is filled with the value specified by the *`Padding Character`*. *`Encoding Null Value`* must be set to an empty string. • `NULLLogicalValue`. The *`Encoding Null Value`* is transformed to match the required format for the field. The default value. • `NULLLiteralValue`. The *`Encoding Null Value`* is directly substituted as if it is a string. • `NULLLiteralFill`. The field is filled with the value specified by the *`Encoding Null Value`*. *`Encoding Null Value`* must resolve to a single character.
Encoding Null Value	STRING	The use of this property depends on the *`Encoding Null`* property. If specified, its length must be equal to the length of the string element, except for `NULLLiteralFill`. The default value is empty (not set). If you set the *`Encoding Null`* property to `NULLLiteralFill`, the value must resolve to a single character. Set the character in one of the following ways: • Select `SPACE`, `NUL`, `0x00` or `0xFF` from the displayed list • Enter a character between quotation marks, for example `'c'` or `"c"`, where `c` is any alphanumeric character. • Enter a hexadecimal character code in the form `0xYY` where `YY` is a hexadecimal value. • Enter a decimal character code in the form `YY` where `YY` is a decimal value. • Enter a Unicode value in the form `U+xxxx` where `xxxx` is a Unicode value specified in hexadecimal format.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to all instances.

Occurrences

Property	Type	Meaning
Repeat Reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the displayed list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* logical property when parsing and writing the message, but not for validation of the message.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

Element reference XML properties:

The properties, and their permissible values, vary according to the type of object.

The properties that are displayed on the object page, and the values that those properties can take, can vary according to the type of the object. For example, the properties for type *string* are different from those of type *Boolean*. Select the link for the object type from the following table.

Type of object	**Properties**
Binary types	• base64Binary • hexBinary
Boolean types	• Boolean
DateTime types	• date • dateTime • gDay • gMonth • gMonthDay • gYear • gYearMonth • time
Decimal types	• decimal • integer • negativeInteger • nonNegativeInteger • nonPositiveInteger • positiveInteger • unsignedLong
Float types	• double • float
Integer types	• byte • int • long • short • unsignedByte • unsignedInt • unsignedShort
Interval types	• duration

Type of object	Properties
String types	• anyURI • ENTITIES • ENTITY • ID • IDREF • IDREFS • language • Name • NCName • NMTOKEN • NMTOKENS • normalizedString • NOTATION • QName • string • token

Note: The physical format properties for simple type *duration* are the same as the physical properties of the string logical types.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

XML properties for attribute reference, element reference, local attribute, local element binary types:

XML wire format properties for attribute reference, element reference, local attribute, and local element binary types.

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- Binary schema types: base64Binary, hexBinary

Field identification

A number of the following properties only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated type	Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list: • `XMLElement`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their *`XML Name`*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object , you must set their *`XML Name`* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in *`ID Attribute Name`* and a value as specified in *`ID Attribute Value`*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *`XML Name`*, *`ID Attribute Name`*, *`ID Attribute Value`* to the same values: – You must also set *`Value Attribute Name`* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in *`Value Attribute Name`*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *`ID Attribute Name`*. The value is the value of *`ID Attribute Value`*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *`XML Name`*, *`ID Attribute Name`*, *`ID Attribute Value`* to the same values: – You must also set *`Value Attribute Name`* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 6262 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
XML Name	String	Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message). This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name. You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name. If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).
ID Attribute Name	String	Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
ID Attribute Value	String	Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

Physical representation

Property	Type	Meaning
Encoding	String	Select one of the following values from the drop-down list: : • `CDatahex` (the default). Hexadecimal values in this field are specified with the CDATA qualifier, for example <e1><![CDATA[62]]></e1> • `hex`. Hexadecimal values in this field are specified as digits only, for example <e1>62</e1>. • `base64`. Values in this field are specified as digits only, coded in base 64.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

XML properties for attribute reference, element reference, local attribute, local element Boolean types:

XML wire format properties for attribute reference, element reference, local attribute and local element Boolean types.

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- Boolean schema types: Boolean

Field identification

A number of the following properties only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated type	Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list: • `XMLElement`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their *XML Name*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object , you must set their *XML Name* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in *ID Attribute Name* and a value as specified in *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in *Value Attribute Name*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *ID Attribute Name*. The value is the value of *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 6262 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
XML Name	String	Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message). This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name. You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name. If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).
ID Attribute Name	String	Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
ID Attribute Value	String	Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

XML properties for attribute reference, element reference, local attribute, local element dateTime types:

XML wire format properties for attribute reference, element reference, local attribute, and local element dateTime types.

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- DateTime schema types: date, dateTime, gDay, gMonth, gMonthDay, gYear, gYearMonth, time

Field identification

A number of the following properties only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated type	Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list: • `XMLElement`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their *`XML Name`*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object , you must set their *`XML Name`* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in *`ID Attribute Name`* and a value as specified in *`ID Attribute Value`*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *`XML Name`*, *`ID Attribute Name`*, *`ID Attribute Value`* to the same values: – You must also set *`Value Attribute Name`* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in *`Value Attribute Name`*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *`ID Attribute Name`*. The value is the value of *`ID Attribute Value`*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *`XML Name`*, *`ID Attribute Name`*, *`ID Attribute Value`* to the same values: – You must also set *`Value Attribute Name`* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 6262 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
XML Name	String	Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message). This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name. You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name. If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).
ID Attribute Name	String	Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
ID Attribute Value	String	Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

Physical representation

Property	Type	Meaning
DateTime Format	String	Specify a format string that specifies the rendering of the value for dateTime elements. The default dateTime format is dependent on the logical type of the object. For information about the defaults for the dateTime format according to the logical type see "DateTime defaults by logical type" on page 6320. See "DateTime formats" on page 6310 for details of dateTime formats.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical

model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

XML properties for attribute reference, element reference, local attribute, local element decimal types:

XML wire format properties for attribute reference, element reference, local attribute, and local element decimal types.

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- Decimal schema types: decimal, integer, negativeInteger, nonNegativeInteger, nonPositiveInteger, positiveInteger, unsignedLong

Field identification

A number of the following properties only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated type	Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list: • `XMLElement`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their *XML Name*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object , you must set their *XML Name* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in *ID Attribute Name* and a value as specified in *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in *Value Attribute Name*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *ID Attribute Name*. The value is the value of *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 6262 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
XML Name	String	Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message). This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name. You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name. If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).
ID Attribute Name	String	Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
ID Attribute Value	String	Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

XML properties for attribute reference, element reference, local attribute, local element float types:

XML wire format properties for attribute reference, element reference, local attribute, and local element float types.

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- Float schema types: double, float

Field identification

A number of the following properties only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated type	Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list: • `XMLElement`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their *`XML Name`*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object , you must set their *`XML Name`* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in *`ID Attribute Name`* and a value as specified in *`ID Attribute Value`*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *`XML Name`*, *`ID Attribute Name`*, *`ID Attribute Value`* to the same values: – You must also set *`Value Attribute Name`* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in *`Value Attribute Name`*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *`ID Attribute Name`*. The value is the value of *`ID Attribute Value`*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *`XML Name`*, *`ID Attribute Name`*, *`ID Attribute Value`* to the same values: – You must also set *`Value Attribute Name`* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 6262 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
XML Name	String	Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message). This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name. You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name. If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).
ID Attribute Name	String	Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
ID Attribute Value	String	Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

XML properties for attribute reference, element reference, local attribute, local element integer types:

XML wire format properties for attribute reference, element reference, local attribute, and local element integer types.

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- Integer schema types: byte, int, long, short, unsignedByte, unsignedInt, unsignedShort

Field identification

A number of the following properties only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated type	Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list: • `XMLElement`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their *XML Name*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object , you must set their *XML Name* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in *ID Attribute Name* and a value as specified in *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in *Value Attribute Name*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *ID Attribute Name*. The value is the value of *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 6262 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
XML Name	String	Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message). This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name. You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name. If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).
ID Attribute Name	String	Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
ID Attribute Value	String	Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

XML properties for attribute reference, element reference, local attribute, local element string types:

XML wire format properties for attribute reference, element reference, local attribute, and local element string types.

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- String schema types: anyURI, ENTITIES, ENTITY, ID, IDREF, IDREFS, language, Name, NCName, NMTOKEN, NMTOKENS, normalizedString, NOTATION, QName, string, token

Field identification

A number of the following properties only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated type	Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list: • `XMLElement`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their *`XML Name`*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object , you must set their *`XML Name`* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in *`ID Attribute Name`* and a value as specified in *`ID Attribute Value`*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *`XML Name`*, *`ID Attribute Name`*, *`ID Attribute Value`* to the same values: – You must also set *`Value Attribute Name`* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in *`Value Attribute Name`*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *`ID Attribute Name`*. The value is the value of *`ID Attribute Value`*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *`XML Name`*, *`ID Attribute Name`*, *`ID Attribute Value`* to the same values: – You must also set *`Value Attribute Name`* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 6262 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
XML Name	String	Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message). This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name. You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name. If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).
ID Attribute Name	String	Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
ID Attribute Value	String	Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

Element reference TDS properties:

The properties, and their permissible values, vary according to the object type.

The properties that are displayed on the object page, and the values that those properties can take, can vary according to the type of the object. For example, the properties for type *string* are different from those of type *Boolean*. Select the link for the object type from the following table.

Type of object	**Properties**
Binary types	• base64Binary • hexBinary
Boolean types	• Boolean
Complex types	
DateTime types	• date • dateTime • gDay • gMonth • gMonthDay • gYear • gYearMonth • time
Decimal types	• decimal • integer • negativeInteger • nonNegativeInteger • nonPositiveInteger • positiveInteger • unsignedLong

Type of object	Properties
Float types	• double • float
Integer types	• byte • int • long • short • unsignedByte • unsignedInt • unsignedShort
Interval types	• duration
String types	• anyURI • ENTITIES • ENTITY • ID • IDREF • IDREFS • language • Name • NCName • NMTOKEN • NMTOKENS • normalizedString • NOTATION • QName • string • token

Note: The physical format properties for simple type *duration* are the same as the physical properties of the string logical types.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM TDS format" on page 1221
The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

TDS properties for element reference binary types:

TDS wire format properties for element reference binary types.

The TDS Format properties described here apply to:

- Objects: Element Reference
- Binary schema types: base64Binary, hexBinary

Physical representation

Property	Type	Meaning
Length Reference	Enumerated type	This property is applicable only if `Physical Type` is `Text`, `Binary`, or `TLOG Specific`. If set, this property takes precedence over any value in the `Length Units` property. Specify the identifier of a sibling integer object, the value of which determines the length of the object in question. The sibling object must be defined before the current object within the message structure. For information about reordering elements, see "Reordering objects" on page 2900.
Inclusive Length Reference	Check box	This property is applicable only if `Length Reference` is set. If the check box is selected, the value of the sibling integer object that is identified by `Length Reference` is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by `Length Reference` is the length of the current object only. If the check box is selected, the `Length Units` property of the sibling integer object must be the same as that of the current object.

Occurrences

Property	Type	Meaning
Repeat reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, no integer objects exist before this one in the message structure. If a `Repeat Reference` is specified, it overrides any setting for the `Max Occurs` logical property when parsing and writing the message, but not for validating the message.
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the `Data Element Separation` property of the parent group or complex type is set to `All Elements Delimited` or `Variable Length Elements Delimited`. A default value is provided if the previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; if the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used. If none of the previous conditions are true, a default is not applied.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

TDS properties for element reference Boolean types:

TDS wire format properties for element reference Boolean types.

The TDS Format properties described here apply to:

- Objects: Element Reference
- Boolean schema types: Boolean

Occurrences

Property	Type	Meaning
Repeat reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, no integer objects exist before this one in the message structure. If a `Repeat Reference` is specified, it overrides any setting for the `Max Occurs` logical property when parsing and writing the message, but not for validating the message.
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the `Data Element Separation` property of the parent group or complex type is set to `All Elements Delimited` or `Variable Length Elements Delimited`. A default value is provided if the previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; if the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used. If none of the previous conditions are true, a default is not applied.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

TDS properties for element references to complex elements:

The TDS Format properties that apply to element references where the global element is of complex type.

Occurrences

Property	Type	Meaning
Repeat reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, no integer objects exist before this one in the message structure. If a `Repeat Reference` is specified, it overrides any setting for the `Max Occurs` logical property when parsing and writing the message, but not for validating the message.

Property	Type	Meaning
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the `Data Element Separation` property of the parent group or complex type is set to `All Elements Delimited` or `Variable Length Elements Delimited`. A default value is provided if the previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; if the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used. If none of the previous conditions are true, a default is not applied.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

TDS properties for element reference dateTime types:

TDS wire format properties for attribute reference and element reference dateTime types.

The TDS Format properties described here apply to:

- Objects: Element Reference
- DateTime schema types: date, dateTime, gDay, gMonth, gMonthDay, gYear, gYearMonth, time

Physical representation

Property	Type	Meaning
Length Reference	Enumerated type	This property is applicable only if `Physical Type` is `Text`, `Binary`, or `TLOG Specific`. If set, this property takes precedence over any value in the `Length Units` property. Specify the identifier of a sibling integer object, the value of which determines the length of the object in question. The sibling object must be defined before the current object within the message structure. For information about reordering elements, see "Reordering objects" on page 2900.
Inclusive Length Reference	Check box	This property is applicable only if `Length Reference` is set. If the check box is selected, the value of the sibling integer object that is identified by `Length Reference` is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by `Length Reference` is the length of the current object only. If the check box is selected, the `Length Units` property of the sibling integer object must be the same as that of the current object.

Occurrences

Property	Type	Meaning
Repeat reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, no integer objects exist before this one in the message structure. If a `Repeat Reference` is specified, it overrides any setting for the `Max Occurs` logical property when parsing and writing the message, but not for validating the message.

Property	Type	Meaning
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the `Data Element Separation` property of the parent group or complex type is set to `All Elements Delimited` or `Variable Length Elements Delimited`. A default value is provided if the previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; if the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used. If none of the previous conditions are true, a default is not applied.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

TDS properties for element reference decimal types:

The TDS wire format properties for attribute reference and element reference decimal types.

The TDS Format properties described here apply to:

- Objects: Element Reference
- Decimal schema types: decimal, integer, negativeInteger, nonNegativeInteger, nonPositiveInteger, positiveInteger, unsignedLong

Physical representation

Property	Type	Meaning
Length Reference	Enumerated type	This property is applicable only if `Physical Type` is `Text`, `Binary`, or `TLOG Specific`. If set, this property takes precedence over any value in the `Length Units` property. Specify the identifier of a sibling integer object, the value of which determines the length of the object in question. The sibling object must be defined before the current object within the message structure. For information about reordering elements, see "Reordering objects" on page 2900.
Inclusive Length Reference	Check box	This property is applicable only if `Length Reference` is set. If the check box is selected, the value of the sibling integer object that is identified by `Length Reference` is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by `Length Reference` is the length of the current object only. If the check box is selected, the `Length Units` property of the sibling integer object must be the same as that of the current object.

Occurrences

Property	Type	Meaning
Repeat reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, no integer objects exist before this one in the message structure. If a `Repeat Reference` is specified, it overrides any setting for the `Max Occurs` logical property when parsing and writing the message, but not for validating the message.

Property	Type	Meaning
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the `Data Element Separation` property of the parent group or complex type is set to `All Elements Delimited` or `Variable Length Elements Delimited`. A default value is provided if the previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; if the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used. If none of the previous conditions are true, a default is not applied.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

TDS properties for element reference float types:

TDS wire format properties for attribute reference and element reference float types.

The TDS Format properties described here apply to:

- Objects: Element Reference
- Float schema types: double, float

Physical representation

Property	Type	Meaning
Length Reference	Enumerated type	This property is applicable only if `Physical Type` is `Text`, `Binary`, or `TLOG Specific`. If set, this property takes precedence over any value in the `Length Units` property. Specify the identifier of a sibling integer object, the value of which determines the length of the object in question. The sibling object must be defined before the current object within the message structure. For information about reordering elements, see "Reordering objects" on page 2900.
Inclusive Length Reference	Check box	This property is applicable only if `Length Reference` is set. If the check box is selected, the value of the sibling integer object that is identified by `Length Reference` is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by `Length Reference` is the length of the current object only. If the check box is selected, the `Length Units` property of the sibling integer object must be the same as that of the current object.

Occurrences

Property	Type	Meaning
Repeat reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, no integer objects exist before this one in the message structure. If a `Repeat Reference` is specified, it overrides any setting for the `Max Occurs` logical property when parsing and writing the message, but not for validating the message.

Property	Type	Meaning
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the `Data Element Separation` property of the parent group or complex type is set to `All Elements Delimited` or `Variable Length Elements Delimited`. A default value is provided if the previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; if the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used. If none of the previous conditions are true, a default is not applied.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

TDS properties for element reference integer types:

TDS wire format properties for attribute reference and element reference integer types.

The TDS Format properties described here apply to:

- Objects: Element Reference
- Integer schema types: byte, int, long, short, unsignedByte, unsignedInt, unsignedShort

Physical representation

Property	Type	Meaning
Length Reference	Enumerated type	This property is applicable only if `Physical Type` is `Text`, `Binary`, or `TLOG Specific`. If set, this property takes precedence over any value in the `Length Units` property. Specify the identifier of a sibling integer object, the value of which determines the length of the object in question. The sibling object must be defined before the current object within the message structure. For information about reordering elements, see "Reordering objects" on page 2900.
Inclusive Length Reference	Check box	This property is applicable only if `Length Reference` is set. If the check box is selected, the value of the sibling integer object that is identified by `Length Reference` is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by `Length Reference` is the length of the current object only. If the check box is selected, the `Length Units` property of the sibling integer object must be the same as that of the current object.

Occurrences

Property	Type	Meaning
Repeat reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, no integer objects exist before this one in the message structure. If a `Repeat Reference` is specified, it overrides any setting for the `Max Occurs` logical property when parsing and writing the message, but not for validating the message.

Property	Type	Meaning
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the `Data Element Separation` property of the parent group or complex type is set to `All Elements Delimited` or `Variable Length Elements Delimited`. A default value is provided if the previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; if the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used. If none of the previous conditions are true, a default is not applied.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

TDS properties for element reference string types:

The TDS wire format properties for attribute reference and element reference string types.

The TDS Format properties described here apply to:

- Objects: Element Reference
- String schema types: anyURI, ENTITIES, ENTITY, ID, IDREF, IDREFS, language, Name, NCName, NMTOKEN, NMTOKENS, normalizedString, NOTATION, QName, string, token

Physical representation

Property	Type	Meaning
Length Reference	Enumerated type	This property is applicable only if `Physical Type` is `Text`, `Binary`, or `TLOG Specific`. If set, this property takes precedence over any value in the `Length Units` property. Specify the identifier of a sibling integer object, the value of which determines the length of the object in question. The sibling object must be defined before the current object within the message structure. For information about reordering elements, see "Reordering objects" on page 2900.
Inclusive Length Reference	Check box	This property is applicable only if `Length Reference` is set. If the check box is selected, the value of the sibling integer object that is identified by `Length Reference` is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by `Length Reference` is the length of the current object only. If the check box is selected, the `Length Units` property of the sibling integer object must be the same as that of the current object.

Occurrences

Property	Type	Meaning
Repeat reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, no integer objects exist before this one in the message structure. If a `Repeat Reference` is specified, it overrides any setting for the `Max Occurs` logical property when parsing and writing the message, but not for validating the message.

Property	Type	Meaning
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the `Data Element Separation` property of the parent group or complex type is set to `All Elements Delimited` or `Variable Length Elements Delimited`. A default value is provided if the previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; if the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used. If none of the previous conditions are true, a default is not applied.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

Documentation properties for all message set objects:

Use the documentation property of an object to add information that describes the function of the object.

The documentation property is available on all objects except Key, Keyref, and Unique objects.

The property is a string field, and you can use any standard alphanumeric characters.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

Global attribute properties:

Different types of properties are available for a global attribute.

A global attribute can have the following properties;

- "Global attribute logical properties" on page 5425
- "Global attribute CWF properties" on page 5462
- "Global attribute XML properties" on page 5483
- "Global attribute TDS properties" on page 5511
- "Documentation properties for all message set objects" on page 5413

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

Global attribute logical properties:

The logical properties of a global attribute.

Property	Type	Meaning
Name	String	Specify a name for the object when you create it. Names can consist of alphanumeric characters, including the letters *A* through *Z*, *a* through *z*, and the digits *0* through *9*. They might also include the following punctuation characters; • - the hyphen • _ the underscore • . the period Names can start only with a letter or the underscore character, and not with a number, hyphen, or period. Names that begin with `xml`, or any variant of these characters (for example `XmL`), are reserved by the XML standards specification. Further details of naming conventions and allowable characters can be found in the Extensible Markup Language (XML) specification that can be found on the World Wide Web Consortium (W3C) Web site.

Property	Type	Meaning
Type	Enumerated type	The Type property constrains the type of data that can be present in the object. Select from the following options: • `int` • `string` • `Boolean` • `hexBinary` • `dateTime` • `date` • `time` • `decimal` • `float` • `(More...)` • `(New Simple Type)` • `(New Complex Type)` If you select (More...), the Type Selection wizard starts. In this wizard, you can select any of the available types. If you select (New Simple Type), the New Simple Type wizard starts. In this wizard, you can create an Anonymous simple type that is based on an existing type. This type can be created locally or globally. If you select (New Complex Type), the New Complex Type wizard starts. In this wizard, you can create an Anonymous complex type, which can be derived from an existing base type. This type can be created locally or globally. For further information about these types, and examples of their use, see the XML Schema Part 0: Primer. This document is available on the World Wide Web Consortium (W3C) Web site.
Namespace	Enumerated type	Namespaces are a simple method for qualifying element and attribute names by associating them with namespaces identified by URI references. If <no target namespace> is displayed, a namespace has not been set for this object. If the property is inactive, the message set has not been configured to support namespaces. Where the property is active, namespaces that are available for selection are displayed in the drop-down list.

Value

The *Value* properties are used with the *Usage* property in an Attribute Reference or a Local Attribute.

Property	Type	Meaning
Default	Button and String	This property provides the default value for an element or attribute. **XMLNSC domain** When parsing with validation enabled, default values are applied to missing attributes and empty elements as required by the XML Schema 1.0 specification. When writing, elements or attributes that are missing from the message tree are not automatically added to the output XML bit stream, even if they have default values. If missing elements or attributes are required, the message tree can be serialized and then re-parsed with validation enabled. **MRM (CWF and TDS physical formats)** When writing a fixed-length portion of a message (CWF or fixed-length TDS), if an attribute or element is missing from the message tree, the default value is inserted into the bit stream so that the message structure is preserved. **MRM (XML physical format)** No support for default values **Other domains** No support for default values.
Fixed	Button and String	This property provides the fixed value for an element or attribute. **XMLNSC domain** When parsing with validation enabled, if an attribute or element is present, the value is validated against the fixed value. If the values are not equal, a validation error is signalled. Also, when parsing with validation enabled, fixed values are applied to missing attributes and empty elements as required by the XML Schema 1.0 specification. When writing, elements or attributes that are missing from the message tree are not automatically added to the output XML bit stream, even if a fixed value has been specified. If missing elements or attributes are required, the message tree can be serialized and then re-parsed with validation enabled. **MRM (CWF and TDS physical formats)** When writing a fixed-length portion of a message (CWF or fixed-length TDS), if an attribute or element is missing from the message tree, the fixed value is inserted into the bit stream so that the message structure is preserved. **MRM (XML physical format)** No support for fixed values **Other domains** No support for fixed values.

Property	Type	Meaning
Interpret Value As	Enumerated type	Specify if values stored within this object must be interpreted as having significance for the parser and, if so, the type of interpretation that must occur. Select one of the following options: • None This value is the default value, and indicates that the element or attribute does not have a key value associated with it. • MessageSetIdentity. Specifies that the value of the element or attribute corresponds to the identifier, name, or alias (in that priority order) that is associated with the message set where all subsequent embedded messages that are descendents of the enclosing message are defined. This value remains in force unless a new element or attribute MessageSetIdentity field is encountered which resets the MessageSetIdentity value. • MessageIdentity. Specifies that the value of the element or attribute corresponds to the name or alias (in that priority order) that is associated with a message, and acts as an identifier for subsequent embedded messages which are the immediate children of the enclosing message. This identity applies until a new element or attribute MessageIdentity field is encountered at the same level in the tree. The embedded message can be defined in either the current message set, or in a message set identified by using a MessageSetIdentity. **Note:** This property is applicable only when the type of the object is derived from xsd:string.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Global attribute CWF properties:

There are no properties to show.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM Custom Wire Format" on page 1214
Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not separated by delimiters.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Global attribute XML properties:

The properties, and their permissible values, vary according to the object type.

The properties that are displayed on the object page, and the values that those properties can take, can vary according to the type of the object. For example, the properties for type *string* are different from those of type *Boolean*. Select the link for the object type from the following table.

Type of object	Properties
Binary types	• base64Binary • hexBinary
Boolean types	• Boolean
DateTime types	• date • dateTime • gDay • gMonth • gMonthDay • gYear • gYearMonth • time

Type of object	Properties
Decimal types	• decimal • integer • negativeInteger • nonNegativeInteger • nonPositiveInteger • positiveInteger • unsignedLong
Float types	• double • float
Integer types	• byte • int • long • short • unsignedByte • unsignedInt • unsignedShort
Interval types	• duration
String types	• anyURI • ENTITIES • ENTITY • ID • IDREF • IDREFS • language • Name • NCName • NMTOKEN • NMTOKENS • normalizedString • NOTATION • QName • string • token

Note: The physical format properties for simple type *duration* are the same as the physical properties of the string logical types.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

XML properties for global attribute and global element binary types:

The XML wire format properties for global attribute and global element binary types.

The XML Wire Format properties described here apply to:

- Objects: Global Attribute, Global Element
- Binary schema types: base64Binary, hexBinary

Property	Type	Meaning
XML Name	String	Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message). This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name. You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name. If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).

Physical representation

Property	Type	Meaning
Encoding	String	Select one of the following values from the drop-down list: : • `CDatahex` (the default). Hexadecimal values in this field are specified with the CDATA qualifier, for example <e1><![CDATA[62]]></e1> • `hex`. Hexadecimal values in this field are specified as digits only, for example <e1>62</e1>. • `base64`. Values in this field are specified as digits only, coded in base 64.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

XML properties for global attribute and global element Boolean types:

The XML wire format properties for global attribute and global element Boolean types.

The XML Wire Format properties described here apply to:

- Objects: Global Attribute, Global Element
- Boolean schema types: Boolean

Property	Type	Meaning
XML Name	String	Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message). This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name. You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name. If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the

function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

XML properties for global attribute and global element dateTime types:

The XML wire format properties for global attribute and global element dateTime types.

The XML Wire Format properties described here apply to:

- Objects: Global Attribute, Global Element
- DateTime schema types: date, dateTime, gDay, gMonth, gMonthDay, gYear, gYearMonth, time

Property	Type	Meaning
XML Name	String	Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message). This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name. You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name. If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).

Physical representation

Property	Type	Meaning
DateTime Format	String	Specify a format string that specifies the rendering of the value for dateTime elements. The default dateTime format is dependent on the logical type of the object. For information about the defaults for the dateTime format according to the logical type see "DateTime defaults by logical type" on page 6320. See "DateTime formats" on page 6310 for details of dateTime formats.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

XML properties for global attribute and global element decimal types:

The XML wire format properties for global attribute and global element decimal types.

The XML Wire Format properties described here apply to:

- Objects: Global Attribute, Global Element
- Decimal schema types: decimal, integer, negativeInteger, nonNegativeInteger, nonPositiveInteger, positiveInteger, unsignedLong

Property	Type	Meaning
XML Name	String	Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message). This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name. You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name. If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

XML properties for global attribute and global element float types:

The XML wire format properties for global attribute and global element float types.

The XML Wire Format properties described here apply to:

- Objects: Global Attribute, Global Element
- Float schema types: double, float

Property	Type	Meaning
XML Name	String	Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message). This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name. You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name. If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

XML properties for global attribute and global element integer types:

The XML wire format properties for global attribute and global element integer types.

The XML Wire Format properties described here apply to:

- Objects: Global Attribute, Global Element
- Integer schema types: byte, int, long, short, unsignedByte, unsignedInt, unsignedShort

Property	Type	Meaning
XML Name	String	Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message). This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name. You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name. If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

XML properties for global attribute and global element string types:

The XML wire format properties for global attribute and global element string types.

The XML Wire Format properties described here apply to:

- Objects: Global Attribute, Global Element
- String schema types: anyURI, ENTITIES, ENTITY, ID, IDREF, IDREFS, language, Name, NCName, NMTOKEN, NMTOKENS, normalizedString, NOTATION, QName, string, token

Property	Type	Meaning
XML Name	String	Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message). This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name. You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name. If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

Global attribute TDS properties:

The properties, and their permissible values, vary according to the object type.

The properties that are displayed on the object page, and the values that those properties can take, can vary according to the type of the object. For example, the properties for type *string* are different from those of type *Boolean*. Select the link for the object type from the following table.

Type of object	Properties
Binary types	• base64Binary • hexBinary
Boolean types	• Boolean
DateTime types	• date • dateTime • gDay • gMonth • gMonthDay • gYear • gYearMonth • time

Type of object	Properties
Decimal types	• decimal • integer • negativeInteger • nonNegativeInteger • nonPositiveInteger • positiveInteger • unsignedLong
Float types	• double • float
Integer types	• byte • int • long • short • unsignedByte • unsignedInt • unsignedShort
Interval types	• duration
String types	• anyURI • ENTITIES • ENTITY • ID • IDREF • IDREFS • language • Name • NCName • NMTOKEN • NMTOKENS • normalizedString • NOTATION • QName • string • token

Note: The physical format properties for simple type *duration* are the same as the physical properties of the string logical types.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM TDS format" on page 1221
The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

TDS properties for global attribute binary types:

The TDS format properties for global attribute binary types.

The TDS Format properties described here apply to:

- Objects: Global Attribute
- Binary schema types: base64Binary, hexBinary

Field identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the `Data Element Separation` property of the complex type or types in which the object is a child is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, this property must contain a non-empty value. If the object is a complex element, and the `Data Element Separation` property of its parent is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, the property can contain an empty value. The value for this property must be unique for every element in the message set; that is, no two elements in the message set can contain the same value for this property.

Property	Type	Meaning
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the `Data Element Separation` method is set to `Use Data Pattern` in the complex type. For more details, see "Regular expression syntax" on page 6304.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	This value of this property defaults to Binary. It cannot be changed.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no `Length Reference` is specified. The default is dependent on the setting of the message set property `Derive default length from logical type`. If `Derive default length from logical type` is selected, the default value is derived from any length or maxLength value constraint (schema facet) on the object's simple type.
Length Units	Enumerated type	Select the unit of length for the object. Select one of the following options (some physical types do not offer both options): • `Bytes`. The length is given in bytes. • `Characters`. The length is given in characters. The number of bytes that are processed in the bit stream depends on the code page of the message. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the contents of the bit stream. The parser reads one character at a time and determines whether the character comprises one or more bytes. The default is dependent on the physical type of the object.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

TDS properties for global attribute Boolean types:

The TDS format properties for global attribute Boolean types.

The TDS Format properties described here apply to:

- Objects: Global Attribute
- Boolean schema types: Boolean

Field identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the `Data Element Separation` property of the complex type or types in which the object is a child is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, this property must contain a non-empty value. If the object is a complex element, and the `Data Element Separation` property of its parent is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, the property can contain an empty value. The value for this property must be unique for every element in the message set; that is, no two elements in the message set can contain the same value for this property.
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the `Data Element Separation` method is set to `Use Data Pattern` in the complex type. For more details, see "Regular expression syntax" on page 6304.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the `Messaging Standard` property of the message set is `User Defined Text`, `User Defined Mixed`, `CSV`, or `TLOG`, select one of the following values: • `Text`. The data is in character format. • `Binary`. The data is in bit string format. The default is dependent on the `Messaging Standard` property. For all other `Messaging Standard` values, the `Physical Type` property is set to `Text`.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

TDS properties for global attribute dateTime types:

The TDS format properties for global attribute dateTime types.

The TDS Format properties described here apply to:

- Objects: Global Attribute
- DateTime schema types: date, dateTime, gDay, gMonth, gMonthDay, gYear, gYearMonth, time

Field Identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the `Data Element Separation` property of the complex type or types in which the object is a child is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, this property must contain a non-empty value. If the object is a complex element, and the `Data Element Separation` property of its parent is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, the property can contain an empty value. The value for this property must be unique for every element in the message set; that is, no two elements in the message set can contain the same value for this property.
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the `Data Element Separation` method is set to `Use Data Pattern` in the complex type. For more details, see “Regular expression syntax” on page 6304.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the `Messaging Standard` property of the message set is `User Defined Text`, `User Defined Mixed`, `CSV`, or `TLOG`, select one of the following values: • `Text`. The data is in character format. • `Length Encoded String 1`. The first byte of the data contains the length (in length units) of the data string that follows the length byte. The maximum length of a `Length Encoded String 1` element is 255 length units. • `Length Encoded String 2`. The first two bytes of the data contain the length (in length units) of the data string that follows the two length bytes. The maximum length of a `Length Encoded String 2` element is 65535 length units. • `Null Terminated String`. The data string ends with the hexadecimal NULL character, `X'00'`. • `Packed Decimal`. The data is a packed decimal number that equates to the COMP-3 data type in COBOL. • `Binary`. The data is in bit string format. • `Time Seconds`. This value supports C time_t, and Java Date and Time objects. It is valid only if the `DateTime Format` property represents numeric-only data. • `Time Milliseconds`. This value supports C time_t, and Java Date and Time objects. It is valid only if the `DateTime Format` property represents numeric-only data. The default is dependent on the `Messaging Standard` property. For all other `Messaging Standard` values, the `Physical Type` property is set to `Text`.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no `Length Reference` is specified. If this property is not set and the message set property `Derive default length from logical type` is selected, and the `Physical type` is 'Character', the default value is derived from any length or maxLength value constraint (schema facet) on the simple type.
Length Units	Enumerated type	Select the unit of length for the object. Select one of the following options (some physical types do not offer both options): • `Bytes`. The length is given in bytes. • `Characters`. The length is given in characters. The number of bytes that are processed in the bit stream depends on the code page of the message. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the contents of the bit stream. The parser reads one character at a time and determines whether the character comprises one or more bytes. The default is dependent on the physical type of the object.

Property	Type	Meaning
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • `Not Applicable` • `Left Justify` • `Right Justify`
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the drop-down list. • Enter a character between quotation marks, for example "c" or 'c', where c is any alphanumeric character. • Enter a hexadecimal character code in the form `0xYY`, where `YY` is a hexadecimal value. • Enter a Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.
DateTime Format	String	Specify a template for date and time. The default DateTime format is dependent on the logical type of the object. For information about the defaults for the dateTime format according to the logical type, see “DateTime defaults by logical type” on page 6320. See “DateTime formats” on page 6310 for details of date and time formats.

Numeric representation

Property	Type	Meaning
Signed	Check box	Specify whether the value is signed. This property is applicable only if the `Physical type` property is `Packed Decimal`. By default, this check box is cleared, which indicates that the value is not signed.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

TDS properties for global attribute decimal types:

The TDS format properties for global attribute decimal types.

The TDS Format properties described here apply to:

- Objects: Global Attribute
- Decimal schema types: decimal, integer, negativeInteger, nonNegativeInteger, nonPositiveInteger, positiveInteger, unsignedLong

Field Identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the `Data Element Separation` property of the complex type or types in which the object is a child is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, this property must contain a non-empty value. If the object is a complex element, and the `Data Element Separation` property of its parent is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, the property can contain an empty value. The value for this property must be unique for every element in the message set; that is, no two elements in the message set can contain the same value for this property.
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the `Data Element Separation` method is set to `Use Data Pattern` in the complex type. For more details, see "Regular expression syntax" on page 6304.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the `Messaging Standard` property of the message set is `User Defined Text`, `User Defined Mixed`, `CSV`, or `TLOG`, select one of the following values: • `Text`. The data is in character format. • `Packed Decimal`. The data is a packed decimal number that equates to the COMP-3 data type in COBOL. • `TLOG Specific` - this option can be selected only if the `Message Standard` property of the message set is `TLOG`. This option indicates that the format of the data is specific to the TLOG messaging standard. • `Integer`. Equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • `External Decimal`. The data is a decimal number that equates to the data type PIC 9 USAGE DISPLAY in COBOL. The default is dependent on the `Messaging Standard` property. For all other `Messaging Standard` values, the `Physical Type` property is set to `Text`.

Property	Type	Meaning
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no `Length Reference` is specified. If this property is not set and the message set property `Derive default length from logical type` is selected, and the `Physical type` is 'Character', the default value is derived from any length or maxLength value constraint (schema facet) on the simple type.
Length Units	Enumerated type	Select the unit of length for the object. Select one of the following options (some physical types do not offer both options): • `Bytes`. The length is given in bytes. • `Characters`. The length is given in characters. The number of bytes that are processed in the bit stream depends on the code page of the message. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the contents of the bit stream. The parser reads one character at a time and determines whether the character comprises one or more bytes. The default is dependent on the physical type of the object.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • `Not Applicable` • `Left Justify` • `Right Justify`

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the drop-down list. • Enter a character between quotation marks, for example "c" or 'c', where c is any alphanumeric character. • Enter a hexadecimal character code in the form `0xYY`, where `YY` is a hexadecimal value. • Enter a Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.

Numeric representation

Property	Type	Meaning
Sign Orientation	Enumerated type	The values that you can choose for this property are: • Leading Separate • Trailing Separate • Leading Overpunched • Trailing Overpunched This property is enabled only if you have set Physical Type to Text or External Decimal, and you have selected Signed. If Physical Type is Text, the only valid values of Sign Orientation are Leading Separate and Trailing Separate. If Physical Type is External Decimal and Sign EBCDIC Customer Overpunched is selected, the only valid values of Sign Orientation are Leading Overpunched and Trailing Overpunched.
Positive Sign	String	Specify the value that represents the positive symbol. Do not specify a numeric value. If no value is set, "+" is assumed. The positive sign is not written when creating an output message; it is used only to recognize the positive sign when parsing a message bit stream. This property is applicable only if Physical Type is Text and Signed is selected.
Negative Sign	String	Specify the value that represents the negative symbol. Do not specify a numeric value. If no value is set, "-" is assumed. This property is applicable only if Physical Type is Text and Signed is selected.
Virtual Decimal Point	Button and Integer	Specify a non-zero integer that represents the position of an implied decimal point within a number, or specify 0, the default, to use the formatting of Float or Decimal numbers as specified by the Precision property. If you specify a positive integer, the position of the decimal point is moved left from the right side of the number. For example, if you specify 3, the decimal value 1234 represents 1.234 If you specify a negative integer, the position of the decimal point is moved right from the right side of the number. For example, if you specify -3, the decimal value 1234 represents 1,234,000.

Property	Type	Meaning
Precision	Button and Integer	This property is applicable only if `Physical Type` is `Text`. It is used if the value of the `Virtual Decimal Point` property is 0, which indicates that the decimal point is present in the data. It deals with truncation, and specifies how many digits are to follow the decimal point. Either specify a number of digits: • If you set `Precision` to 0, data is truncated so that the fractional part is lost. For example, the value `123.45` is truncated to `123`. • If you set `Precision` to a number less than the number of fractional digits, data is truncated. For example, the value `123.4567` is truncated to `123.45` if you set `Precision` to 2. • If you set `Precision` to a number greater than the number of fractional digits, the value is padded with extra zeros. For example, the value `12.345` is padded to `12.34500` if you set `Precision` to 5. Or select one option from the list: • `All Significant Digits - decimal separator only required if fractional digits` (the default): all significant digits are written to the output bit stream, and no decimal separator is written if no fractional digits are present. • `Explicit Decimal Separator - decimal separator always required`: all significant digits are written to the output bit stream and the decimal separator is always included, even when no fractional digits are present. The decimal separator must be present in the input bit stream, even when no fractional digits are present.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

TDS properties for global attribute float types:

The TDS format properties for global attribute float types.

The TDS Format properties described here apply to:

- Objects: Global Attribute
- Float schema types: double, float

Field Identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the `Data Element Separation` property of the complex type or types in which the object is a child is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, this property must contain a non-empty value. If the object is a complex element, and the `Data Element Separation` property of its parent is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, the property can contain an empty value. The value for this property must be unique for every element in the message set; that is, no two elements in the message set can contain the same value for this property.
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the `Data Element Separation` method is set to `Use Data Pattern` in the complex type. For more details, see "Regular expression syntax" on page 6304.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the `Messaging Standard` property of the message set is `User Defined Text`, `User Defined Mixed`, `CSV`, or `TLOG`, select one of the following values: • `Text`. The data is in character format. • `Float`. Equates to the data type FLOAT or DOUBLE in C, or the COMP-1 or COMP-2 numeric data type in COBOL. • `Integer`. Equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • `Packed Decimal`. The data is a packed decimal number that equates to the COMP-3 data type in COBOL. • `External Decimal`. The data is a decimal number that equates to the data type PIC 9 USAGE DISPLAY in COBOL. The default is dependent on the `Messaging Standard` property. For all other `Messaging Standard` values, the `Physical Type` property is set to `Text`.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no `Length Reference` is specified. If this property is not set and the message set property `Derive default length from logical type` is selected, and the `Physical type` is 'Character', the default value is derived from any length or maxLength value constraint (schema facet) on the simple type.
Length Units	Enumerated type	Select the unit of length for the object. Select one of the following options (some physical types do not offer both options): • `Bytes`. The length is given in bytes. • `Characters`. The length is given in characters. The number of bytes that are processed in the bit stream depends on the code page of the message. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the contents of the bit stream. The parser reads one character at a time and determines whether the character comprises one or more bytes. The default is dependent on the physical type of the object.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • `Not Applicable` • `Left Justify` • `Right Justify`

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the drop-down list. • Enter a character between quotation marks, for example "c" or 'c', where c is any alphanumeric character. • Enter a hexadecimal character code in the form `0xYY`, where `YY` is a hexadecimal value. • Enter a Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.

Numeric representation

Property	Type	Meaning
Sign Orientation	Enumerated type	The values that you can choose for this property are: • `Leading Separate` • `Trailing Separate` • `Leading Overpunched` • `Trailing Overpunched` This property is enabled only if you have set `Physical Type` to `Text` or `External Decimal`, and you have selected `Signed`. If `Physical Type` is `Text`, the only valid values of `Sign Orientation` are `Leading Separate` and `Trailing Separate`. If `Physical Type` is `External Decimal` and `Sign EBCDIC Customer Overpunched` is selected, the only valid values of `Sign Orientation` are `Leading Overpunched` and `Trailing Overpunched`.
Positive Sign	String	Specify the value that represents the positive symbol. Do not specify a numeric value. If no value is set, "+" is assumed. The positive sign is not written when creating an output message; it is used only to recognize the positive sign when parsing a message bit stream. This property is applicable only if `Physical Type` is `Text` and `Signed` is selected.
Negative Sign	String	Specify the value that represents the negative symbol. Do not specify a numeric value. If no value is set, "-" is assumed. This property is applicable only if `Physical Type` is `Text` and `Signed` is selected.
Virtual Decimal Point	Button and Integer	Specify a non-zero integer that represents the position of an implied decimal point within a number, or specify 0, the default, to use the formatting of Float or Decimal numbers as specified by the `Precision` property. If you specify a positive integer, the position of the decimal point is moved left from the right side of the number. For example, if you specify 3, the decimal value 1234 represents 1.234 If you specify a negative integer, the position of the decimal point is moved right from the right side of the number. For example, if you specify -3, the decimal value 1234 represents 1,234,000.

Property	Type	Meaning
Precision	Button and Integer	This property is applicable only if `Physical Type` is `Text`. It is used if the value of the `Virtual Decimal Point` property is 0, which indicates that the decimal point is present in the data. It deals with truncation, and specifies how many digits are to follow the decimal point. Either specify a number of digits: • If you set `Precision` to 0, data is truncated so that the fractional part is lost. For example, the value `123.45` is truncated to `123`. • If you set `Precision` to a number less than the number of fractional digits, data is truncated. For example, the value `123.4567` is truncated to `123.45` if you set `Precision` to 2. • If you set `Precision` to a number greater than the number of fractional digits, the value is padded with extra zeros. For example, the value `12.345` is padded to `12.34500` if you set `Precision` to 5. Or select one option from the list: • `All Significant Digits - decimal separator only required if fractional digits` (the default): all significant digits are written to the output bit stream, and no decimal separator is written if no fractional digits are present. • `Explicit Decimal Separator - decimal separator always required`: all significant digits are written to the output bit stream and the decimal separator is always included, even when no fractional digits are present. The decimal separator must be present in the input bit stream, even when no fractional digits are present. • `Exponential Notation - Example "1.23456e002"`: data is written out to the bit stream as a signed value having the format `[sign1]a.bbbe[sign2]ccc` where: – `[sign1]` is the value of `Negative Sign` if the value is negative – `a` is a single decimal digit – `bbb` is one or more decimal digits – `[sign2]` is the value of `Negative Sign` if the exponent is negative – `ccc` is exactly three decimal digits (the exponent) `[sign1]` and `[sign2]` are absent if the value and exponent are positive. For example, the value -123.456 is represented as `-1.23456e002` and the value 0.00012 is represented as `1.2e-004` in the output bit stream, assuming that the value of `Negative Sign` is "-", and the value of `Sign Orientation` is `Leading`. The value -0.00012 is represented as `1.2*e*004` if `Negative Sign` is "*" and `Sign Orientation` is `Trailing`.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

TDS properties for global attribute integer types:

The TDS format properties for global attribute integer types.

The TDS Format properties described here apply to:

- Objects: Global Attribute
- Integer schema types: byte, int, long, short, unsignedByte, unsignedInt, unsignedShort

Field Identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the `Data Element Separation` property of the complex type or types in which the object is a child is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, this property must contain a non-empty value. If the object is a complex element, and the `Data Element Separation` property of its parent is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, the property can contain an empty value. The value for this property must be unique for every element in the message set; that is, no two elements in the message set can contain the same value for this property.
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the `Data Element Separation` method is set to `Use Data Pattern` in the complex type. For more details, see "Regular expression syntax" on page 6304.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the `Messaging Standard` property of the message set is `User Defined Text`, `User Defined Mixed`, `CSV`, or `TLOG`, select one of the following values: • `Text`. The data is in character format. • `Integer`. Equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • `Packed Decimal`. The data is a packed decimal number that equates to the COMP-3 data type in COBOL. • `External Decimal`. The data is a decimal number that equates to the data type PIC 9 USAGE DISPLAY in COBOL. The default is dependent on the `Messaging Standard` property. For all other `Messaging Standard` values, the `Physical Type` property is set to `Text`.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no `Length Reference` is specified. If this property is not set and the message set property `Derive default length from logical type` is selected, and the `Physical type` is 'Character', the default value is derived from any length or maxLength value constraint (schema facet) on the simple type.
Length Units	Enumerated type	Select the unit of length for the object. Select one of the following options (some physical types do not offer both options): • `Bytes`. The length is given in bytes. • `Characters`. The length is given in characters. The number of bytes that are processed in the bit stream depends on the code page of the message. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the contents of the bit stream. The parser reads one character at a time and determines whether the character comprises one or more bytes. The default is dependent on the physical type of the object.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • `Not Applicable` • `Left Justify` • `Right Justify`

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the drop-down list. • Enter a character between quotation marks, for example "c" or 'c', where c is any alphanumeric character. • Enter a hexadecimal character code in the form `0xYY`, where `YY` is a hexadecimal value. • Enter a Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.

Numeric representation

Property	Type	Meaning
Sign Orientation	Enumerated type	The values that you can choose for this property are: • `Leading Separate` • `Trailing Separate` • `Leading Overpunched` • `Trailing Overpunched` This property is enabled only if you have set `Physical Type` to `Text` or `External Decimal`, and you have selected `Signed`. If `Physical Type` is `Text`, the only valid values of `Sign Orientation` are `Leading Separate` and `Trailing Separate`. If `Physical Type` is `External Decimal` and `Sign EBCDIC Customer Overpunched` is selected, the only valid values of `Sign Orientation` are `Leading Overpunched` and `Trailing Overpunched`.
Positive Sign	String	Specify the value that represents the positive symbol. Do not specify a numeric value. If no value is set, "+" is assumed. The positive sign is not written when creating an output message; it is used only to recognize the positive sign when parsing a message bit stream. This property is applicable only if `Physical Type` is `Text` and `Signed` is selected.
Negative Sign	String	Specify the value that represents the negative symbol. Do not specify a numeric value. If no value is set, "-" is assumed. This property is applicable only if `Physical Type` is `Text` and `Signed` is selected.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

TDS properties for global attribute string types:

The TDS format properties for global attribute string types.

The TDS Format properties described here apply to:

- Objects: Global Attribute
- String schema types: anyURI, ENTITIES, ENTITY, ID, IDREF, IDREFS, language, Name, NCName, NMTOKEN, NMTOKENS, normalizedString, NOTATION, QName, string, token

Field Identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the `Data Element Separation` property of the complex type or types in which the object is a child is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, this property must contain a non-empty value. If the object is a complex element, and the `Data Element Separation` property of its parent is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, the property can contain an empty value. The value for this property must be unique for every element in the message set; that is, no two elements in the message set can contain the same value for this property.
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the `Data Element Separation` method is set to `Use Data Pattern` in the complex type. For more details, see "Regular expression syntax" on page 6304.

Property	Type	Meaning
Interpret Element Value	Enumerated type	Specify whether values stored within this object are interpreted as having significance for the parser and, if so, the type of interpretation that occurs. This interpretation is standard-specific and is therefore hard coded. The possible values for this property are: • `None` (the default value) • `EDIFACT Service String` • `X12 Service String` • `Message Key` • `EDIFACT Syntax Level ID` • `HL7 Service String` • `HL7 Field Separator` **Note:** The `Message Key` enumeration has been deprecated.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the `Messaging Standard` property of the message set is `User Defined Text`, `User Defined Mixed`, `CSV`, or `TLOG`, select one of the following values: • `Text`. The data is in character format. • `Length Encoded String 1`. The first byte of the data contains the length (in length units) of the data string that follows the length byte. The maximum length of a `Length Encoded String 1` element is 255 length units. • `Length Encoded String 2`. The first two bytes of the data contain the length (in length units) of the data string that follows the two length bytes. The maximum length of a `Length Encoded String 2` element is 65535 length units. • `Null Terminated String`. The data string ends with the hexadecimal NULL character, `X'00'`. • `TLOG Specific` - this option can be selected only if the `Message Standard` property of the message set is `TLOG`. This option indicates that the format of the data is specific to the TLOG messaging standard. The default is dependent on the `Messaging Standard` property. For all other `Messaging Standard` values, the `Physical Type` property is set to `Text`.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no `Length Reference` is specified. If this property is not set and the message set property `Derive default length from logical type` is selected, and the `Physical type` is 'Character', the default value is derived from any length or maxLength value constraint (schema facet) on the simple type.

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the object. Select one of the following options (some physical types do not offer both options): • `Bytes`. The length is given in bytes. • `Characters`. The length is given in characters. The number of bytes that are processed in the bit stream depends on the code page of the message. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the contents of the bit stream. The parser reads one character at a time and determines whether the character comprises one or more bytes. The default is dependent on the physical type of the object.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • `Not Applicable` • `Left Justify` • `Right Justify`

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the drop-down list. • Enter a character between quotation marks, for example "c" or 'c', where c is any alphanumeric character. • Enter a hexadecimal character code in the form `0xYY`, where `YY` is a hexadecimal value. • Enter a Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

Documentation properties for all message set objects:

Use the documentation property of an object to add information that describes the function of the object.

The documentation property is available on all objects except Key, Keyref, and Unique objects.

The property is a string field, and you can use any standard alphanumeric characters.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

Global attribute group properties:

Different types of properties are available for a global attribute group.

A global attribute group can have the following properties;

- "Global attribute group logical properties" on page 5429

- "Global attribute group CWF properties" on page 5463
- "Global attribute group XML properties" on page 5485
- "Global attribute group TDS properties" on page 5513
- "Documentation properties for all message set objects" on page 5413

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

Global attribute group logical properties:

The logical properties of an attribute group.

Property	Type	Meaning
Name	String	Specify a name for the object when you create it. Names can consist of alphanumeric characters, including the letters *A* through *Z*, *a* through *z*, and the digits *0* through *9*. They might also include the following punctuation characters; • - the hyphen • _ the underscore • . the period Names can start only with a letter or the underscore character, and not with a number, hyphen, or period. Names that begin with `xml`, or any variant of these characters (for example `XmL`), are reserved by the XML standards specification. Further details of naming conventions and allowable characters can be found in the Extensible Markup Language (XML) specification that can be found on the World Wide Web Consortium (W3C) Web site.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Global attribute group CWF properties:

There are no properties to show.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM Custom Wire Format" on page 1214
Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not separated by delimiters.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Global attribute group XML properties:

There are no properties to show.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Global attribute group TDS properties:

The TDS properties of a global attribute group.

There are no properties to show.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM TDS format" on page 1221
The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Documentation properties for all message set objects:

Use the documentation property of an object to add information that describes the function of the object.

The documentation property is available on all objects except Key, Keyref, and Unique objects.

The property is a string field, and you can use any standard alphanumeric characters.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

Global element properties:

Different types of properties are available for a global element.

A global element can have the following properties;

- "Global element logical properties" on page 5430
- "Global element CWF properties" on page 5463
- "Global element XML properties" on page 5486
- "Global element TDS properties" on page 5513
- "Documentation properties for all message set objects" on page 5413

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

Global element logical properties:

The logical properties of a global element.

Property	Type	Meaning
Name	String	Specify a name for the object when you create it. Names can consist of alphanumeric characters, including the letters *A* through *Z*, *a* through *z*, and the digits *0* through *9*. They might also include the following punctuation characters; • - the hyphen • _ the underscore • . the period Names can start only with a letter or the underscore character, and not with a number, hyphen, or period. Names that begin with `xml`, or any variant of these characters (for example `XmL`), are reserved by the XML standards specification. Further details of naming conventions and allowable characters can be found in the Extensible Markup Language (XML) specification that can be found on the World Wide Web Consortium (W3C) Web site.

Property	Type	Meaning
Type	Enumerated type	The Type property constrains the type of data that can be present in the object. Select from the following options: • int • string • Boolean • hexBinary • dateTime • date • time • decimal • float • (More...) • (New Simple Type) • (New Complex Type) If you select (More...), the Type Selection wizard starts. In this wizard, you can select any of the available types. If you select (New Simple Type), the New Simple Type wizard starts. In this wizard, you can create an Anonymous simple type that is based on an existing type. This type can be created locally or globally. If you select (New Complex Type), the New Complex Type wizard starts. In this wizard, you can create an Anonymous complex type, which can be derived from an existing base type. This type can be created locally or globally. For further information about these types, and examples of their use, see the XML Schema Part 0: Primer. This document is available on the World Wide Web Consortium (W3C) Web site.
Namespace	Enumerated type	Namespaces are a simple method for qualifying element and attribute names by associating them with namespaces identified by URI references. If <no target namespace> is displayed, a namespace has not been set for this object. If the property is inactive, the message set has not been configured to support namespaces. Where the property is active, namespaces that are available for selection are displayed in the drop-down list.

Value

Property	Type	Meaning
Default	Button and String	This property provides the default value for an element or attribute. **XMLNSC domain** When parsing with validation enabled, default values are applied to missing attributes and empty elements as required by the XML Schema 1.0 specification. When writing, elements or attributes that are missing from the message tree are not automatically added to the output XML bit stream, even if they have default values. If missing elements or attributes are required, the message tree can be serialized and then re-parsed with validation enabled. **MRM (CWF and TDS physical formats)** When writing a fixed-length portion of a message (CWF or fixed-length TDS), if an attribute or element is missing from the message tree, the default value is inserted into the bit stream so that the message structure is preserved. **MRM (XML physical format)** No support for default values **Other domains** No support for default values.
Fixed	Button and String	This property provides the fixed value for an element or attribute. **XMLNSC domain** When parsing with validation enabled, if an attribute or element is present, the value is validated against the fixed value. If the values are not equal, a validation error is signalled. Also, when parsing with validation enabled, fixed values are applied to missing attributes and empty elements as required by the XML Schema 1.0 specification. When writing, elements or attributes that are missing from the message tree are not automatically added to the output XML bit stream, even if a fixed value has been specified. If missing elements or attributes are required, the message tree can be serialized and then re-parsed with validation enabled. **MRM (CWF and TDS physical formats)** When writing a fixed-length portion of a message (CWF or fixed-length TDS), if an attribute or element is missing from the message tree, the fixed value is inserted into the bit stream so that the message structure is preserved. **MRM (XML physical format)** No support for fixed values **Other domains** No support for fixed values.
Nillable	Check box	Select this option if you want the element to be able to be defined as null. A null value is distinct from being empty, when the element contains no data.

Property	Type	Meaning
Interpret Value As	Enumerated type	Specify if values stored within this object must be interpreted as having significance for the parser and, if so, the type of interpretation that must occur. Select one of the following options: • None This value is the default value, and indicates that the element or attribute does not have a key value associated with it. • MessageSetIdentity. Specifies that the value of the element or attribute corresponds to the identifier, name, or alias (in that priority order) that is associated with the message set where all subsequent embedded messages that are descendents of the enclosing message are defined. This value remains in force unless a new element or attribute MessageSetIdentity field is encountered which resets the MessageSetIdentity value. • MessageIdentity. Specifies that the value of the element or attribute corresponds to the name or alias (in that priority order) that is associated with a message, and acts as an identifier for subsequent embedded messages which are the immediate children of the enclosing message. This identity applies until a new element or attribute MessageIdentity field is encountered at the same level in the tree. The embedded message can be defined in either the current message set, or in a message set identified by using a MessageSetIdentity. **Note:** This property is applicable only when the type of the object is derived from xsd:string.

Substitution settings

Substitution Groups provide a means by which one element may be substituted for another in a message. The element which can be substituted is called the 'head' element, and the substitution group is the list of elements that may be used in its place. An element can be in at most one substitution group.

Property	Type	Meaning
Final	Enumerated type	Limit the set of elements that can belong to its substitution group. • Empty • restriction. Prohibit element substitution by elements whose types are restrictions of the type of the head element. • extension. Prohibit element substitution by elements whose types are extensions of the type of the head element. • #all. Prohibit substitution by all methods.
Block	Enumerated type	Limit the set of elements that can be substituted for this element in a message. • Empty • restriction. Prohibit element substitution by elements whose types are restrictions of the type of the head element • extension. Prohibit element substitution by elements whose types are extensions of the type of the head element • substitution. Prohibit element substitution by members of the element's substitution group. • #all. Prohibit substitution by all methods.
Substitution Group	Enumerated type	Specify the name of a 'head' element. Setting this property indicates that this element is a member of the substitution group for the head element.
Abstract	Check box	Select this option if you do not want the element to appear in the message, but require one of the members of its substitution group to appear in its place.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Global element CWF properties:

There are no CWF properties to show for a global element.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM Custom Wire Format" on page 1214
Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not separated by delimiters.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Global element XML properties:

The properties, and their permissible values, vary according to the type of object.

The properties that are displayed on the object page, and the values that those properties can take, can vary according to the type of the object. For example, the properties for type *string* are different from those of type *Boolean*. Select the link for the object type from the following table.

Type of object	**Properties**
Binary types	• base64Binary • hexBinary
Boolean types	• Boolean
DateTime types	• date • dateTime • gDay • gMonth • gMonthDay • gYear • gYearMonth • time
Decimal types	• decimal • integer • negativeInteger • nonNegativeInteger • nonPositiveInteger • positiveInteger • unsignedLong
Float types	• double • float
Integer types	• byte • int • long • short • unsignedByte • unsignedInt • unsignedShort
Interval types	• duration

Type of object	Properties
String types	• anyURI • ENTITIES • ENTITY • ID • IDREF • IDREFS • language • Name • NCName • NMTOKEN • NMTOKENS • normalizedString • NOTATION • QName • string • token

Note: The physical format properties for simple type *duration* are the same as the physical properties of the string logical types.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

XML properties for global attribute and global element binary types:

The XML wire format properties for global attribute and global element binary types.

The XML Wire Format properties described here apply to:

- Objects: Global Attribute, Global Element
- Binary schema types: base64Binary, hexBinary

Property	Type	Meaning
XML Name	String	Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message). This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name. You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name. If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).

Physical representation

Property	Type	Meaning
Encoding	String	Select one of the following values from the drop-down list: : • `CDatahex` (the default). Hexadecimal values in this field are specified with the CDATA qualifier, for example <e1><![CDATA[62]]></e1> • `hex`. Hexadecimal values in this field are specified as digits only, for example <e1>62</e1>. • `base64`. Values in this field are specified as digits only, coded in base 64.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that

are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

XML properties for global attribute and global element Boolean types:

The XML wire format properties for global attribute and global element Boolean types.

The XML Wire Format properties described here apply to:

- Objects: Global Attribute, Global Element
- Boolean schema types: Boolean

Property	Type	Meaning
XML Name	String	Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message). This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name. You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name. If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

XML properties for global attribute and global element dateTime types:

The XML wire format properties for global attribute and global element dateTime types.

The XML Wire Format properties described here apply to:

- Objects: Global Attribute, Global Element
- DateTime schema types: date, dateTime, gDay, gMonth, gMonthDay, gYear, gYearMonth, time

Property	Type	Meaning
XML Name	String	Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message). This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name. You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name. If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).

Physical representation

Property	Type	Meaning
DateTime Format	String	Specify a format string that specifies the rendering of the value for dateTime elements. The default dateTime format is dependent on the logical type of the object. For information about the defaults for the dateTime format according to the logical type see “DateTime defaults by logical type” on page 6320. See “DateTime formats” on page 6310 for details of dateTime formats.

Related concepts:

“Message modeling” on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

“The message model” on page 1160
The message model consists of the following components.

“MRM XML physical format: Relationship to the logical model” on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

“Constructing message models” on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

“Working with MRM message model objects” on page 2870
Add, configure, and delete objects.

Related reference:

“Message model reference information” on page 5366
Reference information in this section can help you develop and configure message models.

“Message model object properties” on page 5416
Access property information by property kind, or by object.

“Logical properties for message model objects” on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

XML properties for global attribute and global element decimal types:

The XML wire format properties for global attribute and global element decimal types.

The XML Wire Format properties described here apply to:

- Objects: Global Attribute, Global Element
- Decimal schema types: decimal, integer, negativeInteger, nonNegativeInteger, nonPositiveInteger, positiveInteger, unsignedLong

Property	Type	Meaning
XML Name	String	Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message). This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name. You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name. If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that

are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

XML properties for global attribute and global element float types:

The XML wire format properties for global attribute and global element float types.

The XML Wire Format properties described here apply to:

- Objects: Global Attribute, Global Element
- Float schema types: double, float

Property	Type	Meaning
XML Name	String	Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message). This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name. You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name. If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined

structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

XML properties for global attribute and global element integer types:

The XML wire format properties for global attribute and global element integer types.

The XML Wire Format properties described here apply to:

- Objects: Global Attribute, Global Element
- Integer schema types: byte, int, long, short, unsignedByte, unsignedInt, unsignedShort

Property	Type	Meaning
XML Name	String	Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message). This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name. You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name. If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

XML properties for global attribute and global element string types:

The XML wire format properties for global attribute and global element string types.

The XML Wire Format properties described here apply to:

- Objects: Global Attribute, Global Element
- String schema types: anyURI, ENTITIES, ENTITY, ID, IDREF, IDREFS, language, Name, NCName, NMTOKEN, NMTOKENS, normalizedString, NOTATION, QName, string, token

Property	Type	Meaning
XML Name	String	Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message). This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name. You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name. If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

Global element TDS properties:

The properties, and their permissible values, vary according to the object type.

The properties that are displayed on the object page, and the values that those properties can take, can vary according to the type of the object. For example, the properties for type *string* are different from those of type *Boolean*. Select the link for the object type from the following table.

Type of object	**Properties**
Binary types	• base64Binary • hexBinary
Boolean types	• Boolean
Complex types	
DateTime types	• date • dateTime • gDay • gMonth • gMonthDay • gYear • gYearMonth • time
Decimal types	• decimal • integer • negativeInteger • nonNegativeInteger • nonPositiveInteger • positiveInteger • unsignedLong
Float types	• double • float

Type of object	Properties
Integer types	• byte • int • long • short • unsignedByte • unsignedInt • unsignedShort
Interval types	• duration
String types	• anyURI • ENTITIES • ENTITY • ID • IDREF • IDREFS • language • Name • NCName • NMTOKEN • NMTOKENS • normalizedString • NOTATION • QName • string • token

Note: The physical format properties for simple type *duration* are the same as the physical properties of the string logical types.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM TDS format" on page 1221
The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

TDS properties for global element binary types:

The TDS format properties for global element binary types.

The TDS Format properties described here apply to:

- Objects: Global Element
- Binary schema types: base64Binary, hexBinary

Field identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the `Data Element Separation` property of the complex type or types in which the object is a child is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, this property must contain a non-empty value. If the object is a complex element, and the `Data Element Separation` property of its parent is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, the property can contain an empty value. The value for this property must be unique for every element in the message set; that is, no two elements in the message set can contain the same value for this property.
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the `Data Element Separation` method is set to `Use Data Pattern` in the complex type. For more details, see "Regular expression syntax" on page 6304.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	For all `Messaging Standard` values, the `Physical Type` property is set to `Binary` and cannot be changed.

Property	Type	Meaning
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no `Length Reference` is specified. The default is dependent on the setting of the message set property `Derive default length from logical type`. If `Derive default length from logical type` is selected, the default value is derived from any length or maxLength value constraint (schema facet) on the object's simple type.
Length Units	Enumerated type	Always set to `Bytes`.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

TDS properties for global element Boolean types:

The TDS format properties for global element Boolean types.

The TDS Format properties described here apply to:

- Objects: Global Element
- Boolean schema types: Boolean

Field identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the `Data Element Separation` property of the complex type or types in which the object is a child is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, this property must contain a non-empty value. If the object is a complex element, and the `Data Element Separation` property of its parent is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, the property can contain an empty value. The value for this property must be unique for every element in the message set; that is, no two elements in the message set can contain the same value for this property.
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the `Data Element Separation` method is set to `Use Data Pattern` in the complex type. For more details, see "Regular expression syntax" on page 6304.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the `Messaging Standard` property of the message set is `User Defined Text`, `User Defined Mixed`, `CSV`, or `TLOG`, select one of the following values: • `Text`. The data is in character format. • `Binary`. The data is in bit string format. The default is dependent on the `Messaging Standard` property. For all other `Messaging Standard` values, the `Physical Type` property is set to `Text`.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

TDS properties for global element of complex type:

The TDS Format properties that apply to global elements of complex type.

Field identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the `Data Element Separation` property of the complex type or types in which the object is a child is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, this property must contain a non-empty value. If the object is a complex element, and the `Data Element Separation` property of its parent is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, the property can contain an empty value. The value for this property must be unique for every element in the message set; that is, no two elements in the message set can contain the same value for this property.
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the `Data Element Separation` method is set to `Use Data Pattern` in the complex type. For more details, see "Regular expression syntax" on page 6304.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

TDS properties for global element dateTime types:

The TDS format properties for global element dateTime types.

The TDS Format properties described here apply to:

- Objects: Global Element
- DateTime schema types: date, dateTime, gDay, gMonth, gMonthDay, gYear, gYearMonth, time

Field Identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the `Data Element Separation` property of the complex type or types in which the object is a child is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, this property must contain a non-empty value. If the object is a complex element, and the `Data Element Separation` property of its parent is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, the property can contain an empty value. The value for this property must be unique for every element in the message set; that is, no two elements in the message set can contain the same value for this property.
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the `Data Element Separation` method is set to `Use Data Pattern` in the complex type. For more details, see "Regular expression syntax" on page 6304.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the `Messaging Standard` property of the message set is `User Defined Text`, `User Defined Mixed`, `CSV`, or `TLOG`, select one of the following values: • `Text`. The data is in character format. • `Length Encoded String 1`. The first byte of the data contains the length (in length units) of the data string that follows the length byte. The maximum length of a `Length Encoded String 1` element is 255 length units. • `Length Encoded String 2`. The first two bytes of the data contain the length (in length units) of the data string that follows the two length bytes. The maximum length of a `Length Encoded String 2` element is 65535 length units. • `Null Terminated String`. The data string ends with the hexadecimal NULL character, `X'00'`. • `Packed Decimal`. The data is a packed decimal number that equates to the COMP-3 data type in COBOL. • `Binary`. The data is in bit string format. • `Time Seconds`. This value supports C time_t, and Java Date and Time objects. It is valid only if the `DateTime Format` property represents numeric-only data. • `Time Milliseconds`. This value supports C time_t, and Java Date and Time objects. It is valid only if the `DateTime Format` property represents numeric-only data. The default is dependent on the `Messaging Standard` property. For all other `Messaging Standard` values, the `Physical Type` property is set to `Text`.

Property	Type	Meaning
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if one of the following statements is true: • `Physical Type` is `Packed Decimal`. • `Physical Type` is `Text`, no `Length Reference` is specified, and the `Data Element Separation` of the parent complex type or group is `Fixed Length`, `Tagged Fixed Length`, or `Fixed Length AL3`. The default is dependent on the physical type of the object. If `Physical Type` is `Length Encoded String 1`, `Length Encoded String 2`, or `Null Terminated String`, this property is not applicable. If `Physical Type` is `Time Seconds`, the value of this property is 4, and cannot be changed. If `Physical Type` is `Time Milliseconds`, the value of this property is 8, and cannot be changed.
Length Units	Enumerated type	Select the unit of length for the object. Select one of the following options (some physical types do not offer both options): • `Bytes`. The length is given in bytes. • `Characters`. The length is given in characters. The number of bytes that are processed in the bit stream depends on the code page of the message. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the contents of the bit stream. The parser reads one character at a time and determines whether the character comprises one or more bytes. The default is dependent on the physical type of the object.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • `Not Applicable` • `Left Justify` • `Right Justify`

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the drop-down list. • Enter a character between quotation marks, for example "c" or 'c', where c is any alphanumeric character. • Enter a hexadecimal character code in the form `0xYY`, where `YY` is a hexadecimal value. • Enter a Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.
DateTime Format	String	Specify a template for date and time. The default DateTime format is dependent on the logical type of the object. For information about the defaults for the dateTime format according to the logical type, see “DateTime defaults by logical type” on page 6320. See “DateTime formats” on page 6310 for details of date and time formats.

Numeric representation

Property	Type	Meaning
Signed	Check box	Specify whether the value is signed. This property is applicable only if the `Physical type` property is `Packed Decimal`. By default, this check box is cleared, which indicates that the value is not signed.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the list: • `NULLPadFill`. This option is valid only for fixed-length objects and is the default value. • `NULLLogicalValue`. The `Encoding Null Value` property is first converted to an actual value, and rendered in the way specified for the field. • `NULLLiteralValue`. This option specifies that `Encoding Null Value` contains a value that is directly substituted as if it is a string. For dateTime elements, use this option if you want to use the `Encoding Null Value` property to test or compare the content of the field in the message. • `NULLLiteralFill`. This option specifies that the field is filled with the value specified by the `Encoding Null Value` property. `Encoding Null Value` must resolve to a single character. The option that you select determines the value that you must set for the property `Encoding Null Value`. For full information about using these options, see "TDS Null handling options" on page 6293.
Encoding Null Value	String	The use of this property depends on the `Encoding Null` property. The default value is zero. If you set the `Encoding Null` property for a dateTime object to `NULLLogicalValue`, the value that you set must be in an ISO8601 dateTime format. These formats are described in "DateTime as string data" on page 6311. For example, specify a value that conforms to the `yyyy-MM-dd'T'HH:mm:ss` format; for example, `1970-12-01`.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

TDS properties for global element decimal types:

The TDS format properties for global element decimal types.

The TDS Format properties described here apply to:

- Objects: Global Element
- Decimal schema types: decimal, integer, negativeInteger, nonNegativeInteger, nonPositiveInteger, positiveInteger, unsignedLong

Field Identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the `Data Element Separation` property of the complex type or types in which the object is a child is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, this property must contain a non-empty value. If the object is a complex element, and the `Data Element Separation` property of its parent is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, the property can contain an empty value. The value for this property must be unique for every element in the message set; that is, no two elements in the message set can contain the same value for this property.
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the `Data Element Separation` method is set to `Use Data Pattern` in the complex type. For more details, see "Regular expression syntax" on page 6304.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the `Messaging Standard` property of the message set is `User Defined Text`, `User Defined Mixed`, `CSV`, or `TLOG`, select one of the following values: • `Text`. The data is in character format. • `Packed Decimal`. The data is a packed decimal number that equates to the COMP-3 data type in COBOL. • `TLOG Specific` - this option can be selected only if the `Message Standard` property of the message set is `TLOG`. This option indicates that the format of the data is specific to the TLOG messaging standard. • `Integer`. Equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • `External Decimal`. The data is a decimal number that equates to the data type PIC 9 USAGE DISPLAY in COBOL. The default is dependent on the `Messaging Standard` property. For all other `Messaging Standard` values, the `Physical Type` property is set to `Text`.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no `Length Reference` is specified. If this property is not set and the message set property `Derive default length from logical type` is selected, and the `Physical type` is 'Character', the default value is derived from any totalDigits value constraint (schema facet) or, if none, any `minInclusive`, `maxInclusive`, `minExclusive`, or `maxExclusive` value constraints (schema facets), on the simple type.
Length Units	Enumerated type	Select the unit of length for the object. Select one of the following options (some physical types do not offer both options): • `Bytes`. The length is given in bytes. • `Characters`. The length is given in characters. The number of bytes that are processed in the bit stream depends on the code page of the message. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the contents of the bit stream. The parser reads one character at a time and determines whether the character comprises one or more bytes. The default is dependent on the physical type of the object.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • `Not Applicable` • `Left Justify` • `Right Justify`

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the drop-down list. • Enter a character between quotation marks, for example "c" or 'c', where c is any alphanumeric character. • Enter a hexadecimal character code in the form `0xYY`, where `YY` is a hexadecimal value. • Enter a Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.

Numeric representation

Property	Type	Meaning
Signed	Check box	Specify whether the value is signed.
Sign EBCDIC Custom Overpunched	Check box	Specify whether EBCDIC custom sign format is used. This property is applicable only if the `Signed` property is selected and the `Physical Type` property is set to `External Decimal`.

Property	Type	Meaning
Sign Orientation	Enumerated type	The values that you can choose for this property are: • Leading Separate • Trailing Separate • Leading Overpunched • Trailing Overpunched This property is enabled only if you have set Physical Type to Text or External Decimal, and you have selected Signed. If Physical Type is Text, the only valid values of Sign Orientation are Leading Separate and Trailing Separate. If Physical Type is External Decimal and Sign EBCDIC Customer Overpunched is selected, the only valid values of Sign Orientation are Leading Overpunched and Trailing Overpunched.
Positive Sign	String	Specify the value that represents the positive symbol. Do not specify a numeric value. If no value is set, "+" is assumed. The positive sign is not written when creating an output message; it is used only to recognize the positive sign when parsing a message bit stream. This property is applicable only if Physical Type is Text and Signed is selected.
Negative Sign	String	Specify the value that represents the negative symbol. Do not specify a numeric value. If no value is set, "-" is assumed. This property is applicable only if Physical Type is Text and Signed is selected.
Virtual Decimal Point	Button and Integer	Specify a non-zero integer that represents the position of an implied decimal point within a number, or specify 0, the default, to use the formatting of Float or Decimal numbers as specified by the Precision property. If you specify a positive integer, the position of the decimal point is moved left from the right side of the number. For example, if you specify 3, the decimal value 1234 represents 1.234 If you specify a negative integer, the position of the decimal point is moved right from the right side of the number. For example, if you specify -3, the decimal value 1234 represents 1,234,000.

Property	Type	Meaning
Precision	Button and Integer	This property is applicable only if `Physical Type` is `Text`. It is used if the value of the `Virtual Decimal Point` property is 0, which indicates that the decimal point is present in the data. It deals with truncation, and specifies how many digits are to follow the decimal point. Either specify a number of digits: • If you set `Precision` to 0, data is truncated so that the fractional part is lost. For example, the value `123.45` is truncated to `123`. • If you set `Precision` to a number less than the number of fractional digits, data is truncated. For example, the value `123.4567` is truncated to `123.45` if you set `Precision` to 2. • If you set `Precision` to a number greater than the number of fractional digits, the value is padded with extra zeros. For example, the value `12.345` is padded to `12.34500` if you set `Precision` to 5. Or select one option from the list: • `All Significant Digits - decimal separator only required if fractional digits` (the default): all significant digits are written to the output bit stream, and no decimal separator is written if no fractional digits are present. • `Explicit Decimal Separator - decimal separator always required`: all significant digits are written to the output bit stream and the decimal separator is always included, even when no fractional digits are present. The decimal separator must be present in the input bit stream, even when no fractional digits are present.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the list: • `NULLPadFill`. This option is valid only for fixed-length objects and is the default value. • `NULLLogicalValue`. The `Encoding Null Value` property is first converted to an actual value, and rendered in the way specified for the field. • `NULLLiteralValue`. This option specifies that `Encoding Null Value` contains a value that is directly substituted as if it is a string. For dateTime elements, use this option if you want to use the `Encoding Null Value` property to test or compare the content of the field in the message. • `NULLLiteralFill`. This option specifies that the field is filled with the value specified by the `Encoding Null Value` property. `Encoding Null Value` must resolve to a single character. The option that you select determines the value that you must set for the property `Encoding Null Value`. For full information about using these options, see “TDS Null handling options” on page 6293.
Encoding Null Value	String	The use of this property depends on the `Encoding Null` property. The default value is zero. If you set the `Encoding Null` property for a dateTime object to `NULLLogicalValue`, the value that you set must be in an ISO8601 dateTime format. These formats are described in “DateTime as string data” on page 6311. For example, specify a value that conforms to the `yyyy-MM-dd'T'HH:mm:ss` format; for example, `1970-12-01`.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

TDS properties for global element float types:

The TDS format properties for global element float types.

The TDS Format properties described here apply to:

- Objects: Global Element
- Float schema types: double, float

Field Identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the `Data Element Separation` property of the complex type or types in which the object is a child is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, this property must contain a non-empty value. If the object is a complex element, and the `Data Element Separation` property of its parent is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, the property can contain an empty value. The value for this property must be unique for every element in the message set; that is, no two elements in the message set can contain the same value for this property.
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the `Data Element Separation` method is set to `Use Data Pattern` in the complex type. For more details, see "Regular expression syntax" on page 6304.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the `Messaging Standard` property of the message set is `User Defined Text`, `User Defined Mixed`, `CSV`, or `TLOG`, select one of the following values: • `Text`. The data is in character format. • `Float`. Equates to the data type FLOAT or DOUBLE in C, or the COMP-1 or COMP-2 numeric data type in COBOL. • `Integer`. Equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • `Packed Decimal`. The data is a packed decimal number that equates to the COMP-3 data type in COBOL. • `External Decimal`. The data is a decimal number that equates to the data type PIC 9 USAGE DISPLAY in COBOL. The default is dependent on the `Messaging Standard` property. For all other `Messaging Standard` values, the `Physical Type` property is set to `Text`.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no `Length Reference` is specified.

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the object. Select one of the following options (some physical types do not offer both options): • `Bytes`. The length is given in bytes. • `Characters`. The length is given in characters. The number of bytes that are processed in the bit stream depends on the code page of the message. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the contents of the bit stream. The parser reads one character at a time and determines whether the character comprises one or more bytes. The default is dependent on the physical type of the object.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • `Not Applicable` • `Left Justify` • `Right Justify`

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the drop-down list. • Enter a character between quotation marks, for example "c" or 'c', where c is any alphanumeric character. • Enter a hexadecimal character code in the form `0xYY`, where `YY` is a hexadecimal value. • Enter a Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.

Numeric representation

Property	Type	Meaning
Signed	Check box	Specify whether the value is signed.
Sign EBCDIC Custom Overpunched	Check box	Specify whether EBCDIC custom sign format is used. This property is applicable only if the `Signed` property is selected and the `Physical Type` property is set to `External Decimal`.

Property	Type	Meaning
Sign Orientation	Enumerated type	The values that you can choose for this property are: • Leading Separate • Trailing Separate • Leading Overpunched • Trailing Overpunched This property is enabled only if you have set Physical Type to Text or External Decimal, and you have selected Signed. If Physical Type is Text, the only valid values of Sign Orientation are Leading Separate and Trailing Separate. If Physical Type is External Decimal and Sign EBCDIC Customer Overpunched is selected, the only valid values of Sign Orientation are Leading Overpunched and Trailing Overpunched.
Positive Sign	String	Specify the value that represents the positive symbol. Do not specify a numeric value. If no value is set, "+" is assumed. The positive sign is not written when creating an output message; it is used only to recognize the positive sign when parsing a message bit stream. This property is applicable only if Physical Type is Text and Signed is selected.
Negative Sign	String	Specify the value that represents the negative symbol. Do not specify a numeric value. If no value is set, "-" is assumed. This property is applicable only if Physical Type is Text and Signed is selected.
Virtual Decimal Point	Button and Integer	Specify a non-zero integer that represents the position of an implied decimal point within a number, or specify 0, the default, to use the formatting of Float or Decimal numbers as specified by the Precision property. If you specify a positive integer, the position of the decimal point is moved left from the right side of the number. For example, if you specify 3, the decimal value 1234 represents 1.234 If you specify a negative integer, the position of the decimal point is moved right from the right side of the number. For example, if you specify -3, the decimal value 1234 represents 1,234,000.

Property	Type	Meaning
Precision	Button and Integer	This property is applicable only if `Physical Type` is `Text`. It is used if the value of the `Virtual Decimal Point` property is 0, which indicates that the decimal point is present in the data. It deals with truncation, and specifies how many digits are to follow the decimal point. Either specify a number of digits: • If you set `Precision` to 0, data is truncated so that the fractional part is lost. For example, the value `123.45` is truncated to `123`. • If you set `Precision` to a number less than the number of fractional digits, data is truncated. For example, the value `123.4567` is truncated to `123.45` if you set `Precision` to 2. • If you set `Precision` to a number greater than the number of fractional digits, the value is padded with extra zeros. For example, the value `12.345` is padded to `12.34500` if you set `Precision` to 5. Or select one option from the list: • `All Significant Digits - decimal separator only required if fractional digits` (the default): all significant digits are written to the output bit stream, and no decimal separator is written if no fractional digits are present. • `Explicit Decimal Separator - decimal separator always required`: all significant digits are written to the output bit stream and the decimal separator is always included, even when no fractional digits are present. The decimal separator must be present in the input bit stream, even when no fractional digits are present. • `Exponential Notation - Example "1.23456e002"`: data is written out to the bit stream as a signed value having the format `[sign1]a.bbbe[sign2]ccc` where: – `[sign1]` is the value of `Negative Sign` if the value is negative – `a` is a single decimal digit – `bbb` is one or more decimal digits – `[sign2]` is the value of `Negative Sign` if the exponent is negative – `ccc` is exactly three decimal digits (the exponent) `[sign1]` and `[sign2]` are absent if the value and exponent are positive. For example, the value -123.456 is represented as `-1.23456e002` and the value 0.00012 is represented as `1.2e-004` in the output bit stream, assuming that the value of `Negative Sign` is "-", and the value of `Sign Orientation` is `Leading`. The value -0.00012 is represented as `1.2*e*004` if `Negative Sign` is "*" and `Sign Orientation` is `Trailing`.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the list: • `NULLPadFill`. This option is valid only for fixed-length objects and is the default value. • `NULLLogicalValue`. The `Encoding Null Value` property is first converted to an actual value, and rendered in the way specified for the field. • `NULLLiteralValue`. This option specifies that `Encoding Null Value` contains a value that is directly substituted as if it is a string. For dateTime elements, use this option if you want to use the `Encoding Null Value` property to test or compare the content of the field in the message. • `NULLLiteralFill`. This option specifies that the field is filled with the value specified by the `Encoding Null Value` property. `Encoding Null Value` must resolve to a single character. The option that you select determines the value that you must set for the property `Encoding Null Value`. For full information about using these options, see "TDS Null handling options" on page 6293.
Encoding Null Value	String	The use of this property depends on the `Encoding Null` property. The default value is zero. If you set the `Encoding Null` property for a dateTime object to `NULLLogicalValue`, the value that you set must be in an ISO8601 dateTime format. These formats are described in "DateTime as string data" on page 6311. For example, specify a value that conforms to the `yyyy-MM-dd'T'HH:mm:ss` format; for example, `1970-12-01`.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

TDS properties for global element integer types:

The TDS format properties for global element integer types.

The TDS Format properties described here apply to:

- Objects: Global Element
- Integer schema types: byte, int, long, short, unsignedByte, unsignedInt, unsignedShort

Field Identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the `Data Element Separation` property of the complex type or types in which the object is a child is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, this property must contain a non-empty value. If the object is a complex element, and the `Data Element Separation` property of its parent is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, the property can contain an empty value. The value for this property must be unique for every element in the message set; that is, no two elements in the message set can contain the same value for this property.
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the `Data Element Separation` method is set to `Use Data Pattern` in the complex type. For more details, see "Regular expression syntax" on page 6304.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the `Messaging Standard` property of the message set is `User Defined Text`, `User Defined Mixed`, `CSV`, or `TLOG`, select one of the following values: • `Text`. The data is in character format. • `Integer`. Equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • `Packed Decimal`. The data is a packed decimal number that equates to the COMP-3 data type in COBOL. • `External Decimal`. The data is a decimal number that equates to the data type PIC 9 USAGE DISPLAY in COBOL. The default is dependent on the `Messaging Standard` property. For all other `Messaging Standard` values, the `Physical Type` property is set to `Text`.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no `Length Reference` is specified. If this property is not set and the message set property `Derive default length from logical type` is selected, and the `Physical type` is 'Character', the default value is derived from any totalDigits value constraint (schema facet) or, if none, any `minInclusive`, `maxInclusive`, `minExclusive`, or `maxExclusive` value constraints (schema facets), on the simple type.
Length Units	Enumerated type	Select the unit of length for the object. Select one of the following options (some physical types do not offer both options): • `Bytes`. The length is given in bytes. • `Characters`. The length is given in characters. The number of bytes that are processed in the bit stream depends on the code page of the message. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the contents of the bit stream. The parser reads one character at a time and determines whether the character comprises one or more bytes. The default is dependent on the physical type of the object.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • `Not Applicable` • `Left Justify` • `Right Justify`

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the drop-down list. • Enter a character between quotation marks, for example "c" or 'c', where c is any alphanumeric character. • Enter a hexadecimal character code in the form `0xYY`, where `YY` is a hexadecimal value. • Enter a Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.

Numeric representation

Property	Type	Meaning
Signed	Check box	Specify whether the value is signed.
Sign EBCDIC Custom Overpunched	Check box	Specify whether EBCDIC custom sign format is used. This property is applicable only if the `Signed` property is selected and the `Physical Type` property is set to `External Decimal`.

Property	Type	Meaning
Sign Orientation	Enumerated type	The values that you can choose for this property are: • Leading Separate • Trailing Separate • Leading Overpunched • Trailing Overpunched This property is enabled only if you have set Physical Type to Text or External Decimal, and you have selected Signed. If Physical Type is Text, the only valid values of Sign Orientation are Leading Separate and Trailing Separate. If Physical Type is External Decimal and Sign EBCDIC Customer Overpunched is selected, the only valid values of Sign Orientation are Leading Overpunched and Trailing Overpunched.
Positive Sign	String	Specify the value that represents the positive symbol. Do not specify a numeric value. If no value is set, "+" is assumed. The positive sign is not written when creating an output message; it is used only to recognize the positive sign when parsing a message bit stream. This property is applicable only if Physical Type is Text and Signed is selected.
Negative Sign	String	Specify the value that represents the negative symbol. Do not specify a numeric value. If no value is set, "-" is assumed. This property is applicable only if Physical Type is Text and Signed is selected.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the list: • NULLPadFill. This option is valid only for fixed-length objects and is the default value. • NULLLogicalValue. The Encoding Null Value property is first converted to an actual value, and rendered in the way specified for the field. • NULLLiteralValue. This option specifies that Encoding Null Value contains a value that is directly substituted as if it is a string. For dateTime elements, use this option if you want to use the Encoding Null Value property to test or compare the content of the field in the message. • NULLLiteralFill. This option specifies that the field is filled with the value specified by the Encoding Null Value property. Encoding Null Value must resolve to a single character. The option that you select determines the value that you must set for the property Encoding Null Value. For full information about using these options, see “TDS Null handling options” on page 6293.
Encoding Null Value	String	The use of this property depends on the Encoding Null property. The default value is zero. If you set the Encoding Null property for a dateTime object to NULLLogicalValue, the value that you set must be in an ISO8601 dateTime format. These formats are described in “DateTime as string data” on page 6311. For example, specify a value that conforms to the yyyy-MM-dd'T'HH:mm:ss format; for example, 1970-12-01.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

TDS properties for global element interval types:

The TDS format properties for global element string types.

The TDS Format properties described here apply to:

- Objects: Global Element
- Interval schema types: duration

Field Identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the `Data Element Separation` property of the complex type or types in which the object is a child is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, this property must contain a non-empty value. If the object is a complex element, and the `Data Element Separation` property of its parent is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, the property can contain an empty value. The value for this property must be unique for every element in the message set; that is, no two elements in the message set can contain the same value for this property.
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the `Data Element Separation` method is set to `Use Data Pattern` in the complex type. For more details, see “Regular expression syntax” on page 6304.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the `Messaging Standard` property of the message set is `User Defined Text`, `User Defined Mixed`, `CSV`, or `TLOG`, select one of the following values: • `Text`. The data is in character format. • `Length Encoded String 1`. The first byte of the data contains the length (in length units) of the data string that follows the length byte. The maximum length of a `Length Encoded String 1` element is 255 length units. • `Length Encoded String 2`. The data's first 2 bytes contains the length (in length units) of the data string that follows the two length bytes. The maximum length of a `Length Encoded String 2` element is 65535 length units. • `Null Terminated String`. The data string ends with the hexadecimal NULL character, `X'00'`. The default is dependent on the `Messaging Standard` property. For all other `Messaging Standard` values, the `Physical Type` property is set to `Text`.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no `Length Reference` is specified. If this property is not set and the message set property `Derive default length from logical type` is selected, and the `Physical type` is 'Character', the default value is derived from any length or maxLength value constraint (schema facet) on the simple type.

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the object. Select one of the following options (some physical types do not offer both options): • `Bytes`. The length is given in bytes. • `Characters`. The length is given in characters. The number of bytes that are processed in the bit stream depends on the code page of the message. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the contents of the bit stream. The parser reads one character at a time and determines whether the character comprises one or more bytes. The default is dependent on the physical type of the object.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • `Not Applicable` • `Left Justify` • `Right Justify`

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the drop-down list. • Enter a character between quotation marks, for example "c" or 'c', where c is any alphanumeric character. • Enter a hexadecimal character code in the form `0xYY`, where `YY` is a hexadecimal value. • Enter a Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the list: • `NULLPadFill`. This option is valid only for fixed-length objects and is the default value. • `NULLLogicalValue`. The `Encoding Null Value` property is first converted to an actual value, and rendered in the way specified for the field. • `NULLLiteralValue`. This option specifies that `Encoding Null Value` contains a value that is directly substituted as if it is a string. For dateTime elements, use this option if you want to use the `Encoding Null Value` property to test or compare the content of the field in the message. • `NULLLiteralFill`. This option specifies that the field is filled with the value specified by the `Encoding Null Value` property. `Encoding Null Value` must resolve to a single character. The option that you select determines the value that you must set for the property `Encoding Null Value`. For full information about using these options, see "TDS Null handling options" on page 6293.
Encoding Null Value	String	The use of this property depends on the `Encoding Null` property. The default value is zero. If you set the `Encoding Null` property for a dateTime object to `NULLLogicalValue`, the value that you set must be in an ISO8601 dateTime format. These formats are described in "DateTime as string data" on page 6311. For example, specify a value that conforms to the `yyyy-MM-dd'T'HH:mm:ss` format; for example, `1970-12-01`.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

TDS properties for global element string types:

The TDS format properties for global element string types.

The TDS Format properties described here apply to:

- Objects: Global Element
- String schema types: anyURI, ENTITIES, ENTITY, ID, IDREF, IDREFS, language, Name, NCName, NMTOKEN, NMTOKENS, normalizedString, NOTATION, QName, string, token

Field Identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the `Data Element Separation` property of the complex type or types in which the object is a child is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, this property must contain a non-empty value. If the object is a complex element, and the `Data Element Separation` property of its parent is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, the property can contain an empty value. The value for this property must be unique for every element in the message set; that is, no two elements in the message set can contain the same value for this property.
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the `Data Element Separation` method is set to `Use Data Pattern` in the complex type. For more details, see "Regular expression syntax" on page 6304.

Property	Type	Meaning
Interpret Element Value	Enumerated type	Specify whether values stored within this object are interpreted as having significance for the parser and, if so, the type of interpretation that occurs. This interpretation is standard-specific and is therefore hard coded. The possible values for this property are: • `None` (the default value) • `EDIFACT Service String` • `X12 Service String` • `Message Key` • `EDIFACT Syntax Level ID` • `HL7 Service String` • `HL7 Field Separator` **Note:** The `Message Key` enumeration has been deprecated.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the `Messaging Standard` property of the message set is `User Defined Text`, `User Defined Mixed`, `CSV`, or `TLOG`, select one of the following values: • `Text`. The data is in character format. • `Length Encoded String 1`. The first byte of the data contains the length (in length units) of the data string that follows the length byte. The maximum length of a `Length Encoded String 1` element is 255 length units. • `Length Encoded String 2`. The first two bytes of the data contain the length (in length units) of the data string that follows the two length bytes. The maximum length of a `Length Encoded String 2` element is 65535 length units. • `Null Terminated String`. The data string ends with the hexadecimal NULL character, `X'00'`. • `TLOG Specific` - this option can be selected only if the `Message Standard` property of the message set is `TLOG`. This option indicates that the format of the data is specific to the TLOG messaging standard. The default is dependent on the `Messaging Standard` property. For all other `Messaging Standard` values, the `Physical Type` property is set to `Text`.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no `Length Reference` is specified. If this property is not set and the message set property `Derive default length from logical type` is selected, and the `Physical type` is 'Character', the default value is derived from any length or maxLength value constraint (schema facet) on the simple type.

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the object. Select one of the following options (some physical types do not offer both options): • `Bytes`. The length is given in bytes. • `Characters`. The length is given in characters. The number of bytes that are processed in the bit stream depends on the code page of the message. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the contents of the bit stream. The parser reads one character at a time and determines whether the character comprises one or more bytes. The default is dependent on the physical type of the object.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • `Not Applicable` • `Left Justify` • `Right Justify`

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the drop-down list. • Enter a character between quotation marks, for example "c" or 'c', where c is any alphanumeric character. • Enter a hexadecimal character code in the form `0xYY`, where `YY` is a hexadecimal value. • Enter a Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the list: • `NULLPadFill`. This option is valid only for fixed-length objects and is the default value. • `NULLLogicalValue`. The `Encoding Null Value` property is first converted to an actual value, and rendered in the way specified for the field. • `NULLLiteralValue`. This option specifies that `Encoding Null Value` contains a value that is directly substituted as if it is a string. For dateTime elements, use this option if you want to use the `Encoding Null Value` property to test or compare the content of the field in the message. • `NULLLiteralFill`. This option specifies that the field is filled with the value specified by the `Encoding Null Value` property. `Encoding Null Value` must resolve to a single character. The option that you select determines the value that you must set for the property `Encoding Null Value`. For full information about using these options, see "TDS Null handling options" on page 6293.
Encoding Null Value	String	The use of this property depends on the `Encoding Null` property. The default value is zero. If you set the `Encoding Null` property for a dateTime object to `NULLLogicalValue`, the value that you set must be in an ISO8601 dateTime format. These formats are described in "DateTime as string data" on page 6311. For example, specify a value that conforms to the `yyyy-MM-dd'T'HH:mm:ss` format; for example, `1970-12-01`.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

Documentation properties for all message set objects:

Use the documentation property of an object to add information that describes the function of the object.

The documentation property is available on all objects except Key, Keyref, and Unique objects.

The property is a string field, and you can use any standard alphanumeric characters.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

Global group properties:

Different types of properties are available for a global group.

A global element can have the following properties;

- "Global group logical properties" on page 5433
- "Global group CWF properties" on page 5464
- "Global group XML properties" on page 5488
- "Global group TDS properties" on page 5516
- "Documentation properties for all message set objects" on page 5413

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

Global group logical properties:

The logical properties of a global group.

Valid children in a global group that depend on both **`Composition`** and **`Content Validation`** are shown in "MRM content validation" on page 5422.

Property	Type	Meaning
Name	String	Specify a name for the object when you create it. Names can consist of alphanumeric characters, including the letters *A* through *Z*, *a* through *z*, and the digits *0* through *9*. They might also include the following punctuation characters; • - the hyphen • _ the underscore • . the period Names can start only with a letter or the underscore character, and not with a number, hyphen, or period. Names that begin with `xml`, or any variant of these characters (for example `XmL`), are reserved by the XML standards specification. Further details of naming conventions and allowable characters can be found in the Extensible Markup Language (XML) specification that can be found on the World Wide Web Consortium (W3C) Web site.

Property	Type	Meaning
Composition	Enumerated type	Define the order, and the number of occurrences, of the elements and groups in your messages. Composition does not affect the attributes in a complex type. Select from the following options: • Empty • sequence. If you select this option, you can define members that are elements or groups. These members, if present, must appear in the specified order in the message. They can repeat, and the same element or group can appear more than once. • choice. If you select this option, you can define members that are elements or groups. Exactly one of the defined members must be present in the message, and can repeat. Use this option if you want to model C unions and COBOL REDEFINES in a Custom Wire Format, or an XML DTD element that uses choice in an XML Wire Format, or a SWIFT field that has more than one option. • all. If you select this option, you can define members that are elements; groups are not allowed. The elements in an all group can appear in any order. Each element can appear once, or not at all. An all group can be used only at the top level of a complex type; it cannot be a member of another group within a type. • unorderedSet. This option is supported only by the MRM domain.If you select this option, you can define members that are elements. The elements can repeat, but the same element cannot appear twice in the list of members. The elements can appear in any order in the message. • orderedSet. This option is supported only by the MRM domain.If you select this option, you can define members that are elements. The elements can repeat, but the same element cannot appear twice in the list of members. The elements must appear in the specified order in the message. • message. This option is supported only by the MRM domain. If you select this option, you can define only messages as members. Each member can repeat, but the same message cannot appear twice in the list of members. Like choice, only one of the defined members can be present in a message. Unlike choice, when writing a message, if the complex type or group has more than one member, the bit stream is not padded to the length of the longest member. Use this option to model multipart messages, which are used in some industry standards; for example, SWIFT. For more information, see the section on multipart messages in “Multipart messages” on page 1191.

Property	Type	Meaning
Content Validation	Enumerated type	`Content Validation` is used only by the MRM domain. If validation is enabled in your message flow, `Content Validation` specifies the strictness of the MRM validation for members of a complex type or group. See "MRM content validation" on page 5422 for further details. Select from the following options: • Closed. The complex type can only contain the child elements that you have added to it. • Open Defined. The complex type can contain any valid element defined within the message set. • Open. The complex type can contain any valid element, not just those that you have added to this complex type. See "Combinations of Composition and Content Validation" on page 5612 for further details of these options.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Global group CWF properties:

There are no properties to show.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM Custom Wire Format" on page 1214
Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not separated by delimiters.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Global group XML properties:

There are no properties to show.

There are no properties to show.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Global group TDS properties:

The TDS properties of a global group.

Field Identification

Property	Type	Meaning
Data Element Separation	Enumerated type	Select one of the following values to specify the method that is used to separate the data elements within the type. • `Tagged Delimited`. This value indicates that all elements within the complex type are identified by a tag, and, if a value is specified in the optional `Delimiter` property, are separated by that value. You must set the `Tag` property for all child elements of simple type, and you can set the `Delimiter` property to a non-empty value. See "Global element TDS properties" on page 5513. You must also set the `Tag Data Separator` or the `Length of Tag` property. • `Tagged Fixed Length`. This value indicates that each element is identified by a tag, and the data has a fixed length. There are no delimiters. You must set the `Tag` property for each of the child elements of this complex type, and each child element must have a `Length` or `Length Reference` property assigned to it. You must also set the `Tag Data Separator` or the `Length of Tag` property. • `Tagged Encoded Length`. This value indicates that all elements within the complex type are separated by a tag, and a length field follows each tag. There are no delimiters. The tag can be fixed length, as set by the `Length of Tag` property, or variable length delimited by the `Tag Data Separator` property. You must also set the `Length Of Encoded Length` property so that the parser knows the size of the length field, and set the `Extra Chars in Encoded Length` property. This property tells the parser what to subtract from the value in the `Length Of Encoded Length` property to get the actual length of the data that follows the length field. This method provides a more flexible way of handling ACORD AL3 standard messages than using the `Fixed Length AL3` value, by allowing different parts of the messages to be at different versions of the ACORD AL3 standard. • `All Elements Delimited`. This value indicates that all elements within the complex type are separated by a delimiter. You must set a value in the `Delimiter` property. • `Variable Length Elements Delimited`. This value indicates that some of the elements within the complex type might be of variable length. Variable length elements must be delimited by the value specified in the `Delimiter` property. • `Use Data Pattern`. This value indicates that the parser determines the elements by matching the data with the regular expression that is set in the `Data Pattern` property of the element or type member. See "Message definition file properties" on page 5409. • `Fixed Length`. This value indicates that all elements within the complex type are fixed length. The next data element is accessed by adding the value of the `Length` property to the offset. See "Global element TDS properties" on page 5513. If you set the `Data Element Separation` property of a complex type to `Fixed Length`, you must also set the `Data Element Separation` property of all complex children of this type to `Fixed Length`. Each child element must have a `Length` or `Length Reference` property assigned to it. • `Fixed Length AL3`. This value has a similar meaning to the separation type `Fixed Length`, but also indicates to the parser that a number of predefined rules regarding missing optional elements, encoded lengths, and versioning, must be applied. If you set the `Data Element Separation` property of a complex type to `Fixed Length AL3`, you must also set the `Data Element Separation` property of all complex children of this type to `Fixed Length AL3`. • `Undefined`. This value is set automatically if you set the `Type Composition` property of a complex type to `Message`, and you cannot change it to any other value. Do not set the `Type Composition` property to `Empty`, `Choice`, `Unordered Set`, `Ordered Set`, `Sequence`, or `Simple Unordered Set`. If you do, you cannot check in the type.

Property	Type	Meaning
Group Indicator	String	Specify the value of a special character, or string, that precedes the data that belongs to a group, or a complex type, within the bit stream.
Group Terminator	String	Specify the value of a special character, or string, that terminates the data that belongs to a group, or a complex type, within the bit stream.
Delimiter	String	Specify the value of a special character, or string, that specifies the delimiter that is used between data elements. This property applies only to the delimited `Data Element Separation` methods (`Tagged Delimited`, `All Elements Delimited`, and `Variable Length Elements Delimited`).
Suppress Absent Element Delimiters	Enumerated type	Use this property to select whether you want delimiters to be suppressed for elements that are missing within a message. Select from: • `End Of Type`. Use this option to suppress the delimiter when an element is missing. For example, if the model has been defined to have up to three elements and only two are present, the last delimiter can be omitted from the message. • `Never`. Use this option to ensure that even if optional elements are not present, all delimiters are written out. Use this option when the same delimiter is used to delimit parent and child objects. For example, if an optional child element is missing, message processing applications cannot tell where the child elements in a message end and the next parent element starts, if the delimiters are all the same.
Observe Element Length	Check box	This property is applicable when `Data Element Separation` is `All Elements Delimited` or `Tagged Delimited`. Select this check box if the `Length` property of child simple elements is significant when parsing and writing. • During parsing, an exception is thrown if the length of the extracted data exceeds the specified length. Otherwise, the data is trimmed according to the `Justification` and `Padding Character` properties of the child element. • During writing, an exception is thrown if the data to write exceeds the specified length. Otherwise, the data is padded according to the `Justification` and `Padding Character` properties of the child element. Clear this check box to ignore the `Length` property when parsing and writing. The default value depends on the setting of the `Messaging Standard` property (at the message set level) and the `Data Element Separation` property. • If `Data Element Separation` is `All Elements Delimited` and the `Messaging Standard` is `TLOG`, the check box is selected. • If `Data Element Separation` is `All Elements Delimited` and the `Messaging Standard` is other than `TLOG`, the check box is cleared. • If `Data Element Separation` is `Tagged Delimited`, the check box is cleared. For all other data element separation methods, the check box is disabled and does not influence the behavior of the TDS parser.
Tag Data Separator	Button and String	Specify the value of a special character or string that separates the Tag from the data. The `Tag Data Separator` and `Length of Tag` properties are mutually exclusive. If you set the property `Tag Data Separator`, it overrides `Length of Tag`. This property applies only to the tagged `Data Element Separation` methods (`Tagged Delimited`, `Tagged Fixed Length`, and `Tagged Encoded Length`).

Property	Type	Meaning
Length of Tag	Button and Integer	Specify the length of a tag value. When the message is parsed, this property allows tags to be extracted from the bit stream if the `Tag Data Separator` property is not set. The `Tag Data Separator` and `Length of Tag` properties are mutually exclusive. If you set the property `Tag Data Separator`, it overrides this value. This property applies only to the tagged `Data Element Separation` methods (`Tagged Delimited`, `Tagged Fixed Length`, and `Tagged Encoded Length`).
Length of Encoded Length	Integer	Specifies the number of characters (not bytes) after a tag that are used for the length field. Enter a value from 0 to 2147483647. You must set this property if you have set the `Data Element Separation` property to `Tagged Encoded Length`; it is not valid otherwise. The actual number of data characters that are parsed depends on the value of the `Extra Chars in Encoded Length` property.
Extra Chars in Encoded Length	Integer	(Only valid if the `Data Element Separation` method is set to `Tagged Encoded Length`.) Specifies the number of extra characters included in the value found in the length field. (For example, the value in the length might include the size of the length field itself as well as the size of the data field, or it might be the total size of the tag, length, and data fields.) Enter a value from 0 to 2147483647. The parser subtracts this number from the number found in the length field to get the number of data characters that follow the length field. You must set this property if you have set the `Data Element Separation` property to `Tagged Encoded Length`, and the actual number of data characters is less than the value found in the length field.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM TDS format" on page 1221
The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Documentation properties for all message set objects:

Use the documentation property of an object to add information that describes the function of the object.

The documentation property is available on all objects except Key, Keyref, and Unique objects.

The property is a string field, and you can use any standard alphanumeric characters.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

Group reference properties:

Different types of properties are available for a group reference.

A group reference can have the following properties;

- "Group reference logical properties" on page 5436
- "Group reference CWF properties" on page 5465
- "Group reference XML properties" on page 5488
- "Group reference TDS properties" on page 5520
- "Documentation properties for all message set objects" on page 5413

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

Group reference logical properties:

The logical properties of a group reference include properties that specify the number of occurrences of the group reference.

Property	Type	Meaning
Reference Name	Enumerated type	The `Reference Name` is the name of the object that this object is referring to. The objects available to reference can be selected from the list.

Occurrence properties

Property	Type	Meaning
Min Occurs	Integer	Specify the minimum number of times that the object can repeat. The default value is 1. If the value is set to 0, the object is optional. With the exception of `Max Occurs` being set to -1, if a value is set for `Min Occurs`, it must be less than or equal to the value in `Max Occurs`.

Property	Type	Meaning
Max Occurs	Integer	Specify the maximum number of times that the object can repeat. The default value is 1. If this property is not set, the object cannot occur more than once. If this property is set to 0, it is interpreted as if the object does not exist in the message. It can also be set to -1, to indicate that the limit is unbounded and there is no maximum to the number of occurrences.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Group reference CWF properties:

The CWF properties of a group reference.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to all instances.

Occurrences

Property	Type	Meaning
Repeat Reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the displayed list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* logical property when parsing and writing the message, but not for validation of the message.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM Custom Wire Format" on page 1214
Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not separated by delimiters.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Group reference XML properties:

The XML properties of a group reference.

There are no properties to show.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Group reference TDS properties:

The following tables describe the TDS properties of a group reference.

Field identification

Property	Type	Meaning
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the `Data Element Separation` method is set to `Use Data Pattern` in the complex type. For more details, see "Regular expression syntax" on page 6304.

Occurrences

Property	Type	Meaning
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the `Data Element Separation` property of the parent group or complex type is set to `All Elements Delimited` or `Variable Length Elements Delimited`. A default value is provided if the previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; if the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used. If none of the previous conditions are true, a default is not applied.
Repeat reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, no integer objects exist before this one in the message structure. If a `Repeat Reference` is specified, it overrides any setting for the `Max Occurs` logical property when parsing and writing the message, but not for validating the message.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM TDS format" on page 1221
The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Documentation properties for all message set objects:

Use the documentation property of an object to add information that describes the function of the object.

The documentation property is available on all objects except Key, Keyref, and Unique objects.

The property is a string field, and you can use any standard alphanumeric characters.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

Key properties:

The different types of properties available for a key.

A key can have the following properties;

- "Key logical properties" on page 5437
- "Key CWF properties" on page 5466
- "Key XML properties" on page 5489
- "Key TDS properties" on page 5521
- "Documentation properties for all message set objects" on page 5413

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

Key logical properties:

There are no properties to show.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Key CWF properties:

There are no properties to show.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Key XML properties:

There are no properties to show.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Key TDS properties:

There are no properties to show.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Documentation properties for all message set objects:

Use the documentation property of an object to add information that describes the function of the object.

The documentation property is available on all objects except Key, Keyref, and Unique objects.

The property is a string field, and you can use any standard alphanumeric characters.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

Keyref properties:

The different types of properties available for a keyref.

A keyref can have the following properties;

- "Keyref logical properties" on page 5437
- "Keyref CWF properties" on page 5467
- "Keyref XML properties" on page 5490
- "Keyref TDS properties" on page 5521
- "Documentation properties for all message set objects" on page 5413

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

Keyref logical properties:

This describes the logical properties of a keyref.

There are no properties to show.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Keyref CWF properties:

This describes the CWF properties of a keyref.

There are no properties to show.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Keyref XML properties:

There are no properties to show.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Keyref TDS properties:

There are no properties to show.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Documentation properties for all message set objects:

Use the documentation property of an object to add information that describes the function of the object.

The documentation property is available on all objects except Key, Keyref, and Unique objects.

The property is a string field, and you can use any standard alphanumeric characters.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

Local attribute properties:

Different types of properties are available for a local attribute.

A local attribute can have the following properties;

- "Local attribute logical properties" on page 5438
- "Local attribute CWF properties" on page 5467
- "Local attribute XML properties" on page 5490
- "Local attribute TDS properties" on page 5522
- "Documentation properties for all message set objects" on page 5413

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

Local attribute logical properties:

The logical properties of a local attribute.

Property	Type	Meaning
Name	String	Specify a name for the object when you create it. Names can consist of alphanumeric characters, including the letters *A* through *Z*, *a* through *z*, and the digits *0* through *9*. They might also include the following punctuation characters; • - the hyphen • _ the underscore • . the period Names can start only with a letter or the underscore character, and not with a number, hyphen, or period. Names that begin with `xml`, or any variant of these characters (for example `XmL`), are reserved by the XML standards specification. Further details of naming conventions and allowable characters can be found in the Extensible Markup Language (XML) specification that can be found on the World Wide Web Consortium (W3C) Web site.

Property	Type	Meaning
Type	Enumerated type	The Type property constrains the type of data that can be present in the object. Select from the following options: • `int` • `string` • `Boolean` • `hexBinary` • `dateTime` • `date` • `time` • `decimal` • `float` • `(More...)` • `(New Simple Type)` • `(New Complex Type)` If you select (More...), the Type Selection wizard starts. In this wizard, you can select any of the available types. If you select (New Simple Type), the New Simple Type wizard starts. In this wizard, you can create an Anonymous simple type that is based on an existing type. This type can be created locally or globally. If you select (New Complex Type), the New Complex Type wizard starts. In this wizard, you can create an Anonymous complex type, which can be derived from an existing base type. This type can be created locally or globally. For further information about these types, and examples of their use, see the XML Schema Part 0: Primer. This document is available on the World Wide Web Consortium (W3C) Web site.
Namespace	Enumerated type	Namespaces are a simple method for qualifying element and attribute names by associating them with namespaces identified by URI references. If <no target namespace> is displayed, a namespace has not been set for this object. If the property is inactive, the message set has not been configured to support namespaces. Where the property is active, namespaces that are available for selection are displayed in the drop-down list.

Value

The *Value* properties are used in conjunction with the *Usage* property in an Attribute Reference or a Local Attribute.

Property	Type	Meaning
Default	Button and String	This property provides the default value for an element or attribute. **XMLNSC domain** When parsing with validation enabled, default values are applied to missing attributes and empty elements as required by the XML Schema 1.0 specification. When writing, elements or attributes that are missing from the message tree are not automatically added to the output XML bit stream, even if they have default values. If missing elements or attributes are required, the message tree can be serialized and then re-parsed with validation enabled. **MRM (CWF and TDS physical formats)** When writing a fixed-length portion of a message (CWF or fixed-length TDS), if an attribute or element is missing from the message tree, the default value is inserted into the bit stream so that the message structure is preserved. **MRM (XML physical format)** No support for default values **Other domains** No support for default values.
Fixed	Button and String	This property provides the fixed value for an element or attribute. **XMLNSC domain** When parsing with validation enabled, if an attribute or element is present, the value is validated against the fixed value. If the values are not equal, a validation error is signalled. Also, when parsing with validation enabled, fixed values are applied to missing attributes and empty elements as required by the XML Schema 1.0 specification. When writing, elements or attributes that are missing from the message tree are not automatically added to the output XML bit stream, even if a fixed value has been specified. If missing elements or attributes are required, the message tree can be serialized and then re-parsed with validation enabled. **MRM (CWF and TDS physical formats)** When writing a fixed-length portion of a message (CWF or fixed-length TDS), if an attribute or element is missing from the message tree, the fixed value is inserted into the bit stream so that the message structure is preserved. **MRM (XML physical format)** No support for fixed values **Other domains** No support for fixed values.

Property	Type	Meaning
Interpret Value As	Enumerated type	Specify if values stored within this object must be interpreted as having significance for the parser and, if so, the type of interpretation that must occur. Select one of the following options: • None This value is the default value, and indicates that the element or attribute does not have a key value associated with it. • MessageSetIdentity. Specifies that the value of the element or attribute corresponds to the identifier, name, or alias (in that priority order) that is associated with the message set where all subsequent embedded messages that are descendents of the enclosing message are defined. This value remains in force unless a new element or attribute MessageSetIdentity field is encountered which resets the MessageSetIdentity value. • MessageIdentity. Specifies that the value of the element or attribute corresponds to the name or alias (in that priority order) that is associated with a message, and acts as an identifier for subsequent embedded messages which are the immediate children of the enclosing message. This identity applies until a new element or attribute MessageIdentity field is encountered at the same level in the tree. The embedded message can be defined in either the current message set, or in a message set identified by using a MessageSetIdentity. **Note:** This property is applicable only when the type of the object is derived from xsd:string.

Usage properties

Property	Type	Meaning
Usage	Enumerated type	Use this property with the `Value` property found in an attribute object. The default value for the `Usage` property is optional. Select from the following options: • optional. – If the `Value` property is set to default, and no data has been entered in the `Value` property, the attribute can appear once and can have any value. – If the `Value` property is set to default, the attribute can appear once. If it does not appear, its value is the data that has been entered in the `Value` property. If it does appear, it is the value given. – Where the `Value` property is set to fixed, the attribute can appear once. If it does appear, its value must match the data that has been entered in the `Value` property. If it does not appear, its value is the data that has been entered in the `Value` property. • prohibited. The attribute must not appear. • required. – If the `Value` property is set to default, and no data has been entered in the `Value` property, the attribute must appear once and can have any value. – If the `Value` property is set to fixed, the attribute must appear once, and it must match the data that has been entered in the `Value` property.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Local attribute CWF properties:

The properties, and their permissible values, vary according to the object type.

The properties that are displayed on the object page, and the values that those properties can take, can vary according to the type of the object. For example, the properties for type *string* are different from those of type *Boolean*. Select the link for the object type from the following table.

Type of object	Properties
Binary types	• base64Binary • hexBinary
Boolean types	• Boolean
DateTime types	• date • dateTime • gDay • gMonth • gMonthDay • gYear • gYearMonth • time
Decimal types	• decimal • integer • negativeInteger • nonNegativeInteger • nonPositiveInteger • positiveInteger • unsignedLong
Float types	• double • float
Integer types	• byte • int • long • short • unsignedByte • unsignedInt • unsignedShort
Interval types	• duration

Type of object	Properties
String types	• anyURI • ENTITIES • ENTITY • ID • IDREF • IDREFS • language • Name • NCName • NMTOKEN • NMTOKENS • normalizedString • NOTATION • QName • string • token

Note: The physical format properties for simple type *duration* are the same as the physical properties of the string logical types.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM Custom Wire Format" on page 1214
Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not separated by delimiters.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

CWF properties for attribute reference and local attribute binary types:

CWF properties for attribute reference and local attribute binary types.

The Custom Wire Format properties described here apply to:

- Objects: Attribute Reference, Local Attribute
- Binary schema types: base64Binary, hexBinary

Physical representation

Property	Type	Meaning
Length	Button and Integer	If you have selected the length to be defined by *Length*, enter the number of length units for the element. The minimum value that you can specify is 1. The maximum value that you can specify is 2147483647. The default value is empty (not set).
Length Reference	Button and Enumerated type	If you have selected the length to be defined by *Length Reference*, select the name of the integer object that specifies the length of this object. Make your selection from the displayed list of integer objects that are defined as siblings of the current object, and occur before it in the structure of the message. For information about reordering elements, see "Reordering objects" on page 2900.
Inclusive Length Reference	Check box	This property is applicable only if *Length Reference* is set. If the check box is selected, the value of the sibling integer object that is identified by *Length Reference* is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by *Length Reference* is the length of the current object only. If the check box is selected, the *Length Units* property of the sibling integer object must be the same as that of the current object.

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. The length is given in bytes. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value for the *`Length Count`* or *`Length Reference`* property. The default is `Bytes`.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to the first instance only.

Property	Type	Meaning
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to all instances.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

CWF properties for attribute reference and local attribute Boolean types:

CWF properties for attribute reference and local attribute Boolean types.

The Custom Wire Format properties described here apply to:

- Objects: Attribute Reference, Local Attribute
- Boolean schema types: Boolean

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to all instances.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

CWF properties for attribute reference and local attribute dateTime types:

CWF properties for attribute reference and local attribute dateTime types.

The Custom Wire Format properties described here apply to:

- Objects: Attribute Reference, Local Attribute
- DateTime schema types: date, dateTime, gDay, gMonth, gMonthDay, gYear, gYearMonth, time

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select one from the displayed list: • `Fixed Length String`. The element's length is determined by other length properties as follows. • `Length Encoded String 1`. The first byte of the element contains the length of the string following the length byte in length units. The maximum length of a `Length Encoded String 1` element is 255 length units. • `Length Encoded String 2`. The element's first 2 bytes contain the length of the string following the two length bytes in length units. The maximum length of a `Length Encoded String 2` element is 65535 length units. The two length bytes are in the format of the WebSphere MQ queue manager Encoding. • `Null Terminated String`. The string ends with the hexadecimal NULL character, `X'00'`. • `Packed Decimal`. The dateTime is coded as a Packed Decimal number. It is valid only if the *`DateTime Format`* property represents numeric-only data. • `Binary`. The dateTime is encoded as a binary sequence of bytes. If you select this option, the range of symbols that you can specify for the Format String property is less than the range of symbols you can specify if you select a string option (see "DateTime formats" on page 6310 for details). • `Time Seconds`. This value supports C time_t and Java Date and Time objects. It is valid only if the *`DateTime Format`* property represents numeric-only data. • `Time Milliseconds`. This value supports C time_t and Java Date and Time objects. It is valid only if the *`DateTime Format`* property represents numeric-only data. The default value is `fixed length string`.
DateTime Format	String	Specify a template for date and time. The default dateTime format is dependent on the logical type of the object. For information about the defaults for the dateTime format according to the logical type, see "DateTime defaults by logical type" on page 6320. See "DateTime formats" on page 6310 for details of date and time formats.
Length	Button and Integer	If you have selected a *`Physical Type`* of `Fixed Length String`, `Packed Decimal`, or `Binary`, and have selected the length to be defined by *`Length`*, enter the number of length units for the element. The minimum value that you can specify is 1 for all three physical types. The maximum value that you can specify is `256` for `Fixed Length String`, `10` for `Packed Decimal`, and `2147483647` for `Binary`. The default value is 0 (zero).
Length Reference	Button and Enumerated type	If you have selected the length to be defined by *`Length Reference`*, select the name of the integer object that specifies the length of this object. Make your selection from the displayed list of integer objects that are defined as siblings of the current object, and occur before it in the structure of the message. For information about reordering elements, see "Reordering objects" on page 2900.

Property	Type	Meaning
Inclusive Length Reference	Check box	This property is applicable only if *Length Reference* is set. If the check box is selected, the value of the sibling integer object that is identified by *Length Reference* is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by *Length Reference* is the length of the current object only. If the check box is selected, the *Length Units* property of the sibling integer object must be the same as that of the current object.
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • Bytes. The length is given in bytes. • Characters. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • Character Units. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to Characters. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the Length property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • End of Bitstream. All data up to the end of the bitstream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value for the *Length Count* or *Length Reference* property. The default is Bytes.
Justification	Enumerated type	If you have set the *Physical Type* property to Fixed Length String, select Left Justify (the default value) or Right Justify from the list. If you have selected another value for *Physical Type*, this property is inactive.

Property	Type	Meaning
Padding Character	String	If you have set the *Physical Type* property to `Fixed Length String`, and the *Justification* property is either `Left Justify` or `Right Justify`, this property is applicable. When writing an output message, use the padding character to fill out the remaining character positions when the string length is less than the length implied by the *Length* or *Length Reference* property. Whether the string is padded from the left or the right is governed by the *Justification* property. When parsing an input message, the padding character is trimmed from the end of the string. Whether the string is trimmed from the left or the right is governed by the *Justification* property. Specify this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the displayed list. • Enter a character between quotation marks; for example, `"c"` or `'c'`, where c is any alphanumeric character. • Enter a Unicode value in the form `U+xxxx` where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form `0xYY`, where `YY` is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, it is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Numeric representation

Property	Type	Meaning
Signed	Check box	Specify whether the value is signed. This property is applicable only if the *Physical type* property is `Packed Decimal`. By default, this check box is cleared, which indicates that the value is not signed.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to all instances.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message

models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

CWF properties for attribute reference and local attribute decimal types:

CWF properties for attribute reference and local attribute decimal types.

The Custom Wire Format properties described here apply to:

- Objects: Attribute Reference, Local Attribute
- Decimal schema types: decimal, integer, negativeInteger, nonNegativeInteger, nonPositiveInteger, positiveInteger, unsignedLong

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select one from the displayed list: • `Integer`. This equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • `Packed Decimal`. Equates to the COMP-3 data type in COBOL. • `External Decimal`. Equates to the data type PIC 9 USAGE DISPLAY in COBOL. The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager: • Elements that have *`Physical Type`* set to `Integer` and `Packed Decimal` are represented in the appropriate WebSphere MQ Encoding value. • Elements that have *`Physical Type`* set to `External Decimal` are represented in the WebSphere MQ CodedCharSetId value.
Length	Integer	Enter the number of bytes to specify the element length: • If you set the *`Physical Type`* to `Integer`, select `1`, `2`, or `4` (the default) from the displayed list. • If you set the *`Physical Type`* to `Packed Decimal`, enter a value between 1 and 10. • If you set the *`Physical Type`* to `Extended Decimal`, enter a value between 1 and 256. (Numbers greater than the maximum COBOL PICTURE clause of 18 are assumed to be 18.)

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value for the *`Length Count`*. The default is `Bytes`.
Justification	Enumerated type	If you have set the *`Physical Type`* property to `External Decimal`, select `Left Justify` or `Right Justify` (the default value) from the list. If you have selected another value for *`Physical Type`*, this property is inactive.

Property	Type	Meaning
Padding Character	String	The padding character is used to fill out the remaining character positions when the string length is less than the specified string size. If you have set the *Physical Type* property to Extended Decimal, and the *Justification* property is either Left Justify or Right Justify, specify this character in one of the following ways: • Select NUL, '0', or SPACE from the displayed list. • Enter a character between quotation marks; for example, "c" or 'c', where c is any alphanumeric character. • Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form 0xYY, where YY is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Numeric representation

Property	Type	Meaning
Signed	Check box	Select (the default) or clear this property. This property is used with *Sign Orientation*.

Property	Type	Meaning
Sign EBCDIC Custom Overpunched	Check box	If the *Physical Type* is set to External Decimal and the *Signed EBCDIC Custom* property is set, this indicates that the Sign EBCDIC Custom Overpunched representation is to be used within an ASCII environment. If this check box is not selected (the default), the Sign ASCII representation is used. The setting of the *Sign EBCDIC Custom Overpunched* check box is appropriate only if the *Sign Orientation* property is set to Leading or Trailing (indicating that the element/attribute has an embedded sign representation). The check box is not available if the element/attribute is unsigned (for example, if the *Signed* check box is not set).
Sign Orientation	Enumerated type	If you have set *Physical Type* to External Decimal and you have selected *Signed*, choose from the following options that represent the COBOL options for displaying numeric data: • Leading Overpunched. This option sets a bit in the first byte if the number is negative. No setting is made if the number is positive. For example, the ASCII hexadecimal representation of the number 22 is x'3232'. Using this option, the number +22 would be x'3232' and the number -22 would be x'7232'. The default value. • Leading Separate. This option sets the first byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte. • Trailing Overpunched. This option sets a bit in the last byte if the number is negative. No setting is made if the number is positive. Using this option, the number +22 would be x'3232' and the number -22 would be x'3272'. • Trailing Separate. This option sets the last byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte. If you have set *Physical Type* to any other value, the value Not Applicable is set for you.
Virtual Decimal Point	Integer	Specify the number of places to the left (for a positive value) or right (for a negative value) that a decimal point is to be moved from its assumed position. For example, a decimal element containing 1234 with a Virtual Decimal value of 3 is 1.234, equivalent to 'V' or 'P' in a COBOL picture clause. There is no C equivalent

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • 1 Bytes. The default value. • 2 Bytes • 4 Bytes • 8 Bytes • 16 Bytes
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to the first instance only.

Property	Type	Meaning
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to all instances.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

CWF properties for attribute reference and local attribute float types:

CWF properties for attribute reference and local attribute float types.

The Custom Wire Format properties described here apply to:

- Objects: Attribute Reference, Local Attribute
- Float schema types: double, float

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select one from the displayed list: • `Integer`. This equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • `Float`. This equates to the data type FLOAT or DOUBLE in C or the COMP-1 or COMP-2 data type in COBOL and is the default value. • `Packed Decimal`. Equates to the COMP-3 data type in COBOL. • `External Decimal`. Equates to the data type PIC 9 USAGE DISPLAY in COBOL. The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager: • Elements that have *`Physical Type`* set to `Integer`, `Packed Decimal`, and `Float` are represented in the appropriate WebSphere MQ Encoding value. • Elements that have *`Physical Type`* set to `External Decimal` are represented in the WebSphere MQ CodedCharSetId value.
Length	Integer	Enter the number of bytes to specify the element length: • If you set the *`Physical Type`* to `Float`, select a value from the displayed list. The default value is `8`. • If you set the *`Physical Type`* to `Integer`, select `1`, `2`, or `4` (the default) from the displayed list. • If you set the *`Physical Type`* to `Packed Decimal`, enter a value between 1 and 10. • If you set the *`Physical Type`* to `Extended Decimal`, enter a value between 1 and 256. (Numbers greater than the maximum COBOL PICTURE clause of 18 are assumed to be 18.)

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value for the *`Length Count`*. The default is `Bytes`.
Justification	Enumerated type	If you have set the *`Physical Type`* property to `External Decimal`, select `Left Justify` or `Right Justify` (the default value) from the list. If you have selected another value for *`Physical Type`*, this property is inactive.

Property	Type	Meaning
Padding Character	String	The padding character is used to fill out the remaining character positions when the string length is less than the specified string size. If you have set the *Physical Type* property to Extended Decimal, and the *Justification* property is either Left Justify or Right Justify, specify this character in one of the following ways: • Select NUL, '0', or SPACE from the displayed list. • Enter a character between quotation marks; for example, "c" or 'c', where c is any alphanumeric character. • Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form 0xYY, where YY is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Numeric representation

Property	Type	Meaning
Signed	Check box	Select or clear (unsigned, the default) this property. If you have set *Physical Type* to Float, this is selected. This property is used with *Sign Orientation*.

Property	Type	Meaning
Sign EBCDIC Custom Overpunched	Check box	If the *Physical Type* is set to External Decimal and the *Signed EBCDIC Custom* property is set, this indicates that the Sign EBCDIC Custom Overpunched representation is to be used within an ASCII environment. If this check box is not selected (the default), the Sign ASCII representation is used. The setting of the *Sign EBCDIC Custom Overpunched* check box is appropriate only if the *Sign Orientation* property is set to Leading or Trailing (indicating that the element/attribute has an embedded sign representation). The check box is not available if the element/attribute is unsigned (for example, if the *Signed* check box is not set).
Sign Orientation	Enumerated type	If you have set *Physical Type* to External Decimal and you have selected *Signed*, choose from the following options that represent the COBOL options for displaying numeric data: • Leading Overpunched. This option sets a bit in the first byte if the number is negative. No setting is made if the number is positive. For example, the ASCII hexadecimal representation of the number 22 is x'3232'. Using this option, the number +22 would be x'3232' and the number -22 would be x'7232'. The default value. • Leading Separate. This option sets the first byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte. • Trailing Overpunched. This option sets a bit in the last byte if the number is negative. No setting is made if the number is positive. Using this option, the number +22 would be x'3232' and the number -22 would be x'3272'. • Trailing Separate. This option sets the last byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte. If you have set *Physical Type* to any other value, the value Not Applicable is set for you.
Virtual Decimal Point	Integer	Specify the number of places to the left (for a positive value) or right (for a negative value) that a decimal point is to be moved from its assumed position. For example, a float element containing 1234 with a Virtual Decimal value of 3 is 1.234. This property is not applicable if you have set *Physical Type* to Float.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • 1 Bytes. The default value. • 2 Bytes • 4 Bytes • 8 Bytes • 16 Bytes
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to the first instance only.

Property	Type	Meaning
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to all instances.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

CWF properties for attribute reference and local attribute integer types:

CWF properties for attribute reference and local attribute integer types.

The Custom Wire Format properties described here apply to:

- Objects: Attribute Reference, Local Attribute
- Integer schema types: byte, int, long, short, unsignedByte, unsignedInt, unsignedShort

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select one from the displayed list: • `Integer`. This equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • `Packed Decimal`. Equates to the COMP-3 data type in COBOL. • `External Decimal`. Equates to the data type PIC 9 USAGE DISPLAY in COBOL. The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager: • Elements that have *`Physical Type`* set to `Integer` and `Packed Decimal` are represented in the appropriate WebSphere MQ Encoding value. • Elements that have *`Physical Type`* set to `External Decimal` are represented in the WebSphere MQ CodedCharSetId value.
Length	Integer	Enter the number of bytes to specify the element length: • If you set the *`Physical Type`* to `Integer`, select `1`, `2`, or `4` (the default) from the displayed list. • If you have set *`Physical Type`* to `Packed Decimal`, enter a value between 1 and 6. • If you have set *`Physical Type`* to `Extended Decimal`, enter a value between 1 and 11.

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value for the *`Length Count`*. The default is `Bytes`.
Justification	Enumerated type	If you have set the *`Physical Type`* property to `External Decimal`, select `Left Justify` or `Right Justify` (the default value) from the list. If you have selected another value for *`Physical Type`*, this property is inactive.

Property	Type	Meaning
Padding Character	String	The padding character is used to fill out the remaining character positions when the string length is less than the specified string size. If you have set the *Physical Type* property to Extended Decimal, and the *Justification* property is either Left Justify or Right Justify, specify this character in one of the following ways: • Select NUL, '0', or SPACE from the displayed list. • Enter a character between quotation marks; for example, "c" or 'c', where c is any alphanumeric character. • Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form 0xYY, where YY is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Numeric representation

Property	Type	Meaning
Signed	Check box	Select (the default) or clear this property. This property is used with *Sign Orientation*.

Property	Type	Meaning
Sign EBCDIC Custom Overpunched	Check box	If the *Physical Type* is set to External Decimal and the *Signed EBCDIC Custom* property is set, this indicates that the Sign EBCDIC Custom Overpunched representation is to be used within an ASCII environment. If this check box is not selected (the default), the Sign ASCII representation is used. The setting of the *Sign EBCDIC Custom Overpunched* check box is appropriate only if the *Sign Orientation* property is set to Leading or Trailing (indicating that the element/attribute has an embedded sign representation). The check box is not available if the element/attribute is unsigned (for example, if the *Signed* check box is not set).
Sign Orientation	Enumerated type	If you have set *Physical Type* to External Decimal and you have selected *Signed*, choose from the following options that represent the COBOL options for displaying numeric data: • Leading Overpunched. This option sets a bit in the first byte if the number is negative. No setting is made if the number is positive. For example, the ASCII hexadecimal representation of the number 22 is x'3232'. Using this option, the number +22 would be x'3232' and the number -22 would be x'7232'. The default value. • Leading Separate. This option sets the first byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte. • Trailing Overpunched. This option sets a bit in the last byte if the number is negative. No setting is made if the number is positive. Using this option, the number +22 would be x'3232' and the number -22 would be x'3272'. • Trailing Separate. This option sets the last byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte. If you have set *Physical Type* to any other value, the value Not Applicable is set for you.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • 1 Bytes. The default value. • 2 Bytes • 4 Bytes • 8 Bytes • 16 Bytes
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to the first instance only.

Property	Type	Meaning
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to all instances.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

CWF properties for attribute reference and local attribute string types:

CWF properties for attribute reference and local attribute string types.

The Custom Wire Format properties described here apply to:

- Objects: Attribute Reference, Local Attribute
- String schema types: anyURI, ENTITIES, ENTITY, ID, IDREF, IDREFS, language, Name, NCName, NMTOKEN, NMTOKENS, normalizedString, NOTATION, QName, string, token

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select one from the displayed list: • `Fixed Length String`. The element's length is determined by other length properties as follows. • `Length Encoded String 1`. The first byte of the element contains the length of the string following the length byte in length units. The maximum length of a `Length Encoded String 1` element is 255 length units. • `Length Encoded String 2`. The element's first 2 bytes contain the length of the string following the two length bytes in length units. The maximum length of a `Length Encoded String 2` element is 65535 length units. The two length bytes are in the format of the WebSphere MQ queue manager Encoding. • `Null Terminated String`. The string ends with the hexadecimal NULL character, `X'00'`. The default is `Fixed Length String`.
Length	Button and Integer	If you have selected a *`Physical Type`* of `Fixed Length String` or `Binary`, and have selected the length to be defined by *`Length`*, enter the number of length units for the element. The minimum value that you can specify is `0` (zero), the maximum value that you can specify is `2147483647` The default value is `0` (zero).
Length Reference	Button and Enumerated type	If you have selected the length to be defined by *`Length Reference`*, select the name of the integer object that specifies the length of this object. Make your selection from the displayed list of integer objects that are defined as siblings of the current object, and occur before it in the structure of the message. For information about reordering elements, see “Reordering objects” on page 2900.
Inclusive Length Reference	Check box	This property is applicable only if *`Length Reference`* is set. If the check box is selected, the value of the sibling integer object that is identified by *`Length Reference`* is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by *`Length Reference`* is the length of the current object only. If the check box is selected, the *`Length Units`* property of the sibling integer object must be the same as that of the current object.

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. The length is given in bytes. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value for the *`Length Count`* or *`Length Reference`* property. The default is `Bytes`.
Justification	Enumerated type	If you have set the *`Physical Type`* property to `Fixed Length String`, select `Left Justify` (the default value) or `Right Justify` from the list. If you have selected another value for *`Physical Type`*, this property is inactive.

Property	Type	Meaning
Padding Character	String	If you have set the *Physical Type* property to `Fixed Length String`, and the *Justification* property is either `Left Justify` or `Right Justify`, this property is applicable. When writing an output message, use the padding character to fill out the remaining character positions when the string length is less than the length implied by the *Length* or *Length Reference* property. Whether the string is padded from the left or the right is governed by the *Justification* property. When parsing an input message, the padding character is trimmed from the end of the string. Whether the string is trimmed from the left or the right is governed by the *Justification* property. Specify this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the displayed list. • Enter a character between quotation marks; for example, `"c"` or `'c'`, where c is any alphanumeric character. • Enter a Unicode value in the form `U+xxxx` where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form `0xYY`, where `YY` is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, it is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to all instances.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

Local attribute XML properties:

The properties, and their permissible values, vary according to the object type.

The properties that are displayed on the object page, and the values that those properties can take, can vary according to the type of the object. For example, the properties for type *string* are different from those of type *Boolean*. Select the link for the object type from the following table.

Type of object	**Properties**
Binary types	• base64Binary • hexBinary
Boolean types	• Boolean
DateTime types	• date • dateTime • gDay • gMonth • gMonthDay • gYear • gYearMonth • time
Decimal types	• decimal • integer • negativeInteger • nonNegativeInteger • nonPositiveInteger • positiveInteger • unsignedLong
Float types	• double • float

Type of object	Properties
Integer types	• byte • int • long • short • unsignedByte • unsignedInt • unsignedShort
Interval types	• duration
String types	• anyURI • ENTITIES • ENTITY • ID • IDREF • IDREFS • language • Name • NCName • NMTOKEN • NMTOKENS • normalizedString • NOTATION • QName • string • token

Note: The physical format properties for simple type *duration* are the same as the physical properties of the string logical types.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

XML properties for attribute reference, element reference, local attribute, local element binary types:

XML wire format properties for attribute reference, element reference, local attribute, and local element binary types.

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- Binary schema types: base64Binary, hexBinary

Field identification

A number of the following properties only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated type	Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list: • `XMLElement`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their *`XML Name`*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object , you must set their *`XML Name`* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in *`ID Attribute Name`* and a value as specified in *`ID Attribute Value`*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *`XML Name`*, *`ID Attribute Name`*, *`ID Attribute Value`* to the same values: – You must also set *`Value Attribute Name`* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in *`Value Attribute Name`*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *`ID Attribute Name`*. The value is the value of *`ID Attribute Value`*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *`XML Name`*, *`ID Attribute Name`*, *`ID Attribute Value`* to the same values: – You must also set *`Value Attribute Name`* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 6262 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
XML Name	String	Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message). This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name. You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name. If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).
ID Attribute Name	String	Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
ID Attribute Value	String	Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

Physical representation

Property	Type	Meaning
Encoding	String	Select one of the following values from the drop-down list: : • `CDatahex` (the default). Hexadecimal values in this field are specified with the CDATA qualifier, for example <e1><![CDATA[62]]></e1> • `hex`. Hexadecimal values in this field are specified as digits only, for example <e1>62</e1>. • `base64`. Values in this field are specified as digits only, coded in base 64.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

XML properties for attribute reference, element reference, local attribute, local element Boolean types:

XML wire format properties for attribute reference, element reference, local attribute and local element Boolean types.

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- Boolean schema types: Boolean

Field identification

A number of the following properties only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated type	Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list: • `XMLElement`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their *XML Name*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object , you must set their *XML Name* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in *ID Attribute Name* and a value as specified in *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in *Value Attribute Name*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *ID Attribute Name*. The value is the value of *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 6262 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
XML Name	String	Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message). This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name. You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name. If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).
ID Attribute Name	String	Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
ID Attribute Value	String	Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

XML properties for attribute reference, element reference, local attribute, local element dateTime types:

XML wire format properties for attribute reference, element reference, local attribute, and local element dateTime types.

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- DateTime schema types: date, dateTime, gDay, gMonth, gMonthDay, gYear, gYearMonth, time

Field identification

A number of the following properties only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated type	Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list: • `XMLElement`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their *`XML Name`*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object , you must set their *`XML Name`* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in *`ID Attribute Name`* and a value as specified in *`ID Attribute Value`*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *`XML Name`*, *`ID Attribute Name`*, *`ID Attribute Value`* to the same values: – You must also set *`Value Attribute Name`* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in *`Value Attribute Name`*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *`ID Attribute Name`*. The value is the value of *`ID Attribute Value`*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *`XML Name`*, *`ID Attribute Name`*, *`ID Attribute Value`* to the same values: – You must also set *`Value Attribute Name`* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 6262 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
XML Name	String	Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message). This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name. You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name. If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).
ID Attribute Name	String	Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
ID Attribute Value	String	Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

Physical representation

Property	Type	Meaning
DateTime Format	String	Specify a format string that specifies the rendering of the value for dateTime elements. The default dateTime format is dependent on the logical type of the object. For information about the defaults for the dateTime format according to the logical type see "DateTime defaults by logical type" on page 6320. See "DateTime formats" on page 6310 for details of dateTime formats.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical

model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

XML properties for attribute reference, element reference, local attribute, local element decimal types:

XML wire format properties for attribute reference, element reference, local attribute, and local element decimal types.

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- Decimal schema types: decimal, integer, negativeInteger, nonNegativeInteger, nonPositiveInteger, positiveInteger, unsignedLong

Field identification

A number of the following properties only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated type	Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list: • `XMLElement`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their *XML Name*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object , you must set their *XML Name* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in *ID Attribute Name* and a value as specified in *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in *Value Attribute Name*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *ID Attribute Name*. The value is the value of *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 6262 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
XML Name	String	Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message). This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name. You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name. If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).
ID Attribute Name	String	Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
ID Attribute Value	String	Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

XML properties for attribute reference, element reference, local attribute, local element float types:

XML wire format properties for attribute reference, element reference, local attribute, and local element float types.

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- Float schema types: double, float

Field identification

A number of the following properties only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated type	Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list: • `XMLElement`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their *`XML Name`*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object , you must set their *`XML Name`* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in *`ID Attribute Name`* and a value as specified in *`ID Attribute Value`*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *`XML Name`*, *`ID Attribute Name`*, *`ID Attribute Value`* to the same values: – You must also set *`Value Attribute Name`* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in *`Value Attribute Name`*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *`ID Attribute Name`*. The value is the value of *`ID Attribute Value`*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *`XML Name`*, *`ID Attribute Name`*, *`ID Attribute Value`* to the same values: – You must also set *`Value Attribute Name`* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 6262 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
XML Name	String	Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message). This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name. You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name. If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).
ID Attribute Name	String	Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
ID Attribute Value	String	Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

XML properties for attribute reference, element reference, local attribute, local element integer types:

XML wire format properties for attribute reference, element reference, local attribute, and local element integer types.

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- Integer schema types: byte, int, long, short, unsignedByte, unsignedInt, unsignedShort

Field identification

A number of the following properties only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated type	Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list: • `XMLElement`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their *XML Name*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object , you must set their *XML Name* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in *ID Attribute Name* and a value as specified in *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in *Value Attribute Name*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *ID Attribute Name*. The value is the value of *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 6262 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
XML Name	String	Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message). This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name. You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name. If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).
ID Attribute Name	String	Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
ID Attribute Value	String	Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

XML properties for attribute reference, element reference, local attribute, local element string types:

XML wire format properties for attribute reference, element reference, local attribute, and local element string types.

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- String schema types: anyURI, ENTITIES, ENTITY, ID, IDREF, IDREFS, language, Name, NCName, NMTOKEN, NMTOKENS, normalizedString, NOTATION, QName, string, token

Field identification

A number of the following properties only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated type	Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list: • `XMLElement`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their *`XML Name`*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object , you must set their *`XML Name`* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in *`ID Attribute Name`* and a value as specified in *`ID Attribute Value`*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *`XML Name`*, *`ID Attribute Name`*, *`ID Attribute Value`* to the same values: – You must also set *`Value Attribute Name`* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in *`Value Attribute Name`*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *`ID Attribute Name`*. The value is the value of *`ID Attribute Value`*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *`XML Name`*, *`ID Attribute Name`*, *`ID Attribute Value`* to the same values: – You must also set *`Value Attribute Name`* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 6262 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
XML Name	String	Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message). This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name. You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name. If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).
ID Attribute Name	String	Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
ID Attribute Value	String	Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

Local attribute TDS properties:

The properties, and their permissible values, vary according to the object type.

The properties that are displayed on the object page, and the values that those properties can take, can vary according to the type of the object. For example, the properties for type *string* are different from those of type *Boolean*. Select the link for the object type from the following table.

Binary types	Boolean types	DateTime types	Decimal types
• base64Binary • hexBinary	• Boolean	• date • dateTime • gDay • gMonth • gMonthDay • gYear • gYearMonth • time	• decimal • integer • negativeInteger • nonNegativeInteger • nonPositiveInteger • positiveInteger • unsignedLong

Float types	Integer types	Interval types	String types
• double • float	• byte • int • long • short • unsignedByte • unsignedInt • unsignedShort	• duration	• anyURI • ENTITIES • ENTITY • ID • IDREF • IDREFS • language • Name • NCName • NMTOKEN • NMTOKENS • normalizedString • NOTATION • QName • string • token

Note: The physical format properties for simple type *duration* are the same as the physical properties of the string logical types.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM TDS format" on page 1221
The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

TDS properties for local attribute binary types:

The TDS format properties for local attribute binary types.

The TDS Format properties described here apply to:

- Objects: Local Attribute
- Binary schema types: base64Binary, hexBinary

Field Identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the `Data Element Separation` property of the complex type or types in which the object is a child is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, this property must contain a non-empty value. If the object is a complex element, and the `Data Element Separation` property of its parent is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, the property can contain an empty value. The value for this property must be unique for every element in the message set; that is, no two elements in the message set can contain the same value for this property.
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the `Data Element Separation` method is set to `Use Data Pattern` in the complex type. For more details, see "Regular expression syntax" on page 6304.

Physical representation

Property	Type	Meaning
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no `Length Reference` is specified. The default is dependent on the setting of the message set property `Derive default length from logical type`. If `Derive default length from logical type` is selected, the default value is derived from any length or maxLength value constraint (schema facet) on the object's simple type.
Length Reference	Enumerated type	This property is applicable only if `Physical Type` is `Text`, `Binary`, or `TLOG Specific`. If set, this property takes precedence over any value in the `Length Units` property. Specify the identifier of a sibling integer object, the value of which determines the length of the object in question. The sibling object must be defined before the current object within the message structure. For information about reordering elements, see "Reordering objects" on page 2900.

Property	Type	Meaning
Inclusive Length Reference	Check box	This property is applicable only if `Length Reference` is set. If the check box is selected, the value of the sibling integer object that is identified by `Length Reference` is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by `Length Reference` is the length of the current object only. If the check box is selected, the `Length Units` property of the sibling integer object must be the same as that of the current object.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

TDS properties for local attribute Boolean types:

The TDS format properties for local attribute Boolean types.

The TDS Format properties described here apply to:

- Objects: Local Attribute
- Boolean schema types: Boolean

Field Identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the `Data Element Separation` property of the complex type or types in which the object is a child is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, this property must contain a non-empty value. If the object is a complex element, and the `Data Element Separation` property of its parent is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, the property can contain an empty value. The value for this property must be unique for every element in the message set; that is, no two elements in the message set can contain the same value for this property.
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the `Data Element Separation` method is set to `Use Data Pattern` in the complex type. For more details, see "Regular expression syntax" on page 6304.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the `Messaging Standard` property of the message set is `User Defined Text`, `User Defined Mixed`, `CSV`, or `TLOG`, select one of the following values: • `Text`. The data is in character format. • `Binary`. The data is in bit string format. The default is dependent on the `Messaging Standard` property. For all other `Messaging Standard` values, the `Physical Type` property is set to `Text`.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

TDS properties for local attribute dateTime types:

The TDS format properties for local attribute dateTime types.

The TDS Format properties described here apply to:

- Objects: Local Attribute
- DateTime schema types: date, dateTime, gDay, gMonth, gMonthDay, gYear, gYearMonth, time

Field Identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the `Data Element Separation` property of the complex type or types in which the object is a child is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, this property must contain a non-empty value. If the object is a complex element, and the `Data Element Separation` property of its parent is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, the property can contain an empty value. The value for this property must be unique for every element in the message set; that is, no two elements in the message set can contain the same value for this property.

Property	Type	Meaning
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the `Data Element Separation` method is set to `Use Data Pattern` in the complex type. For more details, see “Regular expression syntax” on page 6304.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the `Messaging Standard` property of the message set is `User Defined Text`, `User Defined Mixed`, `CSV`, or `TLOG`, select one of the following values: • `Text`. The data is in character format. • `Length Encoded String 1`. The first byte of the data contains the length (in length units) of the data string that follows the length byte. The maximum length of a `Length Encoded String 1` element is 255 length units. • `Length Encoded String 2`. The first two bytes of the data contain the length (in length units) of the data string that follows the two length bytes. The maximum length of a `Length Encoded String 2` element is 65535 length units. • `Null Terminated String`. The data string ends with the hexadecimal NULL character, `X'00'`. • `Packed Decimal`. The data is a packed decimal number that equates to the COMP-3 data type in COBOL. • `Binary`. The data is in bit string format. • `Time Seconds`. This value supports C time_t, and Java Date and Time objects. It is valid only if the `DateTime Format` property represents numeric-only data. • `Time Milliseconds`. This value supports C time_t, and Java Date and Time objects. It is valid only if the `DateTime Format` property represents numeric-only data. The default is dependent on the `Messaging Standard` property. For all other `Messaging Standard` values, the `Physical Type` property is set to `Text`.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no `Length Reference` is specified. If this property is not set and the message set property `Derive default length from logical type` is selected, and the `Physical type` is 'Character', the default value is derived from any length or maxLength value constraint (schema facet) on the simple type.

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the object. Select one of the following options (some physical types do not offer both options): • `Bytes`. The length is given in bytes. • `Characters`. The length is given in characters. The number of bytes that are processed in the bit stream depends on the code page of the message. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the contents of the bit stream. The parser reads one character at a time and determines whether the character comprises one or more bytes. The default is dependent on the physical type of the object.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • `Not Applicable` • `Left Justify` • `Right Justify`

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the drop-down list. • Enter a character between quotation marks, for example "c" or 'c', where c is any alphanumeric character. • Enter a hexadecimal character code in the form `0xYY`, where `YY` is a hexadecimal value. • Enter a Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.
DateTime Format	String	Specify a template for date and time. The default DateTime format is dependent on the logical type of the object. For information about the defaults for the dateTime format according to the logical type, see “DateTime defaults by logical type” on page 6320. See “DateTime formats” on page 6310 for details of date and time formats.
Length Reference	Enumerated type	This property is applicable only if `Physical Type` is `Text`, `Binary`, or `TLOG Specific`. If set, this property takes precedence over any value in the `Length Units` property. Specify the identifier of a sibling integer object, the value of which determines the length of the object in question. The sibling object must be defined before the current object within the message structure. For information about reordering elements, see “Reordering objects” on page 2900.

Property	Type	Meaning
Inclusive Length Reference	Check box	This property is applicable only if `Length Reference` is set. If the check box is selected, the value of the sibling integer object that is identified by `Length Reference` is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by `Length Reference` is the length of the current object only. If the check box is selected, the `Length Units` property of the sibling integer object must be the same as that of the current object.

Numeric representation

Property	Type	Meaning
Signed	Check box	Specify whether the value is signed. This property is applicable only if the `Physical type` property is `Packed Decimal`. By default, this check box is cleared, which indicates that the value is not signed.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

TDS properties for local attribute decimal types:

The TDS format properties for local attribute decimal types.

The TDS Format properties described here apply to:

- Objects: Local Attribute
- Decimal schema types: decimal, integer, negativeInteger, nonNegativeInteger, nonPositiveInteger, positiveInteger, unsignedLong

Field Identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the `Data Element Separation` property of the complex type or types in which the object is a child is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, this property must contain a non-empty value. If the object is a complex element, and the `Data Element Separation` property of its parent is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, the property can contain an empty value. The value for this property must be unique for every element in the message set; that is, no two elements in the message set can contain the same value for this property.
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the `Data Element Separation` method is set to `Use Data Pattern` in the complex type. For more details, see "Regular expression syntax" on page 6304.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the `Messaging Standard` property of the message set is `User Defined Text`, `User Defined Mixed`, `CSV`, or `TLOG`, select one of the following values: • `Text`. The data is in character format. • `Packed Decimal`. The data is a packed decimal number that equates to the COMP-3 data type in COBOL. • `TLOG Specific` - this option can be selected only if the `Message Standard` property of the message set is `TLOG`. This option indicates that the format of the data is specific to the TLOG messaging standard. • `Integer`. Equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • `External Decimal`. The data is a decimal number that equates to the data type PIC 9 USAGE DISPLAY in COBOL. The default is dependent on the `Messaging Standard` property. For all other `Messaging Standard` values, the `Physical Type` property is set to `Text`.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no `Length Reference` is specified. If this property is not set and the message set property `Derive default length from logical type` is selected, and the `Physical type` is 'Character', the default value is derived from any length or maxLength value constraint (schema facet) on the simple type.
Length Units	Enumerated type	Select the unit of length for the object. Select one of the following options (some physical types do not offer both options): • `Bytes`. The length is given in bytes. • `Characters`. The length is given in characters. The number of bytes that are processed in the bit stream depends on the code page of the message. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the contents of the bit stream. The parser reads one character at a time and determines whether the character comprises one or more bytes. The default is dependent on the physical type of the object.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • `Not Applicable` • `Left Justify` • `Right Justify`

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the drop-down list. • Enter a character between quotation marks, for example "c" or 'c', where c is any alphanumeric character. • Enter a hexadecimal character code in the form `0xYY`, where `YY` is a hexadecimal value. • Enter a Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.
Length Reference	Enumerated type	This property is applicable only if `Physical Type` is `Text`, `Binary`, or `TLOG Specific`. If set, this property takes precedence over any value in the `Length Units` property. Specify the identifier of a sibling integer object, the value of which determines the length of the object in question. The sibling object must be defined before the current object within the message structure. For information about reordering elements, see “Reordering objects” on page 2900.

Property	Type	Meaning
Inclusive Length Reference	Check box	This property is applicable only if `Length Reference` is set. If the check box is selected, the value of the sibling integer object that is identified by `Length Reference` is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by `Length Reference` is the length of the current object only. If the check box is selected, the `Length Units` property of the sibling integer object must be the same as that of the current object.

Numeric representation

Property	Type	Meaning
Sign Orientation	Enumerated type	The values that you can choose for this property are: • `Leading Separate` • `Trailing Separate` • `Leading Overpunched` • `Trailing Overpunched` This property is enabled only if you have set `Physical Type` to `Text` or `External Decimal`, and you have selected `Signed`. If `Physical Type` is `Text`, the only valid values of `Sign Orientation` are `Leading Separate` and `Trailing Separate`. If `Physical Type` is `External Decimal` and `Sign EBCDIC Customer Overpunched` is selected, the only valid values of `Sign Orientation` are `Leading Overpunched` and `Trailing Overpunched`.
Positive Sign	String	Specify the value that represents the positive symbol. Do not specify a numeric value. If no value is set, "+" is assumed. The positive sign is not written when creating an output message; it is used only to recognize the positive sign when parsing a message bit stream. This property is applicable only if `Physical Type` is `Text` and `Signed` is selected.
Negative Sign	String	Specify the value that represents the negative symbol. Do not specify a numeric value. If no value is set, "-" is assumed. This property is applicable only if `Physical Type` is `Text` and `Signed` is selected.
Virtual Decimal Point	Button and Integer	Specify a non-zero integer that represents the position of an implied decimal point within a number, or specify 0, the default, to use the formatting of Float or Decimal numbers as specified by the `Precision` property. If you specify a positive integer, the position of the decimal point is moved left from the right side of the number. For example, if you specify 3, the decimal value `1234` represents 1.234 If you specify a negative integer, the position of the decimal point is moved right from the right side of the number. For example, if you specify `-3`, the decimal value `1234` represents 1,234,000.

Property	Type	Meaning
Precision	Button and Integer	This property is applicable only if `Physical Type` is `Text`. It is used if the value of the `Virtual Decimal Point` property is 0, which indicates that the decimal point is present in the data. It deals with truncation, and specifies how many digits are to follow the decimal point. Either specify a number of digits: • If you set `Precision` to 0, data is truncated so that the fractional part is lost. For example, the value `123.45` is truncated to `123`. • If you set `Precision` to a number less than the number of fractional digits, data is truncated. For example, the value `123.4567` is truncated to `123.45` if you set `Precision` to 2. • If you set `Precision` to a number greater than the number of fractional digits, the value is padded with extra zeros. For example, the value `12.345` is padded to `12.34500` if you set `Precision` to 5. Or select one option from the list: • `All Significant Digits - decimal separator only required if fractional digits` (the default): all significant digits are written to the output bit stream, and no decimal separator is written if no fractional digits are present. • `Explicit Decimal Separator - decimal separator always required`: all significant digits are written to the output bit stream and the decimal separator is always included, even when no fractional digits are present. The decimal separator must be present in the input bit stream, even when no fractional digits are present.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

TDS properties for local attribute float types:

The TDS format properties for local attribute float types.

The TDS Format properties described here apply to:

- Objects: Local Attribute
- Float schema types: double, float

Field Identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the `Data Element Separation` property of the complex type or types in which the object is a child is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, this property must contain a non-empty value. If the object is a complex element, and the `Data Element Separation` property of its parent is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, the property can contain an empty value. The value for this property must be unique for every element in the message set; that is, no two elements in the message set can contain the same value for this property.
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the `Data Element Separation` method is set to `Use Data Pattern` in the complex type. For more details, see "Regular expression syntax" on page 6304.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the `Messaging Standard` property of the message set is `User Defined Text`, `User Defined Mixed`, `CSV`, or `TLOG`, select one of the following values: • `Text`. The data is in character format. • `Float`. Equates to the data type FLOAT or DOUBLE in C, or the COMP-1 or COMP-2 numeric data type in COBOL. • `Integer`. Equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • `Packed Decimal`. The data is a packed decimal number that equates to the COMP-3 data type in COBOL. • `External Decimal`. The data is a decimal number that equates to the data type PIC 9 USAGE DISPLAY in COBOL. The default is dependent on the `Messaging Standard` property. For all other `Messaging Standard` values, the `Physical Type` property is set to `Text`.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no `Length Reference` is specified. If this property is not set and the message set property `Derive default length from logical type` is selected, and the `Physical type` is 'Character', the default value is derived from any length or maxLength value constraint (schema facet) on the simple type.
Length Units	Enumerated type	Select the unit of length for the object. Select one of the following options (some physical types do not offer both options): • `Bytes`. The length is given in bytes. • `Characters`. The length is given in characters. The number of bytes that are processed in the bit stream depends on the code page of the message. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the contents of the bit stream. The parser reads one character at a time and determines whether the character comprises one or more bytes. The default is dependent on the physical type of the object.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • `Not Applicable` • `Left Justify` • `Right Justify`

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the drop-down list. • Enter a character between quotation marks, for example "c" or 'c', where c is any alphanumeric character. • Enter a hexadecimal character code in the form `0xYY`, where `YY` is a hexadecimal value. • Enter a Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.
Length Reference	Enumerated type	This property is applicable only if `Physical Type` is `Text`, `Binary`, or `TLOG Specific`. If set, this property takes precedence over any value in the `Length Units` property. Specify the identifier of a sibling integer object, the value of which determines the length of the object in question. The sibling object must be defined before the current object within the message structure. For information about reordering elements, see “Reordering objects” on page 2900.

Property	Type	Meaning
Inclusive Length Reference	Check box	This property is applicable only if `Length Reference` is set. If the check box is selected, the value of the sibling integer object that is identified by `Length Reference` is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by `Length Reference` is the length of the current object only. If the check box is selected, the `Length Units` property of the sibling integer object must be the same as that of the current object.

Numeric representation

Property	Type	Meaning
Sign Orientation	Enumerated type	The values that you can choose for this property are: • `Leading Separate` • `Trailing Separate` • `Leading Overpunched` • `Trailing Overpunched` This property is enabled only if you have set `Physical Type` to `Text` or `External Decimal`, and you have selected `Signed`. If `Physical Type` is `Text`, the only valid values of `Sign Orientation` are `Leading Separate` and `Trailing Separate`. If `Physical Type` is `External Decimal` and `Sign EBCDIC Customer Overpunched` is selected, the only valid values of `Sign Orientation` are `Leading Overpunched` and `Trailing Overpunched`.
Positive Sign	String	Specify the value that represents the positive symbol. Do not specify a numeric value. If no value is set, "+" is assumed. The positive sign is not written when creating an output message; it is used only to recognize the positive sign when parsing a message bit stream. This property is applicable only if `Physical Type` is `Text` and `Signed` is selected.
Negative Sign	String	Specify the value that represents the negative symbol. Do not specify a numeric value. If no value is set, "-" is assumed. This property is applicable only if `Physical Type` is `Text` and `Signed` is selected.
Virtual Decimal Point	Button and Integer	Specify a non-zero integer that represents the position of an implied decimal point within a number, or specify 0, the default, to use the formatting of Float or Decimal numbers as specified by the `Precision` property. If you specify a positive integer, the position of the decimal point is moved left from the right side of the number. For example, if you specify `3`, the decimal value `1234` represents 1.234 If you specify a negative integer, the position of the decimal point is moved right from the right side of the number. For example, if you specify `-3`, the decimal value `1234` represents 1,234,000.

Property	Type	Meaning
Precision	Button and Integer	This property is applicable only if `Physical Type` is `Text`. It is used if the value of the `Virtual Decimal Point` property is 0, which indicates that the decimal point is present in the data. It deals with truncation, and specifies how many digits are to follow the decimal point. Either specify a number of digits: • If you set `Precision` to 0, data is truncated so that the fractional part is lost. For example, the value `123.45` is truncated to `123`. • If you set `Precision` to a number less than the number of fractional digits, data is truncated. For example, the value `123.4567` is truncated to `123.45` if you set `Precision` to 2. • If you set `Precision` to a number greater than the number of fractional digits, the value is padded with extra zeros. For example, the value `12.345` is padded to `12.34500` if you set `Precision` to 5. Or select one option from the list: • `All Significant Digits - decimal separator only required if fractional digits` (the default): all significant digits are written to the output bit stream, and no decimal separator is written if no fractional digits are present. • `Explicit Decimal Separator - decimal separator always required`: all significant digits are written to the output bit stream and the decimal separator is always included, even when no fractional digits are present. The decimal separator must be present in the input bit stream, even when no fractional digits are present. • `Exponential Notation - Example "1.23456e002"`: data is written out to the bit stream as a signed value having the format `[sign1]a.bbbe[sign2]ccc` where: – `[sign1]` is the value of `Negative Sign` if the value is negative – `a` is a single decimal digit – `bbb` is one or more decimal digits – `[sign2]` is the value of `Negative Sign` if the exponent is negative – `ccc` is exactly three decimal digits (the exponent) `[sign1]` and `[sign2]` are absent if the value and exponent are positive. For example, the value -123.456 is represented as `-1.23456e002` and the value 0.00012 is represented as `1.2e-004` in the output bit stream, assuming that the value of `Negative Sign` is "-", and the value of `Sign Orientation` is `Leading`. The value -0.00012 is represented as `1.2*e*004` if `Negative Sign` is "*" and `Sign Orientation` is `Trailing`.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

TDS properties for local attribute integer types:

The TDS format properties for local attribute integer types.

The TDS Format properties described here apply to:

- Objects: Local Attribute
- Integer schema types: byte, int, long, short, unsignedByte, unsignedInt, unsignedShort

Field Identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the `Data Element Separation` property of the complex type or types in which the object is a child is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, this property must contain a non-empty value. If the object is a complex element, and the `Data Element Separation` property of its parent is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, the property can contain an empty value. The value for this property must be unique for every element in the message set; that is, no two elements in the message set can contain the same value for this property.
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the `Data Element Separation` method is set to `Use Data Pattern` in the complex type. For more details, see "Regular expression syntax" on page 6304.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the `Messaging Standard` property of the message set is `User Defined Text`, `User Defined Mixed`, `CSV`, or `TLOG`, select one of the following values: • `Text`. The data is in character format. • `Integer`. Equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • `Packed Decimal`. The data is a packed decimal number that equates to the COMP-3 data type in COBOL. • `External Decimal`. The data is a decimal number that equates to the data type PIC 9 USAGE DISPLAY in COBOL. The default is dependent on the `Messaging Standard` property. For all other `Messaging Standard` values, the `Physical Type` property is set to `Text`.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no `Length Reference` is specified. If this property is not set and the message set property `Derive default length from logical type` is selected, and the `Physical type` is 'Character', the default value is derived from any length or maxLength value constraint (schema facet) on the simple type.
Length Units	Enumerated type	Select the unit of length for the object. Select one of the following options (some physical types do not offer both options): • `Bytes`. The length is given in bytes. • `Characters`. The length is given in characters. The number of bytes that are processed in the bit stream depends on the code page of the message. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the contents of the bit stream. The parser reads one character at a time and determines whether the character comprises one or more bytes. The default is dependent on the physical type of the object.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • `Not Applicable` • `Left Justify` • `Right Justify`

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the drop-down list. • Enter a character between quotation marks, for example "c" or 'c', where c is any alphanumeric character. • Enter a hexadecimal character code in the form `0xYY`, where `YY` is a hexadecimal value. • Enter a Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.
Length Reference	Enumerated type	This property is applicable only if `Physical Type` is `Text`, `Binary`, or `TLOG Specific`. If set, this property takes precedence over any value in the `Length Units` property. Specify the identifier of a sibling integer object, the value of which determines the length of the object in question. The sibling object must be defined before the current object within the message structure. For information about reordering elements, see “Reordering objects” on page 2900.

Property	Type	Meaning
Inclusive Length Reference	Check box	This property is applicable only if `Length Reference` is set. If the check box is selected, the value of the sibling integer object that is identified by `Length Reference` is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by `Length Reference` is the length of the current object only. If the check box is selected, the `Length Units` property of the sibling integer object must be the same as that of the current object.

Numeric representation

Property	Type	Meaning
Sign Orientation	Enumerated type	The values that you can choose for this property are: • `Leading Separate` • `Trailing Separate` • `Leading Overpunched` • `Trailing Overpunched` This property is enabled only if you have set `Physical Type` to `Text` or `External Decimal`, and you have selected `Signed`. If `Physical Type` is `Text`, the only valid values of `Sign Orientation` are `Leading Separate` and `Trailing Separate`. If `Physical Type` is `External Decimal` and `Sign EBCDIC Customer Overpunched` is selected, the only valid values of `Sign Orientation` are `Leading Overpunched` and `Trailing Overpunched`.
Positive Sign	String	Specify the value that represents the positive symbol. Do not specify a numeric value. If no value is set, "+" is assumed. The positive sign is not written when creating an output message; it is used only to recognize the positive sign when parsing a message bit stream. This property is applicable only if `Physical Type` is `Text` and `Signed` is selected.
Negative Sign	String	Specify the value that represents the negative symbol. Do not specify a numeric value. If no value is set, "-" is assumed. This property is applicable only if `Physical Type` is `Text` and `Signed` is selected.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

TDS properties for local attribute string types:

The TDS format properties for local attribute string types.

The TDS Format properties described here apply to:

- Objects: Local Attribute
- String schema types: anyURI, ENTITIES, ENTITY, ID, IDREF, IDREFS, language, Name, NCName, NMTOKEN, NMTOKENS, normalizedString, NOTATION, QName, string, token

Field Identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the `Data Element Separation` property of the complex type or types in which the object is a child is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, this property must contain a non-empty value. If the object is a complex element, and the `Data Element Separation` property of its parent is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, the property can contain an empty value. The value for this property must be unique for every element in the message set; that is, no two elements in the message set can contain the same value for this property.

Property	Type	Meaning
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the `Data Element Separation` method is set to `Use Data Pattern` in the complex type. For more details, see "Regular expression syntax" on page 6304.
Interpret Element Value	Enumerated type	Specify whether values stored within this object are interpreted as having significance for the parser and, if so, the type of interpretation that occurs. This interpretation is standard-specific and is therefore hard coded. The possible values for this property are: • `None` (the default value) • `EDIFACT Service String` • `X12 Service String` • `Message Key` • `EDIFACT Syntax Level ID` • `HL7 Service String` • `HL7 Field Separator` **Note:** The `Message Key` enumeration has been deprecated.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the `Messaging Standard` property of the message set is `User Defined Text`, `User Defined Mixed`, `CSV`, or `TLOG`, select one of the following values: • `Text`. The data is in character format. • `Length Encoded String 1`. The first byte of the data contains the length (in length units) of the data string that follows the length byte. The maximum length of a `Length Encoded String 1` element is 255 length units. • `Length Encoded String 2`. The first two bytes of the data contain the length (in length units) of the data string that follows the two length bytes. The maximum length of a `Length Encoded String 2` element is 65535 length units. • `Null Terminated String`. The data string ends with the hexadecimal NULL character, `X'00'`. • `TLOG Specific` - this option can be selected only if the `Message Standard` property of the message set is `TLOG`. This option indicates that the format of the data is specific to the TLOG messaging standard. The default is dependent on the `Messaging Standard` property. For all other `Messaging Standard` values, the `Physical Type` property is set to `Text`.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no `Length Reference` is specified. If this property is not set and the message set property `Derive default length from logical type` is selected, and the `Physical type` is 'Character', the default value is derived from any length or maxLength value constraint (schema facet) on the simple type.

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the object. Select one of the following options (some physical types do not offer both options): • `Bytes`. The length is given in bytes. • `Characters`. The length is given in characters. The number of bytes that are processed in the bit stream depends on the code page of the message. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the contents of the bit stream. The parser reads one character at a time and determines whether the character comprises one or more bytes. The default is dependent on the physical type of the object.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • `Not Applicable` • `Left Justify` • `Right Justify`

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the drop-down list. • Enter a character between quotation marks, for example "c" or 'c', where c is any alphanumeric character. • Enter a hexadecimal character code in the form `0xYY`, where `YY` is a hexadecimal value. • Enter a Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.
Length Reference	Enumerated type	This property is applicable only if `Physical Type` is `Text`, `Binary`, or `TLOG Specific`. If set, this property takes precedence over any value in the `Length Units` property. Specify the identifier of a sibling integer object, the value of which determines the length of the object in question. The sibling object must be defined before the current object within the message structure. For information about reordering elements, see “Reordering objects” on page 2900.

Property	Type	Meaning
Inclusive Length Reference	Check box	This property is applicable only if `Length Reference` is set. If the check box is selected, the value of the sibling integer object that is identified by `Length Reference` is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by `Length Reference` is the length of the current object only. If the check box is selected, the `Length Units` property of the sibling integer object must be the same as that of the current object.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

Documentation properties for all message set objects:

Use the documentation property of an object to add information that describes the function of the object.

The documentation property is available on all objects except Key, Keyref, and Unique objects.

The property is a string field, and you can use any standard alphanumeric characters.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

Local element properties:

Different types of properties are available for a local element.

A local element can have the following properties;

- "Local element logical properties" on page 5442
- "Local element CWF properties" on page 5469
- "Local element XML properties" on page 5492
- "Local element TDS properties" on page 5524
- "Documentation properties for all message set objects" on page 5413

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

Local element logical properties:

The logical properties of a local element include properties that specify the number of occurrences and value of the local element.

Property	Type	Meaning
Name	String	Specify a name for the object when you create it. Names can consist of alphanumeric characters, including the letters *A* through *Z*, *a* through *z*, and the digits *0* through *9*. They might also include the following punctuation characters; • - the hyphen • _ the underscore • . the period Names can start only with a letter or the underscore character, and not with a number, hyphen, or period. Names that begin with `xml`, or any variant of these characters (for example `XmL`), are reserved by the XML standards specification. Further details of naming conventions and allowable characters can be found in the Extensible Markup Language (XML) specification that can be found on the World Wide Web Consortium (W3C) Web site.

Property	Type	Meaning
Type	Enumerated type	The Type property constrains the type of data that can be present in the object. Select from the following options: • `int` • `string` • `Boolean` • `hexBinary` • `dateTime` • `date` • `time` • `decimal` • `float` • `(More...)` • `(New Simple Type)` • `(New Complex Type)` If you select (More...), the Type Selection wizard starts. In this wizard, you can select any of the available types. If you select (New Simple Type), the New Simple Type wizard starts. In this wizard, you can create an Anonymous simple type that is based on an existing type. This type can be created locally or globally. If you select (New Complex Type), the New Complex Type wizard starts. In this wizard, you can create an Anonymous complex type, which can be derived from an existing base type. This type can be created locally or globally. For further information about these types, and examples of their use, see the XML Schema Part 0: Primer. This document is available on the World Wide Web Consortium (W3C) Web site.
Namespace	Enumerated type	Namespaces are a simple method for qualifying element and attribute names by associating them with namespaces identified by URI references. If <no target namespace> is displayed, a namespace has not been set for this object. If the property is inactive, the message set has not been configured to support namespaces. Where the property is active, namespaces that are available for selection are displayed in the drop-down list.

Occurrences

Property	Type	Meaning
Min Occurs	Integer	Specify the minimum number of times that the object can repeat. The default value is 1. If the value is set to 0, the object is optional. With the exception of `Max Occurs` being set to -1, if a value is set for `Min Occurs`, it must be less than or equal to the value in `Max Occurs`.

Property	Type	Meaning
Max Occurs	Integer	Specify the maximum number of times that the object can repeat. The default value is 1. If this property is not set, the object cannot occur more than once. If this property is set to 0, it is interpreted as if the object does not exist in the message. It can also be set to -1, to indicate that the limit is unbounded and there is no maximum to the number of occurrences.

Value

Property	Type	Meaning
Default	Button and String	This property provides the default value for an element or attribute. **XMLNSC domain** When parsing with validation enabled, default values are applied to missing attributes and empty elements as required by the XML Schema 1.0 specification. When writing, elements or attributes that are missing from the message tree are not automatically added to the output XML bit stream, even if they have default values. If missing elements or attributes are required, the message tree can be serialized and then re-parsed with validation enabled. **MRM (CWF and TDS physical formats)** When writing a fixed-length portion of a message (CWF or fixed-length TDS), if an attribute or element is missing from the message tree, the default value is inserted into the bit stream so that the message structure is preserved. **MRM (XML physical format)** No support for default values **Other domains** No support for default values.
Fixed	Button and String	This property provides the fixed value for an element or attribute. **XMLNSC domain** When parsing with validation enabled, if an attribute or element is present, the value is validated against the fixed value. If the values are not equal, a validation error is signalled. Also, when parsing with validation enabled, fixed values are applied to missing attributes and empty elements as required by the XML Schema 1.0 specification. When writing, elements or attributes that are missing from the message tree are not automatically added to the output XML bit stream, even if a fixed value has been specified. If missing elements or attributes are required, the message tree can be serialized and then re-parsed with validation enabled. **MRM (CWF and TDS physical formats)** When writing a fixed-length portion of a message (CWF or fixed-length TDS), if an attribute or element is missing from the message tree, the fixed value is inserted into the bit stream so that the message structure is preserved. **MRM (XML physical format)** No support for fixed values **Other domains** No support for fixed values.

Property	Type	Meaning
Nillable	Check box	Select this option if you want the element to be able to be defined as null. A null value is distinct from being empty, when the element contains no data.
Interpret Value As	Enumerated type	Specify if values stored within this object must be interpreted as having significance for the parser and, if so, the type of interpretation that must occur. Select one of the following options: • None This value is the default value, and indicates that the element or attribute does not have a key value associated with it. • MessageSetIdentity. Specifies that the value of the element or attribute corresponds to the identifier, name, or alias (in that priority order) that is associated with the message set where all subsequent embedded messages that are descendents of the enclosing message are defined. This value remains in force unless a new element or attribute MessageSetIdentity field is encountered which resets the MessageSetIdentity value. • MessageIdentity. Specifies that the value of the element or attribute corresponds to the name or alias (in that priority order) that is associated with a message, and acts as an identifier for subsequent embedded messages which are the immediate children of the enclosing message. This identity applies until a new element or attribute MessageIdentity field is encountered at the same level in the tree. The embedded message can be defined in either the current message set, or in a message set identified by using a MessageSetIdentity. **Note:** This property is applicable only when the type of the object is derived from xsd:string.

Substitution settings

Substitution Groups provide a means by which one element may be substituted for another in a message. The element which can be substituted is called the 'head' element, and the substitution group is the list of elements that may be used in its place. An element can be in at most one substitution group.

Property	Type	Meaning
Final	Enumerated type	Limit the set of elements that can belong to its substitution group. • Empty • restriction. Prohibit element substitution by elements whose types are restrictions of the type of the head element. • extension. Prohibit element substitution by elements whose types are extensions of the type of the head element. • #all. Prohibit substitution by all methods.
Block	Enumerated type	Limit the set of elements that can be substituted for this element in a message. • Empty • restriction. Prohibit element substitution by elements whose types are restrictions of the type of the head element • extension. Prohibit element substitution by elements whose types are extensions of the type of the head element • substitution. Prohibit element substitution by members of the element's substitution group. • #all. Prohibit substitution by all methods.
Substitution Group	Enumerated type	Specify the name of a 'head' element. Setting this property indicates that this element is a member of the substitution group for the head element.

Property	Type	Meaning
Abstract	Check box	Select this option if you do not want the element to appear in the message, but require one of the members of its substitution group to appear in its place.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Local element CWF properties:

The properties, and their permissible values, vary according to the object type.

The properties that are displayed on the object page, and the values that those properties can take, can vary according to the type of the object. For example, the properties for type *string* are different from those of type *Boolean*. Select the link for the object type from the following table.

Type of object	Properties
Binary types	• base64Binary • hexBinary
Boolean types	• Boolean
DateTime types	• date • dateTime • gDay • gMonth • gMonthDay • gYear • gYearMonth • time
Decimal types	• decimal • integer • negativeInteger • nonNegativeInteger • nonPositiveInteger • positiveInteger • unsignedLong
Float types	• double • float

Type of object	Properties
Integer types	• byte • int • long • short • unsignedByte • unsignedInt • unsignedShort
Interval types	• duration
String types	• anyURI • ENTITIES • ENTITY • ID • IDREF • IDREFS • language • Name • NCName • NMTOKEN • NMTOKENS • normalizedString • NOTATION • QName • string • token

Note: The physical format properties for simple type *duration* are the same as the physical properties of the string logical types.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM Custom Wire Format" on page 1214
Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not separated by delimiters.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

CWF properties for element reference and local element binary types:

CWF wire format properties for element reference and local element binary types.

The Custom Wire Format properties described here apply to:

- Objects: Element Reference, Local Element
- Binary schema types: base64Binary, hexBinary

Physical representation

Property	Type	Meaning
Length	Button and Integer	If you have selected the length to be defined by *Length*, enter the number of length units for the element. The minimum value that you can specify is 1. The maximum value that you can specify is 2147483647. The default value is empty (not set).
Length Reference	Button and Enumerated type	If you have selected the length to be defined by *Length Reference*, select the name of the integer object that specifies the length of this object. Make your selection from the displayed list of integer objects that are defined as siblings of the current object, and occur before it in the structure of the message. For information about reordering elements, see "Reordering objects" on page 2900.
Inclusive Length Reference	Check box	This property is applicable only if *Length Reference* is set. If the check box is selected, the value of the sibling integer object that is identified by *Length Reference* is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by *Length Reference* is the length of the current object only. If the check box is selected, the *Length Units* property of the sibling integer object must be the same as that of the current object.

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. The length is given in bytes. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value for the *`Length Count`* or *`Length Reference`* property. The default is `Bytes`.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to the first instance only.

Property	Type	Meaning
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to all instances.

Occurrences

Property	Type	Meaning
Repeat Reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the displayed list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, there are no integer objects before this one in the message structure. If a *`Repeat Reference`* is specified, it overrides any setting for the *`Max Occurs`* logical property when parsing and writing the message, but not for validation of the message.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

CWF properties for element reference and local element Boolean types:

The CWF wire format properties for element reference and local element Boolean types.

The Custom Wire Format properties described here apply to:

- Objects: Element Reference, Local Element
- Boolean schema types: Boolean

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to all instances.

Occurrences

Property	Type	Meaning
Repeat Reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the displayed list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* logical property when parsing and writing the message, but not for validation of the message.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

CWF properties for element reference and local element dateTime types:

The CWF wire format properties for element reference and local element dateTime types.

The Custom Wire Format properties described here apply to:

- Objects: Element Reference, Local Element
- DateTime schema types: date, dateTime, gDay, gMonth, gMonthDay, gYear, gYearMonth, time

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select one from the displayed list: • `Fixed Length String`. The element's length is determined by other length properties as follows. • `Length Encoded String 1`. The first byte of the element contains the length of the string following the length byte in length units. The maximum length of a `Length Encoded String 1` element is 255 length units. • `Length Encoded String 2`. The element's first 2 bytes contain the length of the string following the two length bytes in length units. The maximum length of a `Length Encoded String 2` element is 65535 length units. The two length bytes are in the format of the WebSphere MQ queue manager Encoding. • `Null Terminated String`. The string ends with the hexadecimal NULL character, `X'00'`. • `Packed Decimal`. The dateTime is coded as a Packed Decimal number. It is valid only if the *`DateTime Format`* property represents numeric-only data. • `Binary`. The dateTime is encoded as a binary sequence of bytes. If you select this option, the range of symbols that you can specify for the Format String property is less than the range of symbols you can specify if you select a string option (see "DateTime formats" on page 6310 for details). • `Time Seconds`. This value supports C time_t and Java Date and Time objects. It is valid only if the *`DateTime Format`* property represents numeric-only data. • `Time Milliseconds`. This value supports C time_t and Java Date and Time objects. It is valid only if the *`DateTime Format`* property represents numeric-only data. The default value is `fixed length string`.
Length	Button and Integer	If you have selected a *`Physical Type`* of `Fixed Length String`, `Packed Decimal`, or `Binary`, and have selected the length to be defined by *`Length`*, enter the number of length units for the element. The minimum value that you can specify is 1 for all three physical types. The maximum value that you can specify is `256` for `Fixed Length String`, `10` for `Packed Decimal`, and `2147483647` for `Binary`. The default value is 0 (zero).
Length Reference	Button and Enumerated type	If you have selected the length to be defined by *`Length Reference`*, select the name of the integer object that specifies the length of this object. Make your selection from the displayed list of integer objects that are defined as siblings of the current object, and occur before it in the structure of the message. For information about reordering elements, see "Reordering objects" on page 2900.

Property	Type	Meaning
Inclusive Length Reference	Check box	This property is applicable only if *Length Reference* is set. If the check box is selected, the value of the sibling integer object that is identified by *Length Reference* is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by *Length Reference* is the length of the current object only. If the check box is selected, the *Length Units* property of the sibling integer object must be the same as that of the current object.
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • Bytes. The length is given in bytes. • Characters. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • Character Units. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to Characters. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the Length property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • End of Bitstream. All data up to the end of the bitstream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value for the *Length Count* or *Length Reference* property. The default is Bytes.
Justification	Enumerated type	If you have set the *Physical Type* property to Fixed Length String, select Left Justify (the default value) or Right Justify from the list. If you have selected another value for *Physical Type*, this property is inactive.

Property	Type	Meaning
Padding Character	String	If you have set the *Physical Type* property to `Fixed Length String`, and the *Justification* property is either `Left Justify` or `Right Justify`, this property is applicable. When writing an output message, use the padding character to fill out the remaining character positions when the string length is less than the length implied by the *Length* or *Length Reference* property. Whether the string is padded from the left or the right is governed by the *Justification* property. When parsing an input message, the padding character is trimmed from the end of the string. Whether the string is trimmed from the left or the right is governed by the *Justification* property. Specify this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the displayed list. • Enter a character between quotation marks; for example, `"c"` or `'c'`, where c is any alphanumeric character. • Enter a Unicode value in the form `U+xxxx` where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form `0xYY`, where `YY` is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, it is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.
DateTime Format	String	Specify a template for date and time. The default dateTime format is dependent on the logical type of the object. For information about the defaults for the dateTime format according to the logical type, see “DateTime defaults by logical type” on page 6320. See “DateTime formats” on page 6310 for details of date and time formats.

Numeric representation

Property	Type	Meaning
Signed	Check box	Specify whether the value is signed. This property is applicable only if the *Physical type* property is Packed Decimal. By default, this check box is cleared, which indicates that the value is not signed.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the displayed list. The option that you select determines the value that you must set for the property *Encoding Null Value*: • NULLPadFill. This option is valid only if *Physical Type* is Fixed Length String. The field is filled with the value specified by the *Padding Character*. The default value. • NULLLogicalValue. The *Encoding Null Value* property is first converted to an actual value, and rendered in the way specified for the field. • NULLLiteralValue. This specifies that *Encoding Null Value* contains a value that is directly substituted as if it is a string. Use this option when the value you have set for *Encoding Null Value* to specify a null date is not a dateTime value, or does not conform to the standard dateTime format yyyy-MM-dd 'T'HH:mm:ss. • NULLLiteralFill. The field is filled with the value specified by the *Encoding Null Value*. *Encoding Null Value* must resolve to a single character.
Encoding Null Value	String	If you set the *Encoding Null* property to NULLPadFill, this property is disabled. If you set the *Encoding Null* property to NULLLogicalValue, you must set this property to an ISO8601 dateTime format. These formats are described in "DateTime as string data" on page 6311. For example, specify a value conforming to yyyy-MM-dd'T'HH:mm:ss such as 1970-12-01. If you set the *Encoding Null* property to NULLLiteralValue, you can enter any value that is the same length as the field. If you set the *Encoding Null* property to NULLLiteralFill, the value must resolve to a single character. Set the character in one of the following ways: • Select SPACE, NUL, 0x00 or 0xFF from the displayed list • Enter a character between quotation marks, for example 'c' or "c", where c is any alphanumeric character. • Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value. • Enter a decimal character code in the form YY where YY is a decimal value. • Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal format.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to all instances.

Occurrences

Property	Type	Meaning
Repeat Reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the displayed list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, there are no integer objects before this one in the message structure. If a *`Repeat Reference`* is specified, it overrides any setting for the *`Max Occurs`* logical property when parsing and writing the message, but not for validation of the message.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

CWF properties for element reference and local element decimal types:

The CWF wire format properties for element reference and local element decimal types.

The Custom Wire Format properties described here apply to:

- Objects: Element Reference, Local Element
- Decimal schema types: decimal, integer, negativeInteger, nonNegativeInteger, nonPositiveInteger, positiveInteger, unsignedLong

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select one from the displayed list: • `Integer`. This equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • `Packed Decimal`. Equates to the COMP-3 data type in COBOL. • `External Decimal`. Equates to the data type PIC 9 USAGE DISPLAY in COBOL. The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager: • Elements that have *`Physical Type`* set to `Integer` and `Packed Decimal` are represented in the appropriate WebSphere MQ Encoding value. • Elements that have *`Physical Type`* set to `External Decimal` are represented in the WebSphere MQ CodedCharSetId value.

Property	Type	Meaning
Length	Integer	Enter the number of bytes to specify the element length: • If you set the *Physical Type* to Integer, select 1, 2, or 4 (the default) from the displayed list. • If you set the *Physical Type* to Packed Decimal, enter a value between 1 and 10. • If you set the *Physical Type* to Extended Decimal, enter a value between 1 and 256. (Numbers greater than the maximum COBOL PICTURE clause of 18 are assumed to be 18.)
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • Bytes. • Characters. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • Character Units. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to Characters. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the Length property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • End of Bitstream. All data up to the end of the bitstream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value for the *Length Count*. The default is Bytes.
Justification	Enumerated type	If you have set the *Physical Type* property to External Decimal, select Left Justify or Right Justify (the default value) from the list. If you have selected another value for *Physical Type*, this property is inactive.

Property	Type	Meaning
Padding Character	String	The padding character is used to fill out the remaining character positions when the string length is less than the specified string size. If you have set the *Physical Type* property to Extended Decimal, and the *Justification* property is either Left Justify or Right Justify, specify this character in one of the following ways: • Select NUL, '0', or SPACE from the displayed list. • Enter a character between quotation marks; for example, "c" or 'c', where c is any alphanumeric character. • Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form 0xYY, where YY is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Numeric representation

Property	Type	Meaning
Signed	Check box	Select (the default) or clear this property. This property is used with *Sign Orientation*.

Property	Type	Meaning
Sign EBCDIC Custom Overpunched	Check box	If the *Physical Type* is set to External Decimal and the *Signed EBCDIC Custom* property is set, this indicates that the Sign EBCDIC Custom Overpunched representation is to be used within an ASCII environment. If this check box is not selected (the default), the Sign ASCII representation is used. The setting of the *Sign EBCDIC Custom Overpunched* check box is appropriate only if the *Sign Orientation* property is set to Leading or Trailing (indicating that the element/attribute has an embedded sign representation). The check box is not available if the element/attribute is unsigned (for example, if the *Signed* check box is not set).
Sign Orientation	Enumerated type	If you have set *Physical Type* to External Decimal and you have selected *Signed*, choose from the following options that represent the COBOL options for displaying numeric data: • Leading Overpunched. This option sets a bit in the first byte if the number is negative. No setting is made if the number is positive. For example, the ASCII hexadecimal representation of the number 22 is x'3232'. Using this option, the number +22 would be x'3232' and the number -22 would be x'7232'. The default value. • Leading Separate. This option sets the first byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte. • Trailing Overpunched. This option sets a bit in the last byte if the number is negative. No setting is made if the number is positive. Using this option, the number +22 would be x'3232' and the number -22 would be x'3272'. • Trailing Separate. This option sets the last byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte. If you have set *Physical Type* to any other value, the value Not Applicable is set for you.
Virtual Decimal Point	Integer	Specify the number of places to the left (for a positive value) or right (for a negative value) that a decimal point is to be moved from its assumed position. For example, a decimal element containing 1234 with a Virtual Decimal value of 3 is 1.234, equivalent to 'V' or 'P' in a COBOL picture clause. There is no C equivalent

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the displayed list: • NULLPadFill. This option is valid only if *Physical Type* is External Decimal. The field is filled with the value specified by the *Padding Character*. *Encoding Null Value* must be set to an empty string. • NULLLogicalValue. The *Encoding Null Value* is transformed to match the required format for the field. The default value. • NULLLiteralValue. The *Encoding Null Value* is directly substituted as if it is a string. You can specify a nonnumeric value for *Encoding Null Value*. • NULLLiteralFill. The field is filled with the value specified by the *Encoding Null Value*. *Encoding Null Value* must resolve to a single character.

Property	Type	Meaning
Encoding Null Value	String	The use of this property depends on the *Encoding Null* property, except for NULLLiteralFill. The default value is zero. If you set the *Encoding Null* property to NULLLiteralFill, the value must resolve to a single character. Set the character in one of the following ways: • Select SPACE, NUL, 0x00 or 0xFF from the displayed list • Enter a character between quotation marks, for example 'c' or "c", where c is any alphanumeric character. • Enter a hexadecimal character in the form 0xYY where YY is a hexadecimal value. • Enter a decimal character code in the form YY where YY is a decimal value. • Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal format.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • 1 Bytes. The default value. • 2 Bytes • 4 Bytes • 8 Bytes • 16 Bytes
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to all instances.

Occurrences

Property	Type	Meaning
Repeat Reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the displayed list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* logical property when parsing and writing the message, but not for validation of the message.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

CWF properties for element reference and local element float types:

The CWF wire format properties for element reference and local element float types.

The Custom Wire Format properties described here apply to:

- Objects: Element Reference, Local Element
- Float schema types: double, float

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select one from the displayed list: • `Integer`. This equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • `Float`. This equates to the data type FLOAT or DOUBLE in C or the COMP-1 or COMP-2 data type in COBOL and is the default value. • `Packed Decimal`. Equates to the COMP-3 data type in COBOL. • `External Decimal`. Equates to the data type PIC 9 USAGE DISPLAY in COBOL. The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager: • Elements that have *`Physical Type`* set to `Integer`, `Packed Decimal`, and `Float` are represented in the appropriate WebSphere MQ Encoding value. • Elements that have *`Physical Type`* set to `External Decimal` are represented in the WebSphere MQ CodedCharSetId value.
Length	Integer	Enter the number of bytes to specify the element length: • If you set the *`Physical Type`* to `Float`, select a value from the displayed list. The default value is 8. • If you set the *`Physical Type`* to `Integer`, select `1`, `2`, or `4` (the default) from the displayed list. • If you set the *`Physical Type`* to `Packed Decimal`, enter a value between 1 and 10. • If you set the *`Physical Type`* to `Extended Decimal`, enter a value between 1 and 256. (Numbers greater than the maximum COBOL PICTURE clause of 18 are assumed to be 18.)

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value for the *`Length Count`*. The default is `Bytes`.
Justification	Enumerated type	If you have set the *`Physical Type`* property to `External Decimal`, select `Left Justify` or `Right Justify` (the default value) from the list. If you have selected another value for *`Physical Type`*, this property is inactive.

Property	Type	Meaning
Padding Character	String	The padding character is used to fill out the remaining character positions when the string length is less than the specified string size. If you have set the *Physical Type* property to Extended Decimal, and the *Justification* property is either Left Justify or Right Justify, specify this character in one of the following ways: • Select NUL, '0', or SPACE from the displayed list. • Enter a character between quotation marks; for example, "c" or 'c', where c is any alphanumeric character. • Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form 0xYY, where YY is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Numeric representation

Property	Type	Meaning
Signed	Check box	Select or clear (unsigned, the default) this property. If you have set *Physical Type* to Float, this is selected. This property is used with *Sign Orientation*.

Property	Type	Meaning
Sign EBCDIC Custom Overpunched	Check box	If the *Physical Type* is set to External Decimal and the *Signed EBCDIC Custom* property is set, this indicates that the Sign EBCDIC Custom Overpunched representation is to be used within an ASCII environment. If this check box is not selected (the default), the Sign ASCII representation is used. The setting of the *Sign EBCDIC Custom Overpunched* check box is appropriate only if the *Sign Orientation* property is set to Leading or Trailing (indicating that the element/attribute has an embedded sign representation). The check box is not available if the element/attribute is unsigned (for example, if the *Signed* check box is not set).
Sign Orientation	Enumerated type	If you have set *Physical Type* to External Decimal and you have selected *Signed*, choose from the following options that represent the COBOL options for displaying numeric data: • Leading Overpunched. This option sets a bit in the first byte if the number is negative. No setting is made if the number is positive. For example, the ASCII hexadecimal representation of the number 22 is x'3232'. Using this option, the number +22 would be x'3232' and the number -22 would be x'7232'. The default value. • Leading Separate. This option sets the first byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte. • Trailing Overpunched. This option sets a bit in the last byte if the number is negative. No setting is made if the number is positive. Using this option, the number +22 would be x'3232' and the number -22 would be x'3272'. • Trailing Separate. This option sets the last byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte. If you have set *Physical Type* to any other value, the value Not Applicable is set for you.
Virtual Decimal Point	Integer	Specify the number of places to the left (for a positive value) or right (for a negative value) that a decimal point is to be moved from its assumed position. For example, a float element containing 1234 with a Virtual Decimal value of 3 is 1.234. This property is not applicable if you have set *Physical Type* to Float.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the displayed list: • NULLPadFill. This option is valid only if *Physical Type* is External Decimal. The field is filled with the value specified by the *Padding Character*. *Encoding Null Value* must be set to an empty string. • NULLLogicalValue. The *Encoding Null Value* is transformed to match the required format for the field. The default value. • NULLLiteralValue. The *Encoding Null Value* is directly substituted as if it is a string. You can specify a nonnumeric value for *Encoding Null Value*. • NULLLiteralFill. The field is filled with the value specified by the *Encoding Null Value*. *Encoding Null Value* must resolve to a single character.

Property	Type	Meaning
Encoding Null Value	String	The use of this property depends on the *Encoding Null* property, except for NULLLiteralFill. The default value is zero. If you set the *Encoding Null* property to NULLLiteralFill, the value must resolve to a single character. Set the character in one of the following ways: • Select SPACE, NUL, 0x00 or 0xFF from the displayed list • Enter a character between quotation marks, for example 'c' or "c", where c is any alphanumeric character. • Enter a hexadecimal character in the form 0xYY where YY is a hexadecimal value. • Enter a decimal character code in the form YY where YY is a decimal value. • Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal format.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • 1 Bytes. The default value. • 2 Bytes • 4 Bytes • 8 Bytes • 16 Bytes
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to all instances.

Occurrences

Property	Type	Meaning
Repeat Reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the displayed list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* logical property when parsing and writing the message, but not for validation of the message.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

CWF properties for element reference and local element integer types:

The CWF wire format properties for element reference and local element integer types.

The Custom Wire Format properties described here apply to:

- Objects: Element Reference, Local Element
- Integer schema types: byte, int, long, short, unsignedByte, unsignedInt, unsignedShort

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select one from the displayed list: • `Integer`. This equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • `Packed Decimal`. Equates to the COMP-3 data type in COBOL. • `External Decimal`. Equates to the data type PIC 9 USAGE DISPLAY in COBOL. The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager: • Elements that have *`Physical Type`* set to `Integer` and `Packed Decimal` are represented in the appropriate WebSphere MQ Encoding value. • Elements that have *`Physical Type`* set to `External Decimal` are represented in the WebSphere MQ CodedCharSetId value.
Length	Integer	Enter the number of bytes to specify the element length: • If you set the *`Physical Type`* to `Integer`, select `1`, `2`, or `4` (the default) from the displayed list. • If you have set *`Physical Type`* to `Packed Decimal`, enter a value between 1 and 6. • If you have set *`Physical Type`* to `Extended Decimal`, enter a value between 1 and 11.
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value for the *`Length Count`*. The default is `Bytes`.

Property	Type	Meaning
Justification	Enumerated type	If you have set the *Physical Type* property to `External Decimal`, select `Left Justify` or `Right Justify` (the default value) from the list. If you have selected another value for *Physical Type*, this property is inactive.
Padding Character	String	The padding character is used to fill out the remaining character positions when the string length is less than the specified string size. If you have set the *Physical Type* property to `Extended Decimal`, and the *Justification* property is either `Left Justify` or `Right Justify`, specify this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the displayed list. • Enter a character between quotation marks; for example, `"c"` or `'c'`, where c is any alphanumeric character. • Enter a Unicode value in the form `U+xxxx` where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form `0xYY`, where `YY` is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Numeric representation

Property	Type	Meaning
Signed	Check box	Select (the default) or clear this property. This property is used with *Sign Orientation*.

Property	Type	Meaning
Sign EBCDIC Custom Overpunched	Check box	If the *Physical Type* is set to External Decimal and the *Signed EBCDIC Custom* property is set, this indicates that the Sign EBCDIC Custom Overpunched representation is to be used within an ASCII environment. If this check box is not selected (the default), the Sign ASCII representation is used. The setting of the *Sign EBCDIC Custom Overpunched* check box is appropriate only if the *Sign Orientation* property is set to Leading or Trailing (indicating that the element/attribute has an embedded sign representation). The check box is not available if the element/attribute is unsigned (for example, if the *Signed* check box is not set).
Sign Orientation	Enumerated type	If you have set *Physical Type* to External Decimal and you have selected *Signed*, choose from the following options that represent the COBOL options for displaying numeric data: • Leading Overpunched. This option sets a bit in the first byte if the number is negative. No setting is made if the number is positive. For example, the ASCII hexadecimal representation of the number 22 is x'3232'. Using this option, the number +22 would be x'3232' and the number -22 would be x'7232'. The default value. • Leading Separate. This option sets the first byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte. • Trailing Overpunched. This option sets a bit in the last byte if the number is negative. No setting is made if the number is positive. Using this option, the number +22 would be x'3232' and the number -22 would be x'3272'. • Trailing Separate. This option sets the last byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte. If you have set *Physical Type* to any other value, the value Not Applicable is set for you.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the displayed list: • NULLPadFill. This option is valid only if *Physical Type* is External Decimal. The field is filled with the value specified by the *Padding Character*. *Encoding Null Value* must be set to an empty string. • NULLLogicalValue. The *Encoding Null Value* is transformed to match the required format for the field. The default value. • NULLLiteralValue. The *Encoding Null Value* is directly substituted as if it is a string. You can specify a nonnumeric value for *Encoding Null Value*. • NULLLiteralFill. The field is filled with the value specified by the *Encoding Null Value*. *Encoding Null Value* must resolve to a single character.

Property	Type	Meaning
Encoding Null Value	String	The use of this property depends on the *Encoding Null* property, except for NULLLiteralFill. The default value is zero. If you set the *Encoding Null* property to NULLLiteralFill, the value must resolve to a single character. Set the character in one of the following ways: • Select SPACE, NUL, 0x00 or 0xFF from the displayed list • Enter a character between quotation marks, for example 'c' or "c", where c is any alphanumeric character. • Enter a hexadecimal character in the form 0xYY where YY is a hexadecimal value. • Enter a decimal character code in the form YY where YY is a decimal value. • Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal format.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • 1 Bytes. The default value. • 2 Bytes • 4 Bytes • 8 Bytes • 16 Bytes
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to all instances.

Occurrences

Property	Type	Meaning
Repeat Reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the displayed list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* logical property when parsing and writing the message, but not for validation of the message.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

CWF properties for element reference and local element string types:

The CWF wire format properties for element reference and local element string types.

The Custom Wire Format properties described here apply to:

- Objects: Attribute Reference, Local Attribute
- String schema types: anyURI, ENTITIES, ENTITY, ID, IDREF, IDREFS, language, Name, NCName, NMTOKEN, NMTOKENS, normalizedString, NOTATION, QName, string, token

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select one from the displayed list: • `Fixed Length String`. The element's length is determined by other length properties as follows. • `Length Encoded String 1`. The first byte of the element contains the length of the string following the length byte in length units. The maximum length of a `Length Encoded String 1` element is 255 length units. • `Length Encoded String 2`. The element's first 2 bytes contain the length of the string following the two length bytes in length units. The maximum length of a `Length Encoded String 2` element is 65535 length units. The two length bytes are in the format of the WebSphere MQ queue manager Encoding. • `Null Terminated String`. The string ends with the hexadecimal NULL character, `X'00'`. The default is `Fixed Length String`.
Length	Button and Integer	If you have selected a *`Physical Type`* of `Fixed Length String` or `Binary`, and have selected the length to be defined by *`Length`*, enter the number of length units for the element. The minimum value that you can specify is `0` (zero), the maximum value that you can specify is `2147483647` The default value is `0` (zero).
Length Reference	Button and Enumerated type	If you have selected the length to be defined by *`Length Reference`*, select the name of the integer object that specifies the length of this object. Make your selection from the displayed list of integer objects that are defined as siblings of the current object, and occur before it in the structure of the message. For information about reordering elements, see “Reordering objects” on page 2900.
Inclusive Length Reference	Check box	This property is applicable only if *`Length Reference`* is set. If the check box is selected, the value of the sibling integer object that is identified by *`Length Reference`* is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by *`Length Reference`* is the length of the current object only. If the check box is selected, the *`Length Units`* property of the sibling integer object must be the same as that of the current object.

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. The length is given in bytes. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value for the *`Length Count`* or *`Length Reference`* property. The default is `Bytes`.
Justification	Enumerated type	If you have set the *`Physical Type`* property to `Fixed Length String`, select `Left Justify` (the default value) or `Right Justify` from the list. If you have selected another value for *`Physical Type`*, this property is inactive.

Property	Type	Meaning
Padding Character	String	If you have set the *Physical Type* property to `Fixed Length String`, and the *Justification* property is either `Left Justify` or `Right Justify`, this property is applicable. When writing an output message, use the padding character to fill out the remaining character positions when the string length is less than the length implied by the *Length* or *Length Reference* property. Whether the string is padded from the left or the right is governed by the *Justification* property. When parsing an input message, the padding character is trimmed from the end of the string. Whether the string is trimmed from the left or the right is governed by the *Justification* property. Specify this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the displayed list. • Enter a character between quotation marks; for example, `"c"` or `'c'`, where c is any alphanumeric character. • Enter a Unicode value in the form `U+xxxx` where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form `0xYY`, where `YY` is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, it is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the displayed list: • NULLPadFill. This is valid only if *Physical Type* is Fixed Length String. The field is filled with the value specified by the *Padding Character*. *Encoding Null Value* must be set to an empty string. • NULLLogicalValue. The *Encoding Null Value* is transformed to match the required format for the field. The default value. • NULLLiteralValue. The *Encoding Null Value* is directly substituted as if it is a string. • NULLLiteralFill. The field is filled with the value specified by the *Encoding Null Value*. *Encoding Null Value* must resolve to a single character.
Encoding Null Value	STRING	The use of this property depends on the *Encoding Null* property. If specified, its length must be equal to the length of the string element, except for NULLLiteralFill. The default value is empty (not set). If you set the *Encoding Null* property to NULLLiteralFill, the value must resolve to a single character. Set the character in one of the following ways: • Select SPACE, NUL, 0x00 or 0xFF from the displayed list • Enter a character between quotation marks, for example 'c' or "c", where c is any alphanumeric character. • Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value. • Enter a decimal character code in the form YY where YY is a decimal value. • Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal format.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • 1 Bytes. The default value. • 2 Bytes • 4 Bytes • 8 Bytes • 16 Bytes
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to all instances.

Occurrences

Property	Type	Meaning
Repeat Reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the displayed list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* logical property when parsing and writing the message, but not for validation of the message.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

Local element XML properties:

The properties, and their permissible values, vary according to the object type.

The properties that are displayed on the object page, and the values that those properties can take, can vary according to the type of the object. For example, the properties for type *string* are different from those of type *Boolean*. Select the link for the object type from the following table.

Type of object	**Properties**
Binary types	• base64Binary • hexBinary
Boolean types	• Boolean
DateTime types	• date • dateTime • gDay • gMonth • gMonthDay • gYear • gYearMonth • time
Decimal types	• decimal • integer • negativeInteger • nonNegativeInteger • nonPositiveInteger • positiveInteger • unsignedLong
Float types	• double • float
Integer types	• byte • int • long • short • unsignedByte • unsignedInt • unsignedShort
Interval types	• duration

Type of object	Properties
String types	• anyURI • ENTITIES • ENTITY • ID • IDREF • IDREFS • language • Name • NCName • NMTOKEN • NMTOKENS • normalizedString • NOTATION • QName • string • token

Note: The physical format properties for simple type *duration* are the same as the physical properties of the string logical types.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

XML properties for attribute reference, element reference, local attribute, local element binary types:

XML wire format properties for attribute reference, element reference, local attribute, and local element binary types.

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- Binary schema types: base64Binary, hexBinary

Field identification

A number of the following properties only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated type	Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list: • `XMLElement`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their *XML Name*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object , you must set their *XML Name* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in *ID Attribute Name* and a value as specified in *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in *Value Attribute Name*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *ID Attribute Name*. The value is the value of *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 6262 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
XML Name	String	Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message). This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name. You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name. If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).
ID Attribute Name	String	Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
ID Attribute Value	String	Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

Physical representation

Property	Type	Meaning
Encoding	String	Select one of the following values from the drop-down list: : • `CDatahex` (the default). Hexadecimal values in this field are specified with the CDATA qualifier, for example <e1><![CDATA[62]]></e1> • `hex`. Hexadecimal values in this field are specified as digits only, for example <e1>62</e1>. • `base64`. Values in this field are specified as digits only, coded in base 64.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

XML properties for attribute reference, element reference, local attribute, local element Boolean types:

XML wire format properties for attribute reference, element reference, local attribute and local element Boolean types.

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- Boolean schema types: Boolean

Field identification

A number of the following properties only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated type	Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list: • `XMLElement`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their *`XML Name`*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object , you must set their *`XML Name`* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in *`ID Attribute Name`* and a value as specified in *`ID Attribute Value`*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *`XML Name`*, *`ID Attribute Name`*, *`ID Attribute Value`* to the same values: – You must also set *`Value Attribute Name`* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in *`Value Attribute Name`*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *`ID Attribute Name`*. The value is the value of *`ID Attribute Value`*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *`XML Name`*, *`ID Attribute Name`*, *`ID Attribute Value`* to the same values: – You must also set *`Value Attribute Name`* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 6262 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
XML Name	String	Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message). This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name. You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name. If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).
ID Attribute Name	String	Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
ID Attribute Value	String	Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

XML properties for attribute reference, element reference, local attribute, local element dateTime types:

XML wire format properties for attribute reference, element reference, local attribute, and local element dateTime types.

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- DateTime schema types: date, dateTime, gDay, gMonth, gMonthDay, gYear, gYearMonth, time

Field identification

A number of the following properties only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated type	Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list: • `XMLElement`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their *XML Name*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object , you must set their *XML Name* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in *ID Attribute Name* and a value as specified in *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in *Value Attribute Name*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *ID Attribute Name*. The value is the value of *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 6262 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
XML Name	String	Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message). This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name. You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name. If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).
ID Attribute Name	String	Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
ID Attribute Value	String	Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

Physical representation

Property	Type	Meaning
DateTime Format	String	Specify a format string that specifies the rendering of the value for dateTime elements. The default dateTime format is dependent on the logical type of the object. For information about the defaults for the dateTime format according to the logical type see “DateTime defaults by logical type” on page 6320. See “DateTime formats” on page 6310 for details of dateTime formats.

Related concepts:

“Message modeling” on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

“The message model” on page 1160
The message model consists of the following components.

“MRM XML physical format: Relationship to the logical model” on page 1251
The MRM XML physical format generally respects all the settings in the logical

model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

XML properties for attribute reference, element reference, local attribute, local element decimal types:

XML wire format properties for attribute reference, element reference, local attribute, and local element decimal types.

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- Decimal schema types: decimal, integer, negativeInteger, nonNegativeInteger, nonPositiveInteger, positiveInteger, unsignedLong

Field identification

A number of the following properties only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated type	Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list: • `XMLElement`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their *`XML Name`*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object , you must set their *`XML Name`* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in *`ID Attribute Name`* and a value as specified in *`ID Attribute Value`*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *`XML Name`*, *`ID Attribute Name`*, *`ID Attribute Value`* to the same values: – You must also set *`Value Attribute Name`* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in *`Value Attribute Name`*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *`ID Attribute Name`*. The value is the value of *`ID Attribute Value`*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *`XML Name`*, *`ID Attribute Name`*, *`ID Attribute Value`* to the same values: – You must also set *`Value Attribute Name`* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 6262 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
XML Name	String	Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message). This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name. You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name. If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).
ID Attribute Name	String	Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
ID Attribute Value	String	Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

XML properties for attribute reference, element reference, local attribute, local element float types:

XML wire format properties for attribute reference, element reference, local attribute, and local element float types.

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- Float schema types: double, float

Field identification

A number of the following properties only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated type	Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list: • `XMLElement`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their *XML Name*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object , you must set their *XML Name* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in *ID Attribute Name* and a value as specified in *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in *Value Attribute Name*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *ID Attribute Name*. The value is the value of *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 6262 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
XML Name	String	Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message). This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name. You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name. If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).
ID Attribute Name	String	Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
ID Attribute Value	String	Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

XML properties for attribute reference, element reference, local attribute, local element integer types:

XML wire format properties for attribute reference, element reference, local attribute, and local element integer types.

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- Integer schema types: byte, int, long, short, unsignedByte, unsignedInt, unsignedShort

Field identification

A number of the following properties only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated type	Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list: • `XMLElement`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their *`XML Name`*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object , you must set their *`XML Name`* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in *`ID Attribute Name`* and a value as specified in *`ID Attribute Value`*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *`XML Name`*, *`ID Attribute Name`*, *`ID Attribute Value`* to the same values: – You must also set *`Value Attribute Name`* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in *`Value Attribute Name`*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *`ID Attribute Name`*. The value is the value of *`ID Attribute Value`*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *`XML Name`*, *`ID Attribute Name`*, *`ID Attribute Value`* to the same values: – You must also set *`Value Attribute Name`* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 6262 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
XML Name	String	Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message). This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name. You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name. If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).
ID Attribute Name	String	Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
ID Attribute Value	String	Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

XML properties for attribute reference, element reference, local attribute, local element string types:

XML wire format properties for attribute reference, element reference, local attribute, and local element string types.

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- String schema types: anyURI, ENTITIES, ENTITY, ID, IDREF, IDREFS, language, Name, NCName, NMTOKEN, NMTOKENS, normalizedString, NOTATION, QName, string, token

Field identification

A number of the following properties only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated type	Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list: • `XMLElement`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their *XML Name*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object , you must set their *XML Name* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in *ID Attribute Name* and a value as specified in *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in *Value Attribute Name*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *ID Attribute Name*. The value is the value of *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 6262 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
XML Name	String	Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message). This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name. You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name. If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).
ID Attribute Name	String	Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
ID Attribute Value	String	Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

Local element TDS properties:

The properties, and their permissible values, vary according to the object type.

The properties that are displayed on the object page, and the values that those properties can take, can vary according to the type of the object. For example, the properties for type *string* are different from those of type *Boolean*. Select the link for the object type from the following table.

Type of object	**Properties**
Binary types	• base64Binary • hexBinary
Boolean types	• Boolean
Complex types	
DateTime types	• date • dateTime • gDay • gMonth • gMonthDay • gYear • gYearMonth • time
Decimal types	• decimal • integer • negativeInteger • nonNegativeInteger • nonPositiveInteger • positiveInteger • unsignedLong

Type of object	Properties
Float types	• double • float
Integer types	• byte • int • long • short • unsignedByte • unsignedInt • unsignedShort
Interval types	• duration
String types	• anyURI • ENTITIES • ENTITY • ID • IDREF • IDREFS • language • Name • NCName • NMTOKEN • NMTOKENS • normalizedString • NOTATION • QName • string • token

Note: The physical format properties for simple type *duration* are the same as the physical properties of the string logical types.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM TDS format" on page 1221
The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

TDS properties for local element binary types:

The TDS format properties for local element binary types.

The TDS Format properties described here apply to:

- Objects: Local Element
- Binary schema types: base64Binary, hexBinary

Field Identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the `Data Element Separation` property of the complex type or types in which the object is a child is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, this property must contain a non-empty value. If the object is a complex element, and the `Data Element Separation` property of its parent is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, the property can contain an empty value. The value for this property must be unique for every element in the message set; that is, no two elements in the message set can contain the same value for this property.
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the `Data Element Separation` method is set to `Use Data Pattern` in the complex type. For more details, see "Regular expression syntax" on page 6304.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	For all `Messaging Standard` values, the `Physical Type` property is set to `Binary` and cannot be changed.

Property	Type	Meaning
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no `Length Reference` is specified. The default is dependent on the setting of the message set property `Derive default length from logical type`. If `Derive default length from logical type` is selected, the default value is derived from any length or maxLength value constraint (schema facet) on the object's simple type.
Length Units	Enumerated type	Always set to `Bytes`.
Length Reference	Enumerated type	This property is applicable only if `Physical Type` is `Text`, `Binary`, or `TLOG Specific`. If set, this property takes precedence over any value in the `Length Units` property. Specify the identifier of a sibling integer object, the value of which determines the length of the object in question. The sibling object must be defined before the current object within the message structure. For information about reordering elements, see “Reordering objects” on page 2900.
Inclusive Length Reference	Check box	This property is applicable only if `Length Reference` is set. If the check box is selected, the value of the sibling integer object that is identified by `Length Reference` is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by `Length Reference` is the length of the current object only. If the check box is selected, the `Length Units` property of the sibling integer object must be the same as that of the current object.

Occurrences

Property	Type	Meaning
Repeat reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, no integer objects exist before this one in the message structure. If a `Repeat Reference` is specified, it overrides any setting for the `Max Occurs` logical property when parsing and writing the message, but not for validating the message.
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the `Data Element Separation` property of the parent group or complex type is set to `All Elements Delimited` or `Variable Length Elements Delimited`. A default value is provided if the previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; if the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used. If none of the previous conditions are true, a default is not applied.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

TDS properties for local element Boolean types:

The TDS format properties for local element Boolean types.

The TDS Format properties described here apply to:

- Objects: Local Element
- Boolean schema types: Boolean

Field Identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the `Data Element Separation` property of the complex type or types in which the object is a child is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, this property must contain a non-empty value. If the object is a complex element, and the `Data Element Separation` property of its parent is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, the property can contain an empty value. The value for this property must be unique for every element in the message set; that is, no two elements in the message set can contain the same value for this property.
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the `Data Element Separation` method is set to `Use Data Pattern` in the complex type. For more details, see “Regular expression syntax” on page 6304.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the `Messaging Standard` property of the message set is `User Defined Text`, `User Defined Mixed`, `CSV`, or `TLOG`, select one of the following values: • `Text`. The data is in character format. • `Binary`. The data is in bit string format. The default is dependent on the `Messaging Standard` property. For all other `Messaging Standard` values, the `Physical Type` property is set to `Text`.

Occurrences

Property	Type	Meaning
Repeat reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, no integer objects exist before this one in the message structure. If a `Repeat Reference` is specified, it overrides any setting for the `Max Occurs` logical property when parsing and writing the message, but not for validating the message.

Property	Type	Meaning
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the `Data Element Separation` property of the parent group or complex type is set to `All Elements Delimited` or `Variable Length Elements Delimited`. A default value is provided if the previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; if the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used. If none of the previous conditions are true, a default is not applied.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

TDS properties for local element of complex type:

The TDS Format properties that apply to local elements of complex type.

Field Identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the `Data Element Separation` property of the complex type or types in which the object is a child is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, this property must contain a non-empty value. If the object is a complex element, and the `Data Element Separation` property of its parent is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, the property can contain an empty value. The value for this property must be unique for every element in the message set; that is, no two elements in the message set can contain the same value for this property.
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the `Data Element Separation` method is set to `Use Data Pattern` in the complex type. For more details, see "Regular expression syntax" on page 6304.

Occurrences

Property	Type	Meaning
Repeat reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, no integer objects exist before this one in the message structure. If a `Repeat Reference` is specified, it overrides any setting for the `Max Occurs` logical property when parsing and writing the message, but not for validating the message.
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the `Data Element Separation` property of the parent group or complex type is set to `All Elements Delimited` or `Variable Length Elements Delimited`. A default value is provided if the previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; if the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used. If none of the previous conditions are true, a default is not applied.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

TDS properties for local element dateTime types:

The TDS format properties for local element dateTime types.

The TDS Format properties described here apply to:

- Objects: Local Element
- DateTime schema types: date, dateTime, gDay, gMonth, gMonthDay, gYear, gYearMonth, time

Field Identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the `Data Element Separation` property of the complex type or types in which the object is a child is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, this property must contain a non-empty value. If the object is a complex element, and the `Data Element Separation` property of its parent is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, the property can contain an empty value. The value for this property must be unique for every element in the message set; that is, no two elements in the message set can contain the same value for this property.
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the `Data Element Separation` method is set to `Use Data Pattern` in the complex type. For more details, see "Regular expression syntax" on page 6304.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the `Messaging Standard` property of the message set is `User Defined Text`, `User Defined Mixed`, `CSV`, or `TLOG`, select one of the following values: • `Text`. The data is in character format. • `Length Encoded String 1`. The first byte of the data contains the length (in length units) of the data string that follows the length byte. The maximum length of a `Length Encoded String 1` element is 255 length units. • `Length Encoded String 2`. The first two bytes of the data contain the length (in length units) of the data string that follows the two length bytes. The maximum length of a `Length Encoded String 2` element is 65535 length units. • `Null Terminated String`. The data string ends with the hexadecimal NULL character, `X'00'`. • `Packed Decimal`. The data is a packed decimal number that equates to the COMP-3 data type in COBOL. • `Binary`. The data is in bit string format. • `Time Seconds`. This value supports C time_t, and Java Date and Time objects. It is valid only if the `DateTime Format` property represents numeric-only data. • `Time Milliseconds`. This value supports C time_t, and Java Date and Time objects. It is valid only if the `DateTime Format` property represents numeric-only data. The default is dependent on the `Messaging Standard` property. For all other `Messaging Standard` values, the `Physical Type` property is set to `Text`.

Property	Type	Meaning
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if one of the following statements is true: • `Physical Type` is `Packed Decimal`. • `Physical Type` is `Text`, no `Length Reference` is specified, and the `Data Element Separation` of the parent complex type or group is `Fixed Length`, `Tagged Fixed Length`, or `Fixed Length AL3`. The default is dependent on the physical type of the object. If `Physical Type` is `Length Encoded String 1`, `Length Encoded String 2`, or `Null Terminated String`, this property is not applicable. If `Physical Type` is `Time Seconds`, the value of this property is 4, and cannot be changed. If `Physical Type` is `Time Milliseconds`, the value of this property is 8, and cannot be changed.
Length Units	Enumerated type	Select the unit of length for the object. Select one of the following options (some physical types do not offer both options): • `Bytes`. The length is given in bytes. • `Characters`. The length is given in characters. The number of bytes that are processed in the bit stream depends on the code page of the message. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the contents of the bit stream. The parser reads one character at a time and determines whether the character comprises one or more bytes. The default is dependent on the physical type of the object.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • `Not Applicable` • `Left Justify` • `Right Justify`

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the drop-down list. • Enter a character between quotation marks, for example "c" or 'c', where c is any alphanumeric character. • Enter a hexadecimal character code in the form `0xYY`, where `YY` is a hexadecimal value. • Enter a Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.
DateTime Format	String	Specify a template for date and time. The default DateTime format is dependent on the logical type of the object. For information about the defaults for the dateTime format according to the logical type, see “DateTime defaults by logical type” on page 6320. See “DateTime formats” on page 6310 for details of date and time formats.
Length Reference	Enumerated type	This property is applicable only if `Physical Type` is `Text`, `Binary`, or `TLOG Specific`. If set, this property takes precedence over any value in the `Length Units` property. Specify the identifier of a sibling integer object, the value of which determines the length of the object in question. The sibling object must be defined before the current object within the message structure. For information about reordering elements, see “Reordering objects” on page 2900.

Property	Type	Meaning
Inclusive Length Reference	Check box	This property is applicable only if `Length Reference` is set. If the check box is selected, the value of the sibling integer object that is identified by `Length Reference` is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by `Length Reference` is the length of the current object only. If the check box is selected, the `Length Units` property of the sibling integer object must be the same as that of the current object.

Numeric representation

Property	Type	Meaning
Signed	Check box	Specify whether the value is signed. This property is applicable only if the `Physical type` property is `Packed Decimal`. By default, this check box is cleared, which indicates that the value is not signed.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the list: • `NULLPadFill`. This option is valid only for fixed-length objects and is the default value. • `NULLLogicalValue`. The `Encoding Null Value` property is first converted to an actual value, and rendered in the way specified for the field. • `NULLLiteralValue`. This option specifies that `Encoding Null Value` contains a value that is directly substituted as if it is a string. For dateTime elements, use this option if you want to use the `Encoding Null Value` property to test or compare the content of the field in the message. • `NULLLiteralFill`. This option specifies that the field is filled with the value specified by the `Encoding Null Value` property. `Encoding Null Value` must resolve to a single character. The option that you select determines the value that you must set for the property `Encoding Null Value`. For full information about using these options, see “TDS Null handling options” on page 6293.
Encoding Null Value	String	The use of this property depends on the `Encoding Null` property. The default value is zero. If you set the `Encoding Null` property for a dateTime object to `NULLLogicalValue`, the value that you set must be in an ISO8601 dateTime format. These formats are described in “DateTime as string data” on page 6311. For example, specify a value that conforms to the `yyyy-MM-dd'T'HH:mm:ss` format; for example, `1970-12-01`.

Occurrences

Property	Type	Meaning
Repeat reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, no integer objects exist before this one in the message structure. If a `Repeat Reference` is specified, it overrides any setting for the `Max Occurs` logical property when parsing and writing the message, but not for validating the message.
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the `Data Element Separation` property of the parent group or complex type is set to `All Elements Delimited` or `Variable Length Elements Delimited`. A default value is provided if the previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; if the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used. If none of the previous conditions are true, a default is not applied.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

TDS properties for local element decimal types:

The TDS format properties for local element decimal types.

The TDS Format properties described here apply to:

- Objects: Local Element
- Decimal schema types: decimal, integer, negativeInteger, nonNegativeInteger, nonPositiveInteger, positiveInteger, unsignedLong

Field Identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the `Data Element Separation` property of the complex type or types in which the object is a child is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, this property must contain a non-empty value. If the object is a complex element, and the `Data Element Separation` property of its parent is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, the property can contain an empty value. The value for this property must be unique for every element in the message set; that is, no two elements in the message set can contain the same value for this property.
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the `Data Element Separation` method is set to `Use Data Pattern` in the complex type. For more details, see "Regular expression syntax" on page 6304.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the `Messaging Standard` property of the message set is `User Defined Text`, `User Defined Mixed`, `CSV`, or `TLOG`, select one of the following values: • `Text`. The data is in character format. • `Packed Decimal`. The data is a packed decimal number that equates to the COMP-3 data type in COBOL. • `TLOG Specific` - this option can be selected only if the `Message Standard` property of the message set is `TLOG`. This option indicates that the format of the data is specific to the TLOG messaging standard. • `Integer`. Equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • `External Decimal`. The data is a decimal number that equates to the data type PIC 9 USAGE DISPLAY in COBOL. The default is dependent on the `Messaging Standard` property. For all other `Messaging Standard` values, the `Physical Type` property is set to `Text`.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no `Length Reference` is specified. If this property is not set and the message set property `Derive default length from logical type` is selected, and the `Physical type` is 'Character', the default value is derived from any totalDigits value constraint (schema facet) or, if none, any `minInclusive`, `maxInclusive`, `minExclusive`, or `maxExclusive` value constraints (schema facets), on the simple type.
Length Units	Enumerated type	Select the unit of length for the object. Select one of the following options (some physical types do not offer both options): • `Bytes`. The length is given in bytes. • `Characters`. The length is given in characters. The number of bytes that are processed in the bit stream depends on the code page of the message. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the contents of the bit stream. The parser reads one character at a time and determines whether the character comprises one or more bytes. The default is dependent on the physical type of the object.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • `Not Applicable` • `Left Justify` • `Right Justify`

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the drop-down list. • Enter a character between quotation marks, for example "c" or 'c', where c is any alphanumeric character. • Enter a hexadecimal character code in the form `0xYY`, where `YY` is a hexadecimal value. • Enter a Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.
Length Reference	Enumerated type	This property is applicable only if `Physical Type` is `Text`, `Binary`, or `TLOG Specific`. If set, this property takes precedence over any value in the `Length Units` property. Specify the identifier of a sibling integer object, the value of which determines the length of the object in question. The sibling object must be defined before the current object within the message structure. For information about reordering elements, see “Reordering objects” on page 2900.

Property	Type	Meaning
Inclusive Length Reference	Check box	This property is applicable only if `Length Reference` is set. If the check box is selected, the value of the sibling integer object that is identified by `Length Reference` is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by `Length Reference` is the length of the current object only. If the check box is selected, the `Length Units` property of the sibling integer object must be the same as that of the current object.

Numeric representation

Property	Type	Meaning
Signed	Check box	Specify whether the value is signed.
Sign EBCDIC Custom Overpunched	Check box	Specify whether EBCDIC custom sign format is used. This property is applicable only if the `Signed` property is selected and the `Physical Type` property is set to `External Decimal`.
Sign Orientation	Enumerated type	The values that you can choose for this property are: • `Leading Separate` • `Trailing Separate` • `Leading Overpunched` • `Trailing Overpunched` This property is enabled only if you have set `Physical Type` to `Text` or `External Decimal`, and you have selected `Signed`. If `Physical Type` is `Text`, the only valid values of `Sign Orientation` are `Leading Separate` and `Trailing Separate`. If `Physical Type` is `External Decimal` and `Sign EBCDIC Customer Overpunched` is selected, the only valid values of `Sign Orientation` are `Leading Overpunched` and `Trailing Overpunched`.
Positive Sign	String	Specify the value that represents the positive symbol. Do not specify a numeric value. If no value is set, "+" is assumed. The positive sign is not written when creating an output message; it is used only to recognize the positive sign when parsing a message bit stream. This property is applicable only if `Physical Type` is `Text` and `Signed` is selected.
Negative Sign	String	Specify the value that represents the negative symbol. Do not specify a numeric value. If no value is set, "-" is assumed. This property is applicable only if `Physical Type` is `Text` and `Signed` is selected.
Virtual Decimal Point	Button and Integer	Specify a non-zero integer that represents the position of an implied decimal point within a number, or specify 0, the default, to use the formatting of Float or Decimal numbers as specified by the `Precision` property. If you specify a positive integer, the position of the decimal point is moved left from the right side of the number. For example, if you specify `3`, the decimal value `1234` represents 1.234 If you specify a negative integer, the position of the decimal point is moved right from the right side of the number. For example, if you specify `-3`, the decimal value `1234` represents 1,234,000.

Property	Type	Meaning
Precision	Button and Integer	This property is applicable only if `Physical Type` is `Text`. It is used if the value of the `Virtual Decimal Point` property is 0, which indicates that the decimal point is present in the data. It deals with truncation, and specifies how many digits are to follow the decimal point. Either specify a number of digits: • If you set `Precision` to 0, data is truncated so that the fractional part is lost. For example, the value `123.45` is truncated to `123`. • If you set `Precision` to a number less than the number of fractional digits, data is truncated. For example, the value `123.4567` is truncated to `123.45` if you set `Precision` to 2. • If you set `Precision` to a number greater than the number of fractional digits, the value is padded with extra zeros. For example, the value `12.345` is padded to `12.34500` if you set `Precision` to 5. Or select one option from the list: • `All Significant Digits - decimal separator only required if fractional digits` (the default): all significant digits are written to the output bit stream, and no decimal separator is written if no fractional digits are present. • `Explicit Decimal Separator - decimal separator always required`: all significant digits are written to the output bit stream and the decimal separator is always included, even when no fractional digits are present. The decimal separator must be present in the input bit stream, even when no fractional digits are present.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the list: • `NULLPadFill`. This option is valid only for fixed-length objects and is the default value. • `NULLLogicalValue`. The `Encoding Null Value` property is first converted to an actual value, and rendered in the way specified for the field. • `NULLLiteralValue`. This option specifies that `Encoding Null Value` contains a value that is directly substituted as if it is a string. For dateTime elements, use this option if you want to use the `Encoding Null Value` property to test or compare the content of the field in the message. • `NULLLiteralFill`. This option specifies that the field is filled with the value specified by the `Encoding Null Value` property. `Encoding Null Value` must resolve to a single character. The option that you select determines the value that you must set for the property `Encoding Null Value`. For full information about using these options, see “TDS Null handling options” on page 6293.
Encoding Null Value	String	The use of this property depends on the `Encoding Null` property. The default value is zero. If you set the `Encoding Null` property for a dateTime object to `NULLLogicalValue`, the value that you set must be in an ISO8601 dateTime format. These formats are described in “DateTime as string data” on page 6311. For example, specify a value that conforms to the `yyyy-MM-dd'T'HH:mm:ss` format; for example, `1970-12-01`.

Occurrences

Property	Type	Meaning
Repeat reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, no integer objects exist before this one in the message structure. If a `Repeat Reference` is specified, it overrides any setting for the `Max Occurs` logical property when parsing and writing the message, but not for validating the message.
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the `Data Element Separation` property of the parent group or complex type is set to `All Elements Delimited` or `Variable Length Elements Delimited`. A default value is provided if the previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; if the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used. If none of the previous conditions are true, a default is not applied.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

TDS properties for local element float types:

The TDS format properties for local element float types.

The TDS Format properties described here apply to:

- Objects: Local Element
- Float schema types: double, float

Field Identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the `Data Element Separation` property of the complex type or types in which the object is a child is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, this property must contain a non-empty value. If the object is a complex element, and the `Data Element Separation` property of its parent is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, the property can contain an empty value. The value for this property must be unique for every element in the message set; that is, no two elements in the message set can contain the same value for this property.
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the `Data Element Separation` method is set to `Use Data Pattern` in the complex type. For more details, see "Regular expression syntax" on page 6304.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the `Messaging Standard` property of the message set is `User Defined Text`, `User Defined Mixed`, `CSV`, or `TLOG`, select one of the following values: • `Text`. The data is in character format. • `Float`. Equates to the data type FLOAT or DOUBLE in C, or the COMP-1 or COMP-2 numeric data type in COBOL. • `Integer`. Equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • `Packed Decimal`. The data is a packed decimal number that equates to the COMP-3 data type in COBOL. • `External Decimal`. The data is a decimal number that equates to the data type PIC 9 USAGE DISPLAY in COBOL. The default is dependent on the `Messaging Standard` property. For all other `Messaging Standard` values, the `Physical Type` property is set to `Text`.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no `Length Reference` is specified.
Length Units	Enumerated type	Select the unit of length for the object. Select one of the following options (some physical types do not offer both options): • `Bytes`. The length is given in bytes. • `Characters`. The length is given in characters. The number of bytes that are processed in the bit stream depends on the code page of the message. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the contents of the bit stream. The parser reads one character at a time and determines whether the character comprises one or more bytes. The default is dependent on the physical type of the object.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • `Not Applicable` • `Left Justify` • `Right Justify`

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the drop-down list. • Enter a character between quotation marks, for example "c" or 'c', where c is any alphanumeric character. • Enter a hexadecimal character code in the form `0xYY`, where `YY` is a hexadecimal value. • Enter a Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.
Length Reference	Enumerated type	This property is applicable only if `Physical Type` is `Text`, `Binary`, or `TLOG Specific`. If set, this property takes precedence over any value in the `Length Units` property. Specify the identifier of a sibling integer object, the value of which determines the length of the object in question. The sibling object must be defined before the current object within the message structure. For information about reordering elements, see “Reordering objects” on page 2900.

Property	Type	Meaning
Inclusive Length Reference	Check box	This property is applicable only if `Length Reference` is set. If the check box is selected, the value of the sibling integer object that is identified by `Length Reference` is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by `Length Reference` is the length of the current object only. If the check box is selected, the `Length Units` property of the sibling integer object must be the same as that of the current object.

Numeric representation

Property	Type	Meaning
Signed	Check box	Specify whether the value is signed.
Sign EBCDIC Custom Overpunched	Check box	Specify whether EBCDIC custom sign format is used. This property is applicable only if the `Signed` property is selected and the `Physical Type` property is set to `External Decimal`.
Sign Orientation	Enumerated type	The values that you can choose for this property are: • `Leading Separate` • `Trailing Separate` • `Leading Overpunched` • `Trailing Overpunched` This property is enabled only if you have set `Physical Type` to `Text` or `External Decimal`, and you have selected `Signed`. If `Physical Type` is `Text`, the only valid values of `Sign Orientation` are `Leading Separate` and `Trailing Separate`. If `Physical Type` is `External Decimal` and `Sign EBCDIC Customer Overpunched` is selected, the only valid values of `Sign Orientation` are `Leading Overpunched` and `Trailing Overpunched`.
Positive Sign	String	Specify the value that represents the positive symbol. Do not specify a numeric value. If no value is set, "+" is assumed. The positive sign is not written when creating an output message; it is used only to recognize the positive sign when parsing a message bit stream. This property is applicable only if `Physical Type` is `Text` and `Signed` is selected.
Negative Sign	String	Specify the value that represents the negative symbol. Do not specify a numeric value. If no value is set, "-" is assumed. This property is applicable only if `Physical Type` is `Text` and `Signed` is selected.
Virtual Decimal Point	Button and Integer	Specify a non-zero integer that represents the position of an implied decimal point within a number, or specify 0, the default, to use the formatting of Float or Decimal numbers as specified by the `Precision` property. If you specify a positive integer, the position of the decimal point is moved left from the right side of the number. For example, if you specify `3`, the decimal value `1234` represents 1.234 If you specify a negative integer, the position of the decimal point is moved right from the right side of the number. For example, if you specify `-3`, the decimal value `1234` represents 1,234,000.

Property	Type	Meaning
Precision	Button and Integer	This property is applicable only if `Physical Type` is `Text`. It is used if the value of the `Virtual Decimal Point` property is 0, which indicates that the decimal point is present in the data. It deals with truncation, and specifies how many digits are to follow the decimal point. Either specify a number of digits: • If you set `Precision` to 0, data is truncated so that the fractional part is lost. For example, the value `123.45` is truncated to `123`. • If you set `Precision` to a number less than the number of fractional digits, data is truncated. For example, the value `123.4567` is truncated to `123.45` if you set `Precision` to 2. • If you set `Precision` to a number greater than the number of fractional digits, the value is padded with extra zeros. For example, the value `12.345` is padded to `12.34500` if you set `Precision` to 5. Or select one option from the list: • `All Significant Digits - decimal separator only required if fractional digits` (the default): all significant digits are written to the output bit stream, and no decimal separator is written if no fractional digits are present. • `Explicit Decimal Separator - decimal separator always required`: all significant digits are written to the output bit stream and the decimal separator is always included, even when no fractional digits are present. The decimal separator must be present in the input bit stream, even when no fractional digits are present. • `Exponential Notation - Example "1.23456e002"`: data is written out to the bit stream as a signed value having the format `[sign1]a.bbbe[sign2]ccc` where: – `[sign1]` is the value of `Negative Sign` if the value is negative – `a` is a single decimal digit – `bbb` is one or more decimal digits – `[sign2]` is the value of `Negative Sign` if the exponent is negative – `ccc` is exactly three decimal digits (the exponent) `[sign1]` and `[sign2]` are absent if the value and exponent are positive. For example, the value -123.456 is represented as `-1.23456e002` and the value 0.00012 is represented as `1.2e-004` in the output bit stream, assuming that the value of `Negative Sign` is "-", and the value of `Sign Orientation` is `Leading`. The value -0.00012 is represented as `1.2*e*004` if `Negative Sign` is "*" and `Sign Orientation` is `Trailing`.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the list: • `NULLPadFill`. This option is valid only for fixed-length objects and is the default value. • `NULLLogicalValue`. The `Encoding Null Value` property is first converted to an actual value, and rendered in the way specified for the field. • `NULLLiteralValue`. This option specifies that `Encoding Null Value` contains a value that is directly substituted as if it is a string. For dateTime elements, use this option if you want to use the `Encoding Null Value` property to test or compare the content of the field in the message. • `NULLLiteralFill`. This option specifies that the field is filled with the value specified by the `Encoding Null Value` property. `Encoding Null Value` must resolve to a single character. The option that you select determines the value that you must set for the property `Encoding Null Value`. For full information about using these options, see "TDS Null handling options" on page 6293.
Encoding Null Value	String	The use of this property depends on the `Encoding Null` property. The default value is zero. If you set the `Encoding Null` property for a dateTime object to `NULLLogicalValue`, the value that you set must be in an ISO8601 dateTime format. These formats are described in "DateTime as string data" on page 6311. For example, specify a value that conforms to the `yyyy-MM-dd'T'HH:mm:ss` format; for example, `1970-12-01`.

Occurrences

Property	Type	Meaning
Repeat reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, no integer objects exist before this one in the message structure. If a `Repeat Reference` is specified, it overrides any setting for the `Max Occurs` logical property when parsing and writing the message, but not for validating the message.
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the `Data Element Separation` property of the parent group or complex type is set to `All Elements Delimited` or `Variable Length Elements Delimited`. A default value is provided if the previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; if the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used. If none of the previous conditions are true, a default is not applied.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

TDS properties for local element integer types:

The TDS format properties for local element integer types.

The TDS Format properties described here apply to:

- Objects: Local Element
- Integer schema types: byte, int, long, short, unsignedByte, unsignedInt, unsignedShort

Field Identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the `Data Element Separation` property of the complex type or types in which the object is a child is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, this property must contain a non-empty value. If the object is a complex element, and the `Data Element Separation` property of its parent is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, the property can contain an empty value. The value for this property must be unique for every element in the message set; that is, no two elements in the message set can contain the same value for this property.
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the `Data Element Separation` method is set to `Use Data Pattern` in the complex type. For more details, see "Regular expression syntax" on page 6304.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the `Messaging Standard` property of the message set is `User Defined Text`, `User Defined Mixed`, `CSV`, or `TLOG`, select one of the following values: • `Text`. The data is in character format. • `Integer`. Equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • `Packed Decimal`. The data is a packed decimal number that equates to the COMP-3 data type in COBOL. • `External Decimal`. The data is a decimal number that equates to the data type PIC 9 USAGE DISPLAY in COBOL. The default is dependent on the `Messaging Standard` property. For all other `Messaging Standard` values, the `Physical Type` property is set to `Text`.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no `Length Reference` is specified. If this property is not set and the message set property `Derive default length from logical type` is selected, and the `Physical type` is 'Character', the default value is derived from any totalDigits value constraint (schema facet) or, if none, any `minInclusive`, `maxInclusive`, `minExclusive`, or `maxExclusive` value constraints (schema facets), on the simple type.

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the object. Select one of the following options (some physical types do not offer both options): • `Bytes`. The length is given in bytes. • `Characters`. The length is given in characters. The number of bytes that are processed in the bit stream depends on the code page of the message. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the contents of the bit stream. The parser reads one character at a time and determines whether the character comprises one or more bytes. The default is dependent on the physical type of the object.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • `Not Applicable` • `Left Justify` • `Right Justify`

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the drop-down list. • Enter a character between quotation marks, for example "c" or 'c', where c is any alphanumeric character. • Enter a hexadecimal character code in the form `0xYY`, where `YY` is a hexadecimal value. • Enter a Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.
Length Reference	Enumerated type	This property is applicable only if `Physical Type` is `Text`, `Binary`, or `TLOG Specific`. If set, this property takes precedence over any value in the `Length Units` property. Specify the identifier of a sibling integer object, the value of which determines the length of the object in question. The sibling object must be defined before the current object within the message structure. For information about reordering elements, see “Reordering objects” on page 2900.

Property	Type	Meaning
Inclusive Length Reference	Check box	This property is applicable only if `Length Reference` is set. If the check box is selected, the value of the sibling integer object that is identified by `Length Reference` is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by `Length Reference` is the length of the current object only. If the check box is selected, the `Length Units` property of the sibling integer object must be the same as that of the current object.

Numeric representation

Property	Type	Meaning
Signed	Check box	Specify whether the value is signed.
Sign EBCDIC Custom Overpunched	Check box	Specify whether EBCDIC custom sign format is used. This property is applicable only if the `Signed` property is selected and the `Physical Type` property is set to `External Decimal`.
Sign Orientation	Enumerated type	The values that you can choose for this property are: • `Leading Separate` • `Trailing Separate` • `Leading Overpunched` • `Trailing Overpunched` This property is enabled only if you have set `Physical Type` to `Text` or `External Decimal`, and you have selected `Signed`. If `Physical Type` is `Text`, the only valid values of `Sign Orientation` are `Leading Separate` and `Trailing Separate`. If `Physical Type` is `External Decimal` and `Sign EBCDIC Customer Overpunched` is selected, the only valid values of `Sign Orientation` are `Leading Overpunched` and `Trailing Overpunched`.
Positive Sign	String	Specify the value that represents the positive symbol. Do not specify a numeric value. If no value is set, "+" is assumed. The positive sign is not written when creating an output message; it is used only to recognize the positive sign when parsing a message bit stream. This property is applicable only if `Physical Type` is `Text` and `Signed` is selected.
Negative Sign	String	Specify the value that represents the negative symbol. Do not specify a numeric value. If no value is set, "-" is assumed. This property is applicable only if `Physical Type` is `Text` and `Signed` is selected.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the list: • `NULLPadFill`. This option is valid only for fixed-length objects and is the default value. • `NULLLogicalValue`. The `Encoding Null Value` property is first converted to an actual value, and rendered in the way specified for the field. • `NULLLiteralValue`. This option specifies that `Encoding Null Value` contains a value that is directly substituted as if it is a string. For dateTime elements, use this option if you want to use the `Encoding Null Value` property to test or compare the content of the field in the message. • `NULLLiteralFill`. This option specifies that the field is filled with the value specified by the `Encoding Null Value` property. `Encoding Null Value` must resolve to a single character. The option that you select determines the value that you must set for the property `Encoding Null Value`. For full information about using these options, see "TDS Null handling options" on page 6293.
Encoding Null Value	String	The use of this property depends on the `Encoding Null` property. The default value is zero. If you set the `Encoding Null` property for a dateTime object to `NULLLogicalValue`, the value that you set must be in an ISO8601 dateTime format. These formats are described in "DateTime as string data" on page 6311. For example, specify a value that conforms to the `yyyy-MM-dd'T'HH:mm:ss` format; for example, `1970-12-01`.

Occurrences

Property	Type	Meaning
Repeat reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, no integer objects exist before this one in the message structure. If a `Repeat Reference` is specified, it overrides any setting for the `Max Occurs` logical property when parsing and writing the message, but not for validating the message.
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the `Data Element Separation` property of the parent group or complex type is set to `All Elements Delimited` or `Variable Length Elements Delimited`. A default value is provided if the previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; if the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used. If none of the previous conditions are true, a default is not applied.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

TDS properties for local element interval types:

The TDS format properties for local element string types.

The TDS Format properties described here apply to:

- Objects: Local Element
- Interval schema types: duration

Field Identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the `Data Element Separation` property of the complex type or types in which the object is a child is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, this property must contain a non-empty value. If the object is a complex element, and the `Data Element Separation` property of its parent is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, the property can contain an empty value. The value for this property must be unique for every element in the message set; that is, no two elements in the message set can contain the same value for this property.
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the `Data Element Separation` method is set to `Use Data Pattern` in the complex type. For more details, see "Regular expression syntax" on page 6304.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the `Messaging Standard` property of the message set is `User Defined Text`, `User Defined Mixed`, `CSV`, or `TLOG`, select one of the following values: • `Text`. The data is in character format. • `Length Encoded String 1`. The first byte of the data contains the length (in length units) of the data string that follows the length byte. The maximum length of a `Length Encoded String 1` element is 255 length units. • `Length Encoded String 2`. The data's first 2 bytes contains the length (in length units) of the data string that follows the two length bytes. The maximum length of a `Length Encoded String 2` element is 65535 length units. • `Null Terminated String`. The data string ends with the hexadecimal NULL character, `X'00'`. The default is dependent on the `Messaging Standard` property. For all other `Messaging Standard` values, the `Physical Type` property is set to `Text`.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no `Length Reference` is specified. If this property is not set and the message set property `Derive default length from logical type` is selected, and the `Physical type` is 'Character', the default value is derived from any length or maxLength value constraint (schema facet) on the simple type.

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the object. Select one of the following options (some physical types do not offer both options): • `Bytes`. The length is given in bytes. • `Characters`. The length is given in characters. The number of bytes that are processed in the bit stream depends on the code page of the message. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the contents of the bit stream. The parser reads one character at a time and determines whether the character comprises one or more bytes. The default is dependent on the physical type of the object.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • `Not Applicable` • `Left Justify` • `Right Justify`

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the drop-down list. • Enter a character between quotation marks, for example "c" or 'c', where c is any alphanumeric character. • Enter a hexadecimal character code in the form `0xYY`, where `YY` is a hexadecimal value. • Enter a Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.
Length Reference	Enumerated type	This property is applicable only if `Physical Type` is `Text`, `Binary`, or `TLOG Specific`. If set, this property takes precedence over any value in the `Length Units` property. Specify the identifier of a sibling integer object, the value of which determines the length of the object in question. The sibling object must be defined before the current object within the message structure. For information about reordering elements, see “Reordering objects” on page 2900.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the list: • `NULLPadFill`. This option is valid only for fixed-length objects and is the default value. • `NULLLogicalValue`. The `Encoding Null Value` property is first converted to an actual value, and rendered in the way specified for the field. • `NULLLiteralValue`. This option specifies that `Encoding Null Value` contains a value that is directly substituted as if it is a string. For dateTime elements, use this option if you want to use the `Encoding Null Value` property to test or compare the content of the field in the message. • `NULLLiteralFill`. This option specifies that the field is filled with the value specified by the `Encoding Null Value` property. `Encoding Null Value` must resolve to a single character. The option that you select determines the value that you must set for the property `Encoding Null Value`. For full information about using these options, see "TDS Null handling options" on page 6293.
Encoding Null Value	String	The use of this property depends on the `Encoding Null` property. The default value is zero. If you set the `Encoding Null` property for a dateTime object to `NULLLogicalValue`, the value that you set must be in an ISO8601 dateTime format. These formats are described in "DateTime as string data" on page 6311. For example, specify a value that conforms to the `yyyy-MM-dd'T'HH:mm:ss` format; for example, `1970-12-01`.

Occurrences

Property	Type	Meaning
Repeat reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, no integer objects exist before this one in the message structure. If a `Repeat Reference` is specified, it overrides any setting for the `Max Occurs` logical property when parsing and writing the message, but not for validating the message.
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the `Data Element Separation` property of the parent group or complex type is set to `All Elements Delimited` or `Variable Length Elements Delimited`. A default value is provided if the previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; if the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used. If none of the previous conditions are true, a default is not applied.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

TDS properties for local element string types:

The TDS format properties for local element string types.

The TDS Format properties described here apply to:

- Objects: Local Element
- String schema types: anyURI, ENTITIES, ENTITY, ID, IDREF, IDREFS, language, Name, NCName, NMTOKEN, NMTOKENS, normalizedString, NOTATION, QName, string, token

Field Identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the `Data Element Separation` property of the complex type or types in which the object is a child is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, this property must contain a non-empty value. If the object is a complex element, and the `Data Element Separation` property of its parent is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, the property can contain an empty value. The value for this property must be unique for every element in the message set; that is, no two elements in the message set can contain the same value for this property.
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the `Data Element Separation` method is set to `Use Data Pattern` in the complex type. For more details, see "Regular expression syntax" on page 6304.
Interpret Element Value	Enumerated type	Specify whether values stored within this object are interpreted as having significance for the parser and, if so, the type of interpretation that occurs. This interpretation is standard-specific and is therefore hard coded. The possible values for this property are: • `None` (the default value) • `EDIFACT Service String` • `X12 Service String` • `Message Key` • `EDIFACT Syntax Level ID` • `HL7 Service String` • `HL7 Field Separator` **Note:** The `Message Key` enumeration has been deprecated.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the `Messaging Standard` property of the message set is `User Defined Text`, `User Defined Mixed`, `CSV`, or `TLOG`, select one of the following values: • `Text`. The data is in character format. • `Length Encoded String 1`. The first byte of the data contains the length (in length units) of the data string that follows the length byte. The maximum length of a `Length Encoded String 1` element is 255 length units. • `Length Encoded String 2`. The first two bytes of the data contain the length (in length units) of the data string that follows the two length bytes. The maximum length of a `Length Encoded String 2` element is 65535 length units. • `Null Terminated String`. The data string ends with the hexadecimal NULL character, `X'00'`. • `TLOG Specific` - this option can be selected only if the `Message Standard` property of the message set is `TLOG`. This option indicates that the format of the data is specific to the TLOG messaging standard. The default is dependent on the `Messaging Standard` property. For all other `Messaging Standard` values, the `Physical Type` property is set to `Text`.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no `Length Reference` is specified. If this property is not set and the message set property `Derive default length from logical type` is selected, and the `Physical type` is 'Character', the default value is derived from any length or maxLength value constraint (schema facet) on the simple type.
Length Units	Enumerated type	Select the unit of length for the object. Select one of the following options (some physical types do not offer both options): • `Bytes`. The length is given in bytes. • `Characters`. The length is given in characters. The number of bytes that are processed in the bit stream depends on the code page of the message. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the contents of the bit stream. The parser reads one character at a time and determines whether the character comprises one or more bytes. The default is dependent on the physical type of the object.

Property	Type	Meaning
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • `Not Applicable` • `Left Justify` • `Right Justify`
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the drop-down list. • Enter a character between quotation marks, for example "c" or 'c', where c is any alphanumeric character. • Enter a hexadecimal character code in the form `0xYY`, where `YY` is a hexadecimal value. • Enter a Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.

Property	Type	Meaning
Length Reference	Enumerated type	This property is applicable only if `Physical Type` is `Text`, `Binary`, or `TLOG Specific`. If set, this property takes precedence over any value in the `Length Units` property. Specify the identifier of a sibling integer object, the value of which determines the length of the object in question. The sibling object must be defined before the current object within the message structure. For information about reordering elements, see "Reordering objects" on page 2900.
Inclusive Length Reference	Check box	This property is applicable only if `Length Reference` is set. If the check box is selected, the value of the sibling integer object that is identified by `Length Reference` is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by `Length Reference` is the length of the current object only. If the check box is selected, the `Length Units` property of the sibling integer object must be the same as that of the current object.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the list: • `NULLPadFill`. This option is valid only for fixed-length objects and is the default value. • `NULLLogicalValue`. The `Encoding Null Value` property is first converted to an actual value, and rendered in the way specified for the field. • `NULLLiteralValue`. This option specifies that `Encoding Null Value` contains a value that is directly substituted as if it is a string. For dateTime elements, use this option if you want to use the `Encoding Null Value` property to test or compare the content of the field in the message. • `NULLLiteralFill`. This option specifies that the field is filled with the value specified by the `Encoding Null Value` property. `Encoding Null Value` must resolve to a single character. The option that you select determines the value that you must set for the property `Encoding Null Value`. For full information about using these options, see "TDS Null handling options" on page 6293.
Encoding Null Value	String	The use of this property depends on the `Encoding Null` property. The default value is zero. If you set the `Encoding Null` property for a dateTime object to `NULLLogicalValue`, the value that you set must be in an ISO8601 dateTime format. These formats are described in "DateTime as string data" on page 6311. For example, specify a value that conforms to the `yyyy-MM-dd'T'HH:mm:ss` format; for example, `1970-12-01`.

Occurrences

Property	Type	Meaning
Repeat reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, no integer objects exist before this one in the message structure. If a `Repeat Reference` is specified, it overrides any setting for the `Max Occurs` logical property when parsing and writing the message, but not for validating the message.
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the `Data Element Separation` property of the parent group or complex type is set to `All Elements Delimited` or `Variable Length Elements Delimited`. A default value is provided if the previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; if the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used. If none of the previous conditions are true, a default is not applied.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

Documentation properties for all message set objects:

Use the documentation property of an object to add information that describes the function of the object.

The documentation property is available on all objects except Key, Keyref, and Unique objects.

The property is a string field, and you can use any standard alphanumeric characters.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

Local group properties:

Different types of properties are available for a local group.

A local group can have the following properties;

- "Local group logical properties" on page 5446
- "Local group CWF properties" on page 5471
- "Local group XML properties" on page 5494
- "Local group TDS properties" on page 5526
- "Documentation properties for all message set objects" on page 5413

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

Local group logical properties:

The logical properties of a local group include properties that specify the number of occurrences of the local group.

Valid children in a local group that depend on both **`Composition`** and **`Content Validation`** are shown in "MRM content validation" on page 5422.

Property	Type	Meaning
Composition	Enumerated type	Define the order, and the number of occurrences, of the elements and groups in your messages. Composition does not affect the attributes in a complex type. Select from the following options: • Empty • sequence. If you select this option, you can define members that are elements or groups. These members, if present, must appear in the specified order in the message. They can repeat, and the same element or group can appear more than once. • choice. If you select this option, you can define members that are elements or groups. Exactly one of the defined members must be present in the message, and can repeat. Use this option if you want to model C unions and COBOL REDEFINES in a Custom Wire Format, or an XML DTD element that uses choice in an XML Wire Format, or a SWIFT field that has more than one option. • all. If you select this option, you can define members that are elements; groups are not allowed. The elements in an all group can appear in any order. Each element can appear once, or not at all. An all group can be used only at the top level of a complex type; it cannot be a member of another group within a type. • unorderedSet. This option is supported only by the MRM domain.If you select this option, you can define members that are elements. The elements can repeat, but the same element cannot appear twice in the list of members. The elements can appear in any order in the message. • orderedSet. This option is supported only by the MRM domain.If you select this option, you can define members that are elements. The elements can repeat, but the same element cannot appear twice in the list of members. The elements must appear in the specified order in the message. • message. This option is supported only by the MRM domain. If you select this option, you can define only messages as members. Each member can repeat, but the same message cannot appear twice in the list of members. Like choice, only one of the defined members can be present in a message. Unlike choice, when writing a message, if the complex type or group has more than one member, the bit stream is not padded to the length of the longest member. Use this option to model multipart messages, which are used in some industry standards; for example, SWIFT. For more information, see the section on multipart messages in “Multipart messages” on page 1191.

Property	Type	Meaning
Content Validation	Enumerated type	`Content Validation` is used only by the MRM domain. If validation is enabled in your message flow, `Content Validation` specifies the strictness of the MRM validation for members of a complex type or group. See "MRM content validation" on page 5422 for further details. Select from the following options: • Closed. The complex type can only contain the child elements that you have added to it. • Open Defined. The complex type can contain any valid element defined within the message set. • Open. The complex type can contain any valid element, not just those that you have added to this complex type. See "Combinations of Composition and Content Validation" on page 5612 for further details of these options.

Occurrences

Property	Type	Meaning
Min Occurs	Integer	Specify the minimum number of times that the object can repeat. The default value is 1. If the value is set to 0, the object is optional. With the exception of `Max Occurs` being set to -1, if a value is set for `Min Occurs`, it must be less than or equal to the value in `Max Occurs`.
Max Occurs	Integer	Specify the maximum number of times that the object can repeat. The default value is 1. If this property is not set, the object cannot occur more than once. If this property is set to 0, it is interpreted as if the object does not exist in the message. It can also be set to -1, to indicate that the limit is unbounded and there is no maximum to the number of occurrences.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Local group CWF properties:

The CWF properties of a local group are described in the following tables.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to all instances.

Occurrences

Property	Type	Meaning
Repeat Reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the displayed list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, there are no integer objects before this one in the message structure. If a *`Repeat Reference`* is specified, it overrides any setting for the *`Max Occurs`* logical property when parsing and writing the message, but not for validation of the message.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM Custom Wire Format" on page 1214
Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not separated by delimiters.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Local group XML properties:

There are no properties to show.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Local group TDS properties:

TDS properties of a local group.

Field Identification

Property	Type	Meaning
Data Element Separation	Enumerated type	Select one of the following values to specify the method that is used to separate the data elements within the type. • `Tagged Delimited`. This value indicates that all elements within the complex type are identified by a tag, and, if a value is specified in the optional `Delimiter` property, are separated by that value. You must set the `Tag` property for all child elements of simple type, and you can set the `Delimiter` property to a non-empty value. See "Global element TDS properties" on page 5513. You must also set the `Tag Data Separator` or the `Length of Tag` property. • `Tagged Fixed Length`. This value indicates that each element is identified by a tag, and the data has a fixed length. There are no delimiters. You must set the `Tag` property for each of the child elements of this complex type, and each child element must have a `Length` or `Length Reference` property assigned to it. You must also set the `Tag Data Separator` or the `Length of Tag` property. • `Tagged Encoded Length`. This value indicates that all elements within the complex type are separated by a tag, and a length field follows each tag. There are no delimiters. The tag can be fixed length, as set by the `Length of Tag` property, or variable length delimited by the `Tag Data Separator` property. You must also set the `Length Of Encoded Length` property so that the parser knows the size of the length field, and set the `Extra Chars in Encoded Length` property. This property tells the parser what to subtract from the value in the `Length Of Encoded Length` property to get the actual length of the data that follows the length field. This method provides a more flexible way of handling ACORD AL3 standard messages than using the `Fixed Length AL3` value, by allowing different parts of the messages to be at different versions of the ACORD AL3 standard. • `All Elements Delimited`. This value indicates that all elements within the complex type are separated by a delimiter. You must set a value in the `Delimiter` property. • `Variable Length Elements Delimited`. This value indicates that some of the elements within the complex type might be of variable length. Variable length elements must be delimited by the value specified in the `Delimiter` property. • `Use Data Pattern`. This value indicates that the parser determines the elements by matching the data with the regular expression that is set in the `Data Pattern` property of the element or type member. See "Message definition file properties" on page 5409. • `Fixed Length`. This value indicates that all elements within the complex type are fixed length. The next data element is accessed by adding the value of the `Length` property to the offset. See "Global element TDS properties" on page 5513. If you set the `Data Element Separation` property of a complex type to `Fixed Length`, you must also set the `Data Element Separation` property of all complex children of this type to `Fixed Length`. Each child element must have a `Length` or `Length Reference` property assigned to it. • `Fixed Length AL3`. This value has a similar meaning to the separation type `Fixed Length`, but also indicates to the parser that a number of predefined rules regarding missing optional elements, encoded lengths, and versioning, must be applied. If you set the `Data Element Separation` property of a complex type to `Fixed Length AL3`, you must also set the `Data Element Separation` property of all complex children of this type to `Fixed Length AL3`. • `Undefined`. This value is set automatically if you set the `Type Composition` property of a complex type to `Message`, and you cannot change it to any other value. Do not set the `Type Composition` property to `Empty`, `Choice`, `Unordered Set`, `Ordered Set`, `Sequence`, or `Simple Unordered Set`. If you do, you cannot check in the type.

Property	Type	Meaning
Group Indicator	String	Specify the value of a special character, or string, that precedes the data that belongs to a group, or a complex type, within the bit stream.
Group Terminator	String	Specify the value of a special character, or string, that terminates the data that belongs to a group, or a complex type, within the bit stream.
Delimiter	String	Specify the value of a special character, or string, that specifies the delimiter that is used between data elements. This property applies only to the delimited `Data Element Separation` methods (`Tagged Delimited`, `All Elements Delimited`, and `Variable Length Elements Delimited`).
Suppress Absent Element Delimiters	Enumerated type	Use this property to select whether you want delimiters to be suppressed for elements that are missing within a message. Select from: • `End Of Type`. Use this option to suppress the delimiter when an element is missing. For example, if the model has been defined to have up to three elements and only two are present, the last delimiter can be omitted from the message. • `Never`. Use this option to ensure that even if optional elements are not present, all delimiters are written out. Use this option when the same delimiter is used to delimit parent and child objects. For example, if an optional child element is missing, message processing applications cannot tell where the child elements in a message end and the next parent element starts, if the delimiters are all the same.
Observe Element Length	Check box	This property is applicable when `Data Element Separation` is `All Elements Delimited` or `Tagged Delimited`. Select this check box if the `Length` property of child simple elements is significant when parsing and writing. • During parsing, an exception is thrown if the length of the extracted data exceeds the specified length. Otherwise, the data is trimmed according to the `Justification` and `Padding Character` properties of the child element. • During writing, an exception is thrown if the data to write exceeds the specified length. Otherwise, the data is padded according to the `Justification` and `Padding Character` properties of the child element. Clear this check box to ignore the `Length` property when parsing and writing. The default value depends on the setting of the `Messaging Standard` property (at the message set level) and the `Data Element Separation` property. • If `Data Element Separation` is `All Elements Delimited` and the `Messaging Standard` is `TLOG`, the check box is selected. • If `Data Element Separation` is `All Elements Delimited` and the `Messaging Standard` is other than `TLOG`, the check box is cleared. • If `Data Element Separation` is `Tagged Delimited`, the check box is cleared. For all other data element separation methods, the check box is disabled and does not influence the behavior of the TDS parser.
Tag Data Separator	Button and String	Specify the value of a special character or string that separates the Tag from the data. The `Tag Data Separator` and `Length of Tag` properties are mutually exclusive. If you set the property `Tag Data Separator`, it overrides `Length of Tag`. This property applies only to the tagged `Data Element Separation` methods (`Tagged Delimited`, `Tagged Fixed Length`, and `Tagged Encoded Length`).

Property	Type	Meaning
Length of Tag	Button and Integer	Specify the length of a tag value. When the message is parsed, this property allows tags to be extracted from the bit stream if the `Tag Data Separator` property is not set. The `Tag Data Separator` and `Length of Tag` properties are mutually exclusive. If you set the property `Tag Data Separator`, it overrides this value. This property applies only to the tagged `Data Element Separation` methods (`Tagged Delimited`, `Tagged Fixed Length`, and `Tagged Encoded Length`).
Length of Encoded Length	Integer	Specifies the number of characters (not bytes) after a tag that are used for the length field. Enter a value from 0 to 2147483647. You must set this property if you have set the `Data Element Separation` property to `Tagged Encoded Length`; it is not valid otherwise. The actual number of data characters that are parsed depends on the value of the `Extra Chars in Encoded Length` property.
Extra Chars in Encoded Length	Integer	(Only valid if the `Data Element Separation` method is set to `Tagged Encoded Length`.) Specifies the number of extra characters included in the value found in the length field. (For example, the value in the length might include the size of the length field itself as well as the size of the data field, or it might be the total size of the tag, length, and data fields.) Enter a value from 0 to 2147483647. The parser subtracts this number from the number found in the length field to get the number of data characters that follow the length field. You must set this property if you have set the `Data Element Separation` property to `Tagged Encoded Length`, and the actual number of data characters is less than the value found in the length field.

Field Identification

Property	Type	Meaning
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the `Data Element Separation` method is set to `Use Data Pattern` in the complex type. For more details, see “Regular expression syntax” on page 6304.

Occurrences

Property	Type	Meaning
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the `Data Element Separation` property of the parent group or complex type is set to `All Elements Delimited` or `Variable Length Elements Delimited`. A default value is provided if the previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; if the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used. If none of the previous conditions are true, a default is not applied.

Property	Type	Meaning
Repeat reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, no integer objects exist before this one in the message structure. If a `Repeat Reference` is specified, it overrides any setting for the `Max Occurs` logical property when parsing and writing the message, but not for validating the message.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM TDS format" on page 1221
The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Documentation properties for all message set objects:

Use the documentation property of an object to add information that describes the function of the object.

The documentation property is available on all objects except Key, Keyref, and Unique objects.

The property is a string field, and you can use any standard alphanumeric characters.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

Message properties:

Different types of properties are available for a message.

A message can have the following properties;

- "Message logical properties" on page 5449
- "Message CWF properties" on page 5473
- "Message XML properties" on page 5495
- "Message TDS properties" on page 5531
- "Documentation properties for all message set objects" on page 5413

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

Message logical properties:

This section describes the logical properties of a message.

Property	Type	Meaning
Name	String	Specify a name for the object when you create it. Names can consist of alphanumeric characters, including the letters *A* through *Z*, *a* through *z*, and the digits *0* through *9*. They might also include the following punctuation characters; • - the hyphen • _ the underscore • . the period Names can start only with a letter or the underscore character, and not with a number, hyphen, or period. Names that begin with `xml`, or any variant of these characters (for example `XmL`), are reserved by the XML standards specification. Further details of naming conventions and allowable characters can be found in the Extensible Markup Language (XML) specification that can be found on the World Wide Web Consortium (W3C) Web site.
Message Alias	String	Specify an alternative unique value that identifies the message. This property is only required if you are using the MRM domain and the Message Identity technique to identify embedded messages, and the bit stream does not contain the actual message name.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Message CWF properties:

There are no properties to show.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM Custom Wire Format" on page 1214
Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not separated by delimiters.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Message XML properties:

The following tables describe the XML properties of a message.

Namespace schema locations

This property is only active if namespaces have been enabled.

Property	Type	Meaning
Namespace URI	String	A unique string, usually in the form of a URL that identifies the schema. If namespaces have not been enabled, this property displays `<no target namespace>`. This property overrides the same property at the message set level.

Property	Type	Meaning
Schema location	String	Enter the location of the schema for the associated namespace name to be used to validate objects in the namespace.

XML declarations

Property	Type	Meaning
Output Namespace Declaration	Enumerated type	The `Output Namespace Declaration` property controls where the namespace declarations are placed in the output XML document. Select from: • `At start of document`. Declarations for all of the entries in the `Namespace schema locations` table above are produced as attributes of the message in the output XML document. The disadvantage of this option is that in some cases unnecessary declarations might be produced. • `As required`. Declarations are produced only when required by an element or attribute that is in that namespace. The disadvantage of this option is that the same namespace declaration might need to be produced more than once in the output XML document. The default option is `At start of document`. This property is active only if namespaces are enabled for this message set.

XML document type settings

Property	Type	Meaning
DOCTYPE System ID	String	Specify the System ID for DOCTYPE external DTD subset. It overrides the equivalent message set property setting for this particular message. If the message set property `Suppress DOCTYPE` is set to `Yes`, this parameter is ignored and cannot be changed (the field is disabled). The default value is the value that you specified for the `DOCTYPE System ID` property for the message set.
DOCTYPE Public ID	String	Specify the Public ID for DOCTYPE external DTD subset. It overrides the equivalent message set property setting for this particular message. If the message set property `Suppress DOCTYPE` is set to `Yes`, this parameter is ignored and cannot be changed (the field is disabled) . The default value is the value that you specified for the `DOCTYPE Public ID` property for the message set.
DOCTYPE Text	String	Enter optional additional text to include within the DOCTYPE. It overrides the message set property for this particular message. If the message set property `Suppress DOCTYPE` is set to `Yes`, this parameter is ignored and cannot be changed (the field is disabled). For more information, see “MRM XML: In-line DTDs and the DOCTYPE text property” on page 5407. The default value is the value that you specified for the `DOCTYPE Text` property for the message set.

Property	Type	Meaning
Root Tag Name	String	Specify the name of the root tag for a message bit stream XML document. It overrides the message set property set for this message. The default value is the value that you specified for the `Root Tag Name` property for the message set. **Note:** This property is deprecated. Do not change its value from its default setting.

Field identification

A number of the following properties will only become active depending on the value that `Render` property is set to.

Property	Type	Meaning
Render	Enumerated type	Specify how the instantiated object or type is rendered (for output) in the resulting XML document. Select one of the following values from the drop-down list: • `XMLElement`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their `XML Name` property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLElementAttrID`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in `ID Attribute Name` and a value as specified in `ID Attribute Value`. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set `XML Name`, `ID Attribute Name`, `ID Attribute Value` to the same values: – You must also set `Value Attribute Name` to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 6262 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.
XML Name	String	Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message). This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name. You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name. If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).

Property	Type	Meaning
ID Attribute Name	String	Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
ID Attribute Value	String	Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Message TDS properties:

Message TDS properties.

Property	Type	Meaning
Message Key	String	Specify an alternative unique value that identifies the message in the bit stream. This property is required if the message is embedded within another message. **Note:** From Version 6.0 onwards, the use of `Message Key` has been deprecated for identifying an embedded message. You now have the option of identifying an embedded message by Message Identity, using the `Message Alias` logical property.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM TDS format" on page 1221
The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

"Multipart messages" on page 1191
A *multipart message* contains one or more other messages within its structure. The contained message is sometimes referred to as an *embedded message*.

"Identifying an embedded message by using a Message Identity" on page 1193
You can identify an embedded message by using the Message Identity. This technique is used by the MRM domain, and replaces the use of the Message Key.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message logical properties" on page 5449
This section describes the logical properties of a message.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Documentation properties for all message set objects:

Use the documentation property of an object to add information that describes the function of the object.

The documentation property is available on all objects except Key, Keyref, and Unique objects.

The property is a string field, and you can use any standard alphanumeric characters.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

Simple type properties:

Different types of properties are available for a simple type.

A simple type can have the following properties;

- "Simple type logical properties" on page 5450
- "Simple type CWF properties" on page 5474
- "Simple type XML properties" on page 5498
- "Simple type TDS properties" on page 5532
- "Documentation properties for all message set objects" on page 5413

A simple type can also have "Simple type logical value constraints" on page 5450.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

Simple type logical properties:

The logical properties of a simple type.

Property	Type	Meaning
Name	String	Specify a name for the object when you create it. Names can consist of alphanumeric characters, including the letters *A* through *Z*, *a* through *z*, and the digits *0* through *9*. They might also include the following punctuation characters; • - the hyphen • _ the underscore • . the period Names can start only with a letter or the underscore character, and not with a number, hyphen, or period. Names that begin with `xml`, or any variant of these characters (for example `XmL`), are reserved by the XML standards specification. Further details of naming conventions and allowable characters can be found in the Extensible Markup Language (XML) specification that can be found on the World Wide Web Consortium (W3C) Web site.
Base Type	Enumerated type	This property only applies to a simple type restriction. You can use this property to select a base type that is used as the starting point to define a new simple type that is derived by setting additional value constraints.
Item Type	Enumerated type	This property only applies to a simple type list. You can use this property to select the type that is used as the item type of the list.

Property	Type	Meaning
Variety	Enumerated type	This property displays the variety of the simple type you have selected, either atomic, list, or union.

A simple type can also have “Simple type logical value constraints” on page 5450.

Related concepts:

“The message model” on page 1160
The message model consists of the following components.

Related tasks:

“Working with MRM message model objects” on page 2870
Add, configure, and delete objects.

Related reference:

“Message model object properties” on page 5416
Access property information by property kind, or by object.

“Logical properties for message model objects” on page 5416
Logical property information is available for certain objects.

“Message model object properties by object” on page 5536
The following objects have properties that can be viewed or set.

Simple type logical value constraints:

The properties, and their permissible values, vary according to the object type.

The properties that are displayed on the object page, and the values that those properties can take, can vary according to the type of the object. For example, the properties for type *string* are different from those of type *Boolean*. Select the link for the object type from the following table.

Type of object	Properties
Binary types	• base64Binary • hexBinary
Boolean types	• Boolean
DateTime types	• date • dateTime • gDay • gMonth • gMonthDay • gYear • gYearMonth • time
Decimal types	• decimal • integer • negativeInteger • nonNegativeInteger • nonPositiveInteger • positiveInteger • unsignedLong

Type of object	Properties
Float types	• double • float
Integer types	• byte • int • long • short • unsignedByte • unsignedInt • unsignedShort
Interval types	• duration
String types	• anyURI • ENTITIES • ENTITY • ID • IDREF • IDREFS • language • Name • NCName • NMTOKEN • NMTOKENS • normalizedString • NOTATION • QName • string • token

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Logical properties for value constraints for simple type binary types:

The logical properties for simple type binary types.

The simple type value constraint properties described here apply to:

- Objects: Simple types

- Binary schema types: base64Binary, hexBinary

Length constraints

Property	Type	Meaning
Length	Integer	Specify the exact length of the simple type in bytes or characters. The value must be greater than 0, and less than 2147483648.
Min	Integer	Specify the minimum length of the simple type in bytes or characters. The value must be greater than 0, and less than 2147483648.
Max	Integer	Specify the maximum length of the simple type in bytes or characters. The value must be greater than 0, and less than 2147483648.

Property	Type	Meaning
White Space	Enumerated type	Set this property to control the processing of white space characters received for this type. Select one of the following values: • preserve. If you set the property to preserve, all white space characters including carriage return, line feed, and tab are preserved. • replace. If you set the property to replace, all carriage return, line feed, and tab characters are replaced with a space character. • collapse. If you set the property to collapse, all carriage return, line feed, and tab characters are replaced with a space character. All adjacent white space characters are then collapsed to a single space character, and leading or trailing spaces are stripped from the data.

Enumerations

Property	Type	Meaning
Enumerations	String	Set this property to constrain the values to the list that is specified in this property. For example, you might create a simple type called *RainbowColors*, and add Red, `Orange`, `Yellow`, `Green`, `Blue`, `Indigo`, and `Violet` to the enumerations list. You must ensure that you have all variations of the data that you are likely to receive in the message defined in the list. For example, Yellow, yellow, yel, and y might be variations of a single color. Select **Add** to add a default enumeration. Overtype the default with the data you require. To change an entry, select the entry, and click on the entry a second time (as distinct from double-click). You can now update the selected entry.

Patterns

Property	Type	Meaning
Patterns	String	`Patterns` are a regular expression, or a series of regular expressions, that are used to constrain the data within the simple type. For further information about patterns and their syntax see "Using regular expressions to parse data elements" on page 6301. Select **Add** to add a default pattern. Overtype the default with the data you require. To change an entry, select the entry, and click on the entry a second time (as distinct from double-click). You can now update the selected entry.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

Logical properties for value constraints for simple type Boolean types:

The logical properties for simple type Boolean types.

The simple type value constraint properties described here apply to:

- Objects: Simple types
- Boolean schema types: Boolean

Property	Type	Meaning
White Space	Enumerated type	Set this property to control the processing of white space characters received for this type. Select one of the following values: • preserve. If you set the property to preserve, all white space characters including carriage return, line feed, and tab are preserved. • replace. If you set the property to replace, all carriage return, line feed, and tab characters are replaced with a space character. • collapse. If you set the property to collapse, all carriage return, line feed, and tab characters are replaced with a space character. All adjacent white space characters are then collapsed to a single space character, and leading or trailing spaces are stripped from the data.

Patterns

Property	Type	Meaning
Patterns	String	`Patterns` are a regular expression, or a series of regular expressions, that are used to constrain the data within the simple type. For further information about patterns and their syntax see "Using regular expressions to parse data elements" on page 6301. Select **Add** to add a default pattern. Overtype the default with the data you require. To change an entry, select the entry, and click on the entry a second time (as distinct from double-click). You can now update the selected entry.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

Logical properties for value constraints for simple type dateTime types:

The logical properties for simple type dateTime types.

The simple type value constraint properties described here apply to:

- Objects: Simple types
- DateTime schema types: date, dateTime, gDay, gMonth, gMonthDay, gYear, gYearMonth, time

Inclusive Constraints

Property	Type	Meaning
Min	Integer	Specify the minimum value for which the data in the message must be greater than, or equal to. When this value is set, it cannot be equal to, or greater than, the `Max` Inclusive Constraint property. You cannot specify both `Min` Inclusive Constraint and `Min` Exclusive Constraint properties together for the same simple type.
Max	Integer	Specify the maximum value for which the data in the message must be less than, or equal to. When this value is set, it cannot be equal to, or less than, the `Min` Inclusive Constraint property. You cannot specify both `Max` Inclusive Constraint and `Max` Exclusive Constraint properties together for the same simple type.

Exclusive Constraints

Property	Type	Meaning
Min	Integer	Specify the minimum value for which the data in the message must be greater than. When this value is set it cannot be equal to, or greater than, the `Max` Inclusive Constraint property. You cannot specify both `Min` Inclusive Constraint and `Min` Exclusive Constraint properties together for the same simple type.
Max	Integer	Specify the maximum value for which the data in the message must be less than. When this value is set, it cannot be equal to, or less than, the `Min` Inclusive Constraint property. You cannot specify both `Max` Inclusive Constraint and `Max` Exclusive Constraint properties together for the same simple type.

Property	Type	Meaning
White Space	Enumerated type	Set this property to control the processing of white space characters received for this type. Select one of the following values: • preserve. If you set the property to preserve, all white space characters including carriage return, line feed, and tab are preserved. • replace. If you set the property to replace, all carriage return, line feed, and tab characters are replaced with a space character. • collapse. If you set the property to collapse, all carriage return, line feed, and tab characters are replaced with a space character. All adjacent white space characters are then collapsed to a single space character, and leading or trailing spaces are stripped from the data.

Enumerations

Property	Type	Meaning
Enumerations	String	Set this property to constrain the values to the list that is specified in this property. For example, you might create a simple type called *RainbowColors*, and add Red, `Orange`, `Yellow`, `Green`, `Blue`, `Indigo`, and `Violet` to the enumerations list. You must ensure that you have all variations of the data that you are likely to receive in the message defined in the list. For example, Yellow, yellow, yel, and y might be variations of a single color. Select **Add** to add a default enumeration. Overtype the default with the data you require. To change an entry, select the entry, and click on the entry a second time (as distinct from double-click). You can now update the selected entry.

Patterns

Property	Type	Meaning
Patterns	String	`Patterns` are a regular expression, or a series of regular expressions, that are used to constrain the data within the simple type. For further information about patterns and their syntax see "Using regular expressions to parse data elements" on page 6301. Select **Add** to add a default pattern. Overtype the default with the data you require. To change an entry, select the entry, and click on the entry a second time (as distinct from double-click). You can now update the selected entry.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

Logical properties for value constraints for simple type decimal types:

The logical properties for simple type decimal types.

The simple type value constraint properties described here apply to:

- Objects: Simple types
- Decimal schema types: decimal, integer, negativeInteger, nonNegativeInteger, nonPositiveInteger, positiveInteger, unsignedLong

Inclusive Constraints

Property	Type	Meaning
Min	Integer	Specify the minimum value for which the data in the message must be greater than, or equal to. When this value is set, it cannot be equal to, or greater than, the `Max` Inclusive Constraint property. You cannot specify both `Min` Inclusive Constraint and `Min` Exclusive Constraint properties together for the same simple type.
Max	Integer	Specify the maximum value for which the data in the message must be less than, or equal to. When this value is set, it cannot be equal to, or less than, the `Min` Inclusive Constraint property. You cannot specify both `Max` Inclusive Constraint and `Max` Exclusive Constraint properties together for the same simple type.

Exclusive Constraints

Property	Type	Meaning
Min	Integer	Specify the minimum value for which the data in the message must be greater than. When this value is set it cannot be equal to, or greater than, the `Max` Inclusive Constraint property. You cannot specify both `Min` Inclusive Constraint and `Min` Exclusive Constraint properties together for the same simple type.
Max	Integer	Specify the maximum value for which the data in the message must be less than. When this value is set, it cannot be equal to, or less than, the `Min` Inclusive Constraint property. You cannot specify both `Max` Inclusive Constraint and `Max` Exclusive Constraint properties together for the same simple type.

Property	Type	Meaning
Fraction Digits	Integer	Set this property to limit the number of digits in the fraction part of a numeric value to the number of digits specified in this property. The value must be greater than, or equal to, 0, and less than 2147483648. The value set for `Fraction Digits` cannot be greater than the value specified for `Total Digits`.

Property	Type	Meaning
Total Digits	Integer	Set this property to set the maximum number of digits in a numeric value to the number of digits specified in this property. The value must be greater than, or equal to, 0, and less than 2147483648. The value set for `Total Digits` cannot be less than the value specified for `Fraction Digits`.

Property	Type	Meaning
White Space	Enumerated type	Set this property to control the processing of white space characters received for this type. Select one of the following values: • preserve. If you set the property to preserve, all white space characters including carriage return, line feed, and tab are preserved. • replace. If you set the property to replace, all carriage return, line feed, and tab characters are replaced with a space character. • collapse. If you set the property to collapse, all carriage return, line feed, and tab characters are replaced with a space character. All adjacent white space characters are then collapsed to a single space character, and leading or trailing spaces are stripped from the data.

Enumerations

Property	Type	Meaning
Enumerations	String	Set this property to constrain the values to the list that is specified in this property. For example, you might create a simple type called *RainbowColors*, and add Red, `Orange`, `Yellow`, `Green`, `Blue`, `Indigo`, and `Violet` to the enumerations list. You must ensure that you have all variations of the data that you are likely to receive in the message defined in the list. For example, Yellow, yellow, yel, and y might be variations of a single color. Select **Add** to add a default enumeration. Overtype the default with the data you require. To change an entry, select the entry, and click on the entry a second time (as distinct from double-click). You can now update the selected entry.

Patterns

Property	Type	Meaning
Patterns	String	`Patterns` are a regular expression, or a series of regular expressions, that are used to constrain the data within the simple type. For further information about patterns and their syntax see "Using regular expressions to parse data elements" on page 6301. Select **Add** to add a default pattern. Overtype the default with the data you require. To change an entry, select the entry, and click on the entry a second time (as distinct from double-click). You can now update the selected entry.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

Logical properties for value constraints for simple type float types:

The logical properties for simple type float types.

The simple type value constraint properties described here apply to:

- Objects: Simple types
- Float schema types: double, float

Inclusive Constraints

Property	Type	Meaning
Min	Integer	Specify the minimum value for which the data in the message must be greater than, or equal to. When this value is set, it cannot be equal to, or greater than, the `Max` Inclusive Constraint property. You cannot specify both `Min` Inclusive Constraint and `Min` Exclusive Constraint properties together for the same simple type.
Max	Integer	Specify the maximum value for which the data in the message must be less than, or equal to. When this value is set, it cannot be equal to, or less than, the `Min` Inclusive Constraint property. You cannot specify both `Max` Inclusive Constraint and `Max` Exclusive Constraint properties together for the same simple type.

Exclusive Constraints

Property	Type	Meaning
Min	Integer	Specify the minimum value for which the data in the message must be greater than. When this value is set it cannot be equal to, or greater than, the `Max` Inclusive Constraint property. You cannot specify both `Min` Inclusive Constraint and `Min` Exclusive Constraint properties together for the same simple type.
Max	Integer	Specify the maximum value for which the data in the message must be less than. When this value is set, it cannot be equal to, or less than, the `Min` Inclusive Constraint property. You cannot specify both `Max` Inclusive Constraint and `Max` Exclusive Constraint properties together for the same simple type.

Property	Type	Meaning
White Space	Enumerated type	Set this property to control the processing of white space characters received for this type. Select one of the following values: • preserve. If you set the property to preserve, all white space characters including carriage return, line feed, and tab are preserved. • replace. If you set the property to replace, all carriage return, line feed, and tab characters are replaced with a space character. • collapse. If you set the property to collapse, all carriage return, line feed, and tab characters are replaced with a space character. All adjacent white space characters are then collapsed to a single space character, and leading or trailing spaces are stripped from the data.

Enumerations

Property	Type	Meaning
Enumerations	String	Set this property to constrain the values to the list that is specified in this property. For example, you might create a simple type called *RainbowColors*, and add Red, `Orange`, `Yellow`, `Green`, `Blue`, `Indigo`, and `Violet` to the enumerations list. You must ensure that you have all variations of the data that you are likely to receive in the message defined in the list. For example, Yellow, yellow, yel, and y might be variations of a single color. Select **Add** to add a default enumeration. Overtype the default with the data you require. To change an entry, select the entry, and click on the entry a second time (as distinct from double-click). You can now update the selected entry.

Patterns

Property	Type	Meaning
Patterns	String	`Patterns` are a regular expression, or a series of regular expressions, that are used to constrain the data within the simple type. For further information about patterns and their syntax see "Using regular expressions to parse data elements" on page 6301. Select **Add** to add a default pattern. Overtype the default with the data you require. To change an entry, select the entry, and click on the entry a second time (as distinct from double-click). You can now update the selected entry.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

Logical properties for value constraints for simple type integer types:

The logical properties for simple type integer types.

The simple type value constraint properties described here apply to:

- Objects: Simple types
- Integer schema types: byte, int, long, short, unsignedByte, unsignedInt, unsignedShort

Inclusive Constraints

Property	Type	Meaning
Min	Integer	Specify the minimum value for which the data in the message must be greater than, or equal to. When this value is set, it cannot be equal to, or greater than, the `Max` Inclusive Constraint property. You cannot specify both `Min` Inclusive Constraint and `Min` Exclusive Constraint properties together for the same simple type.
Max	Integer	Specify the maximum value for which the data in the message must be less than, or equal to. When this value is set, it cannot be equal to, or less than, the `Min` Inclusive Constraint property. You cannot specify both `Max` Inclusive Constraint and `Max` Exclusive Constraint properties together for the same simple type.

Exclusive Constraints

Property	Type	Meaning
Min	Integer	Specify the minimum value for which the data in the message must be greater than. When this value is set it cannot be equal to, or greater than, the `Max` Inclusive Constraint property. You cannot specify both `Min` Inclusive Constraint and `Min` Exclusive Constraint properties together for the same simple type.

Property	Type	Meaning
Max	Integer	Specify the maximum value for which the data in the message must be less than. When this value is set, it cannot be equal to, or less than, the `Min` Inclusive Constraint property. You cannot specify both `Max` Inclusive Constraint and `Max` Exclusive Constraint properties together for the same simple type.

Property	Type	Meaning
Fraction Digits	Integer	Set this property to limit the number of digits in the fraction part of a numeric value to the number of digits specified in this property. The value must be greater than, or equal to, 0, and less than 2147483648. The value set for `Fraction Digits` cannot be greater than the value specified for `Total Digits`.

Property	Type	Meaning
Total Digits	Integer	Set this property to set the maximum number of digits in a numeric value to the number of digits specified in this property. The value must be greater than, or equal to, 0, and less than 2147483648. The value set for `Total Digits` cannot be less than the value specified for `Fraction Digits`.

Property	Type	Meaning
White Space	Enumerated type	Set this property to control the processing of white space characters received for this type. Select one of the following values: • preserve. If you set the property to preserve, all white space characters including carriage return, line feed, and tab are preserved. • replace. If you set the property to replace, all carriage return, line feed, and tab characters are replaced with a space character. • collapse. If you set the property to collapse, all carriage return, line feed, and tab characters are replaced with a space character. All adjacent white space characters are then collapsed to a single space character, and leading or trailing spaces are stripped from the data.

Enumerations

Property	Type	Meaning
Enumerations	String	Set this property to constrain the values to the list that is specified in this property. For example, you might create a simple type called *RainbowColors*, and add Red, `Orange`, `Yellow`, `Green`, `Blue`, `Indigo`, and `Violet` to the enumerations list. You must ensure that you have all variations of the data that you are likely to receive in the message defined in the list. For example, Yellow, yellow, yel, and y might be variations of a single color. Select **Add** to add a default enumeration. Overtype the default with the data you require. To change an entry, select the entry, and click on the entry a second time (as distinct from double-click). You can now update the selected entry.

Patterns

Property	Type	Meaning
Patterns	String	`Patterns` are a regular expression, or a series of regular expressions, that are used to constrain the data within the simple type. For further information about patterns and their syntax see "Using regular expressions to parse data elements" on page 6301. Select **Add** to add a default pattern. Overtype the default with the data you require. To change an entry, select the entry, and click on the entry a second time (as distinct from double-click). You can now update the selected entry.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

Logical properties for value constraints for simple type interval types:

The logical properties for simple type interval types.

The simple type value constraint properties described here apply to:

- Objects: Simple types
- Interval schema types: duration

Inclusive Constraints

Property	Type	Meaning
Min	Integer	Specify the minimum value for which the data in the message must be greater than, or equal to. When this value is set, it cannot be equal to, or greater than, the `Max` Inclusive Constraint property. You cannot specify both `Min` Inclusive Constraint and `Min` Exclusive Constraint properties together for the same simple type.
Max	Integer	Specify the maximum value for which the data in the message must be less than, or equal to. When this value is set, it cannot be equal to, or less than, the `Min` Inclusive Constraint property. You cannot specify both `Max` Inclusive Constraint and `Max` Exclusive Constraint properties together for the same simple type.

Exclusive Constraints

Property	Type	Meaning
Min	Integer	Specify the minimum value for which the data in the message must be greater than. When this value is set it cannot be equal to, or greater than, the `Max` Inclusive Constraint property. You cannot specify both `Min` Inclusive Constraint and `Min` Exclusive Constraint properties together for the same simple type.

Property	Type	Meaning
Max	Integer	Specify the maximum value for which the data in the message must be less than. When this value is set, it cannot be equal to, or less than, the `Min` Inclusive Constraint property. You cannot specify both `Max` Inclusive Constraint and `Max` Exclusive Constraint properties together for the same simple type.

Property	Type	Meaning
White Space	Enumerated type	Set this property to control the processing of white space characters received for this type. Select one of the following values: • preserve. If you set the property to preserve, all white space characters including carriage return, line feed, and tab are preserved. • replace. If you set the property to replace, all carriage return, line feed, and tab characters are replaced with a space character. • collapse. If you set the property to collapse, all carriage return, line feed, and tab characters are replaced with a space character. All adjacent white space characters are then collapsed to a single space character, and leading or trailing spaces are stripped from the data.

Enumerations

Property	Type	Meaning
Enumerations	String	Set this property to constrain the values to the list that is specified in this property. For example, you might create a simple type called *RainbowColors*, and add Red, `Orange`, `Yellow`, `Green`, `Blue`, `Indigo`, and `Violet` to the enumerations list. You must ensure that you have all variations of the data that you are likely to receive in the message defined in the list. For example, Yellow, yellow, yel, and y might be variations of a single color. Select **Add** to add a default enumeration. Overtype the default with the data you require. To change an entry, select the entry, and click on the entry a second time (as distinct from double-click). You can now update the selected entry.

Patterns

Property	Type	Meaning
Patterns	String	`Patterns` are a regular expression, or a series of regular expressions, that are used to constrain the data within the simple type. For further information about patterns and their syntax see "Using regular expressions to parse data elements" on page 6301. Select **Add** to add a default pattern. Overtype the default with the data you require. To change an entry, select the entry, and click on the entry a second time (as distinct from double-click). You can now update the selected entry.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

Logical properties for value constraints for simple type string types:

The logical properties for simple type string types.

The simple type value constraint properties described here apply to:

- Objects: Simple types
- Integer schema types: byte, int, long, short, unsignedByte, unsignedInt, unsignedShort

Length constraints

Property	Type	Meaning
Length	Integer	Specify the exact length of the simple type in bytes or characters. The value must be greater than 0, and less than 2147483648.
Min	Integer	Specify the minimum length of the simple type in bytes or characters. The value must be greater than 0, and less than 2147483648.

Property	Type	Meaning
Max	Integer	Specify the maximum length of the simple type in bytes or characters. The value must be greater than 0, and less than 2147483648.

Property	Type	Meaning
White Space	Enumerated type	Set this property to control the processing of white space characters received for this type. Select one of the following values: • preserve. If you set the property to preserve, all white space characters including carriage return, line feed, and tab are preserved. • replace. If you set the property to replace, all carriage return, line feed, and tab characters are replaced with a space character. • collapse. If you set the property to collapse, all carriage return, line feed, and tab characters are replaced with a space character. All adjacent white space characters are then collapsed to a single space character, and leading or trailing spaces are stripped from the data.

Enumerations

Property	Type	Meaning
Enumerations	String	Set this property to constrain the values to the list that is specified in this property. For example, you might create a simple type called *RainbowColors*, and add Red, `Orange`, `Yellow`, `Green`, `Blue`, `Indigo`, and `Violet` to the enumerations list. You must ensure that you have all variations of the data that you are likely to receive in the message defined in the list. For example, Yellow, yellow, yel, and y might be variations of a single color. Select **Add** to add a default enumeration. Overtype the default with the data you require. To change an entry, select the entry, and click on the entry a second time (as distinct from double-click). You can now update the selected entry.

Patterns

Property	Type	Meaning
Patterns	String	`Patterns` are a regular expression, or a series of regular expressions, that are used to constrain the data within the simple type. For further information about patterns and their syntax see "Using regular expressions to parse data elements" on page 6301. Select **Add** to add a default pattern. Overtype the default with the data you require. To change an entry, select the entry, and click on the entry a second time (as distinct from double-click). You can now update the selected entry.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined

structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

Simple type CWF properties:

There are no properties to show.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM Custom Wire Format" on page 1214
Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not separated by delimiters.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical

formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Simple type XML properties:

There are no properties to show.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the

function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Simple type TDS properties:

There are no properties to show.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM TDS format" on page 1221
The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Documentation properties for all message set objects:

Use the documentation property of an object to add information that describes the function of the object.

The documentation property is available on all objects except Key, Keyref, and Unique objects.

The property is a string field, and you can use any standard alphanumeric characters.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

Unique properties:

Unique logical, CWF, XML, TDS, and documentation properties.

A unique can have the following properties;

- "Unique logical properties" on page 5452
- "Unique CWF properties" on page 5474
- "Unique XML properties" on page 5499
- "Unique TDS properties" on page 5532
- "Documentation properties for all message set objects" on page 5413

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the

function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

Unique logical properties:

There are no properties to show.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Unique CWF properties:

There are no properties to show.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Unique XML properties:

There are no properties to show.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical

model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Unique TDS properties:

There are no properties to show.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Documentation properties for all message set objects:

Use the documentation property of an object to add information that describes the function of the object.

The documentation property is available on all objects except Key, Keyref, and Unique objects.

The property is a string field, and you can use any standard alphanumeric characters.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

Wildcard attribute properties:

Different types of properties are available for a wildcard attribute.

A wildcard attribute can have the following properties;

- "Wildcard attribute logical properties" on page 5453
- "Wildcard attribute CWF properties" on page 5475
- "Wildcard attribute XML properties" on page 5499
- "Wildcard attribute TDS properties" on page 5534
- "Documentation properties for all message set objects" on page 5413

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

Wildcard attribute logical properties:

The logical properties of a wildcard attribute.

Property	Type	Meaning
Namespace	String	Namespaces are a simple method for qualifying element and attribute names by associating them with namespaces identified by URI references. This field is initially blank.

Property	Type	Meaning
Process Content	Enumerated type	If a message contains an attribute that corresponds to a wildcard in the message model, `Process Content` defines how the attribute is validated. Select one of the following options: • strict. The parser can match only against attributes declared in the specified namespace. • lax. The parser attempts to match against attributes declared in all accessible namespaces. If the specified namespace cannot be found, an error is not generated. • skip. If you select skip, the parser does not perform validation on the attribute.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Wildcard attribute CWF properties:

There are no properties to show.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM Custom Wire Format" on page 1214
Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not separated by delimiters.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Wildcard attribute XML properties:

There are no properties to show.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Wildcard attribute TDS properties:

There are no properties to show.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM TDS format" on page 1221
The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Documentation properties for all message set objects:

Use the documentation property of an object to add information that describes the function of the object.

The documentation property is available on all objects except Key, Keyref, and Unique objects.

The property is a string field, and you can use any standard alphanumeric characters.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

Wildcard element properties:

Different types of properties are available for a wildcard element.

A wildcard element can have the following properties;

- "Wildcard element logical properties" on page 5453
- "Wildcard element CWF properties" on page 5475
- "Wildcard element XML properties" on page 5500
- "Wildcard element TDS properties" on page 5534
- "Documentation properties for all message set objects" on page 5413

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

Wildcard element logical properties:

The logical properties of a wildcard element include properties that specify the number of occurrences of the wildcard element.

Property	Type	Meaning
Namespace	String	Namespaces are a simple method for qualifying element and attribute names by associating them with namespaces identified by URI references. This field is initially blank.

Property	Type	Meaning
Process Content	Enumerated type	If a message contains an element that corresponds to a wildcard in the message model, `Process Content` defines how the element is validated. Select one of the following options: • strict. The parser can match only against elements declared in the specified namespace. • lax. The parser attempts to match against elements declared in any accessible namespace. If the specified namespace cannot be found, an error is not generated. • skip. If you select skip the parser does not perform validation on the element.

Occurrences

Property	Type	Meaning
Min Occurs	Integer	Specify the minimum number of times that the object can repeat. The default value is 1. If the value is set to 0, the object is optional. With the exception of `Max Occurs` being set to -1, if a value is set for `Min Occurs`, it must be less than or equal to the value in `Max Occurs`. This property is ignored by brokers that are earlier than WebSphere Message Broker Version 6.0.

Property	Type	Meaning
Max Occurs	Integer	Specify the maximum number of times that the object can repeat. The default value is 1. If this property is not set, the object cannot occur more than once. If this property is set to 0, it is interpreted as if the object does not exist in the message. It can also be set to -1, to indicate that the limit is unbounded and there is no maximum to the number of occurrences. This property is ignored by brokers that are earlier than WebSphere Message Broker Version 6.0.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Wildcard element CWF properties:

There are no properties to show.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM Custom Wire Format" on page 1214
Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not separated by delimiters.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Wildcard element XML properties:

There are no properties to show.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Wildcard element TDS properties:

There are no properties to show.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM TDS format" on page 1221
The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"XML wire format physical properties for message model objects" on page 5476
XML wire format physical property information is available for some objects.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Documentation properties for all message set objects:

Use the documentation property of an object to add information that describes the function of the object.

The documentation property is available on all objects except Key, Keyref, and Unique objects.

The property is a string field, and you can use any standard alphanumeric characters.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

Deprecated message model object properties

Some objects in the message model are deprecated, but you can reference the information for their properties.

You can access the reference information for the properties of deprecated message model objects in two ways. The following topics allow you to access the property information by property kind:

- "Logical properties for deprecated message model objects" on page 6070
- "Physical properties for deprecated message model objects" on page 6075
- "Documentation properties for all message set objects" on page 5413

Alternatively, you can access the property information by object, starting from the following topic:

- "Deprecated message model object properties by object" on page 6090

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Logical properties for deprecated message model objects" on page 6070
Logical property information for compound elements and embedded simple types.

"Physical properties for deprecated message model objects" on page 6075
CWF, XML, and TDS format physical properties for deprecated objects.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Deprecated message model object properties by object" on page 6090
Properties for compound elements and embedded simple types.

Logical properties for deprecated message model objects:

Logical property information for compound elements and embedded simple types.

Logical property information is available for the following deprecated objects:

- "Compound element logical properties"
- "Embedded simple type logical properties" on page 6074

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

"Message definition files" on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Deprecated message model object properties by object" on page 6090
Properties for compound elements and embedded simple types.

Compound element logical properties:

The logical properties of a compound element include properties that specify the number of occurrences of the compound element.

Property	Type	Meaning
Name	String	Specify a name for the object when you create it. Names can consist of alphanumeric characters, including the letters *A* through *Z*, *a* through *z*, and the digits *0* through *9*. They might also include the following punctuation characters; • - the hyphen • _ the underscore • . the period Names can start only with a letter or the underscore character, and not with a number, hyphen, or period. Names that begin with `xml`, or any variant of these characters (for example `XmL`), are reserved by the XML standards specification. Further details of naming conventions and allowable characters can be found in the Extensible Markup Language (XML) specification that can be found on the World Wide Web Consortium (W3C) Web site.

Property	Type	Meaning
Namespace	Enumerated type	Namespaces are a simple method for qualifying element and attribute names by associating them with namespaces identified by URI references. If <no target namespace> is displayed, a namespace has not been set for this object. If the property is inactive, the message set has not been configured to support namespaces. Where the property is active, namespaces that are available for selection are displayed in the drop-down list.
Nillable	Check box	Select this option if you want the element to be able to be defined as null. A null value is distinct from being empty, when the element contains no data.
Abstract	Check box	Select this option if you do not want the element to appear in the message, but require one of the members of its substitution group to appear in its place.

Value

Property	Type	Meaning
Default	Button and String	This property provides the default value for an element or attribute. **XMLNSC domain** When parsing with validation enabled, default values are applied to missing attributes and empty elements as required by the XML Schema 1.0 specification. When writing, elements or attributes that are missing from the message tree are not automatically added to the output XML bit stream, even if they have default values. If missing elements or attributes are required, the message tree can be serialized and then re-parsed with validation enabled. **MRM (CWF and TDS physical formats)** When writing a fixed-length portion of a message (CWF or fixed-length TDS), if an attribute or element is missing from the message tree, the default value is inserted into the bit stream so that the message structure is preserved. **MRM (XML physical format)** No support for default values **Other domains** No support for default values.

Property	Type	Meaning
Fixed	Button and String	This property provides the fixed value for an element or attribute. **XMLNSC domain** When parsing with validation enabled, if an attribute or element is present, the value is validated against the fixed value. If the values are not equal, a validation error is signalled. Also, when parsing with validation enabled, fixed values are applied to missing attributes and empty elements as required by the XML Schema 1.0 specification. When writing, elements or attributes that are missing from the message tree are not automatically added to the output XML bit stream, even if a fixed value has been specified. If missing elements or attributes are required, the message tree can be serialized and then re-parsed with validation enabled. **MRM (CWF and TDS physical formats)** When writing a fixed-length portion of a message (CWF or fixed-length TDS), if an attribute or element is missing from the message tree, the fixed value is inserted into the bit stream so that the message structure is preserved. **MRM (XML physical format)** No support for fixed values **Other domains** No support for fixed values.
Interpret Value As	Enumerated type	Specify if values stored within this object must be interpreted as having significance for the parser and, if so, the type of interpretation that must occur. Select one of the following options: • None This value is the default value, and indicates that the element or attribute does not have a key value associated with it. • MessageSetIdentity. Specifies that the value of the element or attribute corresponds to the identifier, name, or alias (in that priority order) that is associated with the message set where all subsequent embedded messages that are descendents of the enclosing message are defined. This value remains in force unless a new element or attribute MessageSetIdentity field is encountered which resets the MessageSetIdentity value. • MessageIdentity. Specifies that the value of the element or attribute corresponds to the name or alias (in that priority order) that is associated with a message, and acts as an identifier for subsequent embedded messages which are the immediate children of the enclosing message. This identity applies until a new element or attribute MessageIdentity field is encountered at the same level in the tree. The embedded message can be defined in either the current message set, or in a message set identified by using a MessageSetIdentity. **Note:** This property is applicable only when the type of the object is derived from xsd:string.

Occurrences

Property	Type	Meaning
Min Occurs	Integer	Specify the minimum number of times that the object can repeat. The default value is 1. If the value is set to 0, the object is optional. With the exception of `Max Occurs` being set to -1, if a value is set for `Min Occurs`, it must be less than or equal to the value in `Max Occurs`.

Property	Type	Meaning
Max Occurs	Integer	Specify the maximum number of times that the object can repeat. The default value is 1. If this property is not set, the object cannot occur more than once. If this property is set to 0, it is interpreted as if the object does not exist in the message. It can also be set to -1, to indicate that the limit is unbounded and there is no maximum to the number of occurrences.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Compound element properties" on page 6091
Logical, CWF, XML, TDS, and document properties for a compound element.

Compound element complex type logical properties:

The **Name** property is set to **ANONYMOUS** and cannot be changed.

Name *property:*
Only the complex type properties shown in the following tables are applicable to compound elements.

Property	Type	Meaning
Name	String	This property is set to **ANONYMOUS**, and cannot be changed.

Content:

Property	Type	Meaning
Group Reference	Button	This radio button is already selected and cannot be changed.
Group Name	Enumerated type	The `Group Name` is the name of the group that this complex type is referring to. The groups available to be referenced can be selected from the drop-down list.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Logical properties for deprecated message model objects" on page 6070
Logical property information for compound elements and embedded simple types.

"Deprecated message model object properties by object" on page 6090
Properties for compound elements and embedded simple types.

Compound element value constraint properties:

The value constraints for a compound element.

The properties for compound element value constraints are identical to simple type value constraints. See "Simple type logical value constraints" on page 5450 for details.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Compound element properties" on page 6091
Logical, CWF, XML, TDS, and document properties for a compound element.

"Compound element logical properties" on page 6070
The logical properties of a compound element include properties that specify the number of occurrences of the compound element.

"Simple type logical value constraints" on page 5450
The properties, and their permissible values, vary according to the object type.

Embedded simple type logical properties:

The logical properties of an embedded simple type include properties that specify the number of occurrences of the embedded simple type.

Occurrences

Property	Type	Meaning
Min Occurs	Integer	Specify the minimum number of times that the object can repeat. The default value is 1. If the value is set to 0, the object is optional. With the exception of `Max Occurs` being set to -1, if a value is set for `Min Occurs`, it must be less than or equal to the value in `Max Occurs`.

Property	Type	Meaning
Max Occurs	Integer	Specify the maximum number of times that the object can repeat. The default value is 1. If this property is not set, the object cannot occur more than once. If this property is set to 0, it is interpreted as if the object does not exist in the message. It can also be set to -1, to indicate that the limit is unbounded and there is no maximum to the number of occurrences.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Physical properties for deprecated message model objects:

CWF, XML, and TDS format physical properties for deprecated objects.

Property information is available for deprecated objects within:

- "Custom Wire Format properties for deprecated message model objects" on page 6076
- "XML wire format physical properties for deprecated message model objects" on page 6081
- "TDS format physical properties for deprecated objects" on page 6085

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with a message set" on page 2840
Complete a variety of tasks that are involved in working with a message set.

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

Related reference:

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Custom Wire Format properties for deprecated message model objects"
CWF properties for compound elements and embedded simple types.

"XML wire format physical properties for deprecated message model objects" on page 6081
XML wire format physical properties for compound elements and embedded simple types.

"TDS format physical properties for deprecated objects" on page 6085
TDS format physical properties for compound elements and embedded simple types.

Custom Wire Format properties for deprecated message model objects:

CWF properties for compound elements and embedded simple types.

Custom wire format physical property information is available for the following deprecated objects:

- "Compound element CWF properties" on page 6077
- "Embedded simple type CWF properties" on page 6079

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM Custom Wire Format" on page 1214
Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not separated by delimiters.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Physical properties for deprecated message model objects" on page 6075
CWF, XML, and TDS format physical properties for deprecated objects.

"XML wire format physical properties for deprecated message model objects" on page 6081
XML wire format physical properties for compound elements and embedded simple types.

"TDS format physical properties for deprecated objects" on page 6085
TDS format physical properties for compound elements and embedded simple types.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Deprecated message model object properties by object" on page 6090
Properties for compound elements and embedded simple types.

Compound element CWF properties:

The properties, and their permissible values, vary according to the object type.

The properties that are displayed on the object page, and the values that those properties can take, can vary according to the type of the object. For example, the properties for type *string* are different from those of type *Boolean*. Select the link for the object type from the following table.

Type of object	**Properties**
Binary types	• ComIbmMrm _BaseValueBinary
Boolean types	• ComIbmMrm _BaseValueBoolean
DateTime types	• ComIbmMrm _BaseValueDateTime
Decimal types	• ComIbmMrm _BaseValueDecimal
Float types	• ComIbmMrm _BaseValueFloat
Integer types	• ComIbmMrm _BaseValueInt
String types	• ComIbmMrm _BaseValueString

`

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM Custom Wire Format" on page 1214
Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not separated by delimiters.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

"Working with data structures" on page 2930
You can create a message definition file in a message set by importing from XML Schema, XML DTD, SCA import or export components, IBM supplied messages, WSDL definitions, IDL files, C header files, and COBOL copybooks. This topic area describes how to import from these data structures using the command line or the WebSphere Message Broker Toolkit.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Compound element properties" on page 6091
Logical, CWF, XML, TDS, and document properties for a compound element.

Compound element complex type CWF properties:

There are no properties to show.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM Custom Wire Format" on page 1214
Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not separated by delimiters.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Physical properties for deprecated message model objects" on page 6075
CWF, XML, and TDS format physical properties for deprecated objects.

"Compound element complex type TDS properties" on page 6087
The TDS properties for compound element complex types are identical to the TDS properties for normal complex types.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Deprecated message model object properties by object" on page 6090
Properties for compound elements and embedded simple types.

Embedded simple type CWF properties:

The properties, and their permissible values, vary according to the object type.

The properties that are displayed on the object page, and the values that those properties can take, can vary according to the type of the object. For example, the properties for type *string* are different from those of type *Boolean*. Select the link for the object type from the following table.

Type of object	Properties
Binary types	• ComIbmMrm _AnonBinary
Boolean types	• ComIbmMrm _AnonBoolean
DateTime types	• ComIbmMrm _AnonDate • ComIbmMrm _AnonDateTime • ComIbmMrm _AnonGDay • ComIbmMrm _AnonGMonth • ComIbmMrm _AnonGMonthDay • ComIbmMrm _AnonGYear • ComIbmMrm _AnonGYearMonth • ComIbmMrm _AnonTime
Decimal types	• ComIbmMrm _AnonDecimal

Type of object	Properties
Float types	• ComIbmMrm _AnonFloat
Integer types	• ComIbmMrm _AnonInt
String types	• ComIbmMrm _AnonString

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM Custom Wire Format" on page 1214
Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not separated by delimiters.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Embedded simple type properties" on page 6180
Logical, CWF, XML, TDS, and documentation properties for embedded simple types.

"Embedded simple type logical properties" on page 6074
The logical properties of an embedded simple type include properties that specify the number of occurrences of the embedded simple type.

"Embedded simple type XML properties" on page 6084
The properties, and their permissible values, vary according to the object type.

"Embedded simple type TDS properties" on page 6088
The properties, and their permissible values, vary according to the object type.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

XML wire format physical properties for deprecated message model objects:

XML wire format physical properties for compound elements and embedded simple types.

XML wire format physical property information is available for the following deprecated objects:

- "Compound element XML properties"
- "Embedded simple type XML properties" on page 6084

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Physical properties for deprecated message model objects" on page 6075
CWF, XML, and TDS format physical properties for deprecated objects.

"Custom Wire Format properties for deprecated message model objects" on page 6076
CWF properties for compound elements and embedded simple types.

"TDS format physical properties for deprecated objects" on page 6085
TDS format physical properties for compound elements and embedded simple types.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Deprecated message model object properties by object" on page 6090
Properties for compound elements and embedded simple types.

Compound element XML properties:

The properties, and their permissible values, vary according to the object type.

The properties that are displayed on the object page, and the values that those properties can take, can vary according to the type of the object. For example, the properties for type *string* are different from those of type *Boolean*. Select the link for the object type from the following table.

Type of object	Properties
Binary types	• ComIbmMrm _BaseValueBinary
Boolean types	• ComIbmMrm _BaseValueBoolean
DateTime types	• ComIbmMrm _BaseValueDateTime
Decimal types	• ComIbmMrm _BaseValueDecimal
Float types	• ComIbmMrm _BaseValueFloat
Integer types	• ComIbmMrm _BaseValueInt
String types	• ComIbmMrm _BaseValueString

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Compound element properties" on page 6091
Logical, CWF, XML, TDS, and document properties for a compound element.

"Compound element logical properties" on page 6070
The logical properties of a compound element include properties that specify the number of occurrences of the compound element.

"Compound element CWF properties" on page 6077
The properties, and their permissible values, vary according to the object type.

"Compound element TDS properties" on page 6086
The properties, and their permissible values, vary according to the object type.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

Compound element complex type XML properties:

There are no properties to show.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Physical properties for deprecated message model objects" on page 6075
CWF, XML, and TDS format physical properties for deprecated objects.

"Compound element complex type CWF properties" on page 6078
There are no properties to show.

"Compound element complex type TDS properties" on page 6087
The TDS properties for compound element complex types are identical to the TDS properties for normal complex types.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Deprecated message model object properties by object" on page 6090
Properties for compound elements and embedded simple types.

Embedded simple type XML properties:

The properties, and their permissible values, vary according to the object type.

The properties that are displayed on the object page, and the values that those properties can take, can vary according to the type of the object. For example, the properties for type *string* are different from those of type *Boolean*. Select the link for the object type from the following table.

Type of object	Properties
Binary types	• ComIbmMrm _AnonBinary
Boolean types	• ComIbmMrm _AnonBoolean
DateTime types	• ComIbmMrm _AnonDate • ComIbmMrm _AnonDateTime • ComIbmMrm _AnonGDay • ComIbmMrm _AnonGMonth • ComIbmMrm _AnonGMonthDay • ComIbmMrm _AnonGYear • ComIbmMrm _AnonGYearMonth • ComIbmMrm _AnonTime
Decimal types	• ComIbmMrm _AnonDecimal
Float types	• ComIbmMrm _AnonFloat
Integer types	• ComIbmMrm _AnonInt
String types	• ComIbmMrm _AnonString

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM Custom Wire Format" on page 1214
Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not separated by delimiters.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Embedded simple type properties" on page 6180
Logical, CWF, XML, TDS, and documentation properties for embedded simple types.

"Embedded simple type logical properties" on page 6074
The logical properties of an embedded simple type include properties that specify the number of occurrences of the embedded simple type.

"Embedded simple type CWF properties" on page 6079
The properties, and their permissible values, vary according to the object type.

"Embedded simple type TDS properties" on page 6088
The properties, and their permissible values, vary according to the object type.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

TDS format physical properties for deprecated objects:

TDS format physical properties for compound elements and embedded simple types.

TDS format physical property information is available for the following deprecated objects:

- "Compound element TDS properties" on page 6086
- "Embedded simple type TDS properties" on page 6088

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM TDS format" on page 1221
The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Physical properties for deprecated message model objects" on page 6075
CWF, XML, and TDS format physical properties for deprecated objects.

"Custom Wire Format properties for deprecated message model objects" on page 6076
CWF properties for compound elements and embedded simple types.

"XML wire format physical properties for deprecated message model objects" on page 6081
XML wire format physical properties for compound elements and embedded simple types.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Deprecated message model object properties by object" on page 6090
Properties for compound elements and embedded simple types.

Compound element TDS properties:

The properties, and their permissible values, vary according to the object type.

The properties that are displayed on the object page, and the values that those properties can take, can vary according to the type of the object. For example, the properties for type *string* are different from those of type *Boolean*. Select the link for the object type from the following table.

Type of object	Properties
Binary types	• ComIbmMrm _BaseValueBinary
Boolean types	• ComIbmMrm _BaseValueBoolean
DateTime types	• ComIbmMrm _BaseValueDateTime
Decimal types	• ComIbmMrm _BaseValueDecimal
Float types	• ComIbmMrm _BaseValueFloat
Integer types	• ComIbmMrm _BaseValueInt

Type of object	Properties
String types	• ComIbmMrm _BaseValueString

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM TDS format" on page 1221
The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Compound element properties" on page 6091
Logical, CWF, XML, TDS, and document properties for a compound element.

"Compound element logical properties" on page 6070
The logical properties of a compound element include properties that specify the number of occurrences of the compound element.

"Compound element CWF properties" on page 6077
The properties, and their permissible values, vary according to the object type.

"Compound element XML properties" on page 6081
The properties, and their permissible values, vary according to the object type.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

Compound element complex type TDS properties:

The TDS properties for compound element complex types are identical to the TDS properties for normal complex types.

See "Complex type TDS properties" on page 5505 for details.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM TDS format" on page 1221
The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Physical properties for deprecated message model objects" on page 6075
CWF, XML, and TDS format physical properties for deprecated objects.

"Compound element complex type CWF properties" on page 6078
There are no properties to show.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Deprecated message model object properties by object" on page 6090
Properties for compound elements and embedded simple types.

Embedded simple type TDS properties:

The properties, and their permissible values, vary according to the object type.

The properties that are displayed on the object page, and the values that those properties can take, can vary according to the type of the object. For example, the properties for type *string* are different from those of type *Boolean*. Select the link for the object type from the following table.

Type of object	Properties
Binary types	• ComIbmMrm _AnonBinary
Boolean types	• ComIbmMrm _AnonBoolean

Type of object	Properties
DateTime types	• ComIbmMrm _AnonDate • ComIbmMrm _AnonDateTime • ComIbmMrm _AnonGDay • ComIbmMrm _AnonGMonth • ComIbmMrm _AnonGMonthDay • ComIbmMrm _AnonGYear • ComIbmMrm _AnonGYearMonth • ComIbmMrm _AnonTime
Decimal types	• ComIbmMrm _AnonDecimal
Float types	• ComIbmMrm _AnonFloat
Integer types	• ComIbmMrm _AnonInt
String types	• ComIbmMrm _AnonString

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM Custom Wire Format" on page 1214
Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not separated by delimiters.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Embedded simple type properties" on page 6180
Logical, CWF, XML, TDS, and documentation properties for embedded simple types.

"Embedded simple type logical properties" on page 6074
The logical properties of an embedded simple type include properties that specify the number of occurrences of the embedded simple type.

"Embedded simple type CWF properties" on page 6079
The properties, and their permissible values, vary according to the object type.

"Embedded simple type XML properties" on page 6084
The properties, and their permissible values, vary according to the object type.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

Documentation properties for all message set objects:

Use the documentation property of an object to add information that describes the function of the object.

The documentation property is available on all objects except Key, Keyref, and Unique objects.

The property is a string field, and you can use any standard alphanumeric characters.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

Deprecated message model object properties by object:

Properties for compound elements and embedded simple types.

The following deprecated objects have properties that can be viewed or set:

- "Compound element properties" on page 6091
- "Embedded simple type properties" on page 6180

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined

structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Logical properties for deprecated message model objects" on page 6070
Logical property information for compound elements and embedded simple types.

"Physical properties for deprecated message model objects" on page 6075
CWF, XML, and TDS format physical properties for deprecated objects.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

Compound element properties:

Logical, CWF, XML, TDS, and document properties for a compound element.

- "Compound element logical properties" on page 6070
- "Compound element CWF properties" on page 6077
- "Compound element XML properties" on page 6081
- "Compound element TDS properties" on page 6086
- "Documentation properties for all message set objects" on page 5413

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Compound element logical properties" on page 6070
The logical properties of a compound element include properties that specify the number of occurrences of the compound element.

"Compound element CWF properties" on page 6077
The properties, and their permissible values, vary according to the object type.

"Compound element XML properties" on page 6081
The properties, and their permissible values, vary according to the object type.

"Compound element TDS properties" on page 6086
The properties, and their permissible values, vary according to the object type.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

Compound element logical properties:

The logical properties of a compound element include properties that specify the number of occurrences of the compound element.

Property	Type	Meaning
Name	String	Specify a name for the object when you create it. Names can consist of alphanumeric characters, including the letters *A* through *Z*, *a* through *z*, and the digits *0* through *9*. They might also include the following punctuation characters; • - the hyphen • _ the underscore • . the period Names can start only with a letter or the underscore character, and not with a number, hyphen, or period. Names that begin with `xml`, or any variant of these characters (for example `XmL`), are reserved by the XML standards specification. Further details of naming conventions and allowable characters can be found in the Extensible Markup Language (XML) specification that can be found on the World Wide Web Consortium (W3C) Web site.
Namespace	Enumerated type	Namespaces are a simple method for qualifying element and attribute names by associating them with namespaces identified by URI references. If <no target namespace> is displayed, a namespace has not been set for this object. If the property is inactive, the message set has not been configured to support namespaces. Where the property is active, namespaces that are available for selection are displayed in the drop-down list.

Property	Type	Meaning
Nillable	Check box	Select this option if you want the element to be able to be defined as null. A null value is distinct from being empty, when the element contains no data.
Abstract	Check box	Select this option if you do not want the element to appear in the message, but require one of the members of its substitution group to appear in its place.

Value

Property	Type	Meaning
Default	Button and String	This property provides the default value for an element or attribute. **XMLNSC domain** When parsing with validation enabled, default values are applied to missing attributes and empty elements as required by the XML Schema 1.0 specification. When writing, elements or attributes that are missing from the message tree are not automatically added to the output XML bit stream, even if they have default values. If missing elements or attributes are required, the message tree can be serialized and then re-parsed with validation enabled. **MRM (CWF and TDS physical formats)** When writing a fixed-length portion of a message (CWF or fixed-length TDS), if an attribute or element is missing from the message tree, the default value is inserted into the bit stream so that the message structure is preserved. **MRM (XML physical format)** No support for default values **Other domains** No support for default values.
Fixed	Button and String	This property provides the fixed value for an element or attribute. **XMLNSC domain** When parsing with validation enabled, if an attribute or element is present, the value is validated against the fixed value. If the values are not equal, a validation error is signalled. Also, when parsing with validation enabled, fixed values are applied to missing attributes and empty elements as required by the XML Schema 1.0 specification. When writing, elements or attributes that are missing from the message tree are not automatically added to the output XML bit stream, even if a fixed value has been specified. If missing elements or attributes are required, the message tree can be serialized and then re-parsed with validation enabled. **MRM (CWF and TDS physical formats)** When writing a fixed-length portion of a message (CWF or fixed-length TDS), if an attribute or element is missing from the message tree, the fixed value is inserted into the bit stream so that the message structure is preserved. **MRM (XML physical format)** No support for fixed values **Other domains** No support for fixed values.

Property	Type	Meaning
Interpret Value As	Enumerated type	Specify if values stored within this object must be interpreted as having significance for the parser and, if so, the type of interpretation that must occur. Select one of the following options: • None This value is the default value, and indicates that the element or attribute does not have a key value associated with it. • MessageSetIdentity. Specifies that the value of the element or attribute corresponds to the identifier, name, or alias (in that priority order) that is associated with the message set where all subsequent embedded messages that are descendents of the enclosing message are defined. This value remains in force unless a new element or attribute MessageSetIdentity field is encountered which resets the MessageSetIdentity value. • MessageIdentity. Specifies that the value of the element or attribute corresponds to the name or alias (in that priority order) that is associated with a message, and acts as an identifier for subsequent embedded messages which are the immediate children of the enclosing message. This identity applies until a new element or attribute MessageIdentity field is encountered at the same level in the tree. The embedded message can be defined in either the current message set, or in a message set identified by using a MessageSetIdentity. **Note:** This property is applicable only when the type of the object is derived from xsd:string.

Occurrences

Property	Type	Meaning
Min Occurs	Integer	Specify the minimum number of times that the object can repeat. The default value is 1. If the value is set to 0, the object is optional. With the exception of `Max Occurs` being set to -1, if a value is set for `Min Occurs`, it must be less than or equal to the value in `Max Occurs`.
Max Occurs	Integer	Specify the maximum number of times that the object can repeat. The default value is 1. If this property is not set, the object cannot occur more than once. If this property is set to 0, it is interpreted as if the object does not exist in the message. It can also be set to -1, to indicate that the limit is unbounded and there is no maximum to the number of occurrences.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Compound element properties" on page 6091
Logical, CWF, XML, TDS, and document properties for a compound element.

Compound element value constraint properties:

The value constraints for a compound element.

The properties for compound element value constraints are identical to simple type value constraints. See "Simple type logical value constraints" on page 5450 for details.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Compound element properties" on page 6091
Logical, CWF, XML, TDS, and document properties for a compound element.

"Compound element logical properties" on page 6070
The logical properties of a compound element include properties that specify the number of occurrences of the compound element.

"Simple type logical value constraints" on page 5450
The properties, and their permissible values, vary according to the object type.

Compound element CWF properties:

The properties, and their permissible values, vary according to the object type.

The properties that are displayed on the object page, and the values that those properties can take, can vary according to the type of the object. For example, the properties for type *string* are different from those of type *Boolean*. Select the link for the object type from the following table.

Type of object	Properties
Binary types	• ComIbmMrm _BaseValueBinary
Boolean types	• ComIbmMrm _BaseValueBoolean
DateTime types	• ComIbmMrm _BaseValueDateTime

Type of object	Properties
Decimal types	• ComIbmMrm _BaseValueDecimal
Float types	• ComIbmMrm _BaseValueFloat
Integer types	• ComIbmMrm _BaseValueInt
String types	• ComIbmMrm _BaseValueString

`

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM Custom Wire Format" on page 1214
Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not separated by delimiters.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

"Working with data structures" on page 2930
You can create a message definition file in a message set by importing from XML Schema, XML DTD, SCA import or export components, IBM supplied messages, WSDL definitions, IDL files, C header files, and COBOL copybooks. This topic area describes how to import from these data structures using the command line or the WebSphere Message Broker Toolkit.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Compound element properties" on page 6091
Logical, CWF, XML, TDS, and document properties for a compound element.

CWF properties for compound element binary types:

Physical representation, byte alignment, and occurrences.

The Custom Wire Format properties described here apply to:

- Objects: Compound elements

Physical representation

Property	Type	Meaning
Length	Button and Integer	If you have selected the length to be defined by *Length*, enter the number of length units for the element. The minimum value that you can specify is 1. The maximum value that you can specify is 2147483647. The default value is empty (not set).
Length Reference	Button and Enumerated type	If you have selected the length to be defined by *Length Reference*, select the name of the integer object that specifies the length of this object. Make your selection from the displayed list of integer objects that are defined as siblings of the current object, and occur before it in the structure of the message. For information about reordering elements, see "Reordering objects" on page 2900.

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. The length is given in bytes. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value for the *`Length Count`* or *`Length Reference`* property. The default is `Bytes`.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to the first instance only.

Property	Type	Meaning
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to all instances.

Occurrences

Property	Type	Meaning
Repeat Reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the displayed list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, there are no integer objects before this one in the message structure. If a *`Repeat Reference`* is specified, it overrides any setting for the *`Max Occurs`* logical property when parsing and writing the message, but not for validation of the message.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Compound element properties" on page 6091
Logical, CWF, XML, TDS, and document properties for a compound element.

"Compound element CWF properties" on page 6077
The properties, and their permissible values, vary according to the object type.

CWF properties for compound element Boolean types:

Byte alignment and occurrences.

The Custom Wire Format properties described here apply to:

- Objects: Compound elements

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to all instances.

Occurrences

Property	Type	Meaning
Repeat Reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the displayed list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, there are no integer objects before this one in the message structure. If a *`Repeat Reference`* is specified, it overrides any setting for the *`Max Occurs`* logical property when parsing and writing the message, but not for validation of the message.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Compound element properties" on page 6091
Logical, CWF, XML, TDS, and document properties for a compound element.

"Compound element CWF properties" on page 6077
The properties, and their permissible values, vary according to the object type.

CWF properties for compound element dateTime types:

Physical representation, null values, byte alignment, and occurrences.

The Custom Wire Format properties described here apply to:

- Objects: Compound elements

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select one from the displayed list: • `Fixed Length String`. The element's length is determined by other length properties as follows. • `Length Encoded String 1`. The first byte of the element contains the length of the string following the length byte in length units. The maximum length of a `Length Encoded String 1` element is 255 length units. • `Length Encoded String 2`. The element's first 2 bytes contain the length of the string following the two length bytes in length units. The maximum length of a `Length Encoded String 2` element is 65535 length units. The two length bytes are in the format of the WebSphere MQ queue manager Encoding. • `Null Terminated String`. The string ends with the hexadecimal NULL character, `X'00'`. • `Packed Decimal`. The dateTime is coded as a Packed Decimal number. It is valid only if the *`DateTime Format`* property represents numeric-only data. • `Binary`. The dateTime is encoded as a binary sequence of bytes. If you select this option, the range of symbols that you can specify for the Format String property is less than the range of symbols you can specify if you select a string option (see "DateTime formats" on page 6310 for details). • `Time Seconds`. This value supports C time_t and Java Date and Time objects. It is valid only if the *`DateTime Format`* property represents numeric-only data. • `Time Milliseconds`. This value supports C time_t and Java Date and Time objects. It is valid only if the *`DateTime Format`* property represents numeric-only data. The default value is `fixed length string`.
DateTime Format	String	Specify a template for date and time. The default dateTime format is dependent on the logical type of the object. For information about the defaults for the dateTime format according to the logical type, see "DateTime defaults by logical type" on page 6320. See "DateTime formats" on page 6310 for details of date and time formats.
Length	Button and Integer	If you have selected a *`Physical Type`* of `Fixed Length String`, `Packed Decimal`, or `Binary`, and have selected the length to be defined by *`Length`*, enter the number of length units for the element. The minimum value that you can specify is 1 for all three physical types. The maximum value that you can specify is `256` for `Fixed Length String`, `10` for `Packed Decimal`, and `2147483647` for `Binary`. The default value is 0 (zero).
Length Reference	Button and Enumerated type	If you have selected the length to be defined by *`Length Reference`*, select the name of the integer object that specifies the length of this object. Make your selection from the displayed list of integer objects that are defined as siblings of the current object, and occur before it in the structure of the message. For information about reordering elements, see "Reordering objects" on page 2900.

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. The length is given in bytes. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value for the *`Length Count`* or *`Length Reference`* property. The default is `Bytes`.
Signed	Check box	Specify whether the value is signed. This property is applicable only if the *`Physical type`* property is `Packed Decimal`. By default, this check box is cleared, which indicates that the value is not signed.
Justification	Enumerated type	If you have set the *`Physical Type`* property to `Fixed Length String`, select `Left Justify` (the default value) or `Right Justify` from the list. If you have selected another value for *`Physical Type`*, this property is inactive.

Property	Type	Meaning
Padding Character	String	If you have set the *Physical Type* property to Fixed Length String, and the *Justification* property is either Left Justify or Right Justify, this property is applicable. When writing an output message, use the padding character to fill out the remaining character positions when the string length is less than the length implied by the *Length* or *Length Reference* property. Whether the string is padded from the left or the right is governed by the *Justification* property. When parsing an input message, the padding character is trimmed from the end of the string. Whether the string is trimmed from the left or the right is governed by the *Justification* property. Specify this character in one of the following ways: • Select NUL, '0', or SPACE from the displayed list. • Enter a character between quotation marks; for example, "c" or 'c', where c is any alphanumeric character. • Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form 0xYY, where YY is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, it is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the displayed list. The option that you select determines the value that you must set for the property *Encoding Null Value*: • NULLPadFill. This option is valid only if *Physical Type* is Fixed Length String. The field is filled with the value specified by the *Padding Character*. The default value. • NULLLogicalValue. The *Encoding Null Value* property is first converted to an actual value, and rendered in the way specified for the field. • NULLLiteralValue. This specifies that *Encoding Null Value* contains a value that is directly substituted as if it is a string. Use this option when the value you have set for *Encoding Null Value* to specify a null date is not a dateTime value, or does not conform to the standard dateTime format yyyy-MM-dd 'T'HH:mm:ss. • NULLLiteralFill. The field is filled with the value specified by the *Encoding Null Value*. *Encoding Null Value* must resolve to a single character.
Encoding Null Value	String	If you set the *Encoding Null* property to NULLPadFill, this property is disabled. If you set the *Encoding Null* property to NULLLogicalValue, you must set this property to an ISO8601 dateTime format. These formats are described in "DateTime as string data" on page 6311. For example, specify a value conforming to yyyy-MM-dd'T'HH:mm:ss such as 1970-12-01. If you set the *Encoding Null* property to NULLLiteralValue, you can enter any value that is the same length as the field. If you set the *Encoding Null* property to NULLLiteralFill, the value must resolve to a single character. Set the character in one of the following ways: • Select SPACE, NUL, 0x00 or 0xFF from the displayed list • Enter a character between quotation marks, for example 'c' or "c", where c is any alphanumeric character. • Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value. • Enter a decimal character code in the form YY where YY is a decimal value. • Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal format.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • 1 Bytes. The default value. • 2 Bytes • 4 Bytes • 8 Bytes • 16 Bytes

Property	Type	Meaning
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to all instances.

Occurrences

Property	Type	Meaning
Repeat Reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the displayed list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, there are no integer objects before this one in the message structure. If a *`Repeat Reference`* is specified, it overrides any setting for the *`Max Occurs`* logical property when parsing and writing the message, but not for validation of the message.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Compound element properties" on page 6091
Logical, CWF, XML, TDS, and document properties for a compound element.

"Compound element CWF properties" on page 6077
The properties, and their permissible values, vary according to the object type.

CWF properties for compound element decimal types:

Physical representation, null values, byte alignment, and occurrences.

The Custom Wire Format properties described here apply to:

- Objects: Compound elements

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select one from the displayed list: • `Integer`. This equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • `Packed Decimal`. Equates to the COMP-3 data type in COBOL. • `External Decimal`. Equates to the data type PIC 9 USAGE DISPLAY in COBOL. The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager: • Elements that have *`Physical Type`* set to `Integer` and `Packed Decimal` are represented in the appropriate WebSphere MQ Encoding value. • Elements that have *`Physical Type`* set to `External Decimal` are represented in the WebSphere MQ CodedCharSetId value.
Length	Integer	Enter the number of bytes to specify the element length: • If you set the *`Physical Type`* to `Integer`, select `1`, `2`, or `4` (the default) from the displayed list. • If you set the *`Physical Type`* to `Packed Decimal`, enter a value between 1 and 10. • If you set the *`Physical Type`* to `Extended Decimal`, enter a value between 1 and 256. (Numbers greater than the maximum COBOL PICTURE clause of 18 are assumed to be 18.)

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value for the *`Length Count`*. The default is `Bytes`.
Signed	Check box	Select (the default) or clear this property. This property is used with *`Sign Orientation`*.
Sign EBCDIC Custom Overpunched	Check box	If the *`Physical Type`* is set to `External Decimal` and the *`Signed EBCDIC Custom`* property is set, this indicates that the Sign EBCDIC Custom Overpunched representation is to be used within an ASCII environment. If this check box is not selected (the default), the Sign ASCII representation is used. The setting of the *`Sign EBCDIC Custom Overpunched`* check box is appropriate only if the *`Sign Orientation`* property is set to `Leading` or `Trailing` (indicating that the element/attribute has an embedded sign representation). The check box is not available if the element/attribute is unsigned (for example, if the *`Signed`* check box is not set).

Property	Type	Meaning
Sign Orientation	Enumerated type	If you have set *Physical Type* to External Decimal and you have selected *Signed*, choose from the following options that represent the COBOL options for displaying numeric data: • Leading Overpunched. This option sets a bit in the first byte if the number is negative. No setting is made if the number is positive. For example, the ASCII hexadecimal representation of the number 22 is x'3232'. Using this option, the number +22 would be x'3232' and the number -22 would be x'7232'. The default value. • Leading Separate. This option sets the first byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte. • Trailing Overpunched. This option sets a bit in the last byte if the number is negative. No setting is made if the number is positive. Using this option, the number +22 would be x'3232' and the number -22 would be x'3272'. • Trailing Separate. This option sets the last byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte. If you have set *Physical Type* to any other value, the value Not Applicable is set for you.
Virtual Decimal Point	Integer	Specify the number of places to the left (for a positive value) or right (for a negative value) that a decimal point is to be moved from its assumed position. For example, a decimal element containing 1234 with a Virtual Decimal value of 3 is 1.234, equivalent to 'V' or 'P' in a COBOL picture clause. There is no C equivalent
Justification	Enumerated type	If you have set the *Physical Type* property to External Decimal, select Left Justify or Right Justify (the default value) from the list. If you have selected another value for *Physical Type*, this property is inactive.

Property	Type	Meaning
Padding Character	String	The padding character is used to fill out the remaining character positions when the string length is less than the specified string size. If you have set the *Physical Type* property to Extended Decimal, and the *Justification* property is either Left Justify or Right Justify, specify this character in one of the following ways: • Select NUL, '0', or SPACE from the displayed list. • Enter a character between quotation marks; for example, "c" or 'c', where c is any alphanumeric character. • Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form 0xYY, where YY is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the displayed list: • NULLPadFill. This option is valid only if *Physical Type* is External Decimal. The field is filled with the value specified by the *Padding Character*. *Encoding Null Value* must be set to an empty string. • NULLLogicalValue. The *Encoding Null Value* is transformed to match the required format for the field. The default value. • NULLLiteralValue. The *Encoding Null Value* is directly substituted as if it is a string. You can specify a nonnumeric value for *Encoding Null Value*. • NULLLiteralFill. The field is filled with the value specified by the *Encoding Null Value*. *Encoding Null Value* must resolve to a single character.

Property	Type	Meaning
Encoding Null Value	String	The use of this property depends on the *Encoding Null* property, except for NULLLiteralFill. The default value is zero. If you set the *Encoding Null* property to NULLLiteralFill, the value must resolve to a single character. Set the character in one of the following ways: • Select SPACE, NUL, 0x00 or 0xFF from the displayed list • Enter a character between quotation marks, for example 'c' or "c", where c is any alphanumeric character. • Enter a hexadecimal character in the form 0xYY where YY is a hexadecimal value. • Enter a decimal character code in the form YY where YY is a decimal value. • Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal format.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • 1 Bytes. The default value. • 2 Bytes • 4 Bytes • 8 Bytes • 16 Bytes
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to all instances.

Occurrences

Property	Type	Meaning
Repeat Reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the displayed list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* logical property when parsing and writing the message, but not for validation of the message.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Compound element properties" on page 6091
Logical, CWF, XML, TDS, and document properties for a compound element.

"Compound element CWF properties" on page 6077
The properties, and their permissible values, vary according to the object type.

CWF properties for compound element float types:

Physical representation, null values, byte alignment, and occurrences.

The Custom Wire Format properties described here apply to:

- Objects: Compound elements

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select one from the displayed list: • `Integer`. This equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • `Float`. This equates to the data type FLOAT or DOUBLE in C or the COMP-1 or COMP-2 data type in COBOL and is the default value. • `Packed Decimal`. Equates to the COMP-3 data type in COBOL. • `External Decimal`. Equates to the data type PIC 9 USAGE DISPLAY in COBOL. The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager: • Elements that have *`Physical Type`* set to `Integer`, `Packed Decimal`, and `Float` are represented in the appropriate WebSphere MQ Encoding value. • Elements that have *`Physical Type`* set to `External Decimal` are represented in the WebSphere MQ CodedCharSetId value.
Length	Integer	Enter the number of bytes to specify the element length: • If you set the *`Physical Type`* to `Float`, select a value from the displayed list. The default value is 8. • If you set the *`Physical Type`* to `Integer`, select `1`, `2`, or `4` (the default) from the displayed list. • If you set the *`Physical Type`* to `Packed Decimal`, enter a value between 1 and 10. • If you set the *`Physical Type`* to `Extended Decimal`, enter a value between 1 and 256. (Numbers greater than the maximum COBOL PICTURE clause of 18 are assumed to be 18.)

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value for the *`Length Count`*. The default is `Bytes`.
Signed	Check box	Select or clear (unsigned, the default) this property. If you have set *`Physical Type`* to `Float`, this is selected. This property is used with *`Sign Orientation`*.
Sign EBCDIC Custom Overpunched	Check box	If the *`Physical Type`* is set to `External Decimal` and the *`Signed EBCDIC Custom`* property is set, this indicates that the Sign EBCDIC Custom Overpunched representation is to be used within an ASCII environment. If this check box is not selected (the default), the Sign ASCII representation is used. The setting of the *`Sign EBCDIC Custom Overpunched`* check box is appropriate only if the *`Sign Orientation`* property is set to `Leading` or `Trailing` (indicating that the element/attribute has an embedded sign representation). The check box is not available if the element/attribute is unsigned (for example, if the *`Signed`* check box is not set).

Property	Type	Meaning
Sign Orientation	Enumerated type	If you have set *Physical Type* to External Decimal and you have selected *Signed*, choose from the following options that represent the COBOL options for displaying numeric data: • Leading Overpunched. This option sets a bit in the first byte if the number is negative. No setting is made if the number is positive. For example, the ASCII hexadecimal representation of the number 22 is x'3232'. Using this option, the number +22 would be x'3232' and the number -22 would be x'7232'. The default value. • Leading Separate. This option sets the first byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte. • Trailing Overpunched. This option sets a bit in the last byte if the number is negative. No setting is made if the number is positive. Using this option, the number +22 would be x'3232' and the number -22 would be x'3272'. • Trailing Separate. This option sets the last byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte. If you have set *Physical Type* to any other value, the value Not Applicable is set for you.
Virtual Decimal Point	Integer	Specify the number of places to the left (for a positive value) or right (for a negative value) that a decimal point is to be moved from its assumed position. For example, a float element containing 1234 with a Virtual Decimal value of 3 is 1.234. This property is not applicable if you have set *Physical Type* to Float.
Justification	Enumerated type	If you have set the *Physical Type* property to External Decimal, select Left Justify or Right Justify (the default value) from the list. If you have selected another value for *Physical Type*, this property is inactive.

Property	Type	Meaning
Padding Character	String	The padding character is used to fill out the remaining character positions when the string length is less than the specified string size. If you have set the *Physical Type* property to Extended Decimal, and the *Justification* property is either Left Justify or Right Justify, specify this character in one of the following ways: • Select NUL, '0', or SPACE from the displayed list. • Enter a character between quotation marks; for example, "c" or 'c', where c is any alphanumeric character. • Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form 0xYY, where YY is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the displayed list: • NULLPadFill. This option is valid only if *Physical Type* is External Decimal. The field is filled with the value specified by the *Padding Character*. *Encoding Null Value* must be set to an empty string. • NULLLogicalValue. The *Encoding Null Value* is transformed to match the required format for the field. The default value. • NULLLiteralValue. The *Encoding Null Value* is directly substituted as if it is a string. You can specify a nonnumeric value for *Encoding Null Value*. • NULLLiteralFill. The field is filled with the value specified by the *Encoding Null Value*. *Encoding Null Value* must resolve to a single character.

Property	Type	Meaning
Encoding Null Value	String	The use of this property depends on the *Encoding Null* property, except for NULLLiteralFill. The default value is zero. If you set the *Encoding Null* property to NULLLiteralFill, the value must resolve to a single character. Set the character in one of the following ways: • Select SPACE, NUL, 0x00 or 0xFF from the displayed list • Enter a character between quotation marks, for example 'c' or "c", where c is any alphanumeric character. • Enter a hexadecimal character in the form 0xYY where YY is a hexadecimal value. • Enter a decimal character code in the form YY where YY is a decimal value. • Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal format.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • 1 Bytes. The default value. • 2 Bytes • 4 Bytes • 8 Bytes • 16 Bytes
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to all instances.

Occurrences

Property	Type	Meaning
Repeat Reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the displayed list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* logical property when parsing and writing the message, but not for validation of the message.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Compound element properties" on page 6091
Logical, CWF, XML, TDS, and document properties for a compound element.

"Compound element CWF properties" on page 6077
The properties, and their permissible values, vary according to the object type.

CWF properties for compound element integer types:

Physical representation, null values, byte alignment, and occurrences.

The Custom Wire Format properties described here apply to:

- Objects: Compound elements

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select one from the displayed list: • `Integer`. This equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • `Packed Decimal`. Equates to the COMP-3 data type in COBOL. • `External Decimal`. Equates to the data type PIC 9 USAGE DISPLAY in COBOL. The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager: • Elements that have *`Physical Type`* set to `Integer` and `Packed Decimal` are represented in the appropriate WebSphere MQ Encoding value. • Elements that have *`Physical Type`* set to `External Decimal` are represented in the WebSphere MQ CodedCharSetId value.
Length	Integer	Enter the number of bytes to specify the element length: • If you set the *`Physical Type`* to `Integer`, select `1`, `2`, or `4` (the default) from the displayed list. • If you have set *`Physical Type`* to `Packed Decimal`, enter a value between 1 and 6. • If you have set *`Physical Type`* to `Extended Decimal`, enter a value between 1 and 11.
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value for the *`Length Count`*. The default is `Bytes`.
Signed	Check box	Select (the default) or clear this property. This property is used with *`Sign Orientation`*.

Property	Type	Meaning
Sign EBCDIC Custom Overpunched	Check box	If the *Physical Type* is set to External Decimal and the *Signed EBCDIC Custom* property is set, this indicates that the Sign EBCDIC Custom Overpunched representation is to be used within an ASCII environment. If this check box is not selected (the default), the Sign ASCII representation is used. The setting of the *Sign EBCDIC Custom Overpunched* check box is appropriate only if the *Sign Orientation* property is set to Leading or Trailing (indicating that the element/attribute has an embedded sign representation). The check box is not available if the element/attribute is unsigned (for example, if the *Signed* check box is not set).
Sign Orientation	Enumerated type	If you have set *Physical Type* to External Decimal and you have selected *Signed*, choose from the following options that represent the COBOL options for displaying numeric data: • Leading Overpunched. This option sets a bit in the first byte if the number is negative. No setting is made if the number is positive. For example, the ASCII hexadecimal representation of the number 22 is x'3232'. Using this option, the number +22 would be x'3232' and the number -22 would be x'7232'. The default value. • Leading Separate. This option sets the first byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte. • Trailing Overpunched. This option sets a bit in the last byte if the number is negative. No setting is made if the number is positive. Using this option, the number +22 would be x'3232' and the number -22 would be x'3272'. • Trailing Separate. This option sets the last byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte. If you have set *Physical Type* to any other value, the value Not Applicable is set for you.
Justification	Enumerated type	If you have set the *Physical Type* property to External Decimal, select Left Justify or Right Justify (the default value) from the list. If you have selected another value for *Physical Type*, this property is inactive.

Property	Type	Meaning
Padding Character	String	The padding character is used to fill out the remaining character positions when the string length is less than the specified string size. If you have set the *Physical Type* property to Extended Decimal, and the *Justification* property is either Left Justify or Right Justify, specify this character in one of the following ways: • Select NUL, '0', or SPACE from the displayed list. • Enter a character between quotation marks; for example, "c" or 'c', where c is any alphanumeric character. • Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form 0xYY, where YY is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the displayed list: • NULLPadFill. This option is valid only if *Physical Type* is External Decimal. The field is filled with the value specified by the *Padding Character*. *Encoding Null Value* must be set to an empty string. • NULLLogicalValue. The *Encoding Null Value* is transformed to match the required format for the field. The default value. • NULLLiteralValue. The *Encoding Null Value* is directly substituted as if it is a string. You can specify a nonnumeric value for *Encoding Null Value*. • NULLLiteralFill. The field is filled with the value specified by the *Encoding Null Value*. *Encoding Null Value* must resolve to a single character.

Property	Type	Meaning
Encoding Null Value	String	The use of this property depends on the *Encoding Null* property, except for NULLLiteralFill. The default value is zero. If you set the *Encoding Null* property to NULLLiteralFill, the value must resolve to a single character. Set the character in one of the following ways: • Select SPACE, NUL, 0x00 or 0xFF from the displayed list • Enter a character between quotation marks, for example 'c' or "c", where c is any alphanumeric character. • Enter a hexadecimal character in the form 0xYY where YY is a hexadecimal value. • Enter a decimal character code in the form YY where YY is a decimal value. • Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal format.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • 1 Bytes. The default value. • 2 Bytes • 4 Bytes • 8 Bytes • 16 Bytes
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to all instances.

Occurrences

Property	Type	Meaning
Repeat Reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the displayed list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* logical property when parsing and writing the message, but not for validation of the message.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Compound element properties" on page 6091
Logical, CWF, XML, TDS, and document properties for a compound element.

"Compound element CWF properties" on page 6077
The properties, and their permissible values, vary according to the object type.

CWF properties for compound element string types:

Physical representation, null values, byte alignment, and occurrences.

The Custom Wire Format properties described here apply to:

- Objects: Compound elements

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select one from the displayed list: • `Fixed Length String`. The element's length is determined by other length properties as follows. • `Length Encoded String 1`. The first byte of the element contains the length of the string following the length byte in length units. The maximum length of a `Length Encoded String 1` element is 255 length units. • `Length Encoded String 2`. The element's first 2 bytes contain the length of the string following the two length bytes in length units. The maximum length of a `Length Encoded String 2` element is 65535 length units. The two length bytes are in the format of the WebSphere MQ queue manager Encoding. • `Null Terminated String`. The string ends with the hexadecimal NULL character, `X'00'`. The default is `Fixed Length String`.
Length	Button and Integer	If you have selected a *`Physical Type`* of `Fixed Length String` or `Binary`, and have selected the length to be defined by *`Length`*, enter the number of length units for the element. The minimum value that you can specify is `0` (zero), the maximum value that you can specify is `2147483647` The default value is `0` (zero).
Length Reference	Button and Enumerated type	If you have selected the length to be defined by *`Length Reference`*, select the name of the integer object that specifies the length of this object. Make your selection from the displayed list of integer objects that are defined as siblings of the current object, and occur before it in the structure of the message. For information about reordering elements, see “Reordering objects” on page 2900.

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. The length is given in bytes. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value for the *`Length Count`* or *`Length Reference`* property. The default is `Bytes`.
Justification	Enumerated type	If you have set the *`Physical Type`* property to `Fixed Length String`, select `Left Justify` (the default value) or `Right Justify` from the list. If you have selected another value for *`Physical Type`*, this property is inactive.

Property	Type	Meaning
Padding Character	String	If you have set the *Physical Type* property to `Fixed Length String`, and the *Justification* property is either `Left Justify` or `Right Justify`, this property is applicable. When writing an output message, use the padding character to fill out the remaining character positions when the string length is less than the length implied by the *Length* or *Length Reference* property. Whether the string is padded from the left or the right is governed by the *Justification* property. When parsing an input message, the padding character is trimmed from the end of the string. Whether the string is trimmed from the left or the right is governed by the *Justification* property. Specify this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the displayed list. • Enter a character between quotation marks; for example, `"c"` or `'c'`, where c is any alphanumeric character. • Enter a Unicode value in the form `U+xxxx` where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form `0xYY`, where `YY` is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, it is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the displayed list: • `NULLPadFill`. This is valid only if *`Physical Type`* is `Fixed Length String`. The field is filled with the value specified by the *`Padding Character`*. *`Encoding Null Value`* must be set to an empty string. • `NULLLogicalValue`. The *`Encoding Null Value`* is transformed to match the required format for the field. The default value. • `NULLLiteralValue`. The *`Encoding Null Value`* is directly substituted as if it is a string. • `NULLLiteralFill`. The field is filled with the value specified by the *`Encoding Null Value`*. *`Encoding Null Value`* must resolve to a single character.
Encoding Null Value	STRING	The use of this property depends on the *`Encoding Null`* property. If specified, its length must be equal to the length of the string element, except for `NULLLiteralFill`. The default value is empty (not set). If you set the *`Encoding Null`* property to `NULLLiteralFill`, the value must resolve to a single character. Set the character in one of the following ways: • Select `SPACE`, `NUL`, `0x00` or `0xFF` from the displayed list • Enter a character between quotation marks, for example `'c'` or `"c"`, where `c` is any alphanumeric character. • Enter a hexadecimal character code in the form `0xYY` where `YY` is a hexadecimal value. • Enter a decimal character code in the form `YY` where `YY` is a decimal value. • Enter a Unicode value in the form `U+xxxx` where `xxxx` is a Unicode value specified in hexadecimal format.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to all instances.

Occurrences

Property	Type	Meaning
Repeat Reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the displayed list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* logical property when parsing and writing the message, but not for validation of the message.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Compound element properties" on page 6091
Logical, CWF, XML, TDS, and document properties for a compound element.

"Compound element CWF properties" on page 6077
The properties, and their permissible values, vary according to the object type.

Compound element XML properties:

The properties, and their permissible values, vary according to the object type.

The properties that are displayed on the object page, and the values that those properties can take, can vary according to the type of the object. For example, the properties for type *string* are different from those of type *Boolean*. Select the link for the object type from the following table.

Type of object	Properties
Binary types	• ComIbmMrm _BaseValueBinary
Boolean types	• ComIbmMrm _BaseValueBoolean
DateTime types	• ComIbmMrm _BaseValueDateTime
Decimal types	• ComIbmMrm _BaseValueDecimal
Float types	• ComIbmMrm _BaseValueFloat
Integer types	• ComIbmMrm _BaseValueInt
String types	• ComIbmMrm _BaseValueString

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Compound element properties" on page 6091
Logical, CWF, XML, TDS, and document properties for a compound element.

"Compound element logical properties" on page 6070
The logical properties of a compound element include properties that specify the number of occurrences of the compound element.

"Compound element CWF properties" on page 6077
The properties, and their permissible values, vary according to the object type.
"Compound element TDS properties" on page 6086
The properties, and their permissible values, vary according to the object type.
"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

XML wire format properties for compound element binary types:

Field identification and physical representation.

The XML Wire Format properties described here apply to:

- Objects: Compound elements

Field identification

A number of the following properties only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated type	Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list: • `XMLElement`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their *XML Name*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object , you must set their *XML Name* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in *ID Attribute Name* and a value as specified in *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in *Value Attribute Name*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *ID Attribute Name*. The value is the value of *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 6262 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
XML Name	String	Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message). This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name. You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name. If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).
ID Attribute Name	String	Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
ID Attribute Value	String	Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

Physical representation

Property	Type	Meaning
Encoding	String	Select one of the following values from the drop-down list: : • `CDatahex` (the default). Hexadecimal values in this field are specified with the CDATA qualifier, for example <e1><![CDATA[62]]></e1> • `hex`. Hexadecimal values in this field are specified as digits only, for example <e1>62</e1>. • `base64`. Values in this field are specified as digits only, coded in base 64.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Compound element properties" on page 6091
Logical, CWF, XML, TDS, and document properties for a compound element.

"Compound element XML properties" on page 6081
The properties, and their permissible values, vary according to the object type.

XML wire format properties for compound element Boolean types:

Field identification.

The XML Wire Format properties described here apply to:

- Objects: Compound elements

Field identification

A number of the following properties only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated type	Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list: • `XMLElement`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their *`XML Name`*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object , you must set their *`XML Name`* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in *`ID Attribute Name`* and a value as specified in *`ID Attribute Value`*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *`XML Name`*, *`ID Attribute Name`*, *`ID Attribute Value`* to the same values: – You must also set *`Value Attribute Name`* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in *`Value Attribute Name`*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *`ID Attribute Name`*. The value is the value of *`ID Attribute Value`*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *`XML Name`*, *`ID Attribute Name`*, *`ID Attribute Value`* to the same values: – You must also set *`Value Attribute Name`* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 6262 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
XML Name	String	Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message). This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name. You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name. If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).
ID Attribute Name	String	Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
ID Attribute Value	String	Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Compound element properties" on page 6091
Logical, CWF, XML, TDS, and document properties for a compound element.

"Compound element XML properties" on page 6081
The properties, and their permissible values, vary according to the object type.

XML wire format properties for compound element dateTime types:

Field identification and physical representation.

The XML Wire Format properties described here apply to:

- Objects: Compound elements

Field identification

A number of the following properties only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated type	Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list: • `XMLElement`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their *XML Name*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object , you must set their *XML Name* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in *ID Attribute Name* and a value as specified in *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in *Value Attribute Name*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *ID Attribute Name*. The value is the value of *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 6262 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
XML Name	String	Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message). This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name. You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name. If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).
ID Attribute Name	String	Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
ID Attribute Value	String	Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

Physical representation

Property	Type	Meaning
DateTime Format	String	Specify a format string that specifies the rendering of the value for dateTime elements. The default dateTime format is dependent on the logical type of the object. For information about the defaults for the dateTime format according to the logical type see “DateTime defaults by logical type” on page 6320. See “DateTime formats” on page 6310 for details of dateTime formats.

Related concepts:

“Message modeling” on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

“The message model” on page 1160
The message model consists of the following components.

“MRM XML physical format: Relationship to the logical model” on page 1251
The MRM XML physical format generally respects all the settings in the logical

model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Compound element properties" on page 6091
Logical, CWF, XML, TDS, and document properties for a compound element.

"Compound element XML properties" on page 6081
The properties, and their permissible values, vary according to the object type.

XML wire format properties for compound element decimal types:

Field identification.

The XML Wire Format properties described here apply to:

- Objects: Compound elements

Field identification

A number of the following properties only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated type	Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list: • `XMLElement`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their *`XML Name`*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object , you must set their *`XML Name`* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in *`ID Attribute Name`* and a value as specified in *`ID Attribute Value`*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *`XML Name`*, *`ID Attribute Name`*, *`ID Attribute Value`* to the same values: – You must also set *`Value Attribute Name`* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in *`Value Attribute Name`*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *`ID Attribute Name`*. The value is the value of *`ID Attribute Value`*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *`XML Name`*, *`ID Attribute Name`*, *`ID Attribute Value`* to the same values: – You must also set *`Value Attribute Name`* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 6262 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
XML Name	String	Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message). This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name. You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name. If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).
ID Attribute Name	String	Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
ID Attribute Value	String	Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Compound element properties" on page 6091
Logical, CWF, XML, TDS, and document properties for a compound element.

"Compound element XML properties" on page 6081
The properties, and their permissible values, vary according to the object type.

XML wire format properties for compound element float types:

Field identification.

The XML Wire Format properties described here apply to:

- Objects: Compound elements

Field identification

A number of the following properties only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated type	Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list: • `XMLElement`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their *XML Name*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object , you must set their *XML Name* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in *ID Attribute Name* and a value as specified in *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in *Value Attribute Name*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *ID Attribute Name*. The value is the value of *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 6262 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
XML Name	String	Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message). This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name. You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name. If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).
ID Attribute Name	String	Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
ID Attribute Value	String	Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Compound element properties" on page 6091
Logical, CWF, XML, TDS, and document properties for a compound element.

"Compound element XML properties" on page 6081
The properties, and their permissible values, vary according to the object type.

XML wire format properties for compound element integer types:

Field identification.

The XML Wire Format properties described here apply to:

- Objects: Compound elements

Field identification

A number of the following properties only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated type	Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list: • `XMLElement`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their *`XML Name`*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object , you must set their *`XML Name`* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in *`ID Attribute Name`* and a value as specified in *`ID Attribute Value`*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *`XML Name`*, *`ID Attribute Name`*, *`ID Attribute Value`* to the same values: – You must also set *`Value Attribute Name`* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in *`Value Attribute Name`*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *`ID Attribute Name`*. The value is the value of *`ID Attribute Value`*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *`XML Name`*, *`ID Attribute Name`*, *`ID Attribute Value`* to the same values: – You must also set *`Value Attribute Name`* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 6262 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
XML Name	String	Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message). This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name. You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name. If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).
ID Attribute Name	String	Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
ID Attribute Value	String	Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Compound element properties" on page 6091
Logical, CWF, XML, TDS, and document properties for a compound element.

"Compound element XML properties" on page 6081
The properties, and their permissible values, vary according to the object type.

XML wire format properties for compound element string types:

Field identification.

The XML Wire Format properties described here apply to:

- Objects: Compound elements

Field identification

A number of the following properties only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated type	Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list: • `XMLElement`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their *XML Name*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object , you must set their *XML Name* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in *ID Attribute Name* and a value as specified in *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in *Value Attribute Name*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *ID Attribute Name*. The value is the value of *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 6262 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
XML Name	String	Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message). This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name. You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name. If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).
ID Attribute Name	String	Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
ID Attribute Value	String	Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Compound element properties" on page 6091
Logical, CWF, XML, TDS, and document properties for a compound element.

"Compound element XML properties" on page 6081
The properties, and their permissible values, vary according to the object type.

Compound element TDS properties:

The properties, and their permissible values, vary according to the object type.

The properties that are displayed on the object page, and the values that those properties can take, can vary according to the type of the object. For example, the properties for type *string* are different from those of type *Boolean*. Select the link for the object type from the following table.

Type of object	Properties
Binary types	• ComIbmMrm _BaseValueBinary
Boolean types	• ComIbmMrm _BaseValueBoolean
DateTime types	• ComIbmMrm _BaseValueDateTime
Decimal types	• ComIbmMrm _BaseValueDecimal
Float types	• ComIbmMrm _BaseValueFloat
Integer types	• ComIbmMrm _BaseValueInt
String types	• ComIbmMrm _BaseValueString

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM TDS format" on page 1221
The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Compound element properties" on page 6091
Logical, CWF, XML, TDS, and document properties for a compound element.

"Compound element logical properties" on page 6070
The logical properties of a compound element include properties that specify the number of occurrences of the compound element.

"Compound element CWF properties" on page 6077
The properties, and their permissible values, vary according to the object type.

"Compound element XML properties" on page 6081
The properties, and their permissible values, vary according to the object type.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

TDS properties for compound element binary types:

Field identification and physical representation.

The TDS properties described here apply to:

- Objects: Compound elements

Field identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the `Data Element Separation` property of the complex type or types in which the object is a child is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, this property must contain a non-empty value. If the object is a complex element, and the `Data Element Separation` property of its parent is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, the property can contain an empty value. The value for this property must be unique for every element in the message set; that is, no two elements in the message set can contain the same value for this property.
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the `Data Element Separation` method is set to `Use Data Pattern` in the complex type. For more details, see "Regular expression syntax" on page 6304.

Property	Type	Meaning
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the `Data Element Separation` property of the parent group or complex type is set to `All Elements Delimited` or `Variable Length Elements Delimited`. A default value is provided if the previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; if the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used. If none of the previous conditions are true, a default is not applied.

Physical representation

Property	Type	Meaning
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no `Length Reference` is specified. The default is dependent on the setting of the message set property `Derive default length from logical type`. If `Derive default length from logical type` is selected, the default value is derived from any length or maxLength value constraint (schema facet) on the object's simple type.
Length Reference	Enumerated type	This property is applicable only if `Physical Type` is `Text`, `Binary`, or `TLOG Specific`. If set, this property takes precedence over any value in the `Length Units` property. Specify the identifier of a sibling integer object, the value of which determines the length of the object in question. The sibling object must be defined before the current object within the message structure. For information about reordering elements, see “Reordering objects” on page 2900.

Related concepts:

“Message modeling” on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

“The message model” on page 1160
The message model consists of the following components.

“MRM TDS format: Relationship to the logical model” on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

“Constructing message models” on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

“Working with MRM message model objects” on page 2870
Add, configure, and delete objects.

Related reference:

“Message model reference information” on page 5366
Reference information in this section can help you develop and configure message

models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Compound element properties" on page 6091
Logical, CWF, XML, TDS, and document properties for a compound element.

"Compound element TDS properties" on page 6086
The properties, and their permissible values, vary according to the object type.

TDS properties for compound element Boolean types:

Field identification and physical representation.

The TDS properties described here apply to:

- Objects: Compound elements

Field identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the `Data Element Separation` property of the complex type or types in which the object is a child is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, this property must contain a non-empty value. If the object is a complex element, and the `Data Element Separation` property of its parent is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, the property can contain an empty value. The value for this property must be unique for every element in the message set; that is, no two elements in the message set can contain the same value for this property.
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the `Data Element Separation` method is set to `Use Data Pattern` in the complex type. For more details, see "Regular expression syntax" on page 6304.
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the `Data Element Separation` property of the parent group or complex type is set to `All Elements Delimited` or `Variable Length Elements Delimited`. A default value is provided if the previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; if the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used. If none of the previous conditions are true, a default is not applied.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the `Messaging Standard` property of the message set is `User Defined Text`, `User Defined Mixed`, `CSV`, or `TLOG`, select one of the following values: • `Text`. The data is in character format. • `Binary`. The data is in bit string format. The default is dependent on the `Messaging Standard` property. For all other `Messaging Standard` values, the `Physical Type` property is set to `Text`.
Length Reference	Enumerated type	This property is applicable only if `Physical Type` is `Text`, `Binary`, or `TLOG Specific`. If set, this property takes precedence over any value in the `Length Units` property. Specify the identifier of a sibling integer object, the value of which determines the length of the object in question. The sibling object must be defined before the current object within the message structure. For information about reordering elements, see "Reordering objects" on page 2900.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Compound element properties" on page 6091
Logical, CWF, XML, TDS, and document properties for a compound element.

"Compound element TDS properties" on page 6086
The properties, and their permissible values, vary according to the object type.

TDS properties for compound element dateTime types:

Field identification, physical representation, and null values.

The TDS properties described here apply to:

- Objects: Compound elements

Field identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the `Data Element Separation` property of the complex type or types in which the object is a child is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, this property must contain a non-empty value. If the object is a complex element, and the `Data Element Separation` property of its parent is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, the property can contain an empty value. The value for this property must be unique for every element in the message set; that is, no two elements in the message set can contain the same value for this property.
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the `Data Element Separation` method is set to `Use Data Pattern` in the complex type. For more details, see "Regular expression syntax" on page 6304.
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the `Data Element Separation` property of the parent group or complex type is set to `All Elements Delimited` or `Variable Length Elements Delimited`. A default value is provided if the previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; if the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used. If none of the previous conditions are true, a default is not applied.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the `Messaging Standard` property of the message set is `User Defined Text`, `User Defined Mixed`, `CSV`, or `TLOG`, select one of the following values: • `Text`. The data is in character format. • `Length Encoded String 1`. The first byte of the data contains the length (in length units) of the data string that follows the length byte. The maximum length of a `Length Encoded String 1` element is 255 length units. • `Length Encoded String 2`. The first two bytes of the data contain the length (in length units) of the data string that follows the two length bytes. The maximum length of a `Length Encoded String 2` element is 65535 length units. • `Null Terminated String`. The data string ends with the hexadecimal NULL character, `X'00'`. • `Packed Decimal`. The data is a packed decimal number that equates to the COMP-3 data type in COBOL. • `Binary`. The data is in bit string format. • `Time Seconds`. This value supports C time_t, and Java Date and Time objects. It is valid only if the `DateTime Format` property represents numeric-only data. • `Time Milliseconds`. This value supports C time_t, and Java Date and Time objects. It is valid only if the `DateTime Format` property represents numeric-only data. The default is dependent on the `Messaging Standard` property. For all other `Messaging Standard` values, the `Physical Type` property is set to `Text`.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no `Length Reference` is specified.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • `Not Applicable` • `Left Justify` • `Right Justify`

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the drop-down list. • Enter a character between quotation marks, for example "c" or 'c', where c is any alphanumeric character. • Enter a hexadecimal character code in the form `0xYY`, where `YY` is a hexadecimal value. • Enter a Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.
DateTime Format	String	Specify a template for date and time. The default DateTime format is dependent on the logical type of the object. For information about the defaults for the dateTime format according to the logical type, see “DateTime defaults by logical type” on page 6320. See “DateTime formats” on page 6310 for details of date and time formats.
Length Reference	Enumerated type	This property is applicable only if `Physical Type` is `Text`, `Binary`, or `TLOG Specific`. If set, this property takes precedence over any value in the `Length Units` property. Specify the identifier of a sibling integer object, the value of which determines the length of the object in question. The sibling object must be defined before the current object within the message structure. For information about reordering elements, see “Reordering objects” on page 2900.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the list: • `NULLPadFill`. This option is valid only for fixed-length objects and is the default value. • `NULLLogicalValue`. The `Encoding Null Value` property is first converted to an actual value, and rendered in the way specified for the field. • `NULLLiteralValue`. This option specifies that `Encoding Null Value` contains a value that is directly substituted as if it is a string. For dateTime elements, use this option if you want to use the `Encoding Null Value` property to test or compare the content of the field in the message. • `NULLLiteralFill`. This option specifies that the field is filled with the value specified by the `Encoding Null Value` property. `Encoding Null Value` must resolve to a single character. The option that you select determines the value that you must set for the property `Encoding Null Value`. For full information about using these options, see "TDS Null handling options" on page 6293.
Encoding Null Value	String	The use of this property depends on the `Encoding Null` property. The default value is zero. If you set the `Encoding Null` property for a dateTime object to `NULLLogicalValue`, the value that you set must be in an ISO8601 dateTime format. These formats are described in "DateTime as string data" on page 6311. For example, specify a value that conforms to the `yyyy-MM-dd'T'HH:mm:ss` format; for example, `1970-12-01`.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Compound element properties" on page 6091
Logical, CWF, XML, TDS, and document properties for a compound element.

"Compound element TDS properties" on page 6086
The properties, and their permissible values, vary according to the object type.

TDS properties for compound element decimal types:

Field identification, physical representation, numeric representation, and null values.

The TDS properties described here apply to:

- Objects: Compound elements

Field identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the `Data Element Separation` property of the complex type or types in which the object is a child is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, this property must contain a non-empty value. If the object is a complex element, and the `Data Element Separation` property of its parent is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, the property can contain an empty value. The value for this property must be unique for every element in the message set; that is, no two elements in the message set can contain the same value for this property.
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the `Data Element Separation` method is set to `Use Data Pattern` in the complex type. For more details, see "Regular expression syntax" on page 6304.
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the `Data Element Separation` property of the parent group or complex type is set to `All Elements Delimited` or `Variable Length Elements Delimited`. A default value is provided if the previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; if the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used. If none of the previous conditions are true, a default is not applied.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the `Messaging Standard` property of the message set is `User Defined Text`, `User Defined Mixed`, `CSV`, or `TLOG`, select one of the following values: • `Text`. The data is in character format. • `Packed Decimal`. The data is a packed decimal number that equates to the COMP-3 data type in COBOL. • `TLOG Specific` - this option can be selected only if the `Message Standard` property of the message set is `TLOG`. This option indicates that the format of the data is specific to the TLOG messaging standard. • `Integer`. Equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • `External Decimal`. The data is a decimal number that equates to the data type PIC 9 USAGE DISPLAY in COBOL. The default is dependent on the `Messaging Standard` property. For all other `Messaging Standard` values, the `Physical Type` property is set to `Text`.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no `Length Reference` is specified. If this property is not set and the message set property `Derive default length from logical type` is selected, and the `Physical type` is 'Character', the default value is derived from any totalDigits value constraint (schema facet) or, if none, any `minInclusive`, `maxInclusive`, `minExclusive`, or `maxExclusive` value constraints (schema facets), on the simple type.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • `Not Applicable` • `Left Justify` • `Right Justify`

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the drop-down list. • Enter a character between quotation marks, for example "c" or 'c', where c is any alphanumeric character. • Enter a hexadecimal character code in the form `0xYY`, where `YY` is a hexadecimal value. • Enter a Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.
Length Reference	Enumerated type	This property is applicable only if `Physical Type` is `Text`, `Binary`, or `TLOG Specific`. If set, this property takes precedence over any value in the `Length Units` property. Specify the identifier of a sibling integer object, the value of which determines the length of the object in question. The sibling object must be defined before the current object within the message structure. For information about reordering elements, see “Reordering objects” on page 2900.

Numeric representation

Property	Type	Meaning
Virtual Decimal Point	Button and Integer	Specify a non-zero integer that represents the position of an implied decimal point within a number, or specify 0, the default, to use the formatting of Float or Decimal numbers as specified by the `Precision` property. If you specify a positive integer, the position of the decimal point is moved left from the right side of the number. For example, if you specify `3`, the decimal value `1234` represents 1.234 If you specify a negative integer, the position of the decimal point is moved right from the right side of the number. For example, if you specify `-3`, the decimal value `1234` represents 1,234,000.
Precision	Button and Integer	This property is applicable only if `Physical Type` is `Text`. It is used if the value of the `Virtual Decimal Point` property is 0, which indicates that the decimal point is present in the data. It deals with truncation, and specifies how many digits are to follow the decimal point. Either specify a number of digits: • If you set `Precision` to 0, data is truncated so that the fractional part is lost. For example, the value `123.45` is truncated to `123`. • If you set `Precision` to a number less than the number of fractional digits, data is truncated. For example, the value `123.4567` is truncated to `123.45` if you set `Precision` to 2. • If you set `Precision` to a number greater than the number of fractional digits, the value is padded with extra zeros. For example, the value `12.345` is padded to `12.34500` if you set `Precision` to 5. Or select one option from the list: • `All Significant Digits - decimal separator only required if fractional digits` (the default): all significant digits are written to the output bit stream, and no decimal separator is written if no fractional digits are present. • `Explicit Decimal Separator - decimal separator always required`: all significant digits are written to the output bit stream and the decimal separator is always included, even when no fractional digits are present. The decimal separator must be present in the input bit stream, even when no fractional digits are present.
Sign Orientation	Enumerated type	The values that you can choose for this property are: • `Leading Separate` • `Trailing Separate` • `Leading Overpunched` • `Trailing Overpunched` This property is enabled only if you have set `Physical Type` to `Text` or `External Decimal`, and you have selected `Signed`. If `Physical Type` is `Text`, the only valid values of `Sign Orientation` are `Leading Separate` and `Trailing Separate`. If `Physical Type` is `External Decimal` and `Sign EBCDIC Customer Overpunched` is selected, the only valid values of `Sign Orientation` are `Leading Overpunched` and `Trailing Overpunched`.
Positive Sign	String	Specify the value that represents the positive symbol. Do not specify a numeric value. If no value is set, "+" is assumed. The positive sign is not written when creating an output message; it is used only to recognize the positive sign when parsing a message bit stream. This property is applicable only if `Physical Type` is `Text` and `Signed` is selected.

Property	Type	Meaning
Negative Sign	String	Specify the value that represents the negative symbol. Do not specify a numeric value. If no value is set, "-" is assumed. This property is applicable only if `Physical Type` is `Text` and `Signed` is selected.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the list: • `NULLPadFill`. This option is valid only for fixed-length objects and is the default value. • `NULLLogicalValue`. The `Encoding Null Value` property is first converted to an actual value, and rendered in the way specified for the field. • `NULLLiteralValue`. This option specifies that `Encoding Null Value` contains a value that is directly substituted as if it is a string. For dateTime elements, use this option if you want to use the `Encoding Null Value` property to test or compare the content of the field in the message. • `NULLLiteralFill`. This option specifies that the field is filled with the value specified by the `Encoding Null Value` property. `Encoding Null Value` must resolve to a single character. The option that you select determines the value that you must set for the property `Encoding Null Value`. For full information about using these options, see "TDS Null handling options" on page 6293.
Encoding Null Value	String	The use of this property depends on the `Encoding Null` property. The default value is zero. If you set the `Encoding Null` property for a dateTime object to `NULLLogicalValue`, the value that you set must be in an ISO8601 dateTime format. These formats are described in "DateTime as string data" on page 6311. For example, specify a value that conforms to the `yyyy-MM-dd'T'HH:mm:ss` format; for example, `1970-12-01`.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Compound element properties" on page 6091
Logical, CWF, XML, TDS, and document properties for a compound element.

"Compound element TDS properties" on page 6086
The properties, and their permissible values, vary according to the object type.

TDS properties for compound element float types:

Field identification, physical representation, numeric representation, and null values.

The TDS properties described here apply to:

- Objects: Compound elements

Field identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the `Data Element Separation` property of the complex type or types in which the object is a child is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, this property must contain a non-empty value. If the object is a complex element, and the `Data Element Separation` property of its parent is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, the property can contain an empty value. The value for this property must be unique for every element in the message set; that is, no two elements in the message set can contain the same value for this property.
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the `Data Element Separation` method is set to `Use Data Pattern` in the complex type. For more details, see "Regular expression syntax" on page 6304.
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the `Data Element Separation` property of the parent group or complex type is set to `All Elements Delimited` or `Variable Length Elements Delimited`. A default value is provided if the previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; if the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used. If none of the previous conditions are true, a default is not applied.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the `Messaging Standard` property of the message set is `User Defined Text`, `User Defined Mixed`, `CSV`, or `TLOG`, select one of the following values: • `Text`. The data is in character format. • `Float`. Equates to the data type FLOAT or DOUBLE in C, or the COMP-1 or COMP-2 numeric data type in COBOL. • `Integer`. Equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • `Packed Decimal`. The data is a packed decimal number that equates to the COMP-3 data type in COBOL. • `External Decimal`. The data is a decimal number that equates to the data type PIC 9 USAGE DISPLAY in COBOL. The default is dependent on the `Messaging Standard` property. For all other `Messaging Standard` values, the `Physical Type` property is set to `Text`.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no `Length Reference` is specified.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • `Not Applicable` • `Left Justify` • `Right Justify`

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the drop-down list. • Enter a character between quotation marks, for example "c" or 'c', where c is any alphanumeric character. • Enter a hexadecimal character code in the form `0xYY`, where `YY` is a hexadecimal value. • Enter a Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.
Length Reference	Enumerated type	This property is applicable only if `Physical Type` is `Text`, `Binary`, or `TLOG Specific`. If set, this property takes precedence over any value in the `Length Units` property. Specify the identifier of a sibling integer object, the value of which determines the length of the object in question. The sibling object must be defined before the current object within the message structure. For information about reordering elements, see “Reordering objects” on page 2900.

Numeric representation

Property	Type	Meaning
Virtual Decimal Point	Button and Integer	Specify a non-zero integer that represents the position of an implied decimal point within a number, or specify 0, the default, to use the formatting of Float or Decimal numbers as specified by the `Precision` property. If you specify a positive integer, the position of the decimal point is moved left from the right side of the number. For example, if you specify 3, the decimal value 1234 represents 1.234 If you specify a negative integer, the position of the decimal point is moved right from the right side of the number. For example, if you specify -3, the decimal value 1234 represents 1,234,000.
Precision	Button and Integer	This property is applicable only if `Physical Type` is `Text`. It is used if the value of the `Virtual Decimal Point` property is 0, which indicates that the decimal point is present in the data. It deals with truncation, and specifies how many digits are to follow the decimal point. Either specify a number of digits: • If you set `Precision` to 0, data is truncated so that the fractional part is lost. For example, the value 123.45 is truncated to 123. • If you set `Precision` to a number less than the number of fractional digits, data is truncated. For example, the value 123.4567 is truncated to 123.45 if you set `Precision` to 2. • If you set `Precision` to a number greater than the number of fractional digits, the value is padded with extra zeros. For example, the value 12.345 is padded to 12.34500 if you set `Precision` to 5. Or select one option from the list: • `All Significant Digits - decimal separator only required if fractional digits` (the default): all significant digits are written to the output bit stream, and no decimal separator is written if no fractional digits are present. • `Explicit Decimal Separator - decimal separator always required`: all significant digits are written to the output bit stream and the decimal separator is always included, even when no fractional digits are present. The decimal separator must be present in the input bit stream, even when no fractional digits are present. • `Exponential Notation - Example "1.23456e002"`: data is written out to the bit stream as a signed value having the format `[sign1]a.bbbe[sign2]ccc` where: – `[sign1]` is the value of `Negative Sign` if the value is negative – `a` is a single decimal digit – `bbb` is one or more decimal digits – `[sign2]` is the value of `Negative Sign` if the exponent is negative – `ccc` is exactly three decimal digits (the exponent) `[sign1]` and `[sign2]` are absent if the value and exponent are positive. For example, the value -123.456 is represented as `-1.23456e002` and the value 0.00012 is represented as `1.2e-004` in the output bit stream, assuming that the value of `Negative Sign` is "-", and the value of `Sign Orientation` is `Leading`. The value -0.00012 is represented as `1.2*e*004` if `Negative Sign` is "*" and `Sign Orientation` is `Trailing`.

Property	Type	Meaning
Sign Orientation	Enumerated type	The values that you can choose for this property are: • Leading Separate • Trailing Separate • Leading Overpunched • Trailing Overpunched This property is enabled only if you have set Physical Type to Text or External Decimal, and you have selected Signed. If Physical Type is Text, the only valid values of Sign Orientation are Leading Separate and Trailing Separate. If Physical Type is External Decimal and Sign EBCDIC Customer Overpunched is selected, the only valid values of Sign Orientation are Leading Overpunched and Trailing Overpunched.
Positive Sign	String	Specify the value that represents the positive symbol. Do not specify a numeric value. If no value is set, "+" is assumed. The positive sign is not written when creating an output message; it is used only to recognize the positive sign when parsing a message bit stream. This property is applicable only if Physical Type is Text and Signed is selected.
Negative Sign	String	Specify the value that represents the negative symbol. Do not specify a numeric value. If no value is set, "-" is assumed. This property is applicable only if Physical Type is Text and Signed is selected.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the list: • NULLPadFill. This option is valid only for fixed-length objects and is the default value. • NULLLogicalValue. The Encoding Null Value property is first converted to an actual value, and rendered in the way specified for the field. • NULLLiteralValue. This option specifies that Encoding Null Value contains a value that is directly substituted as if it is a string. For dateTime elements, use this option if you want to use the Encoding Null Value property to test or compare the content of the field in the message. • NULLLiteralFill. This option specifies that the field is filled with the value specified by the Encoding Null Value property. Encoding Null Value must resolve to a single character. The option that you select determines the value that you must set for the property Encoding Null Value. For full information about using these options, see “TDS Null handling options” on page 6293.
Encoding Null Value	String	The use of this property depends on the Encoding Null property. The default value is zero. If you set the Encoding Null property for a dateTime object to NULLLogicalValue, the value that you set must be in an ISO8601 dateTime format. These formats are described in “DateTime as string data” on page 6311. For example, specify a value that conforms to the yyyy-MM-dd'T'HH:mm:ss format; for example, 1970-12-01.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Compound element properties" on page 6091
Logical, CWF, XML, TDS, and document properties for a compound element.

"Compound element TDS properties" on page 6086
The properties, and their permissible values, vary according to the object type.

TDS properties for compound element integer types:

Field identification, physical representation, numeric representation, and null values.

The TDS properties described here apply to:

- Objects: Compound elements

Field identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the `Data Element Separation` property of the complex type or types in which the object is a child is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, this property must contain a non-empty value. If the object is a complex element, and the `Data Element Separation` property of its parent is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, the property can contain an empty value. The value for this property must be unique for every element in the message set; that is, no two elements in the message set can contain the same value for this property.
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the `Data Element Separation` method is set to `Use Data Pattern` in the complex type. For more details, see "Regular expression syntax" on page 6304.
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the `Data Element Separation` property of the parent group or complex type is set to `All Elements Delimited` or `Variable Length Elements Delimited`. A default value is provided if the previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; if the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used. If none of the previous conditions are true, a default is not applied.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the `Messaging Standard` property of the message set is `User Defined Text`, `User Defined Mixed`, `CSV`, or `TLOG`, select one of the following values: • `Text`. The data is in character format. • `Integer`. Equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • `Packed Decimal`. The data is a packed decimal number that equates to the COMP-3 data type in COBOL. • `External Decimal`. The data is a decimal number that equates to the data type PIC 9 USAGE DISPLAY in COBOL. The default is dependent on the `Messaging Standard` property. For all other `Messaging Standard` values, the `Physical Type` property is set to `Text`.

Property	Type	Meaning
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no `Length Reference` is specified. If this property is not set and the message set property `Derive default length from logical type` is selected, and the `Physical type` is 'Character', the default value is derived from any totalDigits value constraint (schema facet) or, if none, any `minInclusive`, `maxInclusive`, `minExclusive`, or `maxExclusive` value constraints (schema facets), on the simple type.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • `Not Applicable` • `Left Justify` • `Right Justify`

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the drop-down list. • Enter a character between quotation marks, for example "c" or 'c', where c is any alphanumeric character. • Enter a hexadecimal character code in the form `0xYY`, where `YY` is a hexadecimal value. • Enter a Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.
Length Reference	Enumerated type	This property is applicable only if `Physical Type` is `Text`, `Binary`, or `TLOG Specific`. If set, this property takes precedence over any value in the `Length Units` property. Specify the identifier of a sibling integer object, the value of which determines the length of the object in question. The sibling object must be defined before the current object within the message structure. For information about reordering elements, see “Reordering objects” on page 2900.

Numeric representation

Property	Type	Meaning
Sign Orientation	Enumerated type	The values that you can choose for this property are: • Leading Separate • Trailing Separate • Leading Overpunched • Trailing Overpunched This property is enabled only if you have set Physical Type to Text or External Decimal, and you have selected Signed. If Physical Type is Text, the only valid values of Sign Orientation are Leading Separate and Trailing Separate. If Physical Type is External Decimal and Sign EBCDIC Customer Overpunched is selected, the only valid values of Sign Orientation are Leading Overpunched and Trailing Overpunched.
Positive Sign	String	Specify the value that represents the positive symbol. Do not specify a numeric value. If no value is set, "+" is assumed. The positive sign is not written when creating an output message; it is used only to recognize the positive sign when parsing a message bit stream. This property is applicable only if Physical Type is Text and Signed is selected.
Negative Sign	String	Specify the value that represents the negative symbol. Do not specify a numeric value. If no value is set, "-" is assumed. This property is applicable only if Physical Type is Text and Signed is selected.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the list: • NULLPadFill. This option is valid only for fixed-length objects and is the default value. • NULLLogicalValue. The Encoding Null Value property is first converted to an actual value, and rendered in the way specified for the field. • NULLLiteralValue. This option specifies that Encoding Null Value contains a value that is directly substituted as if it is a string. For dateTime elements, use this option if you want to use the Encoding Null Value property to test or compare the content of the field in the message. • NULLLiteralFill. This option specifies that the field is filled with the value specified by the Encoding Null Value property. Encoding Null Value must resolve to a single character. The option that you select determines the value that you must set for the property Encoding Null Value. For full information about using these options, see “TDS Null handling options” on page 6293.

Property	Type	Meaning
Encoding Null Value	String	The use of this property depends on the `Encoding Null` property. The default value is zero. If you set the `Encoding Null` property for a dateTime object to `NULLLogicalValue`, the value that you set must be in an ISO8601 dateTime format. These formats are described in "DateTime as string data" on page 6311. For example, specify a value that conforms to the `yyyy-MM-dd'T'HH:mm:ss` format; for example, `1970-12-01`.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Compound element properties" on page 6091
Logical, CWF, XML, TDS, and document properties for a compound element.

"Compound element TDS properties" on page 6086
The properties, and their permissible values, vary according to the object type.

TDS properties for compound element string types:

Field identification, physical representation, and null values.

The TDS properties described here apply to:

- Objects: Compound elements

Field identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the `Data Element Separation` property of the complex type or types in which the object is a child is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, this property must contain a non-empty value. If the object is a complex element, and the `Data Element Separation` property of its parent is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, the property can contain an empty value. The value for this property must be unique for every element in the message set; that is, no two elements in the message set can contain the same value for this property.
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the `Data Element Separation` method is set to `Use Data Pattern` in the complex type. For more details, see "Regular expression syntax" on page 6304.
Interpret Element Value	Enumerated type	Specify whether values stored within this object are interpreted as having significance for the parser and, if so, the type of interpretation that occurs. This interpretation is standard-specific and is therefore hard coded. The possible values for this property are: • `None` (the default value) • `EDIFACT Service String` • `X12 Service String` • `Message Key` • `EDIFACT Syntax Level ID` • `HL7 Service String` • `HL7 Field Separator`
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the `Data Element Separation` property of the parent group or complex type is set to `All Elements Delimited` or `Variable Length Elements Delimited`. A default value is provided if the previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; if the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used. If none of the previous conditions are true, a default is not applied.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the `Messaging Standard` property of the message set is `User Defined Text`, `User Defined Mixed`, `CSV`, or `TLOG`, select one of the following values: • `Text`. The data is in character format. • `Length Encoded String 1`. The first byte of the data contains the length (in length units) of the data string that follows the length byte. The maximum length of a `Length Encoded String 1` element is 255 length units. • `Length Encoded String 2`. The first two bytes of the data contain the length (in length units) of the data string that follows the two length bytes. The maximum length of a `Length Encoded String 2` element is 65535 length units. • `Null Terminated String`. The data string ends with the hexadecimal NULL character, `X'00'`. • `TLOG Specific` - this option can be selected only if the `Message Standard` property of the message set is `TLOG`. This option indicates that the format of the data is specific to the TLOG messaging standard. The default is dependent on the `Messaging Standard` property. For all other `Messaging Standard` values, the `Physical Type` property is set to `Text`.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no `Length Reference` is specified. If this property is not set and the message set property `Derive default length from logical type` is selected, and the `Physical type` is 'Character', the default value is derived from any length or maxLength value constraint (schema facet) on the simple type.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • `Not Applicable` • `Left Justify` • `Right Justify`

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the drop-down list. • Enter a character between quotation marks, for example "c" or 'c', where c is any alphanumeric character. • Enter a hexadecimal character code in the form `0xYY`, where `YY` is a hexadecimal value. • Enter a Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.
Length Reference	Enumerated type	This property is applicable only if `Physical Type` is `Text`, `Binary`, or `TLOG Specific`. If set, this property takes precedence over any value in the `Length Units` property. Specify the identifier of a sibling integer object, the value of which determines the length of the object in question. The sibling object must be defined before the current object within the message structure. For information about reordering elements, see “Reordering objects” on page 2900.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the list: • `NULLPadFill`. This option is valid only for fixed-length objects and is the default value. • `NULLLogicalValue`. The `Encoding Null Value` property is first converted to an actual value, and rendered in the way specified for the field. • `NULLLiteralValue`. This option specifies that `Encoding Null Value` contains a value that is directly substituted as if it is a string. For dateTime elements, use this option if you want to use the `Encoding Null Value` property to test or compare the content of the field in the message. • `NULLLiteralFill`. This option specifies that the field is filled with the value specified by the `Encoding Null Value` property. `Encoding Null Value` must resolve to a single character. The option that you select determines the value that you must set for the property `Encoding Null Value`. For full information about using these options, see "TDS Null handling options" on page 6293.
Encoding Null Value	String	The use of this property depends on the `Encoding Null` property. The default value is zero. If you set the `Encoding Null` property for a dateTime object to `NULLLogicalValue`, the value that you set must be in an ISO8601 dateTime format. These formats are described in "DateTime as string data" on page 6311. For example, specify a value that conforms to the `yyyy-MM-dd'T'HH:mm:ss` format; for example, `1970-12-01`.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Compound element properties" on page 6091
Logical, CWF, XML, TDS, and document properties for a compound element.

"Compound element TDS properties" on page 6086
The properties, and their permissible values, vary according to the object type.

Documentation properties for all message set objects:

Use the documentation property of an object to add information that describes the function of the object.

The documentation property is available on all objects except Key, Keyref, and Unique objects.

The property is a string field, and you can use any standard alphanumeric characters.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

Embedded simple type properties:

Logical, CWF, XML, TDS, and documentation properties for embedded simple types.

- "Embedded simple type logical properties" on page 6074
- "Embedded simple type CWF properties" on page 6079
- "Embedded simple type XML properties" on page 6084
- "Embedded simple type TDS properties" on page 6088
- "Documentation properties for all message set objects" on page 5413

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Embedded simple type logical properties" on page 6074
The logical properties of an embedded simple type include properties that specify the number of occurrences of the embedded simple type.

"Embedded simple type CWF properties" on page 6079
The properties, and their permissible values, vary according to the object type.

"Embedded simple type XML properties" on page 6084
The properties, and their permissible values, vary according to the object type.

"Embedded simple type TDS properties" on page 6088
The properties, and their permissible values, vary according to the object type.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

Embedded simple type logical properties:

The logical properties of an embedded simple type include properties that specify the number of occurrences of the embedded simple type.

Occurrences

Property	Type	Meaning
Min Occurs	Integer	Specify the minimum number of times that the object can repeat. The default value is 1. If the value is set to 0, the object is optional. With the exception of `Max Occurs` being set to -1, if a value is set for `Min Occurs`, it must be less than or equal to the value in `Max Occurs`.
Max Occurs	Integer	Specify the maximum number of times that the object can repeat. The default value is 1. If this property is not set, the object cannot occur more than once. If this property is set to 0, it is interpreted as if the object does not exist in the message. It can also be set to -1, to indicate that the limit is unbounded and there is no maximum to the number of occurrences.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Logical properties for message model objects" on page 5416
Logical property information is available for certain objects.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Embedded simple type CWF properties:

The properties, and their permissible values, vary according to the object type.

The properties that are displayed on the object page, and the values that those properties can take, can vary according to the type of the object. For example, the properties for type *string* are different from those of type *Boolean*. Select the link for the object type from the following table.

Type of object	Properties
Binary types	• ComIbmMrm _AnonBinary
Boolean types	• ComIbmMrm _AnonBoolean
DateTime types	• ComIbmMrm _AnonDate • ComIbmMrm _AnonDateTime • ComIbmMrm _AnonGDay • ComIbmMrm _AnonGMonth • ComIbmMrm _AnonGMonthDay • ComIbmMrm _AnonGYear • ComIbmMrm _AnonGYearMonth • ComIbmMrm _AnonTime
Decimal types	• ComIbmMrm _AnonDecimal
Float types	• ComIbmMrm _AnonFloat
Integer types	• ComIbmMrm _AnonInt

Type of object	Properties
String types	• ComIbmMrm _AnonString

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM Custom Wire Format" on page 1214
Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not separated by delimiters.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Embedded simple type properties" on page 6180
Logical, CWF, XML, TDS, and documentation properties for embedded simple types.

"Embedded simple type logical properties" on page 6074
The logical properties of an embedded simple type include properties that specify the number of occurrences of the embedded simple type.

"Embedded simple type XML properties" on page 6084
The properties, and their permissible values, vary according to the object type.

"Embedded simple type TDS properties" on page 6088
The properties, and their permissible values, vary according to the object type.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

CWF properties for embedded simple type binary types:

Physical representation, byte alignment, and occurrences.

The Custom Wire Format properties described here apply to:

- Objects: Embedded simple types

Physical representation

Property	Type	Meaning
Length	Button and Integer	If you have selected the length to be defined by *Length*, enter the number of length units for the element. The minimum value that you can specify is 1. The maximum value that you can specify is 2147483647. The default value is empty (not set).
Length Reference	Button and Enumerated type	If you have selected the length to be defined by *Length Reference*, select the name of the integer object that specifies the length of this object. Make your selection from the displayed list of integer objects that are defined as siblings of the current object, and occur before it in the structure of the message. For information about reordering elements, see “Reordering objects” on page 2900.
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • Bytes. The length is given in bytes. • Characters. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • Character Units. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to Characters. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the Length property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • End of Bitstream. All data up to the end of the bitstream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value for the *Length Count* or *Length Reference* property. The default is Bytes.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to all instances.

Occurrences

Property	Type	Meaning
Repeat Reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the displayed list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, there are no integer objects before this one in the message structure. If a *`Repeat Reference`* is specified, it overrides any setting for the *`Max Occurs`* logical property when parsing and writing the message, but not for validation of the message.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Embedded simple type properties" on page 6180
Logical, CWF, XML, TDS, and documentation properties for embedded simple types.

"Embedded simple type CWF properties" on page 6079
The properties, and their permissible values, vary according to the object type.

CWF properties for embedded simple type Boolean types:

Byte alignment and occurrences.

The Custom Wire Format properties described here apply to:

- Objects: Embedded simple types

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to all instances.

Occurrences

Property	Type	Meaning
Repeat Reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the displayed list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* logical property when parsing and writing the message, but not for validation of the message.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Embedded simple type properties" on page 6180
Logical, CWF, XML, TDS, and documentation properties for embedded simple types.

"Embedded simple type CWF properties" on page 6079
The properties, and their permissible values, vary according to the object type.

CWF properties for embedded simple type dateTime types:

Physical representation, null values, byte alignment, and occurrences.

The Custom Wire Format properties described here apply to:

- Objects: Embedded simple types

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select one from the displayed list: • `Fixed Length String`. The element's length is determined by other length properties as follows. • `Length Encoded String 1`. The first byte of the element contains the length of the string following the length byte in length units. The maximum length of a `Length Encoded String 1` element is 255 length units. • `Length Encoded String 2`. The element's first 2 bytes contain the length of the string following the two length bytes in length units. The maximum length of a `Length Encoded String 2` element is 65535 length units. The two length bytes are in the format of the WebSphere MQ queue manager Encoding. • `Null Terminated String`. The string ends with the hexadecimal NULL character, `X'00'`. • `Packed Decimal`. The dateTime is coded as a Packed Decimal number. It is valid only if the *`DateTime Format`* property represents numeric-only data. • `Binary`. The dateTime is encoded as a binary sequence of bytes. If you select this option, the range of symbols that you can specify for the Format String property is less than the range of symbols you can specify if you select a string option (see "DateTime formats" on page 6310 for details). • `Time Seconds`. This value supports C time_t and Java Date and Time objects. It is valid only if the *`DateTime Format`* property represents numeric-only data. • `Time Milliseconds`. This value supports C time_t and Java Date and Time objects. It is valid only if the *`DateTime Format`* property represents numeric-only data. The default value is `fixed length string`.
DateTime Format	String	Specify a template for date and time. The default dateTime format is dependent on the logical type of the object. For information about the defaults for the dateTime format according to the logical type, see "DateTime defaults by logical type" on page 6320. See "DateTime formats" on page 6310 for details of date and time formats.
Length	Button and Integer	If you have selected a *`Physical Type`* of `Fixed Length String`, `Packed Decimal`, or `Binary`, and have selected the length to be defined by *`Length`*, enter the number of length units for the element. The minimum value that you can specify is 1 for all three physical types. The maximum value that you can specify is `256` for `Fixed Length String`, `10` for `Packed Decimal`, and `2147483647` for `Binary`. The default value is 0 (zero).
Length Reference	Button and Enumerated type	If you have selected the length to be defined by *`Length Reference`*, select the name of the integer object that specifies the length of this object. Make your selection from the displayed list of integer objects that are defined as siblings of the current object, and occur before it in the structure of the message. For information about reordering elements, see "Reordering objects" on page 2900.

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. The length is given in bytes. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value for the *`Length Count`* or *`Length Reference`* property. The default is `Bytes`.
Signed	Check box	Specify whether the value is signed. This property is applicable only if the *`Physical type`* property is `Packed Decimal`. By default, this check box is cleared, which indicates that the value is not signed.
Justification	Enumerated type	If you have set the *`Physical Type`* property to `Fixed Length String`, select `Left Justify` (the default value) or `Right Justify` from the list. If you have selected another value for *`Physical Type`*, this property is inactive.

Property	Type	Meaning
Padding Character	String	If you have set the *Physical Type* property to `Fixed Length String`, and the *Justification* property is either `Left Justify` or `Right Justify`, this property is applicable. When writing an output message, use the padding character to fill out the remaining character positions when the string length is less than the length implied by the *Length* or *Length Reference* property. Whether the string is padded from the left or the right is governed by the *Justification* property. When parsing an input message, the padding character is trimmed from the end of the string. Whether the string is trimmed from the left or the right is governed by the *Justification* property. Specify this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the displayed list. • Enter a character between quotation marks; for example, `"c"` or `'c'`, where c is any alphanumeric character. • Enter a Unicode value in the form `U+xxxx` where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form `0xYY`, where `YY` is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, it is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the displayed list. The option that you select determines the value that you must set for the property *Encoding Null Value*: • NULLPadFill. This option is valid only if *Physical Type* is Fixed Length String. The field is filled with the value specified by the *Padding Character*. The default value. • NULLLogicalValue. The *Encoding Null Value* property is first converted to an actual value, and rendered in the way specified for the field. • NULLLiteralValue. This specifies that *Encoding Null Value* contains a value that is directly substituted as if it is a string. Use this option when the value you have set for *Encoding Null Value* to specify a null date is not a dateTime value, or does not conform to the standard dateTime format yyyy-MM-dd 'T'HH:mm:ss. • NULLLiteralFill. The field is filled with the value specified by the *Encoding Null Value*. *Encoding Null Value* must resolve to a single character.
Encoding Null Value	String	If you set the *Encoding Null* property to NULLPadFill, this property is disabled. If you set the *Encoding Null* property to NULLLogicalValue, you must set this property to an ISO8601 dateTime format. These formats are described in "DateTime as string data" on page 6311. For example, specify a value conforming to yyyy-MM-dd'T'HH:mm:ss such as 1970-12-01. If you set the *Encoding Null* property to NULLLiteralValue, you can enter any value that is the same length as the field. If you set the *Encoding Null* property to NULLLiteralFill, the value must resolve to a single character. Set the character in one of the following ways: • Select SPACE, NUL, 0x00 or 0xFF from the displayed list • Enter a character between quotation marks, for example 'c' or "c", where c is any alphanumeric character. • Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value. • Enter a decimal character code in the form YY where YY is a decimal value. • Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal format.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • 1 Bytes. The default value. • 2 Bytes • 4 Bytes • 8 Bytes • 16 Bytes

Property	Type	Meaning
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to all instances.

Occurrences

Property	Type	Meaning
Repeat Reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the displayed list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, there are no integer objects before this one in the message structure. If a *`Repeat Reference`* is specified, it overrides any setting for the *`Max Occurs`* logical property when parsing and writing the message, but not for validation of the message.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Embedded simple type properties" on page 6180
Logical, CWF, XML, TDS, and documentation properties for embedded simple types.

"Embedded simple type CWF properties" on page 6079
The properties, and their permissible values, vary according to the object type.

CWF properties for embedded simple type decimal types:

Physical representation, null values, byte alignment, and occurrences.

The Custom Wire Format properties described here apply to:

- Objects: Embedded simple types

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select one from the displayed list: • `Integer`. This equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • `Packed Decimal`. Equates to the COMP-3 data type in COBOL. • `External Decimal`. Equates to the data type PIC 9 USAGE DISPLAY in COBOL. The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager: • Elements that have *`Physical Type`* set to `Integer` and `Packed Decimal` are represented in the appropriate WebSphere MQ Encoding value. • Elements that have *`Physical Type`* set to `External Decimal` are represented in the WebSphere MQ CodedCharSetId value.
Length	Integer	Enter the number of bytes to specify the element length: • If you set the *`Physical Type`* to `Integer`, select `1`, `2`, or `4` (the default) from the displayed list. • If you set the *`Physical Type`* to `Packed Decimal`, enter a value between 1 and 10. • If you set the *`Physical Type`* to `Extended Decimal`, enter a value between 1 and 256. (Numbers greater than the maximum COBOL PICTURE clause of 18 are assumed to be 18.)

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value for the *`Length Count`*. The default is `Bytes`.
Signed	Check box	Select (the default) or clear this property. This property is used with *`Sign Orientation`*.
Sign EBCDIC Custom Overpunched	Check box	If the *`Physical Type`* is set to `External Decimal` and the *`Signed EBCDIC Custom`* property is set, this indicates that the Sign EBCDIC Custom Overpunched representation is to be used within an ASCII environment. If this check box is not selected (the default), the Sign ASCII representation is used. The setting of the *`Sign EBCDIC Custom Overpunched`* check box is appropriate only if the *`Sign Orientation`* property is set to `Leading` or `Trailing` (indicating that the element/attribute has an embedded sign representation). The check box is not available if the element/attribute is unsigned (for example, if the *`Signed`* check box is not set).

Property	Type	Meaning
Sign Orientation	Enumerated type	If you have set *Physical Type* to External Decimal and you have selected *Signed*, choose from the following options that represent the COBOL options for displaying numeric data: • Leading Overpunched. This option sets a bit in the first byte if the number is negative. No setting is made if the number is positive. For example, the ASCII hexadecimal representation of the number 22 is x'3232'. Using this option, the number +22 would be x'3232' and the number -22 would be x'7232'. The default value. • Leading Separate. This option sets the first byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte. • Trailing Overpunched. This option sets a bit in the last byte if the number is negative. No setting is made if the number is positive. Using this option, the number +22 would be x'3232' and the number -22 would be x'3272'. • Trailing Separate. This option sets the last byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte. If you have set *Physical Type* to any other value, the value Not Applicable is set for you.
Virtual Decimal Point	Integer	Specify the number of places to the left (for a positive value) or right (for a negative value) that a decimal point is to be moved from its assumed position. For example, a decimal element containing 1234 with a Virtual Decimal value of 3 is 1.234, equivalent to 'V' or 'P' in a COBOL picture clause. There is no C equivalent
Justification	Enumerated type	If you have set the *Physical Type* property to External Decimal, select Left Justify or Right Justify (the default value) from the list. If you have selected another value for *Physical Type*, this property is inactive.

Property	Type	Meaning
Padding Character	String	The padding character is used to fill out the remaining character positions when the string length is less than the specified string size. If you have set the *Physical Type* property to Extended Decimal, and the *Justification* property is either Left Justify or Right Justify, specify this character in one of the following ways: • Select NUL, '0', or SPACE from the displayed list. • Enter a character between quotation marks; for example, "c" or 'c', where c is any alphanumeric character. • Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form 0xYY, where YY is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the displayed list: • NULLPadFill. This option is valid only if *Physical Type* is External Decimal. The field is filled with the value specified by the *Padding Character*. *Encoding Null Value* must be set to an empty string. • NULLLogicalValue. The *Encoding Null Value* is transformed to match the required format for the field. The default value. • NULLLiteralValue. The *Encoding Null Value* is directly substituted as if it is a string. You can specify a nonnumeric value for *Encoding Null Value*. • NULLLiteralFill. The field is filled with the value specified by the *Encoding Null Value*. *Encoding Null Value* must resolve to a single character.

Property	Type	Meaning
Encoding Null Value	String	The use of this property depends on the *Encoding Null* property, except for NULLLiteralFill. The default value is zero. If you set the *Encoding Null* property to NULLLiteralFill, the value must resolve to a single character. Set the character in one of the following ways: • Select SPACE, NUL, 0x00 or 0xFF from the displayed list • Enter a character between quotation marks, for example 'c' or "c", where c is any alphanumeric character. • Enter a hexadecimal character in the form 0xYY where YY is a hexadecimal value. • Enter a decimal character code in the form YY where YY is a decimal value. • Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal format.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • 1 Bytes. The default value. • 2 Bytes • 4 Bytes • 8 Bytes • 16 Bytes
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to all instances.

Occurrences

Property	Type	Meaning
Repeat Reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the displayed list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* logical property when parsing and writing the message, but not for validation of the message.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Embedded simple type properties" on page 6180
Logical, CWF, XML, TDS, and documentation properties for embedded simple types.

"Embedded simple type CWF properties" on page 6079
The properties, and their permissible values, vary according to the object type.

CWF properties for embedded simple type float types:

Physical representation, null values, byte alignment, and occurrences.

The Custom Wire Format properties described here apply to:

- Objects: Embedded simple types

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select one from the displayed list: • `Integer`. This equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • `Float`. This equates to the data type FLOAT or DOUBLE in C or the COMP-1 or COMP-2 data type in COBOL and is the default value. • `Packed Decimal`. Equates to the COMP-3 data type in COBOL. • `External Decimal`. Equates to the data type PIC 9 USAGE DISPLAY in COBOL. The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager: • Elements that have *`Physical Type`* set to `Integer`, `Packed Decimal`, and `Float` are represented in the appropriate WebSphere MQ Encoding value. • Elements that have *`Physical Type`* set to `External Decimal` are represented in the WebSphere MQ CodedCharSetId value.
Length	Integer	Enter the number of bytes to specify the element length: • If you set the *`Physical Type`* to `Float`, select a value from the displayed list. The default value is 8. • If you set the *`Physical Type`* to `Integer`, select `1`, `2`, or `4` (the default) from the displayed list. • If you set the *`Physical Type`* to `Packed Decimal`, enter a value between 1 and 10. • If you set the *`Physical Type`* to `Extended Decimal`, enter a value between 1 and 256. (Numbers greater than the maximum COBOL PICTURE clause of 18 are assumed to be 18.)

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value for the *`Length Count`*. The default is `Bytes`.
Signed	Check box	Select or clear (unsigned, the default) this property. If you have set *`Physical Type`* to `Float`, this is selected. This property is used with *`Sign Orientation`*.
Sign EBCDIC Custom Overpunched	Check box	If the *`Physical Type`* is set to `External Decimal` and the *`Signed EBCDIC Custom`* property is set, this indicates that the Sign EBCDIC Custom Overpunched representation is to be used within an ASCII environment. If this check box is not selected (the default), the Sign ASCII representation is used. The setting of the *`Sign EBCDIC Custom Overpunched`* check box is appropriate only if the *`Sign Orientation`* property is set to `Leading` or `Trailing` (indicating that the element/attribute has an embedded sign representation). The check box is not available if the element/attribute is unsigned (for example, if the *`Signed`* check box is not set).

Property	Type	Meaning
Sign Orientation	Enumerated type	If you have set *Physical Type* to External Decimal and you have selected *Signed*, choose from the following options that represent the COBOL options for displaying numeric data: • Leading Overpunched. This option sets a bit in the first byte if the number is negative. No setting is made if the number is positive. For example, the ASCII hexadecimal representation of the number 22 is x'3232'. Using this option, the number +22 would be x'3232' and the number -22 would be x'7232'. The default value. • Leading Separate. This option sets the first byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte. • Trailing Overpunched. This option sets a bit in the last byte if the number is negative. No setting is made if the number is positive. Using this option, the number +22 would be x'3232' and the number -22 would be x'3272'. • Trailing Separate. This option sets the last byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte. If you have set *Physical Type* to any other value, the value Not Applicable is set for you.
Virtual Decimal Point	Integer	Specify the number of places to the left (for a positive value) or right (for a negative value) that a decimal point is to be moved from its assumed position. For example, a float element containing 1234 with a Virtual Decimal value of 3 is 1.234. This property is not applicable if you have set *Physical Type* to Float.
Justification	Enumerated type	If you have set the *Physical Type* property to External Decimal, select Left Justify or Right Justify (the default value) from the list. If you have selected another value for *Physical Type*, this property is inactive.

Property	Type	Meaning
Padding Character	String	The padding character is used to fill out the remaining character positions when the string length is less than the specified string size. If you have set the *Physical Type* property to Extended Decimal, and the *Justification* property is either Left Justify or Right Justify, specify this character in one of the following ways: • Select NUL, '0', or SPACE from the displayed list. • Enter a character between quotation marks; for example, "c" or 'c', where c is any alphanumeric character. • Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form 0xYY, where YY is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the displayed list: • NULLPadFill. This option is valid only if *Physical Type* is External Decimal. The field is filled with the value specified by the *Padding Character*. *Encoding Null Value* must be set to an empty string. • NULLLogicalValue. The *Encoding Null Value* is transformed to match the required format for the field. The default value. • NULLLiteralValue. The *Encoding Null Value* is directly substituted as if it is a string. You can specify a nonnumeric value for *Encoding Null Value*. • NULLLiteralFill. The field is filled with the value specified by the *Encoding Null Value*. *Encoding Null Value* must resolve to a single character.

Property	Type	Meaning
Encoding Null Value	String	The use of this property depends on the *Encoding Null* property, except for NULLLiteralFill. The default value is zero. If you set the *Encoding Null* property to NULLLiteralFill, the value must resolve to a single character. Set the character in one of the following ways: • Select SPACE, NUL, 0x00 or 0xFF from the displayed list • Enter a character between quotation marks, for example 'c' or "c", where c is any alphanumeric character. • Enter a hexadecimal character in the form 0xYY where YY is a hexadecimal value. • Enter a decimal character code in the form YY where YY is a decimal value. • Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal format.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • 1 Bytes. The default value. • 2 Bytes • 4 Bytes • 8 Bytes • 16 Bytes
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to all instances.

Occurrences

Property	Type	Meaning
Repeat Reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the displayed list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* logical property when parsing and writing the message, but not for validation of the message.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Embedded simple type properties" on page 6180
Logical, CWF, XML, TDS, and documentation properties for embedded simple types.

"Embedded simple type CWF properties" on page 6079
The properties, and their permissible values, vary according to the object type.

CWF properties for embedded simple type integer types:

Physical representation, null values, byte alignment, and occurrences.

The Custom Wire Format properties described here apply to:

- Objects: Embedded simple types

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select one from the displayed list: • `Integer`. This equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • `Packed Decimal`. Equates to the COMP-3 data type in COBOL. • `External Decimal`. Equates to the data type PIC 9 USAGE DISPLAY in COBOL. The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager: • Elements that have *`Physical Type`* set to `Integer` and `Packed Decimal` are represented in the appropriate WebSphere MQ Encoding value. • Elements that have *`Physical Type`* set to `External Decimal` are represented in the WebSphere MQ CodedCharSetId value.
Length	Integer	Enter the number of bytes to specify the element length: • If you set the *`Physical Type`* to `Integer`, select `1`, `2`, or `4` (the default) from the displayed list. • If you have set *`Physical Type`* to `Packed Decimal`, enter a value between 1 and 6. • If you have set *`Physical Type`* to `Extended Decimal`, enter a value between 1 and 11.
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value for the *`Length Count`*. The default is `Bytes`.
Signed	Check box	Select (the default) or clear this property. This property is used with *`Sign Orientation`*.

Property	Type	Meaning
Sign EBCDIC Custom Overpunched	Check box	If the *Physical Type* is set to External Decimal and the *Signed EBCDIC Custom* property is set, this indicates that the Sign EBCDIC Custom Overpunched representation is to be used within an ASCII environment. If this check box is not selected (the default), the Sign ASCII representation is used. The setting of the *Sign EBCDIC Custom Overpunched* check box is appropriate only if the *Sign Orientation* property is set to Leading or Trailing (indicating that the element/attribute has an embedded sign representation). The check box is not available if the element/attribute is unsigned (for example, if the *Signed* check box is not set).
Sign Orientation	Enumerated type	If you have set *Physical Type* to External Decimal and you have selected *Signed*, choose from the following options that represent the COBOL options for displaying numeric data: • Leading Overpunched. This option sets a bit in the first byte if the number is negative. No setting is made if the number is positive. For example, the ASCII hexadecimal representation of the number 22 is x'3232'. Using this option, the number +22 would be x'3232' and the number -22 would be x'7232'. The default value. • Leading Separate. This option sets the first byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte. • Trailing Overpunched. This option sets a bit in the last byte if the number is negative. No setting is made if the number is positive. Using this option, the number +22 would be x'3232' and the number -22 would be x'3272'. • Trailing Separate. This option sets the last byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte. If you have set *Physical Type* to any other value, the value Not Applicable is set for you.
Justification	Enumerated type	If you have set the *Physical Type* property to External Decimal, select Left Justify or Right Justify (the default value) from the list. If you have selected another value for *Physical Type*, this property is inactive.

Property	Type	Meaning
Padding Character	String	The padding character is used to fill out the remaining character positions when the string length is less than the specified string size. If you have set the *Physical Type* property to Extended Decimal, and the *Justification* property is either Left Justify or Right Justify, specify this character in one of the following ways: • Select NUL, '0', or SPACE from the displayed list. • Enter a character between quotation marks; for example, "c" or 'c', where c is any alphanumeric character. • Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form 0xYY, where YY is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the displayed list: • NULLPadFill. This option is valid only if *Physical Type* is External Decimal. The field is filled with the value specified by the *Padding Character*. *Encoding Null Value* must be set to an empty string. • NULLLogicalValue. The *Encoding Null Value* is transformed to match the required format for the field. The default value. • NULLLiteralValue. The *Encoding Null Value* is directly substituted as if it is a string. You can specify a nonnumeric value for *Encoding Null Value*. • NULLLiteralFill. The field is filled with the value specified by the *Encoding Null Value*. *Encoding Null Value* must resolve to a single character.

Property	Type	Meaning
Encoding Null Value	String	The use of this property depends on the *Encoding Null* property, except for NULLLiteralFill. The default value is zero. If you set the *Encoding Null* property to NULLLiteralFill, the value must resolve to a single character. Set the character in one of the following ways: • Select SPACE, NUL, 0x00 or 0xFF from the displayed list • Enter a character between quotation marks, for example 'c' or "c", where c is any alphanumeric character. • Enter a hexadecimal character in the form 0xYY where YY is a hexadecimal value. • Enter a decimal character code in the form YY where YY is a decimal value. • Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal format.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • 1 Bytes. The default value. • 2 Bytes • 4 Bytes • 8 Bytes • 16 Bytes
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to all instances.

Occurrences

Property	Type	Meaning
Repeat Reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the displayed list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* logical property when parsing and writing the message, but not for validation of the message.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Embedded simple type properties" on page 6180
Logical, CWF, XML, TDS, and documentation properties for embedded simple types.

"Embedded simple type CWF properties" on page 6079
The properties, and their permissible values, vary according to the object type.

CWF properties for embedded simple type string types:

Physical representation, null values, byte alignment, and occurrences.

The Custom Wire Format properties described here apply to:

- Objects: Embedded simple types

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select one from the displayed list: • `Fixed Length String`. The element's length is determined by other length properties as follows. • `Length Encoded String 1`. The first byte of the element contains the length of the string following the length byte in length units. The maximum length of a `Length Encoded String 1` element is 255 length units. • `Length Encoded String 2`. The element's first 2 bytes contain the length of the string following the two length bytes in length units. The maximum length of a `Length Encoded String 2` element is 65535 length units. The two length bytes are in the format of the WebSphere MQ queue manager Encoding. • `Null Terminated String`. The string ends with the hexadecimal NULL character, `X'00'`. The default is `Fixed Length String`.
Length	Button and Integer	If you have selected a *`Physical Type`* of `Fixed Length String` or `Binary`, and have selected the length to be defined by *`Length`*, enter the number of length units for the element. The minimum value that you can specify is `0` (zero), the maximum value that you can specify is `2147483647` The default value is `0` (zero).
Length Reference	Button and Enumerated type	If you have selected the length to be defined by *`Length Reference`*, select the name of the integer object that specifies the length of this object. Make your selection from the displayed list of integer objects that are defined as siblings of the current object, and occur before it in the structure of the message. For information about reordering elements, see “Reordering objects” on page 2900.

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. The length is given in bytes. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value for the *`Length Count`* or *`Length Reference`* property. The default is `Bytes`.
Justification	Enumerated type	If you have set the *`Physical Type`* property to `Fixed Length String`, select `Left Justify` (the default value) or `Right Justify` from the list. If you have selected another value for *`Physical Type`*, this property is inactive.

Property	Type	Meaning
Padding Character	String	If you have set the *Physical Type* property to `Fixed Length String`, and the *Justification* property is either `Left Justify` or `Right Justify`, this property is applicable. When writing an output message, use the padding character to fill out the remaining character positions when the string length is less than the length implied by the *Length* or *Length Reference* property. Whether the string is padded from the left or the right is governed by the *Justification* property. When parsing an input message, the padding character is trimmed from the end of the string. Whether the string is trimmed from the left or the right is governed by the *Justification* property. Specify this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the displayed list. • Enter a character between quotation marks; for example, `"c"` or `'c'`, where c is any alphanumeric character. • Enter a Unicode value in the form `U+xxxx` where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form `0xYY`, where `YY` is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, it is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the displayed list: • `NULLPadFill`. This is valid only if *`Physical Type`* is `Fixed Length String`. The field is filled with the value specified by the *`Padding Character`*. *`Encoding Null Value`* must be set to an empty string. • `NULLLogicalValue`. The *`Encoding Null Value`* is transformed to match the required format for the field. The default value. • `NULLLiteralValue`. The *`Encoding Null Value`* is directly substituted as if it is a string. • `NULLLiteralFill`. The field is filled with the value specified by the *`Encoding Null Value`*. *`Encoding Null Value`* must resolve to a single character.
Encoding Null Value	STRING	The use of this property depends on the *`Encoding Null`* property. If specified, its length must be equal to the length of the string element, except for `NULLLiteralFill`. The default value is empty (not set). If you set the *`Encoding Null`* property to `NULLLiteralFill`, the value must resolve to a single character. Set the character in one of the following ways: • Select `SPACE`, `NUL`, `0x00` or `0xFF` from the displayed list • Enter a character between quotation marks, for example `'c'` or `"c"`, where `c` is any alphanumeric character. • Enter a hexadecimal character code in the form `0xYY` where `YY` is a hexadecimal value. • Enter a decimal character code in the form `YY` where `YY` is a decimal value. • Enter a Unicode value in the form `U+xxxx` where `xxxx` is a Unicode value specified in hexadecimal format.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is `0`, the minimum value is `0`, and the maximum value is `999999`. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property. For repeating objects, this property is applied to all instances.

Occurrences

Property	Type	Meaning
Repeat Reference	Enumerated type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message. Select an integer object from the displayed list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* logical property when parsing and writing the message, but not for validation of the message.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Embedded simple type properties" on page 6180
Logical, CWF, XML, TDS, and documentation properties for embedded simple types.

"Embedded simple type CWF properties" on page 6079
The properties, and their permissible values, vary according to the object type.

Embedded simple type XML properties:

The properties, and their permissible values, vary according to the object type.

The properties that are displayed on the object page, and the values that those properties can take, can vary according to the type of the object. For example, the properties for type *string* are different from those of type *Boolean*. Select the link for the object type from the following table.

Type of object	Properties
Binary types	• ComIbmMrm _AnonBinary
Boolean types	• ComIbmMrm _AnonBoolean
DateTime types	• ComIbmMrm _AnonDate • ComIbmMrm _AnonDateTime • ComIbmMrm _AnonGDay • ComIbmMrm _AnonGMonth • ComIbmMrm _AnonGMonthDay • ComIbmMrm _AnonGYear • ComIbmMrm _AnonGYearMonth • ComIbmMrm _AnonTime
Decimal types	• ComIbmMrm _AnonDecimal
Float types	• ComIbmMrm _AnonFloat
Integer types	• ComIbmMrm _AnonInt
String types	• ComIbmMrm _AnonString

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM Custom Wire Format" on page 1214
Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not separated by delimiters.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Embedded simple type properties" on page 6180
Logical, CWF, XML, TDS, and documentation properties for embedded simple types.

"Embedded simple type logical properties" on page 6074
The logical properties of an embedded simple type include properties that specify the number of occurrences of the embedded simple type.

"Embedded simple type CWF properties" on page 6079
The properties, and their permissible values, vary according to the object type.

"Embedded simple type TDS properties" on page 6088
The properties, and their permissible values, vary according to the object type.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

XML Wire Format properties for embedded simple type binary types:

Physical representation.

The XML wire format properties described here apply to:

- Objects: Embedded simple types

Physical representation

Property	Type	Meaning
Encoding	String	Select one of the following values from the drop-down list: : • `CDatahex` (the default). Hexadecimal values in this field are specified with the CDATA qualifier, for example <e1><![CDATA[62]]></e1> • `hex`. Hexadecimal values in this field are specified as digits only, for example <e1>62</e1>. • `base64`. Values in this field are specified as digits only, coded in base 64.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Embedded simple type properties" on page 6180
Logical, CWF, XML, TDS, and documentation properties for embedded simple types.

"Embedded simple type XML properties" on page 6084
The properties, and their permissible values, vary according to the object type.

XML wire format properties for embedded simple type Boolean types:

There are no properties to show.

The XML wire format properties described here apply to:

- Objects: Embedded simple types

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Embedded simple type properties" on page 6180
Logical, CWF, XML, TDS, and documentation properties for embedded simple types.

"Embedded simple type XML properties" on page 6084
The properties, and their permissible values, vary according to the object type.

XML wire format properties for embedded simple type dateTime types:

Physical representation.

The XML wire format properties described here apply to:

- Objects: Embedded simple types

Physical representation

Property	Type	Meaning
DateTime Format	String	Specify a format string that specifies the rendering of the value for dateTime elements. The default dateTime format is dependent on the logical type of the object. For information about the defaults for the dateTime format according to the logical type see "DateTime defaults by logical type" on page 6320. See "DateTime formats" on page 6310 for details of dateTime formats.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Embedded simple type properties" on page 6180
Logical, CWF, XML, TDS, and documentation properties for embedded simple types.

"Embedded simple type XML properties" on page 6084
The properties, and their permissible values, vary according to the object type.

XML wire format properties for embedded simple type decimal types:

There are no properties to show.

The XML Wire Format properties described here apply to:

- Objects: Embedded simple types

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Embedded simple type properties" on page 6180
Logical, CWF, XML, TDS, and documentation properties for embedded simple types.

"Embedded simple type XML properties" on page 6084
The properties, and their permissible values, vary according to the object type.

XML wire format properties for embedded simple type float types:

There are no properties to show.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Embedded simple type properties" on page 6180
Logical, CWF, XML, TDS, and documentation properties for embedded simple types.

"Embedded simple type XML properties" on page 6084
The properties, and their permissible values, vary according to the object type.

XML wire format properties for embedded simple type integer types:

There are no properties to show.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Embedded simple type properties" on page 6180
Logical, CWF, XML, TDS, and documentation properties for embedded simple types.

"Embedded simple type XML properties" on page 6084
The properties, and their permissible values, vary according to the object type.

XML wire format properties for embedded simple type string types:

There are no properties to show.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Embedded simple type properties" on page 6180
Logical, CWF, XML, TDS, and documentation properties for embedded simple types.

"Embedded simple type XML properties" on page 6084
The properties, and their permissible values, vary according to the object type.

Embedded simple type TDS properties:

The properties, and their permissible values, vary according to the object type.

The properties that are displayed on the object page, and the values that those properties can take, can vary according to the type of the object. For example, the properties for type *string* are different from those of type *Boolean*. Select the link for the object type from the following table.

Type of object	Properties
Binary types	• ComIbmMrm _AnonBinary
Boolean types	• ComIbmMrm _AnonBoolean
DateTime types	• ComIbmMrm _AnonDate • ComIbmMrm _AnonDateTime • ComIbmMrm _AnonGDay • ComIbmMrm _AnonGMonth • ComIbmMrm _AnonGMonthDay • ComIbmMrm _AnonGYear • ComIbmMrm _AnonGYearMonth • ComIbmMrm _AnonTime
Decimal types	• ComIbmMrm _AnonDecimal
Float types	• ComIbmMrm _AnonFloat
Integer types	• ComIbmMrm _AnonInt
String types	• ComIbmMrm _AnonString

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM Custom Wire Format" on page 1214
Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not separated by delimiters.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Embedded simple type properties" on page 6180
Logical, CWF, XML, TDS, and documentation properties for embedded simple types.

"Embedded simple type logical properties" on page 6074
The logical properties of an embedded simple type include properties that specify the number of occurrences of the embedded simple type.

"Embedded simple type CWF properties" on page 6079
The properties, and their permissible values, vary according to the object type.

"Embedded simple type XML properties" on page 6084
The properties, and their permissible values, vary according to the object type.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

TDS properties for embedded simple type binary types:

Field identification and physical representation.

The TDS properties described here apply to:

- Objects: Embedded simple types

Field identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the `Data Element Separation` property of the complex type or types in which the object is a child is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, this property must contain a non-empty value. If the object is a complex element, and the `Data Element Separation` property of its parent is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, the property can contain an empty value. The value for this property must be unique for every element in the message set; that is, no two elements in the message set can contain the same value for this property.

Property	Type	Meaning
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the `Data Element Separation` method is set to `Use Data Pattern` in the complex type. For more details, see "Regular expression syntax" on page 6304.
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the `Data Element Separation` property of the parent group or complex type is set to `All Elements Delimited` or `Variable Length Elements Delimited`. A default value is provided if the previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; if the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used. If none of the previous conditions are true, a default is not applied.

Physical representation

Property	Type	Meaning
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no `Length Reference` is specified. The default is dependent on the setting of the message set property `Derive default length from logical type`. If `Derive default length from logical type` is selected, the default value is derived from any length or maxLength value constraint (schema facet) on the object's simple type.
Length Reference	Enumerated type	This property is applicable only if `Physical Type` is `Text`, `Binary`, or `TLOG Specific`. If set, this property takes precedence over any value in the `Length Units` property. Specify the identifier of a sibling integer object, the value of which determines the length of the object in question. The sibling object must be defined before the current object within the message structure. For information about reordering elements, see "Reordering objects" on page 2900.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Embedded simple type properties" on page 6180
Logical, CWF, XML, TDS, and documentation properties for embedded simple types.

"Embedded simple type TDS properties" on page 6088
The properties, and their permissible values, vary according to the object type.

TDS properties for embedded simple type Boolean types:

Field identification and physical representation.

The TDS properties described here apply to:

- Objects: Embedded simple types

Field identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the `Data Element Separation` property of the complex type or types in which the object is a child is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, this property must contain a non-empty value. If the object is a complex element, and the `Data Element Separation` property of its parent is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, the property can contain an empty value. The value for this property must be unique for every element in the message set; that is, no two elements in the message set can contain the same value for this property.
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the `Data Element Separation` method is set to `Use Data Pattern` in the complex type. For more details, see "Regular expression syntax" on page 6304.
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the `Data Element Separation` property of the parent group or complex type is set to `All Elements Delimited` or `Variable Length Elements Delimited`. A default value is provided if the previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; if the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used. If none of the previous conditions are true, a default is not applied.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the `Messaging Standard` property of the message set is `User Defined Text`, `User Defined Mixed`, `CSV`, or `TLOG`, select one of the following values: • `Text`. The data is in character format. • `Binary`. The data is in bit string format. The default is dependent on the `Messaging Standard` property. For all other `Messaging Standard` values, the `Physical Type` property is set to `Text`.
Length Reference	Enumerated type	This property is applicable only if `Physical Type` is `Text`, `Binary`, or `TLOG Specific`. If set, this property takes precedence over any value in the `Length Units` property. Specify the identifier of a sibling integer object, the value of which determines the length of the object in question. The sibling object must be defined before the current object within the message structure. For information about reordering elements, see "Reordering objects" on page 2900.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Embedded simple type properties" on page 6180
Logical, CWF, XML, TDS, and documentation properties for embedded simple types.

"Embedded simple type TDS properties" on page 6088
The properties, and their permissible values, vary according to the object type.

TDS properties for embedded simple type dateTime types:

Field identification, physical representation, and null values.

The TDS properties described here apply to:

- Objects: Embedded simple types

Field identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the `Data Element Separation` property of the complex type or types in which the object is a child is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, this property must contain a non-empty value. If the object is a complex element, and the `Data Element Separation` property of its parent is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, the property can contain an empty value. The value for this property must be unique for every element in the message set; that is, no two elements in the message set can contain the same value for this property.
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the `Data Element Separation` method is set to `Use Data Pattern` in the complex type. For more details, see "Regular expression syntax" on page 6304.
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the `Data Element Separation` property of the parent group or complex type is set to `All Elements Delimited` or `Variable Length Elements Delimited`. A default value is provided if the previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; if the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used. If none of the previous conditions are true, a default is not applied.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the `Messaging Standard` property of the message set is `User Defined Text`, `User Defined Mixed`, `CSV`, or `TLOG`, select one of the following values: • `Text`. The data is in character format. • `Length Encoded String 1`. The first byte of the data contains the length (in length units) of the data string that follows the length byte. The maximum length of a `Length Encoded String 1` element is 255 length units. • `Length Encoded String 2`. The first two bytes of the data contain the length (in length units) of the data string that follows the two length bytes. The maximum length of a `Length Encoded String 2` element is 65535 length units. • `Null Terminated String`. The data string ends with the hexadecimal NULL character, `X'00'`. • `Packed Decimal`. The data is a packed decimal number that equates to the COMP-3 data type in COBOL. • `Binary`. The data is in bit string format. • `Time Seconds`. This value supports C time_t, and Java Date and Time objects. It is valid only if the `DateTime Format` property represents numeric-only data. • `Time Milliseconds`. This value supports C time_t, and Java Date and Time objects. It is valid only if the `DateTime Format` property represents numeric-only data. The default is dependent on the `Messaging Standard` property. For all other `Messaging Standard` values, the `Physical Type` property is set to `Text`.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no `Length Reference` is specified.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • `Not Applicable` • `Left Justify` • `Right Justify`

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the drop-down list. • Enter a character between quotation marks, for example "c" or 'c', where c is any alphanumeric character. • Enter a hexadecimal character code in the form `0xYY`, where `YY` is a hexadecimal value. • Enter a Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.
DateTime Format	String	Specify a template for date and time. The default DateTime format is dependent on the logical type of the object. For information about the defaults for the dateTime format according to the logical type, see “DateTime defaults by logical type” on page 6320. See “DateTime formats” on page 6310 for details of date and time formats.
Length Reference	Enumerated type	This property is applicable only if `Physical Type` is `Text`, `Binary`, or `TLOG Specific`. If set, this property takes precedence over any value in the `Length Units` property. Specify the identifier of a sibling integer object, the value of which determines the length of the object in question. The sibling object must be defined before the current object within the message structure. For information about reordering elements, see “Reordering objects” on page 2900.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the list: • `NULLPadFill`. This option is valid only for fixed-length objects and is the default value. • `NULLLogicalValue`. The `Encoding Null Value` property is first converted to an actual value, and rendered in the way specified for the field. • `NULLLiteralValue`. This option specifies that `Encoding Null Value` contains a value that is directly substituted as if it is a string. For dateTime elements, use this option if you want to use the `Encoding Null Value` property to test or compare the content of the field in the message. • `NULLLiteralFill`. This option specifies that the field is filled with the value specified by the `Encoding Null Value` property. `Encoding Null Value` must resolve to a single character. The option that you select determines the value that you must set for the property `Encoding Null Value`. For full information about using these options, see "TDS Null handling options" on page 6293.
Encoding Null Value	String	The use of this property depends on the `Encoding Null` property. The default value is zero. If you set the `Encoding Null` property for a dateTime object to `NULLLogicalValue`, the value that you set must be in an ISO8601 dateTime format. These formats are described in "DateTime as string data" on page 6311. For example, specify a value that conforms to the `yyyy-MM-dd'T'HH:mm:ss` format; for example, `1970-12-01`.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Embedded simple type properties" on page 6180
Logical, CWF, XML, TDS, and documentation properties for embedded simple types.

"Embedded simple type TDS properties" on page 6088
The properties, and their permissible values, vary according to the object type.

TDS properties for embedded simple type decimal types:

Field identification, physical representation, numeric representation, and null values.

The TDS properties described here apply to:

- Objects: Embedded simple types

Field identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the `Data Element Separation` property of the complex type or types in which the object is a child is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, this property must contain a non-empty value. If the object is a complex element, and the `Data Element Separation` property of its parent is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, the property can contain an empty value. The value for this property must be unique for every element in the message set; that is, no two elements in the message set can contain the same value for this property.
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the `Data Element Separation` method is set to `Use Data Pattern` in the complex type. For more details, see "Regular expression syntax" on page 6304.
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the `Data Element Separation` property of the parent group or complex type is set to `All Elements Delimited` or `Variable Length Elements Delimited`. A default value is provided if the previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; if the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used. If none of the previous conditions are true, a default is not applied.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the `Messaging Standard` property of the message set is `User Defined Text`, `User Defined Mixed`, `CSV`, or `TLOG`, select one of the following values: • `Text`. The data is in character format. • `Packed Decimal`. The data is a packed decimal number that equates to the COMP-3 data type in COBOL. • `TLOG Specific` - this option can be selected only if the `Message Standard` property of the message set is `TLOG`. This option indicates that the format of the data is specific to the TLOG messaging standard. • `Integer`. Equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • `External Decimal`. The data is a decimal number that equates to the data type PIC 9 USAGE DISPLAY in COBOL. The default is dependent on the `Messaging Standard` property. For all other `Messaging Standard` values, the `Physical Type` property is set to `Text`.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no `Length Reference` is specified. If this property is not set and the message set property `Derive default length from logical type` is selected, and the `Physical type` is 'Character', the default value is derived from any totalDigits value constraint (schema facet) or, if none, any `minInclusive`, `maxInclusive`, `minExclusive`, or `maxExclusive` value constraints (schema facets), on the simple type.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • `Not Applicable` • `Left Justify` • `Right Justify`

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the drop-down list. • Enter a character between quotation marks, for example "c" or 'c', where c is any alphanumeric character. • Enter a hexadecimal character code in the form `0xYY`, where `YY` is a hexadecimal value. • Enter a Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.
Length Reference	Enumerated type	This property is applicable only if `Physical Type` is `Text`, `Binary`, or `TLOG Specific`. If set, this property takes precedence over any value in the `Length Units` property. Specify the identifier of a sibling integer object, the value of which determines the length of the object in question. The sibling object must be defined before the current object within the message structure. For information about reordering elements, see “Reordering objects” on page 2900.

Numeric representation

Property	Type	Meaning
Virtual Decimal Point	Button and Integer	Specify a non-zero integer that represents the position of an implied decimal point within a number, or specify 0, the default, to use the formatting of Float or Decimal numbers as specified by the `Precision` property. If you specify a positive integer, the position of the decimal point is moved left from the right side of the number. For example, if you specify 3, the decimal value `1234` represents 1.234 If you specify a negative integer, the position of the decimal point is moved right from the right side of the number. For example, if you specify `-3`, the decimal value `1234` represents 1,234,000.
Precision	Button and Integer	This property is applicable only if `Physical Type` is `Text`. It is used if the value of the `Virtual Decimal Point` property is 0, which indicates that the decimal point is present in the data. It deals with truncation, and specifies how many digits are to follow the decimal point. Either specify a number of digits: • If you set `Precision` to 0, data is truncated so that the fractional part is lost. For example, the value `123.45` is truncated to `123`. • If you set `Precision` to a number less than the number of fractional digits, data is truncated. For example, the value `123.4567` is truncated to `123.45` if you set `Precision` to 2. • If you set `Precision` to a number greater than the number of fractional digits, the value is padded with extra zeros. For example, the value `12.345` is padded to `12.34500` if you set `Precision` to 5. Or select one option from the list: • `All Significant Digits - decimal separator only required if fractional digits` (the default): all significant digits are written to the output bit stream, and no decimal separator is written if no fractional digits are present. • `Explicit Decimal Separator - decimal separator always required`: all significant digits are written to the output bit stream and the decimal separator is always included, even when no fractional digits are present. The decimal separator must be present in the input bit stream, even when no fractional digits are present.
Sign Orientation	Enumerated type	The values that you can choose for this property are: • `Leading Separate` • `Trailing Separate` • `Leading Overpunched` • `Trailing Overpunched` This property is enabled only if you have set `Physical Type` to `Text` or `External Decimal`, and you have selected `Signed`. If `Physical Type` is `Text`, the only valid values of `Sign Orientation` are `Leading Separate` and `Trailing Separate`. If `Physical Type` is `External Decimal` and `Sign EBCDIC Customer Overpunched` is selected, the only valid values of `Sign Orientation` are `Leading Overpunched` and `Trailing Overpunched`.
Positive Sign	String	Specify the value that represents the positive symbol. Do not specify a numeric value. If no value is set, "+" is assumed. The positive sign is not written when creating an output message; it is used only to recognize the positive sign when parsing a message bit stream. This property is applicable only if `Physical Type` is `Text` and `Signed` is selected.

Property	Type	Meaning
Negative Sign	String	Specify the value that represents the negative symbol. Do not specify a numeric value. If no value is set, "-" is assumed. This property is applicable only if `Physical Type` is `Text` and `Signed` is selected.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the list: • `NULLPadFill`. This option is valid only for fixed-length objects and is the default value. • `NULLLogicalValue`. The `Encoding Null Value` property is first converted to an actual value, and rendered in the way specified for the field. • `NULLLiteralValue`. This option specifies that `Encoding Null Value` contains a value that is directly substituted as if it is a string. For dateTime elements, use this option if you want to use the `Encoding Null Value` property to test or compare the content of the field in the message. • `NULLLiteralFill`. This option specifies that the field is filled with the value specified by the `Encoding Null Value` property. `Encoding Null Value` must resolve to a single character. The option that you select determines the value that you must set for the property `Encoding Null Value`. For full information about using these options, see “TDS Null handling options” on page 6293.
Encoding Null Value	String	The use of this property depends on the `Encoding Null` property. The default value is zero. If you set the `Encoding Null` property for a dateTime object to `NULLLogicalValue`, the value that you set must be in an ISO8601 dateTime format. These formats are described in “DateTime as string data” on page 6311. For example, specify a value that conforms to the `yyyy-MM-dd'T'HH:mm:ss` format; for example, `1970-12-01`.

Related concepts:

“Message modeling” on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

“The message model” on page 1160
The message model consists of the following components.

“MRM TDS format: Relationship to the logical model” on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

“Constructing message models” on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

“Working with MRM message model objects” on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Embedded simple type properties" on page 6180
Logical, CWF, XML, TDS, and documentation properties for embedded simple types.

"Embedded simple type TDS properties" on page 6088
The properties, and their permissible values, vary according to the object type.

TDS properties for embedded simple type float types:

Field identification, physical representation, numeric representation, and null values.

The TDS properties described here apply to:

- Objects: Embedded simple types

Field identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the `Data Element Separation` property of the complex type or types in which the object is a child is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, this property must contain a non-empty value. If the object is a complex element, and the `Data Element Separation` property of its parent is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, the property can contain an empty value. The value for this property must be unique for every element in the message set; that is, no two elements in the message set can contain the same value for this property.
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the `Data Element Separation` method is set to `Use Data Pattern` in the complex type. For more details, see "Regular expression syntax" on page 6304.
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the `Data Element Separation` property of the parent group or complex type is set to `All Elements Delimited` or `Variable Length Elements Delimited`. A default value is provided if the previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; if the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used. If none of the previous conditions are true, a default is not applied.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the `Messaging Standard` property of the message set is `User Defined Text`, `User Defined Mixed`, `CSV`, or `TLOG`, select one of the following values: • `Text`. The data is in character format. • `Float`. Equates to the data type FLOAT or DOUBLE in C, or the COMP-1 or COMP-2 numeric data type in COBOL. • `Integer`. Equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • `Packed Decimal`. The data is a packed decimal number that equates to the COMP-3 data type in COBOL. • `External Decimal`. The data is a decimal number that equates to the data type PIC 9 USAGE DISPLAY in COBOL. The default is dependent on the `Messaging Standard` property. For all other `Messaging Standard` values, the `Physical Type` property is set to `Text`.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no `Length Reference` is specified.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • `Not Applicable` • `Left Justify` • `Right Justify`

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the drop-down list. • Enter a character between quotation marks, for example "c" or 'c', where c is any alphanumeric character. • Enter a hexadecimal character code in the form `0xYY`, where `YY` is a hexadecimal value. • Enter a Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.
Length Reference	Enumerated type	This property is applicable only if `Physical Type` is `Text`, `Binary`, or `TLOG Specific`. If set, this property takes precedence over any value in the `Length Units` property. Specify the identifier of a sibling integer object, the value of which determines the length of the object in question. The sibling object must be defined before the current object within the message structure. For information about reordering elements, see “Reordering objects” on page 2900.

Numeric representation

Property	Type	Meaning
Virtual Decimal Point	Button and Integer	Specify a non-zero integer that represents the position of an implied decimal point within a number, or specify 0, the default, to use the formatting of Float or Decimal numbers as specified by the `Precision` property. If you specify a positive integer, the position of the decimal point is moved left from the right side of the number. For example, if you specify `3`, the decimal value `1234` represents 1.234 If you specify a negative integer, the position of the decimal point is moved right from the right side of the number. For example, if you specify `-3`, the decimal value `1234` represents 1,234,000.
Precision	Button and Integer	This property is applicable only if `Physical Type` is `Text`. It is used if the value of the `Virtual Decimal Point` property is 0, which indicates that the decimal point is present in the data. It deals with truncation, and specifies how many digits are to follow the decimal point. Either specify a number of digits: • If you set `Precision` to 0, data is truncated so that the fractional part is lost. For example, the value `123.45` is truncated to `123`. • If you set `Precision` to a number less than the number of fractional digits, data is truncated. For example, the value `123.4567` is truncated to `123.45` if you set `Precision` to 2. • If you set `Precision` to a number greater than the number of fractional digits, the value is padded with extra zeros. For example, the value `12.345` is padded to `12.34500` if you set `Precision` to 5. Or select one option from the list: • `All Significant Digits - decimal separator only required if fractional digits` (the default): all significant digits are written to the output bit stream, and no decimal separator is written if no fractional digits are present. • `Explicit Decimal Separator - decimal separator always required`: all significant digits are written to the output bit stream and the decimal separator is always included, even when no fractional digits are present. The decimal separator must be present in the input bit stream, even when no fractional digits are present. • `Exponential Notation - Example "1.23456e002"`: data is written out to the bit stream as a signed value having the format `[sign1]a.bbbe[sign2]ccc` where: – `[sign1]` is the value of `Negative Sign` if the value is negative – `a` is a single decimal digit – `bbb` is one or more decimal digits – `[sign2]` is the value of `Negative Sign` if the exponent is negative – `ccc` is exactly three decimal digits (the exponent) `[sign1]` and `[sign2]` are absent if the value and exponent are positive. For example, the value -123.456 is represented as `-1.23456e002` and the value 0.00012 is represented as `1.2e-004` in the output bit stream, assuming that the value of `Negative Sign` is "-", and the value of `Sign Orientation` is `Leading`. The value -0.00012 is represented as `1.2*e*004` if `Negative Sign` is "*" and `Sign Orientation` is `Trailing`.

Property	Type	Meaning
Sign Orientation	Enumerated type	The values that you can choose for this property are: • Leading Separate • Trailing Separate • Leading Overpunched • Trailing Overpunched This property is enabled only if you have set Physical Type to Text or External Decimal, and you have selected Signed. If Physical Type is Text, the only valid values of Sign Orientation are Leading Separate and Trailing Separate. If Physical Type is External Decimal and Sign EBCDIC Customer Overpunched is selected, the only valid values of Sign Orientation are Leading Overpunched and Trailing Overpunched.
Positive Sign	String	Specify the value that represents the positive symbol. Do not specify a numeric value. If no value is set, "+" is assumed. The positive sign is not written when creating an output message; it is used only to recognize the positive sign when parsing a message bit stream. This property is applicable only if Physical Type is Text and Signed is selected.
Negative Sign	String	Specify the value that represents the negative symbol. Do not specify a numeric value. If no value is set, "-" is assumed. This property is applicable only if Physical Type is Text and Signed is selected.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the list: • NULLPadFill. This option is valid only for fixed-length objects and is the default value. • NULLLogicalValue. The Encoding Null Value property is first converted to an actual value, and rendered in the way specified for the field. • NULLLiteralValue. This option specifies that Encoding Null Value contains a value that is directly substituted as if it is a string. For dateTime elements, use this option if you want to use the Encoding Null Value property to test or compare the content of the field in the message. • NULLLiteralFill. This option specifies that the field is filled with the value specified by the Encoding Null Value property. Encoding Null Value must resolve to a single character. The option that you select determines the value that you must set for the property Encoding Null Value. For full information about using these options, see “TDS Null handling options” on page 6293.
Encoding Null Value	String	The use of this property depends on the Encoding Null property. The default value is zero. If you set the Encoding Null property for a dateTime object to NULLLogicalValue, the value that you set must be in an ISO8601 dateTime format. These formats are described in “DateTime as string data” on page 6311. For example, specify a value that conforms to the yyyy-MM-dd'T'HH:mm:ss format; for example, 1970-12-01.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Embedded simple type properties" on page 6180
Logical, CWF, XML, TDS, and documentation properties for embedded simple types.

"Embedded simple type TDS properties" on page 6088
The properties, and their permissible values, vary according to the object type.

TDS properties for embedded simple type integer types:

Field identification, physical representation, numeric representation, and null values.

The TDS properties described here apply to:

- Objects: Embedded simple types

Field identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the `Data Element Separation` property of the complex type or types in which the object is a child is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, this property must contain a non-empty value. If the object is a complex element, and the `Data Element Separation` property of its parent is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, the property can contain an empty value. The value for this property must be unique for every element in the message set; that is, no two elements in the message set can contain the same value for this property.
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the `Data Element Separation` method is set to `Use Data Pattern` in the complex type. For more details, see "Regular expression syntax" on page 6304.
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the `Data Element Separation` property of the parent group or complex type is set to `All Elements Delimited` or `Variable Length Elements Delimited`. A default value is provided if the previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; if the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used. If none of the previous conditions are true, a default is not applied.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the `Messaging Standard` property of the message set is `User Defined Text`, `User Defined Mixed`, `CSV`, or `TLOG`, select one of the following values: • `Text`. The data is in character format. • `Integer`. Equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • `Packed Decimal`. The data is a packed decimal number that equates to the COMP-3 data type in COBOL. • `External Decimal`. The data is a decimal number that equates to the data type PIC 9 USAGE DISPLAY in COBOL. The default is dependent on the `Messaging Standard` property. For all other `Messaging Standard` values, the `Physical Type` property is set to `Text`.

Property	Type	Meaning
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no `Length Reference` is specified. If this property is not set and the message set property `Derive default length from logical type` is selected, and the `Physical type` is 'Character', the default value is derived from any totalDigits value constraint (schema facet) or, if none, any `minInclusive`, `maxInclusive`, `minExclusive`, or `maxExclusive` value constraints (schema facets), on the simple type.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • `Not Applicable` • `Left Justify` • `Right Justify`

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the drop-down list. • Enter a character between quotation marks, for example "c" or 'c', where c is any alphanumeric character. • Enter a hexadecimal character code in the form `0xYY`, where `YY` is a hexadecimal value. • Enter a Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.
Length Reference	Enumerated type	This property is applicable only if `Physical Type` is `Text`, `Binary`, or `TLOG Specific`. If set, this property takes precedence over any value in the `Length Units` property. Specify the identifier of a sibling integer object, the value of which determines the length of the object in question. The sibling object must be defined before the current object within the message structure. For information about reordering elements, see “Reordering objects” on page 2900.

Numeric representation

Property	Type	Meaning
Sign Orientation	Enumerated type	The values that you can choose for this property are: • Leading Separate • Trailing Separate • Leading Overpunched • Trailing Overpunched This property is enabled only if you have set Physical Type to Text or External Decimal, and you have selected Signed. If Physical Type is Text, the only valid values of Sign Orientation are Leading Separate and Trailing Separate. If Physical Type is External Decimal and Sign EBCDIC Customer Overpunched is selected, the only valid values of Sign Orientation are Leading Overpunched and Trailing Overpunched.
Positive Sign	String	Specify the value that represents the positive symbol. Do not specify a numeric value. If no value is set, "+" is assumed. The positive sign is not written when creating an output message; it is used only to recognize the positive sign when parsing a message bit stream. This property is applicable only if Physical Type is Text and Signed is selected.
Negative Sign	String	Specify the value that represents the negative symbol. Do not specify a numeric value. If no value is set, "-" is assumed. This property is applicable only if Physical Type is Text and Signed is selected.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the list: • NULLPadFill. This option is valid only for fixed-length objects and is the default value. • NULLLogicalValue. The Encoding Null Value property is first converted to an actual value, and rendered in the way specified for the field. • NULLLiteralValue. This option specifies that Encoding Null Value contains a value that is directly substituted as if it is a string. For dateTime elements, use this option if you want to use the Encoding Null Value property to test or compare the content of the field in the message. • NULLLiteralFill. This option specifies that the field is filled with the value specified by the Encoding Null Value property. Encoding Null Value must resolve to a single character. The option that you select determines the value that you must set for the property Encoding Null Value. For full information about using these options, see “TDS Null handling options” on page 6293.

Property	Type	Meaning
Encoding Null Value	String	The use of this property depends on the `Encoding Null` property. The default value is zero. If you set the `Encoding Null` property for a dateTime object to `NULLLogicalValue`, the value that you set must be in an ISO8601 dateTime format. These formats are described in "DateTime as string data" on page 6311. For example, specify a value that conforms to the `yyyy-MM-dd'T'HH:mm:ss` format; for example, `1970-12-01`.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Embedded simple type properties" on page 6180
Logical, CWF, XML, TDS, and documentation properties for embedded simple types.

"Embedded simple type TDS properties" on page 6088
The properties, and their permissible values, vary according to the object type.

TDS properties for embedded simple type string types:

Field identification, physical representation, and null values.

The TDS properties described here apply to:

- Objects: Embedded simple types

Field identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the `Data Element Separation` property of the complex type or types in which the object is a child is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, this property must contain a non-empty value. If the object is a complex element, and the `Data Element Separation` property of its parent is `Tagged Delimited`, `Tagged Fixed Length`, or `Tagged Encoded Length`, the property can contain an empty value. The value for this property must be unique for every element in the message set; that is, no two elements in the message set can contain the same value for this property.
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the `Data Element Separation` method is set to `Use Data Pattern` in the complex type. For more details, see "Regular expression syntax" on page 6304.
Interpret Element Value	Enumerated type	Specify whether values stored within this object are interpreted as having significance for the parser and, if so, the type of interpretation that occurs. This interpretation is standard-specific and is therefore hard coded. The possible values for this property are: • `None` (the default value) • `EDIFACT Service String` • `X12 Service String` • `Message Key` • `EDIFACT Syntax Level ID` • `HL7 Service String` • `HL7 Field Separator`
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the `Data Element Separation` property of the parent group or complex type is set to `All Elements Delimited` or `Variable Length Elements Delimited`. A default value is provided if the previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; if the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used. If none of the previous conditions are true, a default is not applied.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the `Messaging Standard` property of the message set is `User Defined Text`, `User Defined Mixed`, `CSV`, or `TLOG`, select one of the following values: • `Text`. The data is in character format. • `Length Encoded String 1`. The first byte of the data contains the length (in length units) of the data string that follows the length byte. The maximum length of a `Length Encoded String 1` element is 255 length units. • `Length Encoded String 2`. The first two bytes of the data contain the length (in length units) of the data string that follows the two length bytes. The maximum length of a `Length Encoded String 2` element is 65535 length units. • `Null Terminated String`. The data string ends with the hexadecimal NULL character, `X'00'`. • `TLOG Specific` - this option can be selected only if the `Message Standard` property of the message set is `TLOG`. This option indicates that the format of the data is specific to the TLOG messaging standard. The default is dependent on the `Messaging Standard` property. For all other `Messaging Standard` values, the `Physical Type` property is set to `Text`.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no `Length Reference` is specified. If this property is not set and the message set property `Derive default length from logical type` is selected, and the `Physical type` is 'Character', the default value is derived from any length or maxLength value constraint (schema facet) on the simple type.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • `Not Applicable` • `Left Justify` • `Right Justify`

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the drop-down list. • Enter a character between quotation marks, for example "c" or 'c', where c is any alphanumeric character. • Enter a hexadecimal character code in the form `0xYY`, where `YY` is a hexadecimal value. • Enter a Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.
Length Reference	Enumerated type	This property is applicable only if `Physical Type` is `Text`, `Binary`, or `TLOG Specific`. If set, this property takes precedence over any value in the `Length Units` property. Specify the identifier of a sibling integer object, the value of which determines the length of the object in question. The sibling object must be defined before the current object within the message structure. For information about reordering elements, see “Reordering objects” on page 2900.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the list: • `NULLPadFill`. This option is valid only for fixed-length objects and is the default value. • `NULLLogicalValue`. The `Encoding Null Value` property is first converted to an actual value, and rendered in the way specified for the field. • `NULLLiteralValue`. This option specifies that `Encoding Null Value` contains a value that is directly substituted as if it is a string. For dateTime elements, use this option if you want to use the `Encoding Null Value` property to test or compare the content of the field in the message. • `NULLLiteralFill`. This option specifies that the field is filled with the value specified by the `Encoding Null Value` property. `Encoding Null Value` must resolve to a single character. The option that you select determines the value that you must set for the property `Encoding Null Value`. For full information about using these options, see "TDS Null handling options" on page 6293.
Encoding Null Value	String	The use of this property depends on the `Encoding Null` property. The default value is zero. If you set the `Encoding Null` property for a dateTime object to `NULLLogicalValue`, the value that you set must be in an ISO8601 dateTime format. These formats are described in "DateTime as string data" on page 6311. For example, specify a value that conforms to the `yyyy-MM-dd'T'HH:mm:ss` format; for example, `1970-12-01`.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Deprecated message model object properties" on page 6069
Some objects in the message model are deprecated, but you can reference the information for their properties.

"Embedded simple type properties" on page 6180
Logical, CWF, XML, TDS, and documentation properties for embedded simple types.

"Embedded simple type TDS properties" on page 6088
The properties, and their permissible values, vary according to the object type.

Documentation properties for all message set objects:

Use the documentation property of an object to add information that describes the function of the object.

The documentation property is available on all objects except Key, Keyref, and Unique objects.

The property is a string field, and you can use any standard alphanumeric characters.

Related concepts:

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

Additional MRM domain information

More information about the MRM domain.

- "Message set preferences" on page 5366
- "Message set properties" on page 5371
- "Message definition file properties" on page 5409
- "Message category properties" on page 5413
- "Message model object properties" on page 5416
- "Deprecated message model object properties" on page 6069
- "Generated model representations" on page 6338
- "Import formats" on page 6346
- "Message model wizards" on page 6360
- "MRM restrictions" on page 6252
- "Data types for elements in an MRM message" on page 6254
- "Additional CWF information" on page 6255
- "Additional XML information" on page 6257
- "Additional TDS information" on page 6264
- "DateTime formats" on page 6310
- "Message model task list errors that have a quick fix" on page 6336

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with a message set" on page 2840
Complete a variety of tasks that are involved in working with a message set.

"Working with a message definition file" on page 2863
Create, open, and delete a message definition file.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message set properties" on page 5371
Message sets have properties that you can set to define their characteristics and the way in which they are processed.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

MRM restrictions:

The MRM parser does not exactly follow the XML Schema 1.0 specification.

However, the XMLNSC domain fully complies with the XML Schema 1.0 specification when validation is enabled. All of the constructs that are mentioned in this topic are supported by the XMLNSC domain.

XML Schema features supported only in the message editor:
The following features can be created and edited using the message editor, but are not honored by the MRM domain.

- *Pattern facet on non-string data types*. The message broker only validates pattern facets that are applied to simple types based on xsd:string.
- *White space facet*. The message broker does not use the white space facet. However, if necessary, white space facets can be included in the message model. You can accurately control the processing of white space by using the settings on the physical formats.
- *ID attribute*. The message model can contain attributes with the name 'id', but these will not be checked for uniqueness.

XML Schema exceptions:

The following features can be created and edited using the message editor, but the MRM domain processes them in a way that differs from the XML Schema specification.

- *Default and fixed values*. The processing of default and fixed values depends on the physical format in which the message is parsed. For details on how each physical format uses these fields, refer to the concept topic *Relationship to the logical model* for the relevant physical format.
- *xsi:type attribute*. The xsi:type attribute is not automatically processed by the message broker. An attribute with the name 'xsi:type' can be included in the message model, and can be processed using a message flow.

Differences in validation:
If validation is enabled in a message flow, the following features or scenarios are not validated in exactly the same way as a validating XML parser would validate them:

- *Any Element or Any Attribute*. If the message model contains a wildcard ('any element' or 'any attribute'), the message broker validates the 'processContents' field as follows:
  - `skip`. No checking is done; any element or attribute is allowed.
  - `lax`. No checking is done; any element or attribute is allowed.
  - `strict`. Any element or attribute in the same message set is allowed.

  Note: If all of the definitions for a namespace are included within the same message set, the validation of 'strict' is the same as by a validating XML parser.
- *Element substitution and 'all' groups*. If an element can be substituted, and it occurs within an 'all' group, the following exceptions apply to the validation of the element:
  - The element is always validated as if it were optional.
  - An input message is not rejected if more than one of the substitutions is used in the same 'all' group.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"XML Schema" on page 1172
XML Schema is an international standard from the World Wide Web Consortium (W3C) that defines a language for describing the structure of XML documents.

"XML Schema extensions in message sets" on page 1173
WebSphere Message Broker provides some additional facilities that are not specified in the XML Schema 1.0 specification. When using a message set, further extensions are provided.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

Data types for elements in an MRM message:

A parser is supplied for the body of a message in the MRM domain; it associates each field with a specific data type.

For details of mapping from XML Schema data types that you have specified for elements in the MRM to data types used by the broker and supported by ESQL, see "ESQL to XML Schema data type mapping" on page 5044. When you create an element, you might find that associated value constraints are created to ensure a more accurate mapping of the XML Schema type.

Simple type - list

In the message tree, a list type will be represented as a name node with an anonymous value child for each list item. This allows repeating lists to be handled without any loss of information. Repeating lists will appear as sibling name elements, each of which has its own anonymous value child nodes for its respective list items.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Developing Java" on page 2628
When you use the JavaCompute node, you customize it to determine the exact processing that it provides.

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Manipulating messages in the MRM domain" on page 2581
How to use messages that have been modeled in the MRM domain, and that are parsed by the MRM parser.

Related reference:

"ESQL to XML Schema data type mapping" on page 5044
Mapping from XML Schema simple type to ESQL message tree data type.

"Data types of fields and elements" on page 4237

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message

models.

Additional CWF information:

Information about data conversion and options for unll handling.

- "CWF data conversion"
- "CWF Null handling options" on page 6256

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

CWF data conversion:

You can convert an MRM message to a different code page or encoding, or both.

To do this, set the CodedCharSetId and Encoding fields in the Properties folder and the message tree to the target value.

The data conversion that is performed is dependent on the simple type of each element:

- Binary schema types: base64Binary, hexBinary objects are not converted.
- Boolean schema types: Boolean objects are not converted.
- DateTime schema types: date, dateTime, gDay, gMonth, gMonthDay, gYear, gYearMonth, time objects are handled as binary, string, packed decimal, timeSeconds, or timeMilliseconds.

  If a dateTime element is defined as binary, it is not converted.

  If it is defined as string, it is converted as a string element (described later in this section).

  If it is defined as a packed decimal value, it is converted as Decimal (described later in this section).

If it is defined as a timeSeconds or timeMilliseconds value, it is converted as Integer (described later in this section).

- Decimal schema types: decimal, integer, negativeInteger, nonNegativeInteger, nonPositiveInteger, positiveInteger, unsignedLong objects with *Physical Type* set to `External Decimal` are converted to the target CodedCharSetId. Elements with other *Physical Type* settings are converted to the target Encoding.
- Float schema types: double, float objects with *Physical Type* set to `External Decimal` are converted to the target CodedCharSetId. Elements with other *Physical Type* settings are converted to the target Encoding.
- Integer schema types: byte, int, long, short, unsignedByte, unsignedInt, unsignedShort objects with *Physical Type* set to `External Decimal` are converted to the target CodedCharSetId. Elements with other *Physical Type* settings are converted to the target Encoding.
- String schema types: anyURI, ENTITIES, ENTITY, ID, IDREF, IDREFS, language, Name, NCName, NMTOKEN, NMTOKENS, normalizedString, NOTATION, QName, string, token objects are converted to the target CodedCharSetId (the length of an object that has *Physical Type* of `Length Encoded String 2` is converted to the target Encoding).

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

"Additional CWF information" on page 6255
Information about data conversion and options for unll handling.

CWF Null handling options:

The Custom Wire Format (CWF) supports handling of null values within messages. The Boolean Null Value that you set for the message set is applicable for all the defined objects within the message set.

For more information about the use of nulls, refer to the properties *Encoding Null* and *Encoding Null Value* for objects of each simple type, for example, "CWF properties for element reference and local element dateTime types" on page 5630.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

"Additional CWF information" on page 6255
Information about data conversion and options for unll handling.

Additional XML information:

Options for rendering, and null handling.

- "XML Null handling options" on page 6258
- "XML rendering options" on page 6262

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

XML Null handling options:

The XML Wire Format supports the handling of null values in messages. `Encoding null` properties for XML are set only on the message set, and apply to all the defined objects in the message set.

You can use the following two properties to represent the numeric and non-numeric encoding for NULL in the XML Wire Format:
- `Encoding Numeric Null`
- `Encoding Non-Numeric Null`

These properties represent the numeric and non-numeric encoding for NULL respectively.
- The numeric data types are:
  - Decimal schema types: decimal, integer, negativeInteger, nonNegativeInteger, nonPositiveInteger, positiveInteger, unsignedLong
  - Float schema types: double, float
  - Integer schema types: byte, int, long, short, unsignedByte, unsignedInt, unsignedShort
- The non-numeric data types are:
  - Binary schema types: base64Binary, hexBinary
  - Boolean schema types: Boolean
  - DateTime schema types: date, dateTime, gDay, gMonth, gMonthDay, gYear, gYearMonth, time
  - String schema types: anyURI, ENTITIES, ENTITY, ID, IDREF, IDREFS, language, Name, NCName, NMTOKEN, NMTOKENS, normalizedString, NOTATION, QName, string, token

Each of these encodings has the following enumerated values:
- NULLEmpty (default)
- NULLValue
- NULLXMLSchema
- NULLValueAttribute
- NULLAttribute (deprecated)
- NULLElement (deprecated)

You do not have to supply additional information for `NULLEmpty`, `NULLXMLSchema`, and `NULLValueAttribute`, but if you select `NULLValue`, `NULLAttribute`, or `NULLElement`, you must define further values to be assigned to represent the NULL condition in the `Encoding Numeric Null Value` and `Encoding Non-Numeric Null Value` message set properties.

The following table shows how each encoding works. For each encoding, the example XML causes the element `myElem` to be given a value NULL.

Encoding Numeric Null Encoding Non-Numeric Null	Encoding Numeric Null Value Encoding Non-Numeric Null Value	Example XML
NULLEmpty		`<myElem/>` `<myElem></myElem>`
NULLValue	zzz	`<myElem>zzz</myElem>`
NULLXMLSchema		`<myElem xsi:nil='true'/>`[1] [5]
NULLValueAttribute		`<myElem></myElem>`[2] `<parent id="myElem"></parent>`[3]
NULLElement	null[4]	`<myElem><null/></myElem>`
NULLAttribute	null[4]	`<myElem null='true'/>`[1]

Notes:

1. The attribute must evaluate to true, so the value must be `true`, `1`, or the `Boolean True Value` property.
2. This value is valid only for `XMLElementAttrVal` element rendering, as specified in "XML rendering options" on page 6262. Marking an element as being rendered in this way, and setting it to null, is equivalent to removing the attribute of the element that provides the element's value.
3. This value is valid only for `XMLElementAttrIdVal` element rendering, as specified in "XML rendering options" on page 6262. Marking an element as being rendered in this way, and setting it to null, is equivalent to removing the attribute of the element that provides the element's value, but not removing the attribute that provides the element's name.
4. Both `NULLElement` and `NULLAttribute` are deprecated. The element or attribute name provided must not include a namespace URI or prefix. If namespaces are enabled for the message set, the name matches any namespace.
5. `xsi:nil` is not supported with complex elements of MRM-XML.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

"Additional XML information" on page 6257
Options for rendering, and null handling.

"XML Null value"
When you set the *Encoding Null Num* property to NULLValue in XML, the value is taken as a literal.

"XML Null representation for Binary data" on page 6262
If you use the *Encoding Null Non-Num Val* field with a binary object in XML, enter the appropriate hex value.

"XML Wire Format message set properties" on page 5400
The following tables define the properties for the XML Wire Format for the message set.

XML Null value:

When you set the *Encoding Null Num* property to NULLValue in XML, the value is taken as a literal.

A direct comparison is done with the text string, and no logical data conversion is performed. This behavior is in contrast to the TDS and CWF formats.

For example, if you set the message set property *Encoding Null Num* to the value NULLValue, and you set *Encoding Null Num Val* to 0, a FLOAT value of 0.0 or a DECIMAL value of +0 does not match NULL.

Seting *Encoding Null Num* to NULLEmpty is equivalent to setting *Encoding Null Num* to NULLValue and *Encoding Null Num Val* to "".

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

"Additional XML information" on page 6257
Options for rendering, and null handling.

"XML Null handling options" on page 6258
The XML Wire Format supports the handling of null values in messages. `Encoding null` properties for XML are set only on the message set, and apply to all the defined objects in the message set.

XML Null element and NullValAttr:
In XML there are two conventions for storing a value:

1. It can be stored as an XML attribute with a local element or element reference property *`Render`* set to `XMLAttribute`, `XMLElement`, `XMLElementAttrID`, `XMLElementAttrVal`, or `XMLElementAttrIDVal`. For example, `<element1 val="12"></element1>`.
2. It can be stored as XML content with a local element or element reference property *`Render`* set to `XMLElement`. For example, `<element1>12</element1>`.

If you set the message set property *`Encoding Null Num`* to `NULLElement`, there is no way to represent a null value for an attribute value. If a null value is present in the tree (from ESQL or another format), an attribute with an empty string is written in the output message.

Conversely, if you have set the message set property *`Encoding Null Num`* or *`Encoding Null Non-Num`* to `NULLValAttr`, there is no way to represent a null value for a value rendered as XML content. If a null value is present in the tree, when writing an empty string, an element with no character content is written out instead.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

"Additional XML information" on page 6257
Options for rendering, and null handling.

"XML Null handling options" on page 6258
The XML Wire Format supports the handling of null values in messages. `Encoding null` properties for XML are set only on the message set, and apply to all the defined objects in the message set.

XML Null representation for Binary data:

If you use the *`Encoding Null Non-Num Val`* field with a binary object in XML, enter the appropriate hex value.

Do not insert the word CDATA in this field. If `CDataHex` is specified in the *`Encoding`* XML property, CDATA rendering is used when the message is written.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

"Additional XML information" on page 6257
Options for rendering, and null handling.

"XML Null handling options" on page 6258
The XML Wire Format supports the handling of null values in messages. `Encoding null` properties for XML are set only on the message set, and apply to all the defined objects in the message set.

XML rendering options:

You can use four properties on the XML layer that to affect how the XML messages are rendered.

The following table shows examples of the values that you can set for the *`Member Render`* property. In this table, the member element is referred to as *A*, and has the value `value of element`. The parent is referred to as *X*.

The effect of rendering options on XML output

To get XML rendered like this:	Set this Member Render property value:	Set these other property values:
<X> <A>value of element</A> </X>	XMLElement (the default)	Member XML Name = A
<X A='value of element'/>	XMLAttribute	Member XML Name = A
<X> <Field id='A'>value of element</Field> </X>	XMLElementAttrID	Member XML Name = Field Member ID Attribute Name = id Member ID Attribute Value = A
<X> <A val=value of element/> </X>	XMLElementAttrVal	Member XML Name = A Member Value Attribute Name = val
<X> <Field id='A' val='value of element'/> </X>	XMLElementAttrIDVal	Member XML Name = Field Member ID Attribute Name = id Member ID Attribute Value = A Member Value Attribute Name = val

You should not have an element in the model that is rendered as an XML attribute. This can result in incorrect validation of XML documents. Instead the element should be redefined as an attribute in the model.

You should not have an attribute in the model that is rendered as an XML element. This can result in incorrect validation of XML documents. Instead the attribute should be defined as an element in the model.

There is one scenario where this technique is appropriate. When you have created a message model by importing a C header file or a COBOL copybook, it will consist entirely of elements. An XML form of this model can be created by simply adding an XML physical format to the message set. If you are looking for certain elements to appear as XML attributes in the XML form, you can use the `Render` property to achieve this.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM XML physical format" on page 1247
The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

"Configuring physical properties" on page 2912
Working with the physical properties of message model objects.

Related reference:

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Physical properties for message model objects" on page 5455
CWF, XML, and TDS format physical properties for message model objects.

"Custom Wire Format physical properties for message model objects" on page 5455
Custom wire format physical property information is available for some objects.

"TDS format physical properties for message model objects" on page 5501
Some objects have TDS wire format properties.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"Message model object properties by object" on page 5536
The following objects have properties that can be viewed or set.

Additional TDS information:

More information about the TDS physical format.

- "TDS Industry standard formats" on page 6265
- "Message characteristics" on page 6281
- "TDS Null handling options" on page 6293
- "TDS message model integrity" on page 6295
- "Using regular expressions to parse data elements" on page 6301

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

TDS Industry standard formats:

WebSphere Message Broker supports the ACORD AL3, CSV, EDIFACT, FIX, HL7, SWIFT, TLOG, and X12 standards.

For more details about each of these industry standards, see:

- "CSV messaging standard" on page 6276
- "EDIFACT messaging standard"
- "HL7 messaging standard" on page 6267
- "SWIFT messaging standard" on page 6268
- "TLOG messaging standard" on page 6270
- "X12 messaging standard" on page 6271
- "ACORD AL3 messaging standard" on page 6272
- "FIX messaging standard" on page 6275
- "IDoc messaging standard" on page 6278

These topics also contain details of any predefined message sets that are available from IBM. Default property values are supplied as defined in "Default TDS message set properties" on page 5394.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

"Additional TDS information" on page 6264
More information about the TDS physical format.

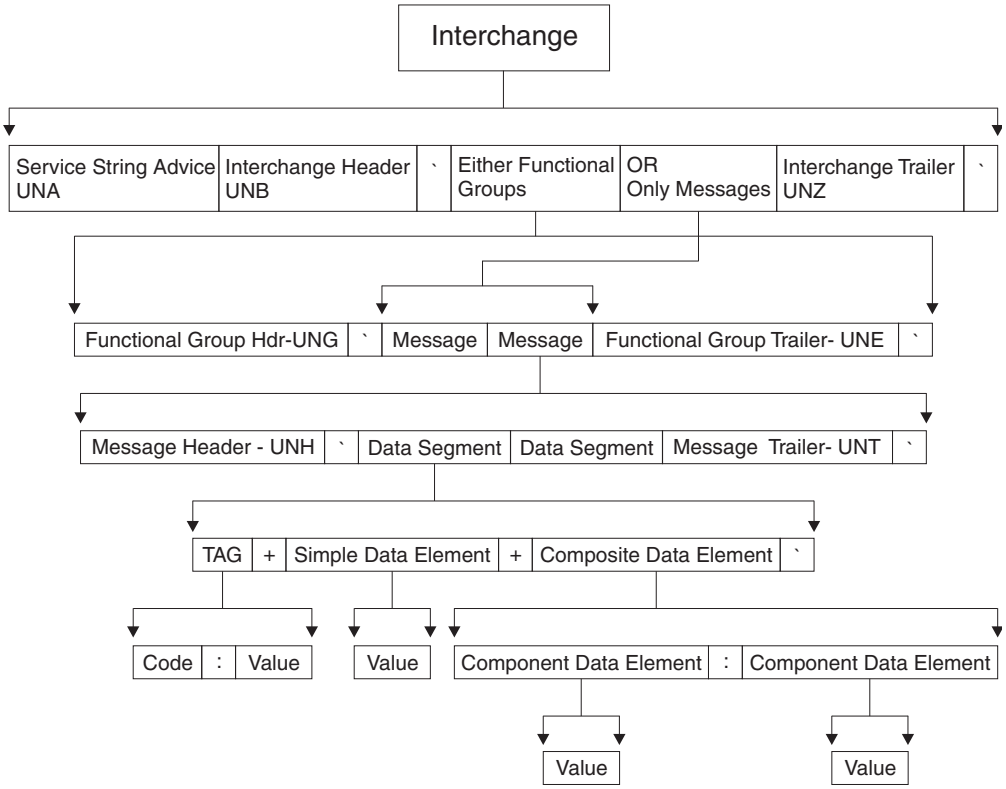
EDIFACT messaging standard:

EDIFACT, an international standard for EDI trading in commercial and non-commercial sectors, has an underlying syntax that is an ISO standard.

Within that syntax, there are directories of data elements, composite data elements, segments, and messages. There are conventions for placing messages in an "envelope" which identifies the sender and receiver and other attributes of a transmission. For more information about the EDIFACT messaging standard, see the United Nations Centre for Trade Facilitation and Electronic Business Web site and click "Standards" on the left side.

EDIFACT messages can be modeled using the MRM Tagged/Delimited String Format (TDS).

The high-level structure of an EDIFACT message is as follows:



You can model the top-level interchange of an EDIFACT message by setting the following properties for the complex type on which the message is based:

Composition = Sequence
Content Validation = Closed
Tag Data Separator = <EDIFACT_TAGDATA_SEP>
Data Element Separation = Tagged Delimited
Delimiter = <EDIFACT_CS>

Within an EDIFACT message, you can define the delimiters to be used in the message itself using the optional Service String Advice element. To enable this element to be recognized as an EDIFACT Service String, you must set the element property *Interpret Element Value* to EDIFACT Service String. You must also set the delimiter values to the mnemonic values that are defaulted when you set the *Message Standard* property to EDIFACT.

A predefined message set solution for EDIFACT can be purchased from IBM.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

"Additional TDS information" on page 6264
More information about the TDS physical format.

"TDS Industry standard formats" on page 6265
WebSphere Message Broker supports the ACORD AL3, CSV, EDIFACT, FIX, HL7, SWIFT, TLOG, and X12 standards.

HL7 messaging standard:

The HL7 messaging standard defines the structure and content of messages that are exchanged between systems in various administrative, financial, and clinical activities in the healthcare industry.

HL7 messages can be modeled using the MRM Tagged/Delimited String Format (TDS).

If you are working with HL7 messages you can specify the messaging standard at the message set level and a number of the properties for this standard are set to default settings for HL7 at the message set, complex type, group, and element levels.

Predefined HL7 message sets are available as part of the following Healthcare sample:

- Healthcare

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Related concepts:

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"TDS Industry standard formats" on page 6265
WebSphere Message Broker supports the ACORD AL3, CSV, EDIFACT, FIX, HL7, SWIFT, TLOG, and X12 standards.

"Additional TDS information" on page 6264
More information about the TDS physical format.

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

SWIFT messaging standard:

SWIFT supplies secure, standardized messaging services and interface software to financial institutions.

SWIFT FIN messages can be modeled using the MRM Tagged/Delimited String Format (TDS).

The high-level block structure of a SWIFT message is shown in the following table.

SWIFT message high level block structure

Block name	Format
Basic header	{1:...}
Application header	{2:...}
User header	{3:...}
Text	{4:...}
Trailer	{5:...}

When they are concatenated in a message, the blocks appear as:
{1:...}{2:...}{3:...}{4:...}{5:...}

You can model this setting the following type properties for the message:

```
Data Element Separation = Tagged Delimited
Group Indicator = {
Delimiter = }{
Group Terminator = }
Tag Data Separator = :
```

Each block is modeled as a complex element with element *Tag* property values of 1,2,3,4, and 5 respectively.

The text body of the message has the following format:

```
{4:
:20:X
:32A:940930USD1,
.....
:72:/A/
-}
```

You can model the complex type of the Text body by setting the following type properties:

```
Data Element Separation = Tagged Delimited
Group Indicator = <CR><LF>:
Delimiter = <CR><LF>:
Group Terminator = <CR><LF>-
Tag Data Separator = :
```

The *Tag* property of the elements within the body has values of 20, 32A, 72, and so on.

A predefined message set solution for SWIFT can be purchased from IBM. See the WebSphere MQ SupportPacs web page.

Swift is a cooperative owned by the financial industry. For more information about the SWIFT messaging standard, see the SWIFT community website.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

"Additional TDS information" on page 6264
More information about the TDS physical format.

"TDS Industry standard formats" on page 6265
WebSphere Message Broker supports the ACORD AL3, CSV, EDIFACT, FIX, HL7, SWIFT, TLOG, and X12 standards.

TLOG messaging standard:

In the retail industry, a TLOG is the Point of Sale (POS) Transaction Log.

The TLOG is a complete, detailed record of everything that occurs at the Point of Sale (POS) terminal, including events that are not directly related to a sales transaction. Typically, the precise TLOG record format is unique to a given POS application, but most formats are based on a tagged/delimited string format called Raw TLOG.

Raw TLOG messages can be modeled using the MRM Tagged/Delimited String Format (TDS).

If you are working with TLOG messages you can specify whether fields in the messages are in character format or in a format that is specific to the message. The message format needs the *Messaging Standard* property (at the message set level) to be set to `TLOG`, and that relevant objects that have this non-character based field in the TDS message have the *Physical Type* property set to `TLOG Specific`.

Predefined TLOG message sets are available as part of the following TLOG Processor samples:

- TLOG Processor

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

"Additional TDS information" on page 6264
More information about the TDS physical format.

"TDS Industry standard formats" on page 6265
WebSphere Message Broker supports the ACORD AL3, CSV, EDIFACT, FIX, HL7, SWIFT, TLOG, and X12 standards.

X12 messaging standard:

X12 is a standard for EDI trading in commercial and non-commercial sectors. X12 has an underlying syntax, which is an ANSI standard.

Within that syntax, there are directories of data elements, composite data elements, segments, and messages. There are conventions for placing messages in an "envelope" which identifies the sender and receiver and other attributes of a transmission. For more information on the X12 messaging standard, see the ASC X12 website.

X12 messages can be modeled using the MRM Tagged/Delimited String Format (TDS).

If you are working with X12 messages, you can define the delimiters to be used in the message itself using the mandatory Interchange Control Header element. To enable this element to be recognized as an X12 Service String, you must set the element property *Interpret Element Value* to `X12 Service String`. You must also set the delimiter values to the mnemonic values defaulted by setting the *Message Standard* property to `X12`.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message

models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

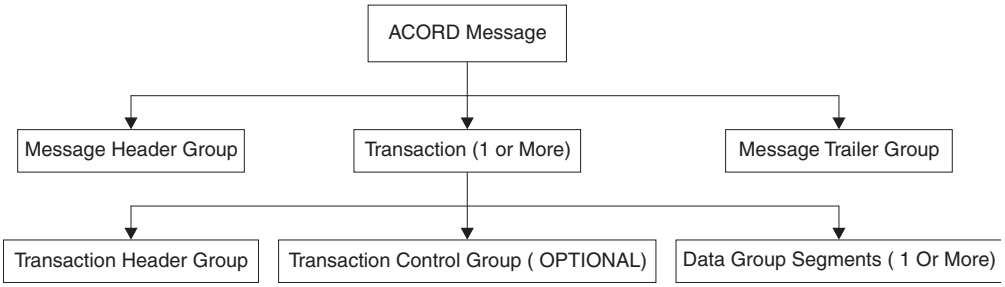
"Additional MRM domain information" on page 6251
More information about the MRM domain.

"Additional TDS information" on page 6264
More information about the TDS physical format.

"TDS Industry standard formats" on page 6265
WebSphere Message Broker supports the ACORD AL3, CSV, EDIFACT, FIX, HL7, SWIFT, TLOG, and X12 standards.

ACORD AL3 messaging standard:

The basic structure of an ACORD AL3 message.



Each group with an ACORD AL3 message has a header consisting of a one-digit number, three letters, plus a three-digit total length count. These first seven characters can be modeled as a tag. The data within the headers is fixed length. Therefore the header type used for the overall message can be modeled as follows:

```
Data Element Separation = Tagged Fixed Length
Length of Tag = 7
```

The Transaction Group contains other groups, and is therefore modeled in the same way as the overall message. The Message Header Group and the Message Trailer group consist of fixed-length elements, therefore the type used can be modeled as:

```
Data Element Separation = Fixed Length
```

Two *Data Element Separation* methods are suited to handling ACORD AL3 messages:

- `Fixed Length AL3` supports basic handling of ACORD AL3 messages, including situations where the message groups conform to a different version of the ACORD AL3 standard. This is deprecated and will be removed in a future version of the product; an alternative will be provided.
- `Tagged Encoded Length` supports handling of more sophisticated situations, including messages containing message groups unknown to the message dictionary.

The following sections describe their use:

- "Using Fixed Length AL3" on page 6273
- "Using Tagged Encoded Length to support reversioning" on page 6274

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

"Additional TDS information" on page 6264
More information about the TDS physical format.

"TDS Industry standard formats" on page 6265
WebSphere Message Broker supports the ACORD AL3, CSV, EDIFACT, FIX, HL7, SWIFT, TLOG, and X12 standards.

Using Fixed Length AL3:

Using Fixed Length AL3 is deprecated.

Fixed Length AL3 will be removed in a future version of the product; an alternative will be provided.

You can select the value `Fixed Length AL3` for the *`Data Element Separation`* property for complex types within a message that conforms to the ACORD AL3 standard. Different versions of the ACORD AL3 standard can be supported using the same message set. This value is like the value `Fixed Length` except for the following:

- A question mark (?) in the left-most position of an element means that it is skipped.
- A sequence of question marks is inserted for all missing optional elements.
- Unused trailing optional elements are truncated.
- Any <CR><LF> after the last element is ignored.
- The length field is extracted on input (and *not* put to the tree), and automatically recalculated on output.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined

structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

"Additional TDS information" on page 6264
More information about the TDS physical format.

"TDS Industry standard formats" on page 6265
WebSphere Message Broker supports the ACORD AL3, CSV, EDIFACT, FIX, HL7, SWIFT, TLOG, and X12 standards.

"ACORD AL3 messaging standard" on page 6272
The basic structure of an ACORD AL3 message.

Using Tagged Encoded Length to support reversioning:

Parse tags containing groups that are not in the current ACORD AL3 standards.

The incoming message might contain a group that is no longer in use within the current ACORD AL3 standards, and has therefore been deleted from the later version of the standards. Similarly, the incoming bit stream might be from a later version of the ACORD AL3 standards, and might contain a new group that was not defined in earlier versions.

In order to correctly parse this self-defining tag, the TDS parser needs to know the length of the group it is parsing and skip to the end of all data associated with that self-defining tag.

Use the *Data Element Separation* method `Tagged Encoded Length` to handle these situations. You must also set these properties:

- `Length of Tag` or `Tag Data Separator`, so that the TDS parser knows where tags end.
- `Length of Encoded Length`, so that the TDS parser knows the size of the length field.
- `Extra Chars in Encoded Length`, are used to indicate to the TDS parser how many characters, apart from the data itself, are counted in the encoded length field.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

"Additional TDS information" on page 6264
More information about the TDS physical format.

"TDS Industry standard formats" on page 6265
WebSphere Message Broker supports the ACORD AL3, CSV, EDIFACT, FIX, HL7, SWIFT, TLOG, and X12 standards.

"ACORD AL3 messaging standard" on page 6272
The basic structure of an ACORD AL3 message.

FIX messaging standard:

FIX messages can be modeled using the MRM Tagged/Delimited String Format (TDS).

The *Financial Information eXchange* (*FIX*) Protocol is a series of messaging specifications. It is a global language describing trade-related messages, and is used for automated trading of securities, derivative, and other financial instruments. For more information about the FIX protocol, see the FIX protocol website.

A predefined message set solution for FIX can be purchased from IBM. See the WebSphere MQ SupportPacs web page.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

"Additional TDS information" on page 6264
More information about the TDS physical format.

"TDS Industry standard formats" on page 6265
WebSphere Message Broker supports the ACORD AL3, CSV, EDIFACT, FIX, HL7, SWIFT, TLOG, and X12 standards.

CSV messaging standard:

The comma separated value (CSV) format is a typical format for describing data in tables or spreadsheets.

The CSV format is used to exchange data between database applications or spreadsheet applications. Although the CSV format is widely used, a definitive specification has not been formally documented. However, these are some of the rules that characterize the CSV format:

- Data fields are separated by commas, and groups of data fields are separated by repeating field delimiters (for example, the <CR><LF> combination of ASCII characters).

  Here is a typical CSV message:

  ```
  12345,Smith,John,"3, North Street"<CR><LF>
  41352,Jones,Ivor,"5, South Road"<CR><LF>
  53421,Edwards,David,"10, East Lane"
  ```

- A comma that occurs within a data field is regarded as part of the data, rather than as a field separator, only if the comma is surrounded by quotation marks (").
- A quotation mark character (") that is within a data field that is enclosed within quotation marks must always be 'escaped' by another instance of the quotation mark character.

  For example, `xx"xx` must be written as `"xx""xx"`, and `"xxxx"` must be written as `"""xxxx"""`.
- In an input message, any variable length data field can be enclosed within quotation mark characters, regardless of whether the field contains any special characters such as quotation mark characters, escape characters, or other reserved characters.

The quotation mark characters must occur at the start and end of the data, are stripped from the data when the field is parsed, and are not added to the output tree. For example, the data `A,"B",C` results in an output tree that contains the values A, B, and C.

- If a data field contains two quotation mark characters and nothing else, the quotation mark characters are removed by the parser and the data field is processed in the same way as an empty field.
- In an output message, any data field that contains a quotation mark character, or any of the special characters that are specified in the TDS message set Reserved Characters property, has quotation mark characters added.

CSV messages can be modeled by using the MRM Tagged/Delimited String Format (TDS). The default message set property values are shown in "Default TDS message set properties" on page 5394.

The following sample is a message set application that shows you how to model some typical CSV message variants, and how to transform the sample CSV messages to and from XML. The XML messages illustrate the logical structure of the data after it has been parsed.

- Comma Separated Value (CSV)

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

You can also import a sample CSV message model by using the New Message Definition File From IBM Supplied Message wizard.

Related concepts:

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: NULL handling" on page 1240
NULL handling dictates the way in which the MRM parser for TDS messages handles elements that have been set to Null.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

"Importing from IBM supplied messages" on page 2942
You can create a new message definition file from an IBM supplied message.

Related reference:

"TDS Industry standard formats" on page 6265
WebSphere Message Broker supports the ACORD AL3, CSV, EDIFACT, FIX, HL7,

SWIFT, TLOG, and X12 standards.

"TDS Format message set properties" on page 5381
The following tables show the properties that you can set for a TDS format message set.

"Default TDS message set properties" on page 5394
The following tables define the defaults for the message set properties for the TDS Format for each of the industry standard messages that you can define.

"Additional TDS information" on page 6264
More information about the TDS physical format.

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

"New message definition file wizard: IBM supplied message" on page 6366
You can create a new message definition file from an IBM supplied message.

IDoc messaging standard:

Receiving data from SAP systems.

WebSphere Message Broker can receive data from SAP systems in various ways.

Two such ways are:

- ALE IDocs exported from SAP across the WebSphere MQ Link for R3.
- File IDocs exported from SAP to the file system.

Such IDocs are a fixed-length text format, and can be modeled using the MRM domain Tagged/Delimited String Format (TDS).

The IDOC domain is deprecated.

Note: For SAP data that is received from the WebSphere Adapter for SAP, use the DataObject domain.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format" on page 1221
The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

"MRM parser and domain" on page 1111
You can use the MRM domain to parse and write a wide range of message formats.

"IDOC parser and domain" on page 1126
The IDOC domain can be used to process messages that are sent to the broker by SAP R3 clients across the WebSphere MQ link for R3. Such messages are known as

SAP ALE IDocs.

"DataObject parser and domain" on page 1114
Use the DataObject domain to parse and write messages for WebSphere Adapters and CORBA applications.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

"Building the MRM TDS model for an IDoc"
The MRM domain Tagged/Delimited String (TDS) physical format is suitable for parsing and writing SAP ALE IDocs and SAP File IDocs. ALE IDoc messages are exported from SAP across the WebSphere MQ Link for R3. File IDocs are exported from SAP to the file system.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

"Additional TDS information" on page 6264
More information about the TDS physical format.

"TDS Industry standard formats" on page 6265
WebSphere Message Broker supports the ACORD AL3, CSV, EDIFACT, FIX, HL7, SWIFT, TLOG, and X12 standards.

Building the MRM TDS model for an IDoc:

The MRM domain Tagged/Delimited String (TDS) physical format is suitable for parsing and writing SAP ALE IDocs and SAP File IDocs. ALE IDoc messages are exported from SAP across the WebSphere MQ Link for R3. File IDocs are exported from SAP to the file system.

About this task

This topic describes how to build the message model that is required by the MRM parser when parsing and writing SAP ALE and File IDocs using its TDS physical format.

Obtaining the IDoc:
About this task

Create an import file of the required IDoc data for the WebSphere Message Broker Toolkit.

Procedure

1. Log on to an SAP system.
2. Run the supplied transaction `we60`, which extracts the IDoc data as a C header file.
   a. In **Basic Type**, select the IDoc type of interest; for example, `MATMAS02`.

b. Leave the **Control**, **Data**, and **Status** check boxes cleared.
c. Select the **Record types** version. A version 4 IDoc is type **3**.
d. Press **F7** to display a C representation of the IDoc.
e. Click **unconverted**.
f. Select **System** > **List** > **Save** > **Local file**.
g. When prompted, enter a file name and directory for the output from the transaction. The C representation of the IDoc is saved to this C header file.

What to do next

Tip: The exported C header can be imported into the WebSphere Message Broker Toolkit without any further manual processing. This situation was not true in previous releases of WebSphere Message Broker.

Modeling the IDoc:
About this task

Create your message model.

Procedure

1. Switch to the Broker Application Development perspective of the WebSphere Message Broker Toolkit.
2. Use the New Message Set wizard to create a message set for your IDoc. Select text data as the data to use. This action creates a Tagged/Delimited String Format (TDS) physical format, and presets the *Default message domain* property to MRM.
3. Use the Message Set editor to rename the TDS physical format to Text_IDoc.
4. Use the New Message Definition File From IBM supplied message wizard to import a prebuilt model of the overall ALE or File IDoc message structure. This model includes definitions of the DC and DD segments. The prebuilt models are called SAP ALE IDoc and SAP File IDoc. The resultant message definition file is called `ale_idoc.mxsd` or `file_idoc.mxsd`. For information about using the New Message Definition File From IBM supplied message wizard, see "Importing from IBM supplied messages" on page 2942.
5. Use the New Message Definition File From C Header File wizard, or the **`mqsicreatemsgdefs`** command, to import the C representation of the IDoc into the new message set. Specify the following settings:
   - Set the Pre-processing option to SAP ALE IDoc or SAP File IDoc. If this option is not specified, the C header is not imported.
   - Create messages for the segments that appear in the IDoc.
   - Use the `String Encoding` option to import character arrays as fixed-length strings.
   - Use the `Padding Char for String` option to make space (" ") the padding character that is used.

   For information about using the wizard, see "Importing from C" on page 2934.

Using the IDoc message model:
About this task

You can now use your message model to help you to construct a message flow that processes instances of your IDoc message, in the same way as any other message that belongs to the MRM domain.

Tip: SupportPac IA0F contains a more detailed description of the steps involved in building the IDoc message model. You can ignore utilities IDocHeaderTweak and IDocMsgSetTweak because that processing has been incorporated into the New Message Definition File From C Header File wizard.

Related concepts:

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

"MRM parser and domain" on page 1111
You can use the MRM domain to parse and write a wide range of message formats.

Related tasks:

"Importing from C" on page 2934
Create a message definition file from a C header file for use in the MRM and IDOC domains.

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Manipulating messages in the MRM domain" on page 2581
How to use messages that have been modeled in the MRM domain, and that are parsed by the MRM parser.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

"Additional TDS information" on page 6264
More information about the TDS physical format.

"TDS Industry standard formats" on page 6265
WebSphere Message Broker supports the ACORD AL3, CSV, EDIFACT, FIX, HL7, SWIFT, TLOG, and X12 standards.

"**mqsicreatemsgdefs** command" on page 3702
Use the **mqsicreatemsgdefs** command to create message definition files.

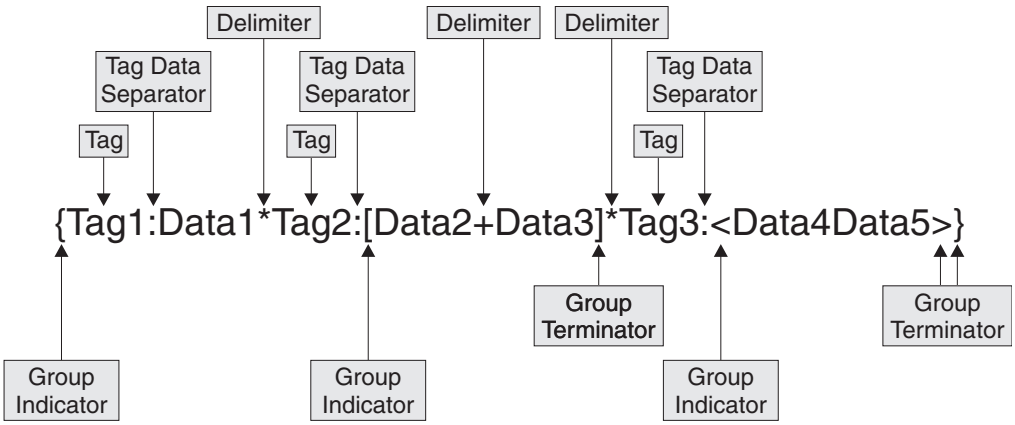
Message characteristics:

Features that are supported by the TDS wire format.

A number of features of text string messages are common across many formats. The following sections give an overview of the main features that are supported by the TDS wire format:

- The text strings in the message can have a tag or a label preceding the data value. The *tag* is a string that uniquely identifies the data value. The TDS format allows you to associate a tag with each element when you define the element in the WebSphere Message Broker Toolkit.
- The message can contain various special characters or strings in addition to the tags and text string data values. The TDS format supports a number of different types of special characters or strings. Some messages have a special character or string that separates each data value from the next. In the TDS format this is a known as a *delimiter*. In formats that have a tag before each data value, the tag can be separated from its data value by a special character or string. In the TDS format this is known as a *tag data separator*.
- A message can be split into a number of substructures in a similar manner to a to COBOL or C structure. You can model each of these substructures separately by defining complex types or elements for each one. Complex types and elements are described in "Message model objects" on page 1174. A substructure can have a special character or string that indicates its start within the data. This is known in the TDS format as a *group indicator*. A substructure can also have a special character or string that indicates its end in the data. In the TDS format, this is known as a *group terminator*. A group indicator and group terminator can also be defined for the whole message. Group indicators and group terminators are optional for the message and each substructure.
- Some text strings within a message can be of fixed length, so a delimiter between each data value is not necessary. This is supported by the TDS format. If you use a fixed length tag, a tagged data separator is not required.
- The TDS property that controls the way text strings are separated is *Data Element Separation*. It has several options that let you choose, for example, if tags are used, if strings lengths are fixed or variable, and what types of text strings are permitted. See "Specifying data element separation methods to model a message" on page 6284.
- The substructures within a message can use different types of *Data Element Separation* and use different special characters. Therefore the TDS format allows you to define different types of data element separation and special characters for each complex type within the message.
- If you use the Use Data Pattern method of *Data Element Separation*, you can use regular expressions to identify parts of the message data to be assigned to subfields. This is done by setting the regular expression in the Data Pattern property. See "Using regular expressions to parse data elements" on page 6301 for further details.

The following figure illustrates the tags and special characters in a TDS message, showing an example data message with each of its components labeled.



- At the top level, each data value has a tag associated with it, each tag is separated from its data value using a tag data separator of colon (:), and the data values are separated from each other using the asterisk delimiter (*).
- The group indicator for the message is the left brace ({) and the group terminator is the right brace (}).
- The data values `Data2` and `Data3` are in a substructure in which there are no tags, and each data element is separated from the next using the delimiter plus (+). The group indicator for this substructure is the left bracket ([) and the group terminator is the right bracket (]).
- The data values `Data4` and `Data5` are in a substructure in which the values are fixed length, and are therefore not separated by a delimiter. The group indicator for this substructure is the less than symbol (<) and the group terminator is the greater than symbol (>).

The following sections describe data element separation and the special characters in more detail:

- "Specifying data element separation methods to model a message" on page 6284
- "Specifying special characters to model a message" on page 6287

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

"Additional TDS information" on page 6264
More information about the TDS physical format.

Specifying data element separation methods to model a message:

Specify the appropriate method of data element separation to identify data elements in a TDS message.

Elements of data in a TDS message are identified according to the data element separation method that you must specify for the *Data Element Separation* property for a complex type. Depending on the value that you have set for *Data Element Separation*, the properties *Tag Data Separator* and *Delimiter* (for a message set and a complex type) might also be required to identify each element.

The following describes the methods that you can specify for each complex type. The examples given are all based on a complex type that contains three elements of type STRING. The *Tag Data Separator*, where used, is the colon (:), and the *Delimiter*, where used, is the asterisk (*).

Tagged Delimited
Each data value is preceded by a tag that is specified as an element property. If the tag has an associated *Length of Tag*, indicating that the tag has a fixed length, each data value follows immediately after the tag. If the tag is not specified as fixed length, the tag is separated from the next element by a *Tag Data Separator*. Each data value is separated from the next by a *Delimiter*. There is no *Delimiter* after the last element in the complex type.

The following example shows tags of fixed length:

```
tag1data1*tag2data2*tag3data3
```

The following example shows tags of variable length:

```
tag1:data1*tag11:data2*tag111:data3
```

Tagged Fixed Length
This method is the similar to Tagged Delimited, but the data values are always fixed length. Therefore, no delimiter is required after each data value. The tags themselves can be fixed length or variable length, depending the setting of *Tag Data Separator* and *Length of Tag*.

The following example shows tags of fixed length:

```
tag1data1tag2data2tag3data3
```

The following example shows tags of variable length:

```
tag1:data1tag11:data2tag111:data3
```

Tagged Encoded Length
This method has a tag and a length field before the data. It indicates to the parser that following each tag in the bit stream there is data defining the length of data to be associated with that tag. You must set the *Length of Encoded Length* parameter. If the value in *Length of Encoded Length* includes extra characters, you must also set the *Extra Chars in Encoded Length* parameter.

The following example shows a tag of fixed length of four characters (*Length of Tag* has been set to four), a three-character length field (*Length of Encoded Length* has been set to three), and several characters of data. *Extra Chars in Encoded Length* has been set to zero:

```
tagA007dataAAAtagB006dataBBtagC009dataCCCCC
```

Given the bit stream above, the parser finds the tag "tagA" and extracts the length value 7. Because *Extra Chars in Encoded Length* is set to zero, the next seven (7 - 0) characters are the data. Then follow the characters for the next tag "tagB" and the length value of 6, and so on for tag "tagC". In each case in this example, the value in the length field is exactly the length of data.

The following example shows tags with a fixed length of four characters (*Length of Tag* has been set to four), a three-character length field (*Length of Encoded Length* has been set to three), and several characters of data. *Extra Chars in Encoded Length* has been set to three (because in this example the length field value includes the three-character length field as well as the data field):

```
tagA012dataAAAAAtagB010dataBBBtagC016dataCCCCCCCCC
```

Given the bit stream above, after "tagA" the parser extracts the length value 12. But because *Extra Chars in Encoded Length* is set to three, only the next nine (12 - 3) characters are the data. Then follow the characters for "tagB" and length value 10, and so on. In each case in this example, the value in the length field is three more than the actual length of data.

All Elements Delimited

The data values have no tag, but each data value is separated from the next by a delimiter.

The following example shows this:

```
data1*data2*data3
```

Variable Length Elements Delimited

If a data element is fixed length, the next data value follows immediately after it. If the data element is variable length, the next data value is separated from it by the delimiter. There are no tags.

The following example shows element 2 as fixed length, and elements 1 and 3 as variable length:

```
data1*data2data3
```

Use Data Pattern

The data associated with each element is determined by the parser matching the data with the regular expression in the *Data Pattern* property for that element. The TDS parser uses the regular expression in the *Data Pattern* to:

- Determine the length of data to associate with each element.
- Determine if, in the case of a repeating element, another occurrence of an element is present in the bit stream.
- Determine the presence (if the pattern is matched) or absence (if the pattern is not matched) of an element in the bit stream.

There are no delimiters or tags, other than those coded as part of the regular expression patterns. See “Regular expression syntax” on page 6304 for an explanation of how pattern matching works.

The following example shows three elements, each having the regular expression *Data Pattern* shown:

```
First Data Pattern = [A-Z]{1,3}
Second Data Pattern = [0-9]+
Third Data Pattern = [a-z]*

Message data = 'DT31758934information for you'

First element data: 'DT'
Second element data: '31758934'
Third element data: 'information'
```

The first *Data Pattern* means "from one to three characters in the range A to Z", the second means "one or more characters in the range 0 to 9", and the third means "zero or more characters in the range a to z". Notice how each element's data was terminated by the first character that did not match the element's *Data Pattern*.

Fixed Length

All elements are fixed length, and each data value immediately follows the next with no delimiter. There are no tags.

The following example shows this:

```
data1data2data3
```

Fixed Length AL3

This method is the same as Fixed Length, but it also notifies the parser to implement a number of rules in relation to missing elements, length encoding, and versioning that are predefined in the ACORD AL3 standard.

Undefined

This value is set automatically when you set the *Type Composition* property of a complex type to `Message`, and you cannot set it to any other value. You are also unable to set values for the TDS Type properties *Group Indicator*, *Group Terminator*, *Tag Data Separator*, *Length of Tag*, and *Delimiter*.

If you set the *Data Element Separation* method to `Undefined`, you must not set the *Type Composition* property to `Empty`, `Choice`, `Unordered Set`, `Ordered Set`, `Sequence`, or `Simple Unordered Set`.

For more information about *Type Composition* set to `Message`, see "Multipart messages" on page 1191.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"Multipart messages" on page 1191
A *multipart message* contains one or more other messages within its structure. The contained message is sometimes referred to as an *embedded message*.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

"Additional TDS information" on page 6264
More information about the TDS physical format.

"Message characteristics" on page 6281
Features that are supported by the TDS wire format.

Specifying special characters to model a message:

You can specify a number of different types of special character in the WebSphere Message Broker Toolkit.

You can also specify special character values for message sets, types, and type members. The values that you set for a type override the corresponding values that are set for the message set in which it is defined.

You can specify a special character value in one of the following ways:

- As a literal string of one or more characters.
- As a mnemonic value.
- As a combination of both mnemonics and literals.

The types of special character are described in the following table.

Special character type	Description	Set as a property of...
Group Indicator	This is a string that indicates the start of a group or complex type within a message	Message set, complex type
Group Terminator	This is a string that indicates that the end of a group or complex type within a message	Message set, complex type
Tag Data Separator	This is the string that is used to separate a tag from its data.	Message set, complex type
Delimiter	This is the string used to separate data elements from one another	Message set, complex type
Repeating Element Delimiter	This is the string used to separate repeating data elements from one another	Local element or element reference
Tag	This is the string that indicates the start of a piece of data.	Local element or global element
Escape character	This is the character that is used to allow special reserved characters (such as delimiters) to be included as part of data	Message set

Special character type	Description	Set as a property of...
Quotation marks	This is the character that is used to allow special reserved characters (such as delimiters) to be included as part of data.	Message set
Reserved characters	These are characters that have a special meaning; for example, escape characters, quotation marks, delimiters, and group indicators, are all examples of reserved characters.	Message set
Decimal point	This is the character that is used as the separator between the integer and fractional components of a decimal number.	Message set

If you create a complex type and set the `Data Element Separation` property to **`Tagged Delimited`**, the `Group Indicator` property to left brace ({) , the Group Terminator to right brace (}), the Tag Data Separator to colon (:), and the Delimiter to asterisk (*), the bit stream has the following format:

```
{tag1:data1*tag2:data2*tag3:data3}
```

In some message formats, a special character is specified before each element or after each element, as shown in the following two examples:

```
:data1:data2:data3
```

```
data1:data2:data3:
```

You can model these formats by using a combination of the Data Element Separation method, the Delimiter value, the Group Indicator value, and the Group Terminator value.

For the first example, specify **`Data Element Separation`** as All Elements Delimited, Delimiter as colon (:), and Group Indicator as colon (:).

For the second example, specify Data Element Separation as All Elements Delimited, Delimiter as colon (:), and Group Terminator as colon (:).

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

"Additional TDS information" on page 6264
More information about the TDS physical format.

"Message characteristics" on page 6281
Features that are supported by the TDS wire format.

Using mnemonics for special characters:

A mnemonic is a tag that is delimited by < and >. The broker translates the mnemonic to obtain the actual value of the special character.

Mnemonics can be used in TDS properties Decimal Point, Escape Character, Reserved Characters, Delimiter, Group Indicator, Tag data Separator, Tag, and Repeating Element Delimiter to specify special characters.

There are two types of mnemonic:

- Control code mnemonics, which map to the common non-printing characters.

  These are mapped using the local code page for your system. This is typically an ASCII code page on distributed platforms and an EBCDIC code page on other platforms.

  This means that characters are generally mapped to the 'expected' values for your system. This depends on your code page setting; for more information, refer to your system documentation. If a specific mnemonic is not mapped to the value that you need, you can use the explicit representation (<U+xxxx>, <0xNN>, or <0XNN>) that is described later in this section.
- Message mnemonics for use with specific industry message standards such as X12.

  These are mapped according to their associated message standard. Each mnemonic has a default mapping, but in message standards such as EDIFACT and X12, this default can be overridden by a 'service string' that is specified in the message itself.

Mnemonics can be specified in one of the following ways:

- `<Mnemonic_Name>`, where `Mnemonic_Name` can comprise alphanumeric characters and underscore (_) characters.
- `<U+xxxx>`, where `xxxx` are hexadecimal digits. The mnemonic is interpreted as the Unicode character that corresponds to the value of the digits.
- `<0xNN> or <0XNN>`, where `N` is a hexadecimal digit. The mnemonic is interpreted as the raw byte value given by the digits.

For more details about the supported mnemonics, see "TDS Mnemonics" on page 5391.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

"Additional TDS information" on page 6264
More information about the TDS physical format.

"Message characteristics" on page 6281
Features that are supported by the TDS wire format.

"Specifying special characters to model a message" on page 6287
You can specify a number of different types of special character in the WebSphere Message Broker Toolkit.

TDS Mnemonics:

The Tagged/Delimited String Format (TDS) uses mnemonics for a number of properties for a message set, complex type, or both.

These TDS mnemonics and their associated properties are listed in the following table.

Mnemonic string	Meaning	Default value	Associated property
<EDIFACT_CS>	Component separator in EDIFACT	:	Message set and complex type or group, `Delimiter`
<EDIFACT_DEC_NOTATION>	Decimal notation in EDIFACT	.	Message set, `Decimal Point`
<EDIFACT_DS>	Data element separator in EDIFACT	+	Message set and complex type or group, `Delimiter`
<EDIFACT_ESC_CHAR>	Escape character in EDIFACT	?	Message set, `Escape Character`
<EDIFACT_GROUP_TERM>	Tag terminator in EDIFACT	'	Message set, `Group Terminator`
<EDIFACT_TAGDATA_SEP>	Tag data separator in EDIFACT This is overridden with the same value as that which overrides <EDIFACT_DS>	+	Message set and complex type or group, `Tag Data Separator`
<HL7_CS>	Component separator in HL7	^	Message set and complex type or group, `Delimiter`

Mnemonic string	Meaning	Default value	Associated property
<HL7_FS>	Data element separator in HL7	|	Message set and complex type or group, `Delimiter`
<HL7_RS>	Repeating element delimiter in HL7	~	Local element and element reference, `Repeating Element Delimiter`
<HL7_SCS>	Sub-component separator in HL7	&	Message set and complex type or group, `Delimiter`
<X12_CS>	Component separator for X12	:	Message set and complex type or group, `Delimiter`
<X12_DS>	Data element separator for X12	*	Message set and complex type or group, `Delimiter`
<X12_ERS>	Element repetition separator for X12	{	Local element and element reference, `Repeating Element Delimiter`
<X12_GROUP_TERM>	Tag terminator in X12	!	Message set level, `Group Terminator`

Mnemonics for control characters are shown in the following table.

Mnemonic	Hex value	Unicode	Description
<ACK>	X'06'	<U+0006>	Acknowledge
<BEL>	X'07'	<U+0007>	Bell
<BS>	X'08'	<U+0008>	Backspace
<CAN>	X'18'	<U+0018>	Cancel
<CR>	X'0D'	<U+000D>	Carriage Return
<DC1>	X'11'	<U+0011>	Device Control One
<DC2>	X'12'	<U+0012>	Device Control Two
<DC3>	X'13'	<U+0013>	Device Control Three
<DC4>	X'14'	<U+0014>	Device Control Four
<DLE>	X'10'	<U+0010>	Data Link Escape
<EM>	X'19'	<U+0019>	End of Medium
<ENQ>	X'05'	<U+0005>	Inquiry
<EOT>	X'04'	<U+0004>	End of Transmission
<ESC>	X'1B'	<U+001B>	Escape
<ETB>	X'17'	<U+0017>	End of Transmission Block
<ETX>	X'03'	<U+0003>	End of Text
<FF>	X'0C'	<U+000C>	Form Feed
<FS>	X'1C'	<U+001C>	File Separator
<GS>	X'1D'	<U+001D>	Group Separator
<GT>	X'3E'	<U+003E>	Greater Than
<HT>	X'09'	<U+0009>	Horizontal Tabulation
<LF>	X'0A'	<U+000A>	Line Feed
<LT>	X'3C'	<U+003C>	Less Than

Mnemonic	Hex value	Unicode	Description
<NAK>	X'15'	<U+0015>	Negative Acknowledge
<NUL>	X'00'	<U+0000>	Null-
<RS>	X'1E'	<U+001E>	Record Separator
<SI>	X'0F'	<U+000F>	Locking Shift Zero (Shift In)
<SO>	X'0E'	<U+000E>	Locking Shift One (Shift Out)
<SOH>	X'01'	<U+0001>	Start of Heading
<SP>	X'20'	<U+0020>	Space
<STX>	X'02'	<U+0002>	Start of Text
<SUB>	X'1A'	<U+001A>	Substitute
<SYN>	X'16'	<U+0016>	Synchronous Idle
<US>	X'1F'	<U+001F>	Unit Separator
<VT>	X'0B'	<U+000B>	Vertical Tabulation

These mnemonics were created for characters that cannot be entered into the message editor.

You can enter a mnemonic in the form `<U+NNNN>`, where `NNNN` are hexadecimal digits. None of the characters in this structure are case-sensitive. Do not enclose spaces inside the angle brackets. These numbers represent a Unicode character, not a character in the code page of the input message.

You can enter a mnemonic in the form `<0xNN>`, where `NN` are hexadecimal digits. None of the characters in this structure are case-sensitive. Do not enclose spaces inside the angle brackets. These numbers represent a raw hexadecimal byte value, not a character in the code page of the input message.

If a mnemonic is of the form `<0xNN>`, it is applied directly to the input data, and no code page conversion takes place. Otherwise, a mnemonic is applied to the data after the data has been converted into Unicode from the code page of the input data.

Related concepts:

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MRM TDS format" on page 1221
The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Working with physical formats" on page 2847
If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

Related reference:

"Message set properties" on page 5371
Message sets have properties that you can set to define their characteristics and the way in which they are processed.

"TDS Format message set properties" on page 5381
The following tables show the properties that you can set for a TDS format message set.

"Default TDS message set properties" on page 5394
The following tables define the defaults for the message set properties for the TDS Format for each of the industry standard messages that you can define.

TDS Null handling options:

TDS supports the handling of null values within messages, provided that the logical Nillable property of the element is set.

You can use the message set property `Boolean Null Representation` to specify the value to be used for Boolean Null representation. You can use the object properties `Encoding Null` and `Encoding Null Value` to control how null handling is represented for individual objects.

You can select the `Encoding Null` property from the enumerated values `NULLPadFill`, `NULLLogicalValue`, `NULLLiteralValue`, and `NULLLiteralFill`:

- Only use the `NULLPadFill` option for fixed length objects. If you select this option for an object of simple type dateTime, a null dateTime is written out, which is an empty tag with a delimiter. (This is equivalent to selecting NullLiteralValue, with the `Encoding Null Value` property set to the empty string "".) If you select this option for an object of another simple type, the object is filled with the value specified by the `Padding Character` property. If you select this option, the `Encoding Null Value` property is disabled.
- If you select the `NULLLogicalValue` option, the value entered for the `Encoding Null Value` property is converted to its logical value. For writing, the logical value is written in the same way as any other value. For parsing, the converted logical value is compared against the converted message data.
- If you select the `NULLLiteralValue` option, the value entered for the `Encoding Null Value` property is directly substituted as if it were a string value. The value is not case sensitive. For fixed length objects, the literal value must be no longer than the length of the object.

  If the literal value is shorter, the `Encoding Null Value` is padded (using `Padding Character`) on output. On input, if the `NULLLiteralValue`'s length does not match the `Length` field, set the message set level `Trim Fix Len String` property so that padded nulls are correctly parsed.
- If you select the `NULLLiteralFill` option, the value entered for the `Encoding Null Value` property is interpreted as a single character string value. Therefore, each character of the value of the element in the bit stream must match exactly the character value specified, to be interpreted as a null value.

The use of the `Encoding Null Value` property is dependent on the value that you select for the `Encoding Null` property described above. Null values are not defined for binary types. The properties `Encoding Null` and `Encoding Null Value` are therefore not set for binary types.

Handling missing fields in a delimited format

When dealing with delimited message formats, it is common for fields to be empty. For example, in a line-oriented format, blank lines might be inserted to separate lines of data.

```
This is Line 1<CR><LF>
<CR><LF>
This is Line 3<CR><LF>
This is Line 4
```

If the TDS property `Suppress Absent Element Delimiters` of the parent complex type is set to `Never`, such a message is successfully parsed, but the blank line does not appear in the message tree:

```
MRM
   - line1 = 'This is Line 1'
   - line3 = 'This is Line 3'
   - line4 = 'This is Line 4'
```

If you need to preserve the blank lines in the message tree, you can use TDS null handling to treat the blank line as NULL. Configure the following properties on the element:

- Select `Nillable` .
- Set `TDS Encoding Null` to `NullLiteralValue`.
- Leave `TDS Encoding Null Value` empty.

The message tree then looks like:

```
MRM
   - line1 = 'This is Line 1'
   - line2 = NULL
   - line3 = 'This is Line 3'
   - line4 = 'This is Line 4'
```

The example above assumes that each line is modeled as an element of simple type string. If each line is modeled as an element of complex type, with each line consisting of a repeating number of word elements, set the three null handling properties on the word element instead, because an element of complex type cannot have a null value.

The message tree then looks like:

```
MRM
   - line1
        - word = 'This'
        - word = 'is'
        - word = 'Line'
        - word = '1'
   - line2
        - word = NULL
   - line3
        - word = 'This'
        - word = 'is'
        - word = 'Line'
        - word = '3'
   - line4
        - word = 'This'
        - word = 'is'
        - word = 'Line'
        - word = '4'
```

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

"Additional TDS information" on page 6264
More information about the TDS physical format.

TDS message model integrity:

When you use the TDS wire format, you must conform to a number of rules that apply to the setting of values of properties. These rules are checked whenever the project is saved.

If an inconsistency is found, the error is displayed in the task list of the WebSphere Message Broker Toolkit.

The following sections cover the rules for TDS wire format properties:

- "General rules: TDS message model integrity" on page 6296
- "Restrictions for nesting complex types" on page 6298
- "Omission and truncation of elements" on page 6300

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with a message definition file" on page 2863
Create, open, and delete a message definition file.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

"Additional TDS information" on page 6264
More information about the TDS physical format.

General rules: TDS message model integrity:

General rules govern each value that you can set for the *Data Element Separation* property of a type.

Tagged Delimited

- The *Tag* property for every simple child element must contain a non-empty value.

Tagged Encoded Length

- The *Tag* property for every simple child element must contain a non-empty value.
- The *Length Of Encoded Length* property must contain a positive integer greater than zero.

Variable Length Elements Delimited

- The *Delimiter* property must contain a non-empty value.

Use Data Pattern

- Each simple element that is a child of the complex type must have a regular expression specified for *Data Pattern*. See "Regular expression syntax" on page 6304.

All Elements Delimited

- The *Delimiter* property must contain a non-empty value.

Fixed Length

- All simple child elements must specify a length, unless their data type is Boolean (or derived from Boolean).
- All compound child elements must specify a length, unless their data type is Boolean (or derived from Boolean).
- The length can be specified using either the *Length* property, or the *Length Value Of member* property.

Fixed Length AL3

- All complex child elements with a non-Boolean compound element and non-Boolean simple child elements must have either a nonzero value in their *Length* property, or a non-empty value for their *Length Value Of* type member property.

Tagged Fixed Length

- All complex child elements with a non-Boolean compound element and non-Boolean simple child elements must have either a nonzero value in their *Length* property or a non-empty value for their *Length Value Of* type member property.
- The *Tag* property for each and every simple child element must contain a non-empty value.

The following rules also apply:

- If you have set the parent *Type Composition* to Choice, and the parent *Data Element Separation* property to Variable Length Elements Delimited, All Elements Delimited, Fixed Length, or Fixed Length AL3:
  - You must not set the *Type Composition* to Message for any child complex types.
  - You must not set the *Data Element Separation* method to Tagged Delimited or Tagged Fixed Length for any child complex types.

  If you do so, the message set does not deploy successfully.
- If you have set the type's *Data Element Separation* property to Fixed Length, Fixed Length AL3, or Tagged Fixed Length, you must set either the *Length* or *Length Value Of* property for all simple elements under this parent, and also for all complex elements with a simple content and compound elements.
- For a Choice in a fixed-length environment (*Data Element Separation* set to Fixed Length, Tagged Fixed Length, or Fixed Length AL3), length references are not valid. Use element lengths.
- Elements specified in a *Length Value Of* property must be simple elements of type INTEGER, they must exist in the same structure as the referring element, and they must appear before the referring element in that structure.
- Complex types with simple content and Compound elements must have an empty *Length Value Of* type member property. Otherwise, *Length Value Of* element would occur after the referring element in the parent structure, which is disallowed by the previous rule.
- Complex types with simple content cannot have a separation type of Use Data Pattern.
- Compound elements cannot have a separation type of Use Data Pattern.
- Regardless of the setting of the type's *Data Element Separation* property, if the type of a simple element is BINARY, you must set either the *Length* or *Length Value Of* property.
- For fixed-length elements, the *Justification* property must be set to something other than Not Applicable, and the *Padding Character* property cannot be an empty value.
- If any element within a message has its *Interpret Element Value* property set to Message Key, the *Message Key* property must be set for all messages within the message set.
- If you have set the *Repeat* property in the type member to Yes, you must set a value for the *Max Occurs* property in the following two situations:
  - If you have defined an element as a member of a complex type that has the property *Data Element Separation* set to Fixed Length.

– If you have defined a fixed-length element as a member of a complex type that has the property *Data Element Separation* set to `Variable Length Elements Delimited`.

When it is invoked by the broker to interpret an input message, the parser assumes that the number of occurrences of the element is equal to the value that you set for *Max Occurs*. When the parser constructs an output message, if there are fewer elements than the value of *Max Occurs*, the missing elements are inserted with default values.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with a message definition file" on page 2863
Create, open, and delete a message definition file.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

"Additional TDS information" on page 6264
More information about the TDS physical format.

Restrictions for nesting complex types:

If you include a group within another group or complex type, the *Data Element Separation* property for the nested group must be compatible with the *Data Element Separation* property of the parent group or complex type.

For example, you cannot set the parent property to `Fixed Length` and the child property to `Tagged Delimited`, because the length of the `Tagged Delimited` structure would not be known, and would therefore conflict with the parent definition. If groups are nested to three or more levels, the *Data Element Separation* property for each nested group must be compatible with all of its parent groups.

The rules for compatibility are listed in the following table.

	Parent				
Child	**Tagged Delimited, Tagged Encoded Length**	**All Elements Delimited, Variable Length Elements Delimited**	**Fixed Length, Fixed Length AL3**	**Tagged Fixed Length**	**Use Data Pattern**
Tagged Delimited, Tagged Encoded Length	Allowed	Allowed	Not allowed	Not allowed	Allowed
All Elements Delimited, Variable Length Elements Delimited	Allowed	Allowed	Not allowed	Not allowed	Allowed
Fixed Length, Fixed Length AL3	Allowed	Allowed	Allowed	Allowed	Allowed
Tagged Fixed Length	Allowed	Allowed	Not allowed[1]	Allowed	Allowed
Use Data Pattern	Allowed	Allowed	Allowed	Allowed	Allowed

Note:

1. `Tagged Fixed Length` cannot exist at the inner level if any outer level has a *`Data Element Separation`* method of `Fixed Length` or `Fixed Length AL3`. This is because an item of `Tagged Fixed Length` can repeat a variable number of times. `Fixed Length` and `Fixed Length AL3` are parsed by moving a set number of bytes: with a variable number of repeats, it is not possible to calculate the number of bytes that need to be parsed.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with a message definition file" on page 2863
Create, open, and delete a message definition file.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

"Additional TDS information" on page 6264
More information about the TDS physical format.

Omission and truncation of elements:

Omitting and truncating elements depends on the setting of the property *Suppress Absent Element Delimiters*.

See "Complex type TDS properties" on page 5505, "Global group TDS properties" on page 5516, or "Local group TDS properties" on page 5526 for a description of this property.

If you have created a message in which some elements are optional, an input message might not contain all defined elements. If the elements are in a complex type that you have defined with the *Data Element Separation* property of the type set to `All Elements Delimited` or `Variable Length Elements Delimited` (in which the elements are separated by a delimiter and have no tag), any elements that are missing from the end of the complex type must be indicated by the application that creates the message in one of two ways. These ways provide techniques to avoid long sequences of delimiters, and to preserve consistent representation of missing elements.

1. If you have set the *Delimiter* property for the complex type to a value that does not match the value that you have set for the *Delimiter* property for any of the complex type's parent types, the elements at the end of the message can be indicated by the occurrence of a *Delimiter* of one of its parents after the last actual element in the complex type data.

   This is known as the truncation method, in which missing elements are treated as not expected, and both data and delimiters are omitted in the bit stream.

   For example, you define a complex element `C` that has four optional elements. You set the *Delimiter* property to the character plus (+). You define complex element `P`, and set the *Delimiter* property of `P` to asterisk (*). You add three elements to `P`, the first is a string, the second is complex element `C`, and the third is a string.

   When a particular instance of the message is received by the broker, all the elements of `P` are present, but only the first two elements of `C` are present. The data in the message appears as follows if the truncation method is used (where `Pn` are the values of the elements of `P` and `Cn` the values of the elements of `C`):

   ```
   P1*C1+C2*P3
   ```

   When the parser encounters the second asterisk delimiter, it determines that the last two elements of complex element `C` are not present, and the next element is the third element of `P`.

   You can use truncation successfully only when both omission and truncation cause the parser to exhibit the same behavior, unless the elements truncated are fixed length.

2. If the *Delimiter* of the complex type matches that of one of its parents, the truncation method cannot be used. This is because the parser cannot determine whether the delimiter after the last element is for the current complex type, or for one of its parents. Therefore a delimiter must be included in the message data for each missing element to ensure that the parser can match the elements with the model.

   This is known as the omission method, in which missing simple elements are represented by an empty sequence of characters between two delimiters.

For example, you define `P` and `C` as in the previous example, but set the *`Delimiter`* property for `P` to plus (+). When the same message is received by the broker (all elements of `P` are present, the first two elements of `C` are present), the data in the message appears as follows:

```
P1+C1+C2++P3
```

Two delimiter characters have been inserted in the message data for the missing elements of complex element `C`. If the truncation method had been used, the parser would have interpreted the data value `P3` as the value of the third element of complex element `C` and not the third element of complex element `P`.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with a message definition file" on page 2863
Create, open, and delete a message definition file.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

"Additional TDS information" on page 6264
More information about the TDS physical format.

Using regular expressions to parse data elements:

Use regular expressions to identify parts of an input message that are associated with subfields.

If your input messages can contain subfields whose presence or absence can be determined only by examining the actual value of the data (for example, an optional field of numeric digits followed by one or more alphabetic characters) you must use the Data Element Separation method Use Data Pattern.

This situation is particularly relevant to messages that conform to the SWIFT industry standard. To use this method, you must provide regular expressions to

identify those portions of an input message that are to be associated with subfields. You must provide a regular expression value for the `Data Pattern` property of each child of the complex type.

When parsing, data is matched in turn with each child of the complex type. The parser does this by using the regular expression for the child to determine the number of characters from the message that apply for that child. This number of characters is the length of the longest string, starting from the current position in the message, that matches the regular expression. If the longest string that matches the regular expression is of length zero, the element is present in the message, and the empty string is used for the value. If no string matches the regular expression, the element is not present. This situation might cause a subsequent validation error if the element is required.

After the number of characters from the input message has been determined, normal data conversion, or further parsing in the case of a complex element, is performed on the text of the input message to assign values to elements. This might lead to data overrun or underrun errors if the length identified by the pattern is not appropriate for the definition of the child.

"Regular expression syntax" on page 6304 explains the full syntax rules and how to apply them, but the following table gives a few simple examples of parsing using data patterns. A more complex example appears after the table.

Input message	Data Pattern	Value matched
`"123456ABC"`	`[0-9]*`	`"123456"`
`"123"`	`[A-Z]*`	`""`
`"123"`	`[A-Z]+`	Not present
`"0x2A2B"`	`\x2A+`	X'2A'
`"ABCD123"`	`[A-Z]{1,3}` first field `[A-Z]{2,4}` second field	`"ABC"` - first field (the longest string matching the pattern) Not present - second field (minimum length of two alphabetic characters is not present)
`"ABCDEFGHIJ1234"`	`[A-Z]{1,3}` first field, repeat `[0-9]+` second field	`"ABC"` - first field [1] `"DEF"` - first field [2] `"GHI"` - first field [3] `"J"` - first field [4] `"1234"` - second field (the repeating field is terminated when the data `"1234"` no longer matches the data pattern specified for the first field.)

The following example shows three-field pattern matching.

```
Message definition:
 Complex type: Data Element Separation=Use Data Pattern
 Field1: xsd:string minOccurs=1, maxOccurs=1, Length=5, Pad=SPACE,
    Data Pattern=".{5}"
 Field2: xsd:int minOccurs=0, maxOccurs=1,
    Data Pattern="[0-9]{0,6}"
 Field3: xsd:string minOccurs=1, maxOccurs=1, minLength=3, maxLength=4,
    Data Pattern="[A-Z][A-Za-z0-9]{2,3}"

Input1:  "ABCDE123F12"
Result1:  Field1="ABCDE", Field2="123", Field3="F12"

Input2:  "ABCDEF12"
Result2:  Field1="ABCDE", Field2=not present, Field3="F12"

Input3:  "ABCDE123456XXXX"
Result3:  Field1="ABCDE", Field2="123456", Field3="XXXX"

Input4:  "ABCDE1234567"
Result4:  Field1="ABCDE", Field2="123456", Field3=not present,
    which causes an exception if validation is enabled. One
    character ("7") remains unassigned to any element, which
    also causes an exception.
```

In the case of a repeating child, instances of the child are parsed for as many times as the pattern is matched. This is applied even if *Max Occurs* is specified for the repeating element and the number of occurrences exceeds the upper bound. Therefore some terminating condition must be determinable from the regular expression pattern for the element. The table above includes an example of a repeating element.

When parsing, the data from the input message that matches the *Data Pattern*, and that is assigned to an element, is *not* further scanned for delimiters of a higher level complex type. This behavior is similar to that of *Data Element Separation* method `Fixed Length`. However, you can code a regular expression that will match data to one of a number of possible delimiters.

When writing, if a length is specified for a child, the value is padded as appropriate to that length. This behavior is similar to that of *Data Element Separation* method `Variable Length Elements Delimited`, but without delimiters.

If the message includes a complex type that has *Composition* set to `Choice`, you can set the *Data Element Separation* method to `Use Data Pattern`. In this case, the *Data Pattern* values of the children are used to resolve the choice. Starting with the first child, the first pattern to provide a match determines which child is present. Therefore the order of children in a choice might be important.

A complex type can contain repeating children with *Max Occurs* unbounded. *Length*, and other associated properties such as *Justification* and *Padding*, can optionally be specified for the children.

See "TDS message model integrity" on page 6295 for rules that you must follow when using the *Data Element Separation* method `Use Data Pattern`, and refer to "Combinations of Composition and Content Validation" on page 5612 for details of valid settings of *Composition* and *Content Validation*.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with a message definition file" on page 2863
Create, open, and delete a message definition file.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

"Additional TDS information" on page 6264
More information about the TDS physical format.

Regular expression syntax:

Regular expression syntax elements and example syntax rules.

A regular expression allows you to specify the conditions that a string must satisfy. For example, you might use a regular expression to specify that a string must contain eight characters and start with an alphabetic character. Use the syntax in the following tables to write regular expressions to specify the sets of strings that are permitted. A regular expression can be made up of one or more branches (choices), each of which can be a string made up of characters, character classes, or parenthesized expressions with modifiers to specify repetition rules.

The regular expression syntax that is supported is a subset of XML Schema regular expressions, with the addition of the \xNN hexadecimal syntax. For the full syntax, see Appendix F in XML Schema Part 2: Datatypes that can be found on the World Wide Web Consortium (W3C) Web site.

The following table lists the supported regular expression syntax elements:

Metacharacter	Meaning
\	escape
.	any single character
*	preceding character 0 or more times
+	preceding character 1 or more times

Metacharacter	Meaning
?	preceding character 0 or 1 time
{...}	occurrences of preceding [1]
[...]	match one of the class contained
[^...]	match one of the class not contained [1]
(...)	group the expressions [1]
|	match either preceding or following
Escape sequence	**Meaning**
\n	new line
\r	carriage return
\t	tab
\e	escape
Class code	**Meaning**
\d	digit [0-9]
\D	non-digit [^0-9] [2]
\s	white space[\t\n\r]
\S	non-whitespace character[^ \t\n\r] [2]
\p{L}	all letters [3]
\p{N}	all numbers, similar to \d [4]
[\p{N}\p{L}]	all numbers and all letters, similar to \w [4]
\P{L}	not letters, equivalent to [^\p{L}]
\P{N}	not numbers, equivalent to [^\p{N}]
\xNN	hexadecimal digits in the range 0 to F (\x00 not supported)
Range	**Meaning**
{n}	exactly n times
{n,}	at least n times
{n,m}	at least n, but no more than m, times
{0,m)	zero to m times

Notes:

1. The ellipsis (...) is used to indicate anything inside the { }, or [ ], or ( ) characters.
2. The caret (^) means "not" when inside the [ ] characters.
3. Consult Appendix F of the document XML Schema Part 2: Datatypes for other characters that can be used in place of L and N.
4. Consult Appendix F of the document XML Schema Part 2: Datatypes for the precise differences.

The following table gives some examples of the syntax rules for regular expression syntax. See “Using regular expressions to parse data elements” on page 6301 for some examples of their use.

Regular expression data pattern	Meaning
a	Match character "a"

Regular expression data pattern	Meaning
`.`	Match any one character
`a+`	Match a string of one or more `"a"`
`a*`	Match a string of zero or more `"a"`
`a?`	Match zero or one `"a"`
`a{3}`	Match a string of exactly three "a", that is `"aaa"`
`a{3,}`	Match a string of three or more `"a"`
`a{2,4}`	Match a string with a minimum of two and a maximum of four occurrences of `"a"`
`[abc]`	Match any one of the characters `"a"`, `"b"`, or `"c"`
`[a-zA-Z]`	Match any one character in the range `"a"` to `"z"`, or in the range `"A"` to `"Z"`. Note that the range of characters matched is based on the Unicodes of the characters specified.
`[^abc]`	Match any character except one of `"a"`, `"b"`, or `"c"`
`(ab)+`	Match one or more repetitions of the string `"ab"`
`(ab)|(cd)`	Match either of the strings `"ab"` or `"cd"`

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with a message definition file" on page 2863
Create, open, and delete a message definition file.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

"Additional TDS information" on page 6264
More information about the TDS physical format.

Using multiple delimiters:

To parse messages in which fields are delimited by one of a set of characters or strings, set *Data Element Separation* to the method Use Data Pattern.

For example, consider a simple message with two numeric fields that can have either of the characters ';' or '/' delimiting them. You can use two approaches:

- Model the delimiter as a data element which is added to the message tree. If the message is rewritten, it looks like the input message.

  Consider this model:

  ```
  Composition = Sequence
  Data Element Separation = Use Data Pattern
     FieldA    Data Pattern = [0-9]*
     Delim     Data Pattern = [;/] optionally with a default value.
     FieldB    Data Pattern = [0-9]*
  ```

  After parsing, the elements FieldA and FieldB each contain any number of the digits 0 - 9, and the element Delim contains either ";" or "/".

- Recognize the delimiter as a delimiter, which is *not* added to the tree. If the message is rewritten, a preferred delimiter (as specified in the model) is used.

  Consider this model:

  ```
  Composition = Choice
  Data Element Separation = Use Data Pattern
    SubType1 Data Pattern = [0-9]*;[0-9]*
    (Composition = Sequence
    Data Element Separation = All Elements Delimited
    Delimiter = ';')
     FieldA
     FieldB
    SubType2 Data Pattern = [0-9]*/[0-9]*
    (Composition = Sequence
    Data Element Separation = All Elements Delimited
    Delimiter = '/')
     FieldA
     FieldB
  ```

  The regular expressions differentiate between the two options that can occur in the message, which are then parsed as a normal delimited structure. After parsing, the elements FieldA and FieldB each contain any number of the digits 0 to 9. The delimiter found in the input message is *not* saved in an element.

  You could refine this approach by using different names for the children, or elements for SubType1 and SubType2, to provide the knowledge of which delimiter is used, or to control which delimiter is included in the output message.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM TDS format: Relationship to the logical model" on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with a message definition file" on page 2863
Create, open, and delete a message definition file.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

"Additional TDS information" on page 6264
More information about the TDS physical format.

Using a variable number of repeats:

You can use the *Data Element Separation* method `Use Data Pattern` to support a variable number of repetitions in an otherwise fixed length environment, where there is no markup to indicate the end of the repetitions.

However, it relies on the ability to recognize the end of the repetitions based on the data content.

In its simplest form, you can do this by specifying a regular expression `Data Pattern` that matches a fixed number of characters that is terminated by reaching the end of the message bit stream.

For example, consider a message with one fixed length field (length 10), followed by another fixed length field (length 20) that repeats indefinitely to the end of the bit stream:

Message *Data Element Separation*=`Use Data Pattern`
    FieldA *Data Pattern*=`.{10}`
    FieldB Repeat, Min Occurs=1, no Max Occurs, *Data Pattern*=`.{20}`

The following example message contains a fixed length field (length 20) that repeats a variable number of times, and is separated from a second field by the string ";". The pattern specifies a string of 20 characters starting with anything except a semicolon:

Message *Data Element Separation*=`All Elements Delimited`, Delimiter=`;`
    SubType1 *Data Element Separation*=`Use Data Pattern`
        FieldA Repeat, Min Occurs=1, no Max Occurs, *Data Pattern*=`[^;].{19}`
        FieldB

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

“The message model” on page 1160
The message model consists of the following components.

“MRM TDS format: Relationship to the logical model” on page 1243
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

Related tasks:

“Constructing message models” on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

“Working with a message definition file” on page 2863
Create, open, and delete a message definition file.

“Working with MRM message model objects” on page 2870
Add, configure, and delete objects.

Related reference:

“Message model reference information” on page 5366
Reference information in this section can help you develop and configure message models.

“Message model object properties” on page 5416
Access property information by property kind, or by object.

“Additional MRM domain information” on page 6251
More information about the MRM domain.

“Additional TDS information” on page 6264
More information about the TDS physical format.

Performance considerations when using regular expressions:

Take care when specifying regular expressions: some forms of regular expression can involve a large amount of work to find the best match, which might degrade performance.

Other expressions might produce a result that you did not expect.

For example, to match text up to and including a delimiter character ';' do *not* use the pattern ".*;", which matches up to the *last* ';' character in the message, including all prior ';' characters in the matched text. Instead, use the pattern "[^;]*;".

Similarly, avoid using the pattern ".*", which always forces a search to the end of the message to try and find the best match, and therefore might result in poor performance. However, you must use the pattern ".*" if you intend to match all remaining data in a message.

For best performance, avoid expressions with redundant nested repeats, such as "([0-9]+)*". Keep the expressions simple, with precise matching criteria. Simple expressions avoid the need to perform multiple searches for the best match.

Related concepts:

“Message modeling” on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

“The message model” on page 1160
The message model consists of the following components.

“MRM TDS format: Relationship to the logical model” on page 1243
TDS separation types and logical model properties have some restrictions, such as

group composition and group content validation.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with a message definition file" on page 2863
Create, open, and delete a message definition file.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

"Additional TDS information" on page 6264
More information about the TDS physical format.

DateTime formats:

When you create an element or attribute with a simple type of dateTime, you must specify a format string in the object's *Format String* property for each physical format layer (CWF, TDS, XML).

You can use the symbols defined in the following information to control the format in which the dateTime appears in the message data.

You can only use dateTime for Gregorian calendar dates.

DateTime information can appear in a message as:

- String data. This includes XML, and all TDS and CWF physical types except those mentioned in this section. This is described further in "DateTime as string data" on page 6311.
- Binary data. This is for the TDS or CWF Binary physical type. See "DateTime as BINARY data" on page 6318 for more information.
- An offset from an epoch in seconds or milliseconds. This is used if you have set the TDS or CWF *Physical Type* property to `Time Seconds` or `Time Milliseconds` respectively. See "DateTime as encoded values" on page 6319 for details of this option.

The defaults that are set for each message set property that relates to dateTime, for each physical representation (CWF, TDS, XML), are defined in "Message set defaults" on page 6321.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with a message definition file" on page 2863
Create, open, and delete a message definition file.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

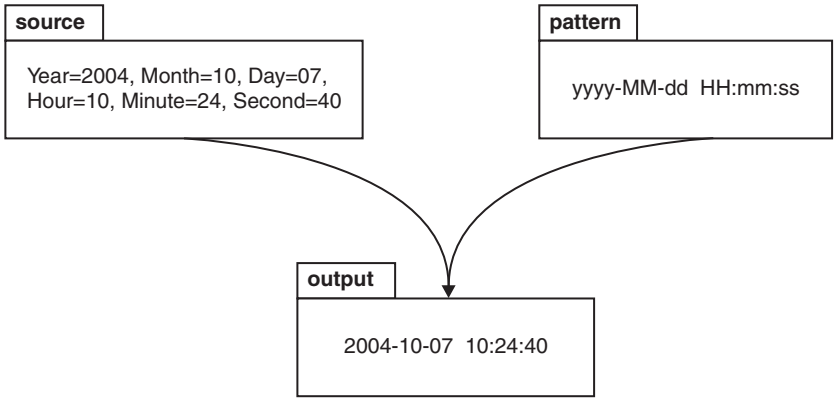
"Message model object properties" on page 5416
Access property information by property kind, or by object.

DateTime as string data:

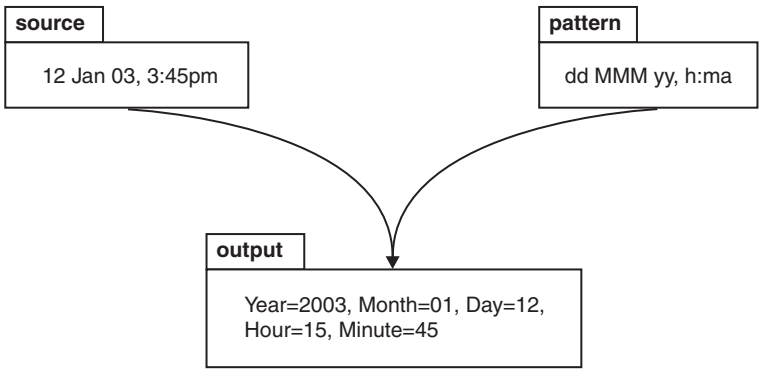
You can use a string of pattern letters to specify the dateTime format.

When you convert a date or time into a string, a format pattern must be applied that directs the conversion. Apply the format pattern to convert a date or time into a string, or to parse a string into a date or time.

During the conversion (for example, of a dateTime into a string), a pattern or a set of tokens is replaced with the equivalent source. The following diagram shows how a pattern is used to format a dateTime source to produce a character string output.



When a string is parsed (for example, when converting the string to a dateTime), the pattern or set of tokens is used to determine which part of the target dateTime is represented by which part of the string. The following diagram shows how this is done.



Syntax

The expression pattern is defined by:

Where:

symbol
is a character in the set **adDeEFGhHIkKmMsSTUwWyYzZ**.

string is a sequence of characters enclosed in single quotation marks. If a single quotation mark is required within the string, use two single quotation marks ('').

Characters for formatting a dateTime as a string

The following table lists the characters that you can use in a pattern for formatting or parsing strings in relation to a dateTime. The table is followed by some notes that explain more about some of the examples in the table.

Symbol	Meaning	Presentation	Examples
a	am or pm marker	Text	Input am, AM, pm, PM. Output AM or PM
d	day in month (1-31)	Number	1, 20
dd	day in month (01-31)	Number	01, 31
D	day in year (1-366)	Number	3, 80, 100
DD	day in year (01-366)	Number	03, 80, 366
DDD	day in year (001-366)	Number	003
e	day in week (1-7)[1]	Number	2
EEE	day in week[1]	Text	Tue
EEEE	day in week[1]	Text	Tuesday

Symbol	Meaning	Presentation	Examples
F	day of week in month (1-5)[2]	Number	2
G	Era	Text	BC or AD
h	hour in am or pm (1-12)	Number	6
hh	hour in am or pm (01-12)	Number	06
H	hour of day in 24 hour form (0-23)[3]	Number	7
HH	hour of day in 24 hour form (00-23)[3]	Number	07
I	ISO8601 Date/Time (up to yyyy-MM-dd'T'HH:mm:ss.SSSZZZ)[4]	Text	2006-10-07T12:06:56.568+01:00
IU	ISO8601 Date/Time (similar to I, but ZZZ with output "Z" if the time zone is +00:00)[4]	Text	2006-10-07T12:06:56.568+01:00, 2003-12 -15T15:42:12.000Z
k	hour of day in 24 hour form (1-24)[3]	Number	8
kk	hour of day in 24 hour form (01-24)[3]	Number	08
K	hour in am or pm (0-11)	Number	9
KK	hour in am or pm (00-11)	Number	09
m	minute	Number	4
mm	minute	Number	04
M	numeric month	Number	5, 12
MM	numeric month	Number	05, 12
MMM	named month	Text	Jan, Feb
MMMM	named month	Text	January, February
s	seconds[10]	Number	5
ss	seconds[10]	Number	05
S	decisecond[5]	Number	7
SS	centisecond[5]	Number	70
SSS	millisecond[5]	Number	700
SSSS	0.0001 second[5]	Number	7000
SSSSS	0.00001 second[5]	Number	70000
SSSSSS	0.000001 second[5]	Number	700000
T	ISO8601 Time (up to HH:mm:ss.SSSZZZ)[4]	Text	12:06:56.568+01:00
TU	ISO8601 Time (similar to T, but a time zone of +00:00 is replaced with 'Z')[4]	Text	12:06:56.568+01:00, 15:42:12.000Z
w	week in year[6]	Number	7, 53
ww	week in year[6]	Number	07, 53
W	week in month[7]	Number	2
yy	year[8]	Number	06

Symbol	Meaning	Presentation	Examples
yyyy	year[8]	Number	2006
YY	year: use with week in year only[6]	Number	06
YYYY	year: use with week in year only[6]	Number	2006
zzz	time zone (abbreviated name)[9]	Text	EST
zzzz	time zone (full name)	Text	Eastern Standard Time
Z	time zone (+/-n)	Text	+3
ZZ	time zone (+/-nn)	Text	+03
ZZZ	time zone (+/-nn:nn)	Text	+03:00
ZZZU	time zone (as ZZZ, "+00:00" is replaced by "Z")	Text	+03:00, Z
ZZZZ	time zone (GMT+/-nn:nn)	Text	GMT+03:00
ZZZZZ	time zone (as ZZZ, but no colon) (+/-nnnn)	Text	+0300
'	escape for text		'User text'
''	(two single quotation marks) single quotation mark within escaped text		'o''clock'

The presentation of the dateTime object depends on the symbols that you specify.

- **Text**: If you specify four or more of the symbols, the full form is presented. If you specify less than four, the short or abbreviated form, if it exists, is presented. For example, `EEEE` produces `Monday`, `EEE` produces `Mon`.
- **Number**: The number of characters for a numeric dateTime component must be within the bounds of the corresponding formatting symbols. Repeat the symbol to specify the minimum number of digits required. The maximum number of digits permitted is the upper bound for a particular symbol. For example, day in month has an upper bound of 31; therefore, a format string of `d` allows the values 2 or 21 to be parsed but does not allow the values 32 or 210 to be parsed. On output, numbers are padded with zeros to the specified length. A year is a special case; see note 8. Fractional seconds are also a special case; see note 5.
- **Lenient dateTime checking**: The parser converts out-of-band dateTime values to the appropriate in-band value. For example, the date 2005-05-32 is converted to 2005-06-01. Output of dateTimes always adheres to the symbol count. For example, a formatting string of `yyyy-MM-dd` (where '-' is the field separator) allows one or more characters to be parsed against `MM` and `dd`. This conversion allows dates such as 2006-01-123 and 2006-011-12, which are not valid, to be input. The value of 2006-01-123 is written as the date 2006-05-03, and the value of 2006-011-12 is written as the date 2006-11-12. The number of occurrences of the time zone formatting symbol Z applies only to the output dateTime format. White space is skipped over.
- **Physical Type**: If you specify the `Physical Type` property of the dateTime object to be `Packed Decimal`, the only pattern formatting symbols that are valid are those that represent numbers; that is, those that have Number in the Presentation column of the table. No other characters are allowed in the format

pattern. For example, yyyyMMdd is valid, but yyyyMMMdd is not valid because MM is a numeric representation of the month, and MMM is a textual representation of the month.

- Any characters in the pattern that are not in the ranges of ['a'..'z'] and ['A'..'Z'] are treated as quoted text. For example, characters like colon (:), comma (,), period (.), the number sign (hash or pound, #), the at sign (@), and space are displayed in the resulting time text even if they are not enclosed within single quotation marks.
- You can create formatting strings that produce unpredictable results; therefore, you must use these symbols with care. For example, if you specify `dMyyyy`, it is impossible to distinguish between day, month, and year. `dMyyyy` tells the broker that a minimum of one character represents the day, a minimum of one character represents the month, and four characters represent the year. Therefore `3112006` might be interpreted as 3/11/2006 or as 31/1/2006.

Notes: The following notes apply to the preceding table.

1. The day in week field is the numeric offset into a week and varies according to the value of the physical message set property `First Day of Week`. For example, the third day in the week is Wednesday if the physical message set property `First Day of Week` is set to Monday.
2. 12th July 2006 is the second Wednesday in July and can be expressed as `2006 July Wednesday 2` using the format string `yyyy MMMM EEEE F`. Note that this format does not represent the Wednesday in week 2 of July 2006, which is 5th July 2006; the format string for this is `yyyy MMMM EEEE W`.
3. 24-hour fields might result in an ambiguous time, if specified with a conflicting am/pm field.
4. See "ISO8601, I and T DateTime tokens" on page 6316.
5. Fractional seconds are represented by uppercase S. The length must implicitly match the number of format symbols on input. The format string `ss SSS` or `ss.SSS`, for example, represents seconds and milliseconds. However, the format string `ss.sss` represents a repeated field (of seconds); the value after the period (.) is taken as a seconds field, not as fractional seconds. The output is truncated to the specified length.
6. The start of a year typically falls in the middle of a week. If the number of days in that week is less than the value specified by the physical message set property `Days in First Week of Year`, the week is considered to be the last week of the previous year; in this case, week 1 starts some days into the new year. Otherwise, the week is considered to be the first week of the new year; in this case, week 1 starts some days before the new year. For example, Monday of week 1 in 2004 (2004 01 Monday, where Days in First Week of Year = 4 and First Day of Week = Monday) using format string `YYYY ww EEEE` is in fact 29th December 2003. If you use `Y`, the day of week (`E`) and week in year (`w`) are adjusted if necessary to indicate that the date falls in the previous year.

   If you use the lowercase `y` symbol, the adjustment is not done and unpredictable results might occur for dates around year end. For example, if the string 2002 01 Monday is formatted:

   - Monday of week 1 in 2002 using format string `YYYY ww EEEE` is correctly interpreted as `31st December 2001`
   - Monday of week 1 in 2002 using format string `yyyy ww EEEE` is incorrectly interpreted as `30th December 2002`

   Use `Y` together with `w` only; if you do not specify `w`, use `y`.

7. The first and last week in a month might include days from neighboring months. For example, Monday 31st July 2006 can be expressed as *Monday in week one of August 2006*, which is `2006 08 1 Monday` using format string `yyyy MM W EEEE`.
8. Year is handled as a special case:
   - On output, if the count of `y` is 2, the year is truncated to 2 digits. For example, if `yyyy` produces `2006`, `yy` produces `06`.
   - On input, for 2-digit years, the physical message set property of `Start of century for 2 digit years` is used to determine the century. For example, if `Start of century for 2 digit years` is set to 53, year 97 is 1997, year 52 is 2052, and year 53 is 1953.
9. Using the `zzz` option can have ambiguous results. For example, `BST` can be interpreted as `Bangladesh Standard Time` or `British Summer Time`. For compatibility reasons, WebSphere Message Broker uses the former interpretation.

   To avoid these problems, use the `zzzz` option with a well-defined name; for example, Europe/London, Asia/Dhaka, or America/Los_Angeles.
10. Seconds `s` & `ss`, must be in the range 0-59. If you need to construct a TIMESTAMP representing the time during a leap second, where the value being created or cast uses the value 60 for seconds, you must handle this case within your ESQL code. The CURRENT_ datetime functions (for example, CURRENT_TIME) within the product never produce a time where the seconds value falls outside of the 0-59 range.

ISO8601, I and T DateTime tokens

If your dateTime values comply with the ISO8601:2000 'Representation of dates and times' standard, consider using the formatting symbols I and T, which match the following subset of the ISO8601 standard.

- The restricted profile as proposed by the W3C at http://www.w3.org/TR/NOTE-datetime
- Truncated representations of calendar dates, as specified in section 5.2.1.3 of ISO8601:2000
  - Basic format (subsections c, e, and f)
  - Extended format (subsections a, b, and d)

Use the formatting symbols I and T only on their own:

- The I formatting symbol matches any dateTime string that conforms to the supported subset.
- The T formatting symbol matches any dateTime string that conforms to the supported subset that consists of a time portion only.

The following table shows how the output form relates to the logical data type.

Logical model data type	ESQL data type	Output form
xsd:dateTime	TIMESTAMP or GMTTIMESTAMP	yyyy-MM-dd'T'HH:mm:ss.SSSZZZ
xsd:date	DATE	yyyy-MM-dd
xsd:gYear	INTERVAL	yyyy
xsd:gYearMonth	INTERVAL	yyyy-MM
xsd:gMonth	INTERVAL	--MM
xsd:gmonthDay	INTERVAL	--MM-dd

Logical model data type	ESQL data type	Output form
xsd:gDay	INTERVAL	---dd
xsd:time	TIME / GMTTIME	'T'HH:mm:ss.SSSZZZ

Note:

- On input, both I and T accept both '+00:00' and 'Z' to indicate a zero time difference from Coordinated Universal Time (UTC), but on output they always generate '+00:00'. If you want 'Z' to always be generated on output, use the IU or TU formatting symbols instead.
- ZZZ always writes '+00:00' to indicate a zero time difference from Coordinated Universal Time (UTC). If you want 'Z' to always be generated on output, use ZZZU instead.

Using the input UTC format on output

An element or attribute of logical type xsd:dateTime or xsd:time that contains a dateTime as a string can specify Coordinated Universal Time (UTC) by using either the Z symbol or time zone +00:00. On input, the MRM parser remembers the UTC format of such elements and attributes. On output, you can specify whether Z or +00:00 is displayed by using the `Default DateTime Format` property of the element or attribute. Alternatively, you can preserve the input UTC format by selecting the message set property `Use input UTC format on output`. If this property is selected, the UTC format is preserved in the output message and overrides the format that is implied by the dateTime format property.

Examples

The following table shows a few examples of dateTime formats.

Format pattern	Result
"yyyy.MM.dd 'at' HH:mm:ss ZZZ"	2006.07.10 at 15:08:56 -05:00
"EEE, MMM d, "yy"	Wed, July 10, '06
"h:mm a"	8:08 PM
"hh o"clock a, ZZZZ"	09 o'clock AM, GMT+09:00
"K:mm a, ZZZ"	9:34 AM, -05:00
"yyyy.MMMMM.dd hh:mm aaa"	1996.July.10 12:08 PM

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with a message definition file" on page 2863
Create, open, and delete a message definition file.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

DateTime as BINARY data:

The count of pattern letters determines the number of bytes used to represent a value. The symbol used in the pattern of letters can be used only in groups of 1, 2, or 4; for example, y, yy, or yyyy.

The following table shows the dateTime symbols for binary data:

Symbol	Meaning	Example
y	year	1996
M	month in year	7
d	day in month	10
H	hour in day (0-23)	13
m	minute in hour	30
s	second in minute	55
S	millisecond	978
X	Ignore on input Pad with zeros on output	

The following example shows the C language structure `tm` with an integer of four bytes:

```
struct tm
{ int tm_sec;      /* seconds after the minute   - [0,59]*/
{ int tm_min;      /* minutes after the hour     - [0,59]*/
{ int tm_hour;     /* hours since midnight       - [0,23]*/
{ int tm_mday;     /* day of the month           - [1,31]*/
{ int tm_mon;      /* months since January       - [0,11]*/
{ int tm_year;     /* years since 1900           */
{ int tm_wday;     /* days since Sunday          - [0,6]*/
{ int tm_yday;     /* days since January 1       - [0,365]*/
{ int tm_isdst;    /* daylight saving time flag */
};
```

You can format this structure by specifying the string "`ssssmmmmHHHHddddMMMM+1yyyy+1900XXXXXXXXXXXX`". The number of pattern letters determines the number of bytes. There are 36 A-Z characters specified in this pattern, which match the 36 byte structure `tm`. A field followed by a plus sign (+) has the succeeding numeric characters added to it. Therefore `MMMM+1` adds one to the month, `yyyy+1900` adds 1900 to the year. `X` expects one byte of input, but ignores its value. On output, it writes the byte as 0.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined

structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with a message definition file" on page 2863
Create, open, and delete a message definition file.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

DateTime as encoded values:

You can represent a dateTime element with the `TimeSeconds` and `TimeMilliSeconds` physical types.

`TimeSeconds`
: A 4 byte integer that represents the number of seconds since the epoch.

`TimeMilliSeconds`
: An 8 byte integer that represents the number of milliseconds since the epoch.

These types provide a way for C time_t and Java dateTime representations to be parsed.

The epoch (time 0) is specified by a format string. To change the epoch you must update the physical properties of your dateTime element:

- In the Physical representation section, you must set the `Physical Type` to either Time Seconds or Time Milliseconds.
- In the `Format` field, set the value to the format of "yyyy-MM-dd'T'HH:mm ZZZ". For example, 2000-01-01T12:59 +00:00.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM Custom wire format: Relationship to the logical model" on page 1219
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with a message definition file" on page 2863
Create, open, and delete a message definition file.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

DateTime defaults by logical type:

The default value that is assigned to the dateTime Format property is dependent on the logical type of the property.

The following table lists the default for each of the logical dateTime types:

Logical Type	Default Format
date	yyyy-MM-dd
dateTime	yyyy-MM-dd'T'HH:mm:ss
gDay	- - -dd
gMonth	- -MM
gMonthDay	- -MM-dd
gYear	yyyy
gYearMonth	yyyy-MM
time	HH:mm:ssZZZ

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with a message definition file" on page 2863
Create, open, and delete a message definition file.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

DateTime component defaults:

Default values are assumed if any part of a dateTime element is not present on input.

For example, the formatting string `yyyy-MM'T'HH:mm` does not contain any information about day in month (d), seconds (s), or milliseconds (S). The following table shows the defaults for all dateTime components.

Component	Default value
Year	1970
Month	First month of year
Day	First day of month
Hour	First hour of day
Minute	Minute 0 of hour
Second	Second 0 of minute
Millisecond	Millisecond 0 of second

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with a message definition file" on page 2863
Create, open, and delete a message definition file.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

Message set defaults:

The default dateTime formatting property settings for the different MRM physical formats.

Message set property	CWF default	TDS default	XML default
Default DateTime Format	See Note 1.	See Note 1.	See Note 1.

Message set property	CWF default	TDS default	XML default
Default Time Zone ID	Use Broker Locale (see Note 2)	Use Broker Locale (see Note 2)	Use Broker Locale (see Note 2)
Century Window	53	53 (80 for SWIFT)	53
Days in First Week of Year	4	Use Broker Locale (see Note 2)	Use Broker Locale (see Note 2)
First Day of Week	Monday	Use Broker Locale (see Note 2)	Use Broker Locale (see Note 2)

Note:

1. You can either set the default dateTime format to be derived from its logical type (the default), or specify the dateTime format that is to be used. This is set at the message set level for each physical format that has been added.
2. The key phrase `Use Broker Locale` causes the broker to get the information from the underlying platform.

You can update all these default values. The CWF defaults are set for all values of the *Physical Type* property. If you change the CWF *Physical Type* to `Binary`, `Packed Decimal`, `TimeSeconds`, or `TimeMilliseconds`, you must update the *Default DateTime Format* property manually to ensure consistent results.

For more information about these message set properties, see "Custom Wire Format message set properties" on page 5375, "TDS Format message set properties" on page 5381, or "XML Wire Format message set properties" on page 5400.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with a message definition file" on page 2863
Create, open, and delete a message definition file.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

Additional MIME domain information

Standard header fields, and parser use and restrictions.

- "MIME standard header fields" on page 6323

- "MIME parser use and restrictions" on page 6327

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"Physical formats in the MRM domain" on page 1211
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

"MIME parser and domain" on page 1117
Use the MIME domain if your messages use the MIME standard for multipart messages.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with a message set" on page 2840
Complete a variety of tasks that are involved in working with a message set.

"Working with a message definition file" on page 2863
Create, open, and delete a message definition file.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message set properties" on page 5371
Message sets have properties that you can set to define their characteristics and the way in which they are processed.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

MIME standard header fields:

Check this quick reference to the common MIME headers.

This information does not provide a definitive specification of MIME. In some cases, the MIME parser allows documents that are not strictly valid according to the standard. For example, it does not insist on the presence of a MIME-Version header. All the standard MIME header fields are simply written to the logical tree as they appear in the MIME document. The MIME parser takes special note only of the Content-Type header field.

All MIME headers can include comments enclosed by parentheses, as shown in the example for the MIME-Version header.

MIME header fields

MIME-Version

Example:

```
MIME-version: 1.0 (generated by my-application 1.2)
```

For a MIME document to conform with RFC 2045, this field is required in the top-level header with a value of 1.0. MIME-Version should not be specified on individual parts.

Content-Type

Content-Type is not required for a document to conform with RFC 2045, but a top-level Content-Type is required by the MIME parser. Content-Type defaults to `text/plain`. Content-Type defines the type of data in each part as a type/subtype. The MIME parser accepts most values for Content-Type and stores them in the logical tree. The only exceptions are:

- The MIME parser rejects any Content-Type value with `type = message`.
- The MIME parser assumes that a Content-Type value with `type = multipart` introduces a multipart MIME document, and rejects such a value if it does not contain a valid boundary parameter. The value of the boundary parameter defines the separator between message parts in a multipart message. In a nested multipart message, a unique boundary value is required for each nesting level.

Syntax:

```
Content-Type: type/subtype;parameter
```

where type and subtype define the Content-Type, and all optional parameters are delimited by semicolons.

Example 1:

```
Content-Type: multipart/related;type=text/xml
```

In example 1, the Content-Type is defined as `multipart/related`, and also has an optional parameter definition (`type=text/xml`). Although this structure is syntactically correct, because a valid boundary parameter does not exist, this message is rejected.

Example 2:

```
Content-Type: multipart/related;boundary=Boundary;type=text/xml
```

Example 2 shows a valid Content-Type definition, both in terms of syntax and semantics. The boundary value optionally can be enclosed in quotation marks. When it appears in the MIME body, the value is preceded by the sequence '--', and you must ensure that the resulting value (in this example, --Boundary) cannot appear in the message body. If the message data is encoded as quoted-printable, you must include a boundary that includes a sequence such as “=_”, which cannot appear in a quoted-printable body.

The following table shows some common Content-Type values. Other values are allowed, and stored in the logical tree.

Content-Type	Description
text/plain	Typically used for a typical mail or news message. text/richtext is also common.
text/xml	Typically used with SwA (SOAP with Attachments).
application/octet-stream	Used where the message is an unknown type and contains any kind of data as bytes.
application/xml	Used for application-specific xml data.
x-type	Used for non-standard content type. It must start with the characters x-.

Content-Type	Description
image/jpeg	Used for images. image/jpeg and image/gif are common image formats that are used
multipart/related	Used for multiple related parts in a message. Specifically used with SwA (SOAP with Attachments)
multipart/signed	Used for multiple related parts in a message including signature. Specifically used with S/MIME
multipart/mixed	Used for multiple independent parts in a message

Content-Transfer-Encoding

Optional. Many Content-Types are represented as 8-bit character or binary data, and can include XML, which typically uses UTF-8 or UTF-16 encoding. This type of data cannot be transmitted over some transport protocols, and might be encoded to 7-bit.

The Content-Transfer-Encoding header field is used to indicate the type of transformation that has been used for encoding this type of data into a 7-bit format.

The following values only are allowed by the WS-I Basic Profile:

- 7bit - the default
- 8bit
- binary
- base64
- quoted-printable

The values 7bit, 8bit, and binary all effectively mean that no encoding took place. A MIME conformant mail gateway might use this value to control how it handles the message. For example, encoding it as 7bit before passing routing it over SMTP.

The values base64 and quoted-printable mean that the content has been encoded. The value quoted-printable means that only non-7-bit characters in the original are encoded, and is intended to yield a document which is still human-readable. This setting is most likely to be used in conjunction with a Content-Type of `text/plain`.

Content-ID

Optional. This field enables parts to be labeled, and referenced from other parts of the message. These parts are typically referenced from part 0 (the first) of the message.

Content-Description

Optional. This field enables parts to be described.

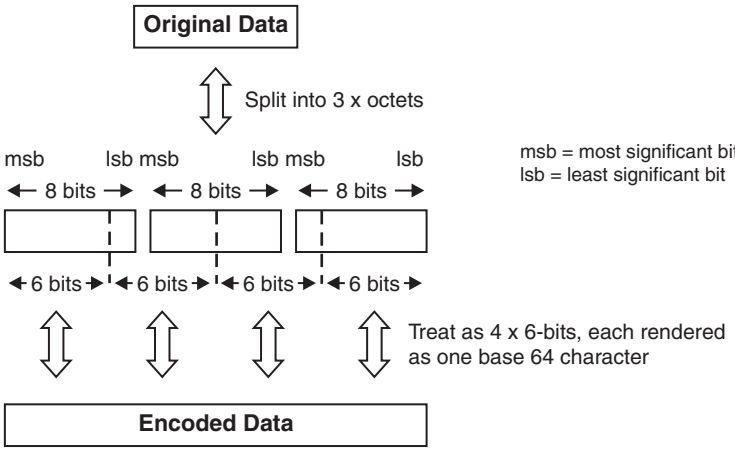
MIME encodings

The following section provides a basic guide to the base64 and quoted-printable encoding; refer to RFC 1521 (linked at the end of this topic) for a definitive specification of MIME encodings.

base64

The original data is broken into groups of 3 octets. Each group is then treated as 4 concatenated 6-bit groups, each of which is translated into a single digit in the base64 alphabet. The base64 alphabet is A-Z, a-z, 0-9,

and / (with A=0 and /=63).



If fewer than 24 bits are available at the end of the data, the encoded data is padded using the "=" character. The maximum line length in the encoded data is 76 characters and line breaks (and all other characters not in the alphabet above) are ignored when decoding.

Examples:

Input	Output
Some data encoded in base64.	U29tZSBkYXRhIGVuY29kZWQgaW4gYmFzZTY0Lg==
life of brian	bGlmZSBvZiBicmlhbg==\012
what	d2hhdA==

quoted-printable

This encoding is appropriate only if most of the data comprises printable characters. Specifically, characters in the ranges 33-60 and 62-126 are typically represented by the corresponding ASCII characters. Control characters and 8-bit data must be represented by the sequence = followed by a pair of hexadecimal digits.

The standard ASCII space <SP> and horizontal tab <HT> represent themselves, unless they appear at the end of an encoded line (without a soft line break), in which case the equivalent hexadecimal format must be used (=09 and =20 respectively).

Line breaks in the data are represented by the RFC 822 line break sequence <CR><LF> and should be encoded as "=0D=0A" if binary data is being encoded.

For base64, the maximum line length in the encoded data is 76 characters. An '=' sign at the end of an encoded line (a 'soft' line break) is used to tell the decoder that the line is to be continued.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

"Additional MIME domain information" on page 6322
Standard header fields, and parser use and restrictions.

Related information:

RFC 1521: MIME Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies

RFC 822: Standard for the format of ARPA Internet text messages

MIME parser use and restrictions:

The MIME domain does not support the full MIME standard, but supports specific known uses of MIME. Read this general introduction to the MIME parser, and information about some of the restrictions in its use.

MIME stands for Multipurpose Internet Mail Extensions. A multipart MIME message comprises a number of message parts, each qualified by MIME headers. The MIME domain and parser enable you to parse and write multipart MIME messages.

MIME is used to send email messages. When the email includes attachments, a multipart MIME message is used. Multipart MIME is becoming more widely used as a convenient physical format for sending other kinds of message that have attachments or that consist of multiple separate parts.

Examples are:

- **RosettaNet.** Each part is typically a separate XML document but there might also be non-XML attachments. The MIME parser enables the parsing of MIME messages of the style used by RosettaNet, including nested multipart messages. However, it does not offer specific support for the wider RosettaNet architecture or PIPs (Partner Interface Processes).
- **SOAP with Attachments (SwA).** The first part is a normal SOAP XML message and the other parts contain XML or non-XML attachments.
- **TLOG.** This is a specialized use of SwA in which the attachments are groups of point-of-sale Transaction Log records in either one of two XML forms or a tagged/delimited string form. Different POS devices generate different TLOG record formats such as ACE. In addition, the record can either be processed before it is uploaded or it can be sent unchanged.

Restrictions

The MIME parser is driven by bit streams and has no external metadata; it relies exclusively on bitstream metadata when parsing, and on tree metadata when writing. The parser does not validate MIME messages against a message model and it ignores the tooling Validate property. The parts of a MIME message are handled as BLOBs. You can parse specific MIME parts by using a different parser. If you are using an MRM parser, messages can be validated in the usual way. The MIME parser does not support on-demand parsing and ignores the Parse Timing property.

You can specify the new MIME domain either at run time in an MQRFH2 header (WebSphere MQ only) or statically in their message flow in the tooling (on the input nodes MQGet, HTTPRequest, ResetContentDescriptor, or XSLTransform). The MIME parser is then invoked to own the last child of root (for example, the message body). The MIME domain can be specified with the ESQL CREATE PARSE clause and ASBITSTREAM function to parse and write bit streams. The MIME parser handles documents received both over the HTTP transport (where the Content-Type appears as an HTTP header) and over other transports (where the Content-Type header is part of the message body). In both cases, set the Content-Type value using the ContentType property in the MIME domain. Setting the Content-Type value directly in the MIME tree or HTTP trees can lead to the value being ignored or used inconsistently.

Typically, the MIME parser handles the majority of uses of MIME in application-to-application messaging, including multipart MIME with a single part and non-multipart MIME documents. However, you should use the SOAP domain for SOAP with Attachments (SwA).

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Additional MRM domain information" on page 6251
More information about the MRM domain.

"Additional MIME domain information" on page 6322
Standard header fields, and parser use and restrictions.

"Additional TDS information" on page 6264
More information about the TDS physical format.

Related information:

RFC 1521: MIME Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies

Additional IDOC domain information

More information about the deprecated IDOC domain.

This section provides additional information in relation to the IDOC domain. This information is categorized into:

- "Building the message model for the IDOC parser" on page 6330
- "Field names of the IDOC parser structures" on page 6333

Note: The IDOC domain is deprecated and is not recommended for developing new message flows. Instead use the MRM domain with a TDS physical format when you want to process SAP ALE IDocs that are sent to the broker by SAP R3 clients across the WebSphere MQ link for R3.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"IDOC parser and domain" on page 1126
The IDOC domain can be used to process messages that are sent to the broker by SAP R3 clients across the WebSphere MQ link for R3. Such messages are known as SAP ALE IDocs.

"MRM parser and domain" on page 1111
You can use the MRM domain to parse and write a wide range of message formats.

"DataObject parser and domain" on page 1114
Use the DataObject domain to parse and write messages for WebSphere Adapters and CORBA applications.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with a message set" on page 2840
Complete a variety of tasks that are involved in working with a message set.

"Working with a message definition file" on page 2863
Create, open, and delete a message definition file.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message set properties" on page 5371
Message sets have properties that you can set to define their characteristics and the way in which they are processed.

"Message model object properties" on page 5416
Access property information by property kind, or by object.
"Additional MRM domain information" on page 6251
More information about the MRM domain.

Building the message model for the IDOC parser:

The ALE IDoc messages that are sent to, and received from, SAP applications by using the WebSphere MQ Link for R3, can be processed by the IDOC parser, which requires a message model to interpret the data correctly. This topic describes how to build the message model.

Before you begin

The IDOC domain is deprecated. To develop new message flows, use the MRM domain with a TDS physical format when you want to process SAP ALE IDocs that are sent to the broker by SAP R3 clients across the WebSphere MQ link for R3.

Obtaining the IDoc:
About this task

Create an import file of the required IDoc data for the WebSphere Message Broker Toolkit.

Procedure

1. Log on to an SAP system.
2. Run the supplied transaction `we60`, which extracts the IDoc data as a C header file.
   a. In **Basic Type**, select the IDoc type of interest; for example, `MATMAS02`.
   b. Leave the **Control**, **Data**, and **Status** check boxes cleared.
   c. Select the **Record types** version. A version 4 IDoc is type 3.
   d. Press **F7** to display a C representation of the IDoc.
   e. Select **System** > **List** > **Save** > **Local file**.
   f. Click **unconverted**.
   g. When prompted, enter a file name and directory for the output from the transaction. The C representation of the IDoc is saved to this C header file.

What to do next

Tip: The exported C header can be imported into the WebSphere Message Broker Toolkit without any further manual processing.

Modeling the IDoc:
About this task

Create your message model.

Procedure

1. Switch to the Broker Application Development perspective.
2. Use the New Message Set wizard to create a message set for your IDoc. Select binary data as the data to use. This option creates a message set with a Custom Wire Format (CWF) physical format, and presets the `Default message domain` property to MRM.

3. Use the Message Set editor to change the Default message domain property to IDOC.
4. Use the New Message Definition File wizard to import a prebuilt model of the ALE IDoc message structure. To start the wizard, click **File** > **New** > **Message Definition File From**. When the wizard opens, select IBM supplied message, then SAP ALE IDoc. This SAP ALE IDoc prebuilt model includes definitions of the DC and DD segments. The resulting message definition file is called ale_idoc.mxsd. For information about using the New Message Definition File wizard, see "Importing from IBM supplied messages" on page 2942.
5. Use the New Message Definition File wizard, or the **mqsicreatemsgdefs** command, to import the C representation of the IDoc into the new message set. To start the wizard, click **File** > **New** > **Message Definition File From**.

   Specify the following settings:

   - Select C Header file.
   - Set Select the pre-processing option to apply to SAP ALE IDoc. If this option is not specified, the import of the C header fails. If this option is specified, the message prefix preference is ignored.
   - Create messages for the segments that appear in the IDoc.
   - Use the String Encoding option to import character arrays as fixed-length strings.
   - Use the Padding Char for String option to make space (" ") the padding character that is used.

   For information about using the New Message Definition File From C Header File wizard, see "Importing from C" on page 2934.

Using the IDoc message model:
About this task

You can now use your message model to help you to construct a message flow that processes instances of your IDoc message. You can use ESQL or Java to access the fields of the IDoc. You cannot use graphical maps to access the fields of the IDoc because the IDOC domain is not supported by the mapping editor.

When you set the properties of the MQInput node that is to receive your IDoc from the WebSphere MQ Link for R3, the Message Domain property must be IDOC, the Message Set property must be the name of your message set, and the Message Format property must be the name of your Custom Wire Format. You do not need to set a Message Type property on the MQInput node because it is not needed by the IDOC parser.

When your message flow is complete, add the message set and the message flow to a broker archive (BAR) file and deploy the BAR file to a broker execution group.

When an IDoc is received by the MQInput node, the IDOC parser processes the SAP-defined elements in the DC, then, for each DD, processes the SAP-defined elements, and calls the MRM parser to process the user-defined segment data, as described by your exported IDoc, using the CWF physical format. The MRM parser knows the Message Type property to use for the user-defined segment, because it is obtained from the SAP-defined DD field *segnam* by the IDOC parser.

Tip: SupportPac IA0F contains a more detailed description of the steps involved in building the IDoc message model. You can ignore utilities IDocHeaderTweak and IDocMsgSetTweak because that processing has been incorporated into the New Message Definition File From C Header File wizard.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"ESQL overview" on page 2371
Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data within a message flow.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"Message model objects" on page 1174
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is a WebSphere Message Broker extension to XML Schema.

"IDOC parser and domain" on page 1126
The IDOC domain can be used to process messages that are sent to the broker by SAP R3 clients across the WebSphere MQ link for R3. Such messages are known as SAP ALE IDocs.

Related tasks:

"Importing from C" on page 2934
Create a message definition file from a C header file for use in the MRM and IDOC domains.

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Manipulating messages in the MRM domain" on page 2581
How to use messages that have been modeled in the MRM domain, and that are parsed by the MRM parser.

"Manipulating messages in the IDOC domain" on page 2610
Use ESQL from a Compute node to copy the incoming IDoc to the outgoing IDoc, and manipulate the message.

"Building the MRM TDS model for an IDoc" on page 6279
The MRM domain Tagged/Delimited String (TDS) physical format is suitable for parsing and writing SAP ALE IDocs and SAP File IDocs. ALE IDoc messages are exported from SAP across the WebSphere MQ Link for R3. File IDocs are exported from SAP to the file system.

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

"**mqsicreatemsgdefs** command" on page 3702
Use the **mqsicreatemsgdefs** command to create message definition files.

"Field names of the IDOC parser structures" on page 6333
The field names of the Control Structure (DC) and the Data Structure (DD) that are used by the IDOC parser.

Field names of the IDOC parser structures:

The field names of the Control Structure (DC) and the Data Structure (DD) that are used by the IDOC parser.

The field names are documented in the form that they are used in a SET statement of ESQL; for example:

```
SET OutputRoot.Properties = InputRoot.Properties;
SET OutputRoot.MQMD = InputRoot.MQMD;
```

Control structure (DC) fields:
All the fields must be specified and set.

The syntax is:

```
<rootname>.<ParserName>.<foldername>.<fieldname>=
```

For example:

```
SET "OutputRoot"."IDOC"."DC"."docnum" = '0000000000000001';
SET "OutputRoot"."IDOC"."DC"."idoctyp" = 'MATMAS01'
```

The field names, which must be specified in order, are:

1. tabnam
2. mandt
3. docnum
4. docrel
5. status
6. direct
7. outmod
8. exprss
9. test
10. idoctyp
11. cimtyp
12. mestyp
13. mescod
14. mesfct
15. std
16. stdvrs
17. stdmes
18. sndpor
19. sndprt
20. sndpfc
21. sndprn
22. sndsad
23. sndlad
24. rcvpor
25. rcvprt
26. rcvpfc
27. rcvprn
28. rcvsad

29. rcvlad
30. credat
31. cretim
32. refint
33. refgrp
34. refmes
35. arckey
36. serial

Data structure (DD) fields:
To access each DD segment, use the array suffix DD[1], DD[2], and so on.

The syntax is:

```
<rootname>.<ParserName>.DD[1].<fieldname>=
```

For example:

```
SET OutputRoot.IDOC.DD[I].segnam  = 'E2MAKTM001';
SET OutputRoot.IDOC.DD[I].mandt2   = '111';
```

The following list illustrates how the suffix 2 is used to give unique field names to the `mandt` and `docnum` fields.

The field names, which must be supplied in order, are:

1. segnam
2. mandt2
3. docnum2
4. segnum
5. psgnum
6. hlevel

The last 1000 bytes of data in the DD segment are the bytes that are parsed by the MRM domain. The DD `segnam` describes the model that the MRM uses.

Segment fields:
The syntax is:

```
<rootname>.<ParserName>.DD[1].sdatatag.MRM.<fieldname>=
```

For example:

```
SET OutputRoot.IDOC.DD[I].sdatatag.MRM.msgfn = '006'
SET OutputRoot.IDOC.DD[I].sdatatag.MRM.spras_iso = 'EN'
```

The `sdatatag` field indicates to the parser that it is the element that contains the data to be manipulated. The `MRM` field indicates that the MRM handles the transformation.

- msgfn
- spras
- maktx
- msgfn
- spras_iso
- fill954

The `fill954` field is the filler for the segment because an incoming IDoc to SAP must have 1000 byte segments.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"IDOC parser and domain" on page 1126
The IDOC domain can be used to process messages that are sent to the broker by SAP R3 clients across the WebSphere MQ link for R3. Such messages are known as SAP ALE IDocs.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Developing Java" on page 2628
When you use the JavaCompute node, you customize it to determine the exact processing that it provides.

"Manipulating messages in the MRM domain" on page 2581
How to use messages that have been modeled in the MRM domain, and that are parsed by the MRM parser.

"Manipulating messages in the IDOC domain" on page 2610
Use ESQL from a Compute node to copy the incoming IDoc to the outgoing IDoc, and manipulate the message.

"Building the message model for the IDOC parser" on page 6330
The ALE IDoc messages that are sent to, and received from, SAP applications by using the WebSphere MQ Link for R3, can be processed by the IDOC parser, which requires a message model to interpret the data correctly. This topic describes how to build the message model.

Related reference:

"Data types of fields and elements" on page 4237

"SET statement" on page 5159
The SET statement assigns a value to a variable.

JSON parser use and restrictions

WebSphere Message Broker provides support for the JSON domain.

JSON (JavaScript Object Notation) is a simple data-interchange format based on a subset of the JavaScript programming language. A JSON message consists of objects and arrays, where an object is a set of name-value pairs, and an array is a list of values. A JSON value can be a simple value (string, number, Boolean, or null), an array, or an object.

For more information about the JSON parser and domain, see "JSON parser and domain" on page 1128.

Message modeling and validation

WebSphere Message Broker does not provide support for JSON message modeling. Because of this, the Message Set Editor, Message Mapping Editor, Mapping node, and the XSLTransform node do not support the JSON domain.

Related concepts:

"Parsers" on page 1072
A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from the internal message tree representation.

"JSON parser and domain" on page 1128
JSON (JavaScript Object Notation) is a simple data-interchange format based on a subset of the JavaScript programming language.

"JSON message details" on page 1135
A JSON message consists of name-value pairs (objects), and ordered collections of values (arrays). Objects, arrays, or both structures can be nested.

"Manipulating messages in the JSON domain" on page 2617
You can manipulate messages that belong to the JSON domain, which are parsed by the JSON parser.

Message model task list errors that have a quick fix

You can apply a quick fix to some message modeling task list warnings or errors to correct them.

Unresolved references

The following table provides a list of those errors that have references that cannot be resolved:

Error type	Description	Quick Fix
Attribute reference error	The attribute reference cannot be resolved	Allows you to add the missing include or import statement
Attribute group reference error	The attribute group reference cannot be resolved	Allows you to add the missing include or import statement
Attribute type reference	The attribute type reference cannot be resolved	Allows you to add the missing include or import statement
Base type error	The type has an unresolved base type	Allows you to add the missing include or import statement
Element reference error	The element reference cannot be resolved	Allows you to add the missing include or import statement
Element type reference error	The element type reference cannot be resolved	Allows you to add the missing include or import statement
Group reference error	The group reference cannot be resolved	Allows you to add the missing include or import statement
Schema directive error	The schema directive cannot be resolved	Allows you to add the missing include or import statement
Sub group error	The element declaration references a head element which cannot be resolved.	Allows you to add the missing include or import statement

Other errors

The following table provides a list of additional warnings or errors that can be cleared using a quick fix:

Error type	Description	Quick Fix
Message key deprecated warning	TDS property "Message Key" has been superseded by logical property "Message Alias".	Will update your message definition to use "Message Alias" instead. (You should use this if you only have Version 6.0 brokers in your domain.)
Message key enumeration deprecated warning	TDS property "Interpret Element Value = Message Key" has been superseded by logical property "Interpret Value As = Message Identity".	Will update your element definition to use logical property "Interpret Value As = Message Identity" instead. You should use this if you only have Version 6.0 brokers in your domain.)
Repeat count deprecated warning #1	CWF property "Repeat Count" has been superseded by "Max Occurs". Both "Repeat Count" and "Max Occurs" have been set, but do not have the same value.	You will have a choice of two quick fixes: • Will update your definition to unset the "Repeat Count" property. • Will update your definition to set "Max Occurs" to the value of the "Repeat Count" property, and to unset the "Repeat Count" property.
Repeat count deprecated warning #2	CWF property "Repeat Count" has been superseded by "Max Occurs". Both "Repeat Count" and "Max Occurs" have been set and have the same value.	Will update your definition to unset the "Repeat Count" property.
Redefine error	An XML Schema Redefine construct has been found but is not supported.	Will update your message definition file to use an XML Schema Include construct instead. Any redefinitions will be lost.
Value does not match Length facet error	The length of a default value, fixed value or enumeration value does not match the effective Length facet for the simple type.	You will have a choice of two quick fixes: • Will update your simple type definition so that the Length facet is converted to a Max length facet. • Will update all the simple type definitions in your message definition file so that all Length facets are converted to Max Length facets, then save the file to remove all the associated task list errors.

Error type	Description	Quick Fix
Facet not applicable for simple type error	A facet has been found on a simple type, but the facet is either not permitted on that simple type or is a duplicate.	Will update your simple type definition so that all invalid facets and all duplicate facets are removed.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"Message model integrity" on page 1210
When you create your message model, it is important that it is internally consistent.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

"Applying a Quick Fix to a task list error" on page 2862
During the creation, migration and manipulation of message models, warnings or errors might occur; these are listed in the Problems view of the Broker Application Development perspective. Some of these warnings or errors can be cleared by applying a Quick Fix.

Generated model representations

Information about generating documents, WSDLs, Broker SCA definitions, and XML schemas.

- "Document generation" on page 6339
- "Broker SCA definition generation" on page 6340
- "WSDL generation" on page 6340
- "XML Schema generation" on page 6343

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with a message definition file" on page 2863
Create, open, and delete a message definition file.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

Document generation:

The document generator produces a set of HTML pages and any necessary files (for example, images) that are required to display the pages correctly.

Output Files:
There is one page for each message definition file in the message set, and one additional index page linking these pages together.

The index page (*index.html*), is intended to be the "entry point" into the documentation.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"Message set projects" on page 1161
A *message set project* is a specialized container in which you create and maintain all the resources associated with one message set.

"Message sets overview" on page 1162
A message set is a container for grouping messages and associated message resources (elements, types, groups).

"Message definition files" on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

"Generate model representations" on page 1270
After you have created and populated a message set, you can generate a message model in several different representations for use by a broker, a parser, or your applications.

"Generating message model documentation" on page 1277
When you have created one or more message models, it can be useful to generate documentation for business analysis and for developers who are involved with the messages.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Generating documentation from message sets and message flows" on page 2962
You can generate documentation from your message sets, message flows, message definition files, message maps, Java files, ESQL files, and deployable WSDL files.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Documentation properties for all message set objects" on page 5413
Use the documentation property of an object to add information that describes the function of the object.

"WSDL generation"
Files and other objects are created by the WSDL Generator.

"XML Schema generation" on page 6343
The behavior of XML Schema generation. For example, use the schema generated from a message definition file to validate XML instance documents written by WebSphere Message Broker.

Broker SCA definition generation:

The Broker SCA Definition wizard creates a Broker SCA definition that includes objects that define the binding, interface, and message format information to permit interoperation between WebSphere Message Broker and WebSphere Process Server.

Generated files

The following files are generated:

- A `.outsca` or `.insca` Broker SCA Definition is generated and stored under the Broker SCA Definitions folder in the chosen message set project. The `.outsca` or `.insca` Broker SCA Definition contains a SCA Import or Export, as well as all XSD files and WSDL files referenced directly or indirectly by the SCA Import or Export.
- A WSDL file containing one portType with one or more operations.
- XSD files that correspond to the messages used in the operations defined in the portType.
- If the user has chosen a Web service binding, a WSDL file containing a service and binding that imports the WSDL containing the portType.
- If the user has chosen an MQ binding, the SCA Import or Export contains the binding information.

Related reference:

"Generate Broker SCA Definition wizard" on page 6372
The Generate Broker SCA Definition wizard creates a Broker SCA definition from a message set.

WSDL generation:

Files and other objects are created by the WSDL Generator.

Generated Files

The default file and definition element names are shown in the following table. *Message Set* is the supplied message set name and <*Definition Name*> is the supplied Definition Name solicited by the wizard.

File	File Name	File Extension	Value of name attribute on WSDL <definitions> element
Service File (single-file format)	*Message_Set*	wsdl	<Definition Name>
Service File (multi-file format)	*Message_Set*Service	wsdl	<Definition Name>Service
Binding File	*Message_Set*Binding	wsdl	<Definition Name>Binding
Interface File	*Message_Set*	wsdl	<Definition Name>

If 'Deployable WSDL' is generated, no additional XML schema (xsd) files are generated, and the WSDL refers directly to the broker message definition (mxsd) files; otherwise, separate XML schema (xsd) files are generated, unless you selected 'inline schema'.

Report File

The WSDL generator appends the result of the generation operation to a report file, listing all errors which occurred. The file name is:

Message_Set.wsdlgen.report.txt

WSDL Content

The following tables show the element or attribute values to be set in the generated WSDL. The elements are described top-down as they appear in a conventionally ordered WSDL document. The <schema> section of the WSDL definition is not shown, because this section corresponds directly to the broker message definitions.

Element names are from the WSDL 1.1 namespace except where prefixed by soap: for the WSDL SOAP namespace. Operation elements occur in both the binding and portType sections, so operation is qualified as necessary; for example, portType / operation.

The values shown in the following table apply to the WSDL definition as a whole.

Element	Attribute	Value
definitions	xmlns	Assign namespace prefixes.
definitions	targetNamespace	The WSDL Namespace solicited by the wizard, defaulting to http://tempuri.org/<Message Set>.
message	name	<operation>_<role> where <operation> is the operation name and <role> is in, out, or fault
part	name	The name of the broker message. If Style is set to rpc, the body parts are defined using the type attribute. If not, the body parts are defined using the element attribute.
portType	name	*Message_Set*PortType
binding	name	• "*Message_Set*SOAP_HTTP_ Binding" • "*Message_Set*SOAP_JMS_ Binding"

Element	Attribute	Value
soap:binding	style	From the value of `Style` set in the wizard.

The following values apply to each individual WSDL operation:

Element	Attribute	Value
operation	name	The name of the operation specified in the wizard.
soap:operation	style	From the value of `Style` set in the wizard.
input, output, fault	name	<operation>_<role>, where <operation> is the operation name, and <role> is Input, Output, or Fault.
soap:body	namespace	• If `Style` has been set to rpc, it is the namespace of the corresponding broker message. • If `Style` has been set to document the attribute is not generated.
soap:header, soap:fault, soap:body	use	This element is set to *literal*.

Message Set

The message set provides the basis for many important broker features, including mapping support and ESQL code completion at development time, and validation at run time.

Therefore, the WSDL that you use in the broker at development time (for example, when configuring SOAP nodes) is integrated with the message set, and references the broker message definitions (mxsd) rather than ordinary Schema (xsd) files. This is referred to as *deployable WSDL* and is displayed under the category `Deployable WSDL` in the WebSphere Message Broker Toolkit.

Deployable WSDL is generated when you specify your Message Set Folder (the immediate child of your Message Set Project) as the `destination directory` for your WSDL.

Otherwise, regular WSDL is generated, along with separate XML schema (xsd) files if these were requested. Regular WSDL cannot be used to configure SOAP nodes, but is suitable for consumption by external applications such as .NET.

Assuming that you are generating deployable WSDL for use in a message flow, the flow typically must be able to parse and validate the runtime SOAP messages described by that WSDL. The WSDL generator, therefore, adds additional definitions to your message set:

- For rpc-style WSDL, additional definitions for the WSDL operations themselves are added to your message set
- For the version of the SOAP Envelope used by the WSDL an mxsd file is added – this will be soapenv11.mxsd or soapenv12.mxsd.
- For use by ESQL Content assist and the Mapping editor primarily, a definition of the SOAP_Domain_Msg tree.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with a message definition file" on page 2863
Create, open, and delete a message definition file.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

"Generating a WSDL definition from a message set" on page 2968
To ensure the highest interoperability of your Web services, use the document style of WSDL whenever possible. If rpc style WSDL is necessary, use literal encoding.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

XML Schema generation:

The behavior of XML Schema generation. For example, use the schema generated from a message definition file to validate XML instance documents written by WebSphere Message Broker.

Lax generation

Lax generation affects how complex types that have *Content Validation* set to Open or OpenDefined or have *Composition* set to UnorderedSet are rendered in the generated schema. Note that such a validating schema will permit a wider range of messages than MRM parser validation.

Content Validation is set to Open or OpenDefined

Here a complex type (global or anonymous) has its content replaced by a single wildcard element that repeats an unbounded number of times. The following generation pattern is used for complex types with *Content Validation* set to Open:

```
<element name="xmlNameOfMessage">
 <complexType>
  <sequence>
   <any processContent="lax"
   minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
 </complexType>
</element>
```

Where *Content Validation* is set to OpenDefined, the following pattern is used. (The namespaces listed are all those defined in the containing message set.)

```
<element name="xmlNameOfMessage">
 <complexType>
  <sequence>
   <any processContent="lax"
     minOccurs="0" maxOccurs="unbounded"
     namespace="http://www.ns1 http://www.ns2" />
  </sequence>
 </complexType>
</element>
```

Composition is set to UnorderedSet

Where *Composition* is set to UnorderedSet, to mimic the unordered aspect, a choice is inserted with appropriate cardinality. This is shown as follows:

```
<element name="xmlNameOfMessage">
 <complexType>
  <sequence maxOccurs="unbounded"
   minOccurs= "(minOccurs of original sequence) *
          (items in original sequence)">
   <choice>
         .. sequence contents ..
   </choice>
  </sequence>
 </complexType>
</element>
```

Strict generation

Strict generation affects how complex types that have *Content Validation* set to Open or OpenDefined or have *Composition* set to UnorderedSet are rendered in the generated schema. Note that such a validating schema will permit a narrower range of messages than MRM parser validation.

Strict is the default generation option and generates a schema that matches the schema held in the message definition file, without the model extensions.

Content Validation set to Open/OpenDefined

A complex type (global or anonymous) will lose the ability to contain self-defining elements and becomes closed.

Composition set to UnorderedSet

A complex type (global or anonymous) will lose the ability to be unordered and becomes a sequence.

Rendering of xsd:elements

If an XML physical format is specified when generating the schema, the wire format customization is applied to the logical model. These properties control how an element in the model is rendered when it appears in a message for an XML wire format. See “XML rendering options” on page 6262 for the different render options available. The following is a generated schema example showing what is generated for the different render options available for local elements; note these examples do not modify the Namespace of any ID Attribute Name or Value Attribute Name properties and assume that all elements specified in the complexType1 are of schema built-in type string.

```
<xsd:complexType name="complexType1">
    <xsd:sequence>
     <!-- Local element Render = 'XMLElement' -->
        <xsd:element name="localElement1" type="xsd:string"/>
     <!-- Local element Render = 'XMLElementAttrID'
                                    ID Attribute Name = 'id' -->
        <xsd:element name="localElement2">
```

```
            <xsd:complexType>
                <xsd:simpleContent>
                    <xsd:extension base="xsd:string">
                        <xsd:attribute name="id" type="xsd:string"/>
                    </xsd:extension>
                </xsd:simpleContent>
            </xsd:complexType>
        </xsd:element>
     <!-- Local element Render = 'XMLElementAttrVal'
                                  Val Attribute Name = 'val' -->
        <xsd:element name="localElement3">
            <xsd:complexType>
                <xsd:simpleContent>
                    <xsd:extension base="xsd:string">
                        <xsd:attribute name="val" type="xsd:string"/>
                    </xsd:extension>
                </xsd:simpleContent>
            </xsd:complexType>
        </xsd:element>
     <!-- Local element Render = 'XMLElementAttrIDVal'
          ID Attribute Name = 'id' Val Attribute Name = 'val' -->
        <xsd:element name="localElement4">
            <xsd:complexType>
                <xsd:simpleContent>
                    <xsd:extension base="xsd:string">
                        <xsd:attribute name="val" type="xsd:string"/>
                        <xsd:attribute name="id" type="xsd:string"/>
                    </xsd:extension>
                </xsd:simpleContent>
            </xsd:complexType>
        </xsd:element>
    </xsd:sequence>
     <!-- Local element Render = 'XMLAttribute' -->
    <xsd:attribute name="localElement5" type="xsd:string"/>
</xsd:complexType>
```

Rendering of xsd:attributes

The rendering of `xsd:Attributes` is not supported. The user can only change the name of the attribute.

Embedded simple types and Compound Elements

These are deprecated objects that are only encountered if the message set was created using WebSphere MQ Integrator Broker Version 2.1.

They are modeled in the message definition file as elements with both *minOccurs* and *maxOccurs* set to `0` and have one of the predefined ComIbmMrm_xxx types. During the schema generation, the type of such elements is changed to the base type of the respective ComIbmMrm_xxx type.

If there are global simple types that inherit from one of these ComIbmMrm_xxx types, these are changed to inherit from the base type of the corresponding ComIbmMrm_xxx type.

Generated schema files will not have any occurrence of these ComIbmMrm_xxx types.

For example the global element with type defined as follows:

```
<element name="globalElement1" type="ns1:ComIbmMrm_BaseValueBinary"/>
```

will result in the generated schema file and a global element with the corresponding xsd base type defined as follows:

```
<element name="globalElement1" type="hexBinary"/>
```

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with a message definition file" on page 2863
Create, open, and delete a message definition file.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

Import formats

Supported features of formats that have been imported from C, COBOL, SCA Export, SCA Import, WSDL, or XML Schema.

This section provides information about the supported features of formats that have been imported from an external source. Details are provided for:

- "Importing from C (MRM): supported features" on page 6347
- "Importing from COBOL: supported features" on page 6350
- "Importing from SCA Export or SCA Import: generated objects" on page 6355
- "Importing from WSDL: generated objects and restrictions" on page 6355
- "Importing from XML Schema: unsupported features" on page 6359

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with a message definition file" on page 2863
Create, open, and delete a message definition file.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

Importing from C (MRM): supported features:

The C importer uses default values when mapping C data types to message model elements.

The following table shows how the C definitions influence the XML Schema settings in the message model. Some xsd types have '-' after the type. This character indicates that it is an anonymous simple type based on this type. For strings, the purpose of the anonymous type is to add a maximum length restriction; for numeric types, the purpose of the anonymous type is to add either a minimum or a maximum value restriction.

C data type	XML Schema data	Notes
Char	xsd:string-	maxlength=1
Char[10]	xsd:string-	maxlength=10
Char[10][3]	xsd:string-	maxlength=3 maxOccurs=10
Char[10][3][6]	xsd:string-	maxlength=6 maxOccurs=30
Unsigned Char	xsd:unsignedByte	
Unsigned Char[2]	xsd:unsignedByte	maxOccurs=2
Signed Char	xsd:byte	
Signed Char[2]	xsd:byte	maxOccurs=2
Int	xsd:int	
Int[2]	xsd:int	maxOccurs=2
Int[2][3]	xsd:int	maxOccurs=6
Unsigned Int	xsd:unsignedInt	
Short	xsd:short	
Unsigned Short	xsd:unsignedShort	
Long	xsd:int	
Long Long Int	xsd:long	
Float	xsd:float	
Double	xsd:double	
Long Double (see note 1)	xsd:double	
<any pointer type>	xsd:hexBinary-	maxlength=(see note 2)
<any enum>		(see note 3)

The following table shows how C definitions influence the physical MRM CWF characteristics of the elements that are generated in the message model.

C data type	CWF Physical type	CWF Length characteristics	Other CWF characteristics
Char	Fixed Length	Length = 1 Length Units = Bytes	
Char[10]	Fixed Length	Length = 10 Length Units = Bytes	Left justify
Char[10][3]	Fixed Length	Length = 3 (and Max Occurs = 10) Length Units = bytes	Left justify
Char[10][3][6]	Fixed Length	Length =6 (and Max Occurs = 30) Length Units = bytes	Left justify
Unsigned Char	Integer	Length = 1	Signed = no
Unsigned Char[2]	Integer	Length = 1 (and Max Occurs = 2)	Signed = no
Signed Char	Integer	Length = 1	Signed = yes
Signed Char[2]	Integer	Length = 1 (and Max Occurs = 2)	Signed = yes
Int	Integer	Length = 4	Signed = yes
Int[2]	Integer	Length = 4 (and Max Occurs = 2)	Signed = yes
Int[2][3]	Integer	Length = 4 (and Max Occurs = 6)	Signed = yes
Unsigned Int	Integer	Length = 4	Signed = no
Short	Integer	Length = 2	Signed = yes
Unsigned Short	Integer	Length = 2	Signed = no
Long	Integer	Length = 4 (see note 4)	Signed = yes
Long Long Int	Integer	Length = 8	Signed = yes
Float	Float	Length = 4	
Double	Float	Length = 8	
Long Double (see note 1)	Float	Length = 8	
<any pointer type>		(see note 2)	
<any enum>	Integer	(see note 3)	

Notes:

1. Do not set the value of C importer option *size of long double* to 128 bit. This option does not import successfully; use the default 64 bit.
2. The length is affected by the *Address Size* C importer option:
   - For 32 bit, CWF length = 4 bytes.
   - For 64 bit, CWF length = 8 bytes.

3. The type and length of an enum is affected by the *Size of enum* C importer option:
   - For *1*: Logical type = xsd:byte, CWF physical type = Integer, CWF length = 1 byte.
   - For *2*: Logical type = xsd:short, CWF physical type = Integer, CWF length = 2 bytes.
   - For *4*: Logical type = xsd:int, CWF physical type = Integer, CWF length = 4 bytes.
   - For *Compact*: The smallest representation is chosen that the enumeration fits into.
4. The length of a *long* is affected by the *Address Size* C importer option:
   - For 32 bit: CWF length = 4 bytes.
   - For 64 bit: CWF length = 8 bytes.
5. Element names that clash with Java language keywords are modified by prefixing them with a single underscore character.
6. The _Packed keyword is not supported. Only ANSI C declarations are supported.
7. The C long long data type is not supported.
8. C++ object oriented extensions are not supported. Only ANSI C declarations are supported.
9. Pointers will be imported as xsd:integer with CWF length set to 4.
10. Recursive C structures are not supported. If a nested structure contains a structure with a name that is the same as the parent structure, the import succeeds, but the logical definitions are not correct. To avoid this problem, ensure that the name of the nested structure is not the same as that of the outer or parent structure.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with a message definition file" on page 2863
Create, open, and delete a message definition file.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

Importing from COBOL: supported features:

The COBOL importer uses a set of default values and behaviors when mapping COBOL data types to message model elements.

The following table shows how COBOL definitions influence the XML Schema settings in the message model.

COBOL Clause	XML Schema data type	Notes
PIC A	xsd:string	
PIC G	xsd:string	Set the compile-time locale name to ja_JP in **Window** > **Preferences** > **Importer** > **COBOL** to process this.
PIC N	xsd:string	Set the compile-time locale name to ja_JP in **Window** > **Preferences** > **Importer** > **COBOL** to process this.
PIC X	xsd:string	
PIC 9(n) n = 1-4	xsd:short	DISPLAY, COMP, or COMP-3
PIC 9(n) n = 5-9	xsd:int	DISPLAY, COMP, or COMP-3
PIC 9(n) n = 10-18	xsd:long	DISPLAY, COMP, or COMP-3
PIC 9(n) n = 19-31	xsd:integer	DISPLAY, COMP, or COMP-3
PIC 9(n)V9(m)	xsd:decimal	DISPLAY, COMP, or COMP-3 any virtual decimal point value
COMP-1	xsd:float	
COMP-2	xsd:double	
Any edited string	xsd:string	
Any edited number	xsd:string	For example, a COBOL PICTURE clause that contains any of the following characters: 'Z' '+' '-' '.' ',' 'B' '0' or a currency symbol. If you want your broker logical type to be a numeric one, make sure that the COBOL PICTURE clause does not contain any of these characters.
VALUE	All	Non-88 Level VALUE clauses can be imported as schema default values (option on import wizard).

The following table shows how COBOL definitions influence the physical MRM CWF characteristics of the elements that are generated in the message model.

COBOL Clause	CWF Physical Type	CWF Length Characteristics	Other CWF characteristics
PIC X(n) PIC A(n)	Fixed Length String	Length = n Length Units = Bytes	Justification = Left Justify Padding Character = SPACE

COBOL Clause	CWF Physical Type	CWF Length Characteristics	Other CWF characteristics
PIC G(n) PIC N(n)	Fixed Length String	Length = n Length Units = Characters	Justification = Left Justify Padding Character = SPACE
PIC 9(n) DISPLAY n=1-31	External Decimal	Length = n Length Units = Bytes	Justification = Right Justify Padding Character = '0' Signed = Unticked Sign Orientation = Trailing
PIC 9(n) COMP, COMP-4, COMP-5 or BINARY	Integer	Length = 2, 4 or 8 based on n Length Units = Bytes	Signed = Unticked Sign Orientation = Blank
PIC 9(n) COMP-3 n=1-18	Packed Decimal	Length = CEILING((n+1)/2) Length Units = Bytes	Signed = Unticked Sign Orientation = Blank
PIC S9(n) DISPLAY n=1-31	External Decimal	Length = n Length Units = Bytes	Signed = Ticked Sign Orientation = Trailing *See Note 1
PIC S9(n) COMP or COMP-3 n=1-18	Integer or Packed Decimal	Length = See COMP and COMP-3 definitions above Length Units = Bytes	Signed = Ticked Sign Orientation = Blank
PIC 9(m)V9(n) DISPLAY n=1-31	External Decimal	Length = n+m Length Units = Bytes	Signed = Unticked Sign Orientation = Trailing Virtual Decimal Point = n
PIC 9(m)V9(n) COMP or COMP-3	Integer or Packed Decimal	Length = CEILING((n+m+1)/2) for COMP-3 Length = 2, 4 or 8 for COMP Length Units = Bytes	Signed = Unticked Sign Orientation = Blank Virtual Decimal Point = n
COMP-1	Float	Length = 4 Length Units = Bytes	Signed = Ticked Sign Orientation = Blank
COMP-2	Float	Length = 8 Length Units = Bytes	Signed = Ticked Sign Orientation = Blank
SYNC	Float, Integer or Packed Decimal		Leading Skip Count as appropriate Trailing Skip Count as appropriate Byte alignment as appropriate *See note 2

COBOL Clause	CWF Physical Type	CWF Length Characteristics	Other CWF characteristics

Notes:

1. **Sign Orientation** can take one of the following values, based on the SEPARATE, LEADING, or TRAILING keywords in the COBOL definition:
   - Leading
   - Leading Separate
   - Trailing
   - Trailing Separate
2. The SYNC keyword causes the field to be aligned on a 1, 2, 4, or 8-byte boundary. This might cause 'slack bytes' to be added either before or after a field. **Leading Skip Count** is the number of such bytes that are added before a field; **Trailing Skip Count** is the number of such bytes that are added after a field.

   Leading Skip Count and **Trailing Skip Count** are calculated by the importer for each of the imported elements by the importer, irrespective of the SYNC clause. They have non-zero values when the SYNC clause is present.

   Where there is a repeating element, **Leading Skip Count** and **Trailing Skip Count** are used for the first occurrence of the repeating element; for subsequent occurrences, only the **Trailing Skip Count** is used.

   Refer to COBOL reference material for details of fields that require byte alignment.
3. All files that you import must be syntactically correct. Results are unpredictable if the file being imported is not synctactically correct.
4. COBOL data types that have keywords POINTER, COMP-X, INDEX, or PROCEDURE-POINTER, are not supported.
5. COBOL clauses that contain the keyword NATIVE cause an error, and are not imported.
6. COBOL level 66 and level 77 data items are not imported.
7. Hexadecimal binary values cannot be attributed to non-numeric literals. They cannot reside in the LINKAGE SECTIONs that are imported by the COBOL importer. They can reside elsewhere in the COBOL file. Alternatively, you can convert the hexadecimal value to a character string for PIC X, or to a decimal number for PIC 9.
8. If element names clash with Java language keywords, the element names are modified by prefixing the element name with a single underscore character.
9. Object-oriented extensions to COBOL 85 are not supported. For example, OBJECT-REFERENCE is not supported.
10. COBOL OCCURS DEPENDING ON clause. The **Byte Alignment**, **Leading Skip Count**, and **Trailing Skip Count** CWF properties of elements within such a structure are not set up properly. You must correct these using the message editor.
11. When the imported COBOL source file contains QUOTE or QUOTES in the value clause of a picture string, the default behavior is to enter the data with double quotation marks, unless you set the COBOL QUOTE compile option to SINGLE on the Import Options page of the COBOL importer wizard.

Signed external decimal numbers

The MRM Custom Wire Format (CWF) and TDS components of WebSphere Message Broker support the External Decimal (also known as Zoned Decimal) data format for numeric data. Numeric data in this format is stored internally as decimal character data. For example, in a system that uses the EBCDIC code, the number 1234 stored in a 4-byte external decimal field is stored as the character string '1234', and its actual internal hexadecimal representation is 'F1F2F3F4'.

With signed external decimal numbers, the sign can be incorporated into the actual data by modifying the first half of the first or last byte (depending on whether you are using a sign-leading or sign-trailing representation). Typically, '0xC' is used to represent a positive number, '0xD' is used to represent a negative number and '0xF' is used to represent an unsigned number.

Note: In general, any of '0xA', '0xC', '0xE' or '0xF' can be used to indicate a positive value, and '0xB' or '0xD' can be used to indicate a negative value. The actual preferred representation is dependent upon the actual hardware architecture.

On ASCII machines, there are a number of mechanisms for the internal representation of external decimal data. One representation ('Sign ASCII') that is employed by IBM's pSeries machines, uses the normal ASCII codes ('0' [hex 30] to '9' [hex 39]) for the first or last digit of both unsigned and positive numbers, and the characters 'p' [hex 70] to 'y' [hex 79] for negative numbers.

An alternative method (Sign EBCDIC Custom) is used on some other ASCII based machines. This method uses the same characters as an EBCDIC based machine, even though the actual internal hexadecimal representations of them are different. If you use this technique, the character string for both EBCDIC and ASCII platforms is identical. You could potentially receive a message from an EBCDIC platform (created from a COBOL copybook that contains such entries as PIC XXX and PIC S999), and convert the whole message to ASCII, or the other way around. The character string that represents the external decimal field in the message (after the ASCII to EBCDIC, or EBCDIC to ASCII, conversion) maps to the code point that represents the correct sign for the decimal.

This method includes the limitation that curly brace characters are variant (they have different code points in different EBCDIC code pages). This mechanism works only for those EBCDIC code pages where the curly brace characters '{' and '}' (which are used to represent signed 0) have exactly the code points X'C0' and X'D0'. For example, it works for code page 500 but not for code page 871, where the curly braces have code points X'8E' and X'9C.

In an ASCII environment (determined by the CCSID property at run time), the default for both input and output is the 'Sign ASCII' representation. You can specify the applicable representation in the CWF physical layer for local attributes and local elements of types decimal, float, and integer.

Note: This option is appropriate only for those elements or attributes that have an external decimal physical representation, and that have an embedded ('Leading' or 'Trailing') sign (determined by the **Sign Orientation** property).

The following table shows the internal representation (both character and actual hexadecimal value) of the first or last digit for external decimal numbers with an

included (embedded) leading or trailing sign respectively. (The table does not specify the representation for unsigned values, which are 0x30-0x39 for ASCII and 0xF0-0xF9 for EBCDIC.)

	ASCII environment		EBCDIC environment		ASCII environment		EBCDIC environment
	Positively signed values				Negatively signed values		
Digit	Sign ASCII	Sign EBCDIC Custom			Sign ASCII	Sign EBCDIC Custom	
0	0(30)	{(7B)	{(C0)		p(70)	}(7D)	}(D0)
1	1(31)	A(41)	A(C1)		q(71)	J(4A)	J(D1)
2	2(32)	B(42)	B(C2)		r(72)	K(4B)	K(D2)
3	3(33)	C(43)	C(C3)		s(73)	L(4C)	L(D3)
4	4(34)	D(44)	D(C4)		t(74)	M(4D)	M(D4)
5	5(35)	E(45)	E(C5)		u(75)	N(4E)	N(D5)
6	6(36)	F(46)	F(C6)		v(76)	O(4F)	O(D6)
7	7(37)	G(47)	G(C7)		w(77)	P(50)	P(D7)
8	8(38)	H(48)	H(C8)		x(78)	Q(51)	Q(D8)
9	9(39)	I(49)	I(C9)		y(79)	R(52)	R(D9)

The next table gives some examples for a range of simple numbers that are representative of what can be transmitted or received using these approaches.

	Sign leading			Sign trailing		
	ASCII Environment		EBCDIC Environment	ASCII Environment		EBCDIC Environment
Decimal value	Sign ASCII	Sign EBCDIC Custom		Sign ASCII	Sign EBCDIC Custom	
1234	31 32 33 34 "1234"	31 32 33 34 "1234"	F1 F2 F3 F4 "1234"	31 32 33 34 "1234"	31 32 33 34 "1234"	F1 F2 F3 F4 "1234"
+1234	31 32 33 34 "1234"	41 32 33 34 "A234"	C1 F2 F3 F4 "A234"	31 32 33 34 "1234"	31 32 33 44 "123D"	F1 F2 F3 C4 "123D"
-1234	71 32 33 34 "q234"	4A 32 33 34 "J234"	D1 F2 F3 F4 "J234"	31 32 33 74 "123t"	31 32 33 4D "123M"	F1 F2 F3 D4 "123M"
7890	37 38 39 30 "7890"	37 38 39 30 "7890"	F7 F8 F9 F0 "7890"	37 38 39 30 "7890"	37 38 39 30 "7890"	F7 F8 F9 F0 "7890"
+7890	37 38 39 30 "7890"	47 38 39 30 "G890"	C7 F8 F9 F0 "G890"	37 38 39 30 "7890"	37 38 39 7B "789{"	F7 F8 F9 C0 "789{"
-7890	77 38 39 30 "w890"	50 38 39 30 "P890"	D7 F8 F9 F0 "P890"	37 38 39 70 "789p"	37 38 39 7D "789}"	F7 F8 F9 D0 "789}"

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with a message definition file" on page 2863
Create, open, and delete a message definition file.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

Importing from SCA Export or SCA Import: generated objects:

A number of objects are generated when you import from an SCA export or SCA import.

- A Broker SCA definition file. The file extension is `.insca` for an import component, or `.outsca` for an export component. The `.insca` or `.outsca` files are stored in a Broker SCA Definitions folder under the message set project. If multiple SCA Import/Exports are imported into a message set, each has its own `.insca` or `.outsca` file.

  You can drag the Broker SCA Definition onto the Message Flow Editor to create a skeleton flow.

  – Dragging and dropping a `.insca` object onto the Message Flow editor creates a pair of SCAInput and SCAReply nodes if the Broker SCA definition interface contains one or more request-response operations. Otherwise, only the SCAInput node is created.
  – Dragging and dropping a `.outsca` object onto the Message Flow editor creates either a SCARequest node, or a pair of SCAAsyncRequest and SCAAsyncResponse nodes.
- Corresponding message definitions (`.mxsd`) files are created from the XSD types that are used by the messages specified in the WSDL interface in the SCA Import or Export. The message definitions are stored in their respective target namespaces.
- If the SCA Import or SCA Export being imported uses a Web Services Binding type, the deployable WSDL is imported into the message set.
- If the SCA Import or SCA Export being imported uses an MQ Binding type, a deployable WSDL is not imported into the message set.
- A log is created for the import in the log folder under the message set project.

Related reference:

"New message definition file wizard: Create a new message definition file from an SCA Import or Export" on page 6369
You can create a new message definition file from an SCA import or export.

Importing from WSDL: generated objects and restrictions:

Several objects are generated when you import from WSDL but restrictions might apply.

Generated objects

Files copied by command line import

The **mqsicreatemsgdefsfromwsdl** command copies the WSDL files it needs into the workspace before running the import process. These files are the top level WSDL files and any imports resolved from a relative location. The files are copied under the specified message set into a folder called importFiles.

Report file

The WSDL importer appends the result of the import operation to a report file, listing all errors that occurred during the process. The file name of the report file is *message set*.wsdl.report.txt.

SOAP message definitions

The required SOAP .mxsds files are added to the message set.

To parse SOAP 1.2 instance documents, manually remove the SOAP 1.1 definitions and import the SOAP 1.2 definitions by using the Message Definition File wizard, selecting **IBM supplied message**.

If your message set has TDS or CWF layers, you might find that you get a number of warnings against the imported SOAP definitions. Most of these can be ignored, but take account of the allowed values for Boolean attributes. In SOAP 1.1 the Boolean values are 1 or 0, while in SOAP 1.2 the values are true and false. The XML representation of Boolean values for a message set is specified in the physical properties for the XML physical format, and might need to be set accordingly.

Message definition files

Other message definition file names are created as *input file name*.mxsd and their content depends on the WSDL style.

document-style

WSDL message parts for style="document" (which includes all SOAP header, fault and headerfault parts) refer to an element defined in XML Schema. This element is imported as a global element and broker message in the .xsd or .mxsd file.

The xsi:type output policy on the message is set to "Never".

rpc-style

WSDL message parts for style="rpc" (and exclusively those allocated to the SOAP body) refer to a type defined in XML Schema. In this case, input and output messages are created as shown in the following table.

	An input message	**An output message**
Derived From	The wsdl:input child (if any) of WSDL operation, and the WSDL message and parts that it identifies	The wsdl:output child (if any) of WSDL operation, and the WSDL message and parts that it identifies
Name of Element	The value of the name attribute on the WSDL operation element	The value of the name attribute on the WSDL operation element suffixed by "Response"
Namespace of Element	The value of the namespace attribute on the corresponding soap:body element	The value of the namespace attribute on the corresponding soap:body element

Each message is of local complex type, being a sequence of elements. The name of each element is the value of the name attribute on the WSDL parts

of the message identified by either the input or output element. These elements have no namespace (the underlying schema representation has form="unqualified"), and are locally scoped to avoid name clashes. The type of these local elements is the XML schema type referred to by the type attribute of the corresponding part element. The type is global in the WSDL schema.

If the soap:body element was defined with use="encoded" in the WSDL definition, the message definition includes a reference to the attribute group encodingStyle in the SOAP-ENV namespace and the xsi:type output policy on the message is set to "Follow SOAP encoding rules". Otherwise, the xsi:type output policy on the message is set to "Never".

WSDLs generated using .NET

In some instances, WSDL files that are generated by using .NET include element references to the schema itself; for example:

```
<xsd:complexType>
  <xsd:sequence>
   <xsd:element ref="s:schema"/>
  </xsd:sequence>
</xsd:complexType>
```

For WSDL files of this type to be successfully imported into the WebSphere Message Broker Toolkit without validation errors, you must manually add a namespace import statement to the namespace of the schema; for example:

```
<xsd:import namespace="http://www.w3.org/2001/XMLSchema"/>
```

Place the import statement first in the schema element, and ensure that it appears before any complex type or element definitions. Revalidate the WSDL by right-clicking the updated WSDL and clicking **Validate**.

Restrictions

Restrictions related to importing WSDL definitions exist where the WSDL definitions are not WS-I compliant.

SOAP Arrays

A WSDL 1.1 definition can define a SOAP array (applicable only to the WSDL rpc-encoded style, and not WS-I compliant):

```
<xsd:complexType name="t">
      <xsd:complexContent>
        <xsd:restriction base="SOAP-ENC:Array">
          <xsd:sequence>
            <xsd:element name="item" type="string" maxOccurs="unbounded"/>
          </xsd:sequence>
          <xsd:attribute  ref="SOAP-ENC:arrayType" wsdl:arrayType="xsd:string[]"/>
        </xsd:restriction>
      </xsd:complexContent>
</xsd:complexType>
```

Some uses of SOAP array syntax are not fully supported. Although a useful tree is created during parsing and can be serialized when writing, the following restrictions apply.

- The model does not take any account of the SOAP-ENC:arrayType attribute.
- The model for partially transmitted arrays does not take account of the SOAP-ENC:offset attribute.

For example, the first element of an array specified with offset[2] must be accessed in ESQL, not as InputRoot.MRM.array.item[3], but as InputRoot.MRM.array.item[1].

- The model for multi-dimensional arrays flattens the representation into a single dimension. For example, a 2-dimensional array is accessed in ESQL, not as InputRoot.MRM.array.item[x][y], but as InputRoot.MRM.array.item[i] where the index i has to be calculated appropriately.

Anonymous elements

The WSDL excerpt above describes a SOAP instance document of the following form:

```
<SOAP-ENC:Array SOAP-ENC:arrayType="xsd:string[3]">
   <item xsi:type="xsd:string">A general text string</item>
   <item xsi:type="xsd:token">A restriction of the string type</item>
   <item xsi:type="xsd:Name">ARestrictionOfTheTokenType</item>
</SOAP-ENC:Array>
```

The broker model handles this document as expected, but in SOAP encoding array, elements are also allowed to use the type-elements from the SOAP encoding namespace. Therefore, an application using the same WSDL definition might create an instance document of the following form:

```
<SOAP-ENC:Array SOAP-ENC:arrayType="xsd:string[3]">
   <SOAP-ENC:string>A general text string</SOAP-ENC:string>
   <SOAP-ENC:token>A restriction of the string type</SOAP-ENC:token>
   <SOAP-ENC:Name>ARestrictionOfTheTokenType</ SOAP-ENC:Name>
</SOAP-ENC:Array>
```

To handle this case, you must manually edit the broker model that is created by importing the WSDL, unless it is acceptable to have the parser treat it as a self-defined element.

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with a message definition file" on page 2863
Create, open, and delete a message definition file.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Import formats" on page 6346
Supported features of formats that have been imported from C, COBOL, SCA Export, SCA Import, WSDL, or XML Schema.

Importing from XML Schema: unsupported features:

A number of features in XML Schema are not supported, or their support is restricted in some way.

Message sets with namespace support

- Constructs accepted but not supported when importing from an XML Schema.

  When importing an XML Schema into a message set that supports namespaces, the Redefine construct is accepted, but causes an error message to be displayed in the task list because it is not fully supported.

  The following XML shows an example of the Redefine construct:

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.ibm.com" xmlns:ibm="http://www.ibm.com">

 <!-- Unsupported feature: redefine -->
 <redefine schemaLocation="test.xsd"/>

</schema>
```

Message sets without namespace support

- Constructs accepted and ignored when importing from an XML Schema.

  The list of constructs and the action taken is the same as for a message set with namespace support, as described above.

- Target namespaces not qualified with a prefix.

  When importing an XML Schema into a message set that does not support namespaces, you cannot import a schema document that has a target namespace that is not qualified with a prefix. For example:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
       targetNamespace="http://www.ibm.com" xmlns="http://www.ibm.com">
</xsd:schema>
```

Related concepts:

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"The message model" on page 1160
The message model consists of the following components.

"MRM XML physical format: Relationship to the logical model" on page 1251
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Working with a message definition file" on page 2863
Create, open, and delete a message definition file.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

"Working with data structures" on page 2930
You can create a message definition file in a message set by importing from XML Schema, XML DTD, SCA import or export components, IBM supplied messages, WSDL definitions, IDL files, C header files, and COBOL copybooks. This topic area describes how to import from these data structures using the command line or the WebSphere Message Broker Toolkit.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

"Message model object properties" on page 5416
Access property information by property kind, or by object.

"Import formats" on page 6346
Supported features of formats that have been imported from C, COBOL, SCA Export, SCA Import, WSDL, or XML Schema.

Message model wizards

Wizards help to simplify complex message modeling tasks.

As software grows more complex, wizards are increasingly used to step you through complex tasks or procedures, ensuring that you correctly specify all the parameters that are required, and that you perform the required tasks in the correct order.

This topic provides some additional reference material for those wizards where you might need help in specifying certain parameters.

Each wizard that is documented here has its own high-level topic and a topic for each panel that is displayed by the wizard. The panels are listed in the order that they appear, and the fields on each panel appear in the topic in the same order as they appear on the panel. These topics provide only information about these fields and panels. You can find further information about the wizards in topics that are referenced from the wizard's high-level topic.

The following wizards have additional information:

- "New message definition file wizards"
- "Generate WSDL wizard" on page 6382
- "Export WSDL wizard" on page 6390
- "Configure New Web Service Usage wizard" on page 6392

Related tasks:

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

Related reference:

"Message model reference information" on page 5366
Reference information in this section can help you develop and configure message models.

New message definition file wizards:

Use the New message definition file wizards to create message definition files.

Depending on the selection that you make, you are routed through the correct sequence of panels to create the message definition file from the source that you have requested. Some panels are displayed only if certain conditions are met. These panels are marked as optional.

The following links provide further information about the panels and fields that form the New message definition file wizards.

- New Message Definition File (from scratch)
- C header file
- COBOL file
- CORBA IDL file
- IBM supplied message
- SCA Import or Export
- WSDL file
- XML DTD
- XML Schema file
- Database definition

Related tasks:

"Creating a message definition file" on page 2865
Creating an empty message definition file to contain your message model objects.

"Importing an IDL file" on page 2952
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to create a message definition from an IDL file.

"Importing from XML Schema" on page 2957
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to create a new message definition from an XML Schema

"Importing from IBM supplied messages" on page 2942
You can create a new message definition file from an IBM supplied message.

"Importing from XML DTD" on page 2954
You can create a new message definition from an XML DTD by using the New Message Definition File wizard in the WebSphere Message Broker Toolkit.

"Importing from C" on page 2934
Create a message definition file from a C header file for use in the MRM and IDOC domains.

"Importing from COBOL copybooks" on page 2937
This topic describes how to create a new message definition from a COBOL data structure using the New Message Definition File wizard in the WebSphere Message Broker Toolkit.

"Importing from WSDL" on page 2946
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to create a new message definition from WSDL.

New message definition file wizard: Create a new message definition file from scratch:

Create a new message definition file by using the New message definition file wizard.

Create a new message definition file from scratch

When you create a new message definition file from scratch, you must set the following fields:

Message sets

Select the target message set
This field lists the message set projects that are available in your workspace. Click the down-arrow and select the appropriate message set from the list displayed. Depending on how you started the New message definition file wizard, a message set might be preselected for you, but this does not prevent you from selecting a different message set if you prefer.

Message definition file name
Specify the name of the message definition file that you are creating.

Schema for Schema settings

Prefix Specifies the namespace prefix to use for the namespace shown in the **Namespace** property.

Namespace
Specifies the namespace to be used.

Target namespace settings

Use target namespace
Selecting this check box allows you to specify a target namespace for the message definition file. You can choose a target namespace only if namespaces have been enabled in the message set.

Prefix Specifies the namespace prefix to use for the namespace shown in the **Namespace** property.

Namespace
Specifies the namespace to be used.

Related tasks:

"Creating a message definition file from scratch" on page 2866
Create an empty message definition file to contain message model objects.

Related reference:

"New message definition file wizards" on page 6360
Use the New message definition file wizards to create message definition files.

New message definition file wizard: Create a new message definition file from a C header file:

You can create a new message definition file from a C header file.

Create a new message definition file from a C header file

When you create a new message definition file from a C header file, you must set the following fields:

Select a C header file

Message set
Use this field to choose the message set project that will contain the message definition file that you create. Message set projects are filtered to only show artifacts in the active working set.

Message definition file name
Use this field for the name of the message definition file that you want to create.

Target namespace
Use this field for the name of the target namespace for the message definition file that you want to create.

Select file from workspace
Choose this option if the C header file that you want to add to the message definition file that you are creating is in the current workspace, and select the file from the displayed content of the workspace. C header files are filtered to only show artifacts in the active working set.

Select file from outside workspace
Choose this option if the C header file that you want to add to the message definition file that you are creating is not in the current workspace, and specify the location of the C header file that you want to add.

Copy source file into the 'importFiles' directory of the message set project
Select this check box to copy the source file into the 'importFiles' directory of the message set project.

Create an appropriate physical format if one does not already exist
Select this check box to automatically create the message set physical format that is needed by the pre-canned schema that you have selected for import.

C header file options

Include paths

Preserve case in variable names
Select this check box if you want to preserve the case of the characters that form the names of the variables.

Select the pre-processing option to apply
Choose an option from the list.

Related tasks:

"Importing from XML Schema" on page 2957
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to create a new message definition from an XML Schema

Related reference:

"New message definition file wizards" on page 6360
Use the New message definition file wizards to create message definition files.

New message definition file wizard: Create a new message definition file from a COBOL file:

You can create a new message definition file from a COBOL file.

Create a new message definition file from a COBOL file

When you create a new message definition file from a COBOL file, you must set the following fields:

Select a COBOL file

Message set
Use this field to choose the message set project that will contain the message definition file that you create. Message set projects are filtered to only show artifacts in the active working set.

Message definition file name
Use this field for the name of the message definition file that you want to create.

Target namespace
Use this field for the name of the target namespace for the message definition file that you want to create.

Select file from workspace
Choose this option if the COBOL file that you want to add to the message definition file that you are creating is in the current workspace, and select the file from the displayed content of the workspace. COBOL files are filtered to only show artifacts in the active working set.

Select file from outside workspace
Choose this option if the COBOL file that you want to add to the message definition file that you are creating is not in the current workspace, and specify the location of the COBOL file that you want to add.

Copy source file into the 'importFiles' directory of the message set project
Select this check box to copy the source file into the 'importFiles' directory of the message set project.

Create an appropriate physical format if one does not already exist
Select this check box to automatically create the message set physical format that is needed by the pre-canned schema that you have selected for import.

COBOL file options

Preserve case in variable names
Select this check box if you want to preserve the case of the characters that form the names of the variables.

Related tasks:

"Importing from XML Schema" on page 2957
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to create a new message definition from an XML Schema

Related reference:

"New message definition file wizards" on page 6360
Use the New message definition file wizards to create message definition files.

New message definition file wizard: Create a new message definition file from a CORBA IDL file:

You can create a new message definition file from a CORBA IDL file.

Create a new message definition file from a CORBA IDL file

When you create a new message definition file from an IDL file, you must set the following fields:

Select an IDL file

Message set
Use this field to choose the message set project that will contain the message definition file that you create. Message set projects are filtered to show artifacts in the active working set.

Message definition file name
Use this field for the name of the message definition file that you want to create.

Target namespace
Use this field for the name of the target namespace for the message definition file that you want to create.

Select file from workspace
Choose this option if the IDL file that you want to add to the message definition file that you are creating is in the current workspace, and select the file from the displayed content of the workspace. IDL files are filtered to show artifacts in the active working set.

Select file from outside workspace
Choose this option if the IDL file that you want to add to the message definition file that you are creating is not in the current workspace, and specify the location of the IDL file that you want to add.

Add DataObject to supported message domains if they do not exist
Select this check box to add the DataObject message domain to the list of supported message domains that the message definition supports.

Related concepts:

"Common Object Request Broker Architecture (CORBA)" on page 2145
The Common Object Request Broker Architecture (CORBA) is a standard defined by the Object Management Group (OMG) that enables software components written in multiple computer languages and running on multiple computers to work together.

Related tasks:

"Importing an IDL file" on page 2952
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to create a message definition from an IDL file.

"Connecting to an external CORBA application" on page 2159
Connecting to an external CORBA application involves importing an IDL file, creating a message flow, building a message, and processing the response from the CORBARequest node.

"Processing responses from a CORBARequest node" on page 2167
Configure the CORBARequest node to define the location to which responses are sent.

Related reference:

"New message definition file wizards" on page 6360
Use the New message definition file wizards to create message definition files.

New message definition file wizard: Create a new message definition file from a database definition:

You can create a new message definition from a database definition file (`.dbm`) by using the New Message Definition File wizard in the WebSphere Message Broker Toolkit.

Create a new message definition file from a database definition

When you create a new message definition file from a database definition, you must set the following fields:

Select a database definition

Message set
Use this field to choose the message set project that will contain the message definition file that you create. Message set projects are filtered to only show artifacts in the active working set.

Message definition file name
Use this field for the name of the message definition file that you want to create.

Select file from workspace
Choose this option if the database definition that you want to add to the message definition file that you are creating is in the current workspace, and select the file from the displayed content of the workspace. Database definitions are filtered to only show artifacts in the active working set.

Select file from outside workspace
Choose this option if the database definition that you want to add to the message definition file that you are creating is not in the current workspace, and specify the location of the database definition that you want to add.

Copy source file into the 'importFiles' directory of the message set project
Select this check box to copy the source file into the 'importFiles' directory of the message set project.

Model unknown user-defined database types as XML schema
If the database definition file does not contain information about user-defined data types that are present in the table that you are importing, you must specify how the data is modeled. Choices are `string` or `HexBinary`.

Select the tables to import
Select the tables that you want to import. If you select tables from more than one schema, a `.mxsd` file is created for each schema. Each selected table is created as a global xs:element with a message annotation, so that the table can be used directly in Mapping and other similar nodes.

Related tasks:

"Importing from XML Schema" on page 2957
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to create a new message definition from an XML Schema

Related reference:

"New message definition file wizards" on page 6360
Use the New message definition file wizards to create message definition files.

New message definition file wizard: IBM supplied message:

You can create a new message definition file from an IBM supplied message.

IBM supplied message wizard

When you create a message definition file from an IBM supplied message, you must set the following fields:

Select an IBM supplied message

Message set
Use this field to choose the message set project that will contain the

message definition file that you create. Message set projects are filtered to only show artifacts in the active working set.

Message definition file name
Use this field for the name of the message definition file that you want to create.

IBM supplied messages
Select from the displayed set of IBM supplied message definitions. This field is split into two panes; the pane on the left displays the IBM supplied message definitions that are available, and the pane on the right contains text that gives advice about the usage of the message definition that you have selected in the field's left pane.

Copy source file into the importFiles directory of the message set project
Select this check box to copy the source file into the importFiles directory of the message set project.

Create an appropriate physical format if one does not already exist
Select this check box to automatically create the message set physical format that is required by the precanned schema that you have selected for import.

Related tasks:

"Importing from IBM supplied messages" on page 2942
You can create a new message definition file from an IBM supplied message.

Related reference:

"New message definition file wizards" on page 6360
Use the New message definition file wizards to create message definition files.

"IBM supplied messages that you can import"
You can import IBM supplied messages to create a new message definition file.

IBM supplied messages that you can import:

You can import IBM supplied messages to create a new message definition file.

If the message is to be used with an XML parser, the following points apply:

- If the message set to which you are adding the new message definition file has an XML physical format layer, but does not have namespace support, the imported IBM supplied message is modified to remove namespaces. Therefore, enable namespace support before you import an IBM supplied message.
- If the message set to which you are adding the new message definition file does not have an XML physical format layer, but has namespace support, only the logical information is displayed in the model. The IBM supplied message is not modified to remove namespaces. Add an XML physical format to the message set before you import an IBM supplied message.
- If the message set to which you are adding the new message definition file does not have an XML physical format layer, and does not have namespace support, only the logical information appears in the model and the imported IBM supplied message is modified to remove namespaces.

The IBM supplied messages that you can import are:

SOAP message definitions
These message definitions model the SOAP-defined portions of SOAP XML messages. They are best used with the SOAP parser. The definitions *Soap 1.1 Envelope* and *Soap 1.2 Envelope* model the SOAP envelope structure that is used to wrapper the user-defined body of a SOAP message. The

definitions *Soap 1.1 Encoding* and *Soap 1.2 Encoding* model certain structures for use in "rpc/encoded" style SOAP messages.

An IBM message for the SOAP domain tree is supplied as a schema that provides content-assist in creating a logical model for the SOAP domain by using the ESQL or mapping editor.

Multipart MIME message definitions
These message definitions model the MIME-defined portions of multipart MIME messages and must be used with the message broker's MIME parser. Use the *MIME multipart header* definition for typical multipart MIME messages such as SOAP with Attachments, or RosettaNet. Use the *MIME Nested Multipart header* definition for multipart MIME messages in which the individual parts can themselves be multipart MIME; for example, S/MIME.

SAP IDoc message definitions
These message definitions model the SAP-defined portion of ALE and File IDocs that precede the user-defined content. The ALE IDoc model can be used with the MRM and IDOC parsers. The File IDoc model can be used with the MRM parser only.

Timeout Request message definition
This message definition models the TimeoutRequest message that is used in conjunction with the message broker TimeoutControl and TimeoutNotification nodes. You can use it with any parser.

CSV message definition
This message definition models a CSV (comma separated value) format message. It can be used with the MRM parser.

Related concepts:

"The structure of a SOAP message" on page 1605
A SOAP message is encoded as an XML document, consisting of an `<Envelope>` element, which contains an optional `<Header>` element, and a mandatory `<Body>` element. The `<Fault>` element, contained in `<Body>`, is used for reporting errors.

"MIME messages" on page 1120
A MIME message consists of both data and metadata. MIME metadata consists of HTTP-style headers and MIME boundary delimiters.

Related tasks:

"Importing from IBM supplied messages" on page 2942
You can create a new message definition file from an IBM supplied message.

"Building the message model for the IDOC parser" on page 6330
The ALE IDoc messages that are sent to, and received from, SAP applications by using the WebSphere MQ Link for R3, can be processed by the IDOC parser, which requires a message model to interpret the data correctly. This topic describes how to build the message model.

"Building the MRM TDS model for an IDoc" on page 6279
The MRM domain Tagged/Delimited String (TDS) physical format is suitable for parsing and writing SAP ALE IDocs and SAP File IDocs. ALE IDoc messages are exported from SAP across the WebSphere MQ Link for R3. File IDocs are exported from SAP to the file system.

Related reference:

"New message definition file wizard: IBM supplied message" on page 6366
You can create a new message definition file from an IBM supplied message.

"Example XML timeout request message" on page 2812
The format used here is XML, but you can use any format that is supported by an

installed parser.

"CSV messaging standard" on page 6276
The comma separated value (CSV) format is a typical format for describing data in tables or spreadsheets.

New message definition file wizard: Create a new message definition file from an SCA Import or Export:

You can create a new message definition file from an SCA import or export.

Create a new message definition file from an SCA Import or Export

When you create a new message definition file from an SCA import or export file, you must set the following fields:

Select an SCA Import or Export

Message set
Use this field to choose the message set project that is to contain the message definition file that you create.

Message definition file name
Use this field for the name of the message definition file that you want to create.

Select a file or archive from workspace
Choose this option if the SCA import or export file, or the archive that contains the SCA import or export file, that you want to add to the message definition file that you are creating is in the current workspace, and select the file from the displayed content of the workspace. Only files with the following extensions are shown:

.zip (an archive)

.insca (an archive)

.outsca (an archive)

.import

.export

If you select a file with an extension of .insca, .outsca, .import, or .export, only one SCA import or export file is selected.

Select archive from outside workspace
Choose this option if the archive that contains the SCA import or export file that you want to add to the message definition file that you are creating is not in the current workspace, and specify the location of the archive file that you want to add.

Copy source file into the 'importFiles' directory of the message set project
Select this check box to copy the source file into the 'importFiles' directory of the message set project.

Add XMLNSC to supported message domains if it does not exist
Select this check box to add the XMLNSC message domain to the list of supported message domains that the message definition supports.

Create an appropriate physical format if one does not already exist
Select this check box to automatically create an XML wire format for the message set.

Related tasks:

"Importing SCA import or SCA export components" on page 2943
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to import SCA import or SCA export components from WebSphere Integration Developer. You must import an SCA import or SCA export into the workspace to provide a broker SCA definition for use in configuring the SCA nodes.

"Importing from XML Schema" on page 2957
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to create a new message definition from an XML Schema

Related reference:

"New message definition file wizards" on page 6360
Use the New message definition file wizards to create message definition files.

New message definition file wizard: Create a new message definition file from a WSDL file:

You can create a new message definition file from a WSDL file.

Create a new message definition file from a WSDL file

When you create a new message definition file from a WSDL file, you must set the following fields:

Select a WSDL file

Message set
Use this field to choose the message set project that will contain the message definition file that you create. Message set projects are filtered to only show artifacts in the active working set.

Message definition file name
Use this field for the name of the message definition file that you want to create.

Select file from workspace
Choose this option if the WSDL file that you want to add to the message definition file that you are creating is in the current workspace, and select the file from the displayed content of the workspace. WSDL files are filtered to only show artifacts in the active working set.

Select file from outside workspace
Choose this option if the WSDL file that you want to add to the message definition file that you are creating is not in the current workspace, and specify the location of the WSDL file that you want to add.

Copy source file into the 'importFiles' directory of the message set project
Select this check box to copy the source file into the 'importFiles' directory of the message set project.

Create an appropriate physical format if one does not already exist
Select this check box to automatically create the message set physical format that is needed by the pre-canned schema that you have selected for import.

Add SOAP and XMLNSC to supported message domains if they do not exist
Select this check box to add the SOAP and XMLNSC message domains to the list of supported message domains that the message definition supports.

Related tasks:

"Importing from XML Schema" on page 2957
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to create a new message definition from an XML Schema

Related reference:

"New message definition file wizards" on page 6360
Use the New message definition file wizards to create message definition files.

New message definition file wizard: Create a new message definition file from an XML DTD file:

You can create a new message definition file from an XML DTD file.

Create a new message definition file from an XML DTD file

When you create a new message definition file from an XML DTD file, you must set the following fields:

Select an XML DTD file

Message set
Use this field to choose the message set project that will contain the message definition file that you create. Message set projects are filtered to only show artifacts in the active working set.

Message definition file name
Use this field for the name of the message definition file that you want to create.

Select file from workspace
Choose this option if the XML DTD file that you want to add to the message definition file that you are creating is in the current workspace, and select the file from the displayed content of the workspace. XML DTD files are filtered to only show artifacts in the active working set.

Select file from outside workspace
Choose this option if the XML DTD file that you want to add to the message definition file that you are creating is not in the current workspace, and specify the location of the XML DTD file that you want to add.

Copy source file into the 'importFiles' directory of the message set project
Select this check box to copy the source file into the 'importFiles' directory of the message set project.

Create an appropriate physical format if one does not already exist
Select this check box to automatically create the message set physical format that is needed by the pre-canned schema that you have selected for import.

Related tasks:

"Importing from XML Schema" on page 2957
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to create a new message definition from an XML Schema

Related reference:

"New message definition file wizards" on page 6360
Use the New message definition file wizards to create message definition files.

New message definition file wizard: Create a new message definition file from an XML Schema file:

You can create a new message definition file from an XML Schema file.

Create a new message definition file from an XML Schema file

When you create a new message definition file from an XML Schema file, you must set the following fields:

Select an XML Schema file

Message set
Use this field to choose the message set project that will contain the message definition file that you create.

Message definition file name
Use this field for the name of the message definition file that you want to create.

Select file from workspace
Choose this option if the XML Schema file that you want to add to the message definition file that you are creating is in the current workspace, and select the file from the displayed content of the workspace.

Select file from outside workspace
Choose this option if the XML Schema file that you want to add to the message definition file that you are creating is not in the current workspace, and specify the location of the XML Schema file that you want to add.

Copy source file into the 'importFiles' directory of the message set project
Select this check box to copy the source file into the 'importFiles' directory of the message set project.

Create an appropriate physical format if one does not already exist
Select this check box to automatically create the message set physical format that is needed by the pre-canned schema that you have selected for import.

Related tasks:

"Importing from XML Schema" on page 2957
You can use the New Message Definition File wizard in the WebSphere Message Broker Toolkit to create a new message definition from an XML Schema

Related reference:

"New message definition file wizards" on page 6360
Use the New message definition file wizards to create message definition files.

Generate Broker SCA Definition wizard:

The Generate Broker SCA Definition wizard creates a Broker SCA definition from a message set.

The following links provide further information in relation to the panels and fields that form the Generate Broker SCA Definition wizard. Some panels only appear if certain conditions are met; such panels are marked as (optional).

Open the Generate Broker SCA Definition wizard

To open the Generate Broker SCA Definition wizard:

1. Switch to the Broker Application Development perspective.
2. In the Broker Development view, right-click the message set folder from which you want to generate a Broker SCA definition, and select **Generate** > **Broker SCA Definition**. This action starts the Generate Broker SCA Definition wizard.

Generate Broker SCA Definition wizard

The following panels are shown by the Generate Broker SCA Definition wizard:

- Generate a Broker SCA Definition
- Specify the interface used by the Broker SCA Definition
- Create an interface for the Broker SCA Definition
- Configure Web service binding details
- Configure MQ binding details
- Verify the SCA Import or Export definition that will be generated

Related tasks:

"Generating a Broker SCA definition from a message set" on page 2967
WebSphere Message Broker creates a Broker SCA definition from existing message definitions.

Generate Broker SCA Definition wizard: Generate a Broker SCA Definition:

Use this panel to select the message set folder which contains the source of the Broker SCA definition and to specify the type of Broker SCA definition you want to create.

Generate Broker SCA Definition wizard

The following panels are shown by the Generate Broker SCA Definition wizard:

- Generate a Broker SCA Definition
- Specify the interface used by the Broker SCA Definition
- Create an interface for the Broker SCA Definition
- Configure Web service binding details
- Configure MQ binding details
- Verify the SCA Import or Export definition that will be generated

Panel properties

Select the message set folder from which to generate the Broker SCA definition
Specify the message set folder from which to generate the Broker SCA definition.

Select the type of SCA artifact you want to create
Select from:

- **Inbound Broker SCA definition**

  The Broker SCA definition is an SCA import and has a file extension of `.insca`. This is the default.

- **Outbound Broker SCA definition**

  The Broker SCA definition is an SCA export and has a file extension of `.outsca`.

Binding type
Binding type determines which type of binding the Broker SCA definition is to provide. The choices are:

- Web Service
- WebSphere MQ

Broker SCA definition name
Specify a name for the Broker SCA definition.

Related tasks:

"Generating a Broker SCA definition from a message set" on page 2967
WebSphere Message Broker creates a Broker SCA definition from existing message definitions.

Related reference:

"Generate Broker SCA Definition wizard" on page 6372
The Generate Broker SCA Definition wizard creates a Broker SCA definition from a message set.

Generate Broker SCA Definition wizard: panels to specify the WSDL interface:

Use this panel to specify the WSDL interface to be used by the Broker SCA definition.

Generate Broker SCA Definition wizard

The following panels are shown by the Generate Broker SCA Definition wizard:

- Generate a Broker SCA Definition
- Specify the interface used by the Broker SCA Definition
- Create an interface for the Broker SCA Definition
- Configure Web service binding details
- Configure MQ binding details
- Verify the SCA Import or Export definition that will be generated

Panel properties

The following options are displayed:

Create a new WSDL definition
This is the default. You must specify a namespace

Use an existing deployable WSDL from the workspace
Select from your workspace directory the WSDL to be used with this Broker SCA definition. If you select a WSDL that has multiple bindings or ports, you must select which binding or port is to be used:

- **Select a binding**
  Select the binding that you want to use.
- **Select a port**
- Select the port that you want to use.

Related tasks:

"Generating a Broker SCA definition from a message set" on page 2967
WebSphere Message Broker creates a Broker SCA definition from existing message definitions.

Related reference:

"Generate Broker SCA Definition wizard" on page 6372
The Generate Broker SCA Definition wizard creates a Broker SCA definition from a message set.

Generate Broker SCA Definition wizard: Panels to create a new WSDL definition:

You can create a new WSDL definition to be used by the Broker SCA definition that you are generating with this wizard.

You see the following panels:

- Create an interface for the Broker SCA definition. You see this panel if you chose, on the previous panel, to create a new WSDL definition.
- Configure binding details. This panel is always displayed, but the binding details that are shown depend on the binding type.

Related tasks:

"Generating a Broker SCA definition from a message set" on page 2967
WebSphere Message Broker creates a Broker SCA definition from existing message definitions.

Related reference:

"Generate Broker SCA Definition wizard" on page 6372
The Generate Broker SCA Definition wizard creates a Broker SCA definition from a message set.

Generate Broker SCA Definition wizard: Create an interface for the Broker SCA Definition:

Use this panel to define the operations that you want to add to the WSDL definition that you are creating as the interface for use by this Broker SCA definition.

Generate Broker SCA Definition wizard

The following panels are shown by the Generate Broker SCA Definition wizard:

- Generate a Broker SCA Definition
- Specify the interface used by the Broker SCA Definition
- Create an interface for the Broker SCA Definition
- Configure Web service binding details
- Configure MQ binding details
- Verify the SCA Import or Export definition that will be generated

Panel properties

The panel is divided into two panes.

The top pane is read-only and displays a table that describes the operations that you have defined. The table has four columns with the following headings:

Operation
The name that you have given to the operation.

Input message
The name of the input message.

Output message
The name of the output message. This might be blank if no output message is specified for this operation.

Operation Type
The type of operation. Examples of operation type are:
Request-response
One-way

The bottom pane is where you describe a new operation. The following fields describe the operation:

Name The name that you have given to the operation.

Operation Type
The type of operation. Examples of operation type are:
Request-response
One-way

Input The name of the input message.

Output
The name of the output message. This is omitted for a *One-way* operation.

Fault The name of the fault message. You can specify one or more fault messages.

Related tasks:

"Generating a Broker SCA definition from a message set" on page 2967
WebSphere Message Broker creates a Broker SCA definition from existing message definitions.

Related reference:

"Generate Broker SCA Definition wizard" on page 6372
The Generate Broker SCA Definition wizard creates a Broker SCA definition from a message set.

Generate Broker SCA Definition wizard: Configure Web service binding details:

Use this panel to specify the SOAP/HTTP binding details for the WSDL definition that you are creating for the Broker SCA definition.

Generate Broker SCA Definition wizard

The following panels are shown by the Generate Broker SCA Definition wizard:

- Generate a Broker SCA Definition
- Specify the interface used by the Broker SCA Definition
- Create an interface for the Broker SCA Definition
- Configure Web service binding details
- Configure MQ binding details
- Verify the SCA Import or Export definition that will be generated

Panel properties

The following properties are displayed:

SOAP action
The value for the HTTP SoapAction header. It is possible that an

application might use the SoapAction as a mechanism for relating a SOAP message to an implementation method. This is often true with rpc-style WSDL.

Service name
The value of the name attribute on the service element in the generated WSDL. The exact use of the name depends on products that then use the WSDL such as the SOAP toolkits and UDDI repositories. For example, if you then use a SOAP toolkit to generate Java from your WSDL, the service name is likely to become the Java interface name.

Port name
Port name is the name of a specific WSDL port for this service and would usually be derived from the Service Name. One convention would be to provide a service name of <xyz> Service and a port name of <xyz> Provider.

The value of the name attribute on the port element in the generated WSDL. The exact use of the name depends on products that then use the WSDL such as SOAP toolkits and UDDI repositories. For example if you use a SOAP toolkit to generate Java from your WSDL, the port name could become a Java class name.

Port address
Port address defines the address at which the service is to be made available. It must be a valid URL and it must include the port number, if this is different from the default HTTP port. An example of a port address is: `http://localhost:7800/abcSOAP_HTTP_Service` for an inbound Broker SCA definition.

If you are generating an outbound Broker SCA definition, the port address property expands to present the following additional properties:

- **Module project name**

  The name of the module project in which you intend to place the exported Broker SCA definition in WebSphere Integration Developer.
- **Host name**

  The host name of the server for WebSphere Process Server.
- **Port number**

  The port number of the server for WebSphere Process Server.

The port address is generated for you using the values you provide in **Module project name**, **Host name**, and **Port number** in the format that is required by WebSphere Integration Developer.

Related tasks:

"Generating a Broker SCA definition from a message set" on page 2967
WebSphere Message Broker creates a Broker SCA definition from existing message definitions.

Related reference:

"Generate Broker SCA Definition wizard" on page 6372
The Generate Broker SCA Definition wizard creates a Broker SCA definition from a message set.

Generate Broker SCA Definition wizard: Configure MQ binding details:

Use this panel to specify the MQ binding details for the Broker SCA definition.

Generate Broker SCA Definition wizard

The following panels are shown by the Generate Broker SCA Definition wizard:

- Generate a Broker SCA Definition
- Specify the interface used by the Broker SCA Definition
- Create an interface for the Broker SCA Definition
- Configure Web service binding details
- Configure MQ binding details
- Verify the SCA Import or Export definition that will be generated

Panel properties

Certain properties are displayed only when an outbound binding is being configured. These are indicated in the following list of properties:

Host name
The machine that hosts WebSphere MQ. The default is `localhost`.

Port The port on which to connect to a specific WebSphere MQ queue manager. The default is `1414`.

Channel
The WebSphere MQ channel. The default is `SYSTEM.DEF.SVRCONN`.

Request queue manager
The WebSphere MQ queue manager where the request queue resides.

Request queue
The queue to which request messages are sent.

Response queue
(This property is displayed only if the interface has one at least one request-response operation.) The queue to which response messages are sent.

Response queue manager
(Outbound binding only. This property is displayed only if the interface has one at least one request-response operation.) The WebSphere MQ queue manager where the response queue resides, if different from the request queue manager. This field is optional; if it is left blank, the Response queue manager defaults to the Request queue manager.

Request data binding handler

If the default message domain of the message set in which the Broker SCA definition is to be generated is XMLNSC, XMLNS or XML, the default data binding handler is UTF8XMLDataHandler. Otherwise, the default data binding handler is empty, and you can select one of the following options:

COBOL, C, PL/I language data binding generator

CSVDataHandler

Fixed width

You can enter a custom handler instead; the handler must exist in the WebSphere Integration Developer module.

Response data binding handler
(This property is displayed only if the interface has one at least one request-response operation.) If the default message domain of the message set in which the Broker SCA definition is to be generated is XMLNSC,

XMLNS or XML, the default data binding handler is UTF8XMLDataHandler. Otherwise, the default data binding handler is empty, and you can select one of the following options:

COBOL, C, PL/I language data binding generator

CSVDataHandler

Fixed width

You can enter a custom handler instead; the handler must exist in the WebSphere Integration Developer module.

Response correlation ID options
(Outbound binding only.) The message exchange mechanism used to correlate request messages with their response messages. Choose from `From Request Message ID` (default), or `From Request Correlation ID`.

Related tasks:

"Generating a Broker SCA definition from a message set" on page 2967
WebSphere Message Broker creates a Broker SCA definition from existing message definitions.

Related reference:

"Generate Broker SCA Definition wizard" on page 6372
The Generate Broker SCA Definition wizard creates a Broker SCA definition from a message set.

Generate Broker SCA Definition wizard: Verify the Broker SCA definition to be generated:

This panel provides a summary of the Broker SCA definition that you are generating in this wizard.

Generate Broker SCA Definition wizard

The following panels are shown by the Generate Broker SCA Definition wizard:

- Generate a Broker SCA Definition
- Specify the interface used by the Broker SCA Definition
- Create an interface for the Broker SCA Definition
- Configure Web service binding details
- Configure MQ binding details
- Verify the SCA Import or Export definition that will be generated

This panel consists of two panes. The top pane displays summary information about the Broker SCA definition that is to be generated:

Message set
The message set folder from which the Broker SCA Definition is to be generated.

SCA artifact type
The type of Broker SCA definition: inbound or outbound.

- An inbound Broker SCA definition is the WebSphere Message Broker Toolkit representation of an SCA Import and has a file extension of `.insca`.
- An outbound Broker SCA definition is the WebSphere Message Broker Toolkit representation of an SCA Export and has a file extension of `.outsca`.

SCA artifact name
The name of the Broker SCA definition.

Binding type
The binding type of the Broker SCA definition.

The bottom pane allows you to start the Export SCA Import or Export from Broker SCA Definition wizard:

Start Export SCA Import or Export from Broker SCA Definition wizard after generation is finished
Select the check box for the Export SCA Import or Export from Broker SCA Definition wizard to start on completion of this wizard.

When the wizard has finished, the Broker SCA definition which has been created is added to the specified message set project.

Related tasks:

"Generating a Broker SCA definition from a message set" on page 2967
WebSphere Message Broker creates a Broker SCA definition from existing message definitions.

Related reference:

"Generate Broker SCA Definition wizard" on page 6372
The Generate Broker SCA Definition wizard creates a Broker SCA definition from a message set.

Export SCA Import or Export from Broker SCA Definition wizard:

The Export SCA Import or Export from Broker SCA Definition wizard exports an SCA import or export from a Broker SCA definition in a message set.

The following links provide further information in relation to the panels and fields that form the Export SCA Import or Export from Broker SCA Definition wizard.

Export SCA Import or Export from Broker SCA Definition wizard

The following panels are shown by the Export SCA Import or Export from Broker SCA Definition wizard:

- Select the SCA Import or SCA definition you want to export
- Specify the export location

Related tasks:

"Generating a Broker SCA definition from a message set" on page 2967
WebSphere Message Broker creates a Broker SCA definition from existing message definitions.

Export SCA Import or Export from Broker SCA definition wizard: Select the Broker SCA definition to export:

Use this panel of the Export SCA Import or Export from Broker SCA Definition wizard to select the SCA import or export definition that you want to export from a message set.

The following links provide further information in relation to the panels and fields that form the Export SCA Import or Export from Broker SCA Definition wizard.

Export SCA Import or Export from Broker SCA Definition wizard

The following panels are shown by the Export SCA Import or Export from Broker SCA Definition wizard:

- Select the SCA Import or SCA definition you want to export
- Specify the export location

Panel properties

The top pane of the panel contains a navigation tree of your workspace. Select the SCA import or export definition that you want to export.

SCA artifact name
The name of the exported SCA import or export.

Related tasks:

"Generating a Broker SCA definition from a message set" on page 2967
WebSphere Message Broker creates a Broker SCA definition from existing message definitions.

Related reference:

"Export SCA Import or Export from Broker SCA Definition wizard" on page 6380
The Export SCA Import or Export from Broker SCA Definition wizard exports an SCA import or export from a Broker SCA definition in a message set.

Export SCA Import or Export from Broker SCA definition wizard: Specify the export location:

Use this panel of the Export SCA Import or Export from Broker SCA Definition wizard to specify the location for the SCA import or export definition that you want to export from a message set.

The following links provide further information in relation to the panels and fields that form the Export SCA Import or Export from Broker SCA Definition wizard.

Export SCA Import or Export from Broker SCA Definition wizard

The following panels are shown by the Export SCA Import or Export from Broker SCA Definition wizard:

- Select the SCA Import or SCA definition you want to export
- Specify the export location

Panel properties

Choose one of the following options:

Export to a workspace directory
The structure of the workspace is displayed. Click the folder to which you want the SCA import or export definition to be exported.

Export to an external directory
Specify the name of the external directory to which you want the SCA import or export definition to be exported.

Select the **Overwrite existing files without warning** check box if you do not want to be warned that an existing file is about to be overwritten. By default, the check box is cleared, and you are prompted to confirm that you want to replace existing files.

Related tasks:

"Generating a Broker SCA definition from a message set" on page 2967
WebSphere Message Broker creates a Broker SCA definition from existing message definitions.

Related reference:

"Export SCA Import or Export from Broker SCA Definition wizard" on page 6380
The Export SCA Import or Export from Broker SCA Definition wizard exports an SCA import or export from a Broker SCA definition in a message set.

Generate WSDL wizard:

The Generate WSDL wizard creates a WSDL definition from a message set.

The following links provide further information in relation to the panels and fields that form the Generate WSDL wizard. Some panels only appear if certain conditions are met. These are marked as (optional).

Open the Generate WSDL wizard

To open the Generate WSDL wizard:

1. Switch to the Broker Application Development perspective.
2. In the Broker Development view, right-click the folder that contains the message set file from which you want to generate a Web service definition, and select **Generate** > **WSDL Definition**. This starts the Generate WSDL wizard.

Generate WSDL wizard

The following panels are shown by the Generate WSDL wizard:

- Select the action you want to perform
- Select a message set folder and destination directory
- Specify WSDL details
- Add operations to the WSDL definition
- Configure binding details
- Summary of tasks

Related tasks:

"Generating a WSDL definition from a message set" on page 2968
To ensure the highest interoperability of your Web services, use the document style of WSDL whenever possible. If rpc style WSDL is necessary, use literal encoding.

Generate WSDL wizard: Select the action you want to perform:

Use this panel to select how you want to generate the WSDL definition.

Generate WSDL wizard

The following panels are shown by the Generate WSDL wizard:

- Select the action you want to perform
- Select a message set folder and destination directory
- Specify WSDL details
- Add operations to the WSDL definition
- Configure binding details

- Summary of tasks

Panel properties

A choice of three options is presented.

Generate a new WSDL definition from existing message definitions
Select this option to generate a new WSDL definition from existing message definitions. This is the default option.

Export an existing WSDL definition to another directory in the workspace or file system.
Select this option to load the Export WSDL wizard.

Generate a new WSDL definition from existing message definitions using message categories (deprecated)
Select this option to generate a new WSDL definition using existing message definitions and message categories. This option is available to provide compatibility with previous releases of WebSphere Message Broker.

Related tasks:

"Generating a WSDL definition from a message set" on page 2968
To ensure the highest interoperability of your Web services, use the document style of WSDL whenever possible. If rpc style WSDL is necessary, use literal encoding.

Related reference:

"Generate WSDL wizard" on page 6382
The Generate WSDL wizard creates a WSDL definition from a message set.

Generate WSDL wizard: Select a message set folder and destination directory:

Use Generate WSDL wizard to select both the source of the WSDL definition and where you want the generated WSDL definition to be placed.

Generate WSDL wizard

The following panels are shown by the Generate WSDL wizard:

- Select the action you want to perform
- Select a message set folder and destination directory
- Specify WSDL details
- Add operations to the WSDL definition
- Configure binding details
- Summary of tasks

Panel properties

Select the message set folder from which to generate the WSDL definition:
Specify the message set folder from which to generate the WSDL definition.

Choose one of the following options to select the destination for the generated WSDL definition:

Create in a workspace directory
Select from your workspace directory the message set folder that will contain the generated WSDL definition.

Export to an external directory
Specify the address of the directory, outside your workspace, that you want to contain the generated WSDL definition.

Options
Specify the structure of the generated XML schema

Choose one of the following options to specify the structure of the generated XML schema:

Generate XML schema definitions with current directory structure
Generates the schema definition using the current directory structure; this is the default.

Generate XML schema definitions with flat structure
Generates the schema definition as a single level directory structure.

Related tasks:

"Generating a WSDL definition from a message set" on page 2968
To ensure the highest interoperability of your Web services, use the document style of WSDL whenever possible. If rpc style WSDL is necessary, use literal encoding.

Related reference:

"Generate WSDL wizard" on page 6382
The Generate WSDL wizard creates a WSDL definition from a message set.

Generate WSDL wizard: Specify WSDL details:

Use this panel to describe some details of the WSDL definition that you want to generate.

Generate WSDL wizard

The following panels are shown by the Generate WSDL wizard:

- Select the action you want to perform
- Select a message set folder and destination directory
- Specify WSDL details
- Add operations to the WSDL definition
- Configure binding details
- Summary of tasks

Panel properties

File Format
Select from:

- **Generate as a single WSDL file**
  The WSDL definition is written to a single file. This format is widely understood by external applications and SOAP toolkits.
- **Generate as a single WSDL file with all XML schema inlined**
  The WSDL definition is written to a single file with the XML added.
- **Generate as three WSDL files (one each for port type, service, and binding)**
  The WSDL definition is split into multiple files. This format offers better reuse of the component files.

WSDL Version
Select the required version of WSDL.

SOAP Version
Select the required version of SOAP

Style The style determines the format of the runtime SOAP messages described by the generated WSDL. The choices are:

- rpc
- document

WSDL Namespace
This must be a valid URI and becomes the target namespace for the WSDL definitions. This value has no particular significance outside of the WSDL definition itself and does not correspond to the namespace of SOAP messages described by the generated WSDL. A default value of http://tempuri.org/<message set name> is set.

RPC Namespace
This field is only enabled if you selected the Style as rpc. It is the namespace for the immediate children of your SOAP body. The value must be a valid URI. A default value of http://tempuri.org/<message set name> is set.

Definition Name
This is used in deriving the names of the WSDL file or files that are created. The default value is the name of your message set.

Documentation
Optional: This text is included as documentation for the PortType element on the generated WSDL. It has no implications for the SOAP messages that are described by the generated WSDL.

Related tasks:

"Generating a WSDL definition from a message set" on page 2968
To ensure the highest interoperability of your Web services, use the document style of WSDL whenever possible. If rpc style WSDL is necessary, use literal encoding.

Related reference:

"Generate WSDL wizard" on page 6382
The Generate WSDL wizard creates a WSDL definition from a message set.

Generate WSDL wizard: Add operations to the WSDL details:

Use this panel to define the operations that you want to add to the WSDL definition.

Generate WSDL wizard

The following panels are shown by the Generate WSDL wizard:

- Select the action you want to perform
- Select a message set folder and destination directory
- Specify WSDL details
- Add operations to the WSDL definition
- Configure binding details
- Summary of tasks

Panel properties

The panel is divided into two panes.

The top pane is read-only and displays a table that describes the operations that you have defined. The table has four columns with the following headings:

Operation
The name that you have given to the operation.

Input message
The name of the input message. This might be blank if the operation is a *Notification* type operation.

Output message
The name of the output message. This might be blank if no output message is specified for this operation.

Operation Type
The type of operation. Examples of operation type are:

Request-response

One-way

Solicit-response

Notification

The bottom pane is where you describe a new operation. The following fields describe the operation:

Name The name that you have given to the operation.

Operation Type
The type of operation. Examples of operation type are:

Request-response

One-way

Solicit-response

Notification

Input The name of the input message. This is omitted for a *Notification* operation.

Output
The name of the output message. This is omitted for a *One-way* operation.

Fault The name of the fault message. This is omitted for a *Notification* operation. Otherwise, you can specify one or more fault messages.

Related tasks:

"Generating a WSDL definition from a message set" on page 2968
To ensure the highest interoperability of your Web services, use the document style of WSDL whenever possible. If rpc style WSDL is necessary, use literal encoding.

Related reference:

"Generate WSDL wizard" on page 6382
The Generate WSDL wizard creates a WSDL definition from a message set.

Generate WSDL wizard: Configure binding details:

Use this panel to specify your SOAP/HTTP or SOAP/JMS binding details.

Generate WSDL wizard

The following panels are shown by the Generate WSDL wizard:

- Select the action you want to perform
- Select a message set folder and destination directory

- Specify WSDL details
- Add operations to the WSDL definition
- Configure binding details
- Summary of tasks

Panel properties

Service name
The Service Name is the value of the name attribute on the service element in the generated WSDL. The exact use of the name depends on products that later use the WSDL such as the SOAP toolkits and UDDI repositories. For example if you later use a SOAP toolkit to generate Java from your WSDL, the Service Name is likely to become the Java interface name.

Port name
This property is the name of a specific WSDL port for this service and would typically be derived from the Service Name. One convention would be to provide a Service Name of <xyz> Service and a Port Name of <xyz> Provider.

The Port Name is the value of the name attribute on the port element in the generated WSDL. The exact use of the name depends on products that later use the WSDL such as SOAP toolkits and UDDI repositories. For example if you use a SOAP toolkit to generate Java from your WSDL, the Port Name could become a Java class name.

A choice of two options is presented:

SOAP/HTTP
Select this option to generate a new WSDL definition using existing message definitions with an HTTP transport. This is the default option.

SOAP/JMS
Select this option to generate a new WSDL definition using existing message definitions with a JMS transport.

If you select **SOAP/HTTP**, the following additional properties are displayed:

SOAP action
This property defines the value for the HTTP SoapAction header. It is possible that an application will use the SoapAction as a mechanism for relating a SOAP message to an implementation method. This is often true with rpc-style WSDL.

If the WSDL definition is to contain multiple operations and they use different SOAP actions, you must add the unique SOAP action values to the WSDL after it has been generated. If all operations use the same SOAP action, specify the value here.

Port address
This property defines the address at which the service is made available. It must be a valid URL and it must include the port number, if it is different from the default HTTP port. An example of a port address is:
`http://localhost:9080/wassoap/servlet/router`

If you select **SOAP/JMS**, the following additional properties are displayed:

Destination style
The style in which you specify the destination name of the JMS message. The default value is `jndi`.

Destination name
The name of the destination of the JMS message. This property must be specified according to the `Destination style` property.

JMS provider name
Select a JMS vendor name from the list. When you select a name from the list, the `Initial context factory` property is updated automatically with the relevant Java class. The name must match the name of a configurable service that is defined for the broker to which you deploy the message flow.

Initial context factory

The starting point for a JNDI namespace. A JMS application uses the initial context to obtain and look up the connection factory and queue or topic objects for the JMS provider. When you select a JMS provider name from the list in `JMS provider name`, the `Initial context factory` property is updated automatically with the relevant Java class. The default value is `com.sun.jndi.fscontext.RefFSContextFactory`, which defines the file-based `Initial context factory` for the WebSphere MQ JMS provider.

JNDI connection factory
The name of the connection factory that is used. This name must exist in the bindings file. The `JNDI connection factory` is a JMS QueueConnectionFactory. Alternatively, you can specify the generic JMS ConnectionFactory.

JNDI URL
The JNDI URL for the JMS provider.

Delivery mode
This property controls the persistence mode used for a message. Valid values are:

- `Persistent`: the message survives if the JMS provider has a system failure.
- `Non Persistent`: the message is lost if the JMS provider has a system failure.

Request message lifetime
This property controls the length of time, in seconds, for which the output JMS message is kept. The default value, 0, is used to indicate that the message must not expire.

JMS request message priority

This property assigns relative importance to the message and can be used for message selection by a receiving web service.

Select a value between 0 (lowest priority) and 9 (highest priority). The default value is 4, which indicates medium priority. Priorities in the range 0 - 4 indicate typical delivery. Priorities in the range 5 - 9 indicate faster delivery.

Reply to Name
The name of the JMS destination to which the receiving application must send a reply message. For a reply message to be returned to this JMS destination, the JMS destination name must be known to the domain of the JMS provider that is used by the receiving client.

Specify JNDI parameters
Enter JNDI context parameters, to be included in the generated WSDL URI,

in this table as name-value pairs. If the `Use SOAP/JMS interoperability protocol` check box is cleared, the `JNDI parameters` table is disabled, and its values are not generated in the resultant WSDL.

Specify user parameters
Enter additional user parameters, to be included in the generated WSDL URI, in this table as name-value pairs.

Use SOAP/JMS interoperability protocol
This check box is selected by default. If this check box is selected, the generated SOAP/JMS WSDL is in the W3C format, otherwise it is IBM-style WSDL. If you clear this check box, the `JNDI parameters` table is disabled, and its values are not generated in the resultant WSDL.

Generate WSDL Definition wizard: Summary of tasks:

Generate WSDL Definition wizard: provides a summary of the actions that will occur on finalizing the wizard.

Generate WSDL wizard

The following panels are shown by the Generate WSDL wizard:

- Select the action you want to perform
- Select a message set folder and destination directory
- Specify WSDL details
- Add operations to the WSDL definition
- Configure binding details
- Summary of tasks

Summary information

This panel consists of two panes. The top pane displays a summary of the selections you have made and the bottom pane lists the message definition files generated.

The selected message set
The message set you selected on the Select a message set folder and destination directory panel.

The generated WSDL files will go into:
The destination directory you selected on the Select a message set folder and destination directory panel.

The version of WSDL to be generated
The version you selected on the Specify WSDL details panel.

The version of SOAP to be generated
The version you selected on the Specify WSDL details panel.

The selected style for WSDL generation:
The style you selected on the Specify WSDL details panel.

The WSDL namespace:
The namespace you selected on the Specify WSDL details panel.

If you selected `rpc` as the style there is an entry for **RPC namespace**.

The following bindings are selected:

SOAP over HTTP
See Configure binding details for further details.

SOAP over JMS
See Configure binding details for further details.

The following WSDL files will be generated:
The name of the generated file.

Related tasks:

"Generating a WSDL definition from a message set" on page 2968
To ensure the highest interoperability of your Web services, use the document style of WSDL whenever possible. If rpc style WSDL is necessary, use literal encoding.

Related reference:

"Generate WSDL wizard" on page 6382
The Generate WSDL wizard creates a WSDL definition from a message set.

Export WSDL wizard:

The Export WSDL wizard exports a WSDL definition from a message set.

The following links provide further information in relation to the panels and fields that form the Export WSDL wizard.

Export WSDL wizard

The following panels are shown by the Export WSDL wizard:

- Select the WSDL definition that you want to export
- Specify the export location

Related tasks:

"Generating a WSDL definition from a message set" on page 2968
To ensure the highest interoperability of your Web services, use the document style of WSDL whenever possible. If rpc style WSDL is necessary, use literal encoding.

Export WSDL wizard: Select the WSDL definition you want to export:

Use this panel of the Export WSDL wizard to select the WSDL definition that you want to export from a message set.

The following links provide further information in relation to the panels and fields that form the Export WSDL wizard.

Export WSDL wizard

The following panels are shown by the Export WSDL wizard:

- Select the WSDL definition that you want to export
- Specify the export location

Panel properties

The top pane of the panel shows a map of your workspace. Select the WSDL definition that you want to export.

Export file format
Choose one of the following options:

- **Export to a single WSDL file**

The WSDL definition is written to a single file. This format is widely understood by external applications and SOAP toolkits.

- **Export to a single WSDL file with all XML schema inlined**

  The WSDL definition is written to a single file. This format is widely understood by external applications and SOAP toolkits.

- **Export to three WSDL files (one each for port type, service, and binding)**

  The WSDL definition is split into multiple files. This format offers better reuse of the component files.

- **Export based on the existing file structure**

  The WSDL definition is written to a single file. This format is widely understood by external applications and SOAP toolkits.

WSDL definition name
Select a name for the exported WSDL.

Related tasks:

"Generating a WSDL definition from a message set" on page 2968
To ensure the highest interoperability of your Web services, use the document style of WSDL whenever possible. If rpc style WSDL is necessary, use literal encoding.

Export WSDL wizard: Specify the export location:

Use this panel of the Export WSDL wizard to specify the location for the WSDL definition that you want to export from a message set.

The following links provide further information in relation to the panels and fields that form the Export WSDL wizard.

Export WSDL wizard

The following panels are shown by the Export WSDL wizard:

- Select the WSDL definition that you want to export
- Specify the export location

Panel properties

Choose one of the following options:

Export to a workspace directory
The structure of the workspace is displayed. Click the folder that you want the WSDL definition to be exported to.

Export to an external directory
Specify the name of the external directory that you want the WSDL definition to be exported to.

Select the **Overwrite existing files without warning** check box if you do not want to be warned that a file with the name that you specified is being overwritten. By default, the check box is cleared; if a file exists with the same name as the name that you have selected, you are prompted to confirm whether you want this file to be overwritten by the file that you are exporting.

Related tasks:

"Generating a WSDL definition from a message set" on page 2968
To ensure the highest interoperability of your Web services, use the document style of WSDL whenever possible. If rpc style WSDL is necessary, use literal encoding.

Configure New Web Service Usage wizard:

This provides additional reference information in relation to the Configure New Web Service Usage wizard.

You can launch this wizard by dragging deployable WSDL onto the message flow canvas.

The following links provide further information in relation to the panels and fields that form the Configure New Web Service Usage wizard.

Configure New Web Service Usage wizard

List of panels:

- Configure web service usage
- File generation details

Related tasks:

"Generating a WSDL definition from a message set" on page 2968
To ensure the highest interoperability of your Web services, use the document style of WSDL whenever possible. If rpc style WSDL is necessary, use literal encoding.

Related reference:

"Generate WSDL wizard" on page 6382
The Generate WSDL wizard creates a WSDL definition from a message set.

Configure New Web Service Usage wizard: Configure Web service usage details:

Use this panel of the Configure New Web Service Usage wizard to configure a new Web service.

The following links provide further information in relation to the panels and fields that form the Configure New Web Service Usage wizard.

Configure New Web Service Usage wizard

List of panels:

- Configure web service usage
- File generation details

Panel properties

Web service usage

Select from:

- **Expose message flow as Web service**

  The message flow is exposed as a Web service to its clients.
- **Invoke Web service from message flow**

  The Web service is invoked from the message flow.

Web service parameters

Configure the WSDL-related fields:

- **Port type**

  `Port type` must be specified, and lists all the port types defined in the WSDL document.

By default, the drop down is populated with all the port types from the WSDL, in the order in which they appear in the WSDL file. The initially selected port type is the first port type that has at least one `http` binding associated with it.

- **Binding**

  `Binding` must be specified and lists all SOAP bindings with HTTP transport, associated with the selected port type.

  Bindings related to the selected port type are populated in the order in which they appear in the WSDL file. The initially selected binding is the one that has at least one port and one operation associated with it; if there is no such binding, the first binding with at least one port is selected.

  If no binding has ports associated with it, the first binding in the list is selected.

- **Service port**

  Lists all WSDL ports that point to the selected binding.

- **Binding operations**

  Lists all operations defined by the selected port type. Note, that only those operations implemented by the selected binding are selected by default.

  For every selected operation, the subflow generation process produces an output terminal, in the generated subflow.

  If you select an operation, that is not implemented by the selected binding, you receive a warning message; however, you can continue with the selection.

Related tasks:

"Generating a WSDL definition from a message set" on page 2968
To ensure the highest interoperability of your Web services, use the document style of WSDL whenever possible. If rpc style WSDL is necessary, use literal encoding.

Related reference:

"Generate WSDL wizard" on page 6382
The Generate WSDL wizard creates a WSDL definition from a message set.

Configure New Web Service Usage wizard: File generation details:

Use the Configure New Web Service Usage wizard to specify file generation details.

Configure New Web Service Usage wizard

List of panels:

- Configure web service usage
- File generation details

Panel properties

Flow Generation Details

Only one file is generated, namely the subflow. The subflow name is constructed as follows:

	Format of the generated subflow name
Request operation	`OperationName_WSDLFileName_MainFlow.msgflow`

	Format of the generated subflow name
Extract operation	`WSDLFileName_MainFlow.msgflow`

This page of the wizard lists the name of the file to be generated together with its location.

Typically, this file represents the subflow that is about to be generated. The default subflow name is prefixed by the name of the selected WSDL file, however, you have the option to change the name.

If the file to be generated already exists in the workspace, a warning is issued and the **Finish** button is no longer enabled.

You either have to change the name of the file, or select the **Overwrite existing file** check box.

Node type to be used by the Web service flows

Select from:

- **SOAP nodes**

  Select this option to use the SOAP domain and the SOAP nodes. This is the default option.

  Using SOAP nodes is WSDL driven and allows you to take advantage of various WS_* standards; for example WS_Security and WS_Addressing.

  If the message set does not support the SOAP domain you receive a warning.

- **HTTP nodes**

  Select this option if you want to use HTTP nodes rather than SOAP nodes.

  You can select this option only if the message set supports the XMLNSC, MRM, or XMLNS domains.

  If you select **HTTP nodes**, you see a message explaining the advantages of the SOAP nodes together with a suggestion that you import WSDL files.

  If you use the `ImportFiles` folder as your source, you can only select HTTP node generation.

Details

This pane appears if any additional warnings about the subflow that is generated apply. Possible warnings are as follows:

- When Service Definition is not found in the WSDL file, the URL property is not set on the node.
- You have selected one or more operations that are not implemented by the selected binding.
- When message domain is MRM, but XML wire format not found, message format property is not set on the HTTPInput or Request node.

Related tasks:

"Generating a WSDL definition from a message set" on page 2968
To ensure the highest interoperability of your Web services, use the document style of WSDL whenever possible. If rpc style WSDL is necessary, use literal encoding.

Related reference:

"Generate WSDL wizard" on page 6382
The Generate WSDL wizard creates a WSDL definition from a message set.

Publish/subscribe

Use the reference information in this section to help you develop publish/subscribe applications.

Publish/subscribe reference information is available for the following topics:
- “Special characters in topics”
- “Topic semantics and usage” on page 6396
- “MQRFH2 header” on page 6397
- “Command messages” on page 6403

Special characters in topics

A topic can contain any character in the Unicode character set, but some characters have a special meaning.

The following three characters have a special meaning:
The topic level separator "/".
The multilevel wildcard "#".
The single-level wildcard "+".

The topic level separator is used to introduce structure into the topic, and can therefore be specified within the topic for that purpose.

The multilevel wildcard and single-level wildcard can be used for subscriptions, but they cannot be used within a topic by the publisher of a message.

However, if a publisher uses the characters "+" or "#" together with other characters in any topic level within a topic, these characters are not treated as wildcards, and they do not have any special meaning.

The topic level separator:
The topic level separator character "/" is used to provide a hierarchical structure to the topic space. It must be used by applications to separate levels within a topic tree. The use of the topic level separator is significant when the two wildcard characters are encountered in topics specified by subscribers.

Topic hierarchy is important in the administration of access control.

The multilevel wildcard:
The multilevel wildcard character "#" is used to match any number of levels within a topic. For example, using the example topic tree shown above, if you subscribe to "USA/Alaska/#", you receive messages on topics "USA/Alaska" and "USA/Alaska/Juneau".

The multilevel wildcard can represent zero or more levels. Therefore, "USA/#" can also match the singular "USA", where # represents zero levels. The topic level separator is meaningless in this context, because there are no levels to separate.

The multilevel wildcard can be specified only on its own or next to the topic level separator character. Therefore, "#" and "USA/#" are valid topics where the "#" character is treated as a wildcard. However, although "USA#" is also a valid topic, the "#" character is not regarded as a wildcard and does not have any special meaning. See “When wildcards are not wild” on page 6396 for more information.

The single-level wildcard:

The single-level wildcard character "+" matches one, and only one, topic level. For example, "USA/+" matches "USA/Alabama", but not "USA/Alabama/Auburn". Also, because the single-level wildcard matches only a single level, "USA/+" does not match "USA".

The single-level wildcard can be used at any level in the topic tree, and in conjunction with the multilevel wildcard. The single-level wildcard must be specified next to the topic level separator, except when it is specified on its own. Therefore, "+" and "USA/+" are valid topics where the "+" character is treated as a wildcard. However, although "USA+" is also a valid topic, the "+" character is not regarded as a wildcard and does not have any special meaning. See "When wildcards are not wild" for more information.

When wildcards are not wild:
The wildcard characters "+" and "#" have no special meaning when they are mixed with other characters (including themselves) in a topic level.

This means that topics that contain "+" or "#" together with other characters in a topic level can be published.

For example, consider the following two topics:

1. level0/level1/+/level4/#
2. level0/level1/#+/level4/level#

In the first example, the characters "+" and "#" are treated as wildcards and are therefore not valid in a topic that is to be published.

In the second example, the characters "+" and "#" are not treated as wildcards and therefore the topic can be both published and subscribed to.

Related reference:

"Topic semantics and usage"

Topic semantics and usage

When you build an application, the design of the topic tree should take into account the following principles of topic name syntax and semantics:

- Topic names are case sensitive.

  For example, "ACCOUNTS" and "Accounts" are two different topics.
- Topic names can include the space character.

  For example, "Accounts payable" is a valid topic.
- A leading "/" creates a distinct topic.

  For example, "/USA" is different from "USA" and "/USA' matches "+/+" and "/+", but not "+".
- A topic name that contains '//' is not a valid name. An attempt to subscribe to a topic with such a name causes an error.
- Do not include the null character (Unicode \x0000) in any topic.
- The wildcard characters "+" and "#" are not treated as wild cards if they are mixed with any other characters (including themselves but excluding the topic level separator "/") within a topic level.

The following principles apply to the construction and content of a topic tree:

- There is no limit to the number of levels in a topic tree.
- There is no limit to the length of the name of a level in a topic tree.

- There can be any number of "root" nodes; that is, there can be any number of topic trees. These are defined below the root "", which is the root of all root nodes. It is referred to as "topicRoot", although there is no corresponding topic name. Applications cannot publish or subscribe to this virtual root.
- The topic trees with roots of "$SYS" and "$ISYS" are reserved for use by WebSphere Message Broker only.

Related reference:

"Special characters in topics" on page 6395
A topic can contain any character in the Unicode character set, but some characters have a special meaning.

MQRFH2 header

The MQRFH2 header is used to pass messages to and from a message broker that belongs to WebSphere Message Broker.

In a message, the MQRFH2 header follows the WebSphere MQ message descriptor (MQMD) and precedes the message body, if present. The MQRFH2 header can be parsed by either the MQRFH2 parser or the MQRFH2C parser.

Other headers, such as the IMS/ESA® or CICS bridge headers, are allowed either before or after the MQRFH2 header, but before the message body.

If you are using the Message Queuing Interface (MQI) to write application programs you need to understand the structure and content of the MQRFH2 header.

For more information, refer to:
- "MQRFH2 structure"
- "Message service folders" on page 6401

Multiple MQRFH2 headers:
A message can have more than one MQRFH2 header.

For example, if an application forwards a message, including its header, to another application, a second MQRFH2 header precedes the header in the message being forwarded.

- Attributes that describe the body of the message, such as the domain, set, type, and format, or the character set ID and encoding, are taken from the *last* MQRFH2 header, which is immediately in front of the body of the message.
- Anything else, such as the topic for a publish/subscribe message, is taken from the *first* MQRFH2 header.

Related reference:

"The MQRFH2 and MQRFH2C parsers" on page 4253
The MQRFH2 header can be parsed using either the MQRFH2 parser or the MQRFH2C compact parser.

"MQInput node" on page 4594
Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

MQRFH2 structure:
The MQRFH2 header contains information about the structure of a message, and its intended consumers, to enable a message broker to process the message and deliver or publish the message to those consumers.

The value 'MQHRF2 ' should be put in the **Format** field of the preceding header (usually the MQMD). The constant *MQFMT_RF_HEADER_2* is defined with this value.

For the C programming language, the constant *MQFMT_RF_HEADER_2_ARRAY* is also defined. This constant has the same value as *MQFMT_RF_HEADER_2*, but it is an array of characters, not a character string.

The character set and encoding of the fields in the MQRFH2 header are as follows:

- Fields other than **NameValueData** are in the character set and encoding defined by the fields **CodedCharSetId** and **Encoding** in the header structure that precedes the MQRFH2 header, or by the same fields in the MQMD structure if the MQRFH2 header is at the start of the application message data. The character set should be one that has single-byte characters for the characters that are valid in queue names.
- **NameValueData** is in the character set defined by the **NameValueCCSID** field. Note that not all Unicode character sets are valid for **NameValueCCSID**; see the description of **NameValueCCSID** for details.

  Some character sets have a representation that is dependent on the encoding. If **NameValueCCSID** defines one of these character sets, **NameValueData** must be in the same encoding as the other fields in the MQRFH2 header.
- The user data (if any) that follows **NameValueData** can be in any supported character set (single-byte, double-byte, or multi-byte), and in any supported encoding.

The MQRFH2 header contains the following fields:

Field Name	Description	Details
StrucId	Structure identifier	The value must be `MQRFH_STRUC_ID`, which is the identifier for the rules and formatting header structure,. WebSphere Message Broker populates this field for you if you are constructing an MQRFH2 header in a message flow. For the C programming language, the constant `MQRFH_STRUC_ID_ARRAY` is also defined; this constant has the same value as `MQRFH_STRUC_ID`, but it is an array of characters, not a character string.
Version	Structure version number	The value must be `MQRFH_VERSION_2`, which is the Version-2 rules and formatting header structure.

Field Name	Description	Details
Struclength	Total length of MQRFH2 (including **NameValueData**)	The initial value of this field is MQRFH_STRUC_LENGTH_FIXED_2, which is the length of the fixed part of the MQRFH2 header structure. This is the length in bytes of the MQRFH2 header structure, including any **NameValueLength** and **NameValueData** fields at the end of the structure. WebSphere Message Broker populates this field for you if you are constructing an MQRFH2 header in a message flow. There might be more than one pair of these fields at the end of the structure, in the sequence: length1, data1, length2, data2, ... The length of any user data that follows the last **NameValueData** field at the end of the structure is not included in **StrucLength**. **Note:** If **Struclength** is not a multiple of four, problems might occur with the data conversion of user data in some operating system environments.
Encoding	Numeric encoding of data that follows **NameValueData**	The initial value of this field is MQENC_NATIVE. This field specifies how numeric values in any data that follows the last **NameValueData** field are represented. This applies to binary integer data, packed decimal integer data and floating-point data.
CodedCharSetId	Character set identifier of data that follows **NameValueData**	The initial value of this field is MQCCSI_INHERIT, which means that the character set identifier is the same as that of the current structure. This field identifies the coded character set for any character strings in the data that follows the last **NameValueData** field.
Format	Format name of data that follows **NameValueData**	The initial value of this field is **MQFMT_NONE**. This field specifies the format name of any data that follows the last **NameValueData** field. The name should be padded with blanks to the length of the field. **Note:** Do not use a null character to terminate the name before the end of the field; the queue manager does not change to a blank character the null character, or any characters that follow the null character, in the MQRFH2 header. **Note:** Do not specify a name with leading or embedded blank characters.
Flags	Flags	The initial value of this field is MQRFH_NONE, which means that there are no flags.

Field Name	Description	Details
NameValueCCSID	Character set identifier of **NameValueData**	The initial value of this field is 1208, which means that the UTF-8 coded character set is used. This field identifies the coded character set for data in the **NameValueData** field. This is different from the character set for other character strings in the MQRFH2 header structure, and might be different from the character set for any character data that follows the last **NameValueData** field. **NameValueCCSID** must have one of the following values: **1200**: UCS-2 open-ended **1208**: UTF-8 **13488**: UCS-2 2.0 subset **17584**: UCS-2 2.1 subset (includes the euro symbol €) For the UCS-2 character sets, the encoding (byte order) of the **NameValueData** field must be the same as the encoding of the other fields in the MQRFH2 header structure. **Note:** Surrogate characters (X'D800' thru X'DFFF') are not supported.
The following two fields are optional, but if present they must occur as a pair. They can be repeated as a pair as many times as required. If these fields occur more than once, they must occur in the sequence ..... length1, data1, length2, data2, ...		
NameValueLength	Length of **NameValueData**	This field specifies the length, in bytes, of the **NameValueData** field that follows this field. WebSphere Message Broker populates this field for you if you are constructing an MQRFH2 header in a message flow. **Note:** If **NameValueLength** is not a multiple of four, there might be a problem with the conversion of the data that follows the **NameValueData** field.
NameValueData	This is a variable-length character string containing data that is encoded using an XML-like structure	The length, in bytes, of this string is given by the **NameValueLength** field that precedes this **NameValueData** field. To avoid the problem described in the note accompanying the description of the **NameValueLength** field, either extend this field with blanks so that its length is a multiple of four, or terminate the field with a null character.

C programming language definition

The following structure is defined in the cmqc.h header file that is supplied with WebSphere MQ. The constants that are used within the **NameValueData** field are defined in the BipRfc.h header file that is supplied with WebSphere Message Broker.

```
typedef struct tagMQRFH2 {
  MQCHAR4  StrucId;         /*  Structure identifier                     */
  MQLONG   Version;         /*  Structure version number                 */
  MQLONG   StrucLength;     /*  Total length of MQRFH2 including
                                NameValueData                            */
  MQLONG   Encoding;        /*  Numeric encoding of data that follows
                                NameValueData                            */
  MQLONG   CodedCharSetId;  /*  Character set identifier of data that
                                follows NameValueData                    */
  MQCHAR8  Format;          /*  Format name of data that follows
                                NameValueData                            */
  MQLONG   Flags;           /*  Flags                                    */
  MQLONG   NameValueCCSID;  /*  Character set identifier of NameValueData   */
 } MQRFH2;
```

Message service folders:

A number of folders are defined for use by WebSphere MQ products.

<mcd>
: Message content descriptor

<psc> Publish/subscribe command

<pscr>
: Publish/subscribe command response

<usr> Application (user) defined properties

<jms> Java Messaging Service

Each folder is contained in a separate **NameValueData** field, each of which is preceded by a **NameValueLength** field.

Independent software vendors can choose other names for their folders. However, you can prefix your chosen folder name with their internet domain name to avoid naming conflicts and problems. For example, a vendor with domain name ourcompany.com might name its folders:

```
com.ourcompany.xxx  or  com.ourcompany.ourData
```

The mcd folder:

The <mcd> folder can contain the following elements that describe the structure of the message data in a WebSphere MQ message. They are all character strings, and are case sensitive.

<Msd> Message service domain

Valid values are:

mrm The message is parsed by the MRM domain.

xmlnsc
: The message is XML and is parsed by the XMLNSC domain.

xmlns The message is XML and is parsed by the XMLNS domain.

xml The message is XML and is parsed by the XML domain.

mime The message uses the MIME standard and is parsed by the MIME domain.

idoc The message is an SAP ALE IDoc from the WebSphere MQ Link for R/3, and is parsed by the IDOC domain.

none The message is treated as an opaque BLOB, and delivered to the recipient without modification.

See "Parsers" on page 1072 for a description of each domain.

<Set> The name of the message set that contains the definition of the message.

If <Msd> is *mrm* or *idoc*, you must use the <Set> element to supply the name of the message set.

If <Msd> is *xmlnsc*, and validation is enabled in the message flow, you must use the <Set> element to supply the name of the message set.

<Type> The name of the message type, in the specified message set, to which this message corresponds. The format of a simple message type is {namespace-uri}:name where name is the name of the message.

The format {namespace-uri}name (that is, with no colon) is also valid to maintain compatibility with previous versions of WebSphere Message Broker.

If <Msd> is *mrm* or *idoc*, you must use the <Type> element to supply the name of the message definition.

<Fmt> The name of a physical format in the specified message set.

If <Msd> is *mrm* or *idoc*, you must use the <Fmt> element to supply the name of the physical format in the message set.

If the message tree contains an MQRFH2 header, any node that modifies the message tree performs the following steps before propagating the message to the next node:

- If there is no <mcd> folder, one is added, and the Msd, Set, Type, and Format fields are populated with values that match the current domain and the current setting in the Properties folder.
- If there is an <mcd> folder, the Msd, Set, Type, and Format fields are overwritten with values that match the current domain and the current settings in the Properties folder.

Tip: If you need to propagate an MQRFH2 header without an <mcd> folder, you can remove the header by using the following ESQL code with a Compute node:

```
CALL CopyEntireMessage();
DELETE FIELD OutputRoot.MQRFH2.mcd;
PROPAGATE FINALIZE NONE; -- Propagate the message without updating the headers
RETURN FALSE; -- Ensure that the message is not propagated again.
```

The psc folder:
The <psc> folder is used to convey publish/subscribe command messages to the broker.

Only one psc folder is allowed in the **NameValueData** field.

See "Command messages" on page 6403 for full details.

The pscr folder:
The <pscr> folder is used to contain information from the broker, in response to publish/subscribe command messages.

Only one pscr folder is present in a response message.

See “Broker Response message” on page 6407 for full details.

The broker ignores this folder in messages that it receives from publish/subscribe applications.

The usr folder:
The content model of the <usr> folder has the following characteristics.

- Any valid XML name that does not contain a colon can be used as an element name.
- Only simple elements, not groups, are allowed.
- All elements take the default type of string.
- All elements are optional, but must not occur more than once in a folder.
- An MQRFH2 instance can contain, at most, one <usr> folder.

The jms folder:
The content model of the <jms> folder contains the following MQRFH2 JMS fields:

- Dst - represents the JMSDestination header field.
- Dlv - represents the JMSDeliveryMode header field.
- Exp - represents the JMSExpiration header field.
- Pri - represents the JMSPriority header field.
- Tms - represents the JMSTimestamp header field.
- Cid - represents the JMSCorrelationID header field.
- Rto - represents the JMSReplyTo header field.

See “JMS message structure” on page 1688 for more information about the content of JMS messages.

Related concepts:

“Parsers” on page 1072
A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from the internal message tree representation.

“Message modeling” on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Related reference:

“JMS message structure” on page 1688
JMS messages have a defined structure that includes headers and payloads.

Command messages

The following command messages can be sent to WebSphere Message Broker in a publish/subscribe application:

- “Delete Publication message” on page 6404
- “Publish message” on page 6405

If you are using the Message Queue Interface (MQI) to write applications that use the publish/subscribe model, you need to understand these messages and the Broker Response message. Refer to:

- “Broker Response message” on page 6407

The commands are contained in a <psc> folder in the **NameValueData** field of the MQRFH2 header.

The message that can be sent by a broker in response to a command message is contained in a <pscr> folder.

Refer to "Message service folders" on page 6401 for details about the message service folders.

The descriptions of each command list the properties that can be contained in a folder. Unless otherwise specified, the properties are optional and can occur no more than once.

Names of properties are shown as <Command>.

Values must be in string format, for example: Publish.

A string constant representing the value of a property is shown in parentheses, for example: (MQPSC_PUBLISH).

String constants are defined in the header file BipRfc.h which is supplied with WebSphere Message Broker.

Delete Publication message:
The *Delete Publication* command message is sent to a broker from a publisher, or from another broker, to tell the broker to delete any retained publications for the specified topics.

This message is sent to the input queue of a message flow that contains a *Publication* node. You must have the authority to put a message onto this queue, and to publish on the topic, or topics, that are specified in the message.

The input queue should be the queue that the original publication was sent to.

If you have the authority for some, but not all, of the topics that are specified in the *Delete Publication* command message, only those topics are deleted. A *Broker Response* message indicates which topics are not deleted.

Similarly, if a **Publish** command contains more than one topic, a **Delete Publication** command matching some, but not all, of those topics deletes only the publications for the topics that are specified in the *Delete Publication* command.

Properties:

<Command> (*MQPSC_COMMAND*)
The value is DeletePub(*MQPSC_DELETE_PUBLICATION*).

This property must be specified.

<Topic> (*MQPSC_TOPIC*)
The value is a string that contains a topic for which retained publications are to be deleted. Wildcard characters can be included in the string to delete publications on more than one topic.

This property must be specified; it can be repeated for as many topics as needed.

<DelOpt> (*MQPSC_DELETE_OPTION*)
The delete options property can take one of the following values:

Local (*MQPSC_LOCAL*)

All retained publications for the specified topics are deleted at the local broker (that is, the broker to which this message is sent), whether they were published with the `Local` option or not.

Publications at other brokers are not affected.

None (*MQPSC_NONE*)

All options take their default values. This has the same effect as omitting the `DelOpt` property. If other options are specified at the same time, `None` is ignored.

The default if this property is omitted is that all retained publications for the specified topics are deleted at all brokers in the network, regardless of whether they were published with the `Local` option.

Example:
Here is an example of **NameValueData** for a **Delete Publication** command message. This is used by the sample application to delete, at the local broker, the retained publication that contains the latest score in the match between Team1 and Team2.

```
<psc>
 <Command>DeletePub</Command>
 <Topic>Sport/Soccer/State/LatestScore/Team1 Team2</Topic>
 <DelOpt>Local</DelOpt>
</psc>
```

Publish message:
The *Publish* command message is sent from a publisher to a broker, or from a broker to a subscriber, to publish information on a specified topic or topics.

This message is sent to the input queue of a message flow that contains a Publication node. Authority to put a message onto this queue, and to publish on the specified topic or topics, is necessary.

If the user has authority on some, but not all, topics, only those topics are published; a warning response indicates which topics are not published.

If a subscriber has any matching subscriptions, the broker forwards the Publish message to the subscriber queues defined in the corresponding **Register Subscriber** command messages.

See "Broker Response message" on page 6407 for details of the message descriptor (MQMD) parameters needed when sending a command message to the broker, and used when a broker forwards a publication to a subscriber.

The broker forwards the Publish message to other brokers in the network that have matching subscriptions, unless it is a local publication.

Publication data, if any, is included in the body of the message. The data can be described in an `<mcd>` folder in the **NameValueData** field of the MQRFH2 header.

Properties:

<Command> (*MQPSC_COMMAND*)
The value is `Publish`(*MQPSC_PUBLISH*).

This property must be specified.

<Topic> (*MQPSC_TOPIC*)
The value is a string that contains a topic that categorizes this publication. No wildcard characters are allowed.

This property must be specified, and can optionally be repeated for as many topics as needed.

<SubPoint> (*MQPSC_SUBSCRIPTION_POINT*)
The subscription point on which the publication is published.

This property should not be included in a publication message sent to the broker but is added automatically to publication messages by the broker before those messages are sent to any appropriate subscribers. The value of the `<SubPoint>` property is the value of the Subscription Point attribute of the Publication node that is handling the publishing.

<PubOpt> (*MQPSC_PUBLICATION_OPTION*)
The publication options property can take the following values:

RetainPub
(*MQPSC_RETAIN_PUB*)

The broker is to retain a copy of the publication. If this option is not set, the publication is deleted as soon as the broker has sent the publication to all its current subscribers.

IsRetainedPub
(*MQPSC_IS_RETAINED_PUB*)

(Can only be set by a broker.) This publication has been retained by the broker. The broker sets this option to notify a subscriber that this publication was published earlier and has been retained, provided that the subscription has been registered with the `InformIfRetained` option. It is set only in response to a **Register Subscriber** or **Request Update** command message. Retained publications that are sent directly to subscribers do not have this option set.

Local
(*MQPSC_LOCAL*)

This option tells the broker that this publication should not be sent to other brokers. All subscribers that registered at this broker receive this publication if they have matching subscriptions.

OtherSubsOnly
(*MQPSC_OTHER_SUBS_ONLY*)

This option allows simpler processing of conference-type applications, where a publisher is also a subscriber to the same topic. It tells the broker not to send the publication to the publisher's subscriber queue even if it has a matching subscription. The publisher's subscriber queue consists of its `QMgrName`, `QName`, and optional `CorrelId`, as described later in this section.

CorrelAsId
(*MQPSC_CORREL_ID_AS_IDENTITY*)

The `CorrelId` in the MQMD (which must not be zero) is part of the publisher's subscriber queue, in applications where the publisher is also a subscriber.

None (*MQPSC_NONE*)

All options take their default values. This has the same effect as omitting the publication options property. If other options are specified at the same time, `None` is ignored.

You can have more than one publication option by introducing additional `<PubOpt>` elements.

The default, if this property is omitted, is that no publication options are set.

<PubTime> (*MQPSC_PUBLISH_TIMESTAMP***)**
The value is an optional publication timestamp set by the publisher. It is 16 characters long with format:

```
YYYYMMDDHHMMSSTH
```

using Universal Time. This information is not checked by the broker before being sent to the subscribers.

<SeqNum> (*MQPSC_SEQUENCE_NUMBER***)**
The value is an optional sequence number set by the publisher.

It should be incremented by 1 with each publication. However, this is not checked by the broker, which merely transmits this information to subscribers.

If publications on the same topic are published to different interconnected brokers, it is the responsibility of the publishers to ensure that sequence numbers, if used, are meaningful.

<QMgrName> (*MQPSC_Q_MGR_NAME***)**
The value is a string containing the name of the queue manager for the publisher's subscriber queue, in applications where the publisher is also a subscriber (see `OtherSubsOnly`).

If this property is omitted, the default is the `ReplyToQMgr` name in the message descriptor (MQMD). If the resulting name is blank, it defaults to the name of the broker's queue manager.

<QName> (*MQPSC_Q_NAME***)**
The value is a string containing the name of the publisher's subscriber queue, in applications where the publisher is also a subscriber (see `OtherSubsOnly`).

If this property is omitted, the default is the `ReplyToQ` name in the message descriptor (MQMD), which must not be blank if `OtherSubsOnly` is set.

Example:
Here are some examples of *NameValueData* for a **Publish** command message.

The first example is for a publication sent by the match simulator in the sample application to indicate that a match has started.

```
<psc>
 <Command>Publish</Command>
 <Topic>Sport/Soccer/Event/MatchStarted</Topic>
</psc>
```

The second example is for a retained publication. The latest score in the match between Team1 and Team2 is published.

```
<psc>
 <Command>Publish</Command>
 <PubOpt>RetainPub</PubOpt>
 <Topic>Sport/Soccer/State/LatestScore/Team1 Team2</Topic>
</psc>
```

Broker Response message:

A *Broker Response* message is sent from a broker to the ReplyToQ of a publisher or a subscriber, to indicate the success or failure of a command message received by the broker if the command message descriptor specified that a response is required.

The response message is contained within the NameValueData field of the MQRFH2 header, in a <pscr> folder.

In the case of a warning or error, the response message contains the <psc> folder from the command message as well as the <pscr> folder. The message data, if any, is not contained in the broker response message. In the case of an error, none of the message that caused an error has been processed; in the case of a warning, some of the message might have been processed successfully.

If there is a failure sending a response:

- For publication messages, the broker tries to send the response to the WebSphere MQ dead-letter queue if the MQPUT fails. This allows the publication to be sent to subscribers even if the response cannot be sent back to the publisher.
- For other messages, or if the publication response cannot be sent to the dead-letter queue, an error is logged and the command message is normally rolled back. Whether this happens depends on how the MQInput node has been configured.

Properties:

<Completion> (*MQPSCR_COMPLETION***)**
The completion code, which can take one of three values:
ok Command completed successfully
warning
Command completed but with warning
error Command failed

<Response> (*MQPSCR_RESPONSE***)**
The response to a command message, if that command produced a completion code of warning or error. It contains a <Reason> property, and might contain other properties that indicate the cause of the warning or error.

In the case of one or more errors, there is only one response folder, indicating the cause of the first error only. In the case of one or more warnings, there is a response folder for each warning.

<Reason> (*MQPSCR_REASON***)**
The reason code qualifying the completion code, if the completion code is a warning or error. It is set to one of the error codes listed later in this section. The <Reason> property is contained within a <Response> folder. The reason code can be followed by any valid property from the <psc> folder (for example, a topic name), indicating the cause of the error or warning.

Examples:
Here are some examples of NameValueData in a **Broker Response** message. A successful response might be the following:

```
<pscr>
  <Completion>ok</Completion>
</pscr>
```

Here is an example of a failure response; the failure is a filter error. The first NameValueData string contains the response; the second contains the original command.

```
<pscr>
  <Completion>error</Completion>
  <Response>
    <Reason>3150</Reason>
  </Reponse>
</pscr>

<psc>
 ...
 command message (to which
 the broker is responding)
 ...
</psc>
```

Here is an example of a warning response (due to unauthorized topics). The first NameValueData string contains the response; the second NameValueData string contains the original command.

```
<pscr>
  <Completion>warning</Completion>
  <Response>
    <Reason>3081</Reason>
    <Topic>topic1</Topic>
  </Reponse>
  <Response>
    <Reason>3081</Reason>
    <Topic>topic2</Topic>
  </Reponse>
</pscr>

<psc>
 ...
 command message (to which
 the broker is responding)
 ...
</psc>
```

Content based filtering

Content based filtering allows a subscriber to filter messages based on their content.

This section contains the following reference information:

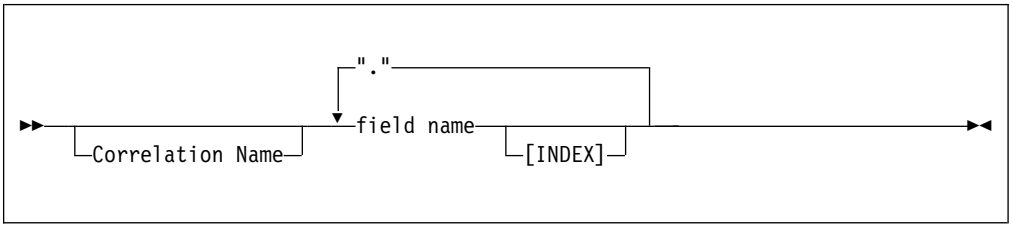
- "Using filters in content-based filtering"
- "Content based filtering examples" on page 6410

For a description of the ESQL language, see "ESQL reference" on page 5019.

Using filters in content-based filtering:

Content based filters are specified as ESQL expressions.

Field references:



The field references that can be used in filters for content based filtering form a subset of those supported by the Filter node. As with the Filter node, to reference a field in a filter, you must specify a path. Each element of the path consists of a, possibly indexed, field name.

The syntax of a field reference is shown above, where `field name` and `Correlation Name` are identifiers. These identifiers represent all messages as a hierarchical syntax element tree. Each path identifies a route through that tree, which leads to a particular syntax element, starting from one of the predefined correlation names that refer to fixed points that every message has. The following correlation names are supported for content based filtering:

Correlation name	Purpose
Root	Identifies the root of a published message.
Properties	Identifies the portion of the message in which the standard properties of a message lie.
Body	Identifies the last child of the root of the message, which is usually, but not always, the application data that follows any headers.

Here are some examples of field references, together with their meanings:

Field reference	Purpose
Body.Person.Address	Refers to the first Address field in the Person entity in the body of the message.
Properties.Topic	Refers to the "Topic" field in the standard properties of a message.
Root.MQMD.UserIdentifier	Refers to the `UserIdentifier` field in the MQMD of the message.

Note that path elements of `"*"` and the array index `"LAST"` are not supported in the filters.

Related concepts:

“Content based filtering” on page 6409
Content based filtering allows a subscriber to filter messages based on their content.

Related information:

“Content based filtering examples”
Some examples of content based filtering.

Content based filtering examples:

Some examples of content based filtering.

Content Filter	Explanation
Body.Person.Salary>10000	Filtering against an integer literal
`"Body.Person.Address"[1]NOT LIKE'Blen% 'AND"Body.Person.Salary">15000`	A more complex filter. Note that field identifiers can optionally be surrounded by double quotation marks.

Content Filter	Explanation
Body.Date1='2000-02-14'	Filtering against a date. The date is matched as a string and care must be taken with its layout (see below).
Body.Person.ApprovalFlag	Filtering against a Boolean field.
Body.Person.Salary+Person.Bonus>Body.Person.Limit	An arithmetic filter.
Properties.Topic='employees/marketing'	Filtering on a message property.
Root.MQMD.UserIdentifier='Blair'	Filtering on a message attribute.
Body.Person.HourlyRate = 10.24	Filtering against a float literal
Body.Planet.DistanceFromSun = 0.93E8	Filtering against a float literal in exponential format

Related concepts:

"Content based filtering" on page 6409
Content based filtering allows a subscriber to filter messages based on their content.

Related information:

"Using filters in content-based filtering" on page 6409
Content based filters are specified as ESQL expressions.

User-defined extensions

Reference material that supports the creation and management of your user-defined extensions.

This section contains the following information:

Related concepts:

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

"Developing user-defined extensions" on page 2970
A user-defined extension is a component that you design and implement to extend the function of WebSphere Message Broker.

Sample node files

Several sample node files are provided on all platforms.

Windows On Windows, the following sample node files are in the *install_dir*\sample\extensions\nodes directory, where *install_dir* is the home directory of your WebSphere Message Broker installation.

Linux On Linux, the following files are in the *install_dir*/sample/extensions/nodes directory, where *install_dir* is the home directory of your WebSphere Message Broker installation.

UNIX On UNIX, the following files are in the *install_dir*/sample/extensions/nodes directory, where *install_dir* is the home directory of your WebSphere Message Broker installation.

z/OS On z/OS, the following files are in the *install_dir*/sample/extensions/nodes directory, where *install_dir* is the home directory of your WebSphere Message Broker installation.

Sample node file	Description
SwitchNode.c	C source file containing a sample implementation of a message processing node that routes a message to one of five output terminals, depending on the content.
SwitchNode.h	The header file for the SwitchNode.c file.
TransformNode.c	C source file containing a sample implementation of a simple fixed transformation of an input message into an output message.
TransformNode.h	The header file for the TransformNode.c file.
BipSampPluginUtil.c	Sample utility functions used by the Switch and Transform nodes.
BipSampPluginUtil.h	The header file for BipSampPluginNode and BipSampPluginUtil.
NodeFactory.c	Common C functions for SwitchNode.c, TransformNode.c, and BipSampPluginUtil.c
NodeFactory.h	The header file for NodeFactory.c
Common.c	Common C functions for SwitchNode.c, TransformNode.c, and BipSampPluginUtil.c
Common.h	The header file for Common.c
PluginSample.add.xml	A sample XML input message that you can use to test the C sample nodes.
PluginSample.change.xml	A sample XML input message that you can use to test the C sample nodes.
PluginSample.delete.xml	A sample XML input message that you can use to test the C sample nodes.
JavaPlugin.add.xml	A sample XML input message that you can use to test the Java sample nodes.
JavaPlugin.change.xml	A sample XML input message that you can use to test the Java sample nodes.

Sample node file	Description
JavaPlugin.delete.xml	A sample XML input message that you can use to test the Java sample nodes.
JavaPlugin.hold.xml	A sample XML input message that you can use to test the Java sample nodes.

Windows On Windows, the following sample node files are in the *install_dir*\sample\extensions\nodes directory, where *install_dir* is the home directory of your WebSphere Message Broker installation.

Linux On Linux, the following files are in the *install_dir*/sample/Javaplugin/com/ibm/samples directory, where *install_dir* is the home directory of your WebSphere Message Broker installation.

UNIX On UNIX, the following files are in the *install_dir*/sample/Javaplugin/com/ibm/samples directory, where *install_dir* is the home directory of your WebSphere Message Broker installation.

z/OS On z/OS, the following files are in the *install_dir*/sample/Javaplugin/com/ibm/samples directory, where *install_dir* is the home directory of your WebSphere Message Broker installation.

Sample node file	Description
JavaSwitchPluginNode.java	Java source file containing a sample implementation of a message processing node that routes a message to one of five output terminals, depending on the content.
JavaTransformPluginNode.java	Java source file containing a sample implementation of a simple fixed transformation of an input message into an output message.

The files that the WebSphere Message Broker Toolkit needs to recognize the Switch node and Transform node are in the *install_dir*\sample\extensions\nodes\com.ibm.samples.nodes directory, where *install_dir* is the home directory of your WebSphere Message Broker installation. You can add this directory to your workspace using the Update Manager, or you can copy it across to your workspace directory and restart the WebSphere Message Broker Toolkit to see the nodes. The help files (HelpContexts.xml, SwitchNode.htm, and TransformNode.htm) demonstrate some features of Eclipse help by adding themselves into the main topic tree, referencing topics in the main tree, and so on.

GIF files that are used to represent the sample nodes in the WebSphere Message Broker Toolkit, which you can use, or replace with your own, are supplied. The GIF files come in three different sizes and can be found in individual directories under the sample\extensions\nodes\com.ibm.samples.nodes\icons\full\ directory.

SupportPacs:
Many other sample nodes are available as SupportPac offerings. For a complete list of available SupportPac offerings see WebSphere MQ SupportPacs web page.

Related concepts:

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

"Developing user-defined extensions" on page 2970
A user-defined extension is a component that you design and implement to extend the function of WebSphere Message Broker.

Related tasks:

"Compiling a Java user-defined node" on page 3074
When you have created the code for your Java user-defined node, you must compile it for your operating system.

"Compiling a C user-defined extension" on page 3047
Compile user-defined extensions in C for all supported operating systems.

Related reference:

"C language user-defined node API" on page 6416
Learn about the different types of call provided by the API.

Related information:

Java user-defined extensions API

Sample parser files

Several sample parser files are provided on all platforms, to help you to create your own parsers.

Windows On Windows, the following sample parser files are in the *install_dir*\sample\extensions\parser directory, where *install_dir* is the home directory of your WebSphere Message Broker installation.

Linux On Linux, the following sample parser files are in the *install_dir*/sample/extensions/parser directory, where *install_dir* is the home directory of your WebSphere Message Broker installation.

UNIX On UNIX, the following sample parser files are in the *install_dir*/sample/extensions/parser directory, where *install_dir* is the home directory of your WebSphere Message Broker installation.

z/OS On z/OS, the following sample parser files are in the *install_dir*/sample/extensions/parser directory, where *install_dir* is the home directory of your WebSphere Message Broker installation.

Sample parser file	Description
BipSampPluginParser.c	C source file containing sample implementations of a simple pseudo-XML parser.
BipSampPluginParser.h	The header file for the BipSampPluginParser.c file.

SupportPacs:
Many other sample parsers are available as SupportPac offerings. For a complete list of available SupportPacs, see http://www.ibm.com/software/integration/support/supportpacs/.

Related concepts:

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

"Developing user-defined extensions" on page 2970
A user-defined extension is a component that you design and implement to extend the function of WebSphere Message Broker.

Related tasks:

"Compiling a C user-defined extension" on page 3047
Compile user-defined extensions in C for all supported operating systems.

Related reference:

"C language user-defined node API" on page 6416
Learn about the different types of call provided by the API.

User-defined nodes

You can define your own nodes to use in WebSphere Message Broker message flows.

User-defined nodes add to the function that is provided by the WebSphere Message Broker built-in nodes. You can also use nodes that are created and supplied by independent software vendors and other companies.

If you create your own editors to handle properties for your nodes, you can access the supported "Property editor API" on page 6686 in this section.

Follow the instructions in "Adding help to the node" on page 3088 to provide help information for user-defined nodes, and how to include that help in this section of the information center.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

"Developing user-defined extensions" on page 2970
A user-defined extension is a component that you design and implement to extend the function of WebSphere Message Broker.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Resolving problems with user-defined extensions" on page 3511
Advice for dealing with some common problems that can arise when you work with user-defined extensions

Related reference:

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"User-defined extensions" on page 6411
Reference material that supports the creation and management of your user-defined extensions.

C Header files

The C interfaces are defined by the following header files.

- **BipCni.h** contains functions for user-defined nodes that have been written in C. For a list of functions, refer to the "C language user-defined node API."
- **BipCpi.h** contains functions for user-defined parsers that have been written in C. For a list of functions, refer to the "C language user-defined parser API" on page 6538.
- **BipCci.h** contains utility functions common to both user-defined nodes and parsers that have been written in C. For a list of functions, refer to "C common utility functions" on page 6643. This file also contains definitions for utility function return codes and values. For more information, see "Utility function return codes and values" on page 6686.
- **BipCos.h** contains operating system specific definitions for user-defined nodes that have been written in C.

Related concepts:

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Resolving problems with user-defined extensions" on page 3511
Advice for dealing with some common problems that can arise when you work with user-defined extensions

Related reference:

"C language user-defined parser API" on page 6538
The C language user-defined parser API consists of two complementary sets of functions that you can use to define the function of your parser.

"C language user-defined node API"
Learn about the different types of call provided by the API.

"C common utility functions" on page 6643
WebSphere Message Broker provides some additional utilities that user-defined nodes and parsers can use.

"Utility function return codes and values" on page 6686
By convention, the return code output parameter of all utility functions is set to indicate successful completion, or an error. The table lists all return codes with their meanings.

C language user-defined node API

Learn about the different types of call provided by the API.

The C language user-defined node API consists of:

1. A set of implementation functions that provide the functionality of the user-defined node. These functions are called by the broker. The implementation functions are mandatory, and if they are not supplied by the developer, an exception is thrown at run time.
2. A set of utility functions that create resources in the broker, or request a service of the broker. These utility functions are called by a user-defined node.

Most of the utilities are shared by all types of node; however, a few are specific to input nodes. This restriction is marked in the text.

These functions are defined in the `BipCni.h` header file.

This section covers the following topics:

"C node implementation functions"

"C node utility functions" on page 6419

Related concepts:

"Why use a user-defined extension?" on page 2972
Use a user-defined node or parser when the built-in resources do not provide the required functions.

"User-defined input nodes" on page 2990
A user-defined input node is an extension to the broker that provides a new input node in addition to the nodes supplied with the product.

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating an input node in C" on page 3027
Create a user-defined input node in C to receive messages into a message flow.

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C language user-defined parser API" on page 6538
The C language user-defined parser API consists of two complementary sets of functions that you can use to define the function of your parser.

"C Header files" on page 6415
The C interfaces are defined by the following header files.

C node implementation functions:

The user-defined node implements a function interface for the broker to call during runtime operation. This interface includes functions to create a local context whenever a node instance is created, functions to set and retrieve attribute values, the function to perform the processing of the node itself, and functions to examine messages.

The following functions are provided:

Mandatory function

"cniCreateNodeContext" on page 6447

Optional and conditional functions

- "cniDeleteNodeContext" on page 6454
- Either "cniEvaluate" on page 6475 (for message processing and output nodes), or "cniRun" on page 6504 (for input nodes)
- "cniGetAttribute" on page 6482
- "cniGetAttribute2" on page 6484
- "cniGetAttributeName" on page 6485
- "cniGetAttributeName2" on page 6487
- "cniSetAttribute" on page 6511

These implementation functions are called by the broker, and implemented by the node.

For certain implementation functions, you might need to specify the name of a parser that is supplied with WebSphere Message Broker. If so, you must use the correct class name of the parser. The following table provides a summary of the parsers, root element names, and class names for different headers.

Parser	Root element name	Class name
BLOB	BLOB	NONE
IDOC	IDOC	IDOC
JMSMap	JMSMap	JMS_MAP
JMSStream	JMSStream	JMS_STREAM
MIME	MIME	MIME
MQCFH	MQPCF	MQPCF
MQCIH	MQCIH	MQCICS
MQDLH	MQDLH	MQDEAD
MQIIH	MQIIH	MQIMS
MQMD	MQMD	MQHMD
MQMDE	MQMDE	MQHMDE
MQRFH	MQRFH	MQHRF
MQRFH2	MQRFH2	MQHRF2
MQRMH	MQRMH	MQHREF
MQSAPH	MQSAPH	MQHSAP
MQWIH	MQWIH	MQHWIH
MRM	MRM	MRM
Properties	Properties	PropertyParser
SMQ_BMH	SMQ_BMH	SMQBAD
XML	XML	xml
XMLNS	XMLNS	xmlns
XMLNSC	XMLNSC	xmlnsC

Related concepts:

"Why use a user-defined extension?" on page 2972
Use a user-defined node or parser when the built-in resources do not provide the required functions.

"User-defined input nodes" on page 2990
A user-defined input node is an extension to the broker that provides a new input node in addition to the nodes supplied with the product.

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating an input node in C" on page 3027
Create a user-defined input node in C to receive messages into a message flow.

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions"
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

C node utility functions:

A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

Functions are also provided to send messages to an output terminal for propagation to connected nodes, and to examine message content.

These utility functions are called by the node, and implemented by the broker.

Initialization and resource creation

- "cniCreateNodeFactory" on page 6449
- "cniDefineNodeClass" on page 6451
- "cniDispatchThread" on page 6456 (for input nodes only)
- "cniCreateInputTerminal" on page 6444
- "cniCreateOutputTerminal" on page 6450
- "cniIsTerminalAttached" on page 6494
- "cniGetBrokerInfo" on page 6488

Message management

- "cniCreateMessage" on page 6446
- "cniDeleteMessage" on page 6453
- "cniFinalize" on page 6479
- "cniGetMessageContext" on page 6491
- "cniGetEnvironmentMessage" on page 6490
- "cniPropagate" on page 6501

Message buffer access

- "cniBufferByte" on page 6426
- "cniBufferPointer" on page 6427
- "cniBufferSize" on page 6428
- "cniSetInputBuffer" on page 6520 (for input nodes only)
- "cniWriteBuffer" on page 6537

Syntax element navigation

- "cniRootElement" on page 6503
- "cniParent" on page 6498
- "cniNextSibling" on page 6497
- "cniPreviousSibling" on page 6499
- "cniFirstChild" on page 6481

- "cniLastChild" on page 6496
- "cniSearchElement group" on page 6506
- "cniSearchElementInNamespace group" on page 6508
- "cniSqlCreateReadOnlyPathExpression" on page 6524
- "cniSqlCreateModifyablePathExpression" on page 6521
- "cniSqlNavigatePath" on page 6532
- "cniSqlDeletePathExpression" on page 6529

Syntax element access

- "cniAddAfter" on page 6421
- "cniAddBefore" on page 6425
- "cniAddasFirstChild" on page 6422
- "cniAddasLastChild" on page 6423
- "cniCopyElementTree" on page 6430
- "cniCreateElementAfter" on page 6431
- "cniCreateElementAfterUsingParser" on page 6432
- "cniCreateElementBefore" on page 6442
- "cniCreateElementBeforeUsingParser" on page 6443
- "cniCreateElementAsFirstChild" on page 6433
- "cniCreateElementAsFirstChildUsingParser" on page 6435
- "cniCreateElementAsLastChild" on page 6436
- "cniCreateElementAsLastChildFromBitstream" on page 6437
- "cniCreateElementAsLastChildUsingParser" on page 6440
- "cniDetach" on page 6455
- "cniElementAsBitstream" on page 6458
- "cniElementName" on page 6464
- "cniElementNamespace" on page 6465
- "cniElementType" on page 6467
- "cniElementValue group" on page 6468
- "cniElementValueState" on page 6471
- "cniElementValueType" on page 6472
- "cniElementValueValue" on page 6473
- "cniGetParserClassName" on page 6492
- "cniSetElementName" on page 6513
- "cniSetElementNamespace" on page 6514
- "cniSetElementType" on page 6515
- "cniSetElementValue group" on page 6516
- "cniSetElementValueValue" on page 6519

SQL statement handling

- "cniSqlCreateStatement" on page 6527
- "cniSqlExecute" on page 6531
- "cniSqlSelect" on page 6535
- "cniSqlDeleteStatement" on page 6530

Miscellaneous

- "cniGetThreadContext" on page 6494

Related concepts:

"Why use a user-defined extension?" on page 2972
Use a user-defined node or parser when the built-in resources do not provide the required functions.

"User-defined input nodes" on page 2990
A user-defined input node is an extension to the broker that provides a new input node in addition to the nodes supplied with the product.

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating an input node in C" on page 3027
Create a user-defined input node in C to receive messages into a message flow.

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node implementation functions" on page 6417
The user-defined node implements a function interface for the broker to call during runtime operation. This interface includes functions to create a local context whenever a node instance is created, functions to set and retrieve attribute values, the function to perform the processing of the node itself, and functions to examine messages.

cniAddAfter:

Use this function to add an unattached syntax element after a specified syntax element. The currently unattached syntax element, and all child elements it possesses, is connected to the syntax element tree after the specified target element.

The added element becomes the *next sibling* of the target element. The target element must be attached to a tree (that is, it must have a parent element).

Syntax:

```
void cniAddAfter(
int*         returnCode,
CciElement*  targetElement,
CciElement*  newElement);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT

targetElement
The address of the target syntax element object (input).

newElement
The address of the new syntax element object that is to be added to the tree structure (input).

Return values:
None. If an error occurs, the *returnCode* parameter indicates the reason for the error.

Related concepts:

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniAddasFirstChild"
Use this function to add an unattached syntax element as the first child of a specified syntax element. The currently unattached syntax element, and all child elements it possesses, is connected to the syntax element tree as the *first child* of the specified target element. The target element need not be attached to a tree.

"cniAddasLastChild" on page 6423
Use this function to add an unattached syntax element as the last child of a specified syntax element. The currently unattached syntax element, and all child elements it possesses, is connected to the syntax element tree as the *last child* of the specified target element. The new element need not be attached to a tree.

"cniAddBefore" on page 6425
Use this function to add an unattached syntax element before a specified syntax element. The currently unattached syntax element, and all child elements it possesses, is connected to the syntax element tree before the specified target element.

cniAddasFirstChild:

Use this function to add an unattached syntax element as the first child of a specified syntax element. The currently unattached syntax element, and all child elements it possesses, is connected to the syntax element tree as the *first child* of the specified target element. The target element need not be attached to a tree.

Syntax:

```
void cniAddAsFirstChild(
  int*         returnCode,
  CciElement*  targetElement,
  CciElement*  newElement);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT

targetElement
The address of the target syntax element object (input).

newElement
The address of the new syntax element object that is to be added to the tree structure (input).

Return values:
None. If an error occurs, the *returnCode* parameter indicates the reason for the error.

Related concepts:

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniAddAfter" on page 6421
Use this function to add an unattached syntax element after a specified syntax element. The currently unattached syntax element, and all child elements it possesses, is connected to the syntax element tree after the specified target element.

"cniAddasLastChild"
Use this function to add an unattached syntax element as the last child of a specified syntax element. The currently unattached syntax element, and all child elements it possesses, is connected to the syntax element tree as the *last child* of the specified target element. The new element need not be attached to a tree.

"cniAddBefore" on page 6425
Use this function to add an unattached syntax element before a specified syntax element. The currently unattached syntax element, and all child elements it possesses, is connected to the syntax element tree before the specified target element.

cniAddasLastChild:

Use this function to add an unattached syntax element as the last child of a specified syntax element. The currently unattached syntax element, and all child

elements it possesses, is connected to the syntax element tree as the *last child* of the specified target element. The new element need not be attached to a tree.

Syntax:

```
void cniAddAsLastChild(
  int*         returnCode,
  CciElement*  targetElement,
  CciElement*  newElement);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT

targetElement
The address of the target syntax element object (input).

newElement
The address of the new syntax element object that is to be added to the tree structure (input).

Return values:
None. If an error occurs, the *returnCode* parameter indicates the reason for the error.

Related concepts:

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniAddAfter" on page 6421
Use this function to add an unattached syntax element after a specified syntax element. The currently unattached syntax element, and all child elements it possesses, is connected to the syntax element tree after the specified target element.

"cniAddasFirstChild" on page 6422
Use this function to add an unattached syntax element as the first child of a specified syntax element. The currently unattached syntax element, and all child elements it possesses, is connected to the syntax element tree as the *first child* of the specified target element. The target element need not be attached to a tree.

"cniAddBefore"
Use this function to add an unattached syntax element before a specified syntax element. The currently unattached syntax element, and all child elements it possesses, is connected to the syntax element tree before the specified target element.

cniAddBefore:

Use this function to add an unattached syntax element before a specified syntax element. The currently unattached syntax element, and all child elements it possesses, is connected to the syntax element tree before the specified target element.

The newly added element becomes the *previous sibling* of the target element. The target element must be attached to a tree (that is, it must have a parent element).

Syntax:

```
void cniAddBefore(
  int*         returnCode,
  CciElement*  targetElement,
  CciElement*  newElement);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT

targetElement
The address of the target syntax element object (input).

newElement
The address of the new syntax element object that is to be added to the tree structure (input).

Return values:
None. If an error occurs, the *returnCode* parameter indicates the reason for the error.

Related concepts:

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniAddAfter" on page 6421
Use this function to add an unattached syntax element after a specified syntax element. The currently unattached syntax element, and all child elements it possesses, is connected to the syntax element tree after the specified target element.

"cniAddasFirstChild" on page 6422
Use this function to add an unattached syntax element as the first child of a specified syntax element. The currently unattached syntax element, and all child elements it possesses, is connected to the syntax element tree as the *first child* of the specified target element. The target element need not be attached to a tree.

"cniAddasLastChild" on page 6423
Use this function to add an unattached syntax element as the last child of a specified syntax element. The currently unattached syntax element, and all child elements it possesses, is connected to the syntax element tree as the *last child* of the specified target element. The new element need not be attached to a tree.

cniBufferByte:

Use this function to get a single byte from the data buffer associated with (and owned by) the message object specified in the message argument. The value of the index argument indicates which byte in the byte array is to be returned.

Syntax:

```
CciByte cniBufferByte(
  int*         returnCode,
  CciMessage*  message,
  CciSize      index);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_MESSAGE_OBJECT

message
The address of the message object for which the size of the data buffer is to be returned (input).

index
The offset to use as an index into the buffer (input).

Return values:
The requested byte is returned. If an error occurred, the *returnCode* parameter indicates the reason for the error.

Related concepts:

"User-defined input nodes" on page 2990
A user-defined input node is an extension to the broker that provides a new input node in addition to the nodes supplied with the product.

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating an input node in C" on page 3027
Create a user-defined input node in C to receive messages into a message flow.

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniBufferPointer"
Use this function to get a pointer to the data buffer associated with (and owned by) the message object specified in the message argument. This function is typically used by output nodes.

"cniBufferSize" on page 6428
Use this function to get the size of the data buffer associated with (and owned by) the message object specified in the message argument.

"cniSetInputBuffer" on page 6520
Use this function to supply a buffer. It is used only by input nodes. The address is specified by the `source` parameter as an input bit stream of the input message to the broker.

"cniWriteBuffer" on page 6537
Use this function to write the syntax element tree associated with the specified message to the data buffer that is owned by the message object. This function is typically used by output nodes.

cniBufferPointer:

Use this function to get a pointer to the data buffer associated with (and owned by) the message object specified in the message argument. This function is typically used by output nodes.

Syntax:

```
const CciByte* cniBufferPointer(
  int*         returnCode,
  CciMessage*  message);
```

Parameters:

`returnCode`
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_MESSAGE_OBJECT

message
The address of the message object for which the address of the data buffer is to be returned (input).

Return values:
If successful, the address of the data buffer is returned. Otherwise, zero (CCI_NULL_ADDR) is returned, and the *returnCode* parameter indicates the reason for the error.

Related concepts:

"User-defined input nodes" on page 2990
A user-defined input node is an extension to the broker that provides a new input node in addition to the nodes supplied with the product.

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating an input node in C" on page 3027
Create a user-defined input node in C to receive messages into a message flow.

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniBufferByte" on page 6426
Use this function to get a single byte from the data buffer associated with (and owned by) the message object specified in the message argument. The value of the index argument indicates which byte in the byte array is to be returned.

"cniBufferSize"
Use this function to get the size of the data buffer associated with (and owned by) the message object specified in the message argument.

"cniSetInputBuffer" on page 6520
Use this function to supply a buffer. It is used only by input nodes. The address is specified by the `source` parameter as an input bit stream of the input message to the broker.

"cniWriteBuffer" on page 6537
Use this function to write the syntax element tree associated with the specified message to the data buffer that is owned by the message object. This function is typically used by output nodes.

cniBufferSize:

Use this function to get the size of the data buffer associated with (and owned by) the message object specified in the message argument.

Syntax:

```
CciSize cniBufferSize(
  int*         returnCode,
  CciMessage*  message);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_MESSAGE_OBJECT

message
The address of the message object for which the size of the data buffer is to be returned (input).

Return values:
The size of the buffer in bytes, or zero if no buffer exists. If an error occurred, (CCI_FAILURE) is returned, and the *returnCode* parameter indicates the reason for the error.

Related concepts:

"User-defined input nodes" on page 2990
A user-defined input node is an extension to the broker that provides a new input node in addition to the nodes supplied with the product.

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating an input node in C" on page 3027
Create a user-defined input node in C to receive messages into a message flow.

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniBufferByte" on page 6426
Use this function to get a single byte from the data buffer associated with (and owned by) the message object specified in the message argument. The value of the index argument indicates which byte in the byte array is to be returned.

"cniBufferPointer" on page 6427
Use this function to get a pointer to the data buffer associated with (and owned by) the message object specified in the message argument. This function is typically used by output nodes.

"cniSetInputBuffer" on page 6520
Use this function to supply a buffer. It is used only by input nodes. The address is specified by the `source` parameter as an input bit stream of the input message to

the broker.

"cniWriteBuffer" on page 6537
Use this function to write the syntax element tree associated with the specified message to the data buffer that is owned by the message object. This function is typically used by output nodes.

cniCopyElementTree:

Use this function to copy a part of the element tree from the source element to the target element. Only the child elements of the source element are copied. All existing child elements of the target element are deleted, and are replaced by the child elements of the source element.

If the target element has not been fully parsed, or represents an unparsed bit stream, the cniCopyElementTree function results in a parse of the target element before its child elements are detached. The function therefore ensures consistency in message-tree formatting so that all references to detached fields by cciElements remain valid. Therefore, if a parsing exception occurs during the execution of the cniCopyElementTree function the cause might be a problem with either the target element or the source element.

Syntax:

```
void cniCopyElementTree(
  int*         returnCode,
  CciElement*  sourceElement,
  CciElement*  targetElement);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_MESSAGE_OBJECT

sourceElement
The address of the source syntax element object (input).

targetElement
The address of the target syntax element object (input).

Return values:
None. If an error occurs, the *returnCode* parameter indicates the reason for the error.

Example:

```
cniCopyElementTree(&rc, inRootElement, outRootElement);
```

Related concepts:

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

cniCreateElementAfter:

Use this function to create a new syntax element and insert it after the specified syntax element. The new element becomes the *next sibling* of the specified element.

Do not use cniCreateElementAfter when you create a message body folder (such as XML, XMLNS, MRM, BLOB), because this function does not associate an owning parser with the folder. To create a message body folder, you can use one of the following functions:

cniCreateElementAsFirstChildUsingParser

cniCreateElementAsLastChildUsingParser

cniCreateElementAfterUsingParser

cniCreateElementBeforeUsingParser

When the message body folder has been created, you can use cniCreateElementAfter to create elements under the folder. You can use cniCreateElementAfter because the parser, which is associated with the message body folder, is inherited.

Syntax:

```
CciElement* cniCreateElementAfter(
  int*         returnCode,
  CciElement*  targetElement);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT

targetElement
The address of the element object (input).

Return values:
If successful, the address of the new element object is returned. Otherwise, a value of zero (CCI_NULL_ADDR) is returned, and the *returnCode* parameter indicates the reason for the error.

Related concepts:

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniCreateElementAfterUsingParser"
Use this function to create a syntax element, insert it after the specified syntax element, and associate it with the specified parser class name. The new element becomes the *next sibling* of the specified element.

"cniCreateElementBefore" on page 6442
Use this function to create a new syntax element and insert it before the specified syntax element. The new element becomes the *previous sibling* of the specified element, and shares the same parent element.

"cniCreateElementBeforeUsingParser" on page 6443
Use this function to create a syntax element, insert it before the specified syntax element, and associate it with the specified parser class name. The new element becomes the *previous sibling* of the specified element.

cniCreateElementAfterUsingParser:

Use this function to create a syntax element, insert it after the specified syntax element, and associate it with the specified parser class name. The new element becomes the *next sibling* of the specified element.

A portion of the syntax element tree that is owned by a parser can *only* have its effective root at the first generation of elements (that is, as *immediate children of root*). The user-defined node interface does not restrict the ability to create a subtree that appears to be owned by a different parser. However, it is not possible to serialize these element trees into a bit stream when producing an output message.

If you specify the name of a parser supplied with WebSphere Message Broker, you must use the correct class name of the parser.

The internal name for the BLOB parser is *none*. Therefore, if you use this function to create a BLOB parser folder, the associated parser name should be *none*.

Syntax:

```
CciElement* cniCreateElementAfterUsingParser(
  int*            returnCode,
  CciElement*     targetElement,
  const CciChar*  parserClassName);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT
- CCI_INV_PARSER_NAME

TargetElement
The address of the element object (input).

parserClassName
The name of the parser class (input).

Return values:
If successful, the address of the new element object is returned. Otherwise, a value of zero (CCI_NULL_ADDR) is returned, and the *returnCode* parameter indicates the reason for the error.

Related concepts:

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniCreateElementAfter" on page 6431
Use this function to create a new syntax element and insert it after the specified syntax element. The new element becomes the *next sibling* of the specified element.

"cniCreateElementBefore" on page 6442
Use this function to create a new syntax element and insert it before the specified syntax element. The new element becomes the *previous sibling* of the specified element, and shares the same parent element.

"cniCreateElementBeforeUsingParser" on page 6443
Use this function to create a syntax element, insert it before the specified syntax element, and associate it with the specified parser class name. The new element becomes the *previous sibling* of the specified element.

cniCreateElementAsFirstChild:

Use this function to create a syntax element as the first child of the specified syntax element.

Do not use cniCreateElementAsFirstChild when creating a message body folder (such as XML, XMLNS, MRM, BLOB), because it does not associate an owning parser with the folder. To create a message body folder, you can use one of the following functions:

cniCreateElementAsFirstChildUsingParser

cniCreateElementAsLastChildUsingParser

cniCreateElementAfterUsingParser

cniCreateElementBeforeUsingParser

When the message body folder has been created, you can call cniCreateElementAsFirstChild to create elements under the folder. You can use cniCreateElementAsFirstChild because the parser, which is associated with the message body folder, is inherited.

Syntax:

```
CciElement* cniCreateElementAsFirstChild(
  int*         returnCode,
  CciElement*  targetElement);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT

targetElement
The address of the element object (input).

Return values:
If successful, the address of the new element object is returned. Otherwise, a value of zero (CCI_NULL_ADDR) is returned, and the *returnCode* parameter indicates the reason for the error.

Related concepts:

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniCreateElementAsFirstChildUsingParser" on page 6435
Use this function to create a syntax element as the first child of the specified syntax element, and associates it with the specified parser class name.

"cniCreateElementAsLastChild" on page 6436
Use this function to create a syntax element as the last child of the specified syntax element.

"cniCreateElementAsLastChildFromBitstream" on page 6437
Use this function to create a syntax element tree as the last child of the specified syntax element, and associates it with the specified parser. The syntax element tree is populated by parsing the specified bit stream.
"cniCreateElementAsLastChildUsingParser" on page 6440
Use this function to create a syntax element as the last child of the specified syntax element, and associate it with the specified parser class name.

cniCreateElementAsFirstChildUsingParser:

Use this function to create a syntax element as the first child of the specified syntax element, and associates it with the specified parser class name.

A portion of the syntax element tree that is owned by a parser can *only* have its effective root at the first generation of elements (that is, as *immediate children of root*). The user-defined node interface does not restrict the ability to create a subtree that appears to be owned by a different parser. However, it is not possible to serialize these element trees into a bit stream when producing a message.

If you specify the name of a parser supplied with WebSphere Message Broker, you must use the correct class name of the parser.

The internal name for the BLOB parser is *none*. Therefore, if you use this function to create a BLOB parser folder, the associated parser name should be *none*.

Syntax:

```
CciElement* cniCreateElementAsFirstChildUsingParser(
  int*            returnCode,
  CciElement*     targetElement,
  const CciChar*  parserClassName);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT
- CCI_INV_PARSER_NAME

targetElement
The address of the element object (input).

parserClassName
The name of the parser class (input).

Return values:
If successful, the address of the new element object is returned. Otherwise, a value of zero (CCI_NULL_ADDR) is returned, and the *returnCode* parameter indicates the reason for the error.

Related concepts:

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniCreateElementAsFirstChild" on page 6433
Use this function to create a syntax element as the first child of the specified syntax element.

"cniCreateElementAsLastChild"
Use this function to create a syntax element as the last child of the specified syntax element.

"cniCreateElementAsLastChildFromBitstream" on page 6437
Use this function to create a syntax element tree as the last child of the specified syntax element, and associates it with the specified parser. The syntax element tree is populated by parsing the specified bit stream.

"cniCreateElementAsLastChildUsingParser" on page 6440
Use this function to create a syntax element as the last child of the specified syntax element, and associate it with the specified parser class name.

cniCreateElementAsLastChild:

Use this function to create a syntax element as the last child of the specified syntax element.

Do not use cniCreateElementAsLastChild when creating a message body folder (such as XML, XMLNS, MRM, BLOB), because it does not associate an owning parser with the folder. To create a message body folder, you can use one of the following functions:

cniCreateElementAsFirstChildUsingParser

cniCreateElementAsLastChildUsingParser

cniCreateElementAfterUsingParser

cniCreateElementBeforeUsingParser

When the message body folder has been created, you can use cniCreateElementAsLastChild to create elements under the folder. You can use cniCreateElementAsLastChild because the parser, which is associated with the message body folder, is inherited.

Syntax:

```
CciElement* cniCreateElementAsLastChild(
  int*         returnCode,
  CciElement*  targetElement);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT

targetElement
The address of the element object (input).

Return values:
If successful, the address of the new element object is returned. Otherwise, a value of zero (CCI_NULL_ADDR) is returned and the *returnCode* parameter indicates the reason for the error.

Example:

```
CciElement* lastChild = cniCreateElementAsLastChild(&rc, outRootElement);
cniSetElementName(&rc, lastChild, elementName);
```

Related concepts:

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniCreateElementAsFirstChild" on page 6433
Use this function to create a syntax element as the first child of the specified syntax element.

"cniCreateElementAsFirstChildUsingParser" on page 6435
Use this function to create a syntax element as the first child of the specified syntax element, and associates it with the specified parser class name.

"cniCreateElementAsLastChildFromBitstream"
Use this function to create a syntax element tree as the last child of the specified syntax element, and associates it with the specified parser. The syntax element tree is populated by parsing the specified bit stream.

"cniCreateElementAsLastChildUsingParser" on page 6440
Use this function to create a syntax element as the last child of the specified syntax element, and associate it with the specified parser class name.

cniCreateElementAsLastChildFromBitstream:

Use this function to create a syntax element tree as the last child of the specified syntax element, and associates it with the specified parser. The syntax element tree is populated by parsing the specified bit stream.

During the execution of this function, the bit stream is copied, so the caller can free or reuse the memory allocated to hold the original bit stream. You can use this function only to create a message body, that is, the last child of the message property. An output message must already exist.

The root element of this output message should be passed in as the target element parameter. Because this call is designed only to be used to create a message body, you cannot use it to build successive elements. For example, it should not be used to create an MQRFH2 header as the last child of root, then an XML message as the last child of root, after the MQRFH2 header.

Syntax:

```
CciElement* cniCreateElementAsLastChildFromBitstream (
  int*                        returnCode,
  CciElement*                 targetElement,
  const struct CciByteArray* value,
  const CciChar*              parserClassName,
  CciChar*                    messageType,
  CciChar*                    messageSet,
  CciChar*                    messageFormat,
  int                         encoding,
  int                         ccsid,
  int                         options);
```

Parameters:

returnCode
The return code from the function (output). Specifying a NULL pointer signifies that the node will not deal with errors. If input is not NULL, the output signifies the success status of the call. Any exceptions thrown during the execution of this call are re-thrown to the next upstream node in the flow. Call cciGetLastExceptionData for details of the exception.

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT
- CCI_INV_PARSER_NAME
- CCI_INV_DATA_POINTER

targetElement
The syntax element under which the new syntax element tree is created (input). This parameter must be the message property.

parserClassName
The name of the parser class to use to parse the bit stream (input). You must use the same parser that was used to parse the whole bit stream.

value
A pointer to a CciByteArray struct containing a pointer to the bit stream to be parsed, and also the size in CciBytes of this bit stream (output).

messageType
The message type definition used to create the element tree from the bit stream (input). A NULL pointer means that this parameter is ignored. Also, if the parser specified has no interest in this value, for example if it is a generic XML parser, the parameter is ignored.

messageSet
The message set definition used to create the element tree from the bit stream

(input). A NULL pointer means that this parameter is ignored. Also, if the parser specified has no interest in this value, for example if it is a generic XML parser, the parameter is ignored.

messageFormat
The format used to create the element tree from the bit stream (input). A NULL pointer means that this parameter is ignored. Also, if the parser specified has no interest in this value, for example if it is a generic XML parser, the parameter is ignored.

encoding
The encoding to use when parsing the bit stream (input). This parameter is mandatory. You can specify a value of 0 to indicate that the queue manager's encoding should be used.

ccsid
The coded character set identifier to use when parsing the bit stream (input). This parameter is mandatory. You can specify a value of 0 to indicate that the queue manager's ccsid should be used.

options
This is reserved for future use. You must specify a value of 0 to maintain forward compatibility.

Return values:
If successful, the address of the new element object is returned. Otherwise, a value zero (CCI_NULL_ADDR) is returned, and the return code parameter indicates the reason for the error. If an exception occurs during execution, *returnCode* is set to CCI_EXCEPTION

Example:

```
  outMQMD = cniCreateElementAsFirstChildUsingParser(&rc,
                                                    outRootElement,
                                                    CciString("MQHMD",BIP_DEF_COMP_CCSID));
  checkRC(rc);

  cniCopyElementTree(&rc, inMQMD, outMQMD);
  checkRC(rc);

  outBlobRoot = cniCreateElementAsLastChildFromBitstream(
                                                      &rc,
                                                      outRootElement,
                                                      &bitstream,
                                                      inParserClassName,
                                                      messageType,
                                                      messageSet,
                                                      messageFormat,
                                                      encoding,
                                                      ccsid,
                                                      0);



  checkRC(rc);
  ...

  return;
}
```

Related concepts:

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"cciGetLastExceptionData" on page 6647
Gets diagnostic information about the last exception generated. Information about the last exception generated on the current thread is returned in a CCI_EXCEPTION_ST output structure. The user-defined extension can use this function to determine whether any recovery is required when a utility function returns an error code.

"cniCreateElementAsFirstChild" on page 6433
Use this function to create a syntax element as the first child of the specified syntax element.

"cniCreateElementAsFirstChildUsingParser" on page 6435
Use this function to create a syntax element as the first child of the specified syntax element, and associates it with the specified parser class name.

"cniCreateElementAsLastChild" on page 6436
Use this function to create a syntax element as the last child of the specified syntax element.

"cniCreateElementAsLastChildUsingParser"
Use this function to create a syntax element as the last child of the specified syntax element, and associate it with the specified parser class name.

cniCreateElementAsLastChildUsingParser:

Use this function to create a syntax element as the last child of the specified syntax element, and associate it with the specified parser class name.

A portion of the syntax element tree that is owned by a parser can *only* have its effective root at the first generation of elements (that is, as *immediate children of root*). The user-defined node interface does not restrict the ability to create a subtree that appears to be owned by a different parser. However, it is not possible to serialize these element trees into a bit stream when producing a message.

If you specify the name of a parser supplied with WebSphere Message Broker, you must use the correct class name of the parser. See "C node implementation functions" on page 6417 for a list of the supplied parsers.

The internal name for the BLOB parser is *none*. Therefore, if you use this function to create a BLOB parser folder, the associated parser name must be *none*.

Syntax:

```
CciElement* cniCreateElementAsLastChildUsingParser(
  int*           returnCode,
  CciElement*    targetElement,
  const CciChar* parserClassName);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT
- CCI_INV_PARSER_NAME

targetElement
The address of the element object (input).

parserClassName
The name of the parser class (input).

Return values:
If successful, the address of the new element object is returned. Otherwise, a value of zero (CCI_NULL_ADDR) is returned, and the *returnCode* parameter indicates the reason for the error.

Example:

```
cniElementName(&rc, firstChild, elementName);
CciElementType type = cniElementType(&rc, firstChild);
CciElement* lastChild = cniCreateElementAsLastChildUsingParser(
                                                              &rc,
                                                              outRootElement,
                                                              parserName);
cniSetElementName(&rc, lastChild, elementName);
cniSetElementType(&rc, lastChild, elementType);
```

Related concepts:

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"C node implementation functions" on page 6417
The user-defined node implements a function interface for the broker to call during runtime operation. This interface includes functions to create a local context whenever a node instance is created, functions to set and retrieve attribute values, the function to perform the processing of the node itself, and functions to examine messages.

"cniCreateElementAsFirstChild" on page 6433
Use this function to create a syntax element as the first child of the specified syntax element.

"cniCreateElementAsFirstChildUsingParser" on page 6435
Use this function to create a syntax element as the first child of the specified syntax element, and associates it with the specified parser class name.

"cniCreateElementAsLastChild" on page 6436
Use this function to create a syntax element as the last child of the specified syntax element.

"cniCreateElementAsLastChildFromBitstream" on page 6437
Use this function to create a syntax element tree as the last child of the specified syntax element, and associates it with the specified parser. The syntax element tree is populated by parsing the specified bit stream.

cniCreateElementBefore:

Use this function to create a new syntax element and insert it before the specified syntax element. The new element becomes the *previous sibling* of the specified element, and shares the same parent element.

DO not use cniCreateElementBefore when creating a message body folder (such as XML, XMLNS, MRM, BLOB), because it does not associate an owning parser with the folder. To create a message body folder, you can use one of the following functions:

cniCreateElementAsFirstChildUsingParser

cniCreateElementAsLastChildUsingParser

cniCreateElementAfterUsingParser

cniCreateElementBeforeUsingParser

When the message body folder has been created, `cniCreateElementBefore` can be used to create elements under the folder. `cniCreateElementBefore` can be used because the parser, which is associated with the message body folder, is inherited.

Syntax:

```
CciElement* cniCreateElementBefore(
  int*         returnCode,
  CciElement*  targetElement);
```

Parameters:

`returnCode`
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT

`targetElement`
The address of the target element object (input).

Return values:
If successful, the address of the new element object is returned. Otherwise, a value of zero (CCI_NULL_ADDR) is returned, and the *returnCode* parameter indicates the reason for the error.

Related concepts:

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new

message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniCreateElementAfter" on page 6431
Use this function to create a new syntax element and insert it after the specified syntax element. The new element becomes the *next sibling* of the specified element.

"cniCreateElementAfterUsingParser" on page 6432
Use this function to create a syntax element, insert it after the specified syntax element, and associate it with the specified parser class name. The new element becomes the *next sibling* of the specified element.

"cniCreateElementBeforeUsingParser"
Use this function to create a syntax element, insert it before the specified syntax element, and associate it with the specified parser class name. The new element becomes the *previous sibling* of the specified element.

cniCreateElementBeforeUsingParser:

Use this function to create a syntax element, insert it before the specified syntax element, and associate it with the specified parser class name. The new element becomes the *previous sibling* of the specified element.

A portion of the syntax element tree that is owned by a parser can *only* have its effective root at the first generation of elements (that is, as *immediate children of root*). The user-defined node interface does not restrict the ability to create a subtree that appears to be owned by a different parser. However, it is not possible to serialize these element trees into a bit stream when producing a message.

If you specify the name of a parser supplied with WebSphere Message Broker, you must use the correct class name of the parser.

The internal name for the BLOB parser is *none*. Therefore, if you use this function to create a BLOB parser folder, the associated parser name should be *none*.

Syntax:

```
CciElement* cniCreateElementBeforeUsingParser(
  int*            returnCode,
  CciElement*     targetElement,
  const CciChar*  parserClassName);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION

- CCI_INV_ELEMENT_OBJECT
- CCI_INV_PARSER_NAME

targetElement
The address of the element object (input).

parserClassName
The name of the parser class (input).

Return values:
If successful, the address of the new element object is returned. Otherwise, a value of zero (CCI_NULL_ADDR) is returned, and the *returnCode* parameter indicates the reason for the error.

Related concepts:

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniCreateElementAfter" on page 6431
Use this function to create a new syntax element and insert it after the specified syntax element. The new element becomes the *next sibling* of the specified element.

"cniCreateElementAfterUsingParser" on page 6432
Use this function to create a syntax element, insert it after the specified syntax element, and associate it with the specified parser class name. The new element becomes the *next sibling* of the specified element.

"cniCreateElementBefore" on page 6442
Use this function to create a new syntax element and insert it before the specified syntax element. The new element becomes the *previous sibling* of the specified element, and shares the same parent element.

cniCreateInputTerminal:

Use this function to create an input terminal on an instance of a node object and return the address of the terminal object that was created.

The terminal object is destroyed by the broker when its owning node is destroyed. You must call this function only from within the implementation function cniCreateNodeContext.

Syntax:

```
CciTerminal* cniCreateInputTerminal(
  int*      returnCode,
  CciNode*  nodeObject,
  CciChar*  name);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_NODE_OBJECT
- CCI_INV_TERMINAL_NAME

nodeObject
Specifies the address of the instance of the node object on which the input terminal is to be created (input). The handle is passed to the cniCreateNodeContext function.

name
Specifies a name for the terminal being created (input).

Return values:
If successful, the address of the node terminal object is returned. Otherwise, a value of zero (CCI_NULL_ADDR) is returned.

Example:

```
entry->handle = cniCreateInputTerminal(
                                        &rc,
                                        context->nodeObject,
                                        (CciChar*)terminalName);
```

Related concepts:

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node implementation functions" on page 6417
The user-defined node implements a function interface for the broker to call during runtime operation. This interface includes functions to create a local context whenever a node instance is created, functions to set and retrieve attribute values, the function to perform the processing of the node itself, and functions to examine messages.

"cniCreateNodeContext" on page 6447
This function creates a context for an instance of a node object. It is called by the broker whenever an instance of a node object is constructed. Nodes are constructed when a message flow is deployed by the broker, or when the execution group is started.

"cniCreateOutputTerminal" on page 6450
Use this function to create an output terminal on an instance of a node object and return the address of the terminal object that was created.

cniCreateMessage:

Use this function to create an output message object. For every call to this function, you must include a matching call to cniDeleteMessage to return allocated resources when the processing on the output message has been completed.

Syntax:

```
CciMessage* cniCreateMessage(
  int*                returnCode,
  CciMessageContext*  messageContext);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_FAILURE
- CCI_EXCEPTION
- CCI_INV_MESSAGE_CONTEXT

messageContext
The address of the context for the message (input). Use cniGetMessageContext to get the context from an incoming message; for example, one received in the cniEvaluate function.

Return values:
If successful, the address of the message object is returned. Otherwise, a value of zero (CCI_NULL_ADDR) is returned, and the *returnCode* parameter indicates the reason for the error.

Example:

```
outMsg = cniCreateMessage(&rc, cniGetMessageContext(&rc, message));
```

Related concepts:

"User-defined input nodes" on page 2990
A user-defined input node is an extension to the broker that provides a new input node in addition to the nodes supplied with the product.

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating an input node in C" on page 3027
Create a user-defined input node in C to receive messages into a message flow.

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniDeleteMessage" on page 6453
Use this function to delete the specified message object. For every call to the cniCreateMessage function, you must include a matching call to cniDeleteMessage to return allocated resources when the processing on the output message has been completed.

"cniEvaluate" on page 6475
This function performs node processing. The broker calls this function when a message is received on one of the input terminals of an instance of a node object.

"cniFinalize" on page 6479
Use this function to cause the broker to request parsers to perform finalize processing on the specified message. Finalization is a process that fixes header chains and makes the Properties folder match the headers.

"cniGetEnvironmentMessage" on page 6490
Use this function to get the CciMessage object that corresponds to the environment tree for the message flow.

"cniGetMessageContext" on page 6491
Use this function to get the address of the message context that is associated with the specified message. The context of an existing message is used to create an output message; for example, by using the cniCreateMessage function.

"cniPropagate" on page 6501
Use this function to propagate a message to a specified terminal object. If the terminal is not attached to another node by a connector, the message is not propagated, and the function is ignored.

cniCreateNodeContext:

This function creates a context for an instance of a node object. It is called by the broker whenever an instance of a node object is constructed. Nodes are constructed when a message flow is deployed by the broker, or when the execution group is started.

The responsibilities of the node, when created, are to:

1. (Optional) Verify that the name of the node specified in the *nodeName* parameter is supported by the factory.
2. Allocate any node instance specific data areas that might be required (for example: context, attribute data, and terminals).
3. Perform all additional resource acquisition or initialization that might be required for the processing of the node.
4. Return the address of the context to the calling function. Whenever an implementation function for this node instance is called, the appropriate context is passed as an argument to that function. Therefore, a user-defined node developed in C does not have to maintain its own static pointers to per-instance data areas.

Defined In	Type	Member
CNI_VFT	Mandatory	iFpCreateNodeContext

Syntax:

```
CciContext* cniCreateNodeContext(
  CciFactory*  factoryObject,
  CciChar*     nodeName,
  CciNode*     nodeObject);
```

Parameters:

factoryObject
The address of the factory object that owns the node being created (input).

nodeName
The name of the node being created (input).

nodeObject
The address of the node object that has just been created (input).

Return values:
If successful, the address of the node context is returned. Otherwise, a value of zero (CCI_NULL_ADDR) is returned.

Example:

```
  static char* functionName = (char *)"_Switch_createNodeContext()";
  NODE_CONTEXT_ST* p;

  /* Allocate a pointer to the local context */
  p = (NODE_CONTEXT_ST *)malloc(sizeof(NODE_CONTEXT_ST));

  if (p) {

    /* Clear the context area */
    memset(p, 0, sizeof(NODE_CONTEXT_ST));

    /* Save our node object pointer in our context */
    p->nodeObject = nodeObject;

    /* Save our node name */
    CciCharNCpy((CciChar*) &p->nodeName, nodeName, MAX_NODE_NAME_LEN);
}
  else
  /* Handle errors */
```

Related concepts:

"User-defined input nodes" on page 2990
A user-defined input node is an extension to the broker that provides a new input node in addition to the nodes supplied with the product.

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating an input node in C" on page 3027
Create a user-defined input node in C to receive messages into a message flow.

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node implementation functions" on page 6417
The user-defined node implements a function interface for the broker to call during runtime operation. This interface includes functions to create a local context whenever a node instance is created, functions to set and retrieve attribute values, the function to perform the processing of the node itself, and functions to examine messages.

"cniDeleteNodeContext" on page 6454
This function deletes any context for an instance of a user-defined node object. It is called by the broker whenever an instance of a node object is destroyed, when a message flow is deleted, or when a configuration is redeployed.

cniCreateNodeFactory:

Use this function to create a node factory in the broker. A single instance of the named message flow node factory is created.

This function must be called only in the initialization function bipGetMessageFlowNodeFactory, which is called when the LIL is loaded by the message broker. If cniCreateNodeFactory is called at any other time, the results are unpredictable.

Syntax:

```
CciFactory* cniCreateNodeFactory(
  int*      returnCode,
  CciChar*  name);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_FAILURE
- CCI_EXCEPTION
- CCI_INV_FACTORY_NAME
- CCI_INV_OBJECT_NAME

name
The name of the factory being created (input).

Return values:
If successful, the address of the node factory object is returned. Otherwise, a value of zero (CCI_NULL_ADDR) is returned, and the *returnCode* parameter indicates the reason for the error.

Example:

```
factoryObject = cniCreateNodeFactory(0, (unsigned short *)constPluginNodeFactory);
if (factoryObject == CCI_NULL_ADDR) {

/* Handle errors */
```

Related concepts:

"User-defined input nodes" on page 2990
A user-defined input node is an extension to the broker that provides a new input node in addition to the nodes supplied with the product.

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating an input node in C" on page 3027
Create a user-defined input node in C to receive messages into a message flow.

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniDefineNodeClass" on page 6451
Use this function to define a node class, as specified by the *name* parameter, which is supported by the node factory specified as the*factoryObject* parameter.

cniCreateOutputTerminal:

Use this function to create an output terminal on an instance of a node object and return the address of the terminal object that was created.

The terminal object is destroyed when its owning node is destroyed. You must call this function only from within the implementation function cniCreateNodeContext.

Syntax:

```
CciTerminal* cniCreateOutputTerminal(
  int*      returnCode,
  CciNode*  nodeObject,
  CciChar*  name);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_FAILURE
- CCI_EXCEPTION
- CCI_INV_NODE_OBJECT
- CCI_INV_TERMINAL_NAME

nodeObject
The address of the instance of the node object on which the output terminal is to be created (input). The handle is passed to the cniCreateNodeContext function.

name
The name of the terminal being created (input).

Return values:

If successful, the address of the node terminal object is returned. Otherwise, a value of zero (CCI_NULL_ADDR) is returned.

Example:

```
entry->handle = cniCreateOutputTerminal(
                                        &rc,
                                        context->nodeObject
                                        (CciChar*)terminalName);
```

Related concepts:

"User-defined input nodes" on page 2990
A user-defined input node is an extension to the broker that provides a new input node in addition to the nodes supplied with the product.

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

Related tasks:

"Creating an input node in C" on page 3027
Create a user-defined input node in C to receive messages into a message flow.

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniCreateInputTerminal" on page 6444
Use this function to create an input terminal on an instance of a node object and return the address of the terminal object that was created.

"cniCreateNodeContext" on page 6447
This function creates a context for an instance of a node object. It is called by the broker whenever an instance of a node object is constructed. Nodes are constructed when a message flow is deployed by the broker, or when the execution group is started.

"cniIsTerminalAttached" on page 6494
Use this function to check whether a terminal is attached to another node by a connector. It returns an integer value that specifies whether the specified terminal object is attached to one or more terminals on other message flow nodes.

cniDefineNodeClass:

Use this function to define a node class, as specified by the *name* parameter, which is supported by the node factory specified as the*factoryObject* parameter.

This function is called by the node during execution of bipGetMessageFlowNodeFactory, when the LIL file is loaded.

If both cniGetAttribute and cniGetAttribute2, or cniGetAttributeName and cniGetAttributeName2 are implemented, cniDefineNodeClass fails with CCI_INV_IMPL_FUNCTION.

Syntax:

```
void cniDefineNodeClass(
  int*         returnCode,
  CciFactory*  factoryObject,
  CciChar*     name,
  CNI_VFT*     functbl);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_FACTORY_OBJECT
- CCI_INV_NODE_NAME
- CCI_INV_OBJECT_NAME
- CCI_INV_VFTP
- CCI_MISSING_IMPL_FUNCTION
- CCI_NAME_EXISTS

factoryObject
The address of the factory object that supports the named node. The address is returned from cniCreateNodeFactory (input).

name
The name of the node to be defined. The name of the node must end with the word `Node` (input).

For example, if you have assigned the name as Basic in the WebSphere Message Broker Toolkit, the class name of the node must be BasicNode.

functbl
The address of the CNI_VFT structure that contains pointers to the node implementation functions (input). Here is an example of a function table:

```
vftable.iFpCreateNodeContext = _Transform_createNodeContext;
vftable.iFpDeleteNodeContext = _deleteNodeContext;
vftable.iFpGetAttributeName2 = _getAttributeName2;
vftable.iFpSetAttribute      = _setAttribute;
vftable.iFpGetAttribute2     = _getAttribute2;
vftable.iFpEvaluate          = _Transform_evaluate; /* if not an input node */
vftable.iFRun                = _run                 /* if an input node */
```

You would typically define only one of the last 2 entries, that is, you define `vftable.iFpEvaluate = _Transform_evaluate;` for a message processing node, or you define `vftable.iFpRun = _run;` for an input node.

Return values:
None. If an error occurs, the *returnCode* parameter indicates the reason for the error.

Related concepts:

"User-defined input nodes" on page 2990
A user-defined input node is an extension to the broker that provides a new input node in addition to the nodes supplied with the product.

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating an input node in C" on page 3027
Create a user-defined input node in C to receive messages into a message flow.

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniCreateNodeFactory" on page 6449
Use this function to create a node factory in the broker. A single instance of the named message flow node factory is created.

cniDeleteMessage:

Use this function to delete the specified message object. For every call to the cniCreateMessage function, you must include a matching call to cniDeleteMessage to return allocated resources when the processing on the output message has been completed.

Syntax:

```
void cniDeleteMessage(
  int*         returnCode,
  CciMessage*  message);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_MESSAGE_OBJECT

message
The address of the message object to be deleted (input).

Return values:
None. If an error occurs, the *returnCode* parameter indicates the reason for the error.

Example:

```
cniDeleteMessage(0, outMsg);
```

Related concepts:

"User-defined input nodes" on page 2990
A user-defined input node is an extension to the broker that provides a new input node in addition to the nodes supplied with the product.

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to

complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating an input node in C" on page 3027
Create a user-defined input node in C to receive messages into a message flow.

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniCreateMessage" on page 6446
Use this function to create an output message object. For every call to this function, you must include a matching call to cniDeleteMessage to return allocated resources when the processing on the output message has been completed.

"cniGetEnvironmentMessage" on page 6490
Use this function to get the CciMessage object that corresponds to the environment tree for the message flow.

"cniGetMessageContext" on page 6491
Use this function to get the address of the message context that is associated with the specified message. The context of an existing message is used to create an output message; for example, by using the cniCreateMessage function.

"cniFinalize" on page 6479
Use this function to cause the broker to request parsers to perform finalize processing on the specified message. Finalization is a process that fixes header chains and makes the Properties folder match the headers.

"cniPropagate" on page 6501
Use this function to propagate a message to a specified terminal object. If the terminal is not attached to another node by a connector, the message is not propagated, and the function is ignored.

cniDeleteNodeContext:

This function deletes any context for an instance of a user-defined node object. It is called by the broker whenever an instance of a node object is destroyed, when a message flow is deleted, or when a configuration is redeployed.

A message flow node might also be deleted when reconfiguring or redeploying a broker.

The responsibilities of the node are to:

1. Release all node instance specific data areas (such as context) that were acquired at construction or during node processing.
2. Release all additional resources that might have been acquired for the processing of the node.

Defined In	Type	Member
CNI_VFT	Optional	iFpDeleteNodeContext

Syntax:

```
void cniDeleteNodeContext(CciContext* context);
```

Parameters:

context
The address of the context for the instance of the node, as created and returned by the **cniCreateNodeContext** function (input).

Example:

```
void _deleteNodeContext(
  CciContext* context
){
  static char* functionName = (char *)"_deleteNodeContext()";

  return;
}
```

Related concepts:

"User-defined input nodes" on page 2990
A user-defined input node is an extension to the broker that provides a new input node in addition to the nodes supplied with the product.

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating an input node in C" on page 3027
Create a user-defined input node in C to receive messages into a message flow.

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node implementation functions" on page 6417
The user-defined node implements a function interface for the broker to call during runtime operation. This interface includes functions to create a local context whenever a node instance is created, functions to set and retrieve attribute values, the function to perform the processing of the node itself, and functions to examine messages.

"cniCreateNodeContext" on page 6447
This function creates a context for an instance of a node object. It is called by the broker whenever an instance of a node object is constructed. Nodes are constructed when a message flow is deployed by the broker, or when the execution group is started.

cniDetach:

Use this function to detach the specified syntax element from the syntax element tree. The element is detached from its parent and siblings, but all child elements are left attached.

Syntax:

```
void cniDetach(
  int*         returnCode,
  CciElement*  targetElement);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT

targetElement
The address of the syntax element object to be detached (input).

Return values:
None. If an error occurs, the *returnCode* parameter indicates the reason for the error.

Related concepts:

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

cniDispatchThread:

Use this function to dispatch a new message flow thread to call another thread instance to run the user-defined message flow input node.

This message flow thread is allocated from a pool of threads maintained for each message flow, under control of the Additional Instances property of the message flow. If no threads are available because they are all in use, CCI_SUCCESS is returned, and *returnCode* is set to CCI_NO_THREADS_AVAILABLE. This value is not an error, but represents one of the following causes:

- The message flow was not configured to run with additional threads.
- All additional threads configured are currently running.

The cniDispatchThread function can be called only from an input node. If it is called at any other time, CCI_FAILURE is returned and *returnCode* is set to CCI_INV_NODE_ENV.

Syntax:

```
int cniDispatchThread(
  int*        returnCode,
  CciNode*    nodeObject);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_NO_THREADS_AVAILABLE
- CCI_INV_NODE_OBJECT
- CCI_INV_NODE_ENV

nodeObject
The address of the node object that is run when the broker creates or reuses the thread. This parameter is passed to the node when its cniCreateNodeContext implementation function is called (input).

Return values:

- If a thread was successfully allocated, CCI_SUCCESS is returned, and *returnCode* is set to CCI_SUCCESS.
- If a thread could not be dispatched because insufficient threads are available in the message flow thread pool to satisfy the request, CCI_SUCCESS is returned and *returnCode* is set to CCI_NO_THREADS_AVAILABLE.
- If the function was not called in an input node, CCI_FAILURE is returned, and *returnCode* is set to CCI_INV_NODE_ENV.
- For all other error conditions, CCI_FAILURE is returned, and *returnCode* indicates the reason for the error.

Example:

```
cniDispatchThread(&rcDispatch, ((NODE_CONTEXT_ST *)context)->nodeObject);
```

Related concepts:

"User-defined input nodes" on page 2990
A user-defined input node is an extension to the broker that provides a new input node in addition to the nodes supplied with the product.

Related tasks:

"Creating an input node in C" on page 3027
Create a user-defined input node in C to receive messages into a message flow.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniCreateNodeContext" on page 6447
This function creates a context for an instance of a node object. It is called by the broker whenever an instance of a node object is constructed. Nodes are constructed when a message flow is deployed by the broker, or when the execution group is started.

cniElementAsBitstream:

Use this function to get the bitstream representation of the specified element.

The parser that is associated with the element serializes the element and all its child elements. The result is copied to memory allocated by the caller. In the special case where all the options that are specified match those of the original bit stream, for example a bit stream that is read from a WebSphere MQ queue by the MQInput node, and the message has not been modified since receiving the original bit stream, this original bit stream is copied into the allocated memory. In this case, the parser is not required to parse and reserialize the message.

The algorithm that is used to generate the bit stream depends on the parser that is used, and the options that are specified. All parsers support the following modes:

- RootBitStream, in which the algorithm that generates the bit stream is the same as that used by an output node. In this mode, a meaningful result is obtained only if the element pointed to is at the head of a subtree with an appropriate structure.
- EmbeddedBitStream, in which not only is the algorithm that generates the bit stream the same as that used by an output node, but also the following elements are determined, if not explicitly specified, in the same way as the output node. Therefore they are determined by searching the previous siblings of *element* on the assumption that these elements represent headers:
  - Encoding
  - CCSID
  - Message set
  - Message type
  - Message format

  In this way, the algorithm for determining these properties is essentially the same as that used for the ESQL BITSTREAM function.

Some parsers also support another mode, FolderBitStream, which generates a meaningful bit stream for any subtree, provided that the field pointed to represents a folder.

Syntax:

```
CciSize cniElementAsBitstream(
  int*                        returnCode,
  CciElement*                 element,
  const struct CciByteArray* value,
  CciChar*                    messageType,
  CciChar*                    messageSet,
  CciChar*                    messageFormat,
  int                         encoding,
  int                         ccsid,
  int                         options);
```

Parameters:

returnCode
The return code from the function (output). If you specify a NULL pointer on input, the value indicates that the node does not handle errors. If input is not NULL, the output signifies the success status of the call. Any exceptions that are produced during the execution of this call are sent to the next upstream node in the flow. Call cciGetLastExceptionData for details of the exception.

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT
- CCI_INV_DATA_POINTER
- CCI_INV_DATA_BUFLEN
- CCI_INV_BUFFER_TOO_SMALL

element
The syntax element to be serialized (input.)

value
A pointer to a CciByteArray struct that contains a pointer to a region of memory allocated by the caller, and the size in CciBytes of this memory (output).

messageType
The message type definition that is used to create the bit stream from the element tree (input). If you specify a NULL pointer, the parameter is ignored. The parameter is also ignored if the value is not relevant to the parser associated with the element; for example, a generic XML parser.

messageSet
The message set definition that is used to create the bit stream from the element tree (input). If you specify a NULL pointer, the parameter is ignored. The parameter is also ignored if the value is not relevant to the parser associated with the element; for example, a generic XML parser.

messageFormat
The format that is used to create the bit stream from the element tree (input). If you specify a NULL pointer, the parameter is ignored. The parameter is also ignored if the value is not relevant to the parser associated with the element; for example, a generic XML parser.

encoding
The encoding to use when writing the bit stream (input). This parameter is mandatory. You can specify a value of `0` to indicate that the encoding for the queue manager must be used.

ccsid
The coded character set identifier to use when writing the bit stream (input). This parameter is mandatory. If you specify a value of `0`, the CCSID of the queue manager is used. A CCSID of `-1` indicates that the bit stream is generated by using CCSID information contained in the subtree consisting of the field pointed to by the element and its child elements. No supplied parsers support this option.

options
The integer value that specifies which bitstream generation mode must be used. Set one of the following values:

- CCI_BITSTREAM_OPTIONS_ROOT
- CCI_BITSTREAM_OPTIONS_EMBEDDED
- CCI_BITSTREAM_OPTIONS_FOLDER

Return values:

- If successful, the correct size of memory that is required to hold the bit stream is returned.

- If the memory allocated by the caller was insufficient, *returnCode* is CCI_BUFFER_TOO_SMALL.
- If an exception occurs during execution, *returnCode* is CCI_EXCEPTION.

Example:
The following example demonstrates how you can use the options parameter to generate the bit stream for different parts of the message tree.

This code can be copied into the _evaluate function of the sample Transform node. For an input message such as:

```
MQMD
MQRFH2
<test><data><aaa>text</aaa></data></test>
```

the node propagates three messages:

- One that contains a copy of the input message in the BLOB domain
- One that contains a copy of the input MQRFH2 as the message body in the BLOB domain
- One that contains the <data></data> folder as the message body in the BLOB domain

```
CciMessage*          outMsg[3];
  CciTerminal*         terminalObject;
  CciElement*          bodyChild;
  CciElement*          inRootElement;
  CciElement*          inSourceElement[3];
  CciElement*          outRootElement;
  CciElement*          outBlobElement;
  CciElement*          outBody;
  struct CciByteArray  bitstream[3];
  int                  bitstreamOptions[3];
  int                  retvalue;
  int                  rc = 0;
  int                  loopCount;
  CCI_EXCEPTION_ST     exception_st = {CCI_EXCEPTION_ST_DEFAULT};
  const CciChar*       constBLOBParserName  =
                cciString("NONE",BIP_DEF_COMP_CCSID);
  const CciChar*       constBLOBElementName =
                cciString("BLOB",BIP_DEF_COMP_CCSID);
  const CciChar*       constEmptyString     =
                cciString("",BIP_DEF_COMP_CCSID);

  /*build up and propagate 3 output messages*/
  /*first message has bit stream for input message body*/
  /*second message has bit stream for input MQRFH2*/
  /*third message has bit stream for sub element from input message*/

  /* Get the root element of the input message */
  inRootElement = cniRootElement(&rc, message);
  /*CCI_CHECK_RC();*/
  checkRC(rc);

  /*set up the array of source elements and bitstream options*/

  /*message body*/
  inSourceElement[0] =  cniLastChild(&rc,inRootElement);
  checkRC(rc);

  /*This is the root of the message body so we use RootBitStream mode*/
  bitstreamOptions[0] = CCI_BITSTREAM_OPTIONS_ROOT;


  /*last header*/
```

```
inSourceElement[1] = cniPreviousSibling(&rc,inSourceElement[0]);
checkRC(rc);

/*This is the root of the MQRFH2 so we use RootBitStream mode*/
bitstreamOptions[1] = CCI_BITSTREAM_OPTIONS_ROOT;


/*body.FIRST(first child of message body) */
inSourceElement[2] = cniFirstChild(&rc,inSourceElement[0]);
checkRC(rc);

/*body.FIRST.FIRST */
inSourceElement[2] = cniFirstChild(&rc,inSourceElement[2]);
checkRC(rc);

/*This is a sub tree within the message body so we use FolderBitStream mode*/
bitstreamOptions[2] = CCI_BITSTREAM_OPTIONS_FOLDER;


for (loopCount=0;loopCount<3;loopCount++) {
  int bufLength;

  /* Create new message for output */
  outMsg[loopCount] = cniCreateMessage(&rc, cniGetMessageContext(&rc, message));
  checkRC(rc);

  /* Get the root element of the output message */
  outRootElement = cniRootElement(&rc, outMsg[loopCount]);
  checkRC(rc);

  /* Copy the contents of the input message to the output message */
  cniCopyElementTree(&rc, inRootElement, outRootElement);
  checkRC(rc);

  /* Get the last child of root (ie the body)  */
  bodyChild = cniLastChild(&rc, outRootElement);
  checkRC(rc);

  /*throw away the message body which was copied from the input message*/
  cniDetach(&rc,
            bodyChild);
  checkRC(rc);

  /*create the new output message body in the BLOB domain*/
  outBody = cniCreateElementAsLastChildUsingParser(&rc,
                                        outRootElement,
                                        constBLOBParserName);
  checkRC(rc);

  /*create the BLOB element*/
  outBlobElement = cniCreateElementAsLastChild(&rc,
                              outBody);
  checkRC(rc);

  cniSetElementName(&rc,
                    outBlobElement,
                    constBLOBElementName);
  checkRC(rc);

  /*Set the value of the blob element by obtaining the bit stream for the
  element */
  bitstream[loopCount].size=512;
  bitstream[loopCount].pointer=(CciByte*)malloc(sizeof(CciByte) * 512);

  bufLength = cniElementAsBitstream(&rc,
                        inSourceElement[loopCount],
                        &bitstream[loopCount],
```

```
                        constEmptyString,/*assume XML message so no interest in*/
                        constEmptyString,/* type, set or format*/
                        constEmptyString,
                        0,/*Use Queue Manager CCSID & Encoding*/
                        0,
                        bitstreamOptions[loopCount]);


  if (rc==CCI_BUFFER_TOO_SMALL)
  {
      free(bitstream[loopCount].pointer);
      bitstream[loopCount].size=bufLength;
      bitstream[loopCount].pointer=(CciByte*)malloc(sizeof(CciByte) * bitstream[loopCount].size);

      bufLength = cniElementAsBitstream(&rc,
                        inSourceElement[loopCount],
                        &bitstream[loopCount],
                        constEmptyString,/*assume XML message so no interest in*/
                        constEmptyString,/* type, set or format*/
                        constEmptyString,
                        0,/*Use Queue Manager CCSID & Encoding*/
                        0,
                        bitstreamOptions[loopCount]);
  }
  checkRC(rc);
  bitstream[loopCount].size=bufLength;

  cniSetElementByteArrayValue(&rc,
                              outBlobElement,
                              &bitstream[loopCount]);
  checkRC(rc);
}

/* Get handle of output terminal */
terminalObject = getOutputTerminalHandle( (NODE_CONTEXT_ST *)context,
                                          (CciChar*)constOut);

/* If the terminal exists and is attached, propagate to it */
if (terminalObject) {
  if (cniIsTerminalAttached(&rc, terminalObject)) {
    /* As this is a new, and changed message, it should be finalized... */
    cniFinalize(&rc, outMsg[0], CCI_FINALIZE_NONE);
    cniFinalize(&rc, outMsg[1], CCI_FINALIZE_NONE);
    cniFinalize(&rc, outMsg[2], CCI_FINALIZE_NONE);
    retvalue = cniPropagate(&rc, terminalObject, localEnvironment, exceptionList, outMsg[0]);
    retvalue = cniPropagate(&rc, terminalObject, localEnvironment, exceptionList, outMsg[1]);
    retvalue = cniPropagate(&rc, terminalObject, localEnvironment, exceptionList, outMsg[2]);
    if (retvalue == CCI_FAILURE) {
      if (rc == CCI_EXCEPTION) {
        /* Get details of the exception */
        memset(&exception_st, 0, sizeof(exception_st));
        cciGetLastExceptionData(&rc, &exception_st);

        /* Any local error handling can go here */

        /* Ensure message is deleted prior to return/throw */
        cniDeleteMessage(0, outMsg[0]);
        cniDeleteMessage(0, outMsg[1]);
        cniDeleteMessage(0, outMsg[2]);

        /* We must "rethrow" the exception; note this does not return */
        cciRethrowLastException(&rc);
      }
      else {

        /* Some other error...the plugin might choose to log it using the CciLog() */
        /* utility function                                                        */
```

```
      }
    }
    else {
    }
  }
}
else {
  /* Terminal did not exist...severe internal error. The plugin might want to */
  /* log an error here by using the cciLog() utility function.                */
}

/* Delete the messages we created now we have finished with them */
cniDeleteMessage(0, outMsg[0]);
cniDeleteMessage(0, outMsg[1]);
cniDeleteMessage(0, outMsg[2]);

free((void*) constBLOBParserName);
free((void*) constBLOBElementName);
free((void*) constEmptyString);
return;
```

Related concepts:

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"cniElementName" on page 6464
Use this function to get the value of the *name* attribute for the specified syntax element. You must sethe syntax element name previously by using cniSetElementName or cpiSetElementName.

"cniElementType" on page 6467
Use this function to get the value of the *type* attribute for the specified syntax element. You must set the syntax element type previously by using cniSetElementType or cpiSetElementType.

"cniElementValue group" on page 6468
Use one or more of the functions in this group to retrieve the value of the specified syntax element.

"cniElementValueState" on page 6471
Use this function to get the state of the value of the specified syntax element.

"cniElementValueType" on page 6472
Use this function to get the *type* attribute for the value of the specified syntax element. The state of an element after creation is undefined. When the value of the element is set, its state becomes valid.

"cniElementValueValue" on page 6473
Use this function to get the address of the value object owned by the specified syntax element.

"cciGetLastExceptionData" on page 6647
Gets diagnostic information about the last exception generated. Information about the last exception generated on the current thread is returned in a CCI_EXCEPTION_ST output structure. The user-defined extension can use this function to determine whether any recovery is required when a utility function returns an error code.

cniElementName:

Use this function to get the value of the *name* attribute for the specified syntax element. You must sethe syntax element name previously by using cniSetElementName or cpiSetElementName.

Syntax:

```
CciSize cniElementName(
  int*            returnCode,
  CciElement*     targetElement,
  const CciChar*  value,
  Ccisize         length);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT
- CCI_INV_DATA_POINTER
- CCI_INV_DATA_BUFLEN
- CCI_INV_BUFFER_TOO_SMALL

targetElement
The address of the target syntax element object (input).

value
The address of a buffer into which the element name is copied (input).

length
The length, in characters, specified by the *value* parameter (input).

Return values:

- If successful, the element name is copied into the supplied buffer and the number of `CciChar` characters copied is returned.
- If the buffer is not large enough to contain the attribute value, *returnCode* is set to CCI_BUFFER_TOO_SMALL, and the number of CciChars required is returned.
- For any other failures, CCI_FAILURE is returned, and *returnCode* indicates the reason for the error.

Related concepts:

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new

message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniElementAsBitstream" on page 6458
Use this function to get the bitstream representation of the specified element.

"cniElementNamespace"
Use this function to get the value of the *namespace* attribute for the specified syntax element.

"cniElementType" on page 6467
Use this function to get the value of the *type* attribute for the specified syntax element. You must set the syntax element type previously by using cniSetElementType or cpiSetElementType.

"cniElementValue group" on page 6468
Use one or more of the functions in this group to retrieve the value of the specified syntax element.

"cniElementValueState" on page 6471
Use this function to get the state of the value of the specified syntax element.

"cniElementValueType" on page 6472
Use this function to get the *type* attribute for the value of the specified syntax element. The state of an element after creation is undefined. When the value of the element is set, its state becomes valid.

"cniElementValueValue" on page 6473
Use this function to get the address of the value object owned by the specified syntax element.

"cniSetElementName" on page 6513
Use this function to set the name of the specified syntax element.

"cpiSetElementName" on page 6598
This function sets the name of the specified syntax element.

cniElementNamespace:

Use this function to get the value of the *namespace* attribute for the specified syntax element.

You must previously have set the syntax element name by using cniSetElementNamespace or cpiSetElementNamespace.

Use this function when you want to convert a message that belongs to a namespace-aware domain to a bit stream.

Syntax:

```
CciSize cniElementNamespace(
 int*           returnCode,
 CciElement*    targetElement,
 const CciChar* value,
 CciSize        length)
```

Parameters:

returnCode
The return code from the function (output). Specifying a NULL pointer signifies that the node does not want to deal with errors. If input is not NULL, the output signifies the success status of the call. All exceptions thrown during the execution of this call are re-thrown to the next upstream node in the flow. Call cciGetLastExceptionData for details of the exception.

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT
- CCI_INV_DATA_POINTER
- CCI_INV_DATA_BUFLEN
- CCI_INV_BUFFER_TOO_SMALL

targetElement
Specifies the address of the target syntax element object (input).

value
Specifies the address of a buffer into which the element namespace value is copied (output). A string of characters (including a NULL terminator) representing the namespace value is copied into this buffer. The buffer must be a portion of memory previously allocated by the caller.

length
The length, in characters, of the buffer specified by the *value* parameter (input).

Return values:

- If successful, the number of CciChars copied into the buffer is returned.
- If the buffer is not large enough to contain the attribute value, *returnCode* is set to CCI_BUFFER_TOO_SMALL, and the number of CciChars required is returned.
- If an exception occurs during execution, *returnCode* is set to CCI_EXCEPTION.

Example:

```
if (element != 0) {
   /*get name*/
   cniElementName(&rc, element, (CciChar*)&elementName, sizeof(elementName));

   /*get namespace*/
   elementNamespace=(CciChar*)malloc(sizeof(CciChar) * elementNamespaceLength);
   elementNamespaceLength = cniElementNamespace(&rc,
                                                element,
                                                elementNamespace,
                                                elementNamespaceLength);

   if (rc==CCI_BUFFER_TOO_SMALL){
     free(elementNamespace);
     elementNamespace=(CciChar*)malloc(sizeof(CciChar) * elementNamespaceLength);
     elementNamespaceLength = cniElementNamespace(&rc,
                                                  element,
                                                  elementNamespace,
                                                  elementNamespaceLength);
   }
   checkRC(rc);
```

Related concepts:

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"cciGetLastExceptionData" on page 6647
Gets diagnostic information about the last exception generated. Information about the last exception generated on the current thread is returned in a CCI_EXCEPTION_ST output structure. The user-defined extension can use this function to determine whether any recovery is required when a utility function returns an error code.

"cniElementName" on page 6464
Use this function to get the value of the *name* attribute for the specified syntax element. You must sethe syntax element name previously by using cniSetElementName or cpiSetElementName.

"cniSearchElementInNamespace group" on page 6508
Use this element to search for an element that matches the specified criteria.

"cniSetElementNamespace" on page 6514
Use this function to set the *namespace* attribute for the specified syntax element when you manipulate a message that belongs to a namespace-aware domain.

"cpiSetElementNamespace" on page 6599
Use this function to set the namespace attribute for the specified syntax element.

cniElementType:

Use this function to get the value of the *type* attribute for the specified syntax element. You must set the syntax element type previously by using cniSetElementType or cpiSetElementType.

Syntax:

```
CciElementType cniElementType(
  int*         returnCode,
  CciElement*  targetElement);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT

targetElement
The address of the target syntax element object (input).

Return values:

The value of the target element type is returned. If an error occurs, CCI_FAILURE is returned, and the *returnCode* parameter indicates the reason for the error.

Related concepts:

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniElementAsBitstream" on page 6458
Use this function to get the bitstream representation of the specified element.

"cniElementName" on page 6464
Use this function to get the value of the *name* attribute for the specified syntax element. You must sethe syntax element name previously by using cniSetElementName or cpiSetElementName.

"cniElementValue group"
Use one or more of the functions in this group to retrieve the value of the specified syntax element.

"cniElementValueState" on page 6471
Use this function to get the state of the value of the specified syntax element.

"cniElementValueType" on page 6472
Use this function to get the *type* attribute for the value of the specified syntax element. The state of an element after creation is undefined. When the value of the element is set, its state becomes valid.

"cniElementValueValue" on page 6473
Use this function to get the address of the value object owned by the specified syntax element.

"cniSetElementType" on page 6515
Use this function to set the type of the specified syntax element.

"cpiSetElementType" on page 6601
This function sets the type of the specified syntax element.

cniElementValue group:

Use one or more of the functions in this group to retrieve the value of the specified syntax element.

Specify the appropriate function from this group that matches the type of data to be retrieved:

- cniElementBitArrayValue
- cniElementBooleanValue
- cniElementByteArrayValue

- cniElementCharacterValue
- cniElementDateValue
- cniElementDecimalValue
- cniElementGmtTimestampValue
- cniElementGmtTimeValue
- cniElementIntegerValue
- cniElementRealValue
- cniElementTimestampValue
- cniElementTimeValue

Syntax:

```
CciSize cniElementBitArrayValue(
  int*           returnCode,
  CciElement*    targetElement,
  const struct   CciBitArray* value);

CciBool cniElementBooleanValue(
  int*           returnCode,
  CciElement*    targetElement);

CciSize cniElementByteArrayValue(
  int*           returnCode,
  CciElement*    targetElement,
  const struct   CciByteArray* value);

CciSize cniElementCharacterValue(
  int*           returnCode,
  CciElement*    targetElement,
  const CciChar* value,
  CciSize        length);

struct CciDate cniElementDateValue(
  int*           returnCode,
  CciElement*    targetElement);

CciSize cniElementDecimalValue(
  int*           returnCode,
  CciElement*    targetElement,
  const CciChar* value,
  CciSize        length);

struct CciTimestamp cniElementGmtTimestampValue(
  int*           returnCode,
  CciElement*    targetElement);

struct CciTime cniElementGmtTimeValue(
  int*           returnCode,
  CciElement*    targetElement);

CciInt cniElementIntegerValue(
  int*           returnCode,
  CciElement*    targetElement);

CciReal cniElementRealValue(
  int*           returnCode,
  CciElement*    targetElement);

struct CciTimestamp cniElementTimestampValue(
  int*           returnCode,
  CciElement*    targetElement);

struct CciTime cniElementTimeValue(
  int*           returnCode,
  CciElement*    targetElement);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT
- CCI_INV_DATA_POINTER
- CCI_INV_DATA_BUFLEN
- CCI_INV_BUFFER_TOO_SMALL

targetElement
The address of the target syntax element object (input).

value
The address of an output buffer into which the value of the syntax element is stored (input). Used on relevant function calls only.

length
The length of the output buffer, in characters, specified by the *value* parameter (input). Used on relevant function calls only.

Return values:

- If successful, the value of the target element is returned.
- If the size of an element's data can vary, the correct data size is returned.
- If the specified length is too small, the error code is set to CCI_BUFFER_TOO_SMALL.
- If an error occurs, the *returnCode* parameter indicates the reason for the error.

Example:

```
 numberOfChars = cniElementCharacterValue(
      &rc, firstChild, (CciChar*)&elementValue, sizeof(elementValue)
      );

if (rc==CCI_BUFFER_TOO_SMALL) {
    free(elementValue);
    elementValue      = (CciChar*)malloc(numberOfChars * sizeof(CciChar));
    numberOfChars     = cniElementCharacterValue(
                  &rc, firstChild, (CciChar*)&elementValue, sizeof(elementValue));
  }
```

Related concepts:

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniElementAsBitstream" on page 6458
Use this function to get the bitstream representation of the specified element.

"cniElementName" on page 6464
Use this function to get the value of the *name* attribute for the specified syntax element. You must sethe syntax element name previously by using cniSetElementName or cpiSetElementName.

"cniElementType" on page 6467
Use this function to get the value of the *type* attribute for the specified syntax element. You must set the syntax element type previously by using cniSetElementType or cpiSetElementType.

"cniElementValueState"
Use this function to get the state of the value of the specified syntax element.

"cniElementValueType" on page 6472
Use this function to get the *type* attribute for the value of the specified syntax element. The state of an element after creation is undefined. When the value of the element is set, its state becomes valid.

"cniElementValueValue" on page 6473
Use this function to get the address of the value object owned by the specified syntax element.

"cniSetElementValue group" on page 6516
Use one or more of the functions in this group to set the value of the specified syntax element.

cniElementValueState:

Use this function to get the state of the value of the specified syntax element.

Syntax:

```
CciValueState cniElementValueState(
  int*         returnCode,
  CciElement*  targetElement);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT

targetElement
The address of the target syntax element object (input).

Return values:
The state of the value of the target syntax element is returned. If an error occurs, CCI_VALUE_STATE_UNDEFINED is returned, and the *returnCode* parameter indicates the reason for the error.

Related concepts:

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new

message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniElementAsBitstream" on page 6458
Use this function to get the bitstream representation of the specified element.

"cniElementName" on page 6464
Use this function to get the value of the *name* attribute for the specified syntax element. You must sethe syntax element name previously by using cniSetElementName or cpiSetElementName.

"cniElementType" on page 6467
Use this function to get the value of the *type* attribute for the specified syntax element. You must set the syntax element type previously by using cniSetElementType or cpiSetElementType.

"cniElementValue group" on page 6468
Use one or more of the functions in this group to retrieve the value of the specified syntax element.

"cniElementValueType"
Use this function to get the *type* attribute for the value of the specified syntax element. The state of an element after creation is undefined. When the value of the element is set, its state becomes valid.

"cniElementValueValue" on page 6473
Use this function to get the address of the value object owned by the specified syntax element.

cniElementValueType:

Use this function to get the *type* attribute for the value of the specified syntax element. The state of an element after creation is undefined. When the value of the element is set, its state becomes valid.

Syntax:

```
CciValueType cniElementValueType(
  int*         returnCode,
  CciElement*  targetElement);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT

targetElement
The address of the target syntax element object (input).

Return values:
The type of the value of the target syntax element is returned. If an error occurs, CCI_ELEMENT_TYPE_UNKNOWN is returned, and the *returnCode* parameter indicates the reason for the error.

Related concepts:

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniElementAsBitstream" on page 6458
Use this function to get the bitstream representation of the specified element.

"cniElementName" on page 6464
Use this function to get the value of the *name* attribute for the specified syntax element. You must sethe syntax element name previously by using cniSetElementName or cpiSetElementName.

"cniElementType" on page 6467
Use this function to get the value of the *type* attribute for the specified syntax element. You must set the syntax element type previously by using cniSetElementType or cpiSetElementType.

"cniElementValue group" on page 6468
Use one or more of the functions in this group to retrieve the value of the specified syntax element.

"cniElementValueState" on page 6471
Use this function to get the state of the value of the specified syntax element.

"cniElementValueValue"
Use this function to get the address of the value object owned by the specified syntax element.

cniElementValueValue:

Use this function to get the address of the value object owned by the specified syntax element.

Syntax:

```
const CciElementValue* cniElementValueValue(
  int*         returnCode,
  CciElement*  targetElement);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT
- CCI_INV_DATA_POINTER

targetElement
The address of the target syntax element object (input).

Return values:
The address of the value object of the target syntax element is returned. If an error occurs, zero (CCI_NULL_ADDR) is returned, and the *returnCode* parameter indicates the reason for the error.

Related concepts:

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniElementAsBitstream" on page 6458
Use this function to get the bitstream representation of the specified element.

"cniElementName" on page 6464
Use this function to get the value of the *name* attribute for the specified syntax element. You must sethe syntax element name previously by using cniSetElementName or cpiSetElementName.

"cniElementType" on page 6467
Use this function to get the value of the *type* attribute for the specified syntax element. You must set the syntax element type previously by using cniSetElementType or cpiSetElementType.

"cniElementValue group" on page 6468
Use one or more of the functions in this group to retrieve the value of the specified syntax element.

"cniElementValueState" on page 6471
Use this function to get the state of the value of the specified syntax element.

"cniElementValueType" on page 6472
Use this function to get the *type* attribute for the value of the specified syntax element. The state of an element after creation is undefined. When the value of the element is set, its state becomes valid.

"cniSetElementValueValue" on page 6519
Use this function to set the value object of the specified syntax element.

cniEvaluate:

This function performs node processing. The broker calls this function when a message is received on one of the input terminals of an instance of a node object.

The function forms the main logic of the message processing node or output node. It is not used with input nodes. You must define a function table before you call this function.

The responsibilities of the node at this point are to:

1. Process the message in accordance with the values of the attributes on the node instance.
2. Process the message based on content, if desired.
3. Propagate the message to any appropriate output terminals.
4. Throw an exception if an error occurs.

Defined In	Type	Member
CNI_VFT	Conditional	iFpEvaluate

Syntax:

```
void cniEvaluate(
  CciContext  *context,
  CciMessage  *localEnvironment,
  CciMessage  *exceptionList,
  CciMessage  *message);
```

Parameters:

context
The address of the context for the instance of the node, as created by the node and returned by the cniCreateNodeContext function (input).

localEnvironment
The address of the input local environment object (input).

For compatibility with earlier versions, you can refer to this parameter as `destinationList`.

exceptionList
The address of the exception list for the message (input).

message
The address of the input message object (input).

Related concepts:

"User-defined input nodes" on page 2990
A user-defined input node is an extension to the broker that provides a new input node in addition to the nodes supplied with the product.

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an

output node is used to produce a message as a bit stream.

Related reference:

"C node implementation functions" on page 6417
The user-defined node implements a function interface for the broker to call during runtime operation. This interface includes functions to create a local context whenever a node instance is created, functions to set and retrieve attribute values, the function to perform the processing of the node itself, and functions to examine messages.

"cniCreateNodeContext" on page 6447
This function creates a context for an instance of a node object. It is called by the broker whenever an instance of a node object is constructed. Nodes are constructed when a message flow is deployed by the broker, or when the execution group is started.

cniGetComplexAttribute:

Use cniGetComplexAttribute to locate the user-defined properties of the node within the deployment message.

Returns a pointer to the part of the deployment message representing the user-defined properties of the node.

Syntax:

```
CciElement* cniGetComplexAttribute(
  int*                  returnCode,
  CciNode*              nodeObject,
  CciChar*              attributeName);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION

nodeObject
The name of the node for which attributes are required (input).

attributeName
The name and value of the user-defined attribute for the selected node.

Return values:
Returns a syntax element tree that can be navigated to extract the names and values of all user-defined properties for that node, or NULL if no user-defined properties exist for this node. If an error occurs, the *returnCode* parameter indicates the reason for the error.

Related concepts:

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniGetOutputTerminal"
Use cniGetOutputTerminal to locate a named output terminal, it if exists.

"cniGetResourceProperty" on page 6478
Use the `resourceManager` to obtain the value of the property.

cniGetOutputTerminal:

Use cniGetOutputTerminal to locate a named output terminal, it if exists.

Returns a pointer to the named output terminal, or NULL if a terminal of this name does not exist.

Syntax:

```
CciTerminal* cniGetOutputTerminal(
  int*                returnCode,
  CciNode*            nodeObject,
  CciChar*            name);
```

Parameters:

`returnCode`
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION

`nodeObject`
The name of the node (input).

`name`
The name of the output terminal (input).

Return values:
A pointer to the named output terminal, or NULL if a terminal of this name does not exist. If an error occurs, the *returnCode* parameter indicates the reason for the error.

Related concepts:

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniGetComplexAttribute" on page 6476
Use cniGetComplexAttribute to locate the user-defined properties of the node within the deployment message.

"cniGetResourceProperty"
Use the `resourceManager` to obtain the value of the property.

cniGetResourceProperty:

Use the `resourceManager` to obtain the value of the property.

Obtains the value of the property `propertyName` as managed by the `resourceManager`

Syntax:

```
CciSize cniGetResourceProperty(
  int*              returnCode,
  CciChar*          resourceManager,
  CciChar*          propertyName,
  CciChar*          value,
  CciSize           length);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_BUFFER_TOO_SMALL

resourceManager
The address of the resource manager controlling the properties (input).

propertyName
The name of the object for which the value is required.

value
The value of the buffer, already allocated, that you use.

length
The size of the buffer, already allocated, that you use.

If this size is not sufficient, the function fails with *returnCode* set to CCI_BUFFER_TOO_SMALL, and returns the required buffer size; that is, the string length plus one null terminator character.

You should then allocate the required size for the buffer, and call this function again; this function copies the property string into this buffer.

This usage pattern allows you to manage the memory on your system.

This function performs the same task as the **mqsireportproperties** command.

Return values:

None or CCI_BUFFER_TOO_SMALL. If any other error occurs, the *returnCode* parameter indicates the reason for the error.

Related concepts:

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniGetOutputTerminal" on page 6477
Use cniGetOutputTerminal to locate a named output terminal, it if exists.

"cniGetComplexAttribute" on page 6476
Use cniGetComplexAttribute to locate the user-defined properties of the node within the deployment message.

"**mqsireportproperties** command" on page 3937
Use the **mqsireportproperties** command to display properties that relate to a broker, an execution group, or a configurable service.

cniFinalize:

Use this function to cause the broker to request parsers to perform finalize processing on the specified message. Finalization is a process that fixes header chains and makes the Properties folder match the headers.

The behavior of this processing is specific to each parser. Some parsers do not support finalization processing.

Call cniFinalize before you propagate a message from the node; for example, before you call cniWriteBuffer.

Syntax:

```
void cniFinalize(
  int*         returnCode,
  CciMessage*  message,
  int          options);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_MESSAGE_OBJECT

message
The address of the message object for which the element tree is to be finalized (input).

options
Set this parameter to CCI_FINALIZE_NONE.

Return values:
None. If an error occurs, the *returnCode* parameter indicates the reason for the error.

Example:

```
cniFinalize(&rc, outMsg, CCI_FINALIZE_NONE);
retvalue = cniPropagate(
                        &rc,
                        terminalObject,
                        localEnvironment,
                        exceptionList,
                        outMsg);

/* Handle errors */
```

Related concepts:

"User-defined input nodes" on page 2990
A user-defined input node is an extension to the broker that provides a new input node in addition to the nodes supplied with the product.

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating an input node in C" on page 3027
Create a user-defined input node in C to receive messages into a message flow.

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniCreateMessage" on page 6446
Use this function to create an output message object. For every call to this function, you must include a matching call to cniDeleteMessage to return allocated resources when the processing on the output message has been completed.

"cniDeleteMessage" on page 6453
Use this function to delete the specified message object. For every call to the cniCreateMessage function, you must include a matching call to cniDeleteMessage to return allocated resources when the processing on the output message has been completed.

"cniGetEnvironmentMessage" on page 6490
Use this function to get the CciMessage object that corresponds to the environment tree for the message flow.

"cniGetMessageContext" on page 6491
Use this function to get the address of the message context that is associated with the specified message. The context of an existing message is used to create an output message; for example, by using the cniCreateMessage function.

"cniPropagate" on page 6501
Use this function to propagate a message to a specified terminal object. If the terminal is not attached to another node by a connector, the message is not propagated, and the function is ignored.

"cniWriteBuffer" on page 6537
Use this function to write the syntax element tree associated with the specified message to the data buffer that is owned by the message object. This function is typically used by output nodes.

cniFirstChild:

Use this function to retrieve the address of the syntax element object that is the first child of the specified syntax element.

Syntax:

```
CciElement* cniFirstChild(
  int*         returnCode,
  CciElement*  targetElement);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT

targetElement
The address of the target syntax element object (input).

Return values:

- If successful, the address of the requested syntax element object is returned.
- If there is no first child, zero is returned, and *returnCode* is set to CCI_SUCCESS.
- If an error occurs, zero (CCI_NULL_ADDR) is returned, and the *returnCode* parameter indicates the reason for the error.

Example:

```
if (element != 0) {
  cniElementName(&rc, element, (CciChar*)&elementName, sizeof(elementName));
  firstChild = cniFirstChild(&rc, element);
```

Related concepts:

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniLastChild" on page 6496
Use this function to retrieve the address of the syntax element object that is the last child of the specified syntax element.

"cniNextSibling" on page 6497
Use this function to retrieve the address of the syntax element object that is the next sibling (right sibling) of the specified syntax element.

"cniParent" on page 6498
Use this function to retrieve the address of the syntax element object that is the parent of the specified syntax element.

"cniPreviousSibling" on page 6499
Use this function to retrieve the address of the syntax element object that is the previous sibling (left sibling) of the specified syntax element.

"cniRootElement" on page 6503
Use this function to get the root syntax element associated with a specified message. It returns the root element that is associated with (and owned by) the message object identified by the *message* parameter. When a message object is constructed by the broker, a root element is automatically created.

"cniSearchElement group" on page 6506
Use one or more of the functions in this group to search previous siblings of the specified element for an element that matches the specified criteria.

cniGetAttribute:

This function gets the value of an attribute on a specific node instance.

Restriction: This function imposes a restriction on the length of the attribute value. This function is provided only for compatibility with earlier versions. You should implement cniGetAttribute2.

This function is invoked by the broker:

- Before the nodes configuration is deployed in order to ascertain default values of any attributes that might override attributes owned by the framework.
- After setting the deployed configuration in order to write the configuration to the broker's database. This call ensures that the configuration persists across shutdown and restarts of the execution group

The responsibilities of the node are to:

1. Return a character representation of the attribute value.
2. Throw an exception if an error occurs.

If both cniGetAttribute and cniGetAttribute2 are implemented, cniDefineNodeClass fails with CCI_INV_IMPL_FUNCTION.

Defined In	Type	Member
CNI_VFT	Optional	iFpGetAttribute

Syntax:

```
int cniGetAttribute(
  CciContext*  context,
  CciChar*     attrName,
  CciChar*     buffer,
  int          bufsize);
```

Parameters:

context
The address of the context for the instance of the node, as created by the node and returned by the **cniCreateNodeContext** function (input).

attrName
The name of the attribute for which the value is to be retrieved (input).

buffer
The address of a buffer into which the attribute value is copied (output).

bufsize
The length, in bytes, of the buffer specified in the *buffer* parameter (input).

Return values:
If successful, zero is returned, and the character representation of the value of the attribute is returned in the specified buffer. If the name of the attribute does not identify one supported by the node, a non-zero value is returned.

Related concepts:

"User-defined input nodes" on page 2990
A user-defined input node is an extension to the broker that provides a new input node in addition to the nodes supplied with the product.

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating an input node in C" on page 3027
Create a user-defined input node in C to receive messages into a message flow.

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node implementation functions" on page 6417
The user-defined node implements a function interface for the broker to call during runtime operation. This interface includes functions to create a local context whenever a node instance is created, functions to set and retrieve attribute values, the function to perform the processing of the node itself, and functions to examine messages.

"cniCreateNodeContext" on page 6447
This function creates a context for an instance of a node object. It is called by the broker whenever an instance of a node object is constructed. Nodes are constructed when a message flow is deployed by the broker, or when the execution group is started.

"cniGetAttribute2" on page 6484
Use this function to get the value of an attribute on a specific node instance. It is

called by the broker after all the attributes that the user deploys are set.

cniGetAttribute2:

Use this function to get the value of an attribute on a specific node instance. It is called by the broker after all the attributes that the user deploys are set.

The results are written to the broker persistent configuration data to ensure that the node is configured correctly after the execution group process is stopped and started.

The responsibilities of the node are to:

1. Return a character representation of the attribute value.
2. Throw an exception if an error occurs.

If both cniGetAttribute and cniGetAttribute2 functions are implemented, cniDefineNodeClass fails with return code CCI_INV_IMPL_FUNCTION.

Defined In	Type	Member
CNI_VFT	Optional	iFpGetAttribute2

Syntax:

```
CciSize cniGetAttribute2(
  int          returnCode,
  CciContext*  context,
  CciChar*     attrName,
  CciChar*     buffer,
  int          bufsize);
```

Parameters:

context
The address of the context for the instance of the node, as created by the node and returned by the `cniCreateNodeContext` function (input).

returnCode (output)
Pointer to an int. On return, the node must ensure that this int stores a value that describes the status of completion. Possible return codes are:

- CCI_SUCCESS
- CCI_ATTRIBUTE_UNKNOWN
- CCI_BUFFER_TOO_SMALL

attrName
The name of the attribute for which the value is to be retrieved (input).

buffer
The address of a buffer into which the attribute value is copied (output).

bufsize
The length, in `CciChars`, of the buffer specified in the **buffer** parameter (input).

Return values:

- If successful, the attribute value is copied into the supplied buffer and the number of `CciChar` characters copied is returned.
- If the buffer is not large enough to contain the attribute value, *returnCode* is set to CCI_BUFFER_TOO_SMALL, and the number of `CciChars` required is returned.

- If the *attrName* is not known to this node, *returnCode* is set to CCI_ATTRIBUTE_UNKNOWN.

Related concepts:

"User-defined input nodes" on page 2990
A user-defined input node is an extension to the broker that provides a new input node in addition to the nodes supplied with the product.

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating an input node in C" on page 3027
Create a user-defined input node in C to receive messages into a message flow.

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node implementation functions" on page 6417
The user-defined node implements a function interface for the broker to call during runtime operation. This interface includes functions to create a local context whenever a node instance is created, functions to set and retrieve attribute values, the function to perform the processing of the node itself, and functions to examine messages.

"cniCreateNodeContext" on page 6447
This function creates a context for an instance of a node object. It is called by the broker whenever an instance of a node object is constructed. Nodes are constructed when a message flow is deployed by the broker, or when the execution group is started.

"cniGetAttributeName2" on page 6487
This function returns the name of a node attribute specified by an index. It is called by the broker when the broker requires the names of the attributes that are supported by an instance of a node.

cniGetAttributeName:

This functions returns the name of a node attribute specified by an index.

Restriction: This function imposes a restriction on the length of the attribute value. This function is provided only for compatibility with earlier versions. You should implement cniGetAttributeName2.

This function is invoked by the message broker when the broker requires the names of attributes supported by a particular instance of a node. The function must guarantee to return the attributes in a known, defined order, and to return the attribute name represented by the index parameter.

If both cniGetAttributeName and cniGetAttributeName2 are implemented, cniDefineNodeClass fails with CCI_INV_IMPL_FUNCTION.

Defined In	Type	Member
CNI_VFT	Optional	iFpGetAttributeName

Syntax:

```
int cniGetAttributeName(
  CciContext*  context,
  int          index,
  CciChar*     buffer,
  int          bufsize);
```

Parameters:

context
The address of the context for the instance of the node, as created by the node and returned by the **cniCreateNodeContext** function (input).

index
Specifies the index of the attribute name (input). The index of the attributes starts from zero.

buffer
The address of a buffer into which the attribute name is copied (output).

bufsize
The length, in bytes, of the buffer specified in the *buffer* parameter (input).

Return values:
If successful, zero is returned, and the name of the attribute is returned in the specified buffer. If the end of the list of attributes is reached, a non-zero value is returned.

Related concepts:

"User-defined input nodes" on page 2990
A user-defined input node is an extension to the broker that provides a new input node in addition to the nodes supplied with the product.

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating an input node in C" on page 3027
Create a user-defined input node in C to receive messages into a message flow.

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node implementation functions" on page 6417
The user-defined node implements a function interface for the broker to call during runtime operation. This interface includes functions to create a local context whenever a node instance is created, functions to set and retrieve attribute values, the function to perform the processing of the node itself, and functions to examine messages.

"cniCreateNodeContext" on page 6447
This function creates a context for an instance of a node object. It is called by the broker whenever an instance of a node object is constructed. Nodes are constructed when a message flow is deployed by the broker, or when the execution group is

started.

"cniGetAttributeName2"
This function returns the name of a node attribute specified by an index. It is called by the broker when the broker requires the names of the attributes that are supported by an instance of a node.

cniGetAttributeName2:

This function returns the name of a node attribute specified by an index. It is called by the broker when the broker requires the names of the attributes that are supported by an instance of a node.

The function must return the attributes in a known, defined order, and to return the attribute name that is represented by the index parameter.

If both cniGetAttributeName and cniGetAttributeName2 are implemented, cniDefineNodeClass fails with CCI_INV_IMPL_FUNCTION.

Defined In	Type	Member
CNI_VFT	Optional	iFpGetAttributeName2

Syntax:

```
CciSize cniGetAttributeName2(
  int          returnCode,
  CciContext*  context,
  int          index,
  CciChar*     buffer,
  int          bufsize);
```

Parameters:

context
The address of the context for the instance of the node, as created by the node and returned by the **cniCreateNodeContext** function (input).

returnCode (output)
Pointer to an int. On return, the node must ensure that this int stores a value that describes the status of completion. Possible return codes are:

- CCI_SUCCESS
- CCI_ATTRIBUTE_UNKNOWN
- CCI_BUFFER_TOO_SMALL

index
Specifies the index of the attribute name (input). The index of the attributes starts from zero.

buffer
The address of a buffer into which the attribute name is copied (output).

bufsize
The length, in CciChars, of the buffer specified in the **buffer** parameter (input).

Return values:

- If successful, the attribute name is copied into the supplied buffer and the number of `CciChar` characters copied is returned.

- If the buffer is not large enough to contain the attribute name, *returnCode* is set to CCI_BUFFER_TOO_SMALL, and the number of `CciChars` required is returned.
- If the end of the list of attributes is reached, and the attribute name is not found, *returnCode* is set to CCI_ATTRIBUTE_UNKNOWN. For example, when `index` is greater than n-1, where n is the number of attributes for this node.

Related concepts:

"User-defined input nodes" on page 2990
A user-defined input node is an extension to the broker that provides a new input node in addition to the nodes supplied with the product.

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating an input node in C" on page 3027
Create a user-defined input node in C to receive messages into a message flow.

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node implementation functions" on page 6417
The user-defined node implements a function interface for the broker to call during runtime operation. This interface includes functions to create a local context whenever a node instance is created, functions to set and retrieve attribute values, the function to perform the processing of the node itself, and functions to examine messages.

"cniCreateNodeContext" on page 6447
This function creates a context for an instance of a node object. It is called by the broker whenever an instance of a node object is constructed. Nodes are constructed when a message flow is deployed by the broker, or when the execution group is started.

"cniGetAttribute2" on page 6484
Use this function to get the value of an attribute on a specific node instance. It is called by the broker after all the attributes that the user deploys are set.

cniGetBrokerInfo:

Use this function to query the current broker environment (for example, for information about broker name and message flow name). The information is returned in a structure of type CNI_BROKER_INFO_ST.

Syntax:

```
void cniGetBrokerInfo(
  int*                returnCode,
  CciNode*            nodeObject,
  CNI_BROKER_INFO_ST* broker_info_st);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_NODE_OBJECT

nodeObject
The message flow processing node for which broker environment information is being requested (input).

broker_info_st
The address of a CNI_BROKER_INFO_ST structure that is used to return a message that represents the input destination (input):

```
typedef struct broker_info_st {
int versionId;   /*Structure version identification*/
CCI_STRING_ST brokerName;  /*The label of the broker*/
CCI_STRING_ST executionGroupName; /*The label of the current execution group*/
CCI_STRING_ST messageFlowName;  /*The label of the current message flow*/
CCI_STRING_ST queueManagerName;  /*The name of the MQ Queue Manager for the broker*/
int commitCount;   /*Commit count value*/
int commitInterval;   /*Commit interval value*/
int coordinatedTransaction;  /*Flag: coordinatedTransaction: 0=no, 1=yes*/
CCI_STRING_ST dataSourceUserId;  /*The user ID that the broker uses to connect to the data sou
} CNI_BROKER_INFO_ST;
```

Note: The dataSourceUserId parameter returns an empty string. The parameter is included to maintain compatibility with previous versions of WebSphere Message Broker that require a broker database.

Return values:
None. If an error occurs, the **returnCode** parameter indicates the reason for the error.

Example:

```
cniGetBrokerInfo(&rc, nodeObject, &broker_info_st);
```

where nodeObject is of type CciNode*

Related concepts:

"User-defined input nodes" on page 2990
A user-defined input node is an extension to the broker that provides a new input node in addition to the nodes supplied with the product.

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating an input node in C" on page 3027
Create a user-defined input node in C to receive messages into a message flow.

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

cniGetEnvironmentMessage:

Use this function to get the CciMessage object that corresponds to the environment tree for the message flow.

Syntax:

```
CciMessage ImportExportPrefix * ImportExportSuffix
  cniGetEnvironmentMessage(
    int*         returnCode,
    CciMessage*  message);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_MESSAGE_OBJECT

message
The address of the message object for which the environment is to be obtained. This message might be an input message that has been received as an argument to the cniEvaluate implementation function, or a message created by the cniCreateMessage utility function.

Return values:
If successful, the address of the message object corresponding to the environment tree is returned. Otherwise, a value of zero is returned, and the *returnCode* parameter indicates the reason for the error.

Related concepts:

"User-defined input nodes" on page 2990
A user-defined input node is an extension to the broker that provides a new input node in addition to the nodes supplied with the product.

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating an input node in C" on page 3027
Create a user-defined input node in C to receive messages into a message flow.

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node implementation functions" on page 6417
The user-defined node implements a function interface for the broker to call during runtime operation. This interface includes functions to create a local context whenever a node instance is created, functions to set and retrieve attribute values, the function to perform the processing of the node itself, and functions to examine messages.

"cniCreateMessage" on page 6446
Use this function to create an output message object. For every call to this function, you must include a matching call to cniDeleteMessage to return allocated resources when the processing on the output message has been completed.

"cniDeleteMessage" on page 6453
Use this function to delete the specified message object. For every call to the cniCreateMessage function, you must include a matching call to cniDeleteMessage to return allocated resources when the processing on the output message has been completed.

"cniEvaluate" on page 6475
This function performs node processing. The broker calls this function when a message is received on one of the input terminals of an instance of a node object.

"cniFinalize" on page 6479
Use this function to cause the broker to request parsers to perform finalize processing on the specified message. Finalization is a process that fixes header chains and makes the Properties folder match the headers.

"cniGetMessageContext"
Use this function to get the address of the message context that is associated with the specified message. The context of an existing message is used to create an output message; for example, by using the cniCreateMessage function.

"cniPropagate" on page 6501
Use this function to propagate a message to a specified terminal object. If the terminal is not attached to another node by a connector, the message is not propagated, and the function is ignored.

cniGetMessageContext:

Use this function to get the address of the message context that is associated with the specified message. The context of an existing message is used to create an output message; for example, by using the cniCreateMessage function.

Syntax:

```
CciMessageContext* cniGetMessageContext(
  int*         returnCode,
  CciMessage*  message);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_INV_MESSAGE_OBJECT

message
The address of the message object (input).

Return values:

If successful, the address of the message context is returned. Otherwise, zero (CCI_NULL_ADDR) is returned, and the *returnCode* parameter indicates the reason for the error.

Example:

```
outMsg = cniCreateMessage(&rc, cniGetMessageContext(&rc, message));
```

Related concepts:

"User-defined input nodes" on page 2990
A user-defined input node is an extension to the broker that provides a new input node in addition to the nodes supplied with the product.

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating an input node in C" on page 3027
Create a user-defined input node in C to receive messages into a message flow.

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniCreateMessage" on page 6446
Use this function to create an output message object. For every call to this function, you must include a matching call to cniDeleteMessage to return allocated resources when the processing on the output message has been completed.

"cniDeleteMessage" on page 6453
Use this function to delete the specified message object. For every call to the cniCreateMessage function, you must include a matching call to cniDeleteMessage to return allocated resources when the processing on the output message has been completed.

"cniFinalize" on page 6479
Use this function to cause the broker to request parsers to perform finalize processing on the specified message. Finalization is a process that fixes header chains and makes the Properties folder match the headers.

"cniGetEnvironmentMessage" on page 6490
Use this function to get the CciMessage object that corresponds to the environment tree for the message flow.

"cniPropagate" on page 6501
Use this function to propagate a message to a specified terminal object. If the terminal is not attached to another node by a connector, the message is not propagated, and the function is ignored.

cniGetParserClassName:

Use this function to get the parser class name that is associated with the specified syntax element.

Syntax:

```
CciSize cniGetParserClassName(
  int*           returnCode,
  CciElement*    targetElement,
  const CciChar* value,
  CciSize        length);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT
- CCI_INV_DATA_POINTER
- CCI_INV_DATA_BUFLEN
- CCI_INV_BUFFER_TOO_SMALL

targetElement
The address of the element for which the parser class name is to be returned (input).

value
The address of an output buffer into which the parser class name is stored (input).

length
The length of the output buffer, expressed as the number of CciChar characters, specified in the *value* parameter (input).

Return values:

- If successful, the *returnCode* parameter indicates CCI_SUCCESS, and the number of characters written to the buffer is returned.
- If the buffer is not large enough to retain the returned name, the *returnCode* parameter indicates CCI_BUFFER_TOO_SMALL, and the returned value indicates the number of characters required to store the name.
- If other errors occur, CCI_FAILURE is returned, and the *returnCode* parameter indicates the reason for the error.

Related concepts:

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using

system-provided functions.

cniGetThreadContext:

This function returns the thread context for the current thread.

Syntax:

```
CciThreadContext *cniGetThreadContext(
  int                    *returnCode,
  CciMessageContext      *msgContext);
```

Parameters:

returnCode
This parameter is the return code from the function (output). If the input value is NULL, this value signifies that errors are silently handled, or are ignored by the broker. If the input value is not NULL, the output signifies the success status of the call. If the `msgContext` parameter is not valid, `*returnCode` is set to CCI_INV_MESSAGE_CONTEXT, and a NULL CciThreadContext is returned.

msgContext
This parameter provides the message context from which to acquire the thread-specific context. It is expected that this parameter is obtained by using the cniGetMessageContext utility function.

Return values:
If this function is successful, it returns a handle to the CciThreadContext for the current thread.

The cciMessageContext value must correspond to a cciMessage, where the cciMessage is passed in to the cniEvaluate or cniRun function on the current thread.

Example:

```
CciMessageContext* messageContext = cniGetMessageContext(NULL,message);
CciThreadContext*  threadContext  = cniGetThreadContext(NULL,messageContext);
```

Related reference:

"cciRegisterForThreadStateChange" on page 6656
This function registers a function to be called when the current thread enters a particular state.

"cciRegCallback" on page 6641
This function can be registered as a callback function, and is called when the registered event occurs. The function is registered by providing a function pointer which matches a particular typedef.

cniIsTerminalAttached:

Use this function to check whether a terminal is attached to another node by a connector. It returns an integer value that specifies whether the specified terminal object is attached to one or more terminals on other message flow nodes.

Use this function to test whether a message can be propagated to a terminal; you do not have to call this function before you propagate a message with the cniPropagate utility function. Use the cniIsTerminalAttached function to modify the node behavior when a terminal is not connected.

Syntax:

```
int cniIsTerminalAttached(
  int*          returnCode,
  CciTerminal*  terminalObject);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_TERMINAL_OBJECT

terminalObject
The address of the input or output terminal to be checked for an attached connector (input). The address is returned from cniCreateOutputTerminal.

Return values:

- If the terminal is attached to another node by a connector, a value of 1 is returned.
- If the terminal is not attached, or a failure occurred, a value of zero is returned.
- If a failure occurs, the value of the *returnCode* parameter indicates the reason for the error.

Example:

```
if (terminalObject) {
  if (cniIsTerminalAttached(&rc, terminalObject)) {
    if (rc == CCI_SUCCESS) {
      retvalue = cniPropagate(
                              &rc,
                              terminalObject,
                              localEnvironment,
                              exceptionList,
                              message);
```

Related concepts:

"User-defined input nodes" on page 2990
A user-defined input node is an extension to the broker that provides a new input node in addition to the nodes supplied with the product.

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating an input node in C" on page 3027
Create a user-defined input node in C to receive messages into a message flow.

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniCreateOutputTerminal" on page 6450
Use this function to create an output terminal on an instance of a node object and return the address of the terminal object that was created.

"cniPropagate" on page 6501
Use this function to propagate a message to a specified terminal object. If the terminal is not attached to another node by a connector, the message is not propagated, and the function is ignored.

cniLastChild:

Use this function to retrieve the address of the syntax element object that is the last child of the specified syntax element.

Syntax:

```
CciElement* cniLastChild(
  int*         returnCode,
  CciElement*  targetElement);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_MESSAGE_OBJECT

targetElement
The address of the target syntax element object (input).

Return values:

- If successful, the address of the requested syntax element object is returned.
- If there is no last child, zero is returned, and *returnCode* is set to CCI_SUCCESS.
- If an error occurs, zero (CCI_NULL_ADDR) is returned, and the *returnCode* parameter indicates the reason for the error.

Example:

```
bodyChild = cniLastChild(&rc, outRootElement);
```

Related concepts:

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using

system-provided functions.

"cniFirstChild" on page 6481
Use this function to retrieve the address of the syntax element object that is the first child of the specified syntax element.

"cniNextSibling"
Use this function to retrieve the address of the syntax element object that is the next sibling (right sibling) of the specified syntax element.

"cniParent" on page 6498
Use this function to retrieve the address of the syntax element object that is the parent of the specified syntax element.

"cniPreviousSibling" on page 6499
Use this function to retrieve the address of the syntax element object that is the previous sibling (left sibling) of the specified syntax element.

"cniRootElement" on page 6503
Use this function to get the root syntax element associated with a specified message. It returns the root element that is associated with (and owned by) the message object identified by the *message* parameter. When a message object is constructed by the broker, a root element is automatically created.

"cniSearchElement group" on page 6506
Use one or more of the functions in this group to search previous siblings of the specified element for an element that matches the specified criteria.

cniNextSibling:

Use this function to retrieve the address of the syntax element object that is the next sibling (right sibling) of the specified syntax element.

Syntax:

```
CciElement* cniNextSibling(
  int*         returnCode,
  CciElement*  targetElement);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT

targetElement
The address of the target syntax element object (input).

Return values:

- If successful, the address of the requested syntax element object is returned.
- If there is no next sibling, zero is returned, and *returnCode* is set to CCI_SUCCESS.
- If an error occurs, zero (CCI_NULL_ADDR) is returned, and the *returnCode* parameter indicates the reason for the error.

Example:

```
nextSibling = cniNextSibling(&rc, element);
```

Related concepts:

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniFirstChild" on page 6481
Use this function to retrieve the address of the syntax element object that is the first child of the specified syntax element.

"cniLastChild" on page 6496
Use this function to retrieve the address of the syntax element object that is the last child of the specified syntax element.

"cniParent"
Use this function to retrieve the address of the syntax element object that is the parent of the specified syntax element.

"cniPreviousSibling" on page 6499
Use this function to retrieve the address of the syntax element object that is the previous sibling (left sibling) of the specified syntax element.

"cniRootElement" on page 6503
Use this function to get the root syntax element associated with a specified message. It returns the root element that is associated with (and owned by) the message object identified by the *message* parameter. When a message object is constructed by the broker, a root element is automatically created.

"cniSearchElement group" on page 6506
Use one or more of the functions in this group to search previous siblings of the specified element for an element that matches the specified criteria.

cniParent:

Use this function to retrieve the address of the syntax element object that is the parent of the specified syntax element.

Syntax:

```
CciElement* cniParent(
  int*         returnCode,
  CciElement*  targetElement);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION

- CCI_INV_ELEMENT_OBJECT

targetElement
The address of the target syntax element object (input).

Return values:

- If successful, the address of the requested syntax element is returned.
- If there is no parent element, zero is returned.
- If an error occurs, zero (CCI_NULL_ADDR) is returned, and the *returnCode* parameter indicates the reason for the error.

Related concepts:

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniFirstChild" on page 6481
Use this function to retrieve the address of the syntax element object that is the first child of the specified syntax element.

"cniLastChild" on page 6496
Use this function to retrieve the address of the syntax element object that is the last child of the specified syntax element.

"cniNextSibling" on page 6497
Use this function to retrieve the address of the syntax element object that is the next sibling (right sibling) of the specified syntax element.

"cniPreviousSibling"
Use this function to retrieve the address of the syntax element object that is the previous sibling (left sibling) of the specified syntax element.

"cniRootElement" on page 6503
Use this function to get the root syntax element associated with a specified message. It returns the root element that is associated with (and owned by) the message object identified by the *message* parameter. When a message object is constructed by the broker, a root element is automatically created.

"cniSearchElement group" on page 6506
Use one or more of the functions in this group to search previous siblings of the specified element for an element that matches the specified criteria.

cniPreviousSibling:

Use this function to retrieve the address of the syntax element object that is the previous sibling (left sibling) of the specified syntax element.

Syntax:

```
CciElement* cniPreviousSibling(
  int*         returnCode,
  CciElement*  targetElement);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT

targetElement
The address of the target syntax element object (input).

Return values:

- If successful, the address of the requested syntax element object is returned.
- If there is no previous sibling, zero is returned, and *returnCode* is set to CCI_SUCCESS.
- If an error occurs, zero (CCI_NULL_ADDR) is returned, and the *returnCode* parameter indicates the reason for the error.

Related concepts:

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniFirstChild" on page 6481
Use this function to retrieve the address of the syntax element object that is the first child of the specified syntax element.

"cniLastChild" on page 6496
Use this function to retrieve the address of the syntax element object that is the last child of the specified syntax element.

"cniNextSibling" on page 6497
Use this function to retrieve the address of the syntax element object that is the next sibling (right sibling) of the specified syntax element.

"cniParent" on page 6498
Use this function to retrieve the address of the syntax element object that is the parent of the specified syntax element.

"cniRootElement" on page 6503
Use this function to get the root syntax element associated with a specified message. It returns the root element that is associated with (and owned by) the message object identified by the *message* parameter. When a message object is constructed by the broker, a root element is automatically created.

"cniSearchElement group" on page 6506
Use one or more of the functions in this group to search previous siblings of the specified element for an element that matches the specified criteria.

cniPropagate:

Use this function to propagate a message to a specified terminal object. If the terminal is not attached to another node by a connector, the message is not propagated, and the function is ignored.

Therefore, you do not have to check whether the terminal is attached before you propagate the message, unless you want the node to take different in this scenario. If required, you can use cniIsTerminalAttached to check whether the terminal is connected before you call this function.

Syntax:

```
int cniPropagate(
  int*          returnCode,
  CciTerminal*  terminalObject,
  CciMessage*   localEnvironment,
  CciMessage*   exceptionList,
  CciMessage*   message);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_TERMINAL_OBJECT
- CCI_INV_MESSAGE_OBJECT

terminalObject
The address of the output terminal to receive the message (input). The address is returned by cniCreateOutputTerminal.

localEnvironment
The address of the local environment object to be sent with the message (input).

This message object is used by the publish/subscribe node supplied by the broker.

For compatibility with earlier versions, you can refer to this parameter as `destinationList`.

exceptionList
The address of the exception list for the message (input).

message
The address of the message object to be sent (input). If the message being sent is the same as the input message, this address is the one passed on the cniEvaluate implementation function.

Return values:
If successful, CCI_SUCCESS is returned. Otherwise, CCI_FAILURE is returned, and the *returnCode* parameter indicates the reason for the error.

Example:

```
if (terminalObject) {
  if (cniIsTerminalAttached(&rc, terminalObject)) {
    if (rc == CCI_SUCCESS) {
      cniPropagate(&rc, terminalObject, destinationList, exceptionList, message);
```

Related concepts:

"User-defined input nodes" on page 2990
A user-defined input node is an extension to the broker that provides a new input node in addition to the nodes supplied with the product.

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating an input node in C" on page 3027
Create a user-defined input node in C to receive messages into a message flow.

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniCreateMessage" on page 6446
Use this function to create an output message object. For every call to this function, you must include a matching call to cniDeleteMessage to return allocated resources when the processing on the output message has been completed.

"cniCreateOutputTerminal" on page 6450
Use this function to create an output terminal on an instance of a node object and return the address of the terminal object that was created.

"cniDeleteMessage" on page 6453
Use this function to delete the specified message object. For every call to the cniCreateMessage function, you must include a matching call to cniDeleteMessage to return allocated resources when the processing on the output message has been completed.

"cniEvaluate" on page 6475
This function performs node processing. The broker calls this function when a message is received on one of the input terminals of an instance of a node object.

"cniFinalize" on page 6479
Use this function to cause the broker to request parsers to perform finalize processing on the specified message. Finalization is a process that fixes header chains and makes the Properties folder match the headers.

"cniGetEnvironmentMessage" on page 6490
Use this function to get the CciMessage object that corresponds to the environment tree for the message flow.

"cniGetMessageContext" on page 6491
Use this function to get the address of the message context that is associated with the specified message. The context of an existing message is used to create an output message; for example, by using the cniCreateMessage function.

"cniIsTerminalAttached" on page 6494
Use this function to check whether a terminal is attached to another node by a connector. It returns an integer value that specifies whether the specified terminal object is attached to one or more terminals on other message flow nodes.

cniRootElement:

Use this function to get the root syntax element associated with a specified message. It returns the root element that is associated with (and owned by) the message object identified by the *message* parameter. When a message object is constructed by the broker, a root element is automatically created.

Syntax:

```
CciElement* cniRootElement(
  int*         returnCode,
  CciMessage*  message);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_MESSAGE_OBJECT

message
The address of the message object (input).

Return values:
If successful, the address of the root element object is returned. Otherwise, zero (CCI_NULL_ADDR) is returned, and the *returnCode* parameter indicates the reason for the error.

Example:

```
inRootElement = cniRootElement(&rc, message);
```

Related concepts:

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniFirstChild" on page 6481
Use this function to retrieve the address of the syntax element object that is the first child of the specified syntax element.

"cniLastChild" on page 6496
Use this function to retrieve the address of the syntax element object that is the last child of the specified syntax element.

"cniNextSibling" on page 6497
Use this function to retrieve the address of the syntax element object that is the next sibling (right sibling) of the specified syntax element.

"cniParent" on page 6498
Use this function to retrieve the address of the syntax element object that is the parent of the specified syntax element.

"cniPreviousSibling" on page 6499
Use this function to retrieve the address of the syntax element object that is the previous sibling (left sibling) of the specified syntax element.

"cniSearchElement group" on page 6506
Use one or more of the functions in this group to search previous siblings of the specified element for an element that matches the specified criteria.

cniRun:

This function declares the node as an input node.

Message processing nodes and output nodes do not use it, and you do not need to call cniEvaluate. The broker allocates a thread, and calls this function on that thread.

Defined In	Type	Member
CNI_VFT	Conditional	iFpRun

Syntax:

```
int cniRun(
  CCiContext*  context,
  CCiMessage*  localEnvironment,
  CciMessage*  exceptionList,
  CciMessage*  message
);
```

Parameters:

context
The address of the context for the instance of the node, as created by the node and returned by the cniCreateNodeContext function (input).

localEnvironment
The address of the input local environment object (input).

For compatibility with earlier versions, you can refer to this parameter as `destinationList`.

exceptionList
The address of the exception list for the message (input).

message
The address of the message object to which the data is attached (input).

The user-defined node can call cniSetInputBuffer to associate a bit stream with this message. Populating the tree of this message is not supported, therefore calls to functions such as cniAddAsLastChild or cniCreateElementAsLastChildFromBitstream do not work. To build parts of the tree, create a new message using cniCreateMessage rather than providing a buffer to be parsed as the whole message.

For example, if you have a bit stream that is to be used as the payload part of the message, and you also want to add a header, take the following steps:

1. Create a new message using cniCreateMessage.
2. Create the header part in this new message by using the Syntax Element Access Utility functions, for example cniCreateElementAsLastChildUsingParser, and passing in the root element of this new message.
3. Add fields to the header by using functions such as cniCreateElementAsLastChild.
4. Create the body of the message by parsing your bit stream through calling cniCreateElementAsLastChildFromBitstream, and passing in the root element of this new message.

Return values:
This function is called by the broker as part of a loop. The meaning of the return value is as follows:

CCI_TIMEOUT
The input node did not receive its input data. This value means that control should be returned to the broker in case message flow reconfiguration is being requested. A user-defined input node should return regularly to give control back to the broker.

CCI_SUCCESS_CONTINUE
A message was successfully processed. The broker performs default transaction commit processing. The input node's cniRun implementation function is called immediately so that the node can continue processing.

CCI_SUCCESS_RETURN
A message has been successfully processed. The broker performs default transaction commit processing. The input node has determined that the thread is not required, and it is returned to the message flow thread pool. If this processing is performed on the only thread, or the last active thread, the broker prevents this last thread being returned to the pool, otherwise no active threads are available to dispatch another thread. In this situation, the broker invokes the cniRun implementation function immediately, as if CCI_SUCCESS_CONTINUE was returned.

CCI_FAILURE_CONTINUE
An error was detected in the processing of a message, and the node is requesting that transaction rollback processing is performed. The input node's cniRun implementation function is called immediately.

CCI_FAILURE_RETURN
An error was detected in the processing of a message, and the node is requesting that transaction rollback processing is performed. However, the input node has determined that the thread is not required and it can be returned to the message flow thread pool. If this processing is performed on the last active thread, the broker prevents this last thread being returned to the

pool, otherwise no active threads are available to dispatch another thread. In this situation the broker invokes the cniRun implementation function immediately, as if CCI_FAILURE_CONTINUE was returned.

Related concepts:

"User-defined input nodes" on page 2990
A user-defined input node is an extension to the broker that provides a new input node in addition to the nodes supplied with the product.

Related tasks:

"Creating an input node in C" on page 3027
Create a user-defined input node in C to receive messages into a message flow.

Related reference:

"C node implementation functions" on page 6417
The user-defined node implements a function interface for the broker to call during runtime operation. This interface includes functions to create a local context whenever a node instance is created, functions to set and retrieve attribute values, the function to perform the processing of the node itself, and functions to examine messages.

cniSearchElement group:

Use one or more of the functions in this group to search previous siblings of the specified element for an element that matches the specified criteria.

The search is performed starting at the syntax element specified in the *targetElement* parameter, and each of the four functions provides a search in a different tree direction:

- cniSearchFirstChild searches the immediate child elements of the starting element from the first child, until either a match is found, or the end of the child element chain is reached.
- cniSearchLastChild searches the immediate child elements of the starting element from the last child, until either a match is found, or the end of the child element chain is reached.
- cniSearchNextSibling searches from the starting element to the next siblings, until either a match is found, or the end of the sibling chain is reached.
- cniSearchPreviousSibling searches from the starting element to the previous siblings, until either a match is found, or the start of the sibling chain is reached.

If you use this command to search for an element within a message that belongs to a namespace-aware domain, the search is performed only on those elements whose namespace is an empty string. If you want to perform a search for elements in all namespaces, use one of the cniSearchElementNamespace functions.

Syntax:

```
CciElement* cniSearchFirstChild(
  int*            returnCode,
  CciElement*     targetElement,
  CciCompareMode* mode,
  CciElementType  type,
  CciChar         name);
CciElement* cniSearchLastChild(
  int*            returnCode,
  CciElement*     targetElement,
  CciCompareMode* mode,
  CciElementType  type,
  CciChar         name);
```

```
CciElement* cniSearchNextSibling(
  int*            returnCode,
  CciElement*     targetElement,
  CciCompareMode* mode,
  CciElementType  type,
  CciChar         name);
CciElement* cniSearchPreviousSibling(
  int*            returnCode,
  CciElement*     targetElement,
  CciCompareMode* mode,
  CciElementType  type,
  CciChar         name);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT

targetElement
The address of the syntax element object from which the search starts (input).

mode
The search mode to use (input). This parameter indicates what combination of element type and element name is to be searched for. The possible values are:

- CCI_COMPARE_MODE_FULL
- CCI_COMPARE_MODE_FULL_TYPE
- CCI_COMPARE_MODE_GENERIC_TYPE
- CCI_COMPARE_MODE_SPECIFIC_TYPE
- CCI_COMPARE_MODE_NAME
- CCI_COMPARE_MODE_NAME_SPECIFIC_TYPE
- CCI_COMPARE_MODE_NAME_GENERIC_TYPE
- CCI_COMPARE_MODE_NAME_FULL_TYPE
- CCI_COMPARE_MODE_NULL

type
The element type to search for (input). Use this parameter only if the search mode involves a match on the type.

name
The element name to search for (input). Use this parameter only if the search mode involves a match on the name.

Example:

```
int rc;
CciElement* firstChild = cniSearchFirstChild(
                                            &rc,
                                             inRootElement,
                                             CCI_COMPARE_MODE_NAME,
                                             elementName,
                                             0);
```

Return values:

- If successful, the address of the requested syntax element object is returned.

- If there is no matching element, zero is returned.
- If an error occurs, zero (CCI_NULL_ADDR) is returned, and the *returnCode* parameter indicates the reason for the error.

Related concepts:

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniFirstChild" on page 6481
Use this function to retrieve the address of the syntax element object that is the first child of the specified syntax element.

"cniLastChild" on page 6496
Use this function to retrieve the address of the syntax element object that is the last child of the specified syntax element.

"cniNextSibling" on page 6497
Use this function to retrieve the address of the syntax element object that is the next sibling (right sibling) of the specified syntax element.

"cniParent" on page 6498
Use this function to retrieve the address of the syntax element object that is the parent of the specified syntax element.

"cniPreviousSibling" on page 6499
Use this function to retrieve the address of the syntax element object that is the previous sibling (left sibling) of the specified syntax element.

"cniRootElement" on page 6503
Use this function to get the root syntax element associated with a specified message. It returns the root element that is associated with (and owned by) the message object identified by the *message* parameter. When a message object is constructed by the broker, a root element is automatically created.

"cniSearchElementInNamespace group"
Use this element to search for an element that matches the specified criteria.

cniSearchElementInNamespace group:

Use this element to search for an element that matches the specified criteria.

The search starts at the syntax element specified in the element argument, and each of the four functions provides a search in a different tree direction:

1. cniSearchFirstChildInNamespace searches the immediate child elements of the starting element from the first child, until either a match is found, or the end of the child element chain is reached.

2. cniSearchLastChildInNamespace searches the immediate child elements of the starting element from the last child, until either a match is found, or the end of the child element chain is reached.
3. cniSearchNextSiblingInNamespace searches from the starting element to the next siblings, until either a match is found, or the end of the sibling chain is reached.
4. cniSearchPreviousSiblingInNamespace searches from the starting element to the previous siblings, until either a match is found, or the start of the sibling chain is reached.

Use this function when you search a message that belongs to a namespace-aware domain.

Syntax:

```
void cniSearchFirstChildInNamespace(
 int*           returnCode,
 CciElement*    targetElement,
 CciCompareMode mode,
 const CciChar* nameSpace,
 const CciChar* name,
 CciElementType type)

void cniSearchLastChildInNamespace(
 int*           returnCode,
 CciElement*    targetElement,
 CciCompareMode mode,
 const CciChar* nameSpace,
 const CciChar* name,
 CciElementType type)

void cniSearchNextSiblingInNamespace(
 int*           returnCode,
 CciElement*    targetElement,
 CciCompareMode mode,
 const CciChar* nameSpace,
 const CciChar* name,
 CciElementType type)

void cniSearchPreviousSiblingInNamespace(
 int*           returnCode,
 CciElement*    targetElement,
 CciCompareMode mode,
 CciElementType type,
 const CciChar* nameSpace,
 const CciChar* name)
```

Parameters:

returnCode

The return code from the function (output). Specifying a NULL pointer signifies that the node does not want to deal with errors. If input is not NULL, the output signifies the success status of the call. All exceptions thrown during the execution of this call are re-thrown to the next upstream node in the flow. Call cciGetLastExceptionData for details of the exception.

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT

targetElement
The address of the syntax element object from which the search starts (input).

mode
The search mode to use (input). This parameter indicates what combination of element namespace, element name, and element type is to be searched for. The possible values are:

- CCI_COMPARE_MODE_SPACE
- CCI_COMPARE_MODE_SPACE_FULL_TYPE
- CCI_COMPARE_MODE_SPACE_GENERIC_TYPE
- CCI_COMPARE_MODE_SPACE_SPECIFIC_TYPE
- CCI_COMPARE_MODE_SPACE_NAME
- CCI_COMPARE_MODE_SPACE_NAME_FULL_TYPE
- CCI_COMPARE_MODE_SPACE_NAME_GENERIC_TYPE
- CCI_COMPARE_MODE_SPACE_NAME_SPECIFIC_TYPE
- CCI_COMPARE_MODE_NULL

When the compare mode does not involve a match on the namespace, all namespaces are searched. This behavior differs from that of the cniSearchElement group, where only the empty string namespace is searched. When you specify one of the valid modes, set the *nameSpace* parameter to the empty string.

type
The element type to search for (input). Use this parameter only if the search mode involves a match on the type.

nameSpace
The namespace to search (input). Use this parameter only if the search mode involves a match on the namespace.

name
The name to search for (input). Use this parameter only if the search mode involves a match on the name.

Return values:
None. If an error occurs, the *returnCode* parameter indicates the reason for the error.

Example:

```
mode=CCI_COMPARE_MODE_SPACE ;
...

  if (forward) {
    firstChild = cniSearchFirstChildInNamespace(&rc, element, mode, space, 0,0);
   }else{
    firstChild = cniSearchLastChildInNamespace(&rc, element, mode, space, 0,0);

   }

   if (firstChild) {
     depth++;
     traceElement(firstChild,forward,space);
     depth--;
   }
   currentElement = firstChild;
   do{

     if (forward) {
```

```
      nextSibling = cniSearchNextSiblingInNamespace(&rc, currentElement,mode,space,0,0);
    }else{
      nextSibling = cniSearchPreviousSiblingInNamespace(&rc, currentElement,mode,space,0,0);
    }
    if (nextSibling) {
      traceElement(nextSibling,forward,space);
      currentElement=nextSibling;
    }

  }while (nextSibling) ;

}
```

Related concepts:

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"cniElementNamespace" on page 6465
Use this function to get the value of the *namespace* attribute for the specified syntax element.

"cniSearchElement group" on page 6506
Use one or more of the functions in this group to search previous siblings of the specified element for an element that matches the specified criteria.

"cniSetElementNamespace" on page 6514
Use this function to set the *namespace* attribute for the specified syntax element when you manipulate a message that belongs to a namespace-aware domain.

"cciGetLastExceptionData" on page 6647
Gets diagnostic information about the last exception generated. Information about the last exception generated on the current thread is returned in a CCI_EXCEPTION_ST output structure. The user-defined extension can use this function to determine whether any recovery is required when a utility function returns an error code.

cniSetAttribute:

This function sets the value of an attribute on a specific node instance. It is called by the broker when a configuration request is received that attempts to set the value of a node attribute, or during initialization of the node.

A node receives requests to set attributes for the base. If an unknown attribute value is received, this function *must* return a non-zero value so that the broker processes the request correctly.

The responsibilities of the node are to:

1. Verify that the value of the attribute is correctly specified. If not, a configuration exception should be thrown using the cciThrowException function.

2. Store the value of the attribute within the context, which should have been allocated in the cniCreateNodeContext function.
3. Throw a configuration exception if an error occurs, by using the cciThrowException function.

Defined In	Type	Member
CNI_VFT	Optional	iFpSetAttribute

Syntax:

```
int cniSetAttribute(
  CciContext*  context,
  CciChar*     attrName,
  CciChar*     attrValue);
```

Parameters:

context
The address of the context for the instance of the node, as created by the node and returned by the **cniCreateNodeContext** function (input).

attrName
The name of the attribute whose value is to be set (input).

attrValue
The value of the attribute (input).

Return values:
If successful, zero is returned. If the name of the attribute does not identify one supported by the node, a non-zero value is returned.

Related concepts:

"User-defined input nodes" on page 2990
A user-defined input node is an extension to the broker that provides a new input node in addition to the nodes supplied with the product.

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating an input node in C" on page 3027
Create a user-defined input node in C to receive messages into a message flow.

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node implementation functions" on page 6417
The user-defined node implements a function interface for the broker to call during runtime operation. This interface includes functions to create a local context whenever a node instance is created, functions to set and retrieve attribute values, the function to perform the processing of the node itself, and functions to examine messages.

"cniCreateNodeContext" on page 6447
This function creates a context for an instance of a node object. It is called by the broker whenever an instance of a node object is constructed. Nodes are constructed when a message flow is deployed by the broker, or when the execution group is started.

"cciThrowException" on page 6666
Use this function to throw an exception. The exception is thrown by the broker interface, and includes the specified arguments as exception data.

cniSetElementName:

Use this function to set the name of the specified syntax element.

Syntax:

```
void cniSetElementName(
  int*            returnCode,
  CciElement*     targetElement,
  const CciChar*  name);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT
- CCI_INV_DATA_POINTER

targetElement
The address of the target syntax element object (input).

name
The name of the element (input).

Return values:
None. If an error occurs, the *returnCode* parameter indicates the reason for the error.

Example:

```
CciElement* lastChild = cniCreateElementAsLastChild(&rc, outRootElement);
cniSetElementName(&rc, lastChild, elementName);
cniSetElementType(&rc, lastChild, CCI_ELEMENT_TYPE_NAME);
```

Related concepts:

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniElementName" on page 6464
Use this function to get the value of the *name* attribute for the specified syntax element. You must sethe syntax element name previously by using cniSetElementName or cpiSetElementName.

"cniSetElementNamespace"
Use this function to set the *namespace* attribute for the specified syntax element when you manipulate a message that belongs to a namespace-aware domain.

"cniSetElementType" on page 6515
Use this function to set the type of the specified syntax element.

"cniSetElementValue group" on page 6516
Use one or more of the functions in this group to set the value of the specified syntax element.

"cniSetElementValueValue" on page 6519
Use this function to set the value object of the specified syntax element.

cniSetElementNamespace:

Use this function to set the *namespace* attribute for the specified syntax element when you manipulate a message that belongs to a namespace-aware domain.

Syntax:

```
void cniSetElementNamespace(
 int*           returnCode,
 CciElement*    targetElement,
 const CciChar* nameSpace)
```

Parameters:

returnCode
The return code from the function (output). Specifying a NULL pointer signifies that the node does not want to deal with errors. If input is not NULL, the output signifies the success status of the call. Any exceptions thrown during the execution of this call are re-thrown to the next upstream node in the flow. Call cciGetLastExceptionData for details of the exception.

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT
- CCI_INV_DATA_POINTER

targetElement
Specifies the address of the target syntax element object (input).

value
Specifies the address of a null terminated string of CciChars representing the namespace value (output). An empty string is a valid value for namespace. By default, elements are created in the empty string namespace, so you could specify an empty string as the namespace, but it has an effect only if the element was previously in another namespace and you want to change the namespace value to empty string.

Return values:
None. If an error occurs, the *returnCode* parameter indicates the reason for the error.

Related concepts:

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"cniElementNamespace" on page 6465
Use this function to get the value of the *namespace* attribute for the specified syntax element.

"cniSearchElementInNamespace group" on page 6508
Use this element to search for an element that matches the specified criteria.

"cniSetElementName" on page 6513
Use this function to set the name of the specified syntax element.

"cciGetLastExceptionData" on page 6647
Gets diagnostic information about the last exception generated. Information about the last exception generated on the current thread is returned in a CCI_EXCEPTION_ST output structure. The user-defined extension can use this function to determine whether any recovery is required when a utility function returns an error code.

cniSetElementType:

Use this function to set the type of the specified syntax element.

Syntax:

```
void cniSetElementType(
  int*            returnCode,
  CciElement*     targetElement,
  CciElementType  type);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT

targetElement
The address of the target syntax element object (input).

type
The type of the element (input).

Return values:
None. If an error occurs, the *returnCode* parameter indicates the reason for the error.

Example:

```
CciElement* lastChild = cniCreateElementAsLastChild(&rc, outRootElement);
cniSetElementName(&rc, lastChild, elementName);
cniSetElementType(&rc, lastChild, CCI_ELEMENT_TYPE_NAME);
```

Related concepts:

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniElementType" on page 6467
Use this function to get the value of the *type* attribute for the specified syntax element. You must set the syntax element type previously by using cniSetElementType or cpiSetElementType.

"cniSetElementName" on page 6513
Use this function to set the name of the specified syntax element.

"cniSetElementValueValue" on page 6519
Use this function to set the value object of the specified syntax element.

cniSetElementValue group:

Use one or more of the functions in this group to set the value of the specified syntax element.

Specify the appropriate function from this group that matches the type of data to be retrieved:

- cniSetElementBitArrayValue
- cniSetElementBooleanValue
- cniSetElementByteArrayValue
- cniSetElementCharacterValue
- cniSetElementDateValue
- cniSetElementDecimalValue
- cniSetElementGmtTimestampValue
- cniSetElementGmtTimeValue
- cniSetElementIntegerValue
- cniSetElementRealValue
- cniSetElementTimestampValue

- cniSetElementTimeValue

Syntax:

```
void cniSetElementBitArrayValue(
  int*                       returnCode,
  CciElement*                targetElement,
  const struct CciBitArray* value);

void cniSetElementBooleanValue(
  int*                        returnCode,
  CciElement*                 targetElement,
  CciBool                     value);

void cniSetElementByteArrayValue(
  int*                        returnCode,
  CciElement*                 targetElement,
  const struct CciByteArray* value);

void cniSetElementCharacterValue(
  int*                        returnCode,
  CciElement*                 targetElement,
  const CciChar*              value,
  CciSize                     length);

void cniSetElementDateValue(
  int*                        returnCode,
  CciElement*                 targetElement,
  const struct CciDate*       value);

void cniSetElementDecimalValue(
  int*                        returnCode,
  CciElement*                 targetElement,
  const CciChar*              value);

void cniSetElementGmtTimestampValue(
  int*                        returnCode,
  CciElement*                 targetElement,
  const struct CciTimestamp* value);

void cniSetElementGmtTimeValue(
  int*                        returnCode,
  CciElement*                 targetElement,
  const struct CciTime*       value);

void cniSetElementIntegerValue(
  int*                        returnCode,
  CciElement*                 targetElement,
  CciInt                      value);

void cniSetElementRealValue(
  int*                        returnCode,
  CciElement*                 targetElement,
  CciReal                     value);

void cniSetElementTimestampValue(
  int*                        returnCode,
  CciElement*                 targetElement,
  const struct CciTimestamp* value);

void cniSetElementTimeValue(
  int*                        returnCode,
  CciElement*                 targetElement,
  const struct CciTime*       value);
```

Parameters:

returnCode

The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION

- CCI_INV_ELEMENT_OBJECT
- CCI_INV_DATA_POINTER
- CCI_INV_DATA_BUFLEN

targetElement
The address of the target syntax element object (input).

value
The value to store in the syntax element (input).

length
The length of the data value (input). Used on relevant function calls only.

Return values:
None. If an error occurs, the *returnCode* parameter indicates the reason for the error.

Example:

```
static char* functionName = (char *)"_Input_run()";
void* buffer;
CciTerminal* terminalObject;
int buflen = 4096;
int rc = CCI_SUCCESS;
int rcDispatch = CCI_SUCCESS;
char xmlData[] = "<A>data</a>";
buffer = malloc(buflen);
memcpy(buffer, &xmlData, sizeof(xmlData));

cniSetInputBuffer(&rc, message, buffer, buflen);
```

Related concepts:

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniElementValue group" on page 6468
Use one or more of the functions in this group to retrieve the value of the specified syntax element.

"cniSetElementName" on page 6513
Use this function to set the name of the specified syntax element.

"cniSetElementType" on page 6515
Use this function to set the type of the specified syntax element.

"cniSetElementValueValue" on page 6519
Use this function to set the value object of the specified syntax element.

cniSetElementValueValue:

Use this function to set the value object of the specified syntax element.

Syntax:

```
void cniSetElementValueValue(
  int*              returnCode,
  CciElement*       targetElement,
  CciElementValue*  value);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT
- CCI_INV_DATA_POINTER

targetElement
The address of the target syntax element object (input).

value
The address of a value object that is used to set the value of the syntax element specified by the *targetElement* parameter (input). You can get the address of the value object by calling cniElementValueValue.

Return values:
None. If an error occurs, the *returnCode* parameter indicates the reason for the error.

Related concepts:

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniElementValueValue" on page 6473
Use this function to get the address of the value object owned by the specified syntax element.

"cniSetElementName" on page 6513
Use this function to set the name of the specified syntax element.

"cniSetElementType" on page 6515
Use this function to set the type of the specified syntax element.

"cniSetElementValue group" on page 6516
Use one or more of the functions in this group to set the value of the specified syntax element.

cniSetInputBuffer:

Use this function to supply a buffer. It is used only by input nodes. The address is specified by the `source` parameter as an input bit stream of the input message to the broker.

By supplying a buffer, an input node can read data into the bit stream that represents an input message from an external data source. The broker takes a copy of the data and the caller can free the storage on return.

Syntax:

```
int cniSetInputBuffer(
  void*        returnCode,
  CciMessage*  message,
  Void*        source,
  CCiInt       length);
```

Parameters:

`returnCode`
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_MESSAGE_OBJECT
- CCI_INV_DATA_POINTER
- CCI_INV_DATA_BUFLEN

`message`
The message object that uses the buffer described by the *source* parameter to represent the input bit stream. (input)

`source`
The address of the buffer to be used as input. (input)

`length`
The length of the input buffer described by the *source* parameter. (input)

Return values:
None. If an error occurs, the*returnCode* parameter indicates the reason for the error.

Example:

```
cniSetInputBuffer(&rc, message, buffer, buflen);
```

Related concepts:

"User-defined input nodes" on page 2990
A user-defined input node is an extension to the broker that provides a new input node in addition to the nodes supplied with the product.

Related tasks:

"Creating an input node in C" on page 3027
Create a user-defined input node in C to receive messages into a message flow.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniBufferByte" on page 6426
Use this function to get a single byte from the data buffer associated with (and owned by) the message object specified in the message argument. The value of the index argument indicates which byte in the byte array is to be returned.

"cniBufferPointer" on page 6427
Use this function to get a pointer to the data buffer associated with (and owned by) the message object specified in the message argument. This function is typically used by output nodes.

"cniBufferSize" on page 6428
Use this function to get the size of the data buffer associated with (and owned by) the message object specified in the message argument.

"cniWriteBuffer" on page 6537
Use this function to write the syntax element tree associated with the specified message to the data buffer that is owned by the message object. This function is typically used by output nodes.

cniSqlCreateModifyablePathExpression:

Use this function to create an SqlPathExpression object that represents the path that is specified by the path argument. When they are navigated, path elements are created if they do not already exist. This function returns a pointer to the PathExpression object, which is used as input to the functions that navigate the path, namely the cniSqlNavigatePath family.

Because an overhead is incurred in creating the expression, if the same path expression is to be used for every message, call this function once, and use the CciSqlPathExpression* that is returned in a call to cniSqlNavigate for each message. You can use the CciSqlPathExpression on threads other than the one on which it was created.

Syntax:

```
CciSqlPathExpression* cniSqlCreateModifiablePathExpression(
 int* returnCode,
 CciNode* nodeObject,
 CciChar* dataSourceName,
 CciChar* path);
```

Parameters:

returnCode (output)
A NULL pointer input signifies that the user-defined node does not handle errors. Any exceptions that are thrown during the execution of this call are re-thrown to the next upstream node in the flow. If input is not NULL, output signifies the success status of the call. If an exception occurs during execution, *returnCode is set to CCI_EXCEPTION on output. Call cciGetLastExceptionData to retrieve details of the exception. If an invalid nodeObject parameter was passed in, *returnCode* is set to CCI_INV_NODE_OBJECT. If an invalid path parameter, such as NULL or an empty string, was passed in, *returnCode* is set to CCI_INV_ESQL_PATH_EXPR.

nodeObject (input)
Specifies the message flow processing node that the ESQL Path Expression is

owned by. This pointer is passed to the cniCreateNodeContext implementation function. This parameter must not be NULL.

dataSourceName (input)
The ODBC data source name to be used if the statement references an external database. This parameter can be NULL.

path (input)
Pointer to a NULL terminated string of CciChars. This parameter specifies the ESQL path expression to be created as defined by the ESQL field reference syntax diagram, except that it cannot include local ESQL variables, ESQL reference variables, user-defined functions, or ESQL namespace constants, because they cannot be declared. This parameter must not be NULL.

Return values:
If successful, the address of the SQLPathExpression object is returned. If an error occurs, CCI_NULL_ADDR is returned, and the return code parameter indicates the reason for the error. When the SQLPathExpression is no longer needed, (typically when the node is deleted) call cniSqlDeletePathExpression to delete it.

Example:
If you add the following code to the Transform node sample, you can create an element, and all necessary ancestor elements, with one function call.

Create the CciSQLPathExpression in the _Transform_createNodeContext function:

```
 {
        CciChar ucsPathExpressionString[32];
        char*   mbPathExpressionString =
                          "OutputRoot.XMLNS.Request.A.B.C.D.E";
        /* convert our path string to unicode*/
        cciMbsToUcs(NULL,
            mbPathExpressionString,
            ucsPathExpressionString,
            32,
            BIP_DEF_COMP_CCSID);

        p->pathExpression =
                     cniSqlCreateModifiablePathExpression(
                                 NULL,
                                 nodeObject,
                                 NULL,/* do not reference Database*/
                                 ucsPathExpressionString);
    }
```

Now use the CciSqlPathExpression later in the _Transform_evaluate function

```
{
      CciElement* newElement =
               cniSqlNavigatePath(
                     NULL,
                    ((NODE_CONTEXT_ST *)context)->pathExpression,
                     message,
                     localEnvironment,
                     exceptionList,
                     outMsg,
                     NULL,/* do not reference OutputLocalEnvironment*/
                     NULL/* do not reference OutputLExceptionList*/);
}
```

Therefore passing in the input message `PluginSample.change.xml`:

```
<Request
type="change">
  <CustomerAccount>01234567</CustomerAccount>
  <CustomerPhone>555-0000</CustomerPhone>
</Request>
```

The following output message is generated:

```
<Request
type="modify">
  <CustomerAccount>01234567</CustomerAccount>
  <CustomerPhone>555-0000</CustomerPhone>
  <A>
    <B>
      <C>
        <D/>
      </C>
    </B>
  </A>
</Request>
```

This approach, rather than using functions such as cniCreateElementAsLastChild, has the following advantages:

- The path is more dynamic: the path string could be determined at deploy time, for example based on a node property (you could create the CciSQLPathExpression in the cniSetAttribute implementation function).
- While navigating to and creating the element, only one function call is made. This technique is more apparent when the target element is deep within the tree structure.

Related concepts:

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniCreateNodeContext" on page 6447
This function creates a context for an instance of a node object. It is called by the broker whenever an instance of a node object is constructed. Nodes are constructed when a message flow is deployed by the broker, or when the execution group is started.

"cniSqlDeleteStatement" on page 6530
Use this function to delete an SQL statement that you have created by using the cniSqlCreateStatement utility function, as defined by the *sqlExpression* parameter.

"cniSqlExecute" on page 6531
Use this function to run an SQL statement that you have created by using the cniSqlCreateStatement utility function, as defined by the *sqlExpression* parameter. Use this function when the statement does not return data, for example, when a PASSTHRU function is used.

"cniSqlSelect" on page 6535
Use this function to run an SQL statement that you have created by using the cniSqlCreateStatement utility function, as defined by the *sqlExpression* parameter. If the statement returns data, the data is written into the message specified by the *outputMessage* parameter.

"cniSqlCreateReadOnlyPathExpression"
Use this function to create an SqlPathExpression object that represents the path that is specified by the path argument. The navigated path does not create path elements if they do not already exist. This function returns a pointer to the PathExpression object, which is used as input to the functions that navigate the path, namely the cniSqlNavigatePath family.

"cniSqlNavigatePath" on page 6532
Use this function to run the SQLPathExpression that you have created by calling the cniSqlCreateReadOnlyPathExpression or the cniSqlCreateModifiablePathExpression utility functions, as defined by the *sqlPathExpression* argument.

"cniSqlDeletePathExpression" on page 6529
Use this function to delete the SQLPathExpression that you have created by calling the cniSqlCreateReadOnlyPathExpression or the cniSqlCreateModifiablePathExpression utility functions, as defined by the *sqlPathExpression* argument.

cniSqlCreateReadOnlyPathExpression:

Use this function to create an SqlPathExpression object that represents the path that is specified by the path argument. The navigated path does not create path elements if they do not already exist. This function returns a pointer to the PathExpression object, which is used as input to the functions that navigate the path, namely the cniSqlNavigatePath family.

Because an overhead is incurred in creating the expression, if the same path expression is to be used for every message, call this function once, and use the CciSqlPathExpression* that is returned in a call to cniSqlNavigate for each message. You can use the CciSqlPathExpression* on threads other than the one on which it was created.

Syntax:

```
CciSqlPathExpression* cniSqlCreateReadOnlyPathExpression(
    int*     returnCode,
     CciNode* nodeObject,
     CciChar* dataSourceName,
     CciChar* path );
```

Parameters:

returnCode (output)
A NULL pointer input signifies that the user-defined node does not handle errors. Any exceptions thrown during the execution of this call are re-thrown to the next upstream node in the flow. If input is not NULL, output signifies the success status of the call. If an exception occurs during execution, *returnCode is set to CCI_EXCEPTION on output. A call to

cciGetLastExceptionData provides details of the exception. If an invalid nodeObject parameter was passed in, returnCode is set to CCI_INV_NODE_OBJECT. If an invalid path parameter, such as a NULL or empty string, was passed in, returnCode is set to CCI_INV_ESQL_PATH_EXPR.

nodeObject (input)
Specifies the message flow processing node that owns the ESQL Path Expression. This pointer is passed to the cniCreateNodeContext implementation function. This parameter must not be NULL.

dataSourceName (input)
The ODBC data source name that is used if the statement references an external database. NULL is allowed.

path (input)
Pointer to a NULL terminated string of CciChars. This parameter specifies the ESQL path expression to be created, as defined by the ESQL field reference syntax diagram. It cannot include local ESQL variables, ESQL reference variables, user-defined functions, or ESQL namespace constants, because they cannot be declared. This parameter must not be NULL.

Return values:
If successful, the address of the SQLPathExpression object is returned. If an error occurs, CCI_NULL_ADDR is returned and the return code parameter indicates the reason for the error. When the SQLPathExpression is no longer needed (typically when the node is deleted), call cniSqlDeletePathExpression to delete it.

Example:
The switch node sample shows how to navigate to a syntax element using functions like cniFirstChild. The following code could be used to achieve the same result.

In _Switch_createNodeContext function, create the CciSqlPathExpression for use later.

```
{
        CciChar ucsPathExpressionString[32];
        char*   mbPathExpressionString = "InputBody.Request.type";
        /* convert our path string to unicode*/
        cciMbsToUcs(
                  NULL,
                  mbPathExpressionString,
                  ucsPathExpressionString,
                  32,
                  BIP_DEF_COMP_CCSID);

        p->pathExpression =
                    cniSqlCreateReadOnlyPathExpression(
                            NULL,
                            nodeObject,
                            NULL, /* do not reference Database*/
                            ucsPathExpressionString);
}
```

This code assumes the addition of the field CciSqlPathExpression* pathExpression to the NODE_CONTEXT_ST struct.

Now use the CciSqlPathExpression in the _Switch_evaluate function.

```
CciElement* targetElement = cniSqlNavigatePath(
                             NULL,
                            ((NODE_CONTEXT_ST *)context)->pathExpression,
                             message,
                             localEnvironment,
                             exceptionList,
                             NULL, /* do not reference any output trees*/
                             NULL,
                             NULL);
```

This approach, rather than using functions such as cniFirstChild and cniNextSibling, has the following advantages:

- The path is more dynamic: the path string could be determined at deploy time based on a node property (you could create the CciSqlPathExpression in the cniSetAttribute implementation function).
- While navigating to the element, only one function call is made. This technique is more apparent when the target element is deep within the tree structure.

Related concepts:

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniCreateNodeContext" on page 6447
This function creates a context for an instance of a node object. It is called by the broker whenever an instance of a node object is constructed. Nodes are constructed when a message flow is deployed by the broker, or when the execution group is started.

"cniSqlDeleteStatement" on page 6530
Use this function to delete an SQL statement that you have created by using the cniSqlCreateStatement utility function, as defined by the *sqlExpression* parameter.

"cniSqlExecute" on page 6531
Use this function to run an SQL statement that you have created by using the cniSqlCreateStatement utility function, as defined by the *sqlExpression* parameter. Use this function when the statement does not return data, for example, when a PASSTHRU function is used.

"cniSqlSelect" on page 6535
Use this function to run an SQL statement that you have created by using the cniSqlCreateStatement utility function, as defined by the *sqlExpression* parameter. If the statement returns data, the data is written into the message specified by the *outputMessage* parameter.

"cniSqlCreateModifyablePathExpression" on page 6521
Use this function to create an SqlPathExpression object that represents the path that is specified by the path argument. When they are navigated, path elements are created if they do not already exist. This function returns a pointer to the PathExpression object, which is used as input to the functions that navigate the path, namely the cniSqlNavigatePath family.

"cniSqlNavigatePath" on page 6532
Use this function to run the SQLPathExpression that you have created by calling the cniSqlCreateReadOnlyPathExpression or the cniSqlCreateModifiablePathExpression utility functions, as defined by the *sqlPathExpression* argument.

"cniSqlDeletePathExpression" on page 6529
Use this function to delete the SQLPathExpression that you have created by calling the cniSqlCreateReadOnlyPathExpression or the cniSqlCreateModifiablePathExpression utility functions, as defined by the *sqlPathExpression* argument.

cniSqlCreateStatement:

Use this function to create an ESQL expression object that represents the statement specified by the statement argument, by using the syntax defined for the Compute node.

You cannot use the following statements:

- CREATE PROCEDURE
- CREATE MODULE
- CREATE SCHEMA
- CREATE FUNCTION

This function returns a pointer to the SQL expression object, which is used as input to the functions that execute the statement, which are cniSqlExecute and cniSqlSelect. You can create multiple SQL expression objects in a single message flow node. Although you can create these objects at any time, you would typically create them when the message flow node is instantiated, within the implementation function cniCreateNodeContext.

Syntax:

```
CciSqlExpression* cniSqlCreateStatement(
  int*               returnCode,
  CciNode*           nodeObject,
  CciChar*           dataSourceName,
  CciSqlTransaction  transaction,
  CciChar*           statement);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_NODE_OBJECT
- CCI_INV_TRANSACTION_TYPE
- CCI_INV_STATEMENT

nodeObject
The message flow processing node that the SQL expression object is owned by (input). This pointer is passed to the cniCreateNodeContext implementation function.

dataSourceName
The ODBC data source name used if the statement references data in an external database (input).

transaction
Specifies whether a database commit is performed after the statement is executed (input). Valid values are:

CCI_SQL_TRANSACTION_AUTO
Specifies that a database commit is performed at the completion of the message flow (that is, as a fully globally coordinated or partially globally coordinated transaction). This value is the default.

CCI_SQL_TRANSACTION_COMMIT
Specifies that a commit is performed after execution of the statement, and in the cniSqlExecute or cniSqlSelect function (that is, the message flow is partially broker coordinated).

statement
The SQL expression to be created, using the syntax as defined for the Compute node (input).

Return values:
If successful, the address of the SQL expression object is returned. If an error occurs, zero (CCI_NULL_ADDR) is returned, and the *returnCode* parameter indicates the reason for the error.

Related concepts:

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniCreateNodeContext" on page 6447
This function creates a context for an instance of a node object. It is called by the broker whenever an instance of a node object is constructed. Nodes are constructed when a message flow is deployed by the broker, or when the execution group is started.

"cniSqlDeleteStatement" on page 6530
Use this function to delete an SQL statement that you have created by using the cniSqlCreateStatement utility function, as defined by the *sqlExpression* parameter.

"cniSqlExecute" on page 6531
Use this function to run an SQL statement that you have created by using the cniSqlCreateStatement utility function, as defined by the *sqlExpression* parameter. Use this function when the statement does not return data, for example, when a PASSTHRU function is used.

"cniSqlSelect" on page 6535
Use this function to run an SQL statement that you have created by using the cniSqlCreateStatement utility function, as defined by the *sqlExpression* parameter. If the statement returns data, the data is written into the message specified by the *outputMessage* parameter.

cniSqlDeletePathExpression:

Use this function to delete the SQLPathExpression that you have created by calling the cniSqlCreateReadOnlyPathExpression or the cniSqlCreateModifiablePathExpression utility functions, as defined by the *sqlPathExpression* argument.

Syntax:

```
void cniSqlDeletePathExpression(
        int*                  returnCode,
        CciSqlPathExpression* sqlPathExpression );
```

Parameters:

returnCode (output)
A NULL pointer input signifies that the user-defined node does not want to deal with errors. All exceptions thrown during the execution of this call are re-thrown to the next upstream node in the flow. If input is not NULL, output signifies the success status of the call. If an exception occurs during execution, *returnCode is set to CCI_EXCEPTION on output. Call cciGetLastExceptionData to get details of the exception. If an invalid *sqlPathExpression* parameter is passed in, *returnCode* is set to CCI_INV_SQL_EXPR_OBJECT.

sqlPathExpression (output)
Specifies the *SQLPathExpression* object to be deleted as returned by either cniCreateModifiablePathExpression or cniCreateReadOnlyPathExpression functions. Tyhis parameter cannot be NULL.

Return values:
None. If an error occurs, the *returnCode* parameter indicates the reason for the error..

Example:
Expanding on the example for cniSqlCreateReadOnlyPathExpression, include the following code in _deleteNodeContext.

```
cniSqlDeletePathExpression(
                    NULL,
                    ((NODE_CONTEXT_ST *)context)->pathExpression);
```

Related concepts:

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniCreateNodeContext" on page 6447
This function creates a context for an instance of a node object. It is called by the broker whenever an instance of a node object is constructed. Nodes are constructed when a message flow is deployed by the broker, or when the execution group is started.

"cniSqlDeleteStatement"
Use this function to delete an SQL statement that you have created by using the cniSqlCreateStatement utility function, as defined by the *sqlExpression* parameter.

"cniSqlExecute" on page 6531
Use this function to run an SQL statement that you have created by using the cniSqlCreateStatement utility function, as defined by the *sqlExpression* parameter. Use this function when the statement does not return data, for example, when a PASSTHRU function is used.

"cniSqlSelect" on page 6535
Use this function to run an SQL statement that you have created by using the cniSqlCreateStatement utility function, as defined by the *sqlExpression* parameter. If the statement returns data, the data is written into the message specified by the *outputMessage* parameter.

cniSqlDeleteStatement:

Use this function to delete an SQL statement that you have created by using the cniSqlCreateStatement utility function, as defined by the *sqlExpression* parameter.

Syntax:

```
void cniSqlDeleteStatement(
  int*               returnCode,
  CciSqlExpression*  sqlExpression);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CC_INV_SQL_EXPR_OBJECT

sqlExpression
The SQL expression object to be deleted, as returned by the cniSqlCreateStatement utility function (input).

Return values:
None. If an error occurs, the *returnCode* parameter indicates the reason for the error.

Related concepts:

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniSqlCreateStatement" on page 6527
Use this function to create an ESQL expression object that represents the statement specified by the statement argument, by using the syntax defined for the Compute node.

"cniSqlExecute"
Use this function to run an SQL statement that you have created by using the cniSqlCreateStatement utility function, as defined by the *sqlExpression* parameter. Use this function when the statement does not return data, for example, when a PASSTHRU function is used.

"cniSqlSelect" on page 6535
Use this function to run an SQL statement that you have created by using the cniSqlCreateStatement utility function, as defined by the *sqlExpression* parameter. If the statement returns data, the data is written into the message specified by the *outputMessage* parameter.

cniSqlExecute:

Use this function to run an SQL statement that you have created by using the cniSqlCreateStatement utility function, as defined by the *sqlExpression* parameter. Use this function when the statement does not return data, for example, when a PASSTHRU function is used.

Syntax:

```
void cniSqlExecute(
  int*               returnCode,
  CciSqlExpression*  sqlExpression,
  CciMessage*        localEnvironment,
  CciMessage*        exceptionList,
  CciMessage*        message);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_INV_SQL_EXPR_OBJECT

- CCI_INV_MESSAGE_OBJECT

sqlExpression
The SQL expression object to be executed, as returned by the cniSqlCreateStatement utility function (input).

localEnvironment
The message representing the input local environment (input).

For compatibility with earlier versions, you can refer to this parameter as `destinationList`.

exceptionList
The message representing the input exception list (input).

message
The message representing the input message (input).

Return values:
None. If an error occurs, the *returnCode* parameter indicates the reason for the error.

Related concepts:

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniSqlCreateStatement" on page 6527
Use this function to create an ESQL expression object that represents the statement specified by the statement argument, by using the syntax defined for the Compute node.

"cniSqlDeleteStatement" on page 6530
Use this function to delete an SQL statement that you have created by using the cniSqlCreateStatement utility function, as defined by the *sqlExpression* parameter.

"cniSqlSelect" on page 6535
Use this function to run an SQL statement that you have created by using the cniSqlCreateStatement utility function, as defined by the *sqlExpression* parameter. If the statement returns data, the data is written into the message specified by the *outputMessage* parameter.

cniSqlNavigatePath:

Use this function to run the SQLPathExpression that you have created by calling the cniSqlCreateReadOnlyPathExpression or the cniSqlCreateModifiablePathExpression utility functions, as defined by the *sqlPathExpression* argument.

Syntax:

```
CciElement* cniSqlNavigatePath(
 int*                  returnCode,
 CciSqlPathExpression* sqlPathExpression,
 CciMessage*           inputMessageRoot,
 CciMessage*           inputLocalEnvironment,
 CciMessage*           inputExceptionList,
 CciMessage*           outputMessageRoot
 CciMessage*           outputLocalEnvironment,
 CciMessage*           outputExceptionList);
```

Parameters:

returnCode (output)
A NULL pointer input signifies that the user-defined node does not handle errors. Any exceptions that are thrown during the execution of this call are re-thrown to the next upstream node in the flow. If input is not NULL, output signifies the success status of the call. If an exception occurs during execution, *returnCode is set to CCI_EXCEPTION on output. A call to cciGetLastExceptionData provides details of the exception. If an invalid *sqlPathExpression* parameter was passed in, *returnCode* is set to CCI_INV_SQL_EXPR_OBJECT. If an invalid CciMessage* value is passed in, *returnCode* is set to CCI_INV_MESSAGE_OBJECT. If the element could not be navigated to or created, *returnCode* is set to CCI_PATH_NOT_NAVIGABLE.

sqlPathExpression (input)
Specifies the SQLPathExpression object to be executed as returned by either the cniCreateReadOnlyPathExpression or the cniCreateModifyablePathExpression function. This parameter cannot be NULL.

inputMessageRoot (input)
The message representing the input message. This parameter cannot be NULL.

inputLocalEnvironment (input)
The message representing the input local environment. This parameter cannot be NULL.

inputExceptionList (input)
The message representing the input exception list. This parameter cannot be NULL.

outputMessageRoot (input)
The message representing the output message. This parameter can be NULL.

outputLocalEnvironment (input)
The message representing the output local environment. This parameter can be NULL.

outputExceptionList (input)
The message representing the output exception list. This parameter can be NULL.

The following table shows the mapping between the correlation names accepted in the ESQL path expression and the data that is accessed.

Correlation name	Data accessed
Environment	The single Environment tree for the flow. This element is determined by the broker and it is not necessary to specify it with this API.
InputLocalEnvironment	inputLocalEnvironment parameter to cniSqlNavigatePath

Correlation name	Data accessed
OutputLocalEnvironment	outputLocalEnvironment parameter to cniSqlNavigatePath
InputRoot	inputMessageRoot parameter to cniSqlNavigatePath
InputBody	Last child of InputRoot
InputProperties	InputRoot.Properties (InputRoot.Properties is the first child of InputRoot, named "Properties")
OutputRoot	outputMessageRoot parameter to cniSqlNavigatePath
InputExceptionList	inputExceptionList parameter to cniSqlNavigatePath
OutputExceptionList	outputExceptionList parameter to cniSqlNavigatePath
Database	ODBC datasource identified by dataSourceName parameter to cniCreateReadOnlyPathExpression or cniCreateModifyablePathExpression
InputDestinationList	Synonym for InputLocalEnvironment that is compatible with earlier versions
OutputDestinationList	Synonym for OutputLocalEnvironment that is compatible with earlier versions

All other rules regarding the actual navigability and validity of paths are defined in "Correlation names" on page 1069.

Return values:
If the path is navigated successfully, the address of the syntax element is returned. However, if the path is not navigable, a value of zero (CCI_NULL_ADDR) is returned, and the *returnCode* parameter indicates the reason for the error.

Example:
Assuming that you have previously created a SQLPathExpression (see the example for cniSqlCreateReadOnlyPathExpression or cniSqlCreateModifiablePathExpression), you could use the following code to navigate to the target element.

```
CciElement* targetElement = cniSqlNavigatePath(
                            NULL,
                           ((NODE_CONTEXT_ST *)context)->pathExpression,
                            message,
                            localEnvironment,
                            exceptionList,
                            NULL, /* do not reference any output trees*/
                            NULL,
                            NULL);
```

Related concepts:

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniCreateNodeContext" on page 6447
This function creates a context for an instance of a node object. It is called by the broker whenever an instance of a node object is constructed. Nodes are constructed when a message flow is deployed by the broker, or when the execution group is started.

"cniSqlDeleteStatement" on page 6530
Use this function to delete an SQL statement that you have created by using the cniSqlCreateStatement utility function, as defined by the *sqlExpression* parameter.

"cniSqlExecute" on page 6531
Use this function to run an SQL statement that you have created by using the cniSqlCreateStatement utility function, as defined by the *sqlExpression* parameter. Use this function when the statement does not return data, for example, when a PASSTHRU function is used.

"cniSqlSelect"
Use this function to run an SQL statement that you have created by using the cniSqlCreateStatement utility function, as defined by the *sqlExpression* parameter. If the statement returns data, the data is written into the message specified by the *outputMessage* parameter.

"cniSqlCreateReadOnlyPathExpression" on page 6524
Use this function to create an SqlPathExpression object that represents the path that is specified by the path argument. The navigated path does not create path elements if they do not already exist. This function returns a pointer to the PathExpression object, which is used as input to the functions that navigate the path, namely the cniSqlNavigatePath family.

"cniSqlCreateModifyablePathExpression" on page 6521
Use this function to create an SqlPathExpression object that represents the path that is specified by the path argument. When they are navigated, path elements are created if they do not already exist. This function returns a pointer to the PathExpression object, which is used as input to the functions that navigate the path, namely the cniSqlNavigatePath family.

"cniSqlDeletePathExpression" on page 6529
Use this function to delete the SQLPathExpression that you have created by calling the cniSqlCreateReadOnlyPathExpression or the cniSqlCreateModifiablePathExpression utility functions, as defined by the *sqlPathExpression* argument.

cniSqlSelect:

Use this function to run an SQL statement that you have created by using the cniSqlCreateStatement utility function, as defined by the *sqlExpression* parameter. If the statement returns data, the data is written into the message specified by the *outputMessage* parameter.

Syntax:

```
void cniSqlSelect(
  int*               returnCode,
  CciSqlExpression*  sqlExpression,
  CciMessage*        localEnvironment,
  CciMessage*        exceptionList,
  CciMessage*        message,
  CciMessage*        outputMessage);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_SQL_EXPR_OBJECT
- CCI_INV_MESSAGE_OBJECT

sqlExpression
The SQL expression object to be executed, as returned by the cniSqlCreateStatement utility function (input).

localEnvironment
The message representing the input local environment (input).

For compatibility with earlier versions, you can refer to this parameter as `destinationList`.

exceptionList
The message representing the input exception list (input).

message
The message representing the input message (input).

outputMessage
The message into which any data returned by the statement is written (output).

Return values:
None. If an error occurs, the *returnCode* parameter indicates the reason for the error.

Related concepts:

"User-defined message processing nodes" on page 2996
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniSqlCreateStatement" on page 6527
Use this function to create an ESQL expression object that represents the statement specified by the statement argument, by using the syntax defined for the Compute node.

"cniSqlDeleteStatement" on page 6530
Use this function to delete an SQL statement that you have created by using the cniSqlCreateStatement utility function, as defined by the *sqlExpression* parameter.

"cniSqlExecute" on page 6531
Use this function to run an SQL statement that you have created by using the cniSqlCreateStatement utility function, as defined by the *sqlExpression* parameter. Use this function when the statement does not return data, for example, when a PASSTHRU function is used.

cniWriteBuffer:

Use this function to write the syntax element tree associated with the specified message to the data buffer that is owned by the message object. This function is typically used by output nodes.

This operation serializes the element tree into a bit stream that can then be processed as a sequence of contiguous bytes. This function should be used when writing the bit stream to a target that is outside the broker.

You must call cniFinalize before this call, or cniWriteBuffer fails.

Syntax:

```
void cniWriteBuffer(
  int*         returnCode,
  CciMessage*  message);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_MESSAGE_OBJECT

message
The address of the message object for which the element tree is to be serialized (input).

Return values:
None. If an error occurs, the *returnCode* parameter indicates the reason for the error.

Example:

```
cniCopyElementTree(&rc, inLastChild, outLastChild);
cniFinalize(&rc, outMessage);
cniWriteBuffer(&rc, outMessage);
```

Related concepts:

"User-defined output nodes" on page 3006
A user-defined output node is an extension to the broker that provides a new message flow output node in addition to the nodes supplied with the product.

Related tasks:

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Related reference:

"C node utility functions" on page 6419
A user-defined node created in the C programming language can create or define broker objects, such as node factories, nodes, and terminals by using system-provided functions.

"cniBufferByte" on page 6426
Use this function to get a single byte from the data buffer associated with (and owned by) the message object specified in the message argument. The value of the index argument indicates which byte in the byte array is to be returned.

"cniBufferPointer" on page 6427
Use this function to get a pointer to the data buffer associated with (and owned by) the message object specified in the message argument. This function is typically used by output nodes.

"cniBufferSize" on page 6428
Use this function to get the size of the data buffer associated with (and owned by) the message object specified in the message argument.

"cniFinalize" on page 6479
Use this function to cause the broker to request parsers to perform finalize processing on the specified message. Finalization is a process that fixes header chains and makes the Properties folder match the headers.

"cniSetInputBuffer" on page 6520
Use this function to supply a buffer. It is used only by input nodes. The address is specified by the `source` parameter as an input bit stream of the input message to the broker.

C language user-defined parser API

The C language user-defined parser API consists of two complementary sets of functions that you can use to define the function of your parser.

The two sets of functions are:

1. A set of implementation functions that provide the functionality of the user-defined parser. These functions are called by the broker. Most implementation functions are mandatory and, if not supplied by the developer, cause an exception at run time.
2. A set of utility functions that create resources in the broker or request a service of the broker. These utility functions can be called by a user-defined parser.

These functions are defined in the `BipCpi.h` header file.

This section covers the following topics:

- "C parser implementation functions."
- "C parser utility functions" on page 6539.

C parser implementation functions:

A user-defined parser implements its capability through a function interface which is called by the broker during runtime operation. This interface includes functions to create and delete all local context storage that is associated with a parser object and the parsing operations.

Some implementation functions are mandatory, and must be implemented when you develop your own parser. Other functions are optional, or conditional.

Mandatory functions

- "cpiCreateContext" on page 6553

- "cpiParseNextSibling" on page 6588
- "cpiParsePreviousSibling" on page 6590
- "cpiParseFirstChild" on page 6586
- "cpiParseLastChild" on page 6587

Optional and conditional functions

- "cpiDeleteContext" on page 6559
- "cpiElementValue" on page 6566
- "cpiNextParserClassName" on page 6573
- "cpiNextParserCodedCharSetId" on page 6574
- "cpiNextParserEncoding" on page 6576
- "cpiParseBuffer" on page 6580
- "cpiParseBufferEncoded" on page 6582
- "cpiParseBufferFormatted" on page 6583
- "cpiParserType" on page 6591
- "cpiSetElementValue" on page 6602
- "cpiSetNextParserClassName" on page 6609
- "cpiWriteBuffer" on page 6610
- "cpiWriteBufferEncoded" on page 6612
- "cpiWriteBufferFormatted" on page 6613

C parser utility functions:

A user-defined parser can call functions provided by the broker to create or define objects, such as message parser factories.

These system-provided functions relate to different types of operation of the parser:

Initialization and resource creation

- "cpiCreateParserFactory" on page 6555
- "cpiDefineParserClass" on page 6557

Message buffer access

- "cpiAppendToBuffer" on page 6545
- "cpiBufferByte" on page 6547
- "cpiBufferPointer" on page 6548
- "cpiBufferSize" on page 6549

Syntax element navigation

- "cpiRootElement" on page 6592
- "cpiParent" on page 6578
- "cpiNextSibling" on page 6577
- "cpiFirstChild" on page 6571
- "cpiLastChild" on page 6572
- "cpiAddAfter" on page 6540

Syntax element access

- "cpiAddBefore" on page 6544
- "cpiAddAsFirstChild" on page 6541
- "cpiAddAsLastChild" on page 6543

- "cpiCreateAndInitializeElement" on page 6551
- "cpiCreateElement" on page 6554
- "cpiElementCompleteNext" on page 6560
- "cpiElementCompletePrevious" on page 6561
- "cpiElementName" on page 6562
- "cpiElementNameSpace" on page 6563
- "cpiElementType" on page 6565
- "cpiElementValue group" on page 6567
- "cpiElementValueValue" on page 6569
- "cpiSetCharacterValueFromBuffer" on page 6594
- "cpiSetElementCompleteNext" on page 6595
- "cpiSetElementCompletePrevious" on page 6596
- "cpiSetElementName" on page 6598
- "cpiSetElementType" on page 6601
- "cpiSetElementValue group" on page 6604
- "cpiSetElementValueValue" on page 6606
- "cpiSetNameFromBuffer" on page 6607

cpiAddAfter:

This function adds a new (and currently unattached) syntax element to the syntax element tree after the specified target element. The newly added element becomes the *next sibling* of the target element.

Syntax

```
void cpiAddAfter(
  int*         returnCode,
  CciElement*  targetElement,
  CciElement*  newElement);
```

Parameters

returnCode
Receives the return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT

targetElement
Specifies the address of the target syntax element object (input).

newElement
Specifies the address of the new syntax element object that is to be added to the tree structure (input).

Return values

None. If an error occurs, *returnCode* indicates the reason for the error.

Sample

```
void cpiSetElementValue(
  CciParser*       parser,
  CciElement*      element,
  CciElementValue* value
){
  CciElement* newElement;
  int         rc;

  if ((cpiElementType(&rc, element) == CCI_ELEMENT_TYPE_VALUE) ||
      (cpiElementType(&rc, element) == CCI_ELEMENT_TYPE_NAME_VALUE))  {
    cpiSetElementValueValue(&rc, element, value);
  }
  else if (cpiElementType(&rc, element) == CCI_ELEMENT_TYPE_NAME) {
    /* Create a new value element, add after the current value element,
   and set the value */
    newElement = cpiCreateElement(&rc, parser);
    cpiSetElementType(&rc, newElement, CCI_ELEMENT_TYPE_VALUE);
    cpiSetElementValueValue(&rc, newElement, value);
    cpiAddAfter(&rc, element, newElement);
  }
  else {
  }

  return;
}
```

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"cpiAddBefore" on page 6544
This function adds a new (and currently unattached) syntax element to the syntax element tree before the specified target element. The newly added element becomes the *previous sibling* of the target element.

"C parser utility functions" on page 6539
A user-defined parser can call functions provided by the broker to create or define objects, such as message parser factories.

cpiAddAsFirstChild:

This function adds a new (and currently unattached) syntax element to the syntax element tree as the first child of the specified target element.

Syntax

```
void cpiAddAsFirstChild(
  int*         returnCode,
  CciElement*  targetElement,
  CciElement*  newElement);
```

Parameters

returnCode
Receives the return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT

targetElement
Specifies the address of the target syntax element object (input).

newElement
Specifies the address of the new syntax element object that is to be added to the tree structure (input).

Return values

None. If an error occurs, *returnCode* indicates the reason for the error.

Sample

This example is taken from the sample parser file `BipSampPluginParser.c`:

```
void cpiSetElementValue(
  CciParser*       parser,
  CciElement*      element,
  CciElementValue* value
){
  CciElement* newElement;
  int         rc;

  if ((cpiElementType(&rc, element) == CCI_ELEMENT_TYPE_VALUE) ||
      (cpiElementType(&rc, element) == CCI_ELEMENT_TYPE_NAME_VALUE))  {
    cpiSetElementValueValue(&rc, element, value);
  }
  else if (cpiElementType(&rc, element) == CCI_ELEMENT_TYPE_NAME) {
    /* Create a new value element, add as a first child, and set the value */
    newElement = cpiCreateElement(&rc, parser);
    cpiSetElementType(&rc, newElement, CCI_ELEMENT_TYPE_VALUE);
    cpiSetElementValueValue(&rc, newElement, value);
    cpiAddAsFirstChild(&rc, element, newElement);
  }
  else {
  }

  return;
}
```

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and

structure.

Related reference:

"cpiAddAsLastChild"
This function adds a new (and currently unattached) syntax element to the syntax element tree as the last child of the specified target element.

"C parser utility functions" on page 6539
A user-defined parser can call functions provided by the broker to create or define objects, such as message parser factories.

cpiAddAsLastChild:

This function adds a new (and currently unattached) syntax element to the syntax element tree as the last child of the specified target element.

Syntax

```
void cpiAddAsLastChild(
  int*         returnCode,
  CciElement*  targetElement,
  CciElement*  newElement);
```

Parameters

returnCode
Receives the return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT

targetElement
Specifies the address of the target syntax element object (input).

newElement
Specifies the address of the new syntax element object that is to be added to the tree structure (input).

Return values

None. If an error occurs, *returnCode* indicates the reason for the error.

Sample

This example is taken from the sample parser file `BipSampPluginParser.c`:

```
/* Convert the attribute value into broker form */
          data = CciNString((char *)startMarker, markedSize, pc->iCcsid);

/* Create a new name-value element for the attribute */
          newElement = cpiCreateElement(&rc, parser);
          cpiSetElementType(&rc, newElement, CCI_ELEMENT_TYPE_NAME_VALUE);
          cpiSetElementName(&rc, newElement, data);

          /* Free the memory created in CciNString() */
          free((void *)data);

          /* Add the element */
          cpiAddAsLastChild(&rc, element, newElement);
```

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"cpiAddAsFirstChild" on page 6541
This function adds a new (and currently unattached) syntax element to the syntax element tree as the first child of the specified target element.

"C parser utility functions" on page 6539
A user-defined parser can call functions provided by the broker to create or define objects, such as message parser factories.

cpiAddBefore:

This function adds a new (and currently unattached) syntax element to the syntax element tree before the specified target element. The newly added element becomes the *previous sibling* of the target element.

Syntax

```
void cpiAddBefore(
  int*         returnCode,
  CciElement*  targetElement,
  CciElement*  newElement);
```

Parameters

returnCode
Receives the return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT

targetElement
Specifies the address of the target syntax element object (input).

newElement
Specifies the address of the new syntax element object that is to be added to the tree structure (input).

Return values

None. If an error occurs, *returnCode* indicates the reason for the error.

Sample

```
void cpiSetElementValue(
  CciParser*       parser,
  CciElement*      element,
  CciElementValue* value
){
  CciElement* newElement;
  int         rc;

  if ((cpiElementType(&rc, element) == CCI_ELEMENT_TYPE_VALUE) ||
      (cpiElementType(&rc, element) == CCI_ELEMENT_TYPE_NAME_VALUE))  {
    cpiSetElementValueValue(&rc, element, value);
  }
  else if (cpiElementType(&rc, element) == CCI_ELEMENT_TYPE_NAME) {
    /* Create a new value element, add before the current value element,
   and set the value */
    newElement = cpiCreateElement(&rc, parser);
    cpiSetElementType(&rc, newElement, CCI_ELEMENT_TYPE_VALUE);
    cpiSetElementValueValue(&rc, newElement, value);
    cpiAddBefore(&rc, element, newElement);
  }
  else {
  }

  return;
}
```

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"cpiAddAfter" on page 6540
This function adds a new (and currently unattached) syntax element to the syntax element tree after the specified target element. The newly added element becomes the *next sibling* of the target element.

"C parser utility functions" on page 6539
A user-defined parser can call functions provided by the broker to create or define objects, such as message parser factories.

cpiAppendToBuffer:

This function appends data to the buffer containing the bit stream representation of a message, for the specified parser object.

Syntax

```
void cpiAppendToBuffer(
  int*        returnCode,
  CciParser*  parser,
  CciByte*    data,
  CciSize     length);
```

Parameters

returnCode
Receives the return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_PARSER_OBJECT
- CCI_INV_DATA_POINTER
- CCI_INV_LENGTH

parser
Specifies the address of the parser object (input).

data
The address of the data to be appended to the buffer (input).

length
The size in bytes of the data to be appended to the buffer (input).

Return values

None. If an error occurs, *returnCode* indicates the reason for the error.

Sample

This example is taken from the sample parser file `BipSampPluginParser.c`:

```
cpiAppendToBuffer(&rc, parser, (char *)"Some test data", 14);
```

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"cpiBufferByte" on page 6547
This function gets a single byte from the buffer containing the bit stream representation of the input message, for the specified parser object. The value of the index argument indicates which byte in the byte array is to be returned.

"cpiBufferPointer" on page 6548
This function gets a pointer to the buffer containing the bit stream representation

of the input message, for the specified parser object.

"cpiBufferSize" on page 6549
This function gets the size of the buffer that contains the bit stream representation of the input message, for the specified parser object.

"C parser utility functions" on page 6539
A user-defined parser can call functions provided by the broker to create or define objects, such as message parser factories.

cpiBufferByte:

This function gets a single byte from the buffer containing the bit stream representation of the input message, for the specified parser object. The value of the index argument indicates which byte in the byte array is to be returned.

Syntax

```
CciByte cpiBufferByte(
  int*        returnCode,
  CciParser*  parser,
  CciSize     index);
```

Parameters

returnCode
Receives the return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_PARSER_OBJECT
- CCI_NO_BUFFER_EXISTS

parser
Specifies the address of the parser object (input).

index
Specifies the offset to use as an index into the buffer (input).

Return values

The requested byte is returned. If an error occurs, *returnCode* indicates the reason for the error.

Sample

This example is taken from the sample parser file `BipSampPluginParser.c`:

```
void advance(
  PARSER_CONTEXT_ST* context,
  CciParser*         parser
){
  int rc = 0;

  /* Advance to the next character */
  context->iIndex++;

  /* Detect and handle the end condition */
  if (context->iIndex == context->iSize) return;
```

```
  /* Obtain the next character from the buffer */
  context->iCurrentCharacter = cpiBufferByte(&rc, parser, context->iIndex);
}
```

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"cpiAppendToBuffer" on page 6545
This function appends data to the buffer containing the bit stream representation of a message, for the specified parser object.

"cpiBufferPointer"
This function gets a pointer to the buffer containing the bit stream representation of the input message, for the specified parser object.

"cpiBufferSize" on page 6549
This function gets the size of the buffer that contains the bit stream representation of the input message, for the specified parser object.

"C parser utility functions" on page 6539
A user-defined parser can call functions provided by the broker to create or define objects, such as message parser factories.

cpiBufferPointer:

This function gets a pointer to the buffer containing the bit stream representation of the input message, for the specified parser object.

Syntax

```
const CciByte* cpiBufferPointer(
  int*        returnCode,
  CciParser*  parser);
```

Parameters

returnCode
Receives the return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_PARSER_OBJECT
- CCI_NO_BUFFER_EXISTS

parser
Specifies the address of the parser object (input).

Return values

If successful, the address of the buffer is returned. Otherwise, a value of zero (CCI_NULL_ADDR) is returned, and *returnCode* indicates the reason for the error.

Sample

This example is taken from the sample parser file `BipSampPluginParser.c`:

```
int cpiParseBufferEncoded(
  CciParser*  parser,
  CciContext* context,
  int         encoding,
  int         ccsid
){
  PARSER_CONTEXT_ST* pc = (PARSER_CONTEXT_ST *)context ;
  int                rc;

  /* Get a pointer to the message buffer and set the offset */
  pc->iBuffer = (void *)cpiBufferPointer(&rc;, parser);
  pc->iIndex = 0;
```

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"cpiAppendToBuffer" on page 6545
This function appends data to the buffer containing the bit stream representation of a message, for the specified parser object.

"cpiBufferByte" on page 6547
This function gets a single byte from the buffer containing the bit stream representation of the input message, for the specified parser object. The value of the index argument indicates which byte in the byte array is to be returned.

"cpiBufferSize"
This function gets the size of the buffer that contains the bit stream representation of the input message, for the specified parser object.

"C parser utility functions" on page 6539
A user-defined parser can call functions provided by the broker to create or define objects, such as message parser factories.

cpiBufferSize:

This function gets the size of the buffer that contains the bit stream representation of the input message, for the specified parser object.

Syntax

```
CciSize cpiBufferSize(
  int*        returnCode,
  CciParser*  parser);
```

Parameters

returnCode
Receives the return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_PARSER_OBJECT
- CCI_NO_BUFFER_EXISTS

parser
Specifies the address of the parser object (input).

Return values

If successful, the size of the buffer, in bytes, is returned. If an error occurs, zero (CCI_NULL_ADDR) is returned, and *returnCode* indicates the reason for the error.

Sample

This example is taken from the sample node file `BipSampPluginParser.c`:

```
int cpiParseBufferEncoded(
  CciParser*  parser,


  CciContext* context,
  int         encoding,
  int         ccsid
){
  PARSER_CONTEXT_ST* pc = (PARSER_CONTEXT_ST *)context ;
  int                rc;

  /* Get a pointer to the message buffer and set the offset */
  pc->iBuffer = (void *)cpiBufferPointer(&rc, parser);
  pc->iIndex = 0;

  /* Save the format of the buffer */
  pc->iEncoding = encoding;
  pc->iCcsid = ccsid;

  /* Save size of the buffer */
  pc->iSize = cpiBufferSize(&rc, parser);
```

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"cpiAppendToBuffer" on page 6545
This function appends data to the buffer containing the bit stream representation of a message, for the specified parser object.

"cpiBufferByte" on page 6547
This function gets a single byte from the buffer containing the bit stream representation of the input message, for the specified parser object. The value of the index argument indicates which byte in the byte array is to be returned.

"cpiBufferPointer" on page 6548
This function gets a pointer to the buffer containing the bit stream representation of the input message, for the specified parser object.

"C parser utility functions" on page 6539
A user-defined parser can call functions provided by the broker to create or define objects, such as message parser factories.

cpiCreateAndInitializeElement:

This function creates a syntax element, owned by the specified parser, that is not attached to a syntax tree. The element is partially initialized with the values of the *type*, *name*, *firstChildComplete*, and *lastChildComplete* parameters.

Syntax

```
CciElement* cpiCreateAndInitializeElement(
  int*            returnCode,
  CciParser*      parser,
  CciElementType  type,
  const CciChar*  name,
  CciBool         firstChildComplete,
  CciBool         lastChildComplete);
```

Parameters

returnCode
Receives the return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_FAILURE
- CCI_INV_PARSER_OBJECT

parser
Specifies the address of the parser object (input). This address is passed to the parser as a parameter of the cpiCreateContext implementation function.

type
Specifies the type of the element being created (input).

name
Specifies a descriptive name for the element (input).

firstChildComplete
Specifies a value for the firstChildComplete flag of the syntax element (input).

lastChildComplete
Specifies a value for the lastChildComplete flag of the syntax element (input).

Return values

If successful, the address of the new element object is returned. Otherwise, a value of zero (CCI_NULL_ADDR) is returned, and *returnCode* indicates the reason for the error.

Sample

```
/* Advance to the end of the value */
          while (pc->iCurrentCharacter != quoteChar) {
            advance( (PARSER_CONTEXT_ST *)context, parser );
          }

          /* Get a pointer to the end of the tag */
          endMarker = (char*)pc->iBuffer+(int)pc->iIndex;

          /* Compute the size of the tag */
          markedSize = (size_t)endMarker-(int)startMarker;

          /* Convert the attribute value into broker form */
          data = CciNString((char *)startMarker, markedSize, pc->iCcsid);

          /* Create a new name-value element for the attribute */
          newElement = cpiCreateAndInitializeElement(&rc, parser, type, name);
          cpiSetElementType(&rc, newElement, CCI_ELEMENT_TYPE_NAME_VALUE);
          cpiSetElementName(&rc, newElement, data);
          if (pc->trace) {
            const char * mbData = mbString(data, pc->iCcsid);
            fprintf(pc->tracefile, "PLUGIN: Created new NAMEVALUE element;
                    object=0x%x type=0x%x name=",
                    newElement, CCI_ELEMENT_TYPE_NAME_VALUE);
            fprintf(pc->tracefile, "%s\n", mbData);
            fflush(pc->tracefile);
            free((void *)mbData);
          }
          /* Free the memory created in CciNString() */
          free((void *)data);
```

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"cpiCreateElement" on page 6554
This function creates a default syntax element that is not attached to a syntax tree. The element is owned by the specified parser. The element is incomplete in that none of its attributes (such as type or name) are set.

"C parser utility functions" on page 6539
A user-defined parser can call functions provided by the broker to create or define objects, such as message parser factories.

cpiCreateContext:

This function creates a user-defined extension context associated with a parser object. It is called by the broker when an instance of a parser object is constructed or allocated. This action occurs when a message flow causes the message data to be parsed; the broker constructs or allocates a parser object to acquire the appropriate section of the message data.

Before this function is called, the broker creates a name element as the effective root element for the parser. However, this element is not named. The parser must name this element in the cpiSetElementName function.

The responsibilities of the extension are to:

1. Allocate all parser-instance specific data areas (such as context) that might be required.
2. Perform all additional resource acquisition or initialization that might be required.
3. Return the address of the context to the calling function. Whenever an implementation function for this parser instance is called, the appropriate context is passed as an argument to that function. Therefore, a user-defined parser developed in C need not maintain its own static pointers to per-instance data areas.

Defined In	Type	Member
CPI_VFT	Mandatory	iFpCreateContext

Syntax

```
void cpiCreateContext(
  CciParser*   parser);
```

Parameters

parser
The address of the parser object (input).

Return values

If successful, the address of the user-defined extension context is returned. Otherwise, a value of zero is returned.

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related reference:

"C parser implementation functions" on page 6538
A user-defined parser implements its capability through a function interface which is called by the broker during runtime operation. This interface includes functions to create and delete all local context storage that is associated with a parser object and the parsing operations.

cpiCreateElement:

This function creates a default syntax element that is not attached to a syntax tree. The element is owned by the specified parser. The element is incomplete in that none of its attributes (such as type or name) are set.

Syntax

```
CciElement* cpiCreateElement(
  int*        returnCode,
  CciParser*  parser);
```

Parameters

returnCode
Receives the return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_FAILURE
- CCI_INV_PARSER_OBJECT

parser
Specifies the address of the parser object (input).

Return values

If successful, the address of the new element object is returned. Otherwise, a value of zero (CCI_NULL_ADDR) is returned, and *returnCode* indicates the reason for the error.

Sample

This example is taken from the sample parser file `BipSampPluginParser.c`:

```
/* Advance to the end of the value */
          while (pc->iCurrentCharacter != quoteChar) {
            advance( (PARSER_CONTEXT_ST *)context, parser );
          }

          /* Get a pointer to the end of the tag */
          endMarker = (char*)pc->iBuffer+(int)pc->iIndex;

          /* Compute the size of the tag */
          markedSize = (size_t)endMarker-(int)startMarker;

          /* Convert the attribute value into broker form */
          data = CciNString((char *)startMarker, markedSize, pc->iCcsid);

          /* Create a new name-value element for the attribute */
          newElement = cpiCreateElement(&rc, parser);
          cpiSetElementType(&rc, newElement, CCI_ELEMENT_TYPE_NAME_VALUE);
          cpiSetElementName(&rc, newElement, data);
          if (pc->trace) {
            const char * mbData = mbString(data, pc->iCcsid);
```

```
    fprintf(pc->tracefile, "PLUGIN: Created new NAMEVALUE element;
            object=0x%x type=0x%x name=",
            newElement, CCI_ELEMENT_TYPE_NAME_VALUE);
    fprintf(pc->tracefile, "%s\n", mbData);
    fflush(pc->tracefile);
    free((void *)mbData);
  }
  /* Free the memory created in CciNString() */
  free((void *)data);
```

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"cpiCreateAndInitializeElement" on page 6551
This function creates a syntax element, owned by the specified parser, that is not attached to a syntax tree. The element is partially initialized with the values of the *type*, *name*, *firstChildComplete*, and *lastChildComplete* parameters.

"C parser utility functions" on page 6539
A user-defined parser can call functions provided by the broker to create or define objects, such as message parser factories.

cpiCreateParserFactory:

This function creates a single instance of the named parser factory in the broker.

Purpose

It must be called only in the initialization function bipGetParserFactory, which is called when the LIL file is loaded by the broker. If cpiCreateParserFactory is called at a later time, the results are unpredictable.

Syntax

```
CciFactory* cpiCreateParserFactory(
  int*      returnCode,
  CciChar*  name);
```

Parameters

returnCode

Receives the return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_FAILURE
- CCI_INV_FACTORY_NAME

- CCI_INV_OBJECT_NAME

name
Specifies the name of the factory being created (input).

Return values

If successful, the address of the parser factory object is returned. Otherwise, a value of zero (CCI_NULL_ADDR) is returned, and *returnCode* indicates the reason for the error.

Sample

This example is taken from the sample parser file `BipSampPluginParser.c`:

```
void LilFactoryExportPrefix * LilFactoryExportSuffix bipGetParserFactory()
{
  /* Declare variables */
  CciFactory*     factoryObject;
  int             rc;
  static CPI_VFT  vftable = {CPI_VFT_DEFAULT};

  /* Before we proceed we need to initialise all the static constants */
  /* that may be used by the plug-in.                                 */
  initParserConstants();

  /* Setup function table with pointers to parser implementation functions */
  vftable.iFpCreateContext            = cpiCreateContext;
  vftable.iFpParseBufferEncoded       = cpiParseBufferEncoded;
  vftable.iFpParseFirstChild          = cpiParseFirstChild;
  vftable.iFpParseLastChild           = cpiParseLastChild;
  vftable.iFpParsePreviousSibling     = cpiParsePreviousSibling;
  vftable.iFpParseNextSibling         = cpiParseNextSibling;
  vftable.iFpWriteBufferEncoded       = cpiWriteBufferEncoded;
  vftable.iFpDeleteContext            = cpiDeleteContext;
  vftable.iFpSetElementValue          = cpiSetElementValue;
  vftable.iFpElementValue             = cpiElementValue;
  vftable.iFpNextParserClassName      = cpiNextParserClassName;
  vftable.iFpSetNextParserClassName   = cpiSetNextParserClassName;
  vftable.iFpNextParserEncoding       = cpiNextParserEncoding;
  vftable.iFpNextParserCodedCharSetId = cpiNextParserCodedCharSetId;

  /* Create the parser factory for this plugin */
  factoryObject = cpiCreateParserFactory(&rc, constParserFactory);
  if (factoryObject) {
    /* Define the classes of message supported by the factory */
    cpiDefineParserClass(&rc, factoryObject, constPXML, &vftable);
  }
  else {
    /* Error: Unable to create parser factory */
  }

  /* Return address of this factory object to the broker */
  return(factoryObject);
}
```

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to

extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"cpiDefineParserClass"
This function defines the name of a parser class that is supported by a parser factory.

"C parser utility functions" on page 6539
A user-defined parser can call functions provided by the broker to create or define objects, such as message parser factories.

cpiDefineParserClass:

This function defines the name of a parser class that is supported by a parser factory.

functbl is a pointer to a virtual function table that contains pointers to the C implementation functions; that is, those functions that provide the function of the parser itself.

Syntax

```
void cpiDefineParserClass(
  int*         returnCode,
  CciFactory*  factoryObject,
  CciChar*     name,
  CPI_VFT*     functbl);
```

Parameters

returnCode
Receives the return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_FACTORY_OBJECT
- CCI_INV_PARSER_NAME
- CCI_PARSER_NAME_TOO_LONG
- CCI_INV_OBJECT_NAME
- CCI_INV_VFTP
- CCI_MISSING_IMPL_FUNCTION
- CCI_INV_IMPL_FUNCTION
- CCI_NAME_EXISTS

factoryObject
Specifies the address of the factory object that supports the named parser (input). The address is returned from **cpiCreateParserFactory**.

name
The name of the parser class to be defined (input). The maximum length of a parser class name is 8 characters.

functbl
The address of the CPI_VFT structure that contains pointers to the implementation functions (input).

Return values

None. If an error occurs, *returnCode* indicates the reason for the error.

Sample

This example is taken from the sample parser file BipSampPluginParser.c:

```
void LilFactoryExportPrefix * LilFactoryExportSuffix bipGetParserFactory()
{
  /* Declare variables */
  CciFactory*     factoryObject;
  int             rc;
  static CPI_VFT  vftable = {CPI_VFT_DEFAULT};

  /* Before we proceed we need to initialise all the static constants */
  /* that may be used by the plug-in.                                 */
  initParserConstants();

  /* Setup function table with pointers to parser implementation functions */
  vftable.iFpCreateContext            = cpiCreateContext;
  vftable.iFpParseBufferEncoded       = cpiParseBufferEncoded;
  vftable.iFpParseFirstChild          = cpiParseFirstChild;
  vftable.iFpParseLastChild           = cpiParseLastChild;
  vftable.iFpParsePreviousSibling     = cpiParsePreviousSibling;
  vftable.iFpParseNextSibling         = cpiParseNextSibling;
  vftable.iFpWriteBufferEncoded       = cpiWriteBufferEncoded;
  vftable.iFpDeleteContext            = cpiDeleteContext;
  vftable.iFpSetElementValue          = cpiSetElementValue;
  vftable.iFpElementValue             = cpiElementValue;
  vftable.iFpNextParserClassName      = cpiNextParserClassName;
  vftable.iFpSetNextParserClassName   = cpiSetNextParserClassName;
  vftable.iFpNextParserEncoding       = cpiNextParserEncoding;
  vftable.iFpNextParserCodedCharSetId = cpiNextParserCodedCharSetId;

  /* Create the parser factory for this plugin */
  factoryObject = cpiCreateParserFactory(&rc, constParserFactory);
  if (factoryObject) {
    /* Define the classes of message supported by the factory */
    cpiDefineParserClass(&rc, factoryObject, constPXML, &vftable);
  }
  else {
    /* Error: Unable to create parser factory */
  }

  /* Return address of this factory object to the broker */
  return(factoryObject);
}
```

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"cpiCreateParserFactory" on page 6555
This function creates a single instance of the named parser factory in the broker.

"C parser utility functions" on page 6539
A user-defined parser can call functions provided by the broker to create or define objects, such as message parser factories.

cpiDeleteContext:

This function deletes the context owned by the parser object. It is invoked by the message broker when an instance of a parser object is destroyed.

The responsibilities of the parser are to:

1. Release all parser-instance specific data areas (such as context) that were acquired at construction or during parser processing.
2. Release all additional resources that might have been acquired for the processing of the parser.

Defined In	Type	Member
CPI_VFT	Optional	iFpDeleteContext

Syntax

```
void cpiDeleteContext(
  CciParser*   parser,
  CciContext*  context);
```

Parameters

parser
The address of the parser object (input).

context
The address of the context owned by the parser object (input).

Return values

None.

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"C parser implementation functions" on page 6538
A user-defined parser implements its capability through a function interface which is called by the broker during runtime operation. This interface includes functions to create and delete all local context storage that is associated with a parser object and the parsing operations.

cpiElementCompleteNext:

This function gets the value of the 'next child complete' flag from the target syntax element. This attribute indicates whether the element tree is complete.

Syntax

```
CciBool cpiElementCompleteNext(
  int*         returnCode,
  CciElement*  targetElement);
```

Parameters

returnCode
Receives the return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT

targetElement
Specifies the address of the target syntax element object (input).

Return values

The value of the attribute is returned. If an error occurs, *returnCode* indicates the reason for the error.

Sample

This example is taken from the sample parser file `BipSampPluginParser.c`:

```
if ((!cpiElementCompleteNext(&rc, element)) &&
      (cpiElementType(&rc, element) == CCI_ELEMENT_TYPE_NAME)) {

    while ((!cpiElementCompleteNext(&rc, element))     &&
           (!cpiFirstChild(&rc, element)) &&
           (pc->iCurrentElement))
    {
      pc->iCurrentElement = parseNextItem(parser, context, pc->iCurrentElement);
    }
```

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"cpiElementCompletePrevious"
This function gets the value of the 'previous child complete' flag from the target syntax element. This attribute indicates whether the element tree is complete.

"C parser utility functions" on page 6539
A user-defined parser can call functions provided by the broker to create or define objects, such as message parser factories.

cpiElementCompletePrevious:

This function gets the value of the 'previous child complete' flag from the target syntax element. This attribute indicates whether the element tree is complete.

Syntax

```
CciBool cpiElementCompletePrevious(
  int*         returnCode,
  CciElement*  targetElement);
```

Parameters

returnCode
Receives the return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT

targetElement
Specifies the address of the target syntax element object (input).

Return values

The value of the attribute is returned. If an error occurs, *returnCode* indicates the reason for the error.

Sample

This example is based on code taken from the sample parser file `BipSampPluginParser.c`. In the sample file, the code uses cpiElementCompleteNext.

```
if ((!cpiElementCompletePrevious(&rc, element)) &&
      (cpiElementType(&rc, element) == CCI_ELEMENT_TYPE_NAME)) {

    while ((!cpiElementCompletePrevious(&rc, element))     &&
           (!cpiFirstChild(&rc, element)) &&
           (pc->iCurrentElement))
    {
      pc->iCurrentElement = parsePreviousItem(parser, context, pc->iCurrentElement);
    }
```

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from

the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"cpiElementCompleteNext" on page 6560
This function gets the value of the 'next child complete' flag from the target syntax element. This attribute indicates whether the element tree is complete.

"C parser utility functions" on page 6539
A user-defined parser can call functions provided by the broker to create or define objects, such as message parser factories.

cpiElementName:

This function gets the name of the target syntax element. The syntax element name must be set previously by using cniSetElementName or cpiSetElementName.

Syntax

```
Ccisize           cpiElementName(
  int*            returnCode,
  CciElement*     targetElement,
  const CciChar*  value,
  CciSize         length);
```

Parameters

returnCode
Receives the return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT
- CCI_INV_DATA_POINTER
- CCI_INV_DATA_BUFLEN
- CCI_INV_BUFFER_TOO_SMALL

targetElement
Specifies the address of the target syntax element object (input).

value
Specifies the address of a buffer into which the element name is copied (input).

length
The length, in characters, specified by the *value* parameter (input).

Return values

If successful, the element name is copied into the supplied buffer and the number of *CciChar* characters copied is returned. If the buffer is not large enough to contain the element name, *returnCode* is set to CCI_BUFFER_TOO_SMALL and the number of characters required is returned. For all other failures, CCI_FAILURE is returned

and *returnCode* indicates the reason for the error.

Sample

```
cpiElementName(&rc;, element, (CciChar*)&elementName;, sizeof(elementName));
```

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"cpiElementType" on page 6565
This function gets the type of the target syntax element. The syntax element type must be set previously by using cniSetElementType or cpiSetElementType.

"cpiElementValue group" on page 6567
This group of functions retrieve the value of the specified syntax element.

"cpiElementValueValue" on page 6569
This function gets the value object from the specified syntax element. This value object is opaque in that it cannot be interrogated. It can be used to set or derive the value of one element from another, without knowing its type, by using the cpiSetElementValueValue function.

"C parser utility functions" on page 6539
A user-defined parser can call functions provided by the broker to create or define objects, such as message parser factories.

cpiElementNameSpace:

This function retrieves the value of the *namespace* attribute for the specified syntax element.

Defined In	Type	Member
CPI_VFT	Optional	iFpElementValue

Syntax

```
CciSize cpiElementNamespace(
   int*           returnCode,
   CciElement*    targetElement,
   const CciChar* value,
   CciSize        length);
```

Parameters

returnCode

A NULL pointer input signifies that the user-defined node does not want to deal with errors. All exceptions thrown during the execution of this call are re-thrown to the next upstream node in the flow. If input is not NULL, output

signifies the success status of the call. If an exception occurs during execution, *returnCode is set to CCI_EXCEPTION on output. You can call CciGetLastExceptionData to get details of the exception. If the caller did not allocate enough memory to hold the namespace value, *returncode is set to CCI_BUFFER_TOO_SMALL.

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT
- CCI_INV_DATA_POINTER
- CCI_INV_DATA_BUFLEN
- CCI_INV_BUFFER_TOO_SMALL

targetElement
Specifies the address of the target syntax element object.

value
Specifies the address of a buffer into which the element namespace value is copied. A string of characters (including a NULL terminator) representing the namespace value is copied into this buffer. The buffer must be a portion of memory previously allocated by the caller.

length
The length in CciChars of the buffer specified by the value parameter.

Return values

If successful, the number of CciChars copied into the buffer is returned.

If the buffer is not large enough to contain the attribute value, *returnCode* is set to CCI_BUFFER_TOO_SMALL, and the number of bytes CciChars required is returned.

Sample

```
elementNamespace=(CciChar*)malloc(sizeof(CciChar) * elementNamespaceLength);
    elementNamespaceLength = cpiElementNamespace(&rc;,
                        element,
                        elementNamespace,
                        elementNamespaceLength);

    if (rc==CCI_BUFFER_TOO_SMALL){
      free(elementNamespace);
      elementNamespace=(CciChar*)malloc(sizeof(CciChar) * elementNamespaceLength);
      elementNamespaceLength = cpiElementNamespace(&rc;,
                        element,
                        elementNamespace,
                        elementNamespaceLength);
    }
    checkRC(rc);
```

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to

extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"cpiSetElementNamespace" on page 6599
Use this function to set the namespace attribute for the specified syntax element.

"C parser implementation functions" on page 6538
A user-defined parser implements its capability through a function interface which is called by the broker during runtime operation. This interface includes functions to create and delete all local context storage that is associated with a parser object and the parsing operations.

cpiElementType:

This function gets the type of the target syntax element. The syntax element type must be set previously by using cniSetElementType or cpiSetElementType.

Syntax

```
CciElementType cpiElementType(
  int*         returnCode,
  CciElement*  targetElement);
```

Parameters

returnCode
Receives the return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT

targetElement
Specifies the address of the target syntax element object (input).

Return values

The value of the element type is returned. If an error occurs, *returnCode* indicates the reason for the error.

Sample

This example is taken from the sample parser file `BipSampPluginParser.c`:

```
if ((!cpiElementCompleteNext(&rc, element)) &&
      (cpiElementType(&rc, element) == CCI_ELEMENT_TYPE_NAME)) {

    while ((!cpiElementCompleteNext(&rc, element))     &&
           (!cpiFirstChild(&rc, element)) &&
           (pc->iCurrentElement))
    {
      pc->iCurrentElement = parseNextItem(parser, context, pc->iCurrentElement);
    }
```

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"cpiElementName" on page 6562
This function gets the name of the target syntax element. The syntax element name must be set previously by using cniSetElementName or cpiSetElementName.

"cpiElementValue group" on page 6567
This group of functions retrieve the value of the specified syntax element.

"cpiElementValueValue" on page 6569
This function gets the value object from the specified syntax element. This value object is opaque in that it cannot be interrogated. It can be used to set or derive the value of one element from another, without knowing its type, by using the cpiSetElementValueValue function.

"C parser utility functions" on page 6539
A user-defined parser can call functions provided by the broker to create or define objects, such as message parser factories.

cpiElementValue:

This function gets the value of a specified element. It is called by the broker when the value of a syntax element is to be retrieved. It provides an opportunity for a user-defined parser to override the behavior for retrieving element values.

Defined In	Type	Member
CPI_VFT	Optional	iFpElementValue

Syntax

```
const CciElementValue* cpiElementValue(
  CciParser*   parser,
  CciElement*  currentElement);
```

Parameters

parser
The address of the parser object (input).

currentElement
The address of the current syntax element (input).

Return values

The value of the target syntax element object is returned. This value has been returned by the cpiElementValueValue function.

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"cpiSetElementValue" on page 6602
This optional function sets the value of a specified element.

"C parser implementation functions" on page 6538
A user-defined parser implements its capability through a function interface which is called by the broker during runtime operation. This interface includes functions to create and delete all local context storage that is associated with a parser object and the parsing operations.

cpiElementValue group:

This group of functions retrieve the value of the specified syntax element.

Specify the appropriate function from this group that matches the type of data to be retrieved:

- cpiElementBitArrayValue
- cpiElementBooleanValue
- cpiElementByteArrayValue
- cpiElementCharacterValue
- cpiElementDateValue
- cpiElementDecimalValue
- cpiElementGmtTimestampValue
- cpiElementGmtTimeValue
- cpiElementIntegerValue
- cpiElementRealValue
- cpiElementTimestampValue
- cpiElementTimeValue

Syntax

```
CciSize cpiElementBitArrayValue(
  int*         returnCode,
  CciElement*  targetElement,
  const struct CciBitArray* value);
CciBool cpiElementBooleanValue(
  int*         returnCode,
  CciElement*  targetElement);
CciSize cpiElementByteArrayValue(
  int*         returnCode,
  CciElement*  targetElement,
  const struct CciByteArray* value);
```

```
CciSize cpiElementCharacterValue(
  int*            returnCode,
  CciElement*     targetElement,
  const CciChar*  value,
  CciSize         length);

struct CciDate cpiElementDateValue(
  int*         returnCode,
  CciElement*  targetElement);

CciSize cpiElementDecimalValue(
  int*            returnCode,
  CciElement*     targetElement,
  const CciChar*  value,
  CciSize         length);

struct CciTimestamp cpiElementGmtTimestampValue(
  int*         returnCode,
  CciElement*  targetElement);

struct CciTime cpiElementGmtTimeValue(
  int*         returnCode,
  CciElement*  targetElement);

CciInt cpiElementIntegerValue(
  int*         returnCode,
  CciElement*  targetElement);

CciReal cpiElementRealValue(
  int*         returnCode,
  CciElement*  targetElement);

struct CciTimestamp cpiElementTimestampValue(
  int*         returnCode,
  CciElement*  targetElement);

struct CciTime cpiElementTimeValue(
  int*         returnCode,
  CciElement*  targetElement);
```

Parameters

returnCode
Receives the return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT
- CCI_INV_DATA_POINTER
- CCI_INV_DATA_BUFLEN
- CCI_INV_BUFFER_TOO_SMALL

targetElement
Specifies the address of the target syntax element object (input).

value
The address of an output buffer into which the value of the syntax element is stored (input). Used on relevant function calls only.

length
The length of the output buffer, in characters, specified by the *value* parameter (input). Used on relevant function calls only.

Return values

The value of the element is returned.

In some cases, if the buffer is not large enough to receive the data, the data is not written into the buffer. The size of the required buffer is passed as the return value, and *returnCode* is set to CCI_BUFFER_TOO_SMALL. For example, cpiElementCharacterValue or cpiElementDecimalValue use this technique.

If an error occurs, *returnCode* indicates the reason for the error.

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"cpiElementName" on page 6562
This function gets the name of the target syntax element. The syntax element name must be set previously by using cniSetElementName or cpiSetElementName.

"cpiElementType" on page 6565
This function gets the type of the target syntax element. The syntax element type must be set previously by using cniSetElementType or cpiSetElementType.

"cpiElementValueValue"
This function gets the value object from the specified syntax element. This value object is opaque in that it cannot be interrogated. It can be used to set or derive the value of one element from another, without knowing its type, by using the cpiSetElementValueValue function.

"C parser utility functions" on page 6539
A user-defined parser can call functions provided by the broker to create or define objects, such as message parser factories.

cpiElementValueValue:

This function gets the value object from the specified syntax element. This value object is opaque in that it cannot be interrogated. It can be used to set or derive the value of one element from another, without knowing its type, by using the cpiSetElementValueValue function.

This function can be used by parsers that override behavior by calling the implementation functions cpiElementValue and cpiSetElementValue.

Syntax

```
const CciElementValue* cpiElementValueValue(
  int*         returnCode,
  CciElement*  targetElement);
```

Parameters

returnCode
Receives the return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT

targetElement
Specifies the address of the target syntax element object (input).

Return values

The address of the *CciElementValue* object stored in the specified target syntax element is returned. If an error occurs, zero (CCI_NULL_ADDR) is returned and *returnCode* indicates the reason for the error.

Sample

This example is taken from the sample parser file `BipSampPluginParser.c`:

```
const CciElementValue* cpiElementValue(
  CciParser*  parser,
  CciElement* element
){
  CciElement* firstChild;
  const CciElementValue* value;
  int         rc;

  if ((cpiElementType(&rc, element) == CCI_ELEMENT_TYPE_VALUE) ||
      (cpiElementType(&rc, element) == CCI_ELEMENT_TYPE_NAME_VALUE)) {
    value = cpiElementValueValue(&rc, element);
  }
  else if (cpiElementType(&rc, element) == CCI_ELEMENT_TYPE_NAME) {
    firstChild = cniFirstChild(&rc, element);
    value = cpiElementValueValue(&rc, firstChild);
  }
  else {
  }

  return(value);
}
```

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"cpiElementName" on page 6562
This function gets the name of the target syntax element. The syntax element name must be set previously by using cniSetElementName or cpiSetElementName.

"cpiElementType" on page 6565
This function gets the type of the target syntax element. The syntax element type

must be set previously by using cniSetElementType or cpiSetElementType.

"cpiElementValue group" on page 6567
This group of functions retrieve the value of the specified syntax element.

"C parser utility functions" on page 6539
A user-defined parser can call functions provided by the broker to create or define objects, such as message parser factories.

cpiFirstChild:

This function returns the address of the syntax element object that is the first child of the specified target element.

Syntax

```
CciElement* cpiFirstChild(
  int*                returnCode,
  const CciElement*   targetElement);
```

Parameters

returnCode
Receives the return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT

targetElement
Specifies the address of the target syntax element object (input).

Return values

The address of the requested syntax element object is returned, unless there is no child in which case zero is returned. If an error occurs, zero (CCI_NULL_ADDR) is returned and *returnCode* indicates the reason for the error.

Sample

This example is taken from the sample node file `BipSampPluginParser.c`:

```
while ((!cpiElementCompleteNext(&rc, element))     &&
           (!cpiFirstChild(&rc, element)) &&
           (pc->iCurrentElement))
```

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"cpiLastChild"
This function returns the address of the syntax element object that is the last child of the specified target element.

"C parser utility functions" on page 6539
A user-defined parser can call functions provided by the broker to create or define objects, such as message parser factories.

cpiLastChild:

This function returns the address of the syntax element object that is the last child of the specified target element.

Syntax

```
CciElement* cpiLastChild(
  int*               returnCode,
  const CciElement*  targetElement);
```

Parameters

returnCode
Receives the return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT

targetElement
Specifies the address of the target syntax element object (input).

Return values

The address of the requested syntax element object is returned, unless there is no child in which case zero is returned. If an error occurs, zero (CCI_NULL_ADDR) is returned and *returnCode* indicates the reason for the error.

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"cpiFirstChild" on page 6571
This function returns the address of the syntax element object that is the first child of the specified target element.

"C parser utility functions" on page 6539
A user-defined parser can call functions provided by the broker to create or define objects, such as message parser factories.

cpiNextParserClassName:

This function returns the name of the next parser class in the chain, if defined. Use this function to return to the broker the name of the parser class that handles the next section, or remainder, of the message content. Typically, for messages that have a simple format type, only one message content parser is defined.

For messages that have a more complex format type with multiple message parsers, each parser can identify the next one in the chain by returning its name in the *buffer* parameter. The last parser in the chain must return an empty string.

If you specify the name of a parser supplied with WebSphere Message Broker, you must use the correct class name of the parser.

Defined In	Type	Member
CPI_VFT	Optional	iFpNextParserClassName

Syntax

```
void cpiNextParserClassName(
  CciParser*   parser,
  CciContext*  context,
  CciChar*     buffer,
  int          size);
```

Parameters

parser
The address of the parser object (input).

context
The address of the context owned by the parser object (input).

buffer
The address of a buffer into which the parser class name should be put (input).

size
The length, in bytes, of the buffer provided by the broker (input).

Return values

None.

Sample

This example is taken from the sample parser file `BipSampPluginParser.c`:

```
void cpiNextParserClassName(
  CciParser*  parser,
  CciContext* context,
  CciChar*    buffer,
  int         size
){
  PARSER_CONTEXT_ST* pc = (PARSER_CONTEXT_ST *)context ;
  int                rc = 0;

  if (pc->trace) {
```

```
    fprintf(pc->tracefile, "PLUGIN: -> cpiNextParserClassName() parser=0x%x context=0x%x\n",
            parser, context);
    fflush(pc->tracefile);
  }

  /* Copy the name to the broker */
  CciCharNCpy(buffer, pc->iNextParserClassName, size);

  if (pc->trace) {
    fprintf(pc->tracefile, "PLUGIN: <- cpiNextParserClassName()\n");
    fflush(pc->tracefile);
  }

  return;
}
```

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"cpiNextParserCodedCharSetId"
This function returns the coded character set ID (CCSID) of the data owned by the next parser class in the chain, if one is defined.

"cpiNextParserEncoding" on page 6576
This function returns the encoding of data owned by the next parser class in the chain, if defined.

"C parser implementation functions" on page 6538
A user-defined parser implements its capability through a function interface which is called by the broker during runtime operation. This interface includes functions to create and delete all local context storage that is associated with a parser object and the parsing operations.

cpiNextParserCodedCharSetId:

This function returns the coded character set ID (CCSID) of the data owned by the next parser class in the chain, if one is defined.

Defined In	Type	Member
CPI_VFT	Optional	iFpNextParserCodedCharSetId

Syntax

```
int cpiNextParserCodedCharSetId(
  CciParser*   parser,
  CciContext*  context);
```

Parameters

parser
The address of the parser object (input).

context
The address of the context owned by the parser object (input).

Return values

The CCSID of the data is returned. If it is not known, zero might be returned, and a default CCSID is assumed.

Sample

This example is taken from the sample parser file `BipSampPluginParser.c`:

```
int cpiNextParserCodedCharSetId(
  CciParser*  parser,
  CciContext* context
){
  PARSER_CONTEXT_ST* pc = (PARSER_CONTEXT_ST *)context ;
  int                ccsid = 0;

  if (pc->trace) {
    fprintf(pc->tracefile, "PLUGIN: -> cpiNextParserCodedCharSetId() parser=0x%x
        context=0x%x\n", parser, context);
    fflush(pc->tracefile);
  }

  if (pc->trace) {
    fprintf(pc->tracefile, "PLUGIN: <- cpiNextParserCodedCharSetId()\n");
    fflush(pc->tracefile);
  }

  return ccsid;
}
```

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"cpiNextParserClassName" on page 6573
This function returns the name of the next parser class in the chain, if defined. Use this function to return to the broker the name of the parser class that handles the next section, or remainder, of the message content. Typically, for messages that have a simple format type, only one message content parser is defined.

"cpiNextParserEncoding" on page 6576
This function returns the encoding of data owned by the next parser class in the chain, if defined.

"C parser implementation functions" on page 6538
A user-defined parser implements its capability through a function interface which is called by the broker during runtime operation. This interface includes functions to create and delete all local context storage that is associated with a parser object and the parsing operations.

cpiNextParserEncoding:

This function returns the encoding of data owned by the next parser class in the chain, if defined.

Defined In	Type	Member
CPI_VFT	Optional	iFpNextParserEncoding

Syntax

```
int cpiNextParserEncoding(
  CciParser*   parser,
  CciContext*  context);
```

Parameters

parser
The address of the parser object (input).

context
The address of the context owned by the parser object (input).

Return values

The encoding of the data is returned. If it is not known, zero might be returned, and default encoding is assumed.

Sample

This example is taken from the sample parser file `BipSampPluginParser.c`:

```
int cpiNextParserEncoding(
  CciParser*  parser,
  CciContext* context
){
  PARSER_CONTEXT_ST* pc = (PARSER_CONTEXT_ST *)context ;
  int                encoding = 0;

  if (pc->trace) {
    fprintf(pc->tracefile, "PLUGIN: -> cpiNextParserEncoding() parser=0x%x context=0x%x\n",
            parser, context);
    fflush(pc->tracefile);
  }

  if (pc->trace) {
    fprintf(pc->tracefile, "PLUGIN: <- cpiNextParserEncoding()\n");
    fflush(pc->tracefile);
  }

  return encoding;
}
```

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"cpiNextParserClassName" on page 6573
This function returns the name of the next parser class in the chain, if defined. Use this function to return to the broker the name of the parser class that handles the next section, or remainder, of the message content. Typically, for messages that have a simple format type, only one message content parser is defined.

"cpiNextParserCodedCharSetId" on page 6574
This function returns the coded character set ID (CCSID) of the data owned by the next parser class in the chain, if one is defined.

"C parser implementation functions" on page 6538
A user-defined parser implements its capability through a function interface which is called by the broker during runtime operation. This interface includes functions to create and delete all local context storage that is associated with a parser object and the parsing operations.

cpiNextSibling:

This function returns the address of the syntax element object that is the next (right) sibling of the specified target element.

Syntax

```
CciElement* cpiNextSibling(
  int*               returnCode,
  const CciElement*  targetElement);
```

Parameters

returnCode
Receives the return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT

targetElement
Specifies the address of the target syntax element object (input).

Return values

The address of the requested syntax element object is returned, unless there is no next sibling in which case zero is returned. If an error occurs, zero (CCI_NULL_ADDR) is returned and *returnCode* indicates the reason for the error.

Sample

This example is taken from the sample node file BipSampPluginParser.c:

```
while ((!cpiElementCompleteNext(&rc, cpiParent(&rc, element))) &&
         (!cpiNextSibling(&rc, element))       &&
         (pc->iCurrentElement))
```

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"C parser utility functions" on page 6539
A user-defined parser can call functions provided by the broker to create or define objects, such as message parser factories.

cpiParent:

This function returns the address of the syntax element object that is the parent of the specified target element.

Syntax

```
CciElement* cpiParent(
  int*               returnCode,
  const CciElement*  targetElement);
```

Parameters

returnCode
Receives the return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT

targetElement
Specifies the address of the target syntax element object (input).

Return values

If successful, the address of the requested syntax element is returned. If there is no parent element, zero is returned. If an error occurs, zero (CCI_NULL_ADDR) is returned and the *returnCode* parameter indicates the reason for the error.

Sample

This example is taken from the sample parser file `BipSampPluginParser.c`:

```
void* parseNextItem(
  CciParser*  parser,
  CciContext* context,
  CciElement* element
){
  void*              endMarker;
  void*              startMarker;
  PARSER_CONTEXT_ST* pc = (PARSER_CONTEXT_ST *)context;
  CciElement*        returnElement = element;
  CciElement*        newElement;
  size_t             markedSize;
  const CciChar*     data;
  int                rc;

  if (pc->trace)

  /* Skip any white space */
  skipWhiteSpace( (PARSER_CONTEXT_ST *)context );

  /* Are we at the end of the buffer? */
  if (pc->iIndex == pc->iSize)
        return(0);
  }

  /* Are we within a tag? */
  if (pc->iInTag) {

    if (pc->iCurrentCharacter == chCloseAngle) {

      /* We have reached the end of a tag */
      pc->iInTag = 0;
      advance( (PARSER_CONTEXT_ST *)context, parser );
    }
    else if (pc->iCurrentCharacter == chForwardSlash) {

      /* We may have reached the end of an empty tag */
      advance( (PARSER_CONTEXT_ST *)context, parser );

      if (pc->iCurrentCharacter == chCloseAngle) {

        pc->iInTag = 0;
        advance( (PARSER_CONTEXT_ST *)context, parser );

        cpiSetElementCompleteNext(&rc, element, 1);

        returnElement = cpiParent(&rc, element);
      }
```

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and

structure.

Related reference:

"C parser utility functions" on page 6539
A user-defined parser can call functions provided by the broker to create or define objects, such as message parser factories.

cpiParseBuffer:

This function prepares a parser to parse a new message object. It is called the first time (for each message) that the message flow causes the message content to be parsed.

This function is called for each user-defined parser that is used to parse a particular message format to complete the following actions:

- Perform all initialization that is required
- Return the length of the message content that it takes ownership for

The *offset* parameter indicates the offset within the message buffer where parsing is to commence. This indication is required because another parser might own a previous portion of the message (for example, an MQMD header is parsed by an internal parser owned by the broker). The offset must be positive and be less than the size of the buffer. Verify in this function that the offset is valid, to remove problems associated with previous parsers.

The parser must return the size of the remaining buffer for which it takes ownership. The size must be less than or equal to the size of the buffer, less the current offset.

A parser must not attempt to cause parsing of other portions of the syntax element tree, for example, by navigating to the root element and to another branch. This action can cause unpredictable results.

If this implementation function is provided in the CPI_VFT structure, you can call neither cpiParseBufferEncoded() nor cpiParseBufferFormatted(), because the cpiDefineParserClass() function fails with a return code of CCI_INVALID_IMPL_FUNCTION.

Defined In	Type	Member
CPI_VFT	Conditional	iFpParseBuffer

Syntax

```
int cpiParseBuffer(
  CciParser*   parser,
  CciContext*  context,
  int          offset);
```

Parameters

parser
The address of the parser object (input).

context
The address of the context owned by the parser object (input).

offset
The offset into the message buffer at which parsing is to commence (input).

Return values

The size (in bytes) of the remaining portion of the message buffer for which the parser takes ownership.

Sample

This example is taken from the sample parser file `BipSampPluginParser.c`:

```
int cpiParseBuffer(
  CciParser*  parser,
  CciContext* context,
  int         offset,
){
  PARSER_CONTEXT_ST* pc = (PARSER_CONTEXT_ST *)context ;
  int                rc;

  /* Get a pointer to the message buffer and set the offset */
  pc->iBuffer = (void *)cpiBufferPointer(&rc, parser);
  pc->iIndex = 0;

  /* Save size of the buffer */
  pc->iSize = cpiBufferSize(&rc, parser);

  /* Prime the first byte in the stream */
  pc->iCurrentCharacter = cpiBufferByte(&rc, parser, pc->iIndex);

  /* Set the current element to the root element */
  pc->iCurrentElement = cpiRootElement(&rc, parser);

  /* Reset flag to ensure parsing is reset correctly */
  pc->iInTag = 0;

  if (pc->trace) {
    fprintf(pc->tracefile, "PLUGIN: <- cpiParseBuffer()
    retvalue=%d\n", pc->iSize);
    fflush(pc->tracefile);
  }
```

Related concepts:

"User-defined input nodes" on page 2990
A user-defined input node is an extension to the broker that provides a new input node in addition to the nodes supplied with the product.

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"cpiParseBufferEncoded" on page 6582
This function extends the capability of the cpiParseBuffer() implementation function, and provides the encoding and coded character set that the input message is represented in.

"cpiParseBufferFormatted" on page 6583
This function extends the capability of the cpiParseBuffer() implementation function, and provides additional information about the input message.

"C parser implementation functions" on page 6538
A user-defined parser implements its capability through a function interface which is called by the broker during runtime operation. This interface includes functions to create and delete all local context storage that is associated with a parser object and the parsing operations.

cpiParseBufferEncoded:

This function extends the capability of the cpiParseBuffer() implementation function, and provides the encoding and coded character set that the input message is represented in.

If this implementation function is provided in the CPI_VFT structure, you cannot specify either cpiParseBuffer() or cpiParseBufferFormatted(); if you do, the cpiDefineParserClass() function fails with a return code of CCI_INVALID_IMPL_FUNCTION.

Defined In	Type	Member
CPI_VFT	Conditional	iFpParseBufferEncoded

Syntax

```
int cpiParseBufferEncoded(
  CciParser*   parser,
  CciContext*  context,
  int          encoding,
  int          ccsid);
```

Parameters

parser
The address of the parser object (input).

context
The address of the context owned by the parser object (input).

encoding
The encoding of the message buffer (input).

ccsid
The CCSID of the message buffer (input).

Return values

The size (in bytes) of the remaining portion of the message buffer for which the parser takes ownership.

Sample

This example is taken from the sample parser file `BipSampPluginParser.c`:

```
int cpiParseBufferEncoded(
  CciParser*  parser,
  CciContext* context,
  int         encoding,
  int         ccsid
){
  PARSER_CONTEXT_ST* pc = (PARSER_CONTEXT_ST *)context ;
  int                rc;

  /* Get a pointer to the message buffer and set the offset */
```

```
pc->iBuffer = (void *)cpiBufferPointer(&rc, parser);
pc->iIndex = 0;

/* Save the format of the buffer */
pc->iEncoding = encoding;
pc->iCcsid = ccsid;

/* Save size of the buffer */
pc->iSize = cpiBufferSize(&rc, parser);

/* Prime the first byte in the stream */
pc->iCurrentCharacter = cpiBufferByte(&rc, parser, pc->iIndex);

/* Set the current element to the root element */
pc->iCurrentElement = cpiRootElement(&rc, parser);

/* Reset flag to ensure parsing is reset correctly */
pc->iInTag = 0;

if (pc->trace) {
  fprintf(pc->tracefile, "PLUGIN: <- cpiParseBufferEncoded()
  retvalue=%d\n", pc->iSize);
  fflush(pc->tracefile);
}
```

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"cpiParseBuffer" on page 6580
This function prepares a parser to parse a new message object. It is called the first time (for each message) that the message flow causes the message content to be parsed.

"cpiParseBufferFormatted"
This function extends the capability of the cpiParseBuffer() implementation function, and provides additional information about the input message.

"C parser implementation functions" on page 6538
A user-defined parser implements its capability through a function interface which is called by the broker during runtime operation. This interface includes functions to create and delete all local context storage that is associated with a parser object and the parsing operations.

cpiParseBufferFormatted:

This function extends the capability of the cpiParseBuffer() implementation function, and provides additional information about the input message.

The following additional information is available:

1. The encoding and coded character set that the input message is represented in.
2. The message set, type, and format for the message.

If this implementation function is provided in the CPI_VFT structure, you cannot specify either cpiParseBuffer() or cpiParseBufferEncoded(); if you do, the cpiDefineParserClass() function fails with a return code of CCI_INVALID_IMPL_FUNCTION.

Defined In	Type	Member
CPI_VFT	Conditional	iFpParseBufferFormatted

Syntax

```
int cpiParseBufferFormatted(
  CciParser*     parser,
  CciContext*    context,
  int            encoding,
  int            ccsid,
  CciChar*       set,
  CciChar*       type,
  CciChar*       format);
```

Parameters

parser
The address of the parser object (input).

context
The address of the context owned by the parser object (input).

encoding
The encoding of the message buffer (input).

ccsid
The CCSID of the message buffer (input).

set
The message set to which the message belongs (input).

type
The message type (input).

format
The message format (input).

Return values

The size (in bytes) of the remaining portion of the message buffer for which the parser takes ownership.

Sample

This example is taken from the sample parser file `BipSampPluginParser.c`:

```
int cpiParseBufferFormatted(
  CciParser*     parser,
  CciContext*    context,
  int            encoding,
  int            ccsid,
  CciChar*       set,
  CciChar*       type,
  CciChar*       format
```

```
){
  PARSER_CONTEXT_ST* pc = (PARSER_CONTEXT_ST *)context ;
  int                rc;

  /* Get a pointer to the message buffer and set the offset */
  pc->iBuffer = (void *)cpiBufferPointer(&rc, parser);
  pc->iIndex = 0;

  /* Save the format of the buffer */
  pc->iEncoding = encoding;
  pc->iCcsid = ccsid;

  /* Save size of the buffer */
  pc->iSize = cpiBufferSize(&rc, parser);

  /* Prime the first byte in the stream */
  pc->iCurrentCharacter = cpiBufferByte(&rc, parser, pc->iIndex);

  /* Set the current element to the root element */
  pc->iCurrentElement = cpiRootElement(&rc, parser);

  /* Reset flag to ensure parsing is reset correctly */
  pc->iInTag = 0;

  if (pc->trace) {
    fprintf(pc->tracefile, "PLUGIN: <- cpiParseBufferFormatted()
    retvalue=%d\n", pc->iSize);
    fflush(pc->tracefile);
  }
```

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"cpiParseBuffer" on page 6580
This function prepares a parser to parse a new message object. It is called the first time (for each message) that the message flow causes the message content to be parsed.

"cpiParseBufferEncoded" on page 6582
This function extends the capability of the cpiParseBuffer() implementation function, and provides the encoding and coded character set that the input message is represented in.

"C parser implementation functions" on page 6538
A user-defined parser implements its capability through a function interface which is called by the broker during runtime operation. This interface includes functions to create and delete all local context storage that is associated with a parser object and the parsing operations.

cpiParseFirstChild:

This function parses the first child of a specified syntax element. It is called by the broker when the first child element of the current syntax element is required.

Defined In	Type	Member
CPI_VFT	Mandatory	iFpParseFirstChild

Syntax

```
void cpiParseFirstChild(
  CciParser*   parser,
  CciContext*  context,
  CciElement*  currentElement);
```

Parameters

parser
The address of the parser object (input).

context
The address of the context owned by the parser object (input).

currentElement
The address of the current syntax element (input).

Return values

None.

Sample

This example is taken from the sample parser file `BipSampPluginParser.c`:

```
void cpiParseFirstChild(
  CciParser*  parser,
  CciContext* context,
  CciElement* element
){
  PARSER_CONTEXT_ST* pc = (PARSER_CONTEXT_ST *)context ;
  int                rc;

  if ((!cpiElementCompleteNext(&rc, element)) &&
      (cpiElementType(&rc, element) == CCI_ELEMENT_TYPE_NAME)) {

    while ((!cpiElementCompleteNext(&rc, element))     &&
           (!cpiFirstChild(&rc, element)) &&
           (pc->iCurrentElement))
    {
      pc->iCurrentElement = parseNextItem(parser, context, pc->iCurrentElement);
    }
  }

  if (pc->trace) {
    fprintf(pc->tracefile, "PLUGIN: <- cpiParseFirstChild()\n");
    fflush(pc->tracefile);
  }

  return;
}
```

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"cpiParseLastChild"
This function parses the last child of a specified syntax element. It is called by the broker when the last child element of the current syntax element is required.

"C parser implementation functions" on page 6538
A user-defined parser implements its capability through a function interface which is called by the broker during runtime operation. This interface includes functions to create and delete all local context storage that is associated with a parser object and the parsing operations.

cpiParseLastChild:

This function parses the last child of a specified syntax element. It is called by the broker when the last child element of the current syntax element is required.

Defined In	Type	Member
CPI_VFT	Mandatory	iFpParseLastChild

Syntax

```
void cpiParseLastChild(
  CciParser*   parser,
  CciContext*  context,
  CciElement*  currentElement);
```

Parameters

parser
The address of the parser object (input).

context
The address of the context owned by the parser object (input).

currentElement
The address of the current syntax element (input).

Return values

None.

Sample

This example is taken from the sample parser file `BipSampPluginParser.c`:

```
void cpiParseLastChild(
  CciParser*  parser,
  CciContext* context,
  CciElement* element
){
  PARSER_CONTEXT_ST* pc = (PARSER_CONTEXT_ST *)context ;
  int                rc;

  if ((cpiElementType(&rc, element) == CCI_ELEMENT_TYPE_NAME)) {

    while ((!cpiElementCompleteNext(&rc, element))     &&
           (pc->iCurrentElement))
    {
      pc->iCurrentElement = parseNextItem(parser, context, pc->iCurrentElement);
    }
  }

  if (pc->trace) {
    fprintf(pc->tracefile, "PLUGIN: <- cpiParseLastChild()\n");
    fflush(pc->tracefile);
  }

  return;
}
```

The purpose of this code is to parse children of an element until the last child is reached. You can use this kind of structure in a parser that does not already know the exact offset in the bit stream of the last child of an element.

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"cpiParseFirstChild" on page 6586
This function parses the first child of a specified syntax element. It is called by the broker when the first child element of the current syntax element is required.

"C parser implementation functions" on page 6538
A user-defined parser implements its capability through a function interface which is called by the broker during runtime operation. This interface includes functions to create and delete all local context storage that is associated with a parser object and the parsing operations.

cpiParseNextSibling:

This function parses the next (right) sibling of a specified syntax element. It is called by the broker when the next (right) sibling element of the current syntax element is required.

Defined In	Type	Member
CPI_VFT	Mandatory	iFpParseNextSibling

Syntax

```
void cpiParseNextSibling(
  CciParser*   parser,
  CciContext*  context,
  CciElement*  currentElement);
```

Parameters

parser
The address of the parser object (input).

context
The address of the context owned by the parser object (input).

currentElement
The address of the current syntax element (input).

Return values

None.

Sample

This example is taken from the sample parser file `BipSampPluginParser.c`:

```
void cpiParseNextSibling(
  CciParser*  parser,
  CciContext* context,
  CciElement* element
){
  PARSER_CONTEXT_ST* pc = (PARSER_CONTEXT_ST *)context ;
  int                rc;

    while ((!cpiElementCompleteNext(&rc, cpiParent(&rc, element))) &&
         (!cpiNextSibling(&rc, element))       &&
         (pc->iCurrentElement))
  {
    pc->iCurrentElement = parseNextItem(parser, context, pc->iCurrentElement);
  }

  if (pc->trace) {
    fprintf(pc->tracefile, "PLUGIN: <- cpiParseNextSibling()\n");
    fflush(pc->tracefile);
  }

  return;
}
```

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"cpiParsePreviousSibling"
This function parses the previous (left) sibling of a specified syntax element. It is called by the broker when the previous (left) sibling element of the current syntax element is required.

"C parser implementation functions" on page 6538
A user-defined parser implements its capability through a function interface which is called by the broker during runtime operation. This interface includes functions to create and delete all local context storage that is associated with a parser object and the parsing operations.

cpiParsePreviousSibling:

This function parses the previous (left) sibling of a specified syntax element. It is called by the broker when the previous (left) sibling element of the current syntax element is required.

Defined In	Type	Member
CPI_VFT	Mandatory	iFpParsePreviousSibling

Syntax

```
void cpiParsePreviousSibling(
  CciParser*   parser,
  CciContext*  context,
  CciElement*  currentElement);
```

Parameters

parser
The address of the parser object (input).

context
The address of the context owned by the parser object (input).

currentElement
The address of the current syntax element (input).

Return values

None.

Sample

```
void cpiParsePreviousSibling(
  CciParser*  parser,
  CciContext* context,
  CciElement* element
){
  PARSER_CONTEXT_ST* pc = (PARSER_CONTEXT_ST *)context ;
  int                rc;

  while ((!cpiElementCompletePrevious(&rc, cpiParent(&rc, element))) &&
         (!cpiPreviousSibling(&rc, element))       &&
         (pc->iCurrentElement))
  {
    pc->iCurrentElement = parsePreviousItem(parser, context, pc->iCurrentElement);
```

```
  }

  if (pc->trace) {
    fprintf(pc->tracefile, "PLUGIN: <- cpiParsePreviousSibling()\n");
    fflush(pc->tracefile);
  }

  return;
}
```

The code sample is similar to that used for cpiParseNextSibling. Use cpiParsePreviousSibling as shown in the example when you are parsing the bit stream right to left.

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"cpiParseNextSibling" on page 6588
This function parses the next (right) sibling of a specified syntax element. It is called by the broker when the next (right) sibling element of the current syntax element is required.

"C parser implementation functions" on page 6538
A user-defined parser implements its capability through a function interface which is called by the broker during runtime operation. This interface includes functions to create and delete all local context storage that is associated with a parser object and the parsing operations.

cpiParserType:

This optional function returns whether the parser is an implementation of a *standard* parser.

A standard parser expects the Format field of the preceding header to contain the name of the parser class that follows. *Non-standard* parsers expects the Domain field to contain the parser class name. If the cpiParserType implementation function is not provided, the broker assumes that the parser is a standard parser.

Defined In	Type	Member
CPI_VFT	Optional	iFpParserType

Syntax

```
CciBool cpiParserType(
  CciParser*   parser,
  CciContext*  context);
```

Parameters

parser
The address of the parser object (input).

context
The address of the context owned by the parser object (input).

Return values

If the implementation is of a standard parser, zero is returned. Otherwise, the implementation is assumed to be that of a non-standard parser, nd a non-zero value is returned.

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"C parser implementation functions" on page 6538
A user-defined parser implements its capability through a function interface which is called by the broker during runtime operation. This interface includes functions to create and delete all local context storage that is associated with a parser object and the parsing operations.

cpiRootElement:

This function gets the address of the root syntax element of the specified parser object.

Syntax

```
CciElement* cpiRootElement(
  int*        returnCode,
  CciParser*  parser);
```

Parameters

returnCode
Receives the return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_PARSER_OBJECT

parser
Specifies the address of the parser object (input).

Return values

The address of the root syntax element is returned. If an error occurs, zero (CCI_NULL_ADDR) is returned, and *returnCode* indicates the reason for the error.

Sample

This example is taken from the sample parser file `BipSampPluginParser.c`:

```
int cpiParseBufferEncoded(
  CciParser*  parser,
  CciContext* context,
  int         encoding,
  int         ccsid
){
  PARSER_CONTEXT_ST* pc = (PARSER_CONTEXT_ST *)context ;
  int                rc;

  /* Get a pointer to the message buffer and set the offset */
  pc->iBuffer = (void *)cpiBufferPointer(&rc, parser);
  pc->iIndex = 0;

  /* Save the format of the buffer */
  pc->iEncoding = encoding;
  pc->iCcsid = ccsid;

  /* Save size of the buffer */
  pc->iSize = cpiBufferSize(&rc, parser);

  /* Prime the first byte in the stream */
  pc->iCurrentCharacter = cpiBufferByte(&rc, parser, pc->iIndex);

  /* Set the current element to the root element */
  pc->iCurrentElement = cpiRootElement(&rc, parser);

  /* Reset flag to ensure parsing is reset correctly */
  pc->iInTag = 0;

  /* We will assume ownership of the remainder of the buffer */
  return(pc->iSize);
}
```

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"C parser utility functions" on page 6539
A user-defined parser can call functions provided by the broker to create or define objects, such as message parser factories.

cpiSetCharacterValueFromBuffer:

This function sets the value of the specified syntax element.

Syntax

```
void cpiSetCharacterValueFromBuffer(
  int*            returnCode,
  CciElement*     targetElement,
  const CciChar*  value,
  CciSize         length);
```

Parameters

returnCode
Receives the return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT
- CCI_INV_DATA_POINTER
- CCI_INV_DATA_BUFLEN

targetElement
Specifies the address of the target syntax element object (input).

value
The value to be set in the target element (input).

length
The length of the character string, expressed as the number of `CciChar` characters, specified by the *value* parameter (input).

Return values

None. If an error occurs, *returnCode* indicates the reason for the error.

Sample

```
/* Convert the attribute value into broker form */
          data = CciNString((char *)startMarker, markedSize, pc->iCcsid);

/* Create a new name-value element for the attribute */
          newElement = cpiCreateElement(&rc, parser);
          cpiSetElementType(&rc, newElement, CCI_ELEMENT_TYPE_NAME_VALUE);
          cpiSetCharacterValueFromBuffer(&rc, newElement, data, length);
          if (pc->trace) {
            const char * mbData = mbString(data, pc->iCcsid);
            fprintf(pc->tracefile, "PLUGIN: Created new NAMEVALUE element;
                    object=0x%x type=0x%x name=",
                    newElement, CCI_ELEMENT_TYPE_NAME_VALUE);
            fprintf(pc->tracefile, "%s\n", mbData);
            fflush(pc->tracefile);
            free((void *)mbData);
          }
          /* Free the memory created in CciNString() */
          free((void *)data);

          /* Add the element */
          cpiAddAsLastChild(&rc, element, newElement);
```

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"C parser utility functions" on page 6539
A user-defined parser can call functions provided by the broker to create or define objects, such as message parser factories.

cpiSetElementCompleteNext:

This function sets the 'next child complete' flag in the target syntax element to the specified value.

Syntax

```
void cpiSetElementCompleteNext(
  int*         returnCode,
  CciElement*  targetElement,
  CciBool      value);
```

Parameters

returnCode
Receives the return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT

targetElement
Specifies the address of the target syntax element object (input).

value
The value to be set in the flag (input).

Return values

None. If an error occurs, *returnCode* indicates the reason for the error.

Sample

This example is taken from the sample parser file `BipSampPluginParser.c`:

```
/* Get a pointer to the start of the tag */
        startMarker = (char*)pc->iBuffer+(int)pc->iIndex;

        /* Skip over the tag */
        goToNameEnd( (PARSER_CONTEXT_ST *)context, parser );
```

```
/* Get a pointer to the end of the tag */
endMarker = (char*)pc->iBuffer+(int)pc->iIndex;

/* Compute the size of the tag */
markedSize = (size_t)endMarker-(int)startMarker;

/* Convert the tag into broker form */
data = CciNString((char *)startMarker, markedSize, pc->iCcsid);

/* Create a new name element for the tag */
newElement = cpiCreateElement(&rc, parser);
cpiSetElementType(&rc, newElement, CCI_ELEMENT_TYPE_NAME);
cpiSetElementName(&rc, newElement, data);
cpiSetElementCompletePrevious(&rc, newElement, 0);
cpiSetElementCompleteNext(&rc, newElement, 0);
if (pc->trace) {
  const char * mbData = mbString(data, pc->iCcsid);
  fprintf(pc->tracefile, "PLUGIN: New tag found\n");
  fprintf(pc->tracefile, "PLUGIN: Created new NAME element;
          object=0x%x type=0x%x name=",
          newElement, CCI_ELEMENT_TYPE_NAME);
  fprintf(pc->tracefile, "%s\n", mbData);
  fflush(pc->tracefile);
  free((void *)mbData);
}
/* Free the memory allocated in CciNString() */
free((void *)data);

/* Add the element */
cpiAddAsLastChild(&rc, element, newElement);
cpiSetElementCompletePrevious(&rc, element, 1);
```

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"cpiSetElementCompletePrevious"
This function sets the 'previous child complete' flag in the target syntax element to the specified value.

"C parser utility functions" on page 6539
A user-defined parser can call functions provided by the broker to create or define objects, such as message parser factories.

cpiSetElementCompletePrevious:

This function sets the 'previous child complete' flag in the target syntax element to the specified value.

Syntax

```
void cpiSetElementCompletePrevious(
  int*         returnCode,
  CciElement*  targetElement,
  CciBool      value);
```

Parameters

returnCode
Receives the return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT

targetElement
Specifies the address of the target syntax element object (input).

value
The value to be set in the flag (input).

Return values

None. If an error occurs, *returnCode* indicates the reason for the error.

Sample

This example is taken from the sample parser file BipSampPluginParser.c:

```
/* Get a pointer to the start of the tag */
        startMarker = (char*)pc->iBuffer+(int)pc->iIndex;

        /* Skip over the tag */
        goToNameEnd( (PARSER_CONTEXT_ST *)context, parser );

        /* Get a pointer to the end of the tag */
        endMarker = (char*)pc->iBuffer+(int)pc->iIndex;

        /* Compute the size of the tag */
        markedSize = (size_t)endMarker-(int)startMarker;

        /* Convert the tag into broker form */
        data = CciNString((char *)startMarker, markedSize, pc->iCcsid);

        /* Create a new name element for the tag */
        newElement = cpiCreateElement(&rc, parser);
        cpiSetElementType(&rc, newElement, CCI_ELEMENT_TYPE_NAME);
        cpiSetElementName(&rc, newElement, data);
        cpiSetElementCompletePrevious(&rc, newElement, 0);
        cpiSetElementCompleteNext(&rc, newElement, 0);
        if (pc->trace) {
          const char * mbData = mbString(data, pc->iCcsid);
          fprintf(pc->tracefile, "PLUGIN: New tag found\n");
          fprintf(pc->tracefile, "PLUGIN: Created new NAME element;
                  object=0x%x type=0x%x name=",
                  newElement, CCI_ELEMENT_TYPE_NAME);
          fprintf(pc->tracefile, "%s\n", mbData);
          fflush(pc->tracefile);
          free((void *)mbData);
        }
        /* Free the memory allocated in CciNString() */
        free((void *)data);
```

```
/* Add the element */
cpiAddAsLastChild(&rc, element, newElement);
cpiSetElementCompletePrevious(&rc, element, 1);
```

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"cpiSetElementCompleteNext" on page 6595
This function sets the 'next child complete' flag in the target syntax element to the specified value.

"C parser utility functions" on page 6539
A user-defined parser can call functions provided by the broker to create or define objects, such as message parser factories.

cpiSetElementName:

This function sets the name of the specified syntax element.

Syntax

```
void cpiSetElementName(
  int*            returnCode,
  CciElement*     targetElement,
  const CciChar*  name);
```

Parameters

returnCode
Receives the return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT
- CCI_INV_DATA_POINTER

targetElement
Specifies the address of the target syntax element object (input).

name
The name to be set in the target element (input).

Return values

None. If an error occurs, *returnCode* indicates the reason for the error.

Sample

This example is taken from the sample parser file `BipSampPluginParser.c`:

```
/* Convert the attribute value into broker form */
          data = CciNString((char *)startMarker, markedSize, pc->iCcsid);

/* Create a new name-value element for the attribute */
          newElement = cpiCreateElement(&rc, parser);
          cpiSetElementType(&rc, newElement, CCI_ELEMENT_TYPE_NAME_VALUE);
          cpiSetElementName(&rc, newElement, data);
          if (pc->trace) {
            const char * mbData = mbString(data, pc->iCcsid);
            fprintf(pc->tracefile, "PLUGIN: Created new NAMEVALUE element;
                    object=0x%x type=0x%x name=",
                    newElement, CCI_ELEMENT_TYPE_NAME_VALUE);
            fprintf(pc->tracefile, "%s\n", mbData);
            fflush(pc->tracefile);
            free((void *)mbData);
          }
          /* Free the memory created in CciNString() */
          free((void *)data);

          /* Add the element */
          cpiAddAsLastChild(&rc, element, newElement);
```

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"cpiSetElementType" on page 6601
This function sets the type of the specified syntax element.

"cpiSetElementValue group" on page 6604
This group of functions set the value of the specified syntax element.

"cpiSetElementValueValue" on page 6606
This function sets the value of the specified syntax element.

"C parser utility functions" on page 6539
A user-defined parser can call functions provided by the broker to create or define objects, such as message parser factories.

cpiSetElementNamespace:

Use this function to set the namespace attribute for the specified syntax element.

Defined In	Type	Member
CPI_VFT	Optional	iFpSetElementValue

Syntax

```
void             cpiSetElementNamespace(
  int*            returnCode,
  CciElement*     targetElement,
  const CciChar*  nameSpace);
```

Parameters

returnCode
A NULL pointer input value indicates that the user-defined node does not want to deal with errors. All exceptions thrown during the execution of this call are rethrown to the next upstream node in the flow. If the input value is not NULL, output signifies the success status of the call. If an exception occurs during execution, *returnCode is set to CCI_EXCEPTION on output. Call CciGetLastExceptionData to obtain details of the exception. (input).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT
- CCI_INV_DATA_POINTER

currentElement
The address of the current syntax element (input).

targetElement
Specifies the address of the target syntax element object.

value
Specifies the address of a null terminated string of `CciChars` that represents the namespace value. An empty string is a valid value for namespace; elements are created in the empty string namespace by default, therefore if you specify an empty string as the namespace by using this call, the call is effective only if the element was previously in another namespace, and the required effect is to change the namespace value to empty string.

Return values

None.

Sample

```
/* Convert the attribute value into broker form */
          data = CciNString((char *)startMarker, markedSize, pc->iCcsid);

/* Create a new name-value element for the attribute */
          newElement = cpiCreateElement(&rc, parser);
          cpiSetElementType(&rc, newElement, CCI_ELEMENT_TYPE_NAME_VALUE);
          cpiSetElementName(&rc, newElement, data);
          cpiSetElementNamespace(&rc, newElement, data);
          if (pc->trace) {
            const char * mbData = mbString(data, pc->iCcsid);
            fprintf(pc->tracefile, "PLUGIN: Created new NAMESPACEVALUE element;
                    object=0x%x type=0x%x name=",
                    newElement, CCI_ELEMENT_TYPE_NAME_VALUE);
            fprintf(pc->tracefile, "%s\n", mbData);
            fflush(pc->tracefile);
            free((void *)mbData);
          }
          /* Free the memory created in CciNString() */
          free((void *)data);
```

```
/* Add the element */
cpiAddAsLastChild(&rc, element, newElement);
```

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"cpiElementNameSpace" on page 6563
This function retrieves the value of the *namespace* attribute for the specified syntax element.

"C parser implementation functions" on page 6538
A user-defined parser implements its capability through a function interface which is called by the broker during runtime operation. This interface includes functions to create and delete all local context storage that is associated with a parser object and the parsing operations.

cpiSetElementType:

This function sets the type of the specified syntax element.

Syntax

```
void cpiSetElementType(
  int*            returnCode,
  CciElement*     targetElement,
  CciElementType  type);
```

Parameters

returnCode
Receives the return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT

targetElement
Specifies the address of the target syntax element object (input).

type
The type to be set in the target element (input).

Return values

None. If an error occurs, *returnCode* indicates the reason for the error.

Sample

This example is taken from the sample parser file `BipSampPluginParser.c`:

```
/* Convert the attribute value into broker form */
          data = CciNString((char *)startMarker, markedSize, pc->iCcsid);

/* Create a new name-value element for the attribute */
          newElement = cpiCreateElement(&rc, parser);
          cpiSetElementType(&rc, newElement, CCI_ELEMENT_TYPE_NAME_VALUE);
          cpiSetElementName(&rc, newElement, data);
          if (pc->trace) {
            const char * mbData = mbString(data, pc->iCcsid);
            fprintf(pc->tracefile, "PLUGIN: Created new NAMEVALUE element;
                    object=0x%x type=0x%x name=",
                    newElement, CCI_ELEMENT_TYPE_NAME_VALUE);
            fprintf(pc->tracefile, "%s\n", mbData);
            fflush(pc->tracefile);
            free((void *)mbData);
          }
          /* Free the memory created in CciNString() */
          free((void *)data);

          /* Add the element */
          cpiAddAsLastChild(&rc, element, newElement);
```

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"cpiSetElementName" on page 6598
This function sets the name of the specified syntax element.

"cpiSetElementValue group" on page 6604
This group of functions set the value of the specified syntax element.

"cpiSetElementValueValue" on page 6606
This function sets the value of the specified syntax element.

"C parser utility functions" on page 6539
A user-defined parser can call functions provided by the broker to create or define objects, such as message parser factories.

cpiSetElementValue:

This optional function sets the value of a specified element.

It called by the broker when the value of a syntax element is to be set. It provides an opportunity for a user-defined parser to override the behavior for setting element values.

Defined In	Type	Member
CPI_VFT	Optional	iFpSetElementValue

Syntax

```
void cpiSetElementValue(
  CciParser*        parser,
  CciElement*       currentElement,
  CciElementValue*  value);
```

Parameters

parser
The address of the parser object (input).

currentElement
The address of the current syntax element (input).

value
The value (input).

Return values

None.

Sample

This example is taken from the sample parser file `BipSampPluginParser.c`:

```
void cpiSetElementValue(
  CciParser*       parser,
  CciElement*      element,
  CciElementValue* value
){
  CciElement* newElement;
  int         rc;

  if ((cpiElementType(&rc, element) == CCI_ELEMENT_TYPE_VALUE) ||
      (cpiElementType(&rc, element) == CCI_ELEMENT_TYPE_NAME_VALUE))  {
    cpiSetElementValueValue(&rc, element, value);
  }
  else if (cpiElementType(&rc, element) == CCI_ELEMENT_TYPE_NAME) {
    /* Create a new value element, add as a first child, and set the value */
    newElement = cpiCreateElement(&rc, parser);
    cpiSetElementType(&rc, newElement, CCI_ELEMENT_TYPE_VALUE);
    cpiSetElementValueValue(&rc, newElement, value);
    cpiAddAsFirstChild(&rc, element, newElement);
  }
  else {
  }

  return;
}
```

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to

extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"cpiElementValue" on page 6566
This function gets the value of a specified element. It is called by the broker when the value of a syntax element is to be retrieved. It provides an opportunity for a user-defined parser to override the behavior for retrieving element values.

"C parser implementation functions" on page 6538
A user-defined parser implements its capability through a function interface which is called by the broker during runtime operation. This interface includes functions to create and delete all local context storage that is associated with a parser object and the parsing operations.

cpiSetElementValue group:

This group of functions set the value of the specified syntax element.

Specify the appropriate function from this group that matches the type of data to be set:

- cpiSetElementBitArrayValue
- cpiSetElementByteArrayValue
- cpiSetElementBooleanValue
- cpiSetElementCharacterValue
- cpiSetElementDateValue
- cpiSetElementDecimalValue
- cpiSetElementGmtTimestampValue
- cpiSetElementGmtTimeValue
- cpiSetElementIntegerValue
- cpiSetElementRealValue
- cpiSetElementTimestampValue
- cpiSetElementTimeValue

Syntax

```
void cpiSetElementBitArrayValue(
  int*                        returnCode,
  CciElement*                 targetElement,
  const struct CciBitArray*   value);
void cpiSetElementByteArrayValue(
  int*                        returnCode,
  CciElement*                 targetElement,
  const struct CciByteArray*  value);
void cpiSetElementBooleanValue(
  int*         returnCode,
  CciElement*  targetElement,
  CciBool      value);
void cpiSetElementCharacterValue(
  int*            returnCode,
  CciElement*     targetElement,
  const CciChar*  value,
  CciSize         length);
```

```
void cpiSetElementDateValue(
  int*                   returnCode,
  CciElement*            targetElement,
  const struct CciDate*  value);

void cpiSetElementDecimalValue(
  int*            returnCode,
  CciElement*     targetElement,
  const CciChar*  value);

void cpiSetElementGmtTimestampValue(
  int*                        returnCode,
  CciElement*                 targetElement,  const struct CciTimestamp*  value);

void cpiSetElementGmtTimeValue(
  int*                   returnCode,
  CciElement*            targetElement,
  const struct CciTime*  value);

void cpiSetElementIntegerValue(
  int*         returnCode,
  CciElement*  targetElement,
  CciInt       value);

void cpiSetElementRealValue(
  int*         returnCode,
  CciElement*  targetElement,
  CciReal      value);

void cpiSetElementTimestampValue(
  int*                        returnCode,
  CciElement*                 targetElement,
  const struct CciTimestamp*  value);

void cpiSetElementTimeValue(
  int*                   returnCode,
  CciElement*            targetElement,
  const struct CciTime*  value);
```

Parameters

returnCode
Receives the return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT
- CCI_INV_DATA_POINTER
- CCI_INV_DATA_BUFLEN

targetElement
Specifies the address of the target syntax element object (input).

value
The value to be set in the target element (input).

length
The length of the data value, expressed as the number of `CciChar` characters. This parameter is used on relevant function calls only.

Return values

None. If an error occurs, *returnCode* indicates the reason for the error.

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"cpiSetElementName" on page 6598
This function sets the name of the specified syntax element.

"cpiSetElementType" on page 6601
This function sets the type of the specified syntax element.

"cpiSetElementValueValue"
This function sets the value of the specified syntax element.

"C parser utility functions" on page 6539
A user-defined parser can call functions provided by the broker to create or define objects, such as message parser factories.

cpiSetElementValueValue:

This function sets the value of the specified syntax element.

Syntax

```
void cpiSetElementValueValue(
  int*              returnCode,
  CciElement*       targetElement,
  CciElementValue*  value);
```

Parameters

returnCode
Receives the return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT
- CCI_INV_DATA_POINTER

targetElement
Specifies the address of the target syntax element object (input).

value
Specifies the address of the *CciElementValue* object that contains the value to be stored in the specified target element (input).

Return values

None. If an error occurs, *returnCode* indicates the reason for the error.

Sample

This example is taken from the sample parser file `BipSampPluginParser.c`:

```
void cpiSetElementValue(
  CciParser*       parser,
  CciElement*      element,
  CciElementValue* value
){
  CciElement* newElement;
  int         rc;

  if ((cpiElementType(&rc, element) == CCI_ELEMENT_TYPE_VALUE) ||
      (cpiElementType(&rc, element) == CCI_ELEMENT_TYPE_NAME_VALUE))  {
    cpiSetElementValueValue(&rc, element, value);
  }
  else if (cpiElementType(&rc, element) == CCI_ELEMENT_TYPE_NAME) {
    /* Create a new value element, add as a first child, and set the value */
    newElement = cpiCreateElement(&rc, parser);
    cpiSetElementType(&rc, newElement, CCI_ELEMENT_TYPE_VALUE);
    cpiSetElementValueValue(&rc, newElement, value);
    cpiAddAsFirstChild(&rc, element, newElement);
  }
  else {
  }

  return;
}
```

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"cpiSetElementName" on page 6598
This function sets the name of the specified syntax element.

"cpiSetElementType" on page 6601
This function sets the type of the specified syntax element.

"cpiSetElementValue group" on page 6604
This group of functions set the value of the specified syntax element.

"C parser utility functions" on page 6539
A user-defined parser can call functions provided by the broker to create or define objects, such as message parser factories.

cpiSetNameFromBuffer:

This function sets the name attribute of the target syntax element using the data supplied in the buffer pointed to by the **name** parameter. The size of the name is specified using the **length** parameter.

Syntax

```
void cpiSetNameFromBuffer(
  int*            returnCode,
  CciElement*     targetElement,
  const CciChar*  name,
  CciSize         length);
```

Parameters

returnCode
Receives the return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_EXCEPTION
- CCI_INV_ELEMENT_OBJECT
- CCI_INV_DATA_POINTER
- CCI_INV_DATA_BUFLEN

targetElement
Specifies the address of the target syntax element object (input).

name
The address of a buffer containing the name (input).

length
The length of the character string, expressed as the number of `CciChar` characters, specified by the name parameter.

Return values

None. If an error occurs, *returnCode* indicates the reason for the error.

Sample

```
/* Convert the attribute value into broker form */
          data = CciNString((char *)startMarker, markedSize, pc->iCcsid);

/* Create a new name-value element for the attribute */
          newElement = cpiCreateElement(&rc, parser);
          cpiSetElementType(&rc, newElement, CCI_ELEMENT_TYPE_NAME_VALUE);
          cpiSetNameFromBuffer(&rc, newElement, data, length);
          if (pc->trace) {
            const char * mbData = mbString(data, pc->iCcsid);
            fprintf(pc->tracefile, "PLUGIN: Created new NAMEVALUE element;
                    object=0x%x type=0x%x name=",
                    newElement, CCI_ELEMENT_TYPE_NAME_VALUE);
            fprintf(pc->tracefile, "%s\n", mbData);
            fflush(pc->tracefile);
            free((void *)mbData);
          }
          /* Free the memory created in CciNString() */
          free((void *)data);

          /* Add the element */
          cpiAddAsLastChild(&rc, element, newElement);
```

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from

the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"C parser utility functions" on page 6539
A user-defined parser can call functions provided by the broker to create or define objects, such as message parser factories.

cpiSetNextParserClassName:

This optional function returns the name of the next parser class in the chain.

It is called during finalize processing, and returns to the caller a string that contains the name of the next parser class in the chain. Using this information, a user-defined parser can, during the finalize phase, modify the syntax element tree before the phase that causes serialization of the bit stream.

If you specify the name of a parser supplied with WebSphere Message Broker, you must use the correct class name of the parser.

Defined In	Type	Member
CPI_VFT	Optional	iFpSetNextParserClassName

Syntax

```
void cpiSetNextParserClassName(
  CciParser*   parser,
  CciContext*  context,
  CciChar*     name,
  CciBool      parserType);
```

Parameters

parser
The address of the parser object (input).

context
The address of the context owned by the parser object (input).

name
The name of the next parser as a string of **CciChar** characters.

parserType
Indicates whether the referenced parser is *standard* (`parserType=0`) or *non-standard* (`parserType=non-zero`) (input). A standard parser expects the Format field of the preceding header in the chain to contain the name of the parser class that follows. Non-standard parsers expects the Domain field to contain the parser class name.

Return values

None.

Sample

This example is taken from the sample parser file BipSampPluginParser.c:

```
void cpiSetNextParserClassName(
  CciParser*  parser,
  CciContext* context,
  CciChar*    name,
  CciBool     isHeaderParser
){
  PARSER_CONTEXT_ST* pc = (PARSER_CONTEXT_ST *)context ;
  int                rc = 0;

  /* Save the name in my context */
  CciCharNCpy(pc->iNextParserClassName, name, CciCharLen(name));

  if (pc->trace) {
    fprintf(pc->tracefile, "PLUGIN: <- cpiSetNextParserClassName()\n");
    fflush(pc->tracefile);
  }

  return;
}
```

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"C parser implementation functions" on page 6538
A user-defined parser implements its capability through a function interface which is called by the broker during runtime operation. This interface includes functions to create and delete all local context storage that is associated with a parser object and the parsing operations.

cpiWriteBuffer:

This function writes a syntax element tree to the message buffer associated with a parser.

It appends data to the bit stream in the message buffer associated with the parser object, using the current syntax element tree as a source. The element tree cannot be modified during the execution of this implementation function. The cpiAppendToBuffer utility function can be used to append the message buffer (bit stream) with data from the element tree.

If this implementation function is provided in the CPI_VFT structure, you cannot specify either cpiWriteBufferEncoded() or cpiWriteBufferFormatted(); if you do, the cpiDefineParserClass() function fails with a return code of CCI_INVALID_IMPL_FUNCTION.

Defined In	Type	Member
CPI_VFT	Conditional	iFpWriteBuffer

Syntax

```
int cpiWriteBuffer(
  CciParser*   parser,
  CciContext*  context);
```

Parameters

parser
The address of the parser object (input).

context
The address of the context owned by the parser object (input).

Return values

The size in bytes of the data appended to the bit stream in the buffer.

Sample

```
int cpiWriteBuffer(
  CciParser*  parser,
  CciContext* context
){
  PARSER_CONTEXT_ST* pc = (PARSER_CONTEXT_ST *)context ;
  int                initialSize = 0;
  int                rc = 0;
  const void* a;
  CciByte b;


  initialSize = cpiBufferSize(&rc, parser);
  a = cpiBufferPointer(&rc, parser);
  b = cpiBufferByte(&rc, parser, 0);

  cpiAppendToBuffer(&rc, parser, (char *)"Some test data", 14);

  return cpiBufferSize(0, parser) - initialSize;
}
```

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"cpiWriteBufferEncoded"
This function is an extension of the capability provided by the existing **cpiWriteBuffer()** implementation function that provides the encoding and coded character set that the output message should be represented in when the parser serializes its element tree to an output bit stream.

"cpiWriteBufferFormatted" on page 6613
This function extends the capability of the cpiWriteBuffer() implementation function by providing additional information about the output message.

"C parser implementation functions" on page 6538
A user-defined parser implements its capability through a function interface which is called by the broker during runtime operation. This interface includes functions to create and delete all local context storage that is associated with a parser object and the parsing operations.

cpiWriteBufferEncoded:

This function is an extension of the capability provided by the existing **cpiWriteBuffer()** implementation function that provides the encoding and coded character set that the output message should be represented in when the parser serializes its element tree to an output bit stream.

If serialization is not required, for example when the output is based on an input bit stream, and the tree has not been modified, this implementation function is not called by the broker. If this implementation function is provided in the CPI_VFT structure, you cannot specify either cpiWriteBuffer() or cpiWriteBufferFormatted(); if you do, the cpiDefineParserClass() function fails with a return code of CCI_INVALID_IMPL_FUNCTION.

Defined In	Type	Member
CPI_VFT	Conditional	iFpWriteBufferEncoded

Syntax

```
int cpiWriteBufferEncoded(
  CciParser*     parser,
  CciContext*    context,
  int            encoding,
  int            ccsid);
```

Parameters

parser
The address of the parser object (input).

context
The address of the context owned by the parser object (input).

encoding
The encoding of the message buffer (input).

ccsid
The CCSID of the message buffer (input).

Return values

The size in bytes of the data appended to the bit stream in the buffer.

Sample

This example is taken from the sample parser file `BipSampPluginParser.c`:

```
int cpiWriteBufferEncoded(
  CciParser*  parser,
  CciContext* context,
  int         encoding,
  int         ccsid
){
  PARSER_CONTEXT_ST* pc = (PARSER_CONTEXT_ST *)context ;
  int                initialSize = 0;
  int                rc = 0;
  const void* a;
  CciByte b;


  initialSize = cpiBufferSize(&rc, parser);
  a = cpiBufferPointer(&rc, parser);
  b = cpiBufferByte(&rc, parser, 0);

  cpiAppendToBuffer(&rc, parser, (char *)"Some test data", 14);

  return cpiBufferSize(0, parser) - initialSize;
}
```

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"cpiWriteBuffer" on page 6610
This function writes a syntax element tree to the message buffer associated with a parser.

"cpiWriteBufferFormatted"
This function extends the capability of the cpiWriteBuffer() implementation function by providing additional information about the output message.

"C parser implementation functions" on page 6538
A user-defined parser implements its capability through a function interface which is called by the broker during runtime operation. This interface includes functions to create and delete all local context storage that is associated with a parser object and the parsing operations.

cpiWriteBufferFormatted:

This function extends the capability of the cpiWriteBuffer() implementation function by providing additional information about the output message.

The following additional information is provided:

1. The encoding and coded character set that the output message is represented in when the parser serializes its element tree to an output bit stream.
2. The message set, type, and format for the output message for those parsers which require such information to correctly serialize its element tree to an output bit stream.

If serialization is not required, for example when the output is based on an input bit stream, and the tree has not been modified, this implementation function will not be invoked by the broker.

If this implementation function is provided in the CPI_VFT structure, you cannot specify either cpiWriteBuffer() or cpiWriteBufferEncoded(); if you do, the cpiDefineParserClass() function fails with a return code of CCI_INVALID_IMPL_FUNCTION.

Defined In	Type	Member
CPI_VFT	Conditional	iFpWriteBufferFormatted

Syntax

```
int cpiWriteBufferFormatted(
  CciParser*     parser,
  CciContext*    context,
  int            encoding,
  int            ccsid,
  CciChar*       set,
  CciChar*       type,
  CciChar*       format);
```

Parameters

parser
The address of the parser object (input).

context
The address of the context owned by the parser object (input).

encoding
The encoding of the message buffer (input).

ccsid
The CCSID of the message buffer (input).

set
The message set to which the message belongs (input).

type
The message type (input).

format
The message format (input).

Return values

The size in bytes of the data appended to the bit stream in the buffer.

Sample

```
int cpiWriteBufferFormatted(
  CciParser*  parser,
  CciContext* context,
  int         encoding,
```

```
  int         ccsid
  CciChar*       set,
  CciChar*       type,
  CciChar*       format
){
  PARSER_CONTEXT_ST* pc = (PARSER_CONTEXT_ST *)context ;
  int                initialSize = 0;
  int                rc = 0;
  const void* a;
  CciByte b;


  initialSize = cpiBufferSize(&rc, parser);
  a = cpiBufferPointer(&rc, parser);
  b = cpiBufferByte(&rc, parser, 0);

  cpiAppendToBuffer(&rc, parser, (char *)"Some test data", 14);

  return cpiBufferSize(0, parser) - initialSize;
}
```

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"cpiWriteBuffer" on page 6610
This function writes a syntax element tree to the message buffer associated with a parser.

"cpiWriteBufferEncoded" on page 6612
This function is an extension of the capability provided by the existing **cpiWriteBuffer()** implementation function that provides the encoding and coded character set that the output message should be represented in when the parser serializes its element tree to an output bit stream.

"C parser implementation functions" on page 6538
A user-defined parser implements its capability through a function interface which is called by the broker during runtime operation. This interface includes functions to create and delete all local context storage that is associated with a parser object and the parsing operations.

C user exit API

The user exit API defines implementation and utility functions.

- A set of implementation functions provides the functionality of the user exits. Some of these implementation functions are mandatory and others are optional.

  These functions are defined in the **BipCci.h** header file. They are described in "C user exit implementation functions" on page 6616.
- A set of utility functions that are invoked by user exits.

These functions are defined in the **BipCpi.h** header file. They are described in "C user exit utility functions" on page 6628.

Related concepts:

"User exits" on page 3015
A user exit is user-provided custom software, written in C, to track data passing through message flows.

"Developing user-defined extensions" on page 2970
A user-defined extension is a component that you design and implement to extend the function of WebSphere Message Broker.

Related tasks:

"Developing a user exit" on page 3114
Develop a user exit by declaring it, implementing its behavior, then compiling it.

"Deploying a user exit" on page 3116
Deploy your user exit to the broker.

Related reference:

"C user exit implementation functions"
A set of implementation functions provide the functionality of the user exits.

"C user exit utility functions" on page 6628
The broker provides several utility functions that you can call from your user exits.

"C Header files" on page 6415
The C interfaces are defined by the following header files.

"C common API" on page 6640
The C language common API consists of implementation and utility functions that you can use in user-defined nodes, parsers, and user exits.

"C language user-defined node API" on page 6416
Learn about the different types of call provided by the API.

"C language user-defined parser API" on page 6538
The C language user-defined parser API consists of two complementary sets of functions that you can use to define the function of your parser.

C user exit implementation functions:

A set of implementation functions provide the functionality of the user exits.

Some implementation functions are mandatory, and must be implemented by the developer; others are optional:

Mandatory functions

- "bipInitializeUserExits" on page 6617
- "bipTerminateUserExits" on page 6618

Optional functions

- "cciInputMessageCallback" on page 6619
- "cciPropagatedMessageCallback" on page 6623
- "cciOutputMessageCallback" on page 6626
- "cciNodeCompletionCallback" on page 6621
- "cciTransactionEventCallback" on page 6625

Related concepts:

"User exits" on page 3015
A user exit is user-provided custom software, written in C, to track data passing through message flows.

"Developing user-defined extensions" on page 2970
A user-defined extension is a component that you design and implement to extend the function of WebSphere Message Broker.

Related tasks:

"Developing a user exit" on page 3114
Develop a user exit by declaring it, implementing its behavior, then compiling it.

"Deploying a user exit" on page 3116
Deploy your user exit to the broker.

Related reference:

"C user exit utility functions" on page 6628
The broker provides several utility functions that you can call from your user exits.

"C user exit API" on page 6615
The user exit API defines implementation and utility functions.

"C common API" on page 6640
The C language common API consists of implementation and utility functions that you can use in user-defined nodes, parsers, and user exits.

"C language user-defined node API" on page 6416
Learn about the different types of call provided by the API.

"C language user-defined parser API" on page 6538
The C language user-defined parser API consists of two complementary sets of functions that you can use to define the function of your parser.

"C Header files" on page 6415
The C interfaces are defined by the following header files.

bipInitializeUserExits:

bipInitializeUserExits is an implementation function that is exported by the User Exit library (`.lel` file).

This function is called when the execution group starts, just after loading the `.lel` file. When this function runs, your exit code must call cciRegisterUserExit to register each user exit provided by that `.lel` file.

Syntax:

```
void bipInitializeUserExits()
```

Parameters:
None.

Return values:
None.

Example:

```
extern "C"{

void bipInitializeUserExits(){

  int rc = CCI_SUCCESS;
  CCI_UE_VFT myVft = {CCI_UE_VFT_DEFAULT};
  myVft.iFpInputMessageCallback      = myInputMessageCallback;
  myVft.iFpTransactionEventCallback  = myTransactionEventCallback;
  myVft.iFpPropagatedMessageCallback = myPropagatedMessageCallback;
  myVft.iFpNodeCompletionCallback    = myNodeCompletionCallback;
```

```
  cciRegisterUserExit(&rc,
                      MyConstants::myUserExitName,
                      0,
                      myVft);

  /*we should now check the rc for unexpected values*/

  return;
}

}/*end of extern "C" */
```

Related concepts:

"User exits" on page 3015
A user exit is user-provided custom software, written in C, to track data passing through message flows.

Related tasks:

"Developing a user exit" on page 3114
Develop a user exit by declaring it, implementing its behavior, then compiling it.

Related reference:

"cciRegisterUserExit" on page 6638
cciRegisterUserExit is a utility function that can be called by the user's code when bipInitializeUserExits is called.

bipTerminateUserExits:

bipInitializeUserExits is an implementation function exported by the User Exit library (`.lel` file).

This function is called just before unloading the `.lel` file, which typically happens when the execution group process is stopping. When this function runs, your exit code must clean up all resources that were allocated during the bipInitializeUserExits function.

If this function is not exported, the `.lel` file fails to load. You cannot call other utility functions while bipTerminateUserExits is running. This function is started on the same thread as the bipInitializeUserExits function.

Syntax:

```
void bipTerminateUserExits()
```

Parameters:
None.

Return values:
None.

Example:

```
extern "C"{

void bipTerminateUserExits(){
  /*Here, we clean up any resources, e.g.
         spawned threads, file handles, sockets  */
  freeResources();
}

}/*end of extern "C" */
```

Related concepts:

"User exits" on page 3015
A user exit is user-provided custom software, written in C, to track data passing through message flows.

Related tasks:

"Developing a user exit" on page 3114
Develop a user exit by declaring it, implementing its behavior, then compiling it.

Related reference:

"bipInitializeUserExits" on page 6617
bipInitializeUserExits is an implementation function that is exported by the User Exit library (`.lel` file).

cciInputMessageCallback:

The cciInputMessageCallback function can be registered as a callback and is called every time a message is read by an input node, and before that message is propagated down the message flow.

The cciInputMessageCallback function is called for every input message that is read in the execution group where the callback is registered, if the user exit state is active. The callback is registered by providing a pointer to the function as the **iFpInputMessageCallback** field of the CCI_UE_VFT struct that is passed to **cciRegisterUserExit**.

Syntax:

```
typedef void (*cciInputMessageCallback)     (
                                CciDataContext*  userContext,
                                CciMessage*      message,
                                CciMessage*      localEnvironment,
                                CciMessage*      exceptionList,
                                CciMessage*      environment,
                                CciMessageOrigin messageOrigin,
                                CciNode*         inputNode);
```

Parameters:

userContext (input)
The value that is passed to the **cciRegisterUserExit** function.

message
A handle to the message object. The user exit code must not update transport headers or Properties elements in this tree. Updating the message can affect performance, particularly if the input message would not otherwise be changed in the message flow.

localEnvironment
A handle to the local environment object.

exceptionList
A handle to the exception list object.

environment
A handle to the environment object for the current message flow.

messageOrigin
Depending on the type of input node, the message might have originated from a bit stream (CCI_MESSAGE_ORIGIN_BITSTREAM) or from a tree (CCI_MESSAGE_ORIGIN_TREE). The user exit can access one of these sources without causing processing by the parser. For example, in the case of the MQInput node, you can access the bitstream safely whereas, in the case of the

JMSInput node, you can access the tree safely. You can access the bit stream by calling **cniBufferPointer**, **cniBufferSize**, or **cniBufferByte**. You can access the tree by calling **cniRootElement** and using the usual syntax element navigation functions (for example, **cniFirstChild**). Although this parameter tells the user exit what it can access safely without causing processing by the parser, the user exit code might ignore this advice and effectively alter the parse timing.

inputNode
A handle to the input node that reads this input message. The handle can be used to make calls to functions such as **cciGetNodeName**, **cciGetNodeType**, and **cniGetBrokerInfo**.

Return values:
None.

Example:

```
void myInputMessageCallback(
                      CciDataContext*  userContext,
                      CciMessage*      message,
                      CciMessage*      localEnvironment,
                      CciMessage*      exceptionList,
                      CciMessage*      environment,
                      CciMessageOrigin messageOrigin,
                      CciNode*         inputNode){
...
...
}
```

Related concepts:

"User exits" on page 3015
A user exit is user-provided custom software, written in C, to track data passing through message flows.

Related tasks:

"Developing a user exit" on page 3114
Develop a user exit by declaring it, implementing its behavior, then compiling it.

Related reference:

"cciRegisterUserExit" on page 6638
cciRegisterUserExit is a utility function that can be called by the user's code when bipInitializeUserExits is called.

"cniBufferByte" on page 6426
Use this function to get a single byte from the data buffer associated with (and owned by) the message object specified in the message argument. The value of the index argument indicates which byte in the byte array is to be returned.

"cniBufferPointer" on page 6427
Use this function to get a pointer to the data buffer associated with (and owned by) the message object specified in the message argument. This function is typically used by output nodes.

"cniBufferSize" on page 6428
Use this function to get the size of the data buffer associated with (and owned by) the message object specified in the message argument.

"cniRootElement" on page 6503
Use this function to get the root syntax element associated with a specified message. It returns the root element that is associated with (and owned by) the message object identified by the *message* parameter. When a message object is constructed by the broker, a root element is automatically created.

"cniFirstChild" on page 6481
Use this function to retrieve the address of the syntax element object that is the first child of the specified syntax element.

"cniGetThreadContext" on page 6494
This function returns the thread context for the current thread.

"cciGetNodeName" on page 6630
This function returns the name of the specified node.

"cciGetNodeType" on page 6631
This function returns the type of the specified node.

"cniGetBrokerInfo" on page 6488
Use this function to query the current broker environment (for example, for information about broker name and message flow name). The information is returned in a structure of type CNI_BROKER_INFO_ST.

cciNodeCompletionCallback:

The cciNodeCompletionCallback function can be registered as a callback and is called whenever a node has completed processing of a message and is returning control to its upstream node.

The cciNodeCompletionCallback function is called for every message that is propagated in the execution group where the callback was registered, if the user exit state is active. The callback is registered by providing a pointer to the function as the **iFpNodeCompletionCallback** field of the CCI_UE_VFT struct that is passed to **cciRegisterUserExit**.

If the node completes due to an unhandled exception, it returns with a **reasonCode** of CCI_EXCEPTION, and that exception's details can be obtained by calling **cciGetLastExceptionData**.

If the node completes normally (including handling an exception on the catch or failure terminal), it returns with a **reasonCode** of CCI_SUCCESS. In this case, calling **cciGetLastExceptionData** returns unpredictable results.

Syntax:

```
typedef void (*cciNodeCompletionCallback)   (
                                CciDataContext* userContext,
                                CciMessage*     message,
                                CciMessage*     localEnvironment,
                                CciMessage*     exceptionList,
                                CciMessage*     environment,
                                CciConnection*  connection,
                                int             reasonCode);
```

Parameters:

userContext (input)
The value that is passed to the **cciRegisterUserExit** function.

message
A handle to the current message object. The current message is the message that is propagated to the node plus one or more of the following modifications:

- Modifications that are applied to the input root in the node
- Modifications that are made from a user exit during the preceding propagate callback

- Modifications that are made from a user exit during the downstream node complete callback if the node does not create a new message; for example, output nodes, Compute nodes with a `Compute Mode` that is set to `LocalEnvironment`, `Exception`, or `Exception And LocalEnvironment`, or request nodes with an `Output data location` that is set to `LocalEnvironment`.

The user exit code must not update transport headers or Properties elements in the message tree. Updating the message can affect performance. Changes that are made during this callback are visible only if the upstream node does not cause a new Message to be created.

`localEnvironment`
A handle to the local environment object that is being propagated.

`exceptionList`
A handle to the exception list object that is being propagated.

`environment`
A handle to the environment object for the current message flow.

`connection`
A handle to the connection object between the two nodes. The handle can be used, for example, in calls to **cciGetSourceNode**, **cciGetTargetNode**, **cciGetSourceTerminalName**, and **cciGetTargetTerminalName**. This handle is valid only for the duration of this instance of the user exit function.

`reasonCode`
A reason code that indicates whether the node completes normally (CCI_SUCCESS) or the node completes as the result of an unhandled exception (CCI_EXCEPTION). If the node completes due to an unhandled exception, you can obtain that exception's details by calling **cciGetLastExceptionData**. If the node completes normally (including handling an exception on the catch or failure terminal), the effect of calling **cciGetLastExceptionData** is undetermined.

Return values:
None.

Example:

```
void myNodeCompletionCallback(
                              CciDataContext* userContext,
                              CciMessage*     message,
                              CciMessage*     localEnvironment,
                              CciMessage*     exceptionList,
                              CciMessage*     environment,
                              CciConnection*  connection
                              int             reasonCode){
...
...
}
```

Related concepts:

"User exits" on page 3015
A user exit is user-provided custom software, written in C, to track data passing through message flows.

Related tasks:

"Developing a user exit" on page 3114
Develop a user exit by declaring it, implementing its behavior, then compiling it.

Related reference:

"cciRegisterUserExit" on page 6638
cciRegisterUserExit is a utility function that can be called by the user's code when bipInitializeUserExits is called.

"cciGetSourceNode" on page 6635
This function returns a handle to the upstream node of a given connection.

"cciGetTargetNode" on page 6637
This function returns a handle to the downstream node of a given connection.

"cciGetSourceTerminalName" on page 6636
This function returns the name of the output terminal of the source node for the specified connection.

"cciGetTargetTerminalName" on page 6637
This function returns the name of the input terminal of the target node for the specified connection.

"cciGetLastExceptionData" on page 6647
Gets diagnostic information about the last exception generated. Information about the last exception generated on the current thread is returned in a CCI_EXCEPTION_ST output structure. The user-defined extension can use this function to determine whether any recovery is required when a utility function returns an error code.

cciPropagatedMessageCallback:

The cciPropagatedMessageCallback function can be registered as a callback and is called whenever a message is propagated from one node to another.

The cciPropagatedMessageCallback function is called for every message that is propagated in the execution group where the callback was registered, if the user exit state is active. The callback is registered by providing a pointer to the function as the **iFpPropagatedMessageCallback** field of the CCI_UE_VFT struct that is passed to **cciRegisterUserExit**.

Syntax:

```
typedef void (*cciPropagatedMessageCallback)(
                                CciDataContext* userContext,
                                CciMessage*     message,
                                CciMessage*     localEnvironment,
                                CciMessage*     exceptionList,
                                CciMessage*     environment,
                                CciConnection*  connection);
```

Parameters:

userContext (input)
The value that is passed to the **cciRegisterUserExit** function.

message
A handle to the message object that is being propagated. The user exit code must not update transport headers or Properties elements in this tree. Updating the message can affect performance, particularly if the input message would not otherwise be changed in the message flow.

localEnvironment
A handle to the local environment object that is being propagated.

exceptionList
A handle to the exception list object that is being propagated.

environment
A handle to the environment object for the current message flow.

connection
A handle to the connection object between the two nodes. The handle can be used, for example, in calls to **cciGetSourceNode**, **cciGetTargetNode**, **cciGetSourceTerminalName**, and **cciGetTargetTerminalName**. This handle is valid only for the duration of this instance of the user exit function.

Return values:
None.

Example:

```
void myPropagatedMessageCallback(
                         CciMessage*    message,
                         CciMessage*    localEnvironment,
                         CciMessage*    exceptionList,
                         CciMessage*    environment,
                         CciConnection* connection){


   int rc = CCI_SUCCESS;
   CciNode* targetNode = cciGetTargetNode(amp rc,
                                          connection);



   CciChar targetNodeName [initialStringBufferLength];
   targetNodeNameLength = cciGetNodeName(amp rc,
                                         targetNode,
                                         targetNodeName,
                                         initialStringBufferLength);
   /*you should now check the rc for unexpected values*/
   /*if rc is CCI_BUFFER_TOO_SMALL, resize and retry*/

}
```

Related concepts:

"User exits" on page 3015
A user exit is user-provided custom software, written in C, to track data passing through message flows.

Related tasks:

"Developing a user exit" on page 3114
Develop a user exit by declaring it, implementing its behavior, then compiling it.

Related reference:

"cciRegisterUserExit" on page 6638
cciRegisterUserExit is a utility function that can be called by the user's code when bipInitializeUserExits is called.

"cciGetSourceNode" on page 6635
This function returns a handle to the upstream node of a given connection.

"cciGetTargetNode" on page 6637
This function returns a handle to the downstream node of a given connection.

"cciGetSourceTerminalName" on page 6636
This function returns the name of the output terminal of the source node for the specified connection.

"cciGetTargetTerminalName" on page 6637
This function returns the name of the input terminal of the target node for the specified connection.

cciTransactionEventCallback:

The cciTransactionEventCallback function can be registered as a callback and is called every time a message flow transaction ends.

This function is called for every message flow transaction within the execution group where the callback was registered, if the user exit state is active. The callback is registered by providing a pointer to the function in the **iFpTransactionEventCallback** field of the CCI_UE_VFT struct passed to **cciRegisterUserExit**.

Syntax:

```
typedef void (*cciTransactionEventCallback) (
                                  CciDataContext*         userContext,
                                  CciTransactionEventType type,
                                  CciMessage*             environment,
                                  CciNode*                inputNode);
```

Parameters:

userContext (input)
This is the value that was passed to the **cciRegisterUserExit** function.

type
This describes the event that occurred. Possible values are:

- CCI_TRANSACTION_EVENT_COMMIT
  A transaction has been successfully committed.
- CCI_TRANSACTION_EVENT_ROLLBACK
  A transaction has been rolled back.

If the transaction was rolled back due to an unhandled exception, you can obtain details of that exception by calling **cciGetLastExceptionData**.

environment
This is a handle to the environment object for the current message flow. Although the user exit can update this tree, it is cleared after returning from this function, so any updates are lost.

inputNode
This is a handle to the input node which reads the input message that triggered the transaction. It can be used to make calls to functions such as **cciGetNodeName**, **cciGetNodeType**, and **cniGetBrokerInfo**.

Return values:
None

Example:

```
void myTransactionEventCallback(
                     CciDataContext*         userContext,
                     CciTransactionEventType type,
                     CciMessage*             environment,
                     CciNode*                inputNode){
...
...
}
```

Related concepts:

"User exits" on page 3015
A user exit is user-provided custom software, written in C, to track data passing through message flows.

Related tasks:

"Developing a user exit" on page 3114
Develop a user exit by declaring it, implementing its behavior, then compiling it.

Related reference:

"cciRegisterUserExit" on page 6638
cciRegisterUserExit is a utility function that can be called by the user's code when bipInitializeUserExits is called.

"cciGetLastExceptionData" on page 6647
Gets diagnostic information about the last exception generated. Information about the last exception generated on the current thread is returned in a CCI_EXCEPTION_ST output structure. The user-defined extension can use this function to determine whether any recovery is required when a utility function returns an error code.

"cciGetNodeName" on page 6630
This function returns the name of the specified node.

"cciGetNodeType" on page 6631
This function returns the type of the specified node.

"cniGetBrokerInfo" on page 6488
Use this function to query the current broker environment (for example, for information about broker name and message flow name). The information is returned in a structure of type CNI_BROKER_INFO_ST.

cciOutputMessageCallback:

The cciOutputMessageCallback function can be registered as a callback and is called whenever a message is sent by an output node.

The cciOutputMessageCallback function is called for every output message that is sent successfully in the execution group or message flow where the callback was registered if the user exit state is active. If the node provides WrittenDestination information in the LocalEnvironment tree, the callback is called after this information is created.

Calls are made after the following operations are completed:
- Sending a message from an output or reply node (for WebSphere MQ, JMS, TCPIP, HTTP, or SOAP nodes).
- Sending a message from a request node (TCPIP, HTTP, or SOAP nodes). The callback is made after the reply has been received.
- Writing to a file (FileOutput node).
- Sending an email (EmailOutput node).
- Completing an adapter request (WebSphere Adapters request nodes).

The callback is registered by providing a pointer to the function as the iFpOutputMessageCallback field of the CCI_UE_VFT struct that is passed to cciRegisterUserExit. The iFpOutputMessageCallback field was added in CCI_UE_VFT struct version 2.

When you implement this callback, check the node type by using cciGetNodeType before you perform any node-specific operations.

Syntax:

```
typedef void (*cciOutputMessageCallback)     (
                                CciDataContext*  userContext,
                                CciMessage*      message,
                                CciMessage*      localEnvironment,
```

```
                                        CciMessage*      exceptionList,
                                        CciMessage*      environment,
                                        CciNode*         node);
```

Parameters:

userContext (input)
The value that is passed to the cciRegisterUserExit function.

message (input)
A handle to the message object. You must not update the transport headers or Properties elements in this tree.

You can update the message body. However, because this callback is called after the message has been sent to the transport, any changes do not appear in that message. Changes are visible only to nodes that are connected downstream of the output node. Updating the message can affect performance, particularly if the message tree would not otherwise be changed in the message flow.

localEnvironment (input)
A handle to the local environment object. The handle can contain information about the destination of the message that is written in the WrittenDestination subtree. See each node's documentation for more details.

exceptionList (input)
A handle to the exception list object.

environment (input)
A handle to the environment object for the current message flow.

node (input)
A handle to the node that has sent the output message. You can use the handle to make calls to functions such as cciGetNodeName, cciGetNodeType, and cniGetBrokerInfo.

Return values:
None.

Example:

```
void myOutputMessageCallback(
                      CciDataContext*  userContext,
                      CciMessage*      message,
                      CciMessage*      localEnvironment,
                      CciMessage*      exceptionList,
                      CciMessage*      environment,
                      CciNode*         node){
}
```

Related concepts:

"User exits" on page 3015
A user exit is user-provided custom software, written in C, to track data passing through message flows.

"Message flow nodes" on page 1024
A message flow node is a processing step in a message flow. It can be a built-in node, a user-defined node, or a subflow node.

Related tasks:

"Developing a user exit" on page 3114
Develop a user exit by declaring it, implementing its behavior, then compiling it.

Related reference:

"cciRegisterUserExit" on page 6638
cciRegisterUserExit is a utility function that can be called by the user's code when bipInitializeUserExits is called.

"cciGetNodeName" on page 6630
This function returns the name of the specified node.

"cciGetNodeType" on page 6631
This function returns the type of the specified node.

"cciGetBrokerInfo" on page 6645
Use the cciGetBrokerInfo function to query the current broker environment (for example, for information about broker name, execution group name, queue manager name). The information is returned in a structure of type CCI_BROKER_INFO_ST.

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

C user exit utility functions:

The broker provides several utility functions that you can call from your user exits.

- "cciGetNodeAttribute" on page 6629
- "cciGetNodeName" on page 6630
- "cciGetNodeType" on page 6631
- "cciGetSourceNode" on page 6635
- "cciGetSourceTerminalName" on page 6636
- "cciGetTargetNode" on page 6637
- "cciGetTargetTerminalName" on page 6637
- "cciRegisterUserExit" on page 6638

Related concepts:

"User exits" on page 3015
A user exit is user-provided custom software, written in C, to track data passing through message flows.

"Developing user-defined extensions" on page 2970
A user-defined extension is a component that you design and implement to extend the function of WebSphere Message Broker.

Related tasks:

"Developing a user exit" on page 3114
Develop a user exit by declaring it, implementing its behavior, then compiling it.

"Deploying a user exit" on page 3116
Deploy your user exit to the broker.

Related reference:

"C user exit implementation functions" on page 6616
A set of implementation functions provide the functionality of the user exits.

"C user exit API" on page 6615
The user exit API defines implementation and utility functions.

"C common API" on page 6640
The C language common API consists of implementation and utility functions that you can use in user-defined nodes, parsers, and user exits.

"C language user-defined node API" on page 6416
Learn about the different types of call provided by the API.

"C language user-defined parser API" on page 6538
The C language user-defined parser API consists of two complementary sets of functions that you can use to define the function of your parser.

"C Header files" on page 6415
The C interfaces are defined by the following header files.

cciGetNodeAttribute:

The cciGetNodeAttribute function returns the value of the specified attribute.

Syntax:

```
CciSize  cciGetNodeAttribute (int*           returnCode,
                              CciNode*       node,
                              CciChar*       name,
                              CciChar*       value,
                              CciSize        length);
```

Parameters:

returnCode (output)
Receives the return code from the function (output).

- CCI_INV_BUFFER_TOO_SMALL
  The provided buffer was not large enough to hold the value of node's type.

node (input)
This is a handle to a node.

name (input)
This is a pointer to a NULL-terminated string of CciChar specifying the name of the node attribute being queried.

value (output)
Address of a buffer, allocated by the caller to hold the value of the attribute.

length
The length, in CciChars, of the buffer allocated by the caller.

Return values:

- If successful, the attribute value is copied into the supplied buffer and the number of CciChar characters copied is returned.
- If the buffer is not large enough to contain the attribute value, **returnCode** is set to CCI_BUFFER_TOO_SMALL, and the number of CciChars required is returned.
- If **name** specifies an attribute name that is not appropriate for the given node, **returnCode** is set to CCI_ATTRIBUTE_UNKOWN.

Example:

```
void myPropagatedMessageCallback(
                        CciMessage*    message,
                        CciMessage*    localEnvironment,
                        CciMessage*    exceptionList,
                        CciMessage*    environment,
                        CciConnection* connection){
   int rc = CCI_SUCCESS;
   CciNode* sourceNode = cciGetSourceNode(&rc,
                                          connection);
   /*you should now check the rc for unexpected values*/
   CciChar   queueNameAttribute[16];
   cciMbsToUcs(&rc,
               "queueName",
```

```
              queueNameAttribute,
              16,
              BIP_DEF_COMP_CCSID);
/*you should now check the rc for unexpected values*/

  CciChar queueName [512];
  sourceNodeQueueNameLength = cciGetNodeType(&rc,
                                     sourceNode,
                                     queueName,
                                     512);
  /*you should now check the rc for unexpected values*/
  /*if rc is CCI_BUFFER_TOO_SMALL, resize and retry*/
/*sourceNodeQueueNameLength will hold the actual or required size */
```

Related concepts:

"User exits" on page 3015
A user exit is user-provided custom software, written in C, to track data passing through message flows.

Related tasks:

"Developing a user exit" on page 3114
Develop a user exit by declaring it, implementing its behavior, then compiling it.

cciGetNodeName:

This function returns the name of the specified node.

The name is assigned internally by the WebSphere Message Broker Toolkit and is unique in the message flow. The label that is assigned to a node by the message flow designer in the Message Flow Editor can be obtained by calling "cciGetNodeAttribute" on page 6629 to read the label attribute.

Syntax:

```
CciSize  getNodeName (int*                  returnCode,
                      CciNode*              node,
                      CciChar*              value,
                      CciSize               length);
```

Parameters:

returnCode (output)
Receives the return code from the function (output)

- CCI_INV_BUFFER_TOO_SMALL

  The provided buffer was not large enough to hold the value of node's name.

node (input)
This is a handle to a node.

value (output)
Address of a buffer, allocated by the caller to hold the value of the node's name.

length
The length, in CciChars, of the buffer allocated by the caller.

Return values:

- If successful, the node name is copied into the supplied buffer and the number of CciChar characters copied is returned.
- If the buffer is not large enough to contain the node name, **returnCode** is set to CCI_BUFFER_TOO_SMALL, and the number of CciChars required is returned.

Example:

```
void myPropagatedMessageCallback(
                        CciMessage*    message,
                        CciMessage*    localEnvironment,
                        CciMessage*    exceptionList,
                        CciMessage*    environment,
                        CciConnection* connection){

  int rc = CCI_SUCCESS;
   CciNode* targetNode = cciGetTargetNode(&rc,
                                          connection);



   CciChar targetNodeName [initialStringBufferLength];
   targetNodeNameLength = cciGetNodeName(&rc,
                                        targetNode,
                                        targetNodeName,
                                        initialStringBufferLength);
   /*you should now check the rc for unexpected values*/
   /*if rc is CCI_BUFFER_TOO_SMALL, resize and retry*/

}
```

Related concepts:

"User exits" on page 3015
A user exit is user-provided custom software, written in C, to track data passing through message flows.

Related tasks:

"Developing a user exit" on page 3114
Develop a user exit by declaring it, implementing its behavior, then compiling it.

cciGetNodeType:

This function returns the type of the specified node.

Syntax:

```
CciSize  cciGetNodeType (int*                    returnCode,
                         CciNode*          node,
                         CciChar*          value,
                         CciSize           length);
```

Parameters:

returnCode (output)
Receives the return code from the function (output).

- CCI_INV_BUFFER_TOO_SMALL

  The provided buffer was not large enough to hold the value of node's type.

node (input)
This is a handle to a node.

value (output)
Address of a buffer, allocated by the caller to hold the value of the node type.

length
The length, in CciChars, of the buffer allocated by the caller.

Return values:

- If the function is successful, the node type is copied into the supplied buffer and the number of CciChar characters copied is returned.

- If the buffer is not large enough to contain the node type, **returnCode** is set to CCI_BUFFER_TOO_SMALL, and the number of CciChars required is returned.

Example:

```
void myPropagatedMessageCallback(
                         CciMessage*    message,
                         CciMessage*    localEnvironment,
                         CciMessage*    exceptionList,
                         CciMessage*    environment,
                         CciConnection* connection){

  int rc = CCI_SUCCESS;
   CciNode* sourceNode = cciGetSourceNode(&rc,
                                          connection);
   /*you should now check the rc for unexpected values*/

   CciChar sourceNodeType[initialStringBufferLength];
   sourceNodeTypeLength = cciGetNodeType(&rc,
                                         sourceNode,
                                         sourceNodeType,
                                         initialStringBufferLength);
   /*you should now check the rc for unexpected values*/
   /*if rc is CCI_BUFFER_TOO_SMALL then you should resize and retry*/
```

Related concepts:

"User exits" on page 3015
A user exit is user-provided custom software, written in C, to track data passing through message flows.

Related tasks:

"Developing a user exit" on page 3114
Develop a user exit by declaring it, implementing its behavior, then compiling it.

Related reference:

"Node types"
Use the cciGetNodeType function to find out the node type of any node.

Node types:

Use the cciGetNodeType function to find out the node type of any node.

Node	Node Type
WebSphere MQ	
MQInput	ComIbmMQInputNode
MQOutput	ComIbmMQOutputNode
MQReply	ComIbmMQOutputNode
MQGet	ComIbmMQGetNode
JMS	
JMSHeader	ComIbmJMSHeader.msgnode
JMSInput	ComIbmJMSClientInputNode
JMSOutput	ComIbmJMSClientOutputNode
JMSReply	ComIbmJMSClientReplyNode
HTTP	
HTTPHeader	ComIbmHTTPHeader
HTTPInput	ComIbmWSInputNode
HTTPReply	ComIbmWSReplyNode

Node	Node Type
HTTPRequest	ComIbmWSRequestNode
Web Services	
SOAPInput	ComIbmSOAPInputNode
SOAPReply	ComIbmSOAPReplyNode
SOAPRequest	ComIbmSOAPRequestNode
SOAPAsyncRequest	ComIbmSOAPAsyncRequestNode
SOAPAsyncResponse	ComIbmSOAPAsyncResponseNode
SOAPEnvelope	ComIbmSOAPWrapperNode
SOAPExtract	ComIbmSOAPExtractNode
RegistryLookup	SRRetrieveEntityNode
EndpointLookup	SRRetrieveITServiceNode
WebSphere Adapters	
JDEdwardsInput	ComIbmJDEdwardsInputNode
JDEdwardsRequest	ComIbmJDEdwardsRequestNode
PeopleSoftInput	ComIbmPeopleSoftInputNode
PeopleSoftRequest	ComIbmPeopleSoftRequestNode
SAPInput	ComIbmSAPInputNode
SAPReply	ComIbmSAPReplyNode
SAPRequest	ComIbmSAPRequestNode
SiebelInput	ComIbmSiebelInputNode
SiebelRequest	ComIbmSiebelRequestNode
TwineballInput	ComIbmTwineBallInputnode
TwineballRequest	ComIbmTwineBallRequestNode
Routing	
Filter	ComIbmFilterNode
Label	ComIbmLabelNode
Publication	The subflow is composed of:
RouteToLabel	ComIbmRouteToLabelNode
Route	ComIbmRouteNode
AggregateControl	ComIbmAggregateControlNode
AggregateReply	ComIbmAggregateReplyNode
AggregateRequest	ComIbmAggregateRequestNode
Collector	ComIbmCollectorNode
Resequence	ComIbmReSequenceNode
Sequence	ComIbmSequenceNode
Transformation	
Mapping	ComIbmComputeNode
XSLTransform	ComIbmXslMqsiNode
Compute	ComIbmComputeNode
JavaCompute	ComIbmJavaComputeNode
PHPCompute	ComIbmPhpCompute

Node	Node Type
Construction	
Throw	ComIbmThrowNode
Trace	ComIbmTraceNode
TryCatch	ComIbmTryCatchNode
FlowOrder	ComIbmFlowOrderNode
Passthrough	ComIbmPassthruNode
ResetContentDescriptor	ComIbmResetContentDescriptorNode
Database	
DatabaseInput	ComIbmDatabaseInputNode
Database	ComIbmDatabaseNode
DataDelete	ComIbmDatabaseNode
DataInsert	ComIbmDatabaseNode
DataUpdate	ComIbmDatabaseNode
Warehouse	ComIbmDatabaseNode
DatabaseRetrieve	ComIbmDatabaseRetrieveNode
DatabaseRoute	ComIbmDatabaseRouteNode
File	
FileInput	ComIbmFileInputNode
FileRead	ComIbmFileReadNode
FileOutput	ComIbmFileOutputNode
FTEInput	ComIbmFTEInputNode
FTEOutput	ComIbmFTEOutputNode
Email	
EmailInput	ComIbmEmailInputNode
EmailOutput	ComIbmEmailOutputNode
TCPIP	
TCPIPClientInput	ComIbmTCPIPClientInputNode
TCPIPClientOutput	ComIbmTCPIPClientOutputNode
TCPIPClientReceive	ComIbmTCPIPClientRequestNode
TCPIPServerInput	ComIbmTCPIPServerInputNode
TCPIPServerOutput	ComIbmTCPIPServerOutputNode
TCPIPServerReceive	ComIbmTCPIPServerRequestNode
CORBA	
CORBARequest	ComIbmCORBARequestNode
CICS	
CICSRequest	ComIbmCICSIPICRequestNode
IMS	
IMSRequest	ComIbmIMSRequestNode
Validation	
Validate	ComIbmValidateNode
Security	

Node	Node Type
SecurityPEP	ComIbmSecurityPEP
Timer	
TimeoutControl	ComIbmTimeoutControlNode
TimeoutNotification	ComIbmTimeoutNotificationNode

Related concepts:

"Message flow nodes" on page 1024
A message flow node is a processing step in a message flow. It can be a built-in node, a user-defined node, or a subflow node.

Related reference:

"cciGetNodeType" on page 6631
This function returns the type of the specified node.

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

cciGetSourceNode:

This function returns a handle to the upstream node of a given connection.

Syntax:

```
CciNode*  cciGetSourceNode(int*                    returnCode,
                           CciConnection * connection);
```

Parameters:

returnCode (output)
Receives the return code from the function.

connection
This parameter is a handle to a connection on the output terminal of the requested node.

Return values:
A handle to the node that is on the source side of the connection.

Example:

```
void myPropagatedMessageCallback(
                        CciMessage*    message,
                        CciMessage*    localEnvironment,
                        CciMessage*    exceptionList,
                        CciMessage*    environment,
                        CciConnection* connection){
    ...
    ...

   int rc = CCI_SUCCESS;
   CciNode* sourceNode = cciGetSourceNode(&rc,
                                          connection);
   /*you should now check the rc for unexpected values*/
```

Related concepts:

"User exits" on page 3015
A user exit is user-provided custom software, written in C, to track data passing through message flows.

Related tasks:

"Developing a user exit" on page 3114
Develop a user exit by declaring it, implementing its behavior, then compiling it.

cciGetSourceTerminalName:

This function returns the name of the output terminal of the source node for the specified connection.

Syntax:

```
CciSize  cciGetSourceTerminalName (int*           returnCode,
                                   CciConnection* connection,
                                   CciChar*       value,
                                   CciSize        length);
```

Parameters:

returnCode (output)
Receives the return code from the function (output).

- CCI_BUFFER_TOO_SMALL

  The provided buffer was not large enough to hold the value of node's name.

connection (input)
This parameter is a handle to a connection between two nodes.

value (output)
Address of a buffer, allocated by the caller to hold the value of the terminal's name.

length
The length, in `CciChars`, of the buffer allocated by the caller.

Return values:

- If successful, the terminal name is copied into the supplied buffer, and the number of `CciChar` characters copied is returned.
- If the buffer is not large enough to contain the attribute value, **returnCode** is set to CCI_BUFFER_TOO_SMALL, and the number of `CciChars` required is returned.

Example:

```
void myPropagatedMessageCallback(
                        CciDataContext* userContext,
                        CciMessage*     message,
                        CciMessage*     localEnvironment,
                        CciMessage*     exceptionList,
                        CciMessage*     environment,
                        CciConnection*  connection){
  int rc = CCI_SUCCESS;
   CciChar sourceTerminalName[initialStringBufferLength];
   cciGetSourceTerminalName(&rc,
                            connection,
                            sourceTerminalName,
                            initialStringBufferLength);
}
```

Related concepts:

"User exits" on page 3015
A user exit is user-provided custom software, written in C, to track data passing through message flows.

Related tasks:

"Developing a user exit" on page 3114
Develop a user exit by declaring it, implementing its behavior, then compiling it.

cciGetTargetNode:

This function returns a handle to the downstream node of a given connection.

Syntax:

```
CciNode*  cciGetTargetNode(int*                    returnCode,
                                                CciConnection * connection);
```

Parameters:

returnCode (output)
Receives the return code from the function (output).

connection
This parameter is a handle to a connection on an input terminal of the requested node.

Return values:
A handle to the node that is on the target side of the connection.

Example:

```
void myPropagatedMessageCallback(
                        CciMessage*    message,
                        CciMessage*    localEnvironment,
                        CciMessage*    exceptionList,
                        CciMessage*    environment,
                        CciConnection* connection){
   ...
   ...

   CciNode* targetNode = cciGetTargetNode(&rc,
                                          connection);
```

Related concepts:

"User exits" on page 3015
A user exit is user-provided custom software, written in C, to track data passing through message flows.

Related tasks:

"Developing a user exit" on page 3114
Develop a user exit by declaring it, implementing its behavior, then compiling it.

cciGetTargetTerminalName:

This function returns the name of the input terminal of the target node for the specified connection.

Syntax:

```
CciSize  cciGetTargetTerminalName (int*           returnCode,
                                   CciConnection* connection,
                                   CciChar*       value,
                                   CciSize        length);
```

Parameters:

returnCode (output)
Receives the return code from the function (output).

- CCI_BUFFER_TOO_SMALL

The provided buffer was not large enough to hold the value of node name.

connection (input)
This parameter is a handle to a connection between two nodes.

value (output)
Address of a buffer, allocated by the caller to hold the value of the terminal's name.

length
The length, in CciChars, of the buffer allocated by the caller.

Return values:

- If successful, the terminal name is copied into the supplied buffer, and the number of CciChar characters copied is returned.
- If the buffer is not large enough to contain the terminal name, **returnCode** is set to CCI_BUFFER_TOO_SMALL, and the number of CciChars required is returned.

Example:

```
void myPropagatedMessageCallback(
                        CciDataContext* userContext,
                        CciMessage*     message,
                        CciMessage*     localEnvironment,
                        CciMessage*     exceptionList,
                        CciMessage*     environment,
                        CciConnection*  connection){
   int rc = CCI_SUCCESS;
   CciChar targetTerminalName[initialStringBufferLength];
   cciGetTargetTerminalName(&rc,
                            connection,
                            targetTerminalName,
                            initialStringBufferLength);
   /*you should now check the rc for unexpected values*/
   /*if rc is CCI_BUFFER_TOO_SMALL then you should resize and retry*/
```

Related concepts:

"User exits" on page 3015
A user exit is user-provided custom software, written in C, to track data passing through message flows.

Related tasks:

"Developing a user exit" on page 3114
Develop a user exit by declaring it, implementing its behavior, then compiling it.

cciRegisterUserExit:

cciRegisterUserExit is a utility function that can be called by the user's code when bipInitializeUserExits is called.

This function is called by the user's code if the user wants to register functions to be called every time certain events occur.

Syntax:

```
typedef struct cci_UEVft {
    int     reserved;
    char    StrucId[4];
    int     Version;
    cciInputMessageCallback      iFpInputMessageCallback;
    cciTransactionEventCallback  iFpTransactionEventCallback;
    cciPropagatedMessageCallback iFpPropagatedMessageCallback;
```

```
    cciNodeCompletionCallback    iFpNodeCompletionCallback;
    cciOutputMessageCallback     iFpOutputMessageCallback;

} CCI_UE_VFT;

void cciRegisterUserExit (
  int*                            returnCode,
  CciChar*                        name,
  CciDataContext*                 userContext,
  CCI_UE_VFT*                     functionTable);
```

Parameters:

returnCode (output)
Requires the return code from the function. Possible values are:

- CCI_DUP_USER_EXIT_NAME

  The specified name matches the name of a user exit previously registered in the current execution group.
- CCI_INV_USER_EXIT_NAME

  The specified name was invalid. This can be caused if a NULL pointer, empty string or a string containing non-alphanumeric characters was specified.

Name (input)
This parameter must contain a pointer to a NULL-terminated string of CciChars specifying a name for the user exit. The name must be unique across all user exits that can be installed on the same broker. This name is used to identify the user exit in, for example:

- User Trace messages
- Exceptions or syslog messages
- Administration commands (for example, **mqsichangeflowuserexits**)

The name has the following restrictions:

- It must consist of alphanumeric characters only.
- It must be no greater than 255 characters.
- The name must be unique across all user exits that can be installed on the same broker.

userContext (input)
This parameter allows the caller to provide a context pointer that is passed to the callback function when it is called. This parameter can be NULL.

functionTable (input)
This parameter is a pointer to a struct whose fields must contain either pointers to functions matching the correct signatures or contain NULL. A NULL value for any of these fields indicates that the user exit must not be called for that event.

Initialize the structure by using the define CCI_UE_VFT_DEFAULT, which sets the version as CCI_UE_VFT_CURRENT_VERSION. The cciOutputMessageCallback was added at version 2, CCI_UE_VFT_VERSION_2.

Return values:
None. If an error occurs, the **returnCode** parameter indicates the reason for the error.

Example:

```
extern "C"{

void bipInitializeUserExits(){

  int rc = CCI_SUCCESS;
  CCI_UE_VFT myVft = {CCI_UE_VFT_DEFAULT};
  myVft.iFpInputMessageCallback      = myInputMessageCallback;
  myVft.iFpTransactionEventCallback  = myTransactionEventCallback;
  myVft.iFpPropagatedMessageCallback = myPropagatedMessageCallback;
  myVft.iFpNodeCompletionCallback    = myNodeCompletionCallback;
  myVft.iFpOutputMessageCallback     = myOutputMessageCallback;

  cciRegisterUserExit(&rc,
                      MyConstants::myUserExitName,
                      0,
                      &myVft);

  /*you should now check the rc for unexpected values*/

  return;
}

}/*end of extern "C" */
```

Related concepts:

"User exits" on page 3015
A user exit is user-provided custom software, written in C, to track data passing through message flows.

Related tasks:

"Developing a user exit" on page 3114
Develop a user exit by declaring it, implementing its behavior, then compiling it.

Related reference:

"bipInitializeUserExits" on page 6617
bipInitializeUserExits is an implementation function that is exported by the User Exit library (`.lel` file).

"cciInputMessageCallback" on page 6619
The cciInputMessageCallback function can be registered as a callback and is called every time a message is read by an input node, and before that message is propagated down the message flow.

"cciTransactionEventCallback" on page 6625
The cciTransactionEventCallback function can be registered as a callback and is called every time a message flow transaction ends.

"cciPropagatedMessageCallback" on page 6623
The cciPropagatedMessageCallback function can be registered as a callback and is called whenever a message is propagated from one node to another.

"cciNodeCompletionCallback" on page 6621
The cciNodeCompletionCallback function can be registered as a callback and is called whenever a node has completed processing of a message and is returning control to its upstream node.

"cciOutputMessageCallback" on page 6626
The cciOutputMessageCallback function can be registered as a callback and is called whenever a message is sent by an output node.

C common API

The C language common API consists of implementation and utility functions that you can use in user-defined nodes, parsers, and user exits.

All these functions are defined in the `BipCpi.h` header file, see “C Header files” on page 6415.

- “C common implementation functions.”
- “C common utility functions” on page 6643.

Related concepts:

“Developing user-defined extensions” on page 2970
A user-defined extension is a component that you design and implement to extend the function of WebSphere Message Broker.

Related reference:

“C language user-defined node API” on page 6416
Learn about the different types of call provided by the API.

“C language user-defined parser API” on page 6538
The C language user-defined parser API consists of two complementary sets of functions that you can use to define the function of your parser.

“C user exit API” on page 6615
The user exit API defines implementation and utility functions.

C common implementation functions:

You can use the common implementation functions in user-defined nodes and user-defined parsers. All the functions are called by the broker on occurrence of certain events.

These functions are defined in the`BipCci.h` header file.

Optional functions

- cciRegCallback

Related reference:

“cciRegCallback”
This function can be registered as a callback function, and is called when the registered event occurs. The function is registered by providing a function pointer which matches a particular typedef.

cciRegCallback:

This function can be registered as a callback function, and is called when the registered event occurs. The function is registered by providing a function pointer which matches a particular typedef.

Syntax:

```
typedef int (*CciRegCallback)(CciDataContext *, cciCallbackType);
```

Parameters:

`type CciDataContext*`
This parameter is the pointer that is provided by the caller to the registration function.

`type CciCallbackType`
This parameter indicates the reason for the callback. The reason is always one of the CciCallbackType values that is specified on the registration call corresponding to this callback.

Return values:

type CciRegCallbackStatus (defined in BipCci.h)

- CCI_THREAD_STATE_REGISTRATION_RETAIN: This return code is used for a callback that is to remain registered as a callback function on a particular thread.
- CCI_THREAD_STATE_REGISTRATION_REMOVE: This return code is used to signify that the callback is to be de-registered, and that it must not be called again on this thread unless it is reregistered.
- If another value is returned, a warning is written to a log, and CCI_THREAD_STATE_REGISTRATION_RETAIN is assumed.

During execution of this function, it is possible that the node or parser that has registered the function has already been deleted. Therefore, you must not call a node or parser utility function that depends on the existence of a node or parser. The only utility functions that can be called from this callback are:

- cciLog
- cciUserTrace
- cciServiceTrace
- cciUserDebugTrace
- cciServiceDebugTrace
- cciIsTraceActive

For each of these five trace utility functions, the CciObject parameter must be NULL.

Example:
Declare the following struct and function:

```
typedef struct {
    int         id;
}MyContext;

static int registered=0;

CciRegCallbackStatus switchThreadStateChange(CciDataContext *context, CciCallbackType type)
{
  char traceText[256];
  char* typeStr=0;
  MyContext* myContext = (MyContext*)context;

  if (type==CCI_THREAD_STATE_IDLE){
      typeStr = "idle";
  }else if(type==CCI_THREAD_STATE_INSTANCE_END){
      typeStr = "instance end";
  }else if (type==CCI_THREAD_STATE_TERMINATION){
      typeStr = "termination";
  }else{
      typeStr = "unknown";
  }

  memset(traceText,0,256);
  sprintf(traceText,"switchThreadStateChange: context id = %d, thread state %s",myContext->id,typeStr);
  cciServiceTrace(NULL,
                  NULL,
                  traceText);
  return CCI_THREAD_STATE_REGISTRATION_RETAIN;

}
```

Place the following code into the _Switch_evaluate function in the samples to enable you to read service trace, and see when the message processing thread changes state:

```
/*register for thread state change*/
  CciMessageContext* messageContext = cniGetMessageContext(NULL,message);
  CciThreadContext*  threadContext  = cniGetThreadContext(NULL,messageContext);

  static MyContext myContext={1};

  if(registered==0){
    cciRegisterForThreadStateChange(
                  NULL,
                  threadContext,
                  & myContext,
                  switchThreadStateChange,
                  CCI_THREAD_STATE_IDLE |
                  CCI_THREAD_STATE_INSTANCE_END |
                  CCI_THREAD_STATE_TERMINATION);

    registered=1;

  }
```

This example registers only on the first thread that receives a message. If it is necessary to register every thread that receives a message, the user-defined extensions must remember on which threads they have registered.

By using the userContext parameter you can see how data is passed from the code where the callback is registered to the actual callback function.

When registering the callback, a pointer to an instance of the **MyContext** struct is passed in. This pointer is the pointer that is passed back to the callback. To ensure that the pointer is still valid when it is passed back to the callback, an instance of the struct is declared as static. Another technique to ensure that the pointer is valid is to allocate storage on the heap.

In the callback function, the userContext parameter can be cast to a **(MyContext*)**. The original MyContext struct can be referenced through this address. This technique permits the passing of data from the code where the callback is registered to the callback function.

Related reference:

"cciRegisterForThreadStateChange" on page 6656
This function registers a function to be called when the current thread enters a particular state.

C common utility functions:

WebSphere Message Broker provides some additional utilities that user-defined nodes and parsers can use.

These utilities belong to the following categories:

- Exception handling and logging
- Character representation handling

These functions are defined in the BipCci.h header file.

The following exception handling and logging functions are provided for use by a user-defined node or parser:

- "cciGetLastExceptionData" on page 6647
- "cciGetLastExceptionDataW" on page 6649
- "cciLog" on page 6651
- "cciLogW" on page 6653
- "cciRethrowLastException" on page 6659
- "cciThrowException" on page 6666
- "cciThrowExceptionW" on page 6668

The following utilities help you convert between the processing code (in UCS-2) that is used internally by WebSphere Message Broker, and file codes (for example, ASCII).

- "cciMbsToUcs" on page 6655
- "cciUcsToMbs" on page 6672

The following utility functions enable you to determine whether trace is active, and write entries that are appropriate for the trace settings.

- "cciIsTraceActive" on page 6670
- "cciUserTrace" on page 6678
- "cciUserTraceW" on page 6681
- "cciUserDebugTrace" on page 6673
- "cciUserDebugTraceW" on page 6676
- "cciServiceTrace" on page 6663
- "cciServiceTraceW" on page 6664
- "cciServiceDebugTrace" on page 6660
- "cciServiceDebugTraceW" on page 6661

The following utility function is used to register a function that is to be called when the current thread enters a particular state:

- "cciRegisterForThreadStateChange" on page 6656

The following utility functions are available for use with user exits:

- "cciGetBrokerInfo" on page 6645
- "cciGetNodeAttribute" on page 6629
- "cciGetNodeName" on page 6630
- "cciGetNodeType" on page 6631
- "cciGetSourceNode" on page 6635
- "cciGetSourceTerminalName" on page 6636
- "cciGetTargetNode" on page 6637
- "cciGetTargetTerminalName" on page 6637
- "cciInputMessageCallback" on page 6619
- "cciNodeCompletionCallback" on page 6621
- "cciPropagatedMessageCallback" on page 6623
- "cciRegisterUserExit" on page 6638
- "cciTransactionEventCallback" on page 6625

cciGetBrokerInfo:

Use the cciGetBrokerInfo function to query the current broker environment (for example, for information about broker name, execution group name, queue manager name). The information is returned in a structure of type CCI_BROKER_INFO_ST.

This function differs from cniGetBrokerInfo in that you do not have to specify a CciNode* handle, and that it does not return information about a message flow. Therefore, you can call cciGetBrokerInfo from initialization functions; for example, bipInitializeUserExits, bipGetMessageParserFactory, and bipGetMessageFlowNodeFactory.

Syntax:

```
void cciGetBrokerInfo(
                      int*                returnCode,
                      CCI_BROKER_INFO_ST* broker_info_st);
```

Parameters:

returnCode (output)
Receives the return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_INV_BROKER_INFO_ST
- CCI_EXCEPTION

broker_info_st (output)
The address of a CCI_BROKER_INFO_ST structure to be populated with the relevant values on successful completion:

```
typedef struct cci_broker_info_st {
int versionId;   /*Structure version identification*/
CCI_STRING_ST brokerName;  /*The label of the broker*/
CCI_STRING_ST executionGroupName; /*The label of the current execution group*/
CCI_STRING_ST queueManagerName;  /*The name of the MQ Queue Manager for the broker*/
CCI_STRING_ST dataSourceUserId;  /*The userid broker connects to datasource as*/
} CCI_BROKER_INFO_ST;
```

Return values:
None. If an error occurs, the **returnCode** parameter indicates the reason for the error.

Example:

```
int rc = CCI_SUCCESS;

CCI_BROKER_INFO_ST brokerInfo = {CCI_BROKER_INFO_ST_DEFAULT};

#define INTITIAL_STR_LEN 256
CciChar brokerNameStr[INTITIAL_STR_LEN];
CciChar executionGroupNameStr[INTITIAL_STR_LEN];
CciChar queueManagerNameStr[INTITIAL_STR_LEN];

brokerInfo.brokerName.bufferLength = INTITIAL_STR_LEN;
brokerInfo.brokerName.buffer       = brokerNameStr;

brokerInfo.executionGroupName.bufferLength = INTITIAL_STR_LEN;
brokerInfo.executionGroupName.buffer = executionGroupNameStr;

brokerInfo.queueManagerName.bufferLength = INTITIAL_STR_LEN;
brokerInfo.queueManagerName.buffer = queueManagerNameStr;
```

```
cciGetBrokerInfo(&rc,&brokerInfo);

/* just in case any of the buffers were too short*/
if ((brokerInfo.brokerName.bytesOutput         < brokerInfo.brokerName.dataLength)  ||
    (brokerInfo.executionGroupName.bytesOutput < brokerInfo.executionGroupName.dataLength) ||
    (brokerInfo.queueManagerName.bytesOutput   < brokerInfo.queueManagerName.dataLength))  {

  /*at least one of the buffer were too short, need to rerty*/
  /* NOTE this is unlikely given that the initial sizes were reasonably large*/

  brokerInfo.brokerName.bufferLength =
     brokerInfo.brokerName.dataLength;
  brokerInfo.brokerName.buffer       =
    (CciChar*)malloc (brokerInfo.brokerName.bufferLength * sizeof(CciChar));

  brokerInfo.executionGroupName.bufferLength =
    brokerInfo.executionGroupName.dataLength;
  brokerInfo.executionGroupName.buffer       =
    (CciChar*)malloc (brokerInfo.executionGroupName.bufferLength * sizeof(CciChar));

  brokerInfo.queueManagerName.bufferLength =
    brokerInfo.queueManagerName.dataLength;
  brokerInfo.queueManagerName.buffer       =
    (CciChar*)malloc (brokerInfo.queueManagerName.bufferLength * sizeof(CciChar));

  cciGetBrokerInfo(&rc,&brokerInfo);

  /*now do something sensible with these strings before the buffers go out of scope*/
  /* for example call a user written function to copy them away*/
  copyBrokerInfo(brokerInfo.brokerName.buffer,
                 brokerInfo.executionGroupName.buffer,
                 brokerInfo.queueManagerName.buffer);

  free((void*)brokerInfo.brokerName.buffer);
  free((void*)brokerInfo.executionGroupName.buffer);
  free((void*)brokerInfo.queueManagerName.buffer);

}else{
  /*now do something sensible with these strings before the buffers go out of scope*/
  /* for example call a user written function to copy them away*/
  copyBrokerInfo(brokerInfo.brokerName.buffer,
                 brokerInfo.executionGroupName.buffer,
                 brokerInfo.queueManagerName.buffer);
}
```

Related concepts:

"User exits" on page 3015
A user exit is user-provided custom software, written in C, to track data passing through message flows.

Related tasks:

"Developing a user exit" on page 3114
Develop a user exit by declaring it, implementing its behavior, then compiling it.

Related reference:

"cniGetBrokerInfo" on page 6488
Use this function to query the current broker environment (for example, for information about broker name and message flow name). The information is returned in a structure of type CNI_BROKER_INFO_ST.

"bipInitializeUserExits" on page 6617
bipInitializeUserExits is an implementation function that is exported by the User Exit library (`.lel` file).

cciGetLastExceptionData:

Gets diagnostic information about the last exception generated. Information about the last exception generated on the current thread is returned in a CCI_EXCEPTION_ST output structure. The user-defined extension can use this function to determine whether any recovery is required when a utility function returns an error code.

You can call this function when a utility function or user exit callback indicates that an exception has occurred, by setting *returnCode* to CCI_EXCEPTION.

You must call cciGetLastExceptionData() only when CCI_EXCEPTION is indicated; at other times, the function returns unpredictable results.

The *traceText* that is associated with the exception converts to a `char*` if the `char*` is US-ASCII. If the *traceText* is in another language, use cciGetLastExceptionDataW and its associated CCI_EXCEPTION_WIDE_ST structure, which stores the *traceText* as UTF-16.

If the exception has been raised by the broker, or by cciThrowExceptionW, the *traceText* element of the CCI_EXCEPTION_ST structure is an empty string.

Syntax:

```
void* cciGetLastExceptionData(
  int*               returnCode,
  CCI_EXCEPTION_ST*  exception_st);
```

Parameters:

`returnCode`
Receives the return code from the function (output). Possible return codes are:

- CCI_INV_DATA_POINTER
- CCI_NO_EXCEPTION_EXISTS
- CCI_EXCEPTION
- CCI_EXCEPTION_UNKNOWN
- CCI_EXCEPTION_FATAL
- CCI_EXCEPTION_RECOVERABLE
- CCI_EXCEPTION_CONFIGURATION
- CCI_EXCEPTION_PARSER
- CCI_EXCEPTION_CONVERSION
- CCI_EXCEPTION_DATABASE
- CCI_EXCEPTION_USER

`exception_st`
Specifies the address of a CCI_EXCEPTION_ST structure to receive data about the last exception (output). The type value returned in the lower four bits of the `exception_st.type` field is one of the following type values:

- CCI_EXCEPTION_ST_TYPE_EXCEPTION_BASE
- CCI_EXCEPTION_ST_TYPE_EXCEPTION_TERMINATION
- CCI_EXCEPTION_ST_TYPE_EXCEPTION_FATAL
- CCI_EXCEPTION_ST_TYPE_EXCEPTION_RECOVERABLE
- CCI_EXCEPTION_ST_TYPE_EXCEPTION_CONFIGURATION
- CCI_EXCEPTION_ST_TYPE_EXCEPTION_PARSER

- CCI_EXCEPTION_ST_TYPE_EXCEPTION_CONVERSION
- CCI_EXCEPTION_ST_TYPE_EXCEPTION_DATABASE
- CCI_EXCEPTION_ST_TYPE_EXCEPTION_USER

The value returned in the `exception_st.messageNumber` field, for exceptions resulting in a BIP cataloged exception message, contains the message level in the high order bytes and the BIP message number in the lower four bytes.

Return values:
None. If an error occurs, the *returnCode* parameter indicates the reason for the error.

Example:

```
typedef struct exception_st {
  int               versionId;     /* Structure version identification */
  int               type;          /* Type of exception */
  int               messageNumber; /* Message number */
  int               insertCount;   /* Number of message inserts */
  CCI_STRING_ST     inserts[CCI_MAX_EXCEPTION_INSERTS];
                                   /* Array of message insert areas */
  const char*       fileName;      /* Source: file name */
  int               lineNumber;    /* Source: line number in file */
  const char*       functionName;  /* Source: function name */
  const char*       traceText;     /* Trace text associated with exception */
  CCI_STRING_ST     objectName;    /* Object name */
  CCI_STRING_ST     objectType;    /* Object type */
} CCI_EXCEPTION_ST;
char msgnumTypeStr[64];


CCI_EXCEPTION_ST exception_st = malloc(sizeof(CCI_EXCEPTION_ST));
int rc = 0;
memset(&exception_st,0,sizeof(exception_st));
cciGetLastExceptionData(&rc, &exception_st);
sprintf(msgnumTypeStr, "MsgNum: %d Type: %d",
(exception_st.messageNumber & 0x0ffff),
(exception_st.type & 0x0f));
```

Related concepts:

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Creating an input node in C" on page 3027
Create a user-defined input node in C to receive messages into a message flow.

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"User-defined extensions" on page 6411
Reference material that supports the creation and management of your user-defined extensions.

"cciLog" on page 6651
Use cciLog to write an error, warning, or informational event.

"cciRethrowLastException" on page 6659
This function re-throws the last exception that has been generated on the current thread. It is used to pass the exception back to the broker for further handling. The function, like a C **exit** call, does not return to the caller.

"cciThrowException" on page 6666
Use this function to throw an exception. The exception is thrown by the broker interface, and includes the specified arguments as exception data.

cciGetLastExceptionDataW:

Gets diagnostic information about the last exception generated. Information about the last exception generated on the current thread is returned in a CCI_EXCEPTION_WIDE_ST output structure. The user-defined extension uses this function to determine whether any recovery is required when a utility function returns an error code.

You can call this function, when a utility function or user exit callback indicates that an exception has occurred, by setting *returnCode* to CCI_EXCEPTION.

Unless CCI_EXCEPTION is indicated you must not call cciGetLastExceptionDataW() because it returns unpredictable results.

Syntax:

```
void* cciGetLastExceptionDataW(
  int*                    returnCode,
  CCI_EXCEPTION_WIDE_ST*  exception_st);
```

Parameters:

returnCode
Receives the return code from the function (output). Possible return codes are:
- CCI_INV_DATA_POINTER
- CCI_NO_EXCEPTION_EXISTS
- CCI_EXCEPTION
- CCI_EXCEPTION_UNKNOWN
- CCI_EXCEPTION_FATAL
- CCI_EXCEPTION_RECOVERABLE
- CCI_EXCEPTION_CONFIGURATION
- CCI_EXCEPTION_PARSER
- CCI_EXCEPTION_CONVERSION
- CCI_EXCEPTION_DATABASE
- CCI_EXCEPTION_USER

exception_st
Specifies the address of a CCI_EXCEPTION_WIDE_ST structure to receive data about the last exception (output). The type value returned in the lower four bits of the **exception_st.type** field is one of the following type values:
- CCI_EXCEPTION_ST_TYPE_EXCEPTION_BASE
- CCI_EXCEPTION_ST_TYPE_EXCEPTION_TERMINATION
- CCI_EXCEPTION_ST_TYPE_EXCEPTION_FATAL
- CCI_EXCEPTION_ST_TYPE_EXCEPTION_RECOVERABLE
- CCI_EXCEPTION_ST_TYPE_EXCEPTION_CONFIGURATION
- CCI_EXCEPTION_ST_TYPE_EXCEPTION_PARSER

- CCI_EXCEPTION_ST_TYPE_EXCEPTION_CONVERSION
- CCI_EXCEPTION_ST_TYPE_EXCEPTION_DATABASE
- CCI_EXCEPTION_ST_TYPE_EXCEPTION_USER

The value returned in the **exception_st.messageNumber** field, for exceptions resulting in a BIP cataloged exception message, contains the message level in the high order bytes and the BIP message number in the lower four bytes.

Return values:
None. If an error occurs, the *returnCode* parameter indicates the reason for the error.

Example:

```
typedef struct exception_wide_st {
  int               versionId;     /* Structure version identification */
  int               type;          /* Type of exception */
  int               messageNumber; /* Message number */
  int               insertCount;   /* Number of message inserts */
  CCI_STRING_ST     inserts[CCI_MAX_EXCEPTION_INSERTS];
                                   /* Array of message insert areas */
  const char*       fileName;     /* Source: file name */
  int               lineNumber;     /* Source: line number in file */
  const char*       functionName;  /* Source: function name */
  CCI_STRING_ST     traceText;     /* Trace text associated with exception */
  CCI_STRING_ST     objectName;    /* Object name */
  CCI_STRING_ST     objectType;    /* Object type */
} CCI_EXCEPTION_WIDE_ST;
char msgnumTypeStr[64];


CCI_EXCEPTION_WIDE_ST exception_st = malloc(sizeof(CCI_EXCEPTION_WIDE_ST));
int rc = 0;
memset(&exception_st,0,sizeof(exception_st));
cciGetLastExceptionDataW(&rc, &exception_st);
sprintf(msgnumTypeStr, "MsgNum: %d Type: %d",
(exception_st.messageNumber & 0x0ffff),
(exception_st.type & 0x0f));
```

Related reference:

"cciUserTrace" on page 6678
Use cciUserTrace to write a message from a message catalog (with inserts) to user trace. A message is also written to service trace, if service trace is active.

"cciServiceTrace" on page 6663
Writes a message to service trace, if service trace is active.

"cciServiceDebugTrace" on page 6660
This function is very similar to cciServiceTrace with the only difference being that the entry is written to service trace only when service trace is active at debug level.

"cciIsTraceActive" on page 6670
cciIsTraceActive reports whether trace is active and the level at which trace is active.

"Trace logging from a user-defined C extension" on page 6693
Message processing nodes and parsers that are written to the C programming language API can write entries to trace.

"cciThrowExceptionW" on page 6668
The cciThrowExceptionW exception is thrown by the broker interface and uses the specified arguments as exception data.

"cciLogW" on page 6653
cciLogW logs an error, warning, or informational event. The event is logged by the

broker interface and uses the specified arguments as log data.

"cciUserTraceW" on page 6681
Use cciUserTraceW to write a message from a message catalog (with inserts) to user trace. A message is also written to service trace, if service trace is active.

"cciUserDebugTraceW" on page 6676
Use cciUserDebugTraceW to write a message from a message catalog (with inserts) to user trace when user trace is active at debug level. A message is also written to service trace, if service trace is active.

"cciServiceTraceW" on page 6664
cciServiceTraceW writes a message to service trace, if service trace is active.

"cciServiceDebugTraceW" on page 6661
The function is very similar to cciServiceTraceW with the only difference being that the entry is written to service trace only when service trace is active at debug level.

"User trace" on page 6873
User trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Administration log. User trace is inactive by default; you must activate it explicitly by using a command, or by selecting options in the WebSphere Message Broker Toolkit.

cciLog:

Use cciLog to write an error, warning, or informational event.

The event is logged by the broker interface, and includes the specified arguments as log data.

Syntax:

```
void cciLog(
  int*          returnCode,
  CCI_LOG_TYPE  type,
  char*         file,
  int           line,
  char*         function,
  CciChar*      messageSource,
  int           messageNumber,
  char*         traceText,
                ...);
```

Parameters:

returnCode
The return code from the function (output). Possible return codes are:
- CCI_SUCCESS
- CCI_INV_DATA_POINTER
- CCI_INV_LOG_TYPE

type The type of event, as defined by CCI_LOG_TYPE (input). Valid values are:
- CCI_LOG_ERROR
- CCI_LOG_WARNING
- CCI_LOG_INFORMATION

file The source file name where the function was invoked (input). The value is optional, but it is useful for debugging purposes.

line The line number in the source file where the function was invoked (input). The value is optional, but it is useful for debugging purposes.

function
The function name that invoked the log function (input). The value is optional, but it is useful for debugging purposes.

messageSource
The fully-qualified location and name of the Windows message source or the Linux, UNIX, or z/OS message catalog.

To use the current broker message catalog, specify BIPmsgs on all operating systems. Alternatively, you can create your own message catalog.

messageNumber
The message number identifying the event (input). If *messageNumber* is specified as zero, it is assumed that a message is not available. If *messageNumber* is non-zero, the specified message is written into the broker event log with any inserts provided in the variable argument list.

traceText
Trace information that is written into the broker service trace log (input). The information is optional, but it is useful for debugging purposes.

... A C variable argument list containing any message inserts that accompany the message (input). These inserts are treated as character strings, and the variable arguments are assumed to be of type `pointer to char`.

`char*` characters must be strings in either ASCII (Latin) or EBCDIC (1047).

The last argument in this list must be `(char*)0`.

Return values:
None. If an error occurs, the *returnCode* parameter indicates the reason for the error.

Related concepts:

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Creating an input node in C" on page 3027
Create a user-defined input node in C to receive messages into a message flow.

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

"Creating message catalogs" on page 3138
Create your own message catalogs to write tailored entries to the local error log.

Related reference:

"User-defined extensions" on page 6411
Reference material that supports the creation and management of your user-defined extensions.

"cciGetLastExceptionData" on page 6647
Gets diagnostic information about the last exception generated. Information about the last exception generated on the current thread is returned in a CCI_EXCEPTION_ST output structure. The user-defined extension can use this function to determine whether any recovery is required when a utility function returns an error code.

"cciRethrowLastException" on page 6659
This function re-throws the last exception that has been generated on the current thread. It is used to pass the exception back to the broker for further handling. The function, like a C **exit** call, does not return to the caller.

"cciThrowException" on page 6666
Use this function to throw an exception. The exception is thrown by the broker interface, and includes the specified arguments as exception data.

cciLogW:

cciLogW logs an error, warning, or informational event. The event is logged by the broker interface and uses the specified arguments as log data.

Syntax:

```
void cciLogW(
  int*              returnCode,
  CCI_LOG_TYPE      type,
  const char*       file,
  int               line,
  const char*       function,
  const CciChar*    messageSource,
  int               messageNumber,
  const CciChar*    traceText,
                    ...
);
```

Parameters:

returnCode
The return code from the function (output). If the *messageSource* parameter is null, the *returnCode* is set to CCI_INV_DATA_POINTER.

type The type of event, as defined by CCI_LOG_TYPE (input). Valid values are:

- CCI_LOG_ERROR
- CCI_LOG_WARNING
- CCI_LOG_INFORMATION

file The source file name where the function was invoked (input). The value is optional, but it is useful for debugging purposes.

line The line number in the source file where the function was invoked (input). The value is optional, but it is useful for debugging purposes.

function
The function name that invoked the log function (input). The value is optional, but it is useful for debugging purposes.

messageSource
The fully-qualified location and name of the Windows message source or the Linux, UNIX, or z/OS message catalog.

To use the current broker message catalog, specify BIPmsgs on all operating systems. Alternatively, you can create your own message catalog.

messageNumber
The message number identifying the event (input). If *messageNumber* is specified as zero, it is assumed that a message is not available. If *messageNumber* is non-zero, the specified message is written into the broker event log with any inserts provided in the variable argument list (see example).

traceText
Trace information that is written into the broker service trace log (input). The information is optional, but it is useful for debugging purposes.

... A C variable argument list containing any message inserts that accompany the message (input). These inserts are treated as character strings and the variable arguments are assumed to be of type `pointer to CciChar`.

The last argument in this list *must* be `(CciChar*)0`.

Return values:
None. If an error occurs, the *returnCode* parameter indicates the reason for the error.

Example:

```
void logSomethingWithBroker(CciChar* helpfulText,
                            char* file,
                            int line,
                            char* func
                            ){
  int rc = CCI_SUCCESS;
  /* set up the message catalog name */
  const CciChar* catalog = CciString("BIPmsgs", BIP_DEF_COMP_CCSID);

  cciLogW(&rc,
          CCI_LOG_INFORMATION
          file, line, func,
          catalog, BIP2111,
          helpfulText,
          helpfulText,
          (CciChar*)0
          );

  if(CCI_SUCCESS != rc){
    const CciChar* message = CciString("Failed to log message",
                                       BIP_DEF_COMP_CCSID);
    raiseExceptionWithBroker(message,
                             __FILE__,
                             __LINE__,
                             "logSomethingWithBroker");
  }
}
```

Related concepts:

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Creating an input node in C" on page 3027
Create a user-defined input node in C to receive messages into a message flow.

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

"Using error logging from a user-defined extension" on page 3137
Program user-defined extensions to write entries in the local error log.

Related reference:

"User-defined extensions" on page 6411
Reference material that supports the creation and management of your user-defined extensions.

"cciGetLastExceptionData" on page 6647
Gets diagnostic information about the last exception generated. Information about the last exception generated on the current thread is returned in a CCI_EXCEPTION_ST output structure. The user-defined extension can use this function to determine whether any recovery is required when a utility function returns an error code.

"cciRethrowLastException" on page 6659
This function re-throws the last exception that has been generated on the current thread. It is used to pass the exception back to the broker for further handling. The function, like a C **exit** call, does not return to the caller.

"cciThrowException" on page 6666
Use this function to throw an exception. The exception is thrown by the broker interface, and includes the specified arguments as exception data.

cciMbsToUcs:

Use this function to convert multibyte string data to Universal Character Set (UCS).

Syntax:

```
int cciMbsToUcs(
  int*         returnCode,
  const char*  mbString,
  CciChar*     ucsString,
  int          ucsStringLength,
  int          codePage);
```

Parameters:

returnCode
The return code from the function (output). Possible return codes are:
- CCI_SUCCESS
- CCI_BUFFER_TOO_SMALL
- CCI_INV_CHARACTER
- CCI_FAILURE
- CCI_INV_CODEPAGE

mbString
The string to be converted, expressed as 'file code' (input).

ucsString
The location of the resulting UCS-2 Unicode string (input). This string has a trailing `CciChar` of 0, just as the *mbString* has a trailing byte of 0.

ucsStringLength
The length (in `CciChar`) of the buffer that you have provided (input). Each byte in *mbString* expands to not more than one `CciChar` character, and this defines an upper limit for the buffer size required.

codePage

The code page of the source string (input). The value of the code page must be suitable for the compiler that you are using to compile the user-defined node.

For an ASCII system, a value of 1208 (meaning code page ibm-1208, which is UTF-8 Unicode) is a good choice if you are using cciMbsToUcs to convert string constants for processing by WebSphere Message Broker. 1208 is appropriate for Linux, UNIX, and Windows systems.

On Linux, UNIX systems, `nl_langinfo(CODEPAGE)` gives you the code page that has been selected by `setlocale`.

On z/OS, the default code page for WebSphere MQ, which is 500, must not be used. Instead, use a code page value of 1047.

Return values:
The *returnCode* parameter is set to the converted length in half-words (UCS-2 characters).

Related concepts:

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Creating an input node in C" on page 3027
Create a user-defined input node in C to receive messages into a message flow.

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"User-defined extensions" on page 6411
Reference material that supports the creation and management of your user-defined extensions.

"cciUcsToMbs" on page 6672
Use this function to convert Universal Character Set (UCS) data to multibyte string data. This function is, typically, used only for formatting diagnostic messages. Normal processing is best done in UCS-2, which can represent all characters from all languages.

cciRegisterForThreadStateChange:

This function registers a function to be called when the current thread enters a particular state.

Syntax:

```
void cciRegisterForThreadStateChange(
          int               *returnCode,
          CciThreadContext  *threadContext,
          CciDataContext    *userContext,
          CciRegCallback     callback,
          CciCallbackType    type);
```

Parameters:

returnCode
The return code from the function (output). An input value of NULL signifies that errors are silently handled, or are ignored by the broker. If the input value is not NULL, the output value signifies the success status of the call. If the `threadContext` parameter is not valid, `*returnCode` is set to CCI_INV_THREAD_CONTEXT, and the callback is not registered.

threadContext
This parameter provides the thread context in which to register the callback function and associated data. It is assumed that this parameter is obtained by calling cniGetThreadContext() on the current thread. If NULL is supplied as `threadContext`, the thread context is determined by the framework. This method is less efficient than calling cniGetThreadContext.

userContext
This parameter allows the caller to provide a context pointer that is passed to the callback function when it is called. This parameter can be NULL.

callback
This parameter is a pointer to the callback function that is to be called. This function must be of the type `CciRegCallback`.

type
This parameter specifies whether the callback is to be called at the time when the thread is ending, or when the thread is in one of the idle states. The idle states can be one of the following values:

- CCI_THREAD_STATE_IDLE:

  The input node for the current thread is actively polling for data from the input source, but no data is available. Messages are not propagated through the message flow until data becomes available for the input node.
- CCI_THREAD_STATE_INSTANCE_END

  The input node for the current thread has stopped polling for data and the thread has been released. The thread is dispatched again either by the same input node or by another input node in the same message flow. This state is entered when additional instances, which have been deployed for a message flow, have been configured to cope with an influx of input data that has now ceased. The input node continues to poll for input data on a single thread, and the other threads are released.
- CCI_THREAD_STATE_TERMINATION

  The current thread is ending. This action can happen when the broker is shut down, the execution group process is ending in a controlled manner, or when the message flow is being deleted. This state can occur after all nodes and parsers in the flow are deleted.

Alternatively, the `type` parameter can be the result of a bit-wise `OR` operation on two or more of these values. In this case, the specified function is called when the thread enters the relevant state for each individual `type` value.

Return values:
None. If an error occurs, the *returnCode* parameter indicates the reason for the error.

Example:
Declaring the struct and function:

```
typedef struct {
    int         id;
}MyContext;
```

```
static int registered=0;

CciRegCallbackStatus switchThreadStateChange(
           CciDataContext *context, CciCallbackType type)
{
  char    traceText[256];
  char*   typeStr=0;
  MyContext* myContext = (MyContext*)context;

  if (type==CCI_THREAD_STATE_IDLE){
      typeStr = "idle";
  }else if(type==CCI_THREAD_STATE_INSTANCE_END){
      typeStr = "instance end";
  }else if (type==CCI_THREAD_STATE_TERMINATION){
      typeStr = "termination";
  }else{
      typeStr = "unknown";
  }

  memset(traceText,0,256);
  sprintf(traceText,"switchThreadStateChange: context id = %d, thread state %s",myContext->id,typeStr);
  cciServiceTrace(NULL,
                  NULL,
                  traceText);
  return CCI_THREAD_STATE_REGISTRATION_RETAIN;

}
```

Place the following code into the `_Switch_evaluate` function in the samples to enable you to read service trace, and to see when the message processing thread changes state:

```
/*register for thread state change*/
  CciMessageContext* messageContext = cniGetMessageContext(NULL,message);
  CciThreadContext*  threadContext  = cniGetThreadContext(NULL,messageContext);

  static MyContext myContext={1};

  if(registered==0){
    cciRegisterForThreadStateChange(
                   NULL,
                   threadContext,
                   & myContext,
                   switchThreadStateChange,
                   CCI_THREAD_STATE_IDLE |
                   CCI_THREAD_STATE_INSTANCE_END |
                   CCI_THREAD_STATE_TERMINATION);
  registered=1;

  }
```

This example registers only on the first thread that receives a message. If it is necessary to register every thread that receives a message, the user-defined extensions must remember on which threads they have registered.

By using the `userContext` parameter, you can see how data is passed from the code where the callback is registered to the actual callback function.

When registering the callback, a pointer to an instance of the **MyContext** struct is passed in. This pointer is the same pointer as is passed back to the callback. To ensure that the pointer is still valid when it is passed back to the callback, an instance of the struct is declared as static. Another technique to ensure that the pointer is valid is to allocate storage on the heap.

In the callback function, the **userContext** parameter can be cast to a **(MyContext*)**. The original **MyContext** struct can be referenced through this address. This technique permits the passing of data from the code where the callback is registered to the callback function.

Related reference:

"cniGetThreadContext" on page 6494
This function returns the thread context for the current thread.

"cciRegCallback" on page 6641
This function can be registered as a callback function, and is called when the registered event occurs. The function is registered by providing a function pointer which matches a particular typedef.

cciRethrowLastException:

This function re-throws the last exception that has been generated on the current thread. It is used to pass the exception back to the broker for further handling. The function, like a C **exit** call, does not return to the caller.

Syntax:

```
void cciRethrowLastException(int* returnCode);
```

Parameters:

returnCode
The return code from the function (output). The possible return code is CCI_NO_EXCEPTION_EXISTS

Return values:
None. If an error occurs, the *returnCode* parameter indicates the reason for the error.

Example:

```
if (rc == CCI_EXCEPTION) {
     cciRethrowLastException(&rc);
   }
```

Related concepts:

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Creating an input node in C" on page 3027
Create a user-defined input node in C to receive messages into a message flow.

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"User-defined extensions" on page 6411
Reference material that supports the creation and management of your user-defined extensions.

"cciGetLastExceptionData" on page 6647
Gets diagnostic information about the last exception generated. Information about the last exception generated on the current thread is returned in a CCI_EXCEPTION_ST output structure. The user-defined extension can use this function to determine whether any recovery is required when a utility function returns an error code.

"cciLog" on page 6651
Use cciLog to write an error, warning, or informational event.

"cciThrowException" on page 6666
Use this function to throw an exception. The exception is thrown by the broker interface, and includes the specified arguments as exception data.

cciServiceDebugTrace:

This function is very similar to cciServiceTrace with the only difference being that the entry is written to service trace only when service trace is active at debug level.

Syntax:

```
void cciServiceDebugTrace(
  int*           returnCode,
  CciObject*     object,
  const char*    traceText
);
```

Parameters:

returnCode
Receives the return code from the function (output). A NULL pointer input signifies that the user-defined node does not want to deal with errors. Any exceptions thrown during the execution of this call will be re-thrown to the next upstream node in the flow. If input is not NULL, output will signify the success status of the call. If an exception occurs during execution, **returnCode* will be set to CCI_EXCEPTION on output. A call to **CciGetLastExceptionData** will provide details of the exception.

object (input)
The address of the object that is to be associated with the trace entry (input). This object can be a CciNode* or a CciParser*. If it is a CciNode*, the name of that node is written to trace. If it is a CciParser*, the name of the node that created the parser is written to trace. This object is also used to determine if the entry should be written to trace. The entry is only written if trace is active for the node. Currently nodes inherit their trace setting from the message flow.

If this parameter is NULL, the trace level for the execution group is returned.

traceText (input)
A string of characters that ends with NULL (input). This string will be written to service trace and provides an easy way to correlate trace entries with paths through the source code. For example, there could be several paths through the code that result in the same message (*messageSource* and *messageNumber*) being written to trace. *traceText* can be used to distinguish between these different paths. That is, the *traceText* string will be a static literal string in the source and therefore the same string will be in both the source code file and the formatted trace file.

This string must be in ISO-8859-1 (ibm-819) code page for user-defined extensions running on distributed platforms and must be in EBCDIC (1047) for user-defined extensions running on Z/OS See NLS section.

Return values:
None. If an error occurs, the *returnCode* parameter indicates the reason for the error.

Example:

```
CciNode*         thisNode = ((NODE_CONTEXT_ST*)context)->nodeObject;

cciServiceTrace(&rc,(CciObject*)thisNode,">>_Switch_evaluate()");
checkRC(rc);
```

Related reference:

"cciUserTrace" on page 6678
Use cciUserTrace to write a message from a message catalog (with inserts) to user trace. A message is also written to service trace, if service trace is active.

"cciUserDebugTrace" on page 6673
Use cciUserDebugTrace to write a message from a message catalog (with inserts) to user trace when user trace is active at debug level.

"cciServiceTrace" on page 6663
Writes a message to service trace, if service trace is active.

"cciIsTraceActive" on page 6670
cciIsTraceActive reports whether trace is active and the level at which trace is active.

"Trace logging from a user-defined C extension" on page 6693
Message processing nodes and parsers that are written to the C programming language API can write entries to trace.

"User trace" on page 6873
User trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Administration log. User trace is inactive by default; you must activate it explicitly by using a command, or by selecting options in the WebSphere Message Broker Toolkit.

cciServiceDebugTraceW:

The function is very similar to cciServiceTraceW with the only difference being that the entry is written to service trace only when service trace is active at debug level.

Syntax:

```
void cciServiceDebugTraceW(
  int*           returnCode,
  CciObject*     object,
  const CciChar* traceText
);
```

Parameters:

returnCode
Receives the return code from the function (output). A NULL pointer input signifies that the user-defined node does not want to deal with errors. Any exceptions thrown during the execution of this call will be re-thrown to the next upstream node in the flow. If input is not NULL, output will signify the success status of the call. If an exception occurs during execution, **returnCode* will be set to CCI_EXCEPTION on output. A call to **CciGetLastExceptionData** will provide details of the exception.

object (input)
The address of the object that is to be associated with the trace entry (input).

This object can be a CciNode* or a CciParser*. If it is a CciNode*, the name of that node is written to trace. If it is a CciParser*, the name of the node that created the parser is written to trace. This object is also used to determine if the entry should be written to trace. The entry is only written if trace is active for the node. Currently nodes inherit their trace setting from the message flow.

If this parameter is NULL, the trace level for the execution group is returned.

traceText (input)
A string of characters that ends with NULL (input). This string will be written to service trace and provides an easy way to correlate trace entries with paths through the source code. For example, there could be several paths through the code that result in the same message (*messageSource* and *messageNumber*) being written to trace. *traceText* can be used to distinguish between these different paths. That is, the *traceText* string will be a static literal string in the source and therefore the same string will be in both the source code file and the formatted trace file.

Return values:
None. If an error occurs, the *returnCode* parameter indicates the reason for the error.

Example:

```
CciNode* thisNode = ((NODE_CONTEXT_ST*)context)->nodeObject;
CciChar* traceText = CciString(">>_Switch_evaluate()",BIP_DEF_COMP_CCSID");
cciServiceTraceW(&rc,(CciObject*)thisNode,traceText);
checkRC(rc);
```

Related reference:

"cciUserTrace" on page 6678
Use cciUserTrace to write a message from a message catalog (with inserts) to user trace. A message is also written to service trace, if service trace is active.

"cciServiceTrace" on page 6663
Writes a message to service trace, if service trace is active.

"cciServiceDebugTrace" on page 6660
This function is very similar to cciServiceTrace with the only difference being that the entry is written to service trace only when service trace is active at debug level.

"cciIsTraceActive" on page 6670
cciIsTraceActive reports whether trace is active and the level at which trace is active.

"Trace logging from a user-defined C extension" on page 6693
Message processing nodes and parsers that are written to the C programming language API can write entries to trace.

"cciThrowExceptionW" on page 6668
The cciThrowExceptionW exception is thrown by the broker interface and uses the specified arguments as exception data.

"cciLogW" on page 6653
cciLogW logs an error, warning, or informational event. The event is logged by the broker interface and uses the specified arguments as log data.

"cciUserTraceW" on page 6681
Use cciUserTraceW to write a message from a message catalog (with inserts) to user trace. A message is also written to service trace, if service trace is active.

"cciUserDebugTraceW" on page 6676
Use cciUserDebugTraceW to write a message from a message catalog (with inserts) to user trace when user trace is active at debug level. A message is also written to

service trace, if service trace is active.

"cciServiceTraceW" on page 6664
cciServiceTraceW writes a message to service trace, if service trace is active.

"User trace" on page 6873
User trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Administration log. User trace is inactive by default; you must activate it explicitly by using a command, or by selecting options in the WebSphere Message Broker Toolkit.

cciServiceTrace:

Writes a message to service trace, if service trace is active.

The message that is written to service trace has the following format:

```
<date-time stamp> <threadNumber> +cciServiceTrace <nodeName> <nodeType> <traceText>, <nodeLabel>
```

Syntax:

```
void cciServiceTrace(
  int*           returnCode,
  CciObject*     object,
  const char*    traceText
);
```

Parameters:

returnCode
Receives the return code from the function (output). A NULL pointer input parameter signifies that the user-defined node does not handle errors. Exceptions thrown during the execution of this call are thrown again to the next upstream node in the flow. If the input parameter is not NULL, output signifies the success status of the call. If an exception occurs during execution, **returnCode* is set to CCI_EXCEPTION on output. Use CciGetLastExceptionData to get details of the exception.

object (input)
The address of the object that is to be associated with the trace entry (input). This object can be the address of a CciNode or a CciParser. If you specify a CciNode, the name of that node is written to trace. If you specify a CciParser, the name of the node that created the parser is written to trace. This object is also used to determine if the entry is written to trace. The entry is written only if trace is active for the node. Currently, nodes inherit their trace setting from the message flow.

If this parameter is NULL, the following actions are taken:

- <nodeName>, <nodeType>, <nodeLabel>, and <messageFlowLabel> are omitted from the trace entry.
- The entry is written based on the trace setting of the execution group.

traceText (input)
A string of characters that ends with NULL (input). This string is written to service trace, and you can use it to correlate trace entries with paths through the source code. For example, you might have several paths through the code that result in the same message (*messageSource* and *messageNumber*) being written to trace. *traceText* can be used to distinguish between these different paths. That is, the *traceText* string is a static literal string in the source, and therefore the same string is in both the source code file and the formatted trace file.

This string must be in ISO-8859-1 (ibm-819) code page for user-defined extensions running on distributed platforms, and must be in EBCDIC (1047) for user-defined extensions running on z/OS.

Return values:
None. If an error occurs, the *returnCode* parameter indicates the reason for the error.

Example:

```
CciNode*         thisNode = ((NODE_CONTEXT_ST*)context)->nodeObject;

cciServiceTrace(&rc,(CciObject*)thisNode,">>_Switch_evaluate()");
checkRC(rc);
```

Related reference:

"cciUserTrace" on page 6678
Use cciUserTrace to write a message from a message catalog (with inserts) to user trace. A message is also written to service trace, if service trace is active.

"cciUserDebugTrace" on page 6673
Use cciUserDebugTrace to write a message from a message catalog (with inserts) to user trace when user trace is active at debug level.

"cciServiceDebugTrace" on page 6660
This function is very similar to cciServiceTrace with the only difference being that the entry is written to service trace only when service trace is active at debug level.

"cciIsTraceActive" on page 6670
cciIsTraceActive reports whether trace is active and the level at which trace is active.

"Trace logging from a user-defined C extension" on page 6693
Message processing nodes and parsers that are written to the C programming language API can write entries to trace.

"User trace" on page 6873
User trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Administration log. User trace is inactive by default; you must activate it explicitly by using a command, or by selecting options in the WebSphere Message Broker Toolkit.

"Multicultural support considerations for message catalogs" on page 6694
WebSphere Message Broker converts any message that is loaded from the listed supported code pages into the local code page of the running processes (brokers) before output to the syslog.

cciServiceTraceW:

cciServiceTraceW writes a message to service trace, if service trace is active.

The message that is written to service trace has the following format:

```
<date-time stamp> <threadNumber> +cciServiceTrace <nodeName> <nodeType> <traceText>, <nodeLabel>
```

Syntax:

```
void cciServiceTraceW(
  int*           returnCode,
  CciObject*     object,
  const CciChar* traceText
);
```

Parameters:

returnCode
Receives the return code from the function (output). A NULL pointer input signifies that the user-defined node does not want to deal with errors. Any exceptions thrown during the execution of this call will be re-thrown to the next upstream node in the flow. If input is not NULL, output will signify the success status of the call. If an exception occurs during execution, **returnCode* will be set to CCI_EXCEPTION on output. A call to **CciGetLastExceptionData** will provide details of the exception.

object (input)
The address of the object that is to be associated with the trace entry (input). This object can be a CciNode* or a CciParser*. If it is a CciNode*, the name of that node is written to trace. If it is a CciParser*, the name of the node that created the parser is written to trace. This object is also used to determine if the entry should be written to trace. The entry is only written if trace is active for the node. Currently nodes inherit their trace setting from the message flow.

If this parameter is NULL, the trace level for the execution group is returned.

traceText (input)
A string of characters that ends with NULL (input). This string will be written to service trace and provides an easy way to correlate trace entries with paths through the source code. For example, there could be several paths through the code that result in the same message (*messageSource* and *messageNumber*) being written to trace. *traceText* can be used to distinguish between these different paths. That is, the *traceText* string will be a static literal string in the source and therefore the same string will be in both the source code file and the formatted trace file.

Return values:
None. If an error occurs, the *returnCode* parameter indicates the reason for the error.

Example:

```
CciNode*         thisNode = ((NODE_CONTEXT_ST*)context)->nodeObject;
const CciChar*   traceText = CciString(">>_Switch_evaluate()",
                                       BIP_DEF_COMP_CCSID);
cciServiceTraceW(&rc,(CciObject*)thisNode,traceText);
checkRC(rc);
```

Related reference:

"cciUserTrace" on page 6678
Use cciUserTrace to write a message from a message catalog (with inserts) to user trace. A message is also written to service trace, if service trace is active.

"cciServiceTrace" on page 6663
Writes a message to service trace, if service trace is active.

"cciServiceDebugTrace" on page 6660
This function is very similar to cciServiceTrace with the only difference being that the entry is written to service trace only when service trace is active at debug level.

"cciIsTraceActive" on page 6670
cciIsTraceActive reports whether trace is active and the level at which trace is active.

"Trace logging from a user-defined C extension" on page 6693
Message processing nodes and parsers that are written to the C programming language API can write entries to trace.

"cciThrowExceptionW" on page 6668
The cciThrowExceptionW exception is thrown by the broker interface and uses the specified arguments as exception data.

"cciLogW" on page 6653
cciLogW logs an error, warning, or informational event. The event is logged by the broker interface and uses the specified arguments as log data.

"cciUserTraceW" on page 6681
Use cciUserTraceW to write a message from a message catalog (with inserts) to user trace. A message is also written to service trace, if service trace is active.

"cciUserDebugTraceW" on page 6676
Use cciUserDebugTraceW to write a message from a message catalog (with inserts) to user trace when user trace is active at debug level. A message is also written to service trace, if service trace is active.

"cciServiceDebugTraceW" on page 6661
The function is very similar to cciServiceTraceW with the only difference being that the entry is written to service trace only when service trace is active at debug level.

"User trace" on page 6873
User trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Administration log. User trace is inactive by default; you must activate it explicitly by using a command, or by selecting options in the WebSphere Message Broker Toolkit.

cciThrowException:

Use this function to throw an exception. The exception is thrown by the broker interface, and includes the specified arguments as exception data.

Syntax:

```
void cciThrowException(
  int*                returnCode,
  CCI_EXCEPTION_TYPE  type,
  char*               file,
  int                 line,
  char*               function,
  CciChar*            messageSource,
  int                 messageNumber,
  char*               traceText,
                      ...);
```

Parameters:

returnCode
The return code from the function (output). The possible return code is CCI_INV_DATA_POINTER.

type The type of exception (input). Valid values are:

- CCI_FATAL_EXCEPTION
- CCI_RECOVERABLE_EXCEPTION
- CCI_CONFIGURATION_EXCEPTION
- CCI_PARSER_EXCEPTION
- CCI_CONVERSION_EXCEPTION
- CCI_DATABASE_EXCEPTION
- CCI_USER_EXCEPTION

file The source file name where the exception was generated (input). The value is optional, but it is useful for debugging purposes.

line The line number in the source file where the exception was generated (input). The value is optional, but it is useful for debugging purposes.

function
The function name which generated the exception (input). The value is optional, but it is useful for debugging purposes.

messageSource
The fully-qualified location and name of the Windows message source or the Linux, UNIX, or z/OS message catalog.

To use the current broker message catalog, specify BIPmsgs on all operating systems. Alternatively, you can create your own message catalog.

messageNumber
The message number identifying the exception (input). If *messageNumber* is specified as zero, it is assumed that a message is not available. If *messageNumber* is non-zero, the specified message is written into the broker event log with any inserts provided in the variable argument list.

traceText
Trace information that is written into the broker service trace log (input). The information is optional, but it is useful in debugging problems.

... A C variable argument list that contains any message inserts that accompany the message (input). These inserts are treated as character strings and the variable arguments are assumed to be of type `pointer to char`.

`char*` characters must be strings in either ASCII (Latin) or EBCDIC (1047).

The last argument in this list must be `(char*)0`.

Return values:
None. If an error occurs, the *returnCode* parameter indicates the reason for the error.

Related concepts:

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Creating an input node in C" on page 3027
Create a user-defined input node in C to receive messages into a message flow.

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

"Creating message catalogs" on page 3138
Create your own message catalogs to write tailored entries to the local error log.

Related reference:

"User-defined extensions" on page 6411
Reference material that supports the creation and management of your user-defined extensions.

"cciGetLastExceptionData" on page 6647
Gets diagnostic information about the last exception generated. Information about the last exception generated on the current thread is returned in a CCI_EXCEPTION_ST output structure. The user-defined extension can use this function to determine whether any recovery is required when a utility function returns an error code.

"cciLog" on page 6651
Use cciLog to write an error, warning, or informational event.

"cciRethrowLastException" on page 6659
This function re-throws the last exception that has been generated on the current thread. It is used to pass the exception back to the broker for further handling. The function, like a C **exit** call, does not return to the caller.

cciThrowExceptionW:

The cciThrowExceptionW exception is thrown by the broker interface and uses the specified arguments as exception data.

Syntax:

```
void cciThrowExceptionW(
  int*                returnCode,
  CCI_EXCEPTION_TYPE  type,
  const char*         file,
  int                 line,
  const char*         function,
  const CciChar*      messageSource,
  int                 messageNumber,
  const CciChar*      traceText,
                      ...
);
```

Parameters:

returnCode
The return code from the function (output). If the *messageSource* parameter is null, the *returnCode* is set to CCI_INV_DATA_POINTER.

type The type of exception (input). Valid values are:

- CCI_FATAL_EXCEPTION
- CCI_RECOVERABLE_EXCEPTION
- CCI_CONFIGURATION_EXCEPTION
- CCI_PARSER_EXCEPTION
- CCI_CONVERSION_EXCEPTION
- CCI_DATABASE_EXCEPTION
- CCI_USER_EXCEPTION

file The source file name where the exception was generated (input). The value is optional, but it is useful for debugging purposes.

line The line number in the source file where the exception was generated (input). The value is optional, but it is useful for debugging purposes.

function
The function name which generated the exception (input). The value is optional, but it is useful for debugging purposes.

messageSource
A string that identifies the Windows message source or the Linux and UNIX message catalog. To use the current broker message catalog, specify BIPmsgs on all operating systems.

messageNumber
The message number identifying the exception (input). If *messageNumber* is specified as zero, it is assumed that a message is not available. If *messageNumber* is non-zero, the specified message is written into the broker event log with any inserts provided in the variable argument list.

traceText
Trace information that is written into the service trace log (input). The information is optional, but it is useful in debugging problems.

... A C variable argument list that contains any message inserts that accompany the message (input). These inserts are treated as character strings and the variable arguments are assumed to be of type `pointer to CciChar`.

The last argument in this list *must* be `(Ccichar*)0`.

Return values:
None. If an error occurs, the *returnCode* parameter indicates the reason for the error.

Example:

```
void raiseExceptionWithBroker(CciChar* helpfulText,
                              char* file, /* which source file is broken */
                              int line,   /* line in above file */
                              char* func  /* function in above file */
                              ){
  int rc = CCI_SUCCESS;

  /* Set up the message catalog name */
  const char* catalog = "BIPmsgs";

  /* Convert the catalog name to wide characters.
   * BIP_DEF_COMP_CCSID is UTF-8 on distributed and LATIN1 on z/OS
   */
  int maxChars =  strlen(catalog)+1;
  CciChar* wCatalog =(CciChar*)malloc(maxChars*sizeof(CciChar));
  cciMbsToUcs(&rc, catalog, wCatalog, maxChars, BIP_DEF_COMP_CCSID);

  /* The above might have failed, but we are already throwing an exception,
   * so rc is now set to type success. */
  rc = CCI_SUCCESS;

  /* Throw the exception. The explanation will be added as the traceText and
   *  as an insert to the message
   */
  cciThrowExceptionW(&rc,
                     CCI_USER_EXCEPTION,
                     file, line, func,
                     wCatalog, BIP2111,
                     helpfulText,
                     helpfulText,
                     (CciChar*)0
                     );
  /* The above might have failed, but we are already throwing an exception,
   * so the value of rc is not important. */
}
```

Related concepts:

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Creating an input node in C" on page 3027
Create a user-defined input node in C to receive messages into a message flow.

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

"Using error logging from a user-defined extension" on page 3137
Program user-defined extensions to write entries in the local error log.

Related reference:

"User-defined extensions" on page 6411
Reference material that supports the creation and management of your user-defined extensions.

"cciGetLastExceptionDataW" on page 6649
Gets diagnostic information about the last exception generated. Information about the last exception generated on the current thread is returned in a CCI_EXCEPTION_WIDE_ST output structure. The user-defined extension uses this function to determine whether any recovery is required when a utility function returns an error code.

"cciThrowExceptionW" on page 6668
The cciThrowExceptionW exception is thrown by the broker interface and uses the specified arguments as exception data.

"cciRethrowLastException" on page 6659
This function re-throws the last exception that has been generated on the current thread. It is used to pass the exception back to the broker for further handling. The function, like a C **exit** call, does not return to the caller.

cciIsTraceActive:

cciIsTraceActive reports whether trace is active and the level at which trace is active.

Syntax:

```
CCI_TRACE_TYPE cciIsTraceActive(
  int*          returnCode,
  CciObject*    object);
```

Parameters:

returnCode
Receives the return code from the function (output). A NULL pointer input signifies that the user-defined node does not want to deal with errors. Any exceptions thrown during the execution of this call will be re-thrown to the next upstream node in the flow. If input is not NULL, output will signify the success status of the call. If an exception occurs during execution, *returnCode will be set to CCI_EXCEPTION on output. A call to **CciGetLastExceptionData** will provide details of the exception.

object
The address of the object that is to be associated with the trace entry (input).

This object can be a CciNode* or a CciParser*. If it is a CciNode*, the name of that node is written to trace. If it is a CciParser*, the name of the node that created the parser is written to trace. This object is also used to determine if the entry should be written to trace. The entry is only written if trace is active for the node. Currently nodes inherit their trace setting from the message flow.

If this parameter is NULL, the trace level for the execution group is returned.

Return values:
A CCI_TRACE_TYPE value indicating the level of trace that is currently active. The CCI_TRACE_TYPE type has the following possible values:

- CCI_USER_NORMAL_TRACE
- CCI_USER_DEBUG_TRACE
- CCI_ SERVICE_NORMAL_TRACE
- CCI_SERVICE_DEBUG_TRACE
- CCI_TRACE_NONE

These return values are bitwise values. Combinations of these values are also possible, for example:

- CCI_USER_NORMAL_TRACE + CCI_ SERVICE_NORMAL_TRACE
- CCI_USER_NORMAL_TRACE + CCI_SERVICE_DEBUG_TRACE
- CCI_USER_DEBUG_TRACE + CCI_ SERVICE_NORMAL_TRACE
- CCI_USER_DEBUG_TRACE + CCI_SERVICE_DEBUG_TRACE

CCI_TRACE_NONE is a zero value and all other values are non zero.

Two further values can be used as bitmasks when querying the active level of trace. These are:

- CCI_USER_TRACE
- CCI_SERVICE_TRACE

For example, the expression `(traceLevel & CCI_USER_TRACE)` will evaluate to a non zero value for *traceLevel* for the following return values:

- CCI_USER_NORMAL_TRACE + CCI_ SERVICE_NORMAL_TRACE
- CCI_USER_NORMAL_TRACE + CCI_SERVICE_DEBUG_TRACE
- CCI_USER_DEBUG_TRACE + CCI_ SERVICE_NORMAL_TRACE
- CCI_USER_DEBUG_TRACE + CCI_SERVICE_DEBUG_TRACE
- CCI_USER_NORMAL_TRACE
- CCI_USER_DEBUG_TRACE

The expression `(traceLevel & CCI_USER_TRACE)` will evaluate to zero for *traceLevel* for the following return values:

- CCI_SERVICE_NORMAL_TRACE
- CCI_SERVICE_DEBUG_TRACE
- CCI_TRACE_NONE

Example:

```
CciNode*         thisNode = ((NODE_CONTEXT_ST*)context)->nodeObject;

const CCI_TRACE_TYPE   traceActive = cciIsTraceActive(&rc, (CciObject*)thisNode);
checkRC(rc);
```

Related reference:

"cciUserTrace" on page 6678
Use cciUserTrace to write a message from a message catalog (with inserts) to user trace. A message is also written to service trace, if service trace is active.

"cciUserDebugTrace" on page 6673
Use cciUserDebugTrace to write a message from a message catalog (with inserts) to user trace when user trace is active at debug level.

"cciServiceTrace" on page 6663
Writes a message to service trace, if service trace is active.

"cciServiceDebugTrace" on page 6660
This function is very similar to cciServiceTrace with the only difference being that the entry is written to service trace only when service trace is active at debug level.

"Trace logging from a user-defined C extension" on page 6693
Message processing nodes and parsers that are written to the C programming language API can write entries to trace.

"User trace" on page 6873
User trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Administration log. User trace is inactive by default; you must activate it explicitly by using a command, or by selecting options in the WebSphere Message Broker Toolkit.

cciUcsToMbs:

Use this function to convert Universal Character Set (UCS) data to multibyte string data. This function is, typically, used only for formatting diagnostic messages. Normal processing is best done in UCS-2, which can represent all characters from all languages.

The sample code in `BipSampPluginUtil.c` contains more utilities for processing UCS-2 characters in a portable way.

Syntax:

```
int cciUcsToMbs(
  int*           returnCode,
  const CciChar* ucsString,
  char*          mbString,
  int            mbStringLength,
  int            codePage);
```

Parameters:

returnCode
The return code from the function (output).

Possible return codes are:

- CCI_SUCCESS
- CCI_BUFFER_TOO_SMALL
- CCI_INV_CHARACTER
- CCI_FAILURE
- CCI_INV_CODEPAGE

ucsString
The string to be converted, expressed as UCS-2 Unicode (input).

mbString
The location of the resulting string (input). The string has a trailing byte of 0, just as the Unicode has a trailing `CciChar` of 0.

mbStringLength
The length (in bytes) of the buffer that you have provided (input). Each `CciChar` in the source string expands to one byte (for SBCS code pages), or up to not more than the code page's MB_CUR_MAX value (typically less than five bytes), which defines an upper limit of the buffer size required.

codePage
The code page of the resulting string (input). The value of the code page must be suitable for the compiler that you are using to compile the user-defined node.

For an ASCII system, a value of 1208 (meaning code page ibm-1208, which is UTF-8 Unicode) is a good choice if you are using cciUcsToMbs to convert string constants for processing by WebSphere Message Broker. 1208 is appropriate for Linux, UNIX, and Windows.

On Linux and UNIX systems, `nl_langinfo(CODEPAGE)` gives you the code page that has been selected by the **setlocale** command.

On z/OS, the default code page for WebSphere MQ, which is 500, must not be used. Instead, use a code page value of 1047.

Return values:
The *returnCode* parameter is set to the converted length in bytes.

Related concepts:

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Creating an input node in C" on page 3027
Create a user-defined input node in C to receive messages into a message flow.

"Creating a message processing or output node in C" on page 3036
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"User-defined extensions" on page 6411
Reference material that supports the creation and management of your user-defined extensions.

"cciMbsToUcs" on page 6655
Use this function to convert multibyte string data to Universal Character Set (UCS).

"Supported code pages" on page 4176
Application messages must conform to supported code pages.

cciUserDebugTrace:

Use cciUserDebugTrace to write a message from a message catalog (with inserts) to user trace when user trace is active at debug level.

If user trace is not active at debug level, an entry is written to service trace when service trace is active.

Syntax:

```
void cciUserDebugTrace(
  int*           returnCode,
  CciObject*     object,
  const CciChar* messageSource,
  int            messageNumber,
  const char*    traceText,
  ...
);
```

Parameters:

returnCode
Receives the return code from the function (output). A NULL pointer input parameter indicates that the user-defined node does not handle errors. All exceptions that are thrown during the execution of this call are thrown again to the next upstream node in the flow. If the input parameter is not NULL, the output signifies the success status of the call. If an exception occurs during execution, **returnCode* is set to CCI_EXCEPTION on output. Call CciGetLastExceptionData to obtain details of the exception.

object
The address of the object that is to be associated with the trace entry (input). This object can be a CciNode* or a CciParser*. If you specify a CciNode*, the name of that node is written to trace. If you specify a CciParser*, the name of the node that created the parser is written to trace. This object is also used to determine if the entry is written to trace. The entry is written only if trace is active for the node. Nodes inherit their trace setting from the message flow.

If this parameter is NULL, the trace level for the execution group is returned.

messageSource
The fully qualified location and name of the Windows message source or the Linux, UNIX, or z/OS message catalog.

To use the current broker message catalog, specify BIPmsgs on all operating systems. Alternatively, you can create your own message catalog.

When trace is formatted, a message from the NLS version of this catalog is written.

The locale used is that of the environment where the trace is formatted. You can run the broker on one operating system, read the log on that operating system, then format the log on a different operating system. For example, if the broker is running on Linux, UNIX, or z/OS but no `.cat` file is available, you could read the log, then transfer it to Windows where the log can be formatted by using the `.properties` file.

If this parameter is NULL, the effect is the same as specifying an empty string. That is, all other information is written to the log, and the catalog field has an empty string value. Therefore, the log formatter cannot find the message source and fails to format this entry.

messageNumber
The number that identifies the message within the specified *messageSource* (input). If the *messageSource* does not contain a message that corresponds to this *messageNumber*, the log formatter fails to format this entry.

traceText
A string of characters that ends with NULL (input). This string is written to service trace and provides an easy way to correlate trace entries with paths through the source code. For example, you might have several paths through the code that result in the same message (*messageSource* and *messageNumber*) being written to trace. *traceText* can be used to distinguish between these different paths. That is, the *traceText* string is a static literal string in the source and therefore the same string is in both the source code file and the formatted trace file.

...
A C variable argument list that contains other message inserts that accompany the message (input). These inserts are treated as character strings and the variable arguments are assumed to be of type `pointer to char`. The last argument in this list *must* be `(char*)0`.

- For user-defined extensions that are running on distributed platforms, the `char*` arguments must be in ISO-8859-1 (ibm-918) code page.
- For user-defined extensions that are running on z/OS platforms, the `char*` arguments must be in EBCIDIC (1047).

These requirements apply to all char* arguments in **traceText** and the variable argument list of inserts (**...**).

Return values:
None. If an error occurs, the *returnCode* parameter indicates the reason for the error.

Example:

```
const CciChar*   myMessageSource=CciString("SwitchMSG",BIP_DEF_COMP_CCSID);
CciNode*         thisNode = ((NODE_CONTEXT_ST*)context)->nodeObject;

const char* mbElementName  = mbString((CciChar*)&elementName,BIP_DEF_COMP_CCSID);
const char* mbElementValue = mbString((CciChar*)&elementValue,BIP_DEF_COMP_CCSID);
const char* traceTextFormat = "Switch Element: name=%s, value=%s";
char* traceText = (char*)malloc(strlen(traceTextFormat) +
                                strlen(mbElementName) +
                                strlen(mbElementValue));
sprintf(traceText,traceTextFormat,mbElementName,mbElementValue);

cciUserDebugTrace(&rc,
                  (CciObject*)thisNode,
                  myMessageSource,
                  2,
                  traceText,
                  mbElementName,
                  mbElementValue,
                  (char*)0);
free((void*)mbElementName);
free((void*)mbElementValue);
free((void*)traceText);
```

Related tasks:

"Creating message catalogs" on page 3138
Create your own message catalogs to write tailored entries to the local error log.

Related reference:

"Trace logging from a user-defined C extension" on page 6693
Message processing nodes and parsers that are written to the C programming language API can write entries to trace.

"cciUserTrace" on page 6678
Use cciUserTrace to write a message from a message catalog (with inserts) to user

trace. A message is also written to service trace, if service trace is active.

"cciServiceTrace" on page 6663
Writes a message to service trace, if service trace is active.

"cciServiceDebugTrace" on page 6660
This function is very similar to cciServiceTrace with the only difference being that the entry is written to service trace only when service trace is active at debug level.

"cciIsTraceActive" on page 6670
cciIsTraceActive reports whether trace is active and the level at which trace is active.

"User trace" on page 6873
User trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Administration log. User trace is inactive by default; you must activate it explicitly by using a command, or by selecting options in the WebSphere Message Broker Toolkit.

cciUserDebugTraceW:

Use cciUserDebugTraceW to write a message from a message catalog (with inserts) to user trace when user trace is active at debug level. A message is also written to service trace, if service trace is active.

If user trace is not active at debug level, an entry is written to service trace when service trace is active at any level.

Syntax:

```
void cciUserDebugTraceW(
  int*             returnCode,
  CciObject*       object,
  const CciChar*   messageSource,
  int              messageNumber,
  const CciChar*   traceText,
  ...
);
```

Parameters:

returnCode
Receives the return code from the function (output). A NULL pointer input signifies that the user-defined node does not want to deal with errors. Any exceptions thrown during the execution of this call will be re-thrown to the next upstream node in the flow. If input is not NULL, output will signify the success status of the call. If an exception occurs during execution, *returnCode will be set to CCI_EXCEPTION on output. A call to **CciGetLastExceptionData** will provide details of the exception.

object
The address of the object that is to be associated with the trace entry (input). This object can be a CciNode* or a CciParser*. If it is a CciNode*, the name of that node is written to trace. If it is a CciParser*, the name of the node that created the parser is written to trace. This object is also used to determine if the entry should be written to trace. The entry is only written if trace is active for the node. Currently nodes inherit their trace setting from the message flow.

If this parameter is NULL, the trace level for the execution group is returned.

messageSource
A string that identifies the Windows message source or the Linux and UNIX message catalog (input). When trace is formatted, a message from the NLS version of this catalog is written. The locale used is that of the environment where the trace is formatted. It is possible to run the broker on one type of operating system, read the log on that operating system, then format the log on a different operating system. For example, if the broker is running on Linux or UNIX but no .cat file is available, the user could read the log, then transfer it to Windows where the log can be formatted by using the .properties file.

If this parameter is NULL, the effect is the same as specifying an empty string. That is, all other information will be written to the log, and the catalog field will have an empty string value. Therefore, the log formatter will not be able to find the message source. Consequently, the log formatter will fail to format this entry.

messageNumber
The number that identifies the message within the specified *messageSource* (input). If the *messageSource* does not contain a message that corresponds to this *messageNumber*, the log formatter fails to format this entry.

traceText
A string of characters that ends with NULL (input). This string will be written to service trace and provides an easy way to correlate trace entries with paths through the source code. For example, there could be several paths through the code that result in the same message (*messageSource* and *messageNumber*) being written to trace. *traceText* can be used to distinguish between these different paths. That is, the *traceText* string will be a static literal string in the source and therefore the same string will be in both the source code file and the formatted trace file.

...
A C variable argument list that contains any message inserts that accompany the message (input). These inserts are treated as character strings and the variable arguments are assumed to be of type `pointer to CciChar`.

The last argument in this list *must* be `(CciChar*)0`.

Return values:
None. If an error occurs, the *returnCode* parameter indicates the reason for the error.

Example:

```
const CciChar*   myMessageSource=CciString("SwitchMSG",BIP_DEF_COMP_CCSID);
CciNode*         thisNode = ((NODE_CONTEXT_ST*)context)->nodeObject;
const CciChar*   traceText = CciString("Found an element name and value",
                                       BIP_DEF_COMP_CCSID);

cciUserDebugTraceW(&rc,
                   (CciObject*)thisNode,
                   myMessageSource,
                   2,
                   traceText,
                   elementName,
                   elementValue,
                   (CciChar*)0);
```

Related reference:

"cciUserTrace" on page 6678
Use cciUserTrace to write a message from a message catalog (with inserts) to user

trace. A message is also written to service trace, if service trace is active.

"cciServiceTrace" on page 6663
Writes a message to service trace, if service trace is active.

"cciServiceDebugTrace" on page 6660
This function is very similar to cciServiceTrace with the only difference being that the entry is written to service trace only when service trace is active at debug level.

"cciIsTraceActive" on page 6670
cciIsTraceActive reports whether trace is active and the level at which trace is active.

"Trace logging from a user-defined C extension" on page 6693
Message processing nodes and parsers that are written to the C programming language API can write entries to trace.

"cciThrowExceptionW" on page 6668
The cciThrowExceptionW exception is thrown by the broker interface and uses the specified arguments as exception data.

"cciLogW" on page 6653
cciLogW logs an error, warning, or informational event. The event is logged by the broker interface and uses the specified arguments as log data.

"cciUserTraceW" on page 6681
Use cciUserTraceW to write a message from a message catalog (with inserts) to user trace. A message is also written to service trace, if service trace is active.

"cciServiceTraceW" on page 6664
cciServiceTraceW writes a message to service trace, if service trace is active.

"cciServiceDebugTraceW" on page 6661
The function is very similar to cciServiceTraceW with the only difference being that the entry is written to service trace only when service trace is active at debug level.

"User trace" on page 6873
User trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Administration log. User trace is inactive by default; you must activate it explicitly by using a command, or by selecting options in the WebSphere Message Broker Toolkit.

cciUserTrace:

Use cciUserTrace to write a message from a message catalog (with inserts) to user trace. A message is also written to service trace, if service trace is active.

The message written to user trace has the following format:

```
<date-time stamp> <threadNumber> UserTrace <Message text with inserts> <Message Explanation>
```

Syntax:

```
void cciUserTrace(
  int*              returnCode,
  CciObject*        object,
  const CciChar*    messageSource,
  int               messageNumber,
  const char*       traceText,
                    ...
);
```

Parameters:

returnCode
Receives the return code from the function (output). A NULL pointer input signifies that the user-defined node does not want to deal with errors. Any exceptions thrown during the execution of this call are re-thrown to the next upstream node in the flow. If input is not NULL, output signifies the success status of the call. If an exception occurs during execution, *returnCode is set to CCI_EXCEPTION on output. A call to CciGetLastExceptionData provides details of the exception.

object
The address of the object that is to be associated with the trace entry (input). This object can be a CciNode* or a CciParser*. If it is a CciNode*, the name of that node is written to trace. If it is a CciParser*, the name of the node that created the parser is written to trace. This object is also used to determine if the entry should be written to trace. The entry is only written if trace is active for the node. Currently, nodes inherit their trace setting from the message flow.

If this parameter is NULL, the trace level for the execution group is returned.

messageSource
The fully-qualified location and name of the Windows message source, or the Linux, UNIX, or z/OS message catalog (input).

To use the current broker message catalog, specify BIPmsgs on all operating systems. Alternatively, you can create your own message catalog.

When trace is formatted, a message from the NLS version of this catalog is written. The locale used is that of the environment where the trace is formatted.

You can run the broker on one operating system, read the log on that operating system, then format it on a different operating system. For example, if the broker is running on Linux, but no `.cat` file is available, you could read the log, then transfer it to Windows, where the log can be formatted by using the `.properties` file.

If this parameter is NULL, the effect is the same as specifying an empty string. That is, all other information is written to the log, and the catalog field has an empty string value. Therefore, the log formatter cannot find the message source. Consequently, the log formatter fails to format this entry.

messageNumber
The number that identifies the message within the specified *messageSource* (input). If the *messageSource* does not contain a message that corresponds to this *messageNumber*, the log formatter fails to format this entry.

traceText
A string of characters that ends with NULL (input). This string is written to service trace, and provides an easy way to correlate trace entries with paths through the source code. For example, if several paths through the code result in the same message (*messageSource* and *messageNumber*) being written to trace, you can specify *traceText* to distinguish between these different paths. That is, the *traceText* string is a static literal string in the source, and therefore the same string appears in both the source code file and the formatted trace file.

...
A C variable argument list that contains any message inserts that accompany the message (input). These inserts are treated as character strings, and the variable arguments are assumed to be of type `pointer to char`.

The last argument in this list must be `(char*)0`.

- For user-defined extensions that are running on distributed platforms, the `char*` arguments must be in ISO-8859-1 (ibm-918) code page.
- For user-defined extensions that are running on z/OS platforms, the `char*` arguments must be in EBCIDIC (1047).

These requirements include all char* arguments in **traceText** and the variable argument list of inserts (**...**).

Return values:
None. If an error occurs, the *returnCode* parameter indicates the reason for the error.

Example:

```
const CciChar*   myMessageSource=CciString("SwitchMSG",BIP_DEF_COMP_CCSID);
CciNode*          thisNode = ((NODE_CONTEXT_ST*)context)->nodeObject;

cciUserTrace(&rc,
             (CciObject*)thisNode,
              myMessageSource,
              1,
              "propagating to add terminal",
              "add",
              (char*)0);
 checkRC(rc);
```

Related tasks:

"Using error logging from a user-defined extension" on page 3137
Program user-defined extensions to write entries in the local error log.

"Creating message catalogs" on page 3138
Create your own message catalogs to write tailored entries to the local error log.

Related reference:

"Trace logging from a user-defined C extension" on page 6693
Message processing nodes and parsers that are written to the C programming language API can write entries to trace.

"cciUserDebugTrace" on page 6673
Use cciUserDebugTrace to write a message from a message catalog (with inserts) to user trace when user trace is active at debug level.

"cciServiceTrace" on page 6663
Writes a message to service trace, if service trace is active.

"cciServiceDebugTrace" on page 6660
This function is very similar to cciServiceTrace with the only difference being that the entry is written to service trace only when service trace is active at debug level.

"cciIsTraceActive" on page 6670
cciIsTraceActive reports whether trace is active and the level at which trace is active.

"User trace" on page 6873
User trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Administration log. User trace is inactive by default; you must activate it explicitly by using a command, or by selecting options in the WebSphere Message Broker Toolkit.

"Service trace" on page 6875
Service trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Event Log or to user trace. Service trace is inactive by

default; you must activate it explicitly by using a command.

cciUserTraceW:

Use cciUserTraceW to write a message from a message catalog (with inserts) to user trace. A message is also written to service trace, if service trace is active.

The message written to user trace has the following format:

```
<date-time stamp> <threadNumber> UserTrace <Message text with inserts> <Message Explanation>
```

Syntax:

```
void cciUserTraceW(
  int*                returnCode,
  CciObject*          object,
  const CciChar*      messageSource,
  int                 messageNumber,
  const CciChar*      traceText,
                      ...
);
```

Parameters:

returnCode
Receives the return code from the function (output). A NULL pointer input signifies that the user-defined node does not want to deal with errors. Any exceptions that are thrown during the execution of this call are re-thrown to the next upstream node in the flow. If input is not NULL, output indicates the success status of the call. If an exception occurs during execution, *returnCode is set to CCI_EXCEPTION on output. Call CciGetLastExceptionData to obtain details of the exception.

object
The address of the object that is to be associated with the trace entry (input). This object can be a CciNode* or a CciParser*. If you specify a CciNode*, the name of that node is written to trace. If you specify a CciParser*, the name of the node that created the parser is written to trace. This object is also used to determine if the entry should be written to trace. The entry is written only if trace is active for the node. Nodes inherit their trace setting from the message flow.

If this parameter is NULL, the trace level for the execution group is returned.

messageSource
The fully-qualified location and name of the Windows message source or the Linux, UNIX, or z/OS message catalog.

To use the current broker message catalog, specify BIPmsgs on all operating systems. Alternatively, you can create your own message catalog.

When trace is formatted, a message from the NLS version of this catalog is written.

The locale used is that of the environment where the trace is formatted. You can run the broker on one operating system, read the log on that operating system, then format the log on a different operating system. For example, if the broker is running on Linux, UNIX, or z/OS but no `.cat` file is available, you could read the log, then transfer it to Windows where the log can be formatted by using the `.properties` file.

If this parameter is NULL, the effect is the same as specifying an empty string. That is, all other information is written to the log, and the catalog field has an empty string value. Therefore, the log formatter cannot find the message source and fails to format this entry.

messageNumber
The number that identifies the message within the specified *messageSource* (input). If the *messageSource* does not contain a message that corresponds to this *messageNumber*, the log formatter fails to format this entry.

traceText
A string of characters that ends with NULL (input). This string is written to service trace and provides an easy way to correlate trace entries with paths through the source code. For example, there could be several paths through the code that result in the same message (*messageSource* and *messageNumber*) being written to trace. Use *traceText* to distinguish between these different paths. That is, the *traceText* string is a static literal string in the source, and therefore the same string is in both the source code file and the formatted trace file.

...
A C variable argument list that contains any message inserts that accompany the message (input). These inserts are treated as character strings and the variable arguments are assumed to be of type `pointer to CciChar`.

The last argument in this list must be `(CciChar*)0`.

Return values:
None. If an error occurs, the *returnCode* parameter indicates the reason for the error.

Example:

```
const CciChar*   myMessageSource=CciString("SwitchMSG",BIP_DEF_COMP_CCSID);
const CciChar*   text = CciString("propagating to add terminal",
                                  BIP_DEF_COMP_CCSID);
const CciChar*   insert = CciString("add", BIP_DEF_COMP_CCSID);
CciNode*         thisNode = ((NODE_CONTEXT_ST*)context)->nodeObject;
int              rc = CCI_SUCCESS;

cciUserTrace(&rc,
             (CciObject*)thisNode,
             myMessageSource,
             1,
             text,
             insert,
             (CciChar*)0);
checkRC(rc);
```

Related tasks:

"Creating message catalogs" on page 3138
Create your own message catalogs to write tailored entries to the local error log.

Related reference:

"Trace logging from a user-defined C extension" on page 6693
Message processing nodes and parsers that are written to the C programming language API can write entries to trace.

"cciUserTrace" on page 6678
Use cciUserTrace to write a message from a message catalog (with inserts) to user trace. A message is also written to service trace, if service trace is active.

"cciServiceTrace" on page 6663
Writes a message to service trace, if service trace is active.

"cciServiceDebugTrace" on page 6660
This function is very similar to cciServiceTrace with the only difference being that the entry is written to service trace only when service trace is active at debug level.

"cciIsTraceActive" on page 6670
cciIsTraceActive reports whether trace is active and the level at which trace is active.

"cciThrowExceptionW" on page 6668
The cciThrowExceptionW exception is thrown by the broker interface and uses the specified arguments as exception data.

"cciLogW" on page 6653
cciLogW logs an error, warning, or informational event. The event is logged by the broker interface and uses the specified arguments as log data.

"cciUserDebugTraceW" on page 6676
Use cciUserDebugTraceW to write a message from a message catalog (with inserts) to user trace when user trace is active at debug level. A message is also written to service trace, if service trace is active.

"cciServiceTraceW" on page 6664
cciServiceTraceW writes a message to service trace, if service trace is active.

"cciServiceDebugTraceW" on page 6661
The function is very similar to cciServiceTraceW with the only difference being that the entry is written to service trace only when service trace is active at debug level.

"User trace" on page 6873
User trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Administration log. User trace is inactive by default; you must activate it explicitly by using a command, or by selecting options in the WebSphere Message Broker Toolkit.

C skeleton code

Use the skeleton code that is supplied as guidance for your C user-defined node. The code has the minimum content that is required to compile a user-defined node successfully.

```
#ifdef __WIN32
#include <windows.h>
#endif
#include <BipCos.h>
#include <BipCci.h>
#include <BipCni.h>
#include <malloc.h>

#define BIP_DEF_COMP_CCSID 437
CciChar* constNodeFactory = 0;
CciChar* constNodeName    = 0;
CciChar* constTerminalName    = 0;
CciChar* constOutTerminalName    = 0;

CciChar* CciString(
  const char* source,
  int         codepage
){
  /* Maximum number of characters in Unicode representation */
  int maxChars = strlen(source) + 1 ;
  CciChar* buffer = (CciChar*)malloc(maxChars * sizeof(CciChar)) ;
  int rc ;
  cciMbsToUcs(&rc, source, buffer, maxChars, codepage) ;
  return buffer ;
```

```
}
void initNodeConstants(){
  constNodeFactory       = CciString("myNodeFactory", BIP_DEF_COMP_CCSID);
  constNodeName          = CciString("myNode",BIP_DEF_COMP_CCSID);
  constTerminalName      = CciString("in",BIP_DEF_COMP_CCSID);
  constOutTerminalName   = CciString("out",BIP_DEF_COMP_CCSID);
}

typedef struct {
  CciTerminal* iOutTerminal;
}MyNodeContext;

CciContext* createNodeContext(
  CciFactory* factoryObject,
  CciChar*    nodeName,
  CciNode*    nodeObject
){

  MyNodeContext * p = (MyNodeContext *)malloc(sizeof(MyNodeContext));

  /*here we would create an instance of some data structure
  where we could store context about this node instance.
  We would return a pointer to this struct and that pointer
  will be passed to our other implementation functions */

  /* now we create an input terminal for the node*/
  cniCreateInputTerminal(NULL, nodeObject, (CciChar*)constTerminalName);
  p->iOutTerminal = cniCreateOutputTerminal(NULL, nodeObject, (CciChar*)constOutTerminalName);
  return((CciContext*)p);
}

/*****************************************************************/
/*                                                               */
/* Plugin Node Implementation Function:           cniEvaluate() */
/*                                                               */
/*****************************************************************/
void evaluate(
  CciContext* context,
  CciMessage* localEnvironment,
  CciMessage* exceptionList,
  CciMessage* message
){
  /* we would place our node's processing logic in here*/
  return;
}

int run(
  CciContext* context,
  CciMessage* localEnvironment,
  CciMessage* exceptionList,
  CciMessage* message
)
{
  char* buffer="<doc><test>hello</test></doc>";
  CciChar* wBuffer=CciString(buffer,BIP_DEF_COMP_CCSID);
  //cniSetInputBuffer(NULL,message,(void*)wBuffer,strlen(buffer) * sizeof(CciChar));
  cniSetInputBuffer(NULL,message,(void*)buffer,strlen(buffer));
  cniFinalize(NULL,message,0);

  cniPropagate(NULL,((MyNodeContext*)context)->iOutTerminal,localEnvironment,exceptionList,message);
  return CCI_SUCCESS_CONTINUE;
}



#ifdef __cplusplus
extern "C"{
```

```
#endif
CciFactory LilFactoryExportPrefix * LilFactoryExportSuffix bipGetMessageflowNodeFactory()
{
  CciFactory*      factoryObject;

  /* Before we proceed, we need to initialize all the static constants */
  /* that might be used by the plug-in.                                */
  initNodeConstants();

  /* Create the Node Factory for this plug-in */
  /* If any errors/exceptions    */
  /* occur during the execution of this utility function, then as we have not */
  /* supplied the returnCode argument, the exception will bypass the plugin   */
  /* and be directly handled by the broker.                                    */
  factoryObject = cniCreateNodeFactory(0, (unsigned short *)constNodeFactory);
  if (factoryObject == CCI_NULL_ADDR) {
    /* Any further local error handling can go here */
  }
  else {
    /* Define the node supported by this factory */
    static CNI_VFT vftable = {CNI_VFT_DEFAULT};
    /* Setup function table with pointers to node implementation functions */
    vftable.iFpCreateNodeContext = createNodeContext;
    vftable.iFpEvaluate          = evaluate;
    vftable.iFpRun               = run;

    /* Define a node type supported by our factory. If any errors/exceptions    */
    /* occur during the execution of this utility function, then as we have not */
    /* supplied the returnCode argument, the exception will bypass the plugin   */
    /* and be directly handled by the broker.                                    */
    cniDefineNodeClass(NULL, factoryObject, (CciChar*)constNodeName, &vftable);

  }

  /* Return address of this factory object to the broker */
  return(factoryObject);
}
#ifdef __cplusplus
}
#endif
```

GNU makefile

The following file is a makefile that lists the files, dependencies, and rules by which the C user-defined node is compiled.

```
.SUFFIXES : .so .a .o .c

R1INC  = .
R1LIB  = .

# WMQI
MQSIDIR  = /cmvc/back/inst.images/x86_linux_2/shipdata/opt/mqsi
MQSIINC  = $(MQSIDIR)/include
MQSILIB  = $(MQSIDIR)/lib

# WMQ
MQIDIR  = /usr/mqm

CC   = /usr/bin/g++
LD   = ${CC}


OBJ = .o
LIL = .lil
THINGSTOCLEAN = *${OBJ}
CFLAGS  = -fpic -c #-pedantic -x c -Wall
```

```
CFLAGSADD = -I${R1INC} -I${MQSIINC} -I${MQSIINC}/plugin ${DEFINES}
DEFINES  = -DLINUX

LIBADD  = -L${MQSILIB} -limbdfplg
LDFLAG  = -shared ${LIBADD}

#CC   = /usr/bin/gcc
#LD   = ${CC}

OBJECTS = skeleton${OBJ}
.c.o : ; ${CC} ${CFLAGS} ${CFLAGSADD} $<

ALL : ${OBJECTS} Samples${LIL}
clean:
 rm *${OBJ} *${LIL}


skeleton${OBJ}: skeleton.c


Samples${LIL}: ${OBJECTS}
 ${LD} -o $@ ${OBJECTS} ${LDFLAG}
```

Related tasks:

"Creating a user-defined extension in C" on page 3026
You must complete a series of tasks to create user-defined extensions that use the C language.

Related reference:

"C language user-defined parser API" on page 6538
The C language user-defined parser API consists of two complementary sets of functions that you can use to define the function of your parser.

"C language user-defined node API" on page 6416
Learn about the different types of call provided by the API.

"C common implementation functions" on page 6641
You can use the common implementation functions in user-defined nodes and user-defined parsers. All the functions are called by the broker on occurrence of certain events.

"C common utility functions" on page 6643
WebSphere Message Broker provides some additional utilities that user-defined nodes and parsers can use.

Property editor API

The property editor Java API is described in this section. Use the API classes and methods described here to develop property editor applications.

The Java classes and methods are described in the Property editor API.

Related tasks:

"Adding a property editor or compiler" on page 3091
Create a property editor by using the IPropertyEditor interface to control how the properties of your user-defined node created in Java or C only, are displayed in the WebSphere Message Broker Toolkit. Create a custom compiler by using the IPropertyCompiler interface; for example, to encrypt a value before sending it to the server.

Utility function return codes and values

By convention, the return code output parameter of all utility functions is set to indicate successful completion, or an error. The table lists all return codes with their meanings.

These return codes are defined in the `BipCci.h` header file.

Return code	Explanation
CCI_BUFFER_TOO_SMALL	The output buffer is not large enough to store the requested data.
CCI_EXCEPTION	An exception occurred.
CCI_EXCEPTION_CONFIGURATION	A configuration exception was detected when invoking the function. [1]
CCI_EXCEPTION_CONVERSION	A conversion exception was detected when invoking the function. [1]
CCI_EXCEPTION_DATABASE	A database exception was detected when invoking the function.
CCI_EXCEPTION_FATAL	A fatal exception was detected when invoking the function. [1]
CCI_EXCEPTION_PARSER	A parser exception was detected when invoking the function. [1]
CCI_EXCEPTION_RECOVERABLE	A recoverable exception was detected when invoking the function. [1]
CCI_EXCEPTION_UNKNOWN	An unknown exception was specified or encountered.
CCI_EXCEPTION_USER	A user exception was detected when invoking the function. [1]
CCI_FAILURE	A function was unsuccessful.
CCI_FAILURE_CONTINUE	cniRun() return value: rollback message processing and continue thread execution
CCI_FAILURE_RETURN	cniRun() return value: rollback message processing and return thread to pool
CCI_INV_CODEPAGE	An invalid code page number was specified.
CCI_INV_CHARACTER	An invalid character was detected in the buffer to be converted.
CCI_INV_DATA_BUFLEN	A data buffer length of zero was specified.
CCI_INV_DATA_POINTER	A null pointer was specified for the address of an output data area.
CCI_INV_ELEMENT_OBJECT	A null pointer was specified for the element object.
CCI_INV_FACTORY_NAME	A factory name that is not valid (blank) was specified.
CCI_INV_FACTORY_OBJECT	A null pointer was specified for the factory object.
CCI_INV_IMPL_FUNCTION	An invalid combination of conditional implementation functions was specified
CCI_INV_LENGTH	A length of zero was specified.
CCI_INV_LOG_TYPE	The specified log type is not valid.
CCI_INV_MESSAGE_CONTEXT	A null pointer was specified for the message context.
CCI_INV_MESSAGE_OBJECT	A null pointer was specified for the message object.
CCI_INV_NODE_ENV	Attempt to dispatch a thread from a non-input node.

Return code	Explanation
CCI_INV_NODE_NAME	A node name that is not valid (blank) was specified.
CCI_INV_NODE_OBJECT	A null pointer was specified for the node object.
CCI_INV_OBJECT_NAME	Characters specified in the object name were not valid.
CCI_INV_PARSER_NAME	A parser class name that is not valid (blank) was specified.
CCI_INV_PARSER_OBJECT	A null pointer was specified for the parser object.
CCI_INV_SQL_EXPR_OBJECT	A null pointer was specified for an SQL expression value.
CCI_INV_STATEMENT	A statement was not specified.
CCI_INV_TERMINAL_NAME	A terminal name that is not valid (blank) was specified.
CCI_INV_TERMINAL_OBJECT	A null pointer was specified for the terminal object.
CCI_INV_TRANSACTION_TYPE	An invalid value was specified for the transaction type.
CCI_INV_VFTP	A null pointer was specified for the address of the user-defined extension virtual function pointer table.
CCI_MISSING_IMPL_FUNCTION	A mandatory implementation function was not defined in the function pointer table.
CCI_NAME_EXISTS	A parser with the same class name already exists.
CCI_NO_BUFFER_EXISTS	No buffer exists for the specified parser object.
CCI_NO_EXCEPTION_EXISTS	No previous exception was found for this thread.
CCI_NO_THREADS_AVAILABLE	No threads were available to be dispatched.
CCI_NULL_ADDR	A function that should return an address was unsuccessful; zero is returned instead.
CCI_PARSER_NAME_TOO_LONG	The name of the parser class is too long.
CCI_SUCCESS	Successful completion.
CCI_SUCCESS_CONTINUE	cniRun() return value: commit message processing and continue thread execution
CCI_SUCCESS_RETURN	cniRun() return value: commit message processing and return thread to pool
CCI_TIMEOUT	cniRun() return value: no message processing but continue thread execution

Note:

1. This return code is returned only by cniGetLastExceptionData to indicate the type of the last exception.

Related reference:

"C common utility functions" on page 6643
WebSphere Message Broker provides some additional utilities that user-defined nodes and parsers can use.

"C Header files" on page 6415
The C interfaces are defined by the following header files.

Available parsers

A parser is called by the broker only when that parser is required. The parser that is called depends upon the parser that has been specified.

For certain implementation functions, it might be necessary to specify the name of a parser supplied with WebSphere Message Broker. For example, functions include:

- cniCreateElementAfterUsingParser
- cniCreateElementAsFirsthChildUsingParser
- cniCreateElementAsLastChildUsingParser
- cniCreateElementAsLastChildFromBitstream
- cniCreateElementBeforeUsingParser

When using these functions, you must specify the correct class name of the parser. The following tables provide a summary of the parsers, root element names, and class names for different headers.

The following table shows the **Body** parsers.

Parser	Root element name	Class name
BLOB	BLOB	NONE
DataObject	DataObject	DataObject
IDOC (deprecated)	IDOC	IDOC
JMSMap	JMSMap	JMS_MAP
JMSStream	JMSStream	JMS_STREAM
MIME	MIME	MIME
MRM	MRM	MRM
SOAP	SOAP	SOAP
XML (deprecated)	XML	xml
XMLNS	XMLNS	xmlns
XMLNSC	XMLNSC	xmlnsC
JSON[1]	JSON	JSON

Note:

1. To enable the JSON parser on a broker, use the **-f** parameter on the **mqsichangebroker** command. For more information, see "**mqsichangebroker** command" on page 3723.

The following table shows the **Header** parsers.

Parser	Root element name	Class name
EmailOutputHeader	EmailOutputHeader	EMAILHDR
HTTPInputHeader	HTTPInputHeader	WSINPHDR

Parser	Root element name	Class name
HTTPReplyHeader	HTTPReplyHeader	WSREPHDR
HTTPRequestHeader	HTTPRequestHeader	WSREQHDR
HTTPResponseHeader	HTTPResponseHeader	WSRSPHDR
JMS header	JMSTransport	jms_transport
MQCFH	MQPCF	MQPCF
MQCIH	MQCIH	MQCICS
MQDLH	MQDLH	MQDEAD
MQIIH	MQIIH	MQIMS
MQMD	MQMD	MQHMD
MQMDE	MQMDE	MQHMDE
MQRFH	MQRFH	MQHRF
MQRFH2	MQRFH2	MQHRF2
MQRFH2C	MQRFH2C	MQHRF2C
MQRMH	MQRMH	MQHREF
MQSAPH	MQSAPH	MQHSAP
MQWIH	MQWIH	MQHWIH
SMQ_BMH	SMQ_BMH	SMQBAD

When using the MQMD parser, the MQMD is assumed to be a V2 MQMD.

The following table shows the **Properties** parser.

Parser	Root element name	Class name
Properties	Properties	PropertyParser

You can also create your own user-defined parsers, or you can make use of user-defined parsers that have been supplied by independent software vendors.

Related concepts:

"User-defined parsers" on page 3010
A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

"User-defined parser life cycle" on page 3011
Various stages exist in the life of a user-defined message flow parser.

"Planning user-defined parsers" on page 3013
Read about the concepts that you should consider before you develop a user-defined parser.

"Parsers" on page 1072
A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from the internal message tree representation.

Related reference:

"XML, MRM, and XMLNSC parser constants" on page 6691
The names of the XML and MRM parser constants, together with their

corresponding values, and a link to the XMLNSC constants.

"C language user-defined parser API" on page 6538
The C language user-defined parser API consists of two complementary sets of functions that you can use to define the function of your parser.

XML, MRM, and XMLNSC parser constants

The names of the XML and MRM parser constants, together with their corresponding values, and a link to the XMLNSC constants.

When you are writing user-defined extensions you might need to know the value of various constants.

XML parser constants:

Name	Value
Element	0x01000000
tag	0x01000000
ParserRoot	0x01000010
Content	0x02000000
pcdata	0x02000000
attr	0x03000000
Attribute	0x03000000
UnparsedEntityDecl	0x05000004
NotationDecl	0x05000008
EntityDecl	0x05000011
ParameterEntityDecl	0x05000012
ExternalEntityDecl	0x05000014
XmlDecl	0x05000018
DocTypeDecl	0x05000020
IntSubset	0x05000021
ExtSubset	0x05000022
AttributeList	0x05000024
AttributeDef	0x05000028
ExternalParameterEntityDecl	0x05000040
WhiteSpace	0x06000002
PublicId	0x06000004
SystemId	0x06000008
NotationReference	0x06000010
Version	0x06000011
Encoding	0x06000012
Standalone	0x06000014
Comment	0x06000018
EntityReferenceStart	0x06000020
EntityReferenceEnd	0x06000021
DocTypeComment	0x06000022

Name	Value
AsisElementContent	0x06000028
CDataSection	0x06000040
EntityDeclValue	0x06000041
AttributeDefValue	0x06000042
AttributeDefDefaultType	0x06000044
DocTypeWhiteSpace	0x06000080
ProcessingInstruction	0x07000002
ElementDef	0x07000004
DocTypePI	0x07000008
AttributeDefType	0x07000010
RequestedDomain	0x07000011

MRM parser constants:

Name	Value
PreDefStructureFav	0x01000000
PreDefStructure	0x01000001
SelfDefStructure	0x01000002
StructureInstance	0x01000004
MrmRoot	0x01000008
mtiSelfDefMessage	0x01000010
mtiPreDefMessage	0x01000012
mtiSelfDefIdentifier	0x02000001
mtiSdfFieldType	0x02000002
mtiSdfCharsCodepage	0x02000008
mtiSdfCharsEcho	0x02000010
mtiSdfCharsScale	0x02000011
mtiSdfCharsDateFmt	0x02000012
mtiSdfCharsTimeFmt	0x02000014
mtiSdfCharsTimeStampFmt	0x02000018
mtiSdfCharsBinaryFmt	0x02000020
mtiSdfCharsBinaryFmtContextLen	0x02000021
mtiSdfCharsBinaryFmtContext	0x02000022
mtiMixedContent	0x02000024
PreDefFieldFav	0x03000000
PreDefField	0x03000001
mtiSelfDefField	0x03000002
PreDefFieldInstance	0x03000004
SelfDefFieldInstance	0x03000008
Namespace	0x03000010
mtiPreDefStructureV	0x03000012

Name	Value
mtiSelfDefStructureV	0x03000014
mtiStructureInstanceV	0x03000016
mtiSelfDefMessageV	0x03000018
mtiPreDefMessageV	0x03000020
mtiUnresolvedChoice	0x04000001

XMLNSC parser constants:
See "XMLNSC: Using field types" on page 1094 for a list of the XMLNSC parser constants.

:

Related reference:

"User-defined extensions" on page 6411
Reference material that supports the creation and management of your user-defined extensions.

Trace logging from a user-defined C extension

Message processing nodes and parsers that are written to the C programming language API can write entries to trace.

You can use two types of trace:

- **Service Trace**: entries usually describe what is happening within the code and are only useful to the owner of the code, such as the user-defined extension developer.
- **User Trace**: entries usually describe what is happening at an external level and are useful to the user of the code. Users of the code include message flow designers, and broker domain administrators.

For each trace type, there are three levels:

- None
- Normal
- Debug

For C user-defined extensions, the following utility functions are available for each trace type:

- **cciServiceTrace** and **cciUserTrace**: these functions write an entry to the respective trace type only when trace has been activated, that is, trace is at normal or debug level.
- **cciServiceDebugTrace** and **cciUserDebugTrace**: these functions write an entry to the respective trace type only when trace is active at debug level.

To help avoid making function calls in the case where no trace is written, the **cciIsTraceActive** utility function is provided. **cciIsTraceActive** reports whether trace is active and the level at which trace is active.

The **cci*Trace** functions can be used by a user-defined extension regardless of the trace settings. The functions determine if trace is active and only write entries which are appropriate for the trace settings. When calling the **cci*Trace** functions, some additional processing can be required. The **cciIsTraceActive** function is provided to allow the user-defined extension to query the trace settings and avoid this extra processing when trace is inactive.

In many cases, it is sufficient to treat the value returned from the **cciIsTraceActive** function as a Boolean value. If the returned value is non zero, trace is active at some level and it is appropriate to call any of the **cci*Trace** functions. The returned value can also be inspected closely in the cases when details of the trace settings are required.

Trace settings can be changed at any time so it is advisable to query them regularly. For example, use **cciIsTraceActive** to query the trace settings when an implementation function is entered.

Trace entries can be associated with certain objects, which allows for further refinement of control for writing trace. A trace entry can be associated with a node or parser and trace is written according to the trace setting for that object. The object's trace setting is inherited from the message flow to which the node or parser belongs. If no object is specified, the trace is associated with the execution group.

Related reference:

"cciUserDebugTrace" on page 6673
Use cciUserDebugTrace to write a message from a message catalog (with inserts) to user trace when user trace is active at debug level.

"cciServiceTrace" on page 6663
Writes a message to service trace, if service trace is active.

"cciServiceDebugTrace" on page 6660
This function is very similar to cciServiceTrace with the only difference being that the entry is written to service trace only when service trace is active at debug level.

"cciIsTraceActive" on page 6670
cciIsTraceActive reports whether trace is active and the level at which trace is active.

"User trace" on page 6873
User trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Administration log. User trace is inactive by default; you must activate it explicitly by using a command, or by selecting options in the WebSphere Message Broker Toolkit.

"Service trace" on page 6875
Service trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Event Log or to user trace. Service trace is inactive by default; you must activate it explicitly by using a command.

Multicultural support considerations for message catalogs

WebSphere Message Broker converts any message that is loaded from the listed supported code pages into the local code page of the running processes (brokers) before output to the syslog.

You must provide symbolic links to your primary message catalogs for all locales that you intend to support. WebSphere Message Broker uses the LC_MESSAGES variable when opening message catalogs.

Multicultural support considerations on Windows:

Windows When you build a message file for Windows that contains multiple locales, ensure that the locale on the computer is set to a western European locale

(for example, English (United Kingdom)) before building the message catalogs. Use the **chcp** (Change Code Page) command to ensure that the code page is 850.

Write or convert all your message files (those with file type .mc) to the following code pages; each message file should be compiled separately by the message compiler with the additional flag that is specified in the following table.

DBCS message files do not need to be in Unicode (no **-U** flag). Use the **RC** command to 'resource compile' all the files, then use the **link** command to build a single message DLL.

Locale	Code page	Additional Flags
English (United States)	437	-U
German (Standard)	850	-U
Spanish (Modern Sort)	850	-U
French (Standard)	850	-U
Italian (Standard)	850	-U
Portuguese (Brazilian)	850	-U
Japan	932	
Simplified Chinese (China)	1381	
Traditional Chinese (Taiwan)	950	
Korean	949	

Multicultural support considerations on Linux and UNIX:
When you build message catalogs for Linux and UNIX systems, ensure that the catalogs are built in the code pages defined in the following table.

Locale	Code page
English	437
German	850
Spanish	850
French	850
Italian	850
Portuguese (Brazilian)	850
Japan	932
Simplified Chinese (China)	1381
Traditional Chinese (Taiwan)	950
Korean	949

Multicultural support considerations on z/OS:
z/OS When you build message catalogs for z/OS systems, ensure that the catalogs are built in the code pages defined in the following table.

Locale	Code page
English	1047
Japan	939
Simplified Chinese (China)	1388

Related tasks:

"Using error logging from a user-defined extension" on page 3137
Program user-defined extensions to write entries in the local error log.

Web services external standards

WebSphere Message Broker support for Web services conforms to a number of industry standards and specifications.

This section contains the topics that describe the WebSphere Message Broker support for Web services.

SOAP 1.1 and 1.2

SOAP is a lightweight, XML-based, protocol for exchange of information in a decentralized, distributed environment.

The protocol consists of three parts:

- An envelope that defines a framework for describing what is in a message and how to process it.
- A set of encoding rules for expressing instances of application-defined data types.
- A convention for representing remote procedure calls and responses.

SOAP can be used with other protocols, such as HTTP.

The specifications for SOAP are published by the World Wide Web Consortium (W3C).

- World Wide Web Consortium (W3C)

The specification for SOAP 1.1 is described in:

- Simple Object Access Protocol 1.1

This specification has not been endorsed by the W3C, but forms the basis for the SOAP 1.2 specification. The specification for SOAP 1.1 expands the SOAP acronym to Simple Object Access Protocol.

SOAP 1.2 is a W3C recommendation and is published in two parts:

- Part 1: Messaging Framework.
- Part 2: Adjuncts.

The specification also includes a primer that is intended to provide a tutorial on the features of the SOAP Version 1.2 specification, including usage scenarios. The specification for SOAP 1.2 does not expand the acronym. The primer is published at:

- SOAP 1.2 Primer

Related concepts:

"Web services external standards" on page 6696
WebSphere Message Broker support for Web services conforms to a number of industry standards and specifications.

"What is SOAP?" on page 1604
SOAP is an XML message format used in Web service interactions. SOAP messages are typically sent over HTTP or JMS, but other transport protocols can be used. The use of SOAP in a specific Web service is described by a WSDL definition.

Related information:

http://www.w3.org/

SOAP with Attachments

SOAP with Attachments (SwA) allows you to use SOAP 1.1 or SOAP 1.2 messages wrapped by MIME.

The SOAP with Attachments (SwA) specification is published as a formal submission by the World Wide Web Consortium (W3C):

- World Wide Web Consortium (W3C)

SwA uses the following specifications, described at:

- http://www.w3.org/TR/2000/NOTE-SOAP-attachments-20001211
- http://www.ws-i.org/Profiles/AttachmentsProfile-1.0.html

Related concepts:

"Web services external standards" on page 6696
WebSphere Message Broker support for Web services conforms to a number of industry standards and specifications.

"What is SOAP?" on page 1604
SOAP is an XML message format used in Web service interactions. SOAP messages are typically sent over HTTP or JMS, but other transport protocols can be used. The use of SOAP in a specific Web service is described by a WSDL definition.

SOAP MTOM

SOAP Message Transmission Optimization Mechanism (MTOM) is one of a related pair of specifications that define conceptually how to optimize the transmission and format of a SOAP message.

MTOM defines:

- How to optimize the transmission of base64binary data in SOAP messages in abstract terms
- How to implement optimized MIME multipart serialization of SOAP messages in a binding independent way using XOP

The implementation of MTOM relies on the related XML-binary Optimized Packaging (XOP) specification. Because these two specifications are so closely linked, they are normally referred to as MTOM/XOP.

The specification is published by the World Wide Web Consortium (W3C) as a W3C Recommendation at SOAP Message Transmission Optimization Mechanism. For further information refer to the following links:

- World Wide Web Consortium (W3C)
- SOAP Message Transmission Optimization Mechanism

Related concepts:

"Web services external standards" on page 6696
WebSphere Message Broker support for Web services conforms to a number of industry standards and specifications.

"XML-Binary Optimised Packaging (XOP)" on page 6702
XML-binary Optimized Packaging (XOP) is one of a related pair of specifications that define how to efficiently serialize XML Infosets that have certain types of content.

"SOAP Binding for MTOM 1.0" on page 6702
SOAP 1.1 Binding for MTOM 1.0 is a specification that describes how to use the SOAP Message Transmission Optimization Mechanism (MTOM) and XML-binary Optimized Packaging (XOP) specifications with SOAP 1.1.

SOAP over JMS

SOAP over Java Message Service 1.0 is a specification that describes how SOAP can bind to a messaging system that supports the Java Message Service (JMS).

SOAP is transport-independent and can be bound to any protocol. Both SOAP 1.1 and SOAP 1.2 can be bound in this way using the SOAP 1.2 Protocol Binding Framework. SOAP over JMS allows you to use the JMS transport with SOAP messages, and is an alternative messaging mechanism to the standard SOAP over HTTP messaging.

JMS can be used as a transport for SOAP. The specification for SOAP over Java Message Service 1.0 is published as a formal recommendation by the World Wide Web Consortium (W3C):

- SOAP over Java Message Service 1.0 (W3C recommendation)

JMS applications send messages to JMS destinations, which can be queues or topics. Queues are used in point-to-point messaging, and topics are used in publish/subscribe (pub/sub) messaging.

WebSphere Message Broker supports the JMS transport on the SOAP nodes, which are configured by importing WSDL with SOAP/JMS bindings. JMS topic destinations for publish/subscribe messaging are not supported for SOAP nodes in WebSphere Message Broker.

WebSphere Message Broker supports both the W3C standard WSDL format and the IBM proprietary (deprecated) WSDL format. For a summary of differences between these two WSDL formats, see "WSDL URI formats for JMS" on page 1668.

Related concepts:

"What is SOAP?" on page 1604
SOAP is an XML message format used in Web service interactions. SOAP messages are typically sent over HTTP or JMS, but other transport protocols can be used. The use of SOAP in a specific Web service is described by a WSDL definition.

"Processing JMS messages" on page 1679
JMS is the standard J2EE messaging API for building enterprise messaging applications. WebSphere Message Broker provides built-in input and output nodes for its supported protocols.

Related reference:

"WebSphere Broker JMS Transport" on page 1681
The WebSphere Broker JMS Transport is a service that connects applications that send and receive messages that conform to the Java Message Service (JMS) standard.

"JMS message structure" on page 1688
JMS messages have a defined structure that includes headers and payloads.

"SOAPInput node" on page 4795
Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

WSDL Version 1.1

Web Services Description Language (WSDL) is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information.

The operations and messages are described abstractly, then bound to a concrete network protocol and message format to define an endpoint. Related concrete end points are combined into abstract endpoints (services).

WSDL is extensible to allow the description of endpoints and their messages regardless of what message formats or network protocols are used to communicate. The WSDL 1.1 specification only defines bindings that describe how to use WSDL in conjunction with:

- SOAP 1.1
- HTTP GET
- HTTP POST
- MIME

The specification for WSDL 1.1 is published by the World Wide Web Consortium (W3C) as a W3C Note at WSDL Version 1.1.

- World Wide Web Consortium (W3C)
- WSDL Version 1.1

Related concepts:

"Web services external standards" on page 6696
WebSphere Message Broker support for Web services conforms to a number of industry standards and specifications.

"WSDL 1.1 Binding Extension for SOAP 1.2" on page 6701
WSDL 1.1 Binding Extension for SOAP 1.2 is a specification that defines the binding extensions that are required to indicate that Web service messages are bound to the SOAP 1.2 protocol.

"What is SOAP?" on page 1604
SOAP is an XML message format used in Web service interactions. SOAP messages are typically sent over HTTP or JMS, but other transport protocols can be used. The use of SOAP in a specific Web service is described by a WSDL definition.

WS-I Simple SOAP Binding Profile Version 1.0

WS-I Simple SOAP Binding Profile Version 1.0 (SSBP 1.0) is a set of non-proprietary Web services specifications, along with clarifications and amendments to those specifications which promote interoperability.

The SSBP 1.0 is derived from the WS-I Basic Profile 1.0 requirements that relate to the serialization of the envelope and its representation in the message.

WS-I Basic Profile 1.0 is split into two separately published profiles. These profiles are:

- WS-I Basic Profile Version 1.1
- WS-I Simple SOAP Binding Profile Version 1.0

Together these two profiles supersede the WS-I Basic Profile Version 1.0.

The specification for SSBP 1.0 is published by the Web Services Interoperability Organization (WS-I):

- Web Services Interoperability Organization (WS-I)

The specification for SSBP 1.0 can be found at:

- WS-I Simple SOAP Binding Profile Version 1.0

Related concepts:

"Web services external standards" on page 6696
WebSphere Message Broker support for Web services conforms to a number of industry standards and specifications.

"WS-I Basic Profile Version 1.1"
WS-I Basic Profile Version 1.1 (WS-I BP 1.1) is a set of non-proprietary Web services specifications, along with clarifications and amendments to those specifications, which together promote interoperability between different implementations of Web services.

"What is SOAP?" on page 1604
SOAP is an XML message format used in Web service interactions. SOAP messages are typically sent over HTTP or JMS, but other transport protocols can be used. The use of SOAP in a specific Web service is described by a WSDL definition.

"Message flow security and security profiles" on page 788
WebSphere Message Broker provides a security manager for implementing message flow security, so that end-to-end processing of a message through a message flow is secured based on an identity carried in that message instance.

WS-I Basic Profile Version 1.1

WS-I Basic Profile Version 1.1 (WS-I BP 1.1) is a set of non-proprietary Web services specifications, along with clarifications and amendments to those specifications, which together promote interoperability between different implementations of Web services.

The WS-I BP 1.1 is derived from Basic Profile Version 1.0 by incorporating its published errata and separating out the requirements that relate to the serialization of envelopes and their representation in messages. These requirements are now part of the Simple SOAP Binding Profile Version 1.0.

To summarize, the WS-I Basic Profile Version 1.0 is split into two separately published profiles. These profiles are:

- WS-I Basic Profile Version 1.1
- WS-I Simple SOAP Binding Profile Version 1.0

Together these two profiles supersede the WS-I Basic Profile Version 1.0.

The reason for this separation is to enable the Basic Profile 1.1 to be composed with any profile that specifies envelope serialization, including the Simple SOAP Binding Profile 1.0.

The specification for WS-I BP 1.1 is published by the Web Services Interoperability Organization (WS-I):

- Web Services Interoperability Organization (WS-I)

The specification for WS-I BP 1.1 can be found at:

- WS-I Basic Profile Version 1.1

Related concepts:

"Message flow security and security profiles" on page 788
WebSphere Message Broker provides a security manager for implementing message flow security, so that end-to-end processing of a message through a message flow is secured based on an identity carried in that message instance.

"Web services external standards" on page 6696
WebSphere Message Broker support for Web services conforms to a number of industry standards and specifications.

"WS-I Simple SOAP Binding Profile Version 1.0" on page 6700
WS-I Simple SOAP Binding Profile Version 1.0 (SSBP 1.0) is a set of non-proprietary Web services specifications, along with clarifications and amendments to those specifications which promote interoperability.

"What is SOAP?" on page 1604
SOAP is an XML message format used in Web service interactions. SOAP messages are typically sent over HTTP or JMS, but other transport protocols can be used. The use of SOAP in a specific Web service is described by a WSDL definition.

WSDL 1.1 Binding Extension for SOAP 1.2

WSDL 1.1 Binding Extension for SOAP 1.2 is a specification that defines the binding extensions that are required to indicate that Web service messages are bound to the SOAP 1.2 protocol.

The aim of this specification is to provide functions that are comparable with the binding for SOAP 1.1.

This specification is published as a formal submission request by the World Wide Web Consortium (W3C):

- World Wide Web Consortium (W3C)

The WSDL 1.1 Binding Extension for SOAP 1.2 specification is described at:

- http://www.w3.org/Submission/wsdl11soap12/

Related concepts:

"Web services external standards" on page 6696
WebSphere Message Broker support for Web services conforms to a number of industry standards and specifications.

"SOAP 1.1 and 1.2" on page 6696
SOAP is a lightweight, XML-based, protocol for exchange of information in a decentralized, distributed environment.

"What is SOAP?" on page 1604
SOAP is an XML message format used in Web service interactions. SOAP messages are typically sent over HTTP or JMS, but other transport protocols can be used. The use of SOAP in a specific Web service is described by a WSDL definition.

Related reference:

"What is WSDL?" on page 1615
WSDL is an XML notation for describing a web service. A WSDL definition tells a client how to compose a web service request and describes the interface that is provided by the web service provider.

XML-Binary Optimised Packaging (XOP)

XML-binary Optimized Packaging (XOP) is one of a related pair of specifications that define how to efficiently serialize XML Infosets that have certain types of content.

XOP defines how to efficiently serialize XML Infosets that have certain types of content by:

- Packaging the XML in some format. This is called the XOP package. The specification mentions MIME Multipart/Related but does not limit it to this format.
- Re-encoding all or part of base64binary content to reduce its size.
- Placing the base64binary content elsewhere in the package and replacing the encoded content with XML that references it.

XOP is used as an implementation of the MTOM specification, which defines the optimization of SOAP messages. Because these two specifications are so closely linked, they are normally referred to as MTOM/XOP.

The specification is published by the World Wide Web Consortium (W3C) as a W3C Recommendation XML-binary Optimized Packaging (XOP):

- World Wide Web Consortium (W3C)
- XML-binary Optimized Packaging (XOP)

Related concepts:

"Web services external standards" on page 6696
WebSphere Message Broker support for Web services conforms to a number of industry standards and specifications.

"SOAP MTOM" on page 6697
SOAP Message Transmission Optimization Mechanism (MTOM) is one of a related pair of specifications that define conceptually how to optimize the transmission and format of a SOAP message.

"SOAP Binding for MTOM 1.0"
SOAP 1.1 Binding for MTOM 1.0 is a specification that describes how to use the SOAP Message Transmission Optimization Mechanism (MTOM) and XML-binary Optimized Packaging (XOP) specifications with SOAP 1.1.

SOAP Binding for MTOM 1.0

SOAP 1.1 Binding for MTOM 1.0 is a specification that describes how to use the SOAP Message Transmission Optimization Mechanism (MTOM) and XML-binary Optimized Packaging (XOP) specifications with SOAP 1.1.

This specification defines the minimum changes required to enable MTOM and XOP to be used interoperably with SOAP 1.1 and to reuse the SOAP 1.2 MTOM/XOP implementation.

The SOAP 1.1 Binding for MTOM 1.0 specification is published as a formal submission by the World Wide Web Consortium (W3C):

- World Wide Web Consortium (W3C)

The SOAP 1.1 Binding for MTOM 1.0 specification is described at:

- http://www.w3.org/Submission/soap11mtom10/

Related concepts:

"Web services external standards" on page 6696
WebSphere Message Broker support for Web services conforms to a number of industry standards and specifications.

"SOAP MTOM" on page 6697
SOAP Message Transmission Optimization Mechanism (MTOM) is one of a related pair of specifications that define conceptually how to optimize the transmission and format of a SOAP message.

"XML-Binary Optimised Packaging (XOP)" on page 6702
XML-binary Optimized Packaging (XOP) is one of a related pair of specifications that define how to efficiently serialize XML Infosets that have certain types of content.

"What is SOAP?" on page 1604
SOAP is an XML message format used in Web service interactions. SOAP messages are typically sent over HTTP or JMS, but other transport protocols can be used. The use of SOAP in a specific Web service is described by a WSDL definition.

Web Services Security: SOAP Message Security

Web Services Security (WSS): SOAP Message Security is a set of enhancements to SOAP messaging that provides message integrity and confidentiality. WSS: SOAP Message Security is extensible, and can accommodate a variety of security models and encryption technologies.

WSS: SOAP Message Security provides three main mechanisms that can be used independently or together:

- The ability to send security tokens as part of a message, and for associating the security tokens with message content
- The ability to protect the contents of a message from unauthorized and undetected modification (message integrity)
- The ability to protect the contents of a message from unauthorized disclosure (message confidentiality).

WSS: SOAP Message Security can be used in conjunction with other Web service extensions and application-specific protocols to satisfy a variety of security requirements.

The specification is published by the Organization for the Advancement of Structures Standards (OASIS). The specification is called Web Services Security: SOAP Message Security 1.0 (WS-Security 2004).

- Organization for the Advancement of Structured Information Standards (OASIS)
- Web Services Security: SOAP Message Security 1.0 (WS-Security 2004)

Related concepts:

"Web services external standards" on page 6696
WebSphere Message Broker support for Web services conforms to a number of industry standards and specifications.

"How WebSphere Message Broker complies with Web Service Security specifications" on page 6705
WebSphere Message Broker conditionally complies with Web Services Security: SOAP Message Security and related specifications by supporting the following aspects.

Organization for the Advancement of Structured Information Standards (OASIS)
Web Services Security: SOAP Message Security 1.0 (WS-Security 2004)

XML Encryption Syntax and Processing

XML Encryption Syntax and Processing specifies a process for encrypting data and representing the result in XML. The data can be arbitrary data (including an XML document), an XML element, or XML element content. The result of encrypting data is an XML Encryption element that contains or references the cipher data.

XML Encryption Syntax and Processing is a recommendation of the World Wide Web Consortium (W3C):

- World Wide Web Consortium (W3C)

The XML Encryption Syntax and Processing recommendation is published at:

- XML Encryption Syntax and Processing

Related concepts:

"Web services external standards" on page 6696
WebSphere Message Broker support for Web services conforms to a number of industry standards and specifications.

XML-Signature Syntax and Processing

XML-Signature Syntax and Processing specifies the processing rules and syntax for XML digital signatures.

XML digital signatures provide integrity, message authentication, and signer authentication services for data of any type, whether located in the XML that includes the signature or elsewhere.

The recommendation for XML-Signature Syntax and Processing is published by the World Wide Web Consortium (W3C):

- World Wide Web Consortium (W3C)

The XML-Signature Syntax and Processing recommendation is published at:

- XML-Signature Syntax and Processing

Related concepts:

"Web services external standards" on page 6696
WebSphere Message Broker support for Web services conforms to a number of industry standards and specifications.

WebSphere Message Broker compliance with Web services standards

WebSphere Message Broker complies with the supported Web services standards and specifications, in that you can generate and deploy Web services that are compliant.

However, WebSphere Message Broker does not enforce this compliance. For example, for support of the WS-I Basic Profile 1.1 specification, you can apply additional qualities of service to your Web service that might break the interoperability outlined in this Profile.

The topics in this section describe how WebSphere Message Broker complies with Web services standards.

- "How WebSphere Message Broker complies with Web Service Security specifications" on page 6705

Related concepts:

"Web services external standards" on page 6696
WebSphere Message Broker support for Web services conforms to a number of industry standards and specifications.

How WebSphere Message Broker complies with Web Service Security specifications:

WebSphere Message Broker conditionally complies with Web Services Security: SOAP Message Security and related specifications by supporting the following aspects.

Compliance with Web Services Security: SOAP Message Security

Security header

The `<wsse:Security>` header provides a mechanism, in the form of a SOAP actor or role, for attaching security-related information that is targeted at a specific recipient. The recipient can be the ultimate recipient of the message or an intermediary. The following attributes are supported in WebSphere Message Broker:

- S11:actor (for an intermediary)
- S11:mustUnderstand
- S12:role (for an intermediary)
- S12:mustUnderstand

Security tokens

The following security tokens are supported in the security header:

- Username and password
- Binary security tokens:
  - X.509 certificate
  - Kerberos ticket
  - LTPA certificate
- SAML assertion

Token references

A security token conveys a set of claims. Sometimes these claims are elsewhere and need to be accessed by the receiving application. The `<wsse:SecurityTokenReference>` element provides an extensible mechanism for referencing security tokens. The following mechanisms are supported:

- Direct reference
- Key identifier
- Key name
- Embedded reference

Signature algorithms

This specification builds on XML Signature and therefore has the same algorithm requirements as those specified in the XML Signature specification. WebSphere Message Broker supports the signature algorithms as shown in the following table.

Algorithm type	Algorithm	URI
Digest	SHA1	`http://www.w3.org/2000/09/xmldsig#sha1`

Algorithm type	Algorithm	URI
Signature	DSA with SHA1 (validation only)	`http://www.w3.org/2000/09/xmldsig#dsa-sha1`
Signature	RSA with SHA1	`http://www.w3.org/2000/09/xmldsig#rsa-sha1`
Canonicalization	Exclusive XML canonicalization (without comments)	`http://www.w3.org/2001/10/xml-exc-c14n#`

Signature signed parts

WebSphere Message Broker allows the following SOAP elements to be signed:

- The SOAP message body
- The identity token (a type of security token) that is used as an asserted identity

Encryption algorithms

The data encryption algorithms that are supported are shown in the following table.

Algorithm	URI
Triple Data Encryption Standard algorithm (Triple DES)	`http://www.w3.org/2001/04/xmlenc#tripledes-cbc`
Advanced Encryption Standard (AES) algorithm with a key length of 128 bits	`http://www.w3.org/2001/04/xmlenc#aes128-cbc`
Advanced Encryption Standard (AES) algorithm with a key length of 192 bits	`http://www.w3.org/2001/04/xmlenc#aes192-cbc`
Advanced Encryption Standard (AES) algorithm with a key length of 256 bits	`http://www.w3.org/2001/04/xmlenc#aes256-cbc`

The key encryption algorithm that is supported is shown in the following table.

Algorithm	URI
Key transport (public key cryptography) RSA Version 1.5	`http://www.w3.org/2001/04/xmlenc#rsa-1_5`

Encryption message parts

WebSphere Message Broker allows the following SOAP elements to be encrypted:

- The SOAP body

Timestamp

The `<wsu:Timestamp>` element provides a mechanism for expressing the creation and expiration times of the security semantics in a message. WebSphere Message Broker tolerates the use of timestamps within the Web services security header on inbound SOAP messages.

Error handling

WebSphere Message Broker generates SOAP fault messages using the standard list of response codes listed in the specification.

Compliance with Web Services Security: Username Token Profile 1.1

The following aspects of this specification are supported:

Password types
Text

Token references
Direct reference

Compliance with Web Services Security: X.509 Certificate Token Profile 1.1

The following aspects of this specification are supported:

Token types

- X.509 Version 3: Single certificate.
- X.509 Version 3: X509PKIPathv1 without certificate revocation lists (CRL).
- X.509 Version 3: PKCS7 with or without CRLs. The IBM Software Development Kit (SDK) supports both. The Sun Java Development Kit (JDK) supports PKCS7 without CRL only.

For more information, refer to Web Services Security X.509 Certificate Token Profile.

Token references

- Key identifier - subject key identifier
- Direct reference
- Custom reference - issuer name and serial number

Compliance with Web Services Security: SAML Token Profile

SAML passthru support is provided, which enables interoperability with WS-Security SAML profiles, without performing subject confirmation processing. This means that it does not provide validation of the trust relationship between the SAML subject and message content signatures.

The token is passed through for processing by the message flow security manager, which passes the token to a WS-Trust STS for processing.

Compliance with Web Services Security: Kerberos Token Profile

The following aspects of this specification are supported:

Token types

- Kerberos GSS v5 AP_REQ
- Kerberos v5 AP_REQ

Aspects that are not supported

The following items are not supported in WebSphere Message Broker:

- Validation of Timestamps for freshness.
- Nonces.
- Web services security for SOAP attachments.
- XrML token profile.
- Web Services Interoperability (WS-I) Basic Security Profile.
- XML enveloping digital signature.
- XML enveloping digital encryption.

- The following transport algorithms for digital signatures:
  - XSLT: `http://www.w3.org/TR/1999/REC-xslt-19991116`.
  - SOAP Message Normalization. For more information, refer to http://www.w3.org/TR/2003/NOTE-soap12-n11n-20031008.
- The Diffie-Hellman key agreement algorithm for encryption. For more information, refer to Diffie-Hellman Key Values.
- The following canonicalization algorithm for encryption, which is optional in the XML encryption specification:
  - Canonical XML with or without comments
  - Exclusive XML canonicalization with or without comments
- The digest password type in the Username Token Version 1.0 Profile specification.

Related concepts:

"Web services external standards" on page 6696
WebSphere Message Broker support for Web services conforms to a number of industry standards and specifications.

"Web Services Security: SOAP Message Security" on page 6703
Web Services Security (WSS): SOAP Message Security is a set of enhancements to SOAP messaging that provides message integrity and confidentiality. WSS: SOAP Message Security is extensible, and can accommodate a variety of security models and encryption technologies.

W3C Working Group Note 8 October 2003

Diffie-Hellman Key Values

Web Services Security X.509 Certificate Token Profile

Testing and debugging applications

Use the reference information in this section to accomplish the testing and debugging tasks that address your business needs.

- "Test Client"
- "Message flow debugger" on page 6718

Test Client

Use the Test Client to test message flow applications.

You can send test messages to message flows that use any of the input nodes specified in "Test Client overview" on page 3144.

The following topics provide reference information to help you to use the Test Client effectively:

- "Test Client **Events** tab" on page 6709
- "Enqueue" on page 6712
- "Dequeue" on page 6712
- "Test Client Configuration tab" on page 6713
- "Test Client preferences" on page 6716
- "Deployment Location wizard" on page 6716
- "JMS events in the Test Client" on page 6717

Test Client Events tab

You can use the **Events** tab in the Test Client to edit the properties and content of your test messages. You also view test events in the **Events** tab when you run the test.

Purpose

Use the **Events** tab to edit and send the test messages, and to monitor the results of the test.

Message Flow Test Events

The **Message Flow Test Events** section of the **Events** tab displays the status and history of the test execution:

Invoke Message Flow
The Invoke Message Flow event starts the selected message flow. The message flow selection is defined by the selected message flow input node. Invoke Message Flow can also be the start of a new test execution with an empty test message. You can enter the message content and start the execution.

Starting
The Starting event indicates the beginning of the test execution.

Sending Message
The Sending Message event indicates that the message is being sent.

Enqueue
The Enqueue event indicates that the current message has been queued on an existing WebSphere MQ queue. Put a message to the specified queue manager, queue, port, and host name as defined in the Detailed Properties section. For more information, see “Enqueue” on page 6712.

Dequeue
The Dequeue event indicates that the current message has been dequeued from an existing WebSphere MQ queue. Click **Get Message** from the specified queue manager, queue, port and host name as defined in the Detailed Properties section. For more information, see “Dequeue” on page 6712.

Monitor
The Monitor event indicates the message was received on an output node monitor.

Stopped
The Stopped event indicates that the test execution has been stopped. The test execution can be stopped either by the Test Client or by the user.

Exception
The Exception event indicates when errors are encountered during the test. A message is displayed in the Exception message box on the right of the page and the message flow execution errors are logged in the Windows Event Viewer application log. To view the Windows Event Viewer application log, click **Event Viewer**.

Exception Trace
An Exception Trace event occurs when errors are encountered during test and Trace has been selected. Trace details are displayed in the Exception Trace panel on the right of the page.

Events tab actions

The following actions can be initiated on the **Events** tab, by right-clicking in **Message Flow Test Events**:

Re-run
Reruns the current message. To clone and rerun a previously started test message, right-click the message flow and click **Re-run**.

Duplicate
Duplicates the current message. To duplicate a previously started test message, right-click the message flow and click **Duplicate**.

Invoke
Restarts the current message. To restart a previously started test message, right-click the message flow and click **Invoke**.

Buttons on the Events tab

Several buttons are provided on the upper right of the **Events** tab:

Invoke
Starts a new Invoke Message Flow event where you can enter a request message and start the test execution.

Enqueue
Puts the message to the specified queue manager, queue, port, and host name as defined in the Detailed Properties section on the right of the page.

Dequeue
Gets the message from the specified queue manager, queue, port, and host name as defined in the Detailed Properties section on the right of the page.

Saved Messages
Displays the Data Pool editor in which you can select values that you have used in a previous test session. You can use the Data Pool editor to save or retrieve values for use during tests. When you use the Data Pool editor to save or retrieve values, the values are saved to, or retrieved from, a global data pool in your workspace. By selecting **Saved Messages**, you ensure that you are always working with the same set of values regardless of how many Test Client instances or test configurations you are using in your tests.

Stop Stops the current test.

Show Event Viewer
Displays the Event Viewer if the operating system is Windows.

Show Console
The Message Broker Runtime Console view opens. This view shows more details of the test run.

Detailed Properties

The Detailed Properties section varies according to the different events that you have selected in the Message Flow Test Events pane. The Detailed Properties section displays details of the current event type. The default details are:

- **Message Flow**. The name of the message flow that is being tested.
- **Input node**. The input node to which the test message is being sent.

Message

The Message section either displays the test message or the output message from a test event. If you are creating a new test message, you can select either the XML Structure editor or the Source editor to edit the test message. The XML Structure editor is available only if the input node that is selected on the message flow is expecting an XML message, and an existing message definition is associated with the message flow.

Header

Select the header to use for your test message if your message flow uses a WebSphere MQ or JMS input node.

SOAP operation

Select the SOAP operation to use for your test message if your message flow uses a SOAP input node.

Viewer

Select the appropriate editor to view or edit your test message or output message from the following editors:

XML Structure editor

Use the XML Structure editor to view and edit an XML test message derived from an associated message definition. You can change the content of the message by editing the entries in the Values column. Right-click to display a menu with options to change the content of the XML structure, including adding and removing elements. To view the generated source code, click **Show Generated Source**. Click **Saved messages** to display a list of saved error messages.

Source editor

Use the Source editor to compose and send test messages if you want to import a test message, or if the test message is not in XML format. To import a test message from a file, click **Import Source**.

Source tab

Edit the test message as plain text using the **Source** tab.

XML Source tab

Edit the test message in an XML editor using the **XML Source** tab.

Hexadecimal (read only) tab

View the test message in hexadecimal format by using the **Hexadecimal (Read Only)** tab.

Related tasks:

"Testing a message flow" on page 3146
You can test your message flows using the Test Client.

Related reference:

"Test Client" on page 6708
Use the Test Client to test message flow applications.

"Test Client Configuration tab" on page 6713
Configure your test environment in the **Configuration** tab in the Test Client.

Enqueue:

Enqueue is the term that is used to describe the process of putting a message on to a WebSphere MQ queue.

The following properties are entered on the Events page:

Host The name of the host computer.

Port The number of the port that is used.

Queue Manager
The name of WebSphere MQ Queue manager.

Queue
The name of the queue on the Queue manager.

Related reference:

"WebSphere MQ resources for the broker" on page 585
Each broker depends on a number of WebSphere MQ resources: some are required, others are optional, and depend on your environment and requirements. Some of these resources are created for you, but others you must define for yourself.

"Test Client **Events** tab" on page 6709
You can use the **Events** tab in the Test Client to edit the properties and content of your test messages. You also view test events in the **Events** tab when you run the test.

"Dequeue"
Dequeue is the term that is used to describe the process of removing a message from a WebSphere MQ queue.

Dequeue:

Dequeue is the term that is used to describe the process of removing a message from a WebSphere MQ queue.

The following properties are entered on the Events page:

Host The name of the host computer.

Port The number of the port that is used.

Queue Manager
The name of WebSphere MQ Queue manager.

Queue
The name of the queue on the Queue manager.

Related reference:

"WebSphere MQ resources for the broker" on page 585
Each broker depends on a number of WebSphere MQ resources: some are required, others are optional, and depend on your environment and requirements. Some of these resources are created for you, but others you must define for yourself.

"Enqueue"
Enqueue is the term that is used to describe the process of putting a message on to a WebSphere MQ queue.

"Test Client **Events** tab" on page 6709
You can use the **Events** tab in the Test Client to edit the properties and content of your test messages. You also view test events in the **Events** tab when you run the test.

Test Client Configuration tab

Configure your test environment in the **Configuration** tab in the Test Client.

Purpose

You can use the Test Client **Configuration** tab to alter the settings that are used when you test your message flow.

You can set the following settings in the **Configuration** tab:
- Message Flows
- Deployment
- MQ Settings
- JMS Settings
- MQ Message Headers
- JMS Message Headers

MQ settings apply to invoke, adding to queue, and removing from queue events; all other settings apply to invoke events only.

Select the relevant options in the left pane to display the properties in the right pane of the **Configuration** tab. The following sections describe the properties on the **Configuration** tab.

Message Flows

Add or remove Message Flows to be tested
In this section of the **Configuration** tab, the message flow that you have selected to test is listed. You can add more message flows to the test configuration so that you can test multiple message flows at the same time; for example, if you have an output message from one message flow that triggers another message flow, or if you are using subflows. Click **Add** to add message flows to your test configuration. Click **Remove** to remove message flows from the configuration.

Deployment

I will deploy the specified Broker Archive manually
You can select this option to prevent the Test Client from deploying the message flow before you send a message to test the message flow. Specify the broker archive (`.bar`) file that you want to deploy manually in the **Specify Broker Archive file** property on the **Deployment** section of the **Configuration** tab. Deploy the broker archive file to the execution group that is specified in the Deployment Location section.

Always rebuild and deploy a Broker Archive automatically
The Test Client always builds and deploys the file irrespective of whether there is a change to the broker archive file or its dependents, including message flows.

You can use this option to force the Test Client to deploy when the Test Client cannot detect changes in the message flow.

Only rebuild and deploy Broker Archive when changes occur
The Test Client rebuilds and deploys the broker archive file only when there is a change in the content of the message flow. This action is the default option.

Override configurable properties when rebuilding Broker Archive file
You can define the configurable properties in the Broker Archive editor.

Use this option to specify whether the user-defined value is overridden when the Test Client rebuilds the broker archive file.

Click **Change** to select a deployment location, see the "Deployment Location wizard" on page 6716.

MQ Settings

Stop when first MQ message is received
Use this option if you want the Test Client to stop receiving events when the first WebSphere MQ Output queue contains a message.

Add one or more conditions. This option is ignored if no MQ queues are being monitored.

The Test Client monitors MQ queues that are defined in the MQOutput nodes in the message flows that are being tested. If you check the option, it instructs the Test Client to stop testing when the message reaches any of the WebSphere MQ queues that are being monitored.

Select Purge or Browse option
Use this option to either purge a message from the queue or to browse the messages on the output queue.

Queue manager connection parameters
Enter the character set ID to use for connection to your queue manager.

JMS Settings

Stop when first JMS message is received
Use this option to stop the Test Client receiving events when the first JMS Output destination contains a message.

Specify JMS Client JARs
Use this option to add and remove JAR files that are used to create JMS connections.

Select **Use preference settings** to configure preference settings. This option is a global setting that can be applied to every Test Client in the same workbench.

WMQ Message Headers

WMQ Message Headers
Use this option to build multiple MQMD header definitions. When you send a message to a message flow that contains an MQInput node, you have the option to select an MQMD with suitable values for your test. Click **Add** to enter additional MQMD headers. Each new header is listed under 'MQ Message Headers' in the left navigator column. Each MQMD definition name must be unique within the Test Client configuration file.

MQ Message Header "Default Header"
This options specifies the default MQ Message Header definition. You can edit the values in this definition for your test configuration, or create a new WebSphere MQ Message Header definition.

If you select **MQ Message Headers** or **JMS Message Headers**, a **Duplicate** context menu is displayed. This context menu gives you the option to create a new message header based upon one that already exists.

If you select **Include RFH V2 header** in the right hand pane you can specify values in a different folder that are applicable to the MQRFH2 header.

If you select **Include MCD folder**, the MCD folder properties are included in the MQRFH2 header.

If you select **Include JMS folder**, the JMS folder properties are included in the MQRFH2 header.

If you select **Include USR folder**, any additional properties you add are included in the MQRFH2 header.

Note that if you have selected **Include JMS folder**, the MCD folder is included automatically. Furthermore, changing the **Message type** results in a change to the **Message domain** in the MCD folder.

When a message containing the MQRFH2 header is received, the header is parsed and the results appear in the **Detailed Properties** pane of the Test Client. To display all the properties of the MQRFH2 header, expand the **Header** tab.

For further information on how the MQRFH2 values work, see "MQRFH2 header" on page 6397

JMS Message Headers

JMS Message Headers

Use this option to enter multiple JMS Message Headers. Click **Add** to enter additional headers. Each new header is listed under 'JMS Headers' in the left navigator column.

Select each JMS Message Header to view and change the settings.

JMS Header

Enter the values for your JMS configuration in the JMS Header page.

Related concepts:

"Testing message flows by using the Test Client" on page 3144
You can test message flows in a safe environment before they are used on a production system by using the Test Client.

Related tasks:

"Configuring the test settings" on page 3148
You can configure the settings in the Test Client to control how your tests are run.

Related reference:

"Test Client" on page 6708
Use the Test Client to test message flow applications.

"Test Client **Events** tab" on page 6709
You can use the **Events** tab in the Test Client to edit the properties and content of your test messages. You also view test events in the **Events** tab when you run the test.

"Deployment Location wizard" on page 6716
Use the Deployment Location wizard to set the execution group to which the test message flow is deployed. You can also use the wizard to create a broker, a connection to a remote broker, and a new execution group.

"MQRFH2 header" on page 6397
The MQRFH2 header is used to pass messages to and from a message broker that belongs to WebSphere Message Broker.

Test Client preferences

You can set test settings for the local test environment that are used by the Websphere Message Broker Test Client.

You can configure the settings for the test in the Deployment Location wizard, or on the Test Client preferences. If you configure the settings on the Test Client preferences, these settings are applied to all tests. You can access the Test Client preferences by using the following instructions:

1. Click **Window** > **Preference**. The Preferences dialog is displayed.
2. Expand the item for Broker Development on the left and click Message Broker Test Client.

Use the following fields to define the settings for the local test engine:

Seconds to wait for deployment completion
The amount of time in seconds to allow for deployment. The default value is 20.

Seconds to wait after Test Client complete the deployment
The amount of time in seconds to wait after the Test Client completes deployment. The default value is 0.

Seconds to wait on launching the debugger for tracing purposes
The amount of time in seconds to wait before launching the debugger. The default value is 20.

Show information dialog before disconnecting the debugger
Select this option to display a dialog when you use the Test Client in Run mode if the flow debugger is already connected.

Seconds to wait for test client to stop
The amount of time in seconds to wait before ending the test. When the number of seconds has elapsed, monitoring of output nodes is stopped, and a Timeout event and a Stop event are displayed in the Test Client. The default value is 120.

Also select the required options:

Create queues of input and output nodes for message flows when host name is localhost
Use this option to create queues for the local host.

Add or modify (but not clear) what has already been deployed on the execution group Use this option to add or change, but not delete, what has previously been deployed on the current execution group.

Related reference:

"Deployment Location wizard"
Use the Deployment Location wizard to set the execution group to which the test message flow is deployed. You can also use the wizard to create a broker, a connection to a remote broker, and a new execution group.

Deployment Location wizard

Use the Deployment Location wizard to set the execution group to which the test message flow is deployed. You can also use the wizard to create a broker, a connection to a remote broker, and a new execution group.

The Deployment Location wizard is displayed when you click **Send Message** on the **Events** tab of the Test Client the first time that you run a test.

You can run the Test Client in one of the following modes:

- Run mode: test events are produced when the test message has been sent successfully from the message flow, or when an error occurs. Although you can also use the debugger, and the Test Client execution halts at the breakpoints, the Test Client does not receive and record message node-level trace information. When you use the Test Client in this mode, you are given the option to disconnect the flow debugger if it is already running. To use the Test Client in Run mode, ensure that **Trace and Debug** is not selected on the Deployment location wizard.
- Trace and debug mode: test trace events are produced when the test message leaves each node so that you can see the path that the message takes through the message flow. The results are stored in the broker. If you save the message flow test file, you can view the results at a later time. The flow debugger is launched in this mode, if it is not already connected. The test message stops at any breakpoints that you have configured in the message flow. You must configure a Java Debug port to run the test in the trace and debug mode.

If you use the Test Client in trace and debug mode, you can select the option **Stop at the beginning of the flow during debug**. This option suspends the test at the beginning of the message flow by setting a breakpoint on every connection after the selected input node. When you use the Test Client in the trace and debug mode, you can use the flow debugger in the Debug perspective.

You can also use the Deployment Location wizard to specify settings for the test. For more information about the test settings, see "Test Client preferences" on page 6716.

After you run a test for the first time, you can access the settings for the deployment location by clicking **Change** in the **Deployment** section of the **Configuration** tab.

Related concepts:

"Test Client overview" on page 3144
Use the Test Client to test message flows in a safe environment before they are used in a production system.

Related tasks:

"Testing a message flow" on page 3146
You can test your message flows using the Test Client.

"Using the Test Client in trace and debug mode" on page 3155
You can run the Test Client in trace and debug mode to trace the path of a test message through a message flow.

Related reference:

"Test Client" on page 6708
Use the Test Client to test message flow applications.

"Test Client Configuration tab" on page 6713
Configure your test environment in the **Configuration** tab in the Test Client.

"Test Client preferences" on page 6716
You can set test settings for the local test environment that are used by the Websphere Message Broker Test Client.

JMS events in the Test Client

Use the information in this topic to help you to understand JMS events in the Test Client.

When you test a message flow that contains JMS nodes, you might see some of the following events in the Message Broker Test Events section of the Test Client.

JMS Information event

The event details are displayed in the following properties.

Property name	Description
Initial Context Factory	The initial context factory that is used for JNDI look up
Location JNDI binding	The JNDI URL that is used for JNDI look up
Connection Factory	The JNDI name of the Connection Factory
Message Properties	The properties of the JMS message
Message Header	The header of the JMS message

The message content is displayed in the XML Structure or Source view.

JMS Response event

The event details are displayed in the following properties.

Property name	Description
Initial Context Factory	The initial context factory that is used for JNDI look up
Location JNDI binding	The JNDI URL that is used for JNDI look up
Connection Factory	The JNDI name of the Connection Factory
Message Properties	The properties of the JMS message
Message Header	The header of the JMS message

The message content is displayed in the XML Structure or Source view.

JMS Exception event

The event details are displayed in the following properties.

Property name	Description
Initial Context Factory	The initial context factory that is used for JNDI look up
Location JNDI binding	The JNDI URL that is used for JNDI lookup
Connection Factory	The JNDI name of the Connection Factory

The detailed trace message is displayed as well as the Exception trace details.

Related tasks:

"Testing a message flow" on page 3146
You can test your message flows using the Test Client.

"Testing a message flow that uses JMS nodes" on page 3150
You can configure settings in the Test Client for testing message flows that uses JMS nodes.

Related reference:

"Test Client" on page 6708
Use the Test Client to test message flow applications.

Message flow debugger

The flow debugger is a visual interface that supports the debugging of message flow applications in the WebSphere Message Broker Toolkit.

The following topics provide reference information to help you use the debugger effectively:

- "Flow debugger shortcuts"
- "Flow debugger icons and symbols" on page 6720

You can also use the "Java Debugger" on page 6723 provided by the Java Development tools to debug Java code within the WebSphere Message Broker Toolkit.

Related concepts:

"Flow debugger overview" on page 3158
Use the flow debugger in the WebSphere Message Broker Toolkit to track messages through your message flows.

"Debug perspective" on page 6789

Related tasks:

"Debugging a message flow" on page 3157
Use the tasks described in this section of the documentation to manage and work with the flow debugger.

Flow debugger shortcuts

You can use function keys and shortcut keys to complete actions in the flow debugger views and windows.

Shortcut keys are shown as a pair that you press together, followed by a subsequent key, for example `Shift-F10, C` means hold the Shift key down and press F10, then release both and press key C.

The following tables describe the main shortcuts that are available in the debug session:

- "Debug view"
- "Breakpoints view"
- "Flow Breakpoint Properties dialog" on page 6720
- "Variables view" on page 6720

To see a complete list of all the shortcuts that are available, press `Shift-F10` and release; the contextual menu is displayed.

Debug view:

Key combination	Function
Shift-F10, C	Run to completion
Shift-F10, E	Disconnect
F5 or Shift-F10, I	Step into
F8 or Shift-F10, M	Resume
Shift-F10, N	Terminate All
F6 or Shift-F10, O	Step over
Shift-F10, T	Terminate
F7 or Shift-F10, U	Step return
Shift-F10, V	Terminate and Remove

Breakpoints view:

Key combination	Function
Shift-F10, A	Select All
Shift-F10, B	Add breakpoint
Shift-F10, D	Disable the selected breakpoints
Shift-F10, E	Enable the selected breakpoints
Shift-F10, L	Remove all breakpoints
Shift-F10, O	Remove the selected breakpoints

Flow Breakpoint Properties dialog:

Key combination	Function
E	Enable the breakpoint
Alt-R, <space>	Restrict the breakpoint to the selected flow instances

Variables view:

Key combination	Function
Shift-F10, A	Select All
Shift-F10, C	Change the value of the selected flow variable
Shift-F10, V	Copy variables

Related concepts:

"Flow debugger overview" on page 3158
Use the flow debugger in the WebSphere Message Broker Toolkit to track messages through your message flows.

"Debug perspective" on page 6789

Related tasks:

"Debugging a message flow" on page 3157
Use the tasks described in this section of the documentation to manage and work with the flow debugger.

Related reference:

"Flow debugger icons and symbols"
The Debug perspective uses various debugger icons and symbols.

Flow debugger icons and symbols

The Debug perspective uses various debugger icons and symbols.

This topic describes the icons and symbols used in the Debug perspective and its views:

- "Debug perspective"
- "Debug view" on page 6721
- "Message Flow editor" on page 6721
- "Breakpoints view" on page 6722
- "Variables view" on page 6722

Debug perspective:

These icons and symbols are used in the Debug perspective *outside* any individual view.

Icon or Symbol	Description
	Debug perspective (symbol)
	Attach to Flow runtime environment (icon)

Debug view:

Icon or Symbol	Description
	Debug view (symbol)
	Flow engine (symbol)
	Flow (symbol)
	Flow instance paused (symbol)
	Flow instance running (symbol)
	Flow instance terminated (symbol)
	Stack frame (symbol)
	Detach from the selected flow engine (icon)
	Resume flow execution (icon)
	Run the flow to completion (icon)
	Step into subflow (icon)
	Step over node (icon)
	Step out of subflow (icon)
	Step into source code (icon)

Message Flow editor:
These icons and symbols in the message flow editor are specific to the flow debugger.

Icon or Symbol	Description
	Enabled breakpoint (symbol)
	Disabled breakpoint (symbol)
	Paused at breakpoint (symbol)

Icon or Symbol	Description
	Source code available (symbol)
	Error or exception (symbol)

Breakpoints view:

Icon or Symbol	Description
	Breakpoints view (symbol)
	Enabled breakpoint (symbol)
	Disabled breakpoint (symbol)
	Remove selected breakpoints (icon)
	Remove all breakpoints (icon)

Variables view:
These icons and symbols in the Variables view are specific to ESQL.

Icon or Symbol	Description
	Variable view (symbol)
	Tree reference variable (symbol)
	Message (symbol)
	ESQL reference variable (symbol)
	ESQL constant (symbol)
	ESQL scalar variable (symbol)
	ESQL schema variable (symbol)
	ESQL module variable (symbol)

Related concepts:

“Flow debugger overview” on page 3158
Use the flow debugger in the WebSphere Message Broker Toolkit to track messages through your message flows.

“Debug perspective” on page 6789

Related tasks:

“Debugging a message flow” on page 3157
Use the tasks described in this section of the documentation to manage and work

with the flow debugger.

Related reference:

"Flow debugger shortcuts" on page 6719

Java Debugger

The Java Development Tools include a debugger that enables you to detect and diagnose errors in your programs running on local or remote systems. You can control the execution of your program by setting breakpoints, suspending launched programs, stepping through your code, and examining the contents of variables.

For further information about the Java debugger, refer to the Java Development User Guide plug-in - Debugger.

Performance and monitoring

Use the reference information in this section to accomplish the performance and monitoring tasks that address your business needs.

- "Message flow accounting and statistics data"
- "Resource statistics data" on page 6745
- "Monitoring message flows" on page 6767

Message flow accounting and statistics data

You can collect message flow accounting and statistics data in various output formats.

Details of the information that is collected, and the output formats in which it can be recorded, are provided in the following topics:

- Statistics details
- Data formats
- Example output

You can also find information on how to use accounting and statistics data to improve the performance of a message flow in this developerWorks article on message flow performance.

Related concepts:

"Message flow accounting and statistics data" on page 3281
Message flow accounting and statistics data is the information that can be collected by a broker to record performance and operating details of message flow execution.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Monitoring message flow performance" on page 3279
You can configure your message flow to collect statistics about the operation and behavior of your message flows. Use this information to assess the performance of your message flows.

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"**mqsichangeflowstats** command" on page 3744
Use the **mqsichangeflowstats** command to control the accumulation of statistics about message flow operation.

"**mqsireportflowstats** command" on page 3929
Use the **mqsireportflowstats** command to display the current options for accounting and statistics that have been set using the **mqsichangeflowstats** command.

Message flow accounting and statistics details

You can collect message flow, thread, node, and terminal statistics for message flows.

Message flow statistics

One record is created for each message flow in an execution group. Each record contains the following details:

- Message flow name and UUID
- Execution group name and UUID
- Broker name and UUID
- Start and end times for data collection
- Type of data collected (snapshot or archive)
- Processor and elapsed time spent processing messages
- Processor and elapsed time spent waiting for input
- Number of messages processed
- Minimum, maximum, and average message sizes
- Number of threads available and maximum assigned at any time
- Number of messages committed and backed out
- Accounting origin

Thread statistics

One record is created for each thread assigned to the message flow. Each record contains the following details:

- Thread number (this has no significance and is for identification only)
- Processor and elapsed time spent processing messages
- Processor and elapsed time spent waiting for input
- Number of messages processed
- Minimum, maximum, and average message sizes

Node statistics

One record is created for each node in the message flow. Each record contains the following details:

- Node name
- Node type (for example MQInput)
- Processor time spent processing messages
- Elapsed time spent processing messages
- Number of times that the node is invoked
- Number of messages processed
- Minimum, maximum, and average message sizes. When the node type is FileInput, the message size is reported as 0 because the size is not known at the time that the message is propagated.

Terminal statistics

One record is created for each terminal on a node. Each record contains the following details:

- Terminal name
- Terminal type (input or output)
- Number of times that a message is propagated to this terminal

For further details about specific output formats, see the following topics:

- "User trace entries for message flow accounting and statistics data" on page 6730
- "XML publication for message flow accounting and statistics data" on page 6726

- "z/OS SMF records for message flow accounting and statistics data" on page 6734

Related concepts:

"Message flow accounting and statistics data" on page 3281
Message flow accounting and statistics data is the information that can be collected by a broker to record performance and operating details of message flow execution.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Monitoring message flow performance" on page 3279
You can configure your message flow to collect statistics about the operation and behavior of your message flows. Use this information to assess the performance of your message flows.

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"XML publication for message flow accounting and statistics data" on page 6726
Certain information is written to the XML publication for message flow accounting and statistics data.

"User trace entries for message flow accounting and statistics data" on page 6730
Certain information is written to the user trace log for message flow accounting and statistics data.

"z/OS SMF records for message flow accounting and statistics data" on page 6734
Certain information is written to z/OS SMF records for message flow accounting and statistics data.

"Message flow accounting and statistics output formats"
The message flow accounting and statistics data can be written in three formats.

"**mqsichangeflowstats** command" on page 3744
Use the **mqsichangeflowstats** command to control the accumulation of statistics about message flow operation.

"**mqsireportflowstats** command" on page 3929
Use the **mqsireportflowstats** command to display the current options for accounting and statistics that have been set using the **mqsichangeflowstats** command.

Message flow accounting and statistics output formats

The message flow accounting and statistics data can be written in three formats.

For more information about the available output formats, see the following topics:
- User trace entries
- XML publication
- z/OS SMF records

Related concepts:

"Message flow accounting and statistics data" on page 3281
Message flow accounting and statistics data is the information that can be collected by a broker to record performance and operating details of message flow execution.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Monitoring message flow performance" on page 3279
You can configure your message flow to collect statistics about the operation and behavior of your message flows. Use this information to assess the performance of your message flows.

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"**mqsichangeflowstats** command" on page 3744
Use the **mqsichangeflowstats** command to control the accumulation of statistics about message flow operation.

"**mqsireportflowstats** command" on page 3929
Use the **mqsireportflowstats** command to display the current options for accounting and statistics that have been set using the **mqsichangeflowstats** command.

"Example message flow accounting and statistics data" on page 6739
You can view message flow accounting and statistic data in an XML publication or user trace entries. To view z/OS SMF records, use a utility program that processes SMF records.

XML publication for message flow accounting and statistics data:

Certain information is written to the XML publication for message flow accounting and statistics data.

The data is created in the folder `WMQIStatisticsAccounting`, which contains subfolders that provide more detailed information. All folders are present in the publication even if you set current data collection parameters to specify that the relevant data is not collected.

Snapshot data is used for performance analysis, and is published as retained and not persistent. Archive data is used for accounting where an audit trail might be required, and is published as retained and persistent. All publications are global and can be collected by a subscriber that is registered anywhere in the network. They can also be collected by more than one subscriber.

One XML publication is generated for each message flow that is producing data for the time period that you choose. For example, if MessageFlowA and MessageFlowB are both producing archive data over a period of 60 minutes, both MessageFlowA and MessageFlowB produce an XML publication every 60 minutes.

If you are concerned about the safe delivery of these messages (for example, for charging purposes), use a secure delivery mechanism such as WebSphere MQ.

The folders and subfolders in the XML publication have the following identifiers:
- WMQIStatisticsAccounting
- MessageFlow
- Threads
- ThreadStatistics
- Nodes
- NodesStatistics
- TerminalStatistics

The tables provided here describe the contents of each of these folders.

The following table describes the general accounting and statistics information, created in folder WMQIStatisticsAccounting.

Field	Data type	Details
RecordType	Character	Type of output, one of: • Archive • Snapshot
RecordCode	Character	Reason for output, one of: • MajorInterval • Snapshot • Shutdown • ReDeploy • StatsSettingsModified

The following table describes the message flow statistics information, created in folder MessageFlow.

Field	Data type	Details
BrokerLabel	Character (maximum 32)	Broker name
BrokerUUID	Character (maximum 32)	Broker universal unique identifier
ExecutionGroupName	Character (maximum 32)	Execution group name
ExecutionGroupUUID	Character (maximum 32)	Execution group universal unique identifier
MessageFlowName	Character (maximum 32)	Message flow name
StartDate	Character	Interval start date (YYYY-MM-DD)
StartTime	Character	Interval start time (HH:MM:SS:NNNNNN)
EndDate	Character	Interval end date (YYYY-MM-DD)
EndTime	Character	Interval end time (HH:MM:SS:NNNNNN)
TotalElapsedTime	Numeric	Total elapsed time spent processing input messages (microseconds)
MaximumElapsedTime	Numeric	Maximum elapsed time that is spent processing an input message (microseconds)
MinimumElapsedTime	Numeric	Minimum elapsed time that is spent processing an input message (microseconds)
TotalCPUTime	Numeric	Total processor time spent processing input messages (microseconds)
MaximumCPUTime	Numeric	Maximum processor time that is spent processing an input message (microseconds)
MinimumCPUTime	Numeric	Minimum processor time that is spent processing an input message (microseconds)

Field	Data type	Details
CPUTimeWaitingForInputMessage	Numeric	Total processor time spent waiting for input messages (microseconds)
ElapsedTimeWaitingForInputMessage	Numeric	Total elapsed time that is spent waiting for input messages (microseconds)
TotalInputMessages	Numeric	Total number of messages processed TotalInputMessages records only those messages that are propagated from input node terminals.
TotalSizeOfInputMessages	Numeric	Total size of input messages (bytes)
MaximumSizeOfInputMessages	Numeric	Maximum input message size (bytes)
MinimumSizeOfInputMessages	Numeric	Minimum message input size (bytes)
NumberOfThreadsInPool	Numeric	Number of threads in pool
TimesMaximumNumberofThreadsReached	Numeric	Number of times the maximum number of threads is reached
TotalNumberOfMQErrors	Numeric	Number of MQGET errors (MQInput node) or web services errors (HTTPInput node) For example, a conversion error occurs when the message is got from the queue.
TotalNumberOfMessagesWithErrors	Numeric	Number of messages that contain errors These errors include exceptions that are thrown downstream of the input node, and errors that are detected by the input node after it successfully retrieves the message from the queue, but before it propagates it to the output terminal (for example, a format error). TotalNumberOfMessagesWithErrors can include messages that are not included in TotalInputMessages.
TotalNumberOfErrorsProcessingMessages	Numeric	Number of errors when processing a message
TotalNumberOfTimeOutsWaitingForRepliesToAggregateMessages	Numeric	Number of timeouts when processing a message (AggregateReply node only)
TotalNumberOfCommits	Numeric	Number of transaction commits
TotalNumberOfBackouts	Numeric	Number of transaction backouts
AccountingOrigin	Character (maximum 32)	Accounting origin

The following table describes the thread statistics information, created in folder `Threads`.

Field	Data type	Details
Number	Numeric	Number of thread statistics subfolders in Threads folder

The following table describes the thread statistics information for each individual thread, created in folder ThreadStatistics, a subfolder of Threads.

Field	Data type	Details
Number	Numeric	Relative thread number in pool
TotalNumberOfInputMessages	Numeric	Total number of messages that are processed by a thread
TotalElapsedTime	Numeric	Total elapsed time spent processing input messages (microseconds)
TotalCPUTime	Numeric	Total processor time spent processing input messages (microseconds)
CPUTimeWaitingForInputMessage	Numeric	Total processor time spent waiting for input messages (microseconds)
ElapsedTimeWaitingForInputMessage	Numeric	Total elapsed time that is spent waiting for input messages (microseconds)
TotalSizeOfInputMessages	Numeric	Total size of input messages (bytes)
MaximumSizeOfInputMessages	Numeric	Maximum size of input messages (bytes)
MinimumSizeOfInputMessages	Numeric	Minimum size of input messages (bytes)

The following table describes the node statistics information, created in folder Nodes.

Field	Data type	Details
Number	Numeric	Number of node statistics subfolders in Nodes folder

The following table describes the node statistics information for each individual node, created in folder NodesStatistics, a subfolder of Nodes.

Field	Data type	Details
Label	Character	Name of node (Label)
Type	Character	Type of node
TotalElapsedTime	Numeric	Total elapsed time spent processing input messages (microseconds)
MaximumElapsedTime	Numeric	Maximum elapsed time spent processing input messages (microseconds)
MinimumElapsedTime	Numeric	Minimum elapsed time spent processing input messages (microseconds)
TotalCPUTime	Numeric	Total processor time spent processing input messages (microseconds)
MaximumCPUTime	Numeric	Maximum processor time spent processing input messages (microseconds)
MinimumCPUTime	Numeric	Minimum processor time spent processing input messages (microseconds)
CountOfInvocations	Numeric	Total number of messages that are processed by this node
NumberOfInputTerminals	Numeric	Number of input terminals
NumberOfOutputTerminals	Numeric	Number of output terminals

The following table describes the terminal statistics information, created in folder TerminalStatistics.

Field	Data type	Details
Label	Character	Name of terminal
Type	Character	Type of terminal, one of: • Input • Output
CountOfInvocations	Numeric	Total number of invocations

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message flow accounting and statistics data" on page 3281
Message flow accounting and statistics data is the information that can be collected by a broker to record performance and operating details of message flow execution.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Monitoring message flow performance" on page 3279
You can configure your message flow to collect statistics about the operation and behavior of your message flows. Use this information to assess the performance of your message flows.

Related reference:

"Message flow accounting and statistics details" on page 6724
You can collect message flow, thread, node, and terminal statistics for message flows.

"Example of an XML publication for message flow accounting and statistics" on page 6739
This example shows an XML publication that contains message flow accounting and statistics data.

User trace entries for message flow accounting and statistics data:

Certain information is written to the user trace log for message flow accounting and statistics data.

The data records are identified by the following message numbers:
- BIP2380I
- BIP2381I
- BIP2382I
- BIP2383I

The inserts for each message are described in the following tables.

This table describes the inserts in message BIP2380I. One message is written for the message flow.

Field	Data type	Details
ProcessID	Numeric	Process ID

Field	Data type	Details
Key	Numeric	Key that is used to associate related accounting and statistics BIP messages
Type	Character	Type of output, one of: • Archive • Snapshot
Reason	Character	Reason for output, one of: • MajorInterval • Snapshot • Shutdown • ReDeploy • StatsSettingsModified
BrokerLabel	Character (maximum 32)	Broker name
BrokerUUID	Character (maximum 32)	Broker universal unique identifier
ExecutionGroupName	Character (maximum 32)	Execution group name
ExecutionGroupUUID	Character (maximum 32)	Execution group universal unique identifier
MessageFlowName	Character (maximum 32)	Message flow name
StartDate	Character	Interval start date (YYYY-MM-DD)
StartTime	Character	Interval start time (HH:MM:SS:NNNNNN)
EndDate	Character	Interval end date (YYYY-MM-DD)
EndTime	Character	Interval end time (HH:MM:SS:NNNNNN)
TotalElapsedTime	Numeric	Total elapsed time spent processing input messages (microseconds)
MaximumElapsedTime	Numeric	Maximum elapsed time that is spent processing an input message (microseconds)
MinimumElapsedTime	Numeric	Minimum elapsed time that is spent processing an input message (microseconds)
TotalCPUTime	Numeric	Total processor time spent processing input messages (microseconds)
MaximumCPUTime	Numeric	Maximum processor time that is spent processing an input message (microseconds)
MinimumCPUTime	Numeric	Minimum processor time that is spent processing an input message (microseconds)
CPUTimeWaitingForInputMessage	Numeric	Total processor time spent waiting for input messages (microseconds)

Field	Data type	Details
ElapsedTimeWaitingForInputMessage	Numeric	Total elapsed time that is spent waiting for input messages (microseconds)
TotalInputMessages	Numeric	Total number of messages processed TotalInputMessages records only those messages that are propagated from input node terminals.
TotalSizeOfInputMessages	Numeric	Total size of input messages (bytes)
MaximumSizeOfInputMessages	Numeric	Maximum input message size (bytes)
MinimumSizeOfInputMessages	Numeric	Minimum input message size (bytes)
NumberOfThreadsInPool	Numeric	Number of threads in pool
TimesMaximumNumberofThreadsReached	Numeric	Number of times the maximum number of threads is reached
TotalNumberOfMQErrors	Numeric	Number of MQGET errors (MQInput node) or web services errors (HTTPInput node) For example, a conversion error occurs when the message is got from the queue.
TotalNumberOfMessagesWithErrors	Numeric	Number of messages that contain errors These errors include exceptions that are thrown downstream of the input node, and errors that are detected by the input node after it successfully retrieves the message from the queue (for example, a format error). TotalNumberOfMessagesWithErrors can include messages that are not included in TotalInputMessages.
TotalNumberOfErrorsProcessingMessages	Numeric	Number of errors while processing a message
TotalNumberOfTimeOutsWaitingForRepliesToAggregateMessages	Numeric	Number of timeouts while processing a message (AggregateReply node only)
TotalNumberOfCommits	Numeric	Number of transaction commits
TotalNumberOfBackouts	Numeric	Number of transaction backouts
AccountingOrigin	Character (maximum 32)	Accounting origin

The following table describes the inserts in message BIP2381I. One message is written for each thread.

Field	Data type	Details
ProcessID	Numeric	Process ID
Key	Numeric	Key that is used to associate related accounting and statistics BIP messages

Field	Data type	Details
Number	Numeric	Relative thread number in pool
TotalNumberOfInputMessages	Numeric	Total number of messages that are processed by a thread
TotalElapsedTime	Numeric	Total elapsed time spent processing input messages (microseconds)
TotalCUPTime	Numeric	Total processor time spent processing input messages (microseconds)
CPUTimeWaitingForInputMessage	Numeric	Total processor time spent waiting for input messages (microseconds)
ElapsedTimeWaitingForInputMessage	Numeric	Total elapsed time that is spent waiting for input messages (microseconds)
TotalSizeOfInputMessages	Numeric	Total size of input messages (bytes)
MaximumSizeOfInputMessages	Numeric	Maximum size of input messages (bytes)
MinimumSizeOfInputMessages	Numeric	Minimum size of input messages (bytes)

The following table describes the inserts in message BIP2382I. One message is written for each node.

Field	Data type	Details
ProcessID	Numeric	Process ID
Key	Numeric	Key that is used to associate related accounting and statistics BIP messages
Label	Character	Name of node (Label)
Type	Character	Type of node
TotalElapsedTime	Numeric	Total elapsed time spent processing input messages (microseconds)
MaximumElapsedTime	Numeric	Maximum elapsed time spent processing input messages (microseconds)
MinimumElapsedTime	Numeric	Minimum elapsed time spent processing input messages (microseconds)
TotalCPUTime	Numeric	Total processor time spent processing input messages (microseconds)
MaximumCPUTime	Numeric	Maximum processor time spent processing input messages (microseconds)
MinimumCPUTime	Numeric	Minimum processor time spent processing input messages (microseconds)
CountOfInvocations	Numeric	Total number of messages that are processed by this node
NumberOfInputTerminals	Numeric	Number of input terminals
NumberOfOutputTerminals	Numeric	Number of output terminals

The following table describes the inserts in message BIP2383I. One message is written for each terminal on each node.

Field	Data type	Details
ProcessID	Numeric	Process ID

Field	Data type	Details
Key	Numeric	Key that is used to associate related accounting and statistics BIP messages
Label	Character	Name of terminal
Type	Character	Type of terminal, one of: • Input • Output
CountOfInvocations	Numeric	Total number of invocations

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message flow accounting and statistics data" on page 3281
Message flow accounting and statistics data is the information that can be collected by a broker to record performance and operating details of message flow execution.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Monitoring message flow performance" on page 3279
You can configure your message flow to collect statistics about the operation and behavior of your message flows. Use this information to assess the performance of your message flows.

Related reference:

"Message flow accounting and statistics details" on page 6724
You can collect message flow, thread, node, and terminal statistics for message flows.

"Example of user trace entries for message flow accounting and statistics" on page 6741
This example shows a user trace that contains message flow accounting and statistics data.

z/OS SMF records for message flow accounting and statistics data:

Certain information is written to z/OS SMF records for message flow accounting and statistics data.

The data records are type 117 records with the following identifiers:
- BipSMFDate
- BipSMFRecordHdr
- BipSMFTriplet
- BipSMFMessageFlow
- BipSMFThread
- BipSMFNode
- BipSMFTerminal

The following tables describe the contents of each of these records.

The following table describes the contents of the BipSMFDate record.

Field	Data type	Details
YYYY	signed short int	2 byte year
MM	char	1 byte month
DD	char	1 byte day

The following table describes the contents of the BipSMFRecordHdr record.

Field	Data type	Details
SM117LEN	unsigned short int	SMF record length
SM117SEG	unsigned short int	System reserved
SM117FLG	char	System indicator
SM117RTY	char	Record type 117 (x'75')
SM117TME	unsigned int	Time when SMF moved the record (time since midnight in hundredths of a second)
SM117DTE	unsigned int	Date when SMF moved the record in packed decimal form 0cyydddF where: c is 0 (19xx) or 1 (20xx) yy is the current year (0-99) ddd is the current day (1-366) F is the sign
SM117SID	unsigned int	System ID
SM117SSI	unsigned int	Subsystem ID
SM117STY	unsigned short int	Record subtype, one of : • 1 (only message flow or threads data is being collected) • 2 (node data is being collected)[1]
SM117TCT	unsigned int	Count of triplets
SM117SRT	unsigned char	Record type, one of: • Archive • Snapshot
SM117SRC	unsigned char	Record code, one of: • 00 = None • 01 = Major Interval • 02 = Snapshot • 03 = Shutdown • 04 = Redeploy • 05 = Stats Settings Modified
SM117RSQ	unsigned short int	Sequence number of the record when multiple records are written for a collection interval.
SM117NOR	unsigned short int	Total number of related records in a collection interval.

Note:

1. When only nodes data is being collected, a single subtype 2 record is written. If nodes and terminals data is being collected, multiple subtype 2 records are written.

The following table describes the contents of the BipSMFTriplet record.

Field	Data type	Details
TRPLTOSE	signed int	Offset of record from start of SMF record

Field	Data type	Details
TRPLTDLE	signed short int	Length of data type
TRPLTNDR	signed short int	Number of data types in SMF record

The following table describes the contents of the BipSMFMessageFlow record.

Field	Data type	Details
IMFLID	short int	Control block hex ID (BipSMFMessageFlow_ID)
IMFLLEN	short int	Length of control block
IMFLEYE	char[4]	Eyecatcher (IMFL)
IMFLVER	int	Version number (BipSMFRecordVersion)
IMFLBKNM	char[32]	Broker name
IMFLBKID	char[36]	Broker universal unique identifier
IMFLEXNM	char[32]	Execution group name
IMFLEXID	char[36]	Execution group universal unique identifier
IMFLMFNM	char[32]	Message flow name
IMFLSTDT	BipSMFDate	Interval start date
IMFLSTTM	unsigned int	Interval start time (format as for SM117TME)
IMFLENDT	BipSMFDate	Interval end date
IMFLENTM	unsigned int	Interval end time (format as for SM117TME)
IMFLTPTM	long long int	Total elapsed time spent processing input messages (8 bytes binary, microseconds)
IMFLMXTM	long long int	Maximum elapsed time spent processing an input message (8 bytes binary, microseconds)
IMFLMNTM	long long int	Minimum elapsed time spent processing an input message (8 bytes binary, microseconds)
IMFLTPCP	long long int	Total processor time spent processing input messages (8 bytes binary, microseconds)
IMFLMXCP	long long int	Maximum processor time spent processing an input message (8 bytes binary, microseconds)
IMFLMNCP	long long int	Minimum processor time spent processing an input message (8 bytes binary, microseconds)
IMFLWTCP	long long int	Total processor time spent waiting for input messages (8 bytes binary, microseconds)
IMFLWTIN	long long int	Total elapsed time spent waiting for input messages (8 bytes binary, microseconds)
IMFLTPMG	unsigned int	Total number of messages processed
IMFLTSMG	long long int	Total size of input messages (bytes)
IMFLMXMG	long long int	Maximum input message size (bytes)
IMFLMNMG	long long int	Minimum input message size (bytes)
IMFLTHDP	unsigned int	Number of threads in pool
IMFLTHDM	unsigned int	Number of times the maximum number of threads is reached
IMFLERMQ[1]	unsigned int	Number of MQGET errors (MQInput node) or Web services errors (HTTPInput node)

Field	Data type	Details
IMFLERMG[2]	unsigned int	Number of messages that contain errors
IMFLERPR	unsigned int	Number of errors processing a message
IMFLTMOU	unsigned int	Number of timeouts processing a message (AggregateReply node only)
IMFLCMIT	unsigned int	Number of transaction commits
IMFLBKOU	unsigned int	Number of transaction backouts
IMFLACCT	char[32]	Accounting origin

Notes:
1. For example, a conversion error occurs when the message is got from the queue.
2. These include exceptions that are thrown downstream of the input node, and errors detected by the input node after it has successfully retrieved the message from the queue (for example, a format error).

The following table describes the contents of the BipSMFThread record.

Field	Data type	Details
ITHDID	short int	Control block hex ID (BipSMFThread_ID)
ITHDLEN	short int	Length of control block
ITHDEYE	char[4]	Eyecatcher (ITHD)
ITHDVER	int	Version number (BipSMFRecordVersion)
ITHDNBR	unsigned int	Relative thread number in pool
ITHDTPMG	unsigned int	Total number of messages processed by thread
ITHDTPTM	long long int	Total elapsed time spent processing input messages (8 bytes binary, microseconds)
ITHDTPCP	long long int	Total processor time spent processing input messages (8 bytes binary, microseconds)
ITHDWTCP	long long int	Total processor time spent waiting for input messages (8 bytes binary, microseconds)
ITHDWTIN	long long int	Total elapsed time spent waiting for input messages (8 bytes binary, microseconds)
ITHDTSMG	long long int	Total size of input messages (bytes)
ITHDMXMG	long long int	Maximum size of input messages (bytes)
ITHDMNMG	long long int	Minimum size of input messages (bytes)

The following table describes the contents of the BipSMFNode record.

Field	Data type	Details
INODID	short int	Control block hex ID (BipSMFNode_ID)
INODLEN	short int	Length of control block
INODEYE	char[4]	Eyecatcher (INOD)
INODVER	int	Version number (BipSMFRecordVersion)
INODNDNM	char[32]	Name of node (Label)
INODTYPE	char[32]	Type of node
INODTPTM	long long int	Total elapsed time spent processing input messages (8 bytes binary, microseconds)

Field	Data type	Details
INODMXTM	long long int	Maximum elapsed time spent processing input messages (8 bytes binary, microseconds)
INODMNTM	long long int	Minimum elapsed time spent processing input messages (8 bytes binary, microseconds)
INODTPCP	long long int	Total processor time spent processing input messages (8 bytes binary, microseconds)
INODMXCP	long long int	Maximum processor time spent processing input messages (8 bytes binary, microseconds)
INODMNCP	long long int	Minimum processor time spent processing input messages (8 bytes binary, microseconds)
INODTPMG	unsigned int	Total number of messages processed by this node
INODNITL	unsigned int	Number of input terminals
INODNOTL	unsigned int	Number of output terminals

The following table describes the contents of the BipSMFTerminal record.

Field	Data type	Details
ITRMID	short int	Control block hex ID (BipSMFTerminal_ID)
ITRMLEN	short int	Length of control block
ITRMEYE	char[4]	Eyecatcher (ITRM)
ITRMVER	int	Version number (BipSMFRecordVersion)
ITRMTLNM	char[32]	Name of terminal
ITRMTYPE	char[8]	Type of terminal, one of: • Input • Output
ITRMTINV	unsigned int	Total number of invocations

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message flow accounting and statistics data" on page 3281
Message flow accounting and statistics data is the information that can be collected by a broker to record performance and operating details of message flow execution.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Monitoring message flow performance" on page 3279
You can configure your message flow to collect statistics about the operation and behavior of your message flows. Use this information to assess the performance of your message flows.

Related reference:

"Message flow accounting and statistics details" on page 6724
You can collect message flow, thread, node, and terminal statistics for message flows.

Example message flow accounting and statistics data

You can view message flow accounting and statistic data in an XML publication or user trace entries. To view z/OS SMF records, use a utility program that processes SMF records.

The following topics give example output in two formats:
- XML publication
- User trace entries

An example is not provided for z/OS SMF records, because these contain hexadecimal data and are not easily viewed in that form. To view SMF records, use any available utility program that processes SMF records.

Related concepts:

"Message flow accounting and statistics data" on page 3281
Message flow accounting and statistics data is the information that can be collected by a broker to record performance and operating details of message flow execution.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Monitoring message flow performance" on page 3279
You can configure your message flow to collect statistics about the operation and behavior of your message flows. Use this information to assess the performance of your message flows.

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"Message flow accounting and statistics output formats" on page 6725
The message flow accounting and statistics data can be written in three formats.

"**mqsichangeflowstats** command" on page 3744
Use the **mqsichangeflowstats** command to control the accumulation of statistics about message flow operation.

"**mqsireportflowstats** command" on page 3929
Use the **mqsireportflowstats** command to display the current options for accounting and statistics that have been set using the **mqsichangeflowstats** command.

Example of an XML publication for message flow accounting and statistics:

This example shows an XML publication that contains message flow accounting and statistics data.

The following example shows what is generated for a snapshot report. The content of this publication message shows that the message flow is called *XMLflow*, and that it is running in an execution group named *default* on broker *MQ02BRK*. The message flow contains the following nodes:
- An MQInput node called *INQueue3*
- An MQOutput node called *OUTQueue*
- An MQOutput node called *FAILQueue*

The MQInput node's Out terminal is connected to the OUTQueue node. The MQInput node's Failure terminal is connected to the FAILQueue node.

During the interval for which statistics have been collected, this message flow processed no messages.

A publication that is generated for this data always includes the appropriate folders, even if there is no current data.

The following command has been issued to achieve these results:

```
mqsichangeflowstats MQ02BRK -s -c active -e default -f XMLFlow -n advanced -t basic -b basic -o xml
```

Blank lines have been added between folders to improve readability.

The broker takes information about statistics and accounting from the operating system. On some operating systems, such as Windows, UNIX, and Linux, rounding can occur because the system calls that are used to determine the processor times are not sufficiently granular. This rounding might affect the accuracy of the data.

The following example is the subscription message. The <psc> and <mcd> elements are part of the RFH header.

```
<psc>
  <Command>Publish</Command>
  <PubOpt>RetainPub</PubOpt>
  <Topic>$SYS/Broker/MQ02BRK/StatisticsAccounting/SnapShot/default/XMLflow
  </Topic>
</psc>


<mcd>
  <Msd>xml</Msd>
</mcd>
```

The following example is the publication that the broker generates:

```
<WMQIStatisticsAccounting RecordType="SnapShot" RecordCode="Snapshot">

<MessageFlow BrokerLabel="MQ02BRK"
 BrokerUUID="7d951e31-f200-0000-0080-efe1b9d849dc"
 ExecutionGroupName="default"
 ExecutionGroupUUID="77cf1e31-f200-0000-0080-efe1b9d849dc"
 MessageFlowName="XMLflow" StartDate="2003-01-17"
 StartTime="14:44:34.581320" EndDate="2003-01-17" EndTime="14:44:44.582926"
 TotalElapsedTime="0"
 MaximumElapsedTime="0" MinimumElapsedTime="0" TotalCPUTime="0"
 MaximumCPUTime="0"MinimumCPUTime="0" CPUTimeWaitingForInputMessage="685"
 ElapsedTimeWaitingForInputMessage="10001425" TotalInputMessages="0"
 TotalSizeOfInputMessages="0" MaximumSizeOfInputMessages="0"
 MinimumSizeOfInputMessages="0" NumberOfThreadsInPool="1"
 TimesMaximumNumberOfThreadsReached="0" TotalNumberOfMQErrors="0"
 TotalNumberOfMessagesWithErrors="0" TotalNumberOfErrorsProcessingMessages="0"
 TotalNumberOfTimeOutsWaitingForRepliesToAggregateMessages="0"
 TotalNumberOfCommits="0" TotalNumberOfBackouts="0" AccoutingOrigin="DEPT1"/>


<Threads Number="1">
<ThreadStatistics Number="5" TotalNumberOfInputMessages="0"
TotalElapsedTime="0" TotalCPUTime="0" CPUTimeWaitingForInputMessage="685"
ElapsedTimeWaitingForInputMessage="10001425" TotalSizeOfInputMessages="0"
MaximumSizeOfInputMessages="0" MinimumSizeOfInputMessages="0"/>
</Threads>


<Nodes Number="3">
```

```
  <NodeStatistics Label="FAILQueue" Type="MQOutput" TotalElapsedTime="0"
    MaximumElapsedTime="0" MinimumElapsedTime="0" TotalCPUTime="0"
    MaximumCPUTime="0" MinimumCPUTime="0" CountOfInvocations="0"
    NumberOfInputTerminals="1" NumberOfOutputTerminals="2">
   <TerminalStatistics Label="failure" Type="Output" CountOfInvocations="0"/>
   <TerminalStatistics Label="in" Type="Input" CountOfInvocations="0"/>
   <TerminalStatistics Label="out" Type="Output" CountOfInvocations="0"/>
  </NodeStatistics>


  <NodeStatistics Label="INQueue3" Type="MQInput" TotalElapsedTime="0"
    MaximumElapsedTime="0" MinimumElapsedTime="0" TotalCPUTime="0"
    MaximumCPUTime="0" MinimumCPUTime="0"CountOfInvocations="0"
    NumberOfInputTerminals="0" NumberOfOutputTerminals="3">
   <TerminalStatistics Label="catch" Type="Output" CountOfInvocations="0"/>
   <TerminalStatistics Label="failure" Type="Output" CountOfInvocations="0"/>
   <TerminalStatistics Label="out" Type="Output" CountOfInvocations="0"/>
  </NodeStatistics>


  <NodeStatistics Label="OUTQueue" Type="MQOutput" TotalElapsedTime="0"
    MaximumElapsedTime="0" MinimumElapsedTime="0" TotalCPUTime="0"
    MaximumCPUTime="0" MinimumCPUTime="0" CountOfInvocations="0"
    NumberOfInputTerminals="1" NumberOfOutputTerminals="2">
    <TerminalStatistics Label="failure" Type="Output" CountOfInvocations="0"/>
    <TerminalStatistics Label="in" Type="Input" CountOfInvocations="0"/>
    <TerminalStatistics Label="out" Type="Output" CountOfInvocations="0"/>
   </NodeStatistics>


 </Nodes>


</WMQIStatisticsAccounting>
```

Related concepts:

"Message flow accounting and statistics data" on page 3281
Message flow accounting and statistics data is the information that can be collected by a broker to record performance and operating details of message flow execution.

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Monitoring message flow performance" on page 3279
You can configure your message flow to collect statistics about the operation and behavior of your message flows. Use this information to assess the performance of your message flows.

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

Related reference:

"Message flow accounting and statistics output formats" on page 6725
The message flow accounting and statistics data can be written in three formats.

Example of user trace entries for message flow accounting and statistics:

This example shows a user trace that contains message flow accounting and statistics data.

The following example shows what is generated for a snapshot report. The messages that are written to the trace show that the message flow is called

myExampleFlow, and that it is running in an execution group named *default* on broker *MQ01BRK*. The message flow contains the following nodes:
- An MQInput node called *inNode*
- A Compute node called *First1*
- An MQOutput node called *outNode*

The nodes are connected together (Out terminal to In terminal for each connection).

During the interval for which statistics have been collected, this message flow processed 150 input messages.

The records show that two threads are assigned to this message flow. One thread is assigned when the message flow is deployed (the default number); an additional thread (thread 0) listens on the input queue. The listening thread starts additional threads to process input messages that are dependent on the number of instances that you have configured for the message flow, and on the rate of arrival of the input messages on the input queue.

The following command has been issued to achieve these results:

```
mqsichangeflowstats MQ01BRK -s -c active -e default -f myExampleFlow -n advanced -t basic -b basic
```

The trace entries have been retrieved with the **mqsireadlog** command and formatted using the **mqsiformatlog** command. The output from **mqsiformatlog** is shown in the following example. Line breaks have been added to aid readability.

The broker takes information about statistics and accounting from the operating system. On some operating systems, such as Windows, UNIX, and Linux, rounding can occur because the system calls that are used to determine the processor times are not sufficiently granular. This rounding might affect the accuracy of the data.

```
BIP2380I: WMQI message flow statistics. ProcessID='328467', Key='6', Type='SnapShot', Reason='Snapshot',
BrokerLabel='MQ01BRK', BrokerUUID='18792e66-e100-0000-0080-f197e5ed81bd',
ExecutionGroupName='default', ExecutionGroupUUID='15d4314a-3607-11d4-8000-09140f7b0000',
MessageFlowName='myExampleFlow',
StartDate='2003-05-20', StartTime='13:44:31.885862',
EndDate='2003-05-20', EndTime='13:44:51.310080',
TotalElapsedTime='9414843', MaximumElapsedTime='1143442', MinimumElapsedTime='35154',
TotalCPUTime='760147', MaximumCPUTime='70729', MinimumCPUTime='3124',
CPUTimeWaitingForInputMessage='45501', ElapsedTimeWaitingForInputMessage='11106438',
TotalInputMessages='150', TotalSizeOfInputMessages='437250',
MaximumSizeOfInputMessages='2915', MinimumSizeOfInputMessages='2915',
NumberOfThreadsInPool='1', TimesMaximumNumberOfThreadsReached='150',
TotalNumberOfMQErrors='0', TotalNumberOfMessagesWithErrors='0',
TotalNumberOfErrorsProcessingMessages='0', TotalNumberOfTimeOuts='0',
TotalNumberOfCommits='150', TotalNumberOfBackouts='0', AccountingOrigin="DEPT2".
Statistical information for message flow 'myExampleFlow' in broker 'MQ01BRK'.
This is an information message produced by WMQI statistics.

BIP2381I: WMQI thread statistics. ProcessID='328467', Key='6', Number='0',
TotalNumberOfInputMessages='0',
TotalElapsedTime='0', TotalCPUTime='0', CPUTimeWaitingForInputMessage='110',
ElapsedTimeWaitingForInputMessage='5000529', TotalSizeOfInputMessages='0',
MaximumSizeOfInputMessages='0', MinimumSizeOfInputMessages='0'.
Statistical information for thread '0'.
This is an information message produced by WMQI statistics.

BIP2381I: WMQI thread statistics. ProcessID='328467', Key='6', Number='18',
TotalNumberOfInputMessages='150',
TotalElapsedTime='9414843', TotalCPUTime='760147', CPUTimeWaitingForInputMessage='45391',
ElapsedTimeWaitingForInputMessage='6105909', TotalSizeOfInputMessages='437250',
```

```
MaximumSizeOfInputMessages='2915', MinimumSizeOfInputMessages='2915'.
Statistical information for thread '18'.
This is an information message produced by WMQI statistics.

BIP2382I: WMQI node statistics. ProcessID='328467', Key='6',
Label='First1', Type='ComputeNode',
TotalElapsedTime='6428815', MaximumElapsedTime='138261', MinimumElapsedTime='28367',
TotalCPUTime='604060', MaximumCPUTime='69645', MinimumCPUTime='2115',
CountOfInvocations='150', NumberOfInputTerminals='1', NumberOfOutputTerminals='2'.
Statistical information for node 'First1'.
This is an information message produced by WMQI statistics.

BIP2383I: WMQI terminal statistics. ProcessID='328467', Key='6',
Label='failure', Type='Output', CountOfInvocations='0',
Statistical information for terminal 'failure'.
This is an information message produced by WMQI statistics.

BIP2383I: WMQI terminal statistics. ProcessID='328467', Key='6',
Label='in', Type='Input', CountOfInvocations='150',
Statistical information for terminal 'in'.
This is an information message produced by WMQI statistics.

BIP2383I: WMQI terminal statistics. ProcessID='328467', Key='6',
Label='out', Type='Output', CountOfInvocations='150',
Statistical information for terminal 'out'.
This is an information message produced by WMQI statistics.

BIP2382I: WMQI node statistics. ProcessID='328467', Key='6',
Label='inNode', Type='MQInputNode',
TotalElapsedTime='1813446', MaximumElapsedTime='1040209', MinimumElapsedTime='1767',
TotalCPUTime='70565', MaximumCPUTime='686', MinimumCPUTime='451',
CountOfInvocations='150', NumberOfInputTerminals='0', NumberOfOutputTerminals='3'.
Statistical information for node 'inNode'.
This is an information message produced by WMQI statistics.

BIP2383I: WMQI terminal statistics. ProcessID='328467', Key='6',
Label='catch', Type='Output', CountOfInvocations='0',
Statistical information for terminal 'catch'.
This is an information message produced by WMQI statistics.

BIP2383I: WMQI terminal statistics. ProcessID='328467', Key='6',
Label='failure', Type='Output', CountOfInvocations='0',
Statistical information for terminal 'failure'.
This is an information message produced by WMQI statistics.

BIP2383I: WMQI terminal statistics. ProcessID='328467', Key='6',
Label='out', Type='Output', CountOfInvocations='150',
Statistical information for terminal 'out'.
This is an information message produced by WMQI statistics.

BIP2382I: WMQI node statistics. ProcessID='328467', Key='6',
Label='outNode', Type='MQOutputNode',
TotalElapsedTime='1172582', MaximumElapsedTime='177516', MinimumElapsedTime='3339',
TotalCPUTime='85522', MaximumCPUTime='762', MinimumCPUTime='536',
CountOfInvocations='150', NumberOfInputTerminals='1', NumberOfOutputTerminals='2'.
Statistical information for node 'outNode'.
This is an information message produced by WMQI statistics.

BIP2383I: WMQI terminal statistics. ProcessID='328467', Key='6',
Label='failure', Type='Output', CountOfInvocations='0',
Statistical information for terminal 'failure'.
This is an information message produced by WMQI statistics.

BIP2383I: WMQI terminal statistics. ProcessID='328467', Key='6',
Label='in', Type='Input', CountOfInvocations='150',
Statistical information for terminal 'in'.
This is an information message produced by WMQI statistics.
```

```
BIP2383I: WMQI terminal statistics. ProcessID='328467', Key='6',
Label='out', Type='Output', CountOfInvocations='0',
Statistical information for terminal 'out'.
This is an information message produced by WMQI statistics.
```

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message flow accounting and statistics data" on page 3281
Message flow accounting and statistics data is the information that can be collected by a broker to record performance and operating details of message flow execution.

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Monitoring message flow performance" on page 3279
You can configure your message flow to collect statistics about the operation and behavior of your message flows. Use this information to assess the performance of your message flows.

Related reference:

"Message flow accounting and statistics output formats" on page 6725
The message flow accounting and statistics data can be written in three formats.

Metrics for accounting and statistics data in the WebSphere Message Broker Explorer

You can filter the metrics displayed for accounting and statistics data in the Broker Statistics and Broker Statistics Graph views in the WebSphere Message Broker Explorer.

Broker statistics

The metrics that you can select in the Broker Statistics Graph view for a broker are:

- Total number of input messages
- Total elapsed time
- Total processor time spent processing messages
- Processor time waiting for input message
- Elapsed time waiting for input message
- Total size of input messages
- Maximum size of input messages
- Minimum size of input messages

Execution group statistics

The metrics that you can select in the Broker Statistics Graph view for an execution group are:

- Total elapsed time
- Maximum elapsed time
- Minimum elapsed time
- Total processor time
- Maximum processor time
- Minimum processor time
- Processor time waiting for input message
- Elapsed time waiting for input message
- Total input messages
- Total size of input messages
- Maximum size of input messages
- Minimum size of input messages

- Number of threads in pool
- Times maximum number of threads reached
- Total number of WebSphere MQ errors
- Total number of messages with errors
- Total number of errors processing messages
- Total number of time-outs waiting for replies to aggregate messages
- Total number of commits
- Total number of messages backed out

Message flow statistics

The metrics that you can select in the Broker Statistics Graph view for a message flow are:

- Total elapsed time
- Maximum elapsed time
- Minimum elapsed time
- Total processor time
- Maximum processor time
- Minimum processor time
- Count of invocations
- Number of input terminals
- Number of output terminals

Related concepts:

"Message flow accounting and statistics data" on page 3281
Message flow accounting and statistics data is the information that can be collected by a broker to record performance and operating details of message flow execution.

Related tasks:

"Viewing message flow accounting and statistics data" on page 3300
You can use the Broker Statistics and Broker Statistics Graph views in the WebSphere Message Broker Explorer to view snapshot accounting and statistics data as it is produced by the broker.

"Starting accounting and statistics data collection in the WebSphere Message Broker Explorer" on page 3299
Use the WebSphere Message Broker Explorer to start collecting snapshot accounting and statistics data for your brokers, execution groups, and message flows. You can then view the accounting and statistics data in the Broker Statistics and Broker Statistics Graph views.

"Filtering message flow accounting and statistics data" on page 3302
You can select the metrics for the snapshot accounting and statistics data that are displayed in the Broker Statistics and Broker Statistics Graph views in the WebSphere Message Broker Explorer.

Resource statistics data

Learn about the measurements for which data is returned when you activate resource statistics collection.

Statistics information is collected for the following resource types:

- CICS
- CORBA
- File
- File Transfer Protocol
- FTEAgent
- IBM Sterling Connect:Direct

- Java Virtual Machine (JVM)
- JDBC connection pools
- ODBC
- Outbound sockets
- Parsers
- Security
- SOAP
- TCPIP Client Nodes
- TCPIP Server Nodes

You can view these statistics in the WebSphere Message Broker Explorer, or you can write a program that subscribes to a publication (single XML message) that returns this data. For an example of the publication message, see Example XML output.

The information collected, which is different for each type, is shown in the following tables.

Related concepts:

"Resource statistics" on page 3306
Resource statistics are collected by a broker to record performance and operating details of resources that are used by execution groups.

Related tasks:

"Monitoring resource performance" on page 3305
You can collect statistics to assess the performance of certain resources used by execution groups.

"Viewing resource statistics data in the WebSphere Message Broker Explorer" on page 3313
Use the WebSphere Message Broker Explorer to view resource statistics data for your execution groups in the Broker Resources and Broker Resources Graph views.

"Subscribing to statistics reports" on page 3317
You can subscribe to topics that return statistics about the operation of your message flows and resource managers.

Related reference:

"**mqsichangeresourcestats** command" on page 3819
Use the **mqsichangeresourcestats** command to control statistics gathering for resources in the broker.

"**mqsireportresourcestats** command" on page 3944
Use the **mqsireportresourcestats** command to display current statistics gathering options for resources in the broker.

Example of an XML publication for resource statistics

This example message shows an XML publication that contains resource statistics data.

A single XML publication includes all the statistics information that is reported for the resource types listed in "Resource statistics data" on page 6745. The publication message starts with information that identifies the broker and the execution group to which the statistics apply.

For information about subscribing to these publication messages, see "Subscribing to statistics reports" on page 3317.

The following message is an example that shows three sections, one for each of three different resource types, and the values that are returned for each type. The output lines in this example have been split to improve readability.

```
<ResourceStatistics
  brokerLabel="STRESS1"
  brokerUUID="1a09649b-c9b9-4efe-a9c0-5275f6dd6c3f"
  executionGroupName="eg.EAS.JVM.1"
  executionGroupUUID="d6a3b481-2401-0000-0080-c5acc915fa62"
  collectionStartDate="2009-10-22" collectionStartTime="11:42:17"
  startDate="2009-10-23" startTime="17:59:36.152"
  endDate="2009-10-23" endTime="17:59:56.154"
  timezone="Europe/London">
  <ResourceType name="JVM">
    <resourceIdentifier name="summary"
      InitialMemoryInMB="32" UsedMemoryInMB="30"
      CommittedMemoryInMB="52" MaxMemoryInMB="-1"
      CumulativeGCTimeInSeconds="0"
      CumulativeNumberOfGCCollections="32"/>
    <resourceIdentifier name="Heap Memory"
      InitialMemoryInMB="32" UsedMemoryInMB="14"
      CommittedMemoryInMB="32" MaxMemoryInMB="256"/>
    <resourceIdentifier name="Non-Heap Memory"
      InitialMemoryInMB="0" UsedMemoryInMB="16"
      CommittedMemoryInMB="20" MaxMemoryInMB="-1"/>
    <resourceIdentifier name="Garbage Collection - J9 GC"
      CumulativeGCTimeInSeconds="0"
      CumulativeNumberOfGCCollections="32"/>
  </ResourceType>

  <ResourceType name="Security">
    <resourceIdentifier name="summary"
      TotalCacheEntries="10"
      TotalOperations="9"
      TotalSuccessfulOperations="6"
      TotalOperationsServicedByCache="3"/>
    <resourceIdentifier name="LDAP"
      TotalOperations="3"
      TotalSuccessfulOperations="1"
      TotalOperationsServicedByCache="0"
      TotalCacheEntries="0"/>
    <resourceIdentifier name="WS-Trust v1.3 STS"
      TotalOperations="6"
      TotalSuccessfulOperations="5"
      TotalOperationsServicedByCache="3"
      TotalCacheEntries="0"/>
  </ResourceType>
  <ResourceType name="Sockets">
    <resourceIdentifier name="summary"
      TotalMessages="826"
      TotalSocketsOpened="0"
      AverageSocketsOpenedPerMinute="0"
      TotalBytesSent="1715189"
      AverageBytesSentPerSecond="85758"
      TotalBytesReceived="116879"
      AverageBytesReceivedPerSecond="5843"
      AverageBytesSentPerMessage="2076"
      AverageBytesReceivedPerMessage="141"
      SentMessageSize_0-1KB="0"
      SentMessageSize_1KB-10KB="826"
      SentMessageSize_10KB-100KB="0"
      SentMessageSize_100KB-1MB="0"
      SentMessageSize_1MB-10MB="0"
      SentMessageSize_Over10MB="0"
      ReceivedMessageSize_0-1KB="826"
      ReceivedMessageSize_1KB-10KB="0"
      ReceivedMessageSize_10KB-100KB="0"
```

```
      ReceivedMessageSize_100KB-1MB="0"
      ReceivedMessageSize_1MB-10MB="0"
      ReceivedMessageSize_Over10MB="0"/>
    <resourceIdentifier name="9.146.149.86.7900"
      TotalMessages="413"
      AverageMessagesPerMinute="1239"
      TotalSocketsOpened="0"
      AverageSocketsOpenedPerMinute="0"
      TotalBytesSent="837977"
      AverageBytesSentPerSecond="41898"
      TotalBytesReceived="60298"
      AverageBytesReceivedPerSecond="3014"
      AverageBytesSentPerMessage="2029"
      AverageBytesReceivedPerMessage="146"
      SentMessageSize_0-1KB="0"
      SentMessageSize_1KB-10KB="413"
      SentMessageSize_10KB-100KB="0"
      SentMessageSize_100KB-1MB="0"
      SentMessageSize_1MB-10MB="0"
      SentMessageSize_Over10MB="0"
      ReceivedMessageSize_0-1KB="413"
      ReceivedMessageSize_1KB-10KB="0"
      ReceivedMessageSize_10KB-100KB="0"
      ReceivedMessageSize_100KB-1MB="0"
      ReceivedMessageSize_1MB-10MB="0"
      ReceivedMessageSize_Over10MB="0"/>
    <resourceIdentifier name="localhost.7900"
      TotalMessages="413"
      AverageMessagesPerMinute="1239"
      TotalSocketsOpened="0"
      AverageSocketsOpenedPerMinute="0"
      TotalBytesSent="877212"
      AverageBytesSentPerSecond="43860"
      TotalBytesReceived="56581"
      AverageBytesReceivedPerSecond="2829"
      AverageBytesSentPerMessage="2124"
      AverageBytesReceivedPerMessage="137"
      SentMessageSize_0-1KB="0"
      SentMessageSize_1KB-10KB="413"
      SentMessageSize_10KB-100KB="0"
      SentMessageSize_100KB-1MB="0"
      SentMessageSize_1MB-10MB="0"
      SentMessageSize_Over10MB="0"
      ReceivedMessageSize_0-1KB="413"
      ReceivedMessageSize_1KB-10KB="0"
      ReceivedMessageSize_10KB-100KB="0"
      ReceivedMessageSize_100KB-1MB="0"
      ReceivedMessageSize_1MB-10MB="0"
      ReceivedMessageSize_Over10MB="0"/>
  </ResourceType>
  <ResourceType name="Message parsers">
    <resourceIdentifier name="summary"
      Threads="1"
      ApproxMemKB="3"
      MaxReadKB="1"
      MaxWrittenKB="0"
      Fields="100"
      Reads="1"
      FailedReads="0"
      Writes="0"
      FailedWrites="0"/>
    <resourceIdentifier name="Flow1.XMLNSC"
      Threads="1"
      ApproxMemKB="3"
      MaxReadKB="1"
      MaxWrittenKB="0"
      Fields="100"
```

```
      Reads="1"
      FailedReads="0"
      Writes="0"
      FailedWrites="0"/>
  </ResourceType>
</ResourceStatistics>
```

All times are reported in local time, and the timezone used is reported by the timezone attribute in the standard TZ Area/Location format. The date is in the format yyyy-MM-dd, and time is in the format HH:mm:ss, independent of locale and timezone.

The collectionStartDate and collectionStartTime specify when resource statistics collection started, typically when the execution group was last started.

The endDate and endTime specify when the statistics data in the current message was gathered. The startDate and startTime is when the previous set of data was reported.

Related concepts:

"Resource statistics" on page 3306
Resource statistics are collected by a broker to record performance and operating details of resources that are used by execution groups.

Related tasks:

"Monitoring resource performance" on page 3305
You can collect statistics to assess the performance of certain resources used by execution groups.

"Subscribing to statistics reports" on page 3317
You can subscribe to topics that return statistics about the operation of your message flows and resource managers.

Related reference:

"**mqsichangeresourcestats** command" on page 3819
Use the **mqsichangeresourcestats** command to control statistics gathering for resources in the broker.

"**mqsireportresourcestats** command" on page 3944
Use the **mqsireportresourcestats** command to display current statistics gathering options for resources in the broker.

Resource statistics data: CICS

Learn about the data that is returned for the CICS resource type when you activate resource statistics collection.

You can view these statistics in the WebSphere Message Broker Explorer, or you can write a program that subscribes to a publication (single XML message) that returns this data. For an example of the publication message, see Example XML output.

A CICSRequest node calls CICS Transaction Server for z/OS programs. Use these resource statistics to review how many successful and unsuccessful calls the node is making to CICS, how many unsuccessful requests are related to security issues such as authentication failures, and how often connection attempts fail.

A statistics summary is returned for the whole execution group, followed by a breakdown for each named CICS Transaction Server for z/OS region.

The following table describes the measurements that are returned for each CICS region. Each row of statistics represents a CICS region, which is identified either by a configurable service, or by the CICSRequest node `CICS server` URL.

- In this example, the configurable service is identified by *myCICSConnectionService*, and the broker is identified by APPLID*BRKApp* and qualifier *BRKQual*:

  myCICSConnectionService.BRKApp.BRKQual

- In this example, the CICSRequest node `CICS server` URL is identified by *tcp://mycicsregion.com:12345*:

  tcp://mycicsregion.com:12345

Measurements	Description
RequestSuccess	The number of requests to CICS that are successful.
RequestFailures	The total number of requests to CICS that fail. The value that is returned by this measurement does not include connection failures. Connection failure values are returned by the `ConnectionAttemptFailures` measurement.
RequestSecurityFailures	The number of failed requests to CICS that were caused by security issues, such as authentication failures.
ConnectionAttemptFailures	The number of connection attempts that fail.

Related concepts:

"Resource statistics" on page 3306
Resource statistics are collected by a broker to record performance and operating details of resources that are used by execution groups.

Related tasks:

"Monitoring resource performance" on page 3305
You can collect statistics to assess the performance of certain resources used by execution groups.

"Viewing resource statistics data in the WebSphere Message Broker Explorer" on page 3313
Use the WebSphere Message Broker Explorer to view resource statistics data for your execution groups in the Broker Resources and Broker Resources Graph views.

"Subscribing to statistics reports" on page 3317
You can subscribe to topics that return statistics about the operation of your message flows and resource managers.

Related reference:

"Resource statistics data" on page 6745
Learn about the measurements for which data is returned when you activate resource statistics collection.

"**mqsichangeresourcestats** command" on page 3819
Use the **mqsichangeresourcestats** command to control statistics gathering for resources in the broker.

"**mqsireportresourcestats** command" on page 3944
Use the **mqsireportresourcestats** command to display current statistics gathering options for resources in the broker.

"CICSRequest node" on page 4321
Use the CICSRequest node to call CICS Transaction Server for z/OS programs over TCP/IP-based IP InterCommunications (IPIC) protocol.

Resource statistics data: CORBA

Learn about the data that is returned for the CORBA resource type when you activate resource statistics collection.

You can view these statistics in the WebSphere Message Broker Explorer, or you can write a program that subscribes to a publication (single XML message) that returns this data. For an example of the publication message, see Example XML output.

A CORBARequest node calls a CORBA server. Use these resource statistics to review how many calls the node is making to the CORBA server, and how many of those calls are successful or result in CORBA exceptions. A statistics summary is returned for the whole execution group.

The following table describes the measurements that are returned for each CORBA node. The measurements apply to the number of calls since the execution group started.

Measurements	Description
OutboundInvocations	The total number of calls made to a CORBA server.
OutboundSuccessfulInvocations	The number of calls to the CORBA server that are successful.
OutboundCorbaExceptions	The number of calls to the CORBA server that result in CORBA exceptions.

Related concepts:

"Resource statistics" on page 3306
Resource statistics are collected by a broker to record performance and operating details of resources that are used by execution groups.

Related tasks:

"Monitoring resource performance" on page 3305
You can collect statistics to assess the performance of certain resources used by execution groups.

"Viewing resource statistics data in the WebSphere Message Broker Explorer" on page 3313
Use the WebSphere Message Broker Explorer to view resource statistics data for your execution groups in the Broker Resources and Broker Resources Graph views.

"Subscribing to statistics reports" on page 3317
You can subscribe to topics that return statistics about the operation of your message flows and resource managers.

Related reference:

"Resource statistics data" on page 6745
Learn about the measurements for which data is returned when you activate resource statistics collection.

"**mqsichangeresourcestats** command" on page 3819
Use the **mqsichangeresourcestats** command to control statistics gathering for resources in the broker.

"**mqsireportresourcestats** command" on page 3944
Use the **mqsireportresourcestats** command to display current statistics gathering options for resources in the broker.

"CORBARequest node" on page 4349
Use the CORBARequest node to call an external CORBA application over Internet

Inter-Orb Protocol (IIOP).

Resource statistics data: File

Learn about the data that is returned for the file resource type when you activate resource statistics collection.

You can view these statistics in the WebSphere Message Broker Explorer, or you can write a program that subscribes to a publication (single XML message) that returns this data. For an example of the publication message, see Example XML output.

A summary is displayed for the local file system of any file actions done by any type of file node. It does not have any other entries; there is only one local file system. Remote file systems are reported in FTEAgent and FTP resource statistics.

Measurements	Description
FilesRead	The number of files successfully read by any file node
RecordsRead	The number of records read by either a file input node, FTE input node, or a file read node
BytesRead	The total number of bytes read by either a file input node, FTE input node, or a file read node
FilesCreated	The number of files successfully created by a file output node
RecordsWritten	The number of records written to files by a file output node or FTE output node
BytesWritten	The total number of bytes written by a file output node or FTE output node

Related concepts:

"Resource statistics" on page 3306
Resource statistics are collected by a broker to record performance and operating details of resources that are used by execution groups.

"Managed file transfers using WebSphere MQ File Transfer Edition" on page 1869
Transfer files, with file transfer metadata, in a timely and reliable manner.

Related tasks:

"Monitoring resource performance" on page 3305
You can collect statistics to assess the performance of certain resources used by execution groups.

"Viewing resource statistics data in the WebSphere Message Broker Explorer" on page 3313
Use the WebSphere Message Broker Explorer to view resource statistics data for your execution groups in the Broker Resources and Broker Resources Graph views.

"Subscribing to statistics reports" on page 3317
You can subscribe to topics that return statistics about the operation of your message flows and resource managers.

Related reference:

"Resource statistics data" on page 6745
Learn about the measurements for which data is returned when you activate resource statistics collection.

"**mqsichangeresourcestats** command" on page 3819
Use the **mqsichangeresourcestats** command to control statistics gathering for resources in the broker.

"**mqsireportresourcestats** command" on page 3944
Use the **mqsireportresourcestats** command to display current statistics gathering options for resources in the broker.

Resource statistics data: File Transfer Protocol

Learn about the data that is returned for the File Transfer Protocol resource type when you activate resource statistics collection.

You can view these statistics in the WebSphere Message Broker Explorer, or you can write a program that subscribes to a publication (single XML message) that returns this data. For an example of the publication message, see Example XML output.

A summary is displayed of file transfers that occur in the execution and a list for each server used.

Measurements	Description
Protocol	The protocol used: either FTP or SFTP
FTPGets	The number of transfers from a remote server to the file system of the broker
BytesReceived	The number of bytes transferred from a remote server to the file system of the broker
FTPPuts	The number of transfers from the file system of the broker to a remote server
BytesSent	The number of bytes transferred from the file system of the broker to a remote server **Note:** The FTPPuts counter can increase in two cases: either when a new file is transferred, or when an append is made to an existing file.

Related concepts:

"Resource statistics" on page 3306
Resource statistics are collected by a broker to record performance and operating details of resources that are used by execution groups.

"Managed file transfers using WebSphere MQ File Transfer Edition" on page 1869
Transfer files, with file transfer metadata, in a timely and reliable manner.

Related tasks:

"Monitoring resource performance" on page 3305
You can collect statistics to assess the performance of certain resources used by execution groups.

"Viewing resource statistics data in the WebSphere Message Broker Explorer" on page 3313
Use the WebSphere Message Broker Explorer to view resource statistics data for your execution groups in the Broker Resources and Broker Resources Graph views.

"Subscribing to statistics reports" on page 3317
You can subscribe to topics that return statistics about the operation of your message flows and resource managers.

Related reference:

"Resource statistics data" on page 6745
Learn about the measurements for which data is returned when you activate resource statistics collection.

"**mqsichangeresourcestats** command" on page 3819
Use the **mqsichangeresourcestats** command to control statistics gathering for resources in the broker.

"**mqsireportresourcestats** command" on page 3944
Use the **mqsireportresourcestats** command to display current statistics gathering options for resources in the broker.

Resource statistics data: FTEAgent

Learn about the data that is returned for the FTEAgent resource type when you activate resource statistics collection.

You can view these statistics in the WebSphere Message Broker Explorer, or you can write a program that subscribes to a publication (single XML message) that returns this data. For an example of the publication message, see Example XML output.

The FTE agent is an embedded FTE component that allows WebSphere Message Broker to send and receive files by using WebSphere MQ File Transfer Edition. A one-to-one mapping exists between an execution group and its FTE agent. Therefore, the resource manager statistics provide an accurate picture of the WebSphere MQ File Transfer Edition activity of that execution group.

The following table describes the measurements that are returned for the agent.

Measurements	Description
inboundTransfers	The number of transfers received by the agent.
outboundTransfers	The number of transfers sent by the agent.
inboundBytes	The number of bytes received by the agent.
outboundBytes	The number of bytes sent by the agent.

Related concepts:

"Resource statistics" on page 3306
Resource statistics are collected by a broker to record performance and operating details of resources that are used by execution groups.

"Managed file transfers using WebSphere MQ File Transfer Edition" on page 1869
Transfer files, with file transfer metadata, in a timely and reliable manner.

Related tasks:

"Monitoring resource performance" on page 3305
You can collect statistics to assess the performance of certain resources used by execution groups.

"Viewing resource statistics data in the WebSphere Message Broker Explorer" on page 3313
Use the WebSphere Message Broker Explorer to view resource statistics data for your execution groups in the Broker Resources and Broker Resources Graph views.

"Subscribing to statistics reports" on page 3317
You can subscribe to topics that return statistics about the operation of your message flows and resource managers.

Related reference:

"Resource statistics data" on page 6745
Learn about the measurements for which data is returned when you activate resource statistics collection.

"**mqsichangeresourcestats** command" on page 3819
Use the **mqsichangeresourcestats** command to control statistics gathering for resources in the broker.

"**mqsireportresourcestats** command" on page 3944
Use the **mqsireportresourcestats** command to display current statistics gathering options for resources in the broker.

Resource statistics data: IBM Sterling Connect:Direct

Learn about the data that is returned for the CDServer resource type when you activate resource statistics collection.

You can view these statistics in the WebSphere Message Broker Explorer, or you can write a program that subscribes to a publication (single XML message) that returns this data. For an example of the publication message, see Example XML output.

Statistics are available for all Connect:Direct server activity, together with one line for each configurable service used.

The following table describes the measurements that are returned.

Measurements	Description
connectionDetails	The host name and port for the Connect:Direct server that is being connected to, in the format `<hostname>:port`. For the summary line, this property is set to an empty string.
inboundTransfers	The number of transfers received by the Connect:Direct server selected for processing in a message flow.
inboundBytes	Total number of bytes received in transfers, selected from Connect:Direct server.
nohitTransfers	The number of transfers received by the Connect:Direct server not selected for processing in a message flow. Reasons for not processing a transfer include: • The WebSphere Message Broker system has no access to the transferred file. • A deployed CDInput node has a filter that matches the transferred file. This number gives the total for the whole broker and not just this execution group.
outboundTransfers	The number of transfers sent to the Connect:Direct server.
outboundBytes	Total number of bytes contained in transfers sent to the Connect:Direct server.

Related concepts:

"Resource statistics" on page 3306
Resource statistics are collected by a broker to record performance and operating details of resources that are used by execution groups.

Related tasks:

"Monitoring resource performance" on page 3305
You can collect statistics to assess the performance of certain resources used by execution groups.

"Viewing resource statistics data in the WebSphere Message Broker Explorer" on page 3313
Use the WebSphere Message Broker Explorer to view resource statistics data for your execution groups in the Broker Resources and Broker Resources Graph views.

"Subscribing to statistics reports" on page 3317
You can subscribe to topics that return statistics about the operation of your message flows and resource managers.

Related reference:

"Resource statistics data" on page 6745
Learn about the measurements for which data is returned when you activate resource statistics collection.

"**mqsichangeresourcestats** command" on page 3819
Use the **mqsichangeresourcestats** command to control statistics gathering for resources in the broker.

"**mqsireportresourcestats** command" on page 3944
Use the **mqsireportresourcestats** command to display current statistics gathering options for resources in the broker.

Resource statistics data: Java Virtual Machine (JVM)

Learn about the data that is returned for the JVM resource type when you activate resource statistics collection.

You can view these statistics in the WebSphere Message Broker Explorer, or you can write a program that subscribes to a publication (single XML message) that returns this data. For an example of the publication message, see Example XML output.

Each execution group starts its own Java Virtual Machine (JVM), which provides support for all Java activities in that execution group. Some Java activity is present in the execution group even if you have not yet deployed or started message flows in that group. Use these resource statistics to review how much memory is in use by the JVM, and how often garbage collection might be occurring in the execution group.

Statistics are collected for the following JVM resources:
- Heap memory
- Non-heap memory
- Garbage collection

A summary is also provided, which adds the values in the three groups.

The special value `-1` is returned for measurements that are undefined or are not set.

The following table describes the measurements that are returned for each of the listed resources. The garbage collection measurements are cumulative, and continue to increase until you stop statistics collection.

Measurements	Resource	Description
InitialMemoryInMB	• Heap memory • Non-heap memory	The initial amount of memory that the JVM requests from the operating system for memory management during startup. Its value might be undefined.

Measurements	Resource	Description
UsedMemoryinMB	• Heap memory • Non-heap memory	The amount of memory that is currently in use.
CommittedMemoryInMB	• Heap memory • Non-heap memory	The amount of memory that is allocated to the JVM by the operating system.
MaxMemoryInMB	• Heap memory • Non-heap memory	The maximum amount of memory that can be used for memory management. Its value might be undefined.
CumulativeGCTimeInSeconds	Garbage collection	The accumulated garbage collection elapsed time in seconds for this instance of the JVM. Its value might be undefined.
CumulativeNumberOfGCCollections	Garbage collection	The total number of garbage collections that have occurred for this instance of the JVM. Its value might be undefined.

Related concepts:

"Resource statistics" on page 3306
Resource statistics are collected by a broker to record performance and operating details of resources that are used by execution groups.

Related tasks:

"Monitoring resource performance" on page 3305
You can collect statistics to assess the performance of certain resources used by execution groups.

"Viewing resource statistics data in the WebSphere Message Broker Explorer" on page 3313
Use the WebSphere Message Broker Explorer to view resource statistics data for your execution groups in the Broker Resources and Broker Resources Graph views.

"Subscribing to statistics reports" on page 3317
You can subscribe to topics that return statistics about the operation of your message flows and resource managers.

Related reference:

"Resource statistics data" on page 6745
Learn about the measurements for which data is returned when you activate resource statistics collection.

"**mqsichangeresourcestats** command" on page 3819
Use the **mqsichangeresourcestats** command to control statistics gathering for resources in the broker.

"**mqsireportresourcestats** command" on page 3944
Use the **mqsireportresourcestats** command to display current statistics gathering options for resources in the broker.

Resource statistics data: JDBC connection pools

Learn about the data that is returned for the JDBC connection pools resource type when you activate resource statistics collection.

You can view these statistics in the WebSphere Message Broker Explorer, or you can write a program that subscribes to a publication (single XML message) that returns this data. For an example of the publication message, see Example XML output.

A statistics summary is returned for each JDBC provider defined in the broker configurable service that is switched to use connection pools. The following examples are named JDBC data sources:

- DB2
- Oracle
- Microsoft SQLServer

The following table describes the measurements that are returned for each JDBC Provider configurable service.

Measurements	Description
NameOfJDBCProvider	The name of the JDBCProviders configurable service that is using connection pooling. If there are no connection pools started, this defaults to `none`.
MaxSizeOfPool	The maximum size of the connection pool.
ActualSizeOfPool	A snapshot of the number of connections currently in the pool when the statistics were reported.
CumulativeRequests	A count of the number of requests received by the connection pool during this accounting period.
CumulativeDelayedRequests	The number of times a request for a connection could not be satisfied immediately, because the number of allocated connections reached the maximum pool size and no connections are currently available.
MaxDelayInMilliseconds	The maximum time a caller waited for a connection to be allocated, in milliseconds.
CumulativeTimedOutRequests	A count of the number of requests for connections that could not be satisfied within 15 seconds.

Related concepts:

"Resource statistics" on page 3306
Resource statistics are collected by a broker to record performance and operating details of resources that are used by execution groups.

Related tasks:

"Monitoring resource performance" on page 3305
You can collect statistics to assess the performance of certain resources used by execution groups.

"Viewing resource statistics data in the WebSphere Message Broker Explorer" on page 3313
Use the WebSphere Message Broker Explorer to view resource statistics data for your execution groups in the Broker Resources and Broker Resources Graph views.

"Subscribing to statistics reports" on page 3317
You can subscribe to topics that return statistics about the operation of your message flows and resource managers.

Related reference:

"Resource statistics data" on page 6745
Learn about the measurements for which data is returned when you activate resource statistics collection.

"**mqsichangeresourcestats** command" on page 3819
Use the **mqsichangeresourcestats** command to control statistics gathering for resources in the broker.

"**mqsireportresourcestats** command" on page 3944
Use the **mqsireportresourcestats** command to display current statistics gathering options for resources in the broker.

Resource statistics data: ODBC

Learn about the data that is returned for the ODBC resource type when you activate resource statistics collection.

You can view these statistics in the WebSphere Message Broker Explorer, or you can write a program that subscribes to a publication (single XML message) that returns this data. For an example of the publication message, see Example XML output.

Statistics are reported for each ODBC DSN that was accessed since the execution group started. XA connections are distinguished from non-XA connections from the same DSN. Each resource is named either *DSN* or *DSN* (XA). This information is published on topic `$SYS/Broker/brokerName/ResourceStatistics/executionGroupLabel`.

Measurements	Description
ExecuteSuccess	The total number of times any statement was run against this DSN.
ExecuteFailure	The total number of times any statement failed against this DSN.
ActiveConnections	The number of connections currently open to this DSN.
ClosedConnections	The number of connections to this DSN that were ever open, but are now closed. This figure includes connections closed due to an error, forced closed by the DBMS or closed by broker because it was no longer required (for example, thread idle for 60 seconds).
ConnectionErrors	The number of times a connection to this DSN was detected to have a connection error (which would have caused the error to be closed and therefore also contributed to the closed connections measurement).

Related concepts:

"Resource statistics" on page 3306
Resource statistics are collected by a broker to record performance and operating details of resources that are used by execution groups.

Related tasks:

"Monitoring resource performance" on page 3305
You can collect statistics to assess the performance of certain resources used by execution groups.

"Viewing resource statistics data in the WebSphere Message Broker Explorer" on page 3313
Use the WebSphere Message Broker Explorer to view resource statistics data for your execution groups in the Broker Resources and Broker Resources Graph views.

"Subscribing to statistics reports" on page 3317
You can subscribe to topics that return statistics about the operation of your message flows and resource managers.

Related reference:

"Resource statistics data" on page 6745
Learn about the measurements for which data is returned when you activate resource statistics collection.

"**mqsichangeresourcestats** command" on page 3819
Use the **mqsichangeresourcestats** command to control statistics gathering for resources in the broker.

"**mqsireportresourcestats** command" on page 3944
Use the **mqsireportresourcestats** command to display current statistics gathering options for resources in the broker.

Resource statistics data: Outbound sockets

Learn about the data that is returned for the sockets resource type when you activate resource statistics collection.

You can view these statistics in the WebSphere Message Broker Explorer, or you can write a program that subscribes to a publication (single XML message) that returns this data. For an example of the publication message, see Example XML output.

Outbound sockets are used by the execution group when a message is sent out through a SOAP, SCA, or HTTP request node. SOAP and SCA nodes always use keepalive sockets; therefore outbound sockets are reused for many requests. HTTPRequest nodes use keepalive sockets only if you set the property Enable HTTP/1.1 keep-alive. Use these resource statistics to review whether you are reusing outbound sockets, and to see the size and volume of message data that is flowing through those sockets.

A statistics summary is returned, followed by an entry for each outbound socket endpoint, which is defined by its URL. Examples of endpoints are `localhost:7080` and `www.soaphub.org:80`.

The following table describes the measurements that are returned for each endpoint that was accessed since the execution group was last restarted.

Measurements	Description
TotalSockets	The number of outbound sockets that have been opened since the last execution group restart.
TotalMessages	The number of requests for a socket; for example, from a SOAPRequest node.
TotalDataSent_KB	The number of bytes sent, in kilobytes (KB).
TotalDataReceived_KB	The number of bytes received, in kilobytes (KB).
SentMessageSize_0-1KB SentMessageSize_1KB-10KB SentMessageSize_10KB-100KB SentMessageSize_100KB-1MB SentMessageSize_1MB-10MB SentMessageSize_Over10MB	The number of messages sent in each size range. For example, a message of 999 bytes is counted in SentMessageSize_0-1KB; a message of 1000 bytes is counted in SentMessageSize_1KB-10KB.
ReceivedMessageSize_0-1KB ReceivedMessageSize_1KB-10KB ReceivedMessageSize_10KB-100KB ReceivedMessageSize_100KB-1MB ReceivedMessageSize_1MB-10MB ReceivedMessageSize_Over10MB	The number of messages received in each size range.

Related concepts:

"Resource statistics" on page 3306
Resource statistics are collected by a broker to record performance and operating details of resources that are used by execution groups.

Related tasks:

"Monitoring resource performance" on page 3305
You can collect statistics to assess the performance of certain resources used by execution groups.

"Viewing resource statistics data in the WebSphere Message Broker Explorer" on page 3313
Use the WebSphere Message Broker Explorer to view resource statistics data for your execution groups in the Broker Resources and Broker Resources Graph views.

"Subscribing to statistics reports" on page 3317
You can subscribe to topics that return statistics about the operation of your message flows and resource managers.

Related reference:

"Resource statistics data" on page 6745
Learn about the measurements for which data is returned when you activate resource statistics collection.

"**mqsichangeresourcestats** command" on page 3819
Use the **mqsichangeresourcestats** command to control statistics gathering for resources in the broker.

"**mqsireportresourcestats** command" on page 3944
Use the **mqsireportresourcestats** command to display current statistics gathering options for resources in the broker.

Resource statistics data: Parsers

Learn about the data that is returned for the parsers resource type when you activate resource statistics collection.

You can view these statistics in the WebSphere Message Broker Explorer, or you can write a program that subscribes to a publication (single XML message) that returns this data. For an example of the publication message, see Example XML output.

All message flows in an execution group create parsers to parse and write input and output messages. Use the Parsers statistics to see how much resource is being used by the message trees and bit streams that these parsers own.

A statistics summary is returned, followed by an entry for accumulation by parser type for each message flow. The rows are named in the style *<Message Flow>.<Parser>*. A row is shown for every parser type used by that message flow. Additional instances are included in the accumulated statistics for each message flow.

The following table describes the statistics that are returned for each message flow parser since the execution group was last restarted.

Measurements	Description
Threads	The number of message flow threads that contributed to the statistics for a message flows parser type accumulation.
ApproxMemKB	The approximate amount of user data-related memory used for the named message flow parser type. It is not possible to calculate the exact amount of memory used by a parser.
MaxReadKB	Shows the largest bit stream parsed by the parser type for the named message flow.

Measurements	Description
MaxWrittenKB	Shows the largest bit stream written by the parser type for the named message flow.
Fields	Shows the number of message fields associated with the named message flow parser type. These fields are retained by the parser and are used for constructing the message trees.
Reads	The number of successful parses that were completed by the named message flow parser type.
FailedReads	The number of failed parses that occurred for the named message flow parser type.
Writes	The number of successful writes that were completed by the named message flow parser type.
FailedWrites	The number of failed writes that occurred for the named message flow parser type.

Related concepts:

"Resource statistics" on page 3306
Resource statistics are collected by a broker to record performance and operating details of resources that are used by execution groups.

Related tasks:

"Monitoring resource performance" on page 3305
You can collect statistics to assess the performance of certain resources used by execution groups.

"Viewing resource statistics data in the WebSphere Message Broker Explorer" on page 3313
Use the WebSphere Message Broker Explorer to view resource statistics data for your execution groups in the Broker Resources and Broker Resources Graph views.

"Subscribing to statistics reports" on page 3317
You can subscribe to topics that return statistics about the operation of your message flows and resource managers.

Related reference:

"Resource statistics data" on page 6745
Learn about the measurements for which data is returned when you activate resource statistics collection.

"**mqsichangeresourcestats** command" on page 3819
Use the **mqsichangeresourcestats** command to control statistics gathering for resources in the broker.

"**mqsireportresourcestats** command" on page 3944
Use the **mqsireportresourcestats** command to display current statistics gathering options for resources in the broker.

Resource statistics data: Security

Learn about the data that is returned for the security resource type when you activate resource statistics collection.

You can view these statistics in the WebSphere Message Broker Explorer, or you can write a program that subscribes to a publication (single XML message) that returns this data. For an example of the publication message, see Example XML output.

When a message flow is configured with a security profile, requests are typically made to a security provider or security token server (STS) to process and approve authentication, mapping, or authorization. Use the security resource statistics to review the number of requests that are made, how many of those requests are successful, and how many are being serviced from the security cache.

A statistics summary is returned for the whole execution group, followed by a breakdown for each named security provider. Examples of named security providers are WS-Trust v1.3 STS and LDAP.

The following table describes the measurements that are returned for each provider.

Measurements	Description
TotalOperations	The number of security operations (authentication, mapping, or authorization) since collection started. A security profile with both authentication and authorization counts as two operations.
TotalSuccessfulOperations	The number of security operations (authentication, mapping, or authorization) that were approved.
TotalOperationsServicedByCache	The number of security operations (authentication, mapping, or authorization) that were serviced from the security cache (without accessing the STS directly).
TotalCacheEntries	The total number of security operation result entries in the security cache. A security operation is defined in the security profile as authentication, mapping, or authorization. A cache entry might include a returned security token.

Related concepts:

"Resource statistics" on page 3306
Resource statistics are collected by a broker to record performance and operating details of resources that are used by execution groups.

Related tasks:

"Monitoring resource performance" on page 3305
You can collect statistics to assess the performance of certain resources used by execution groups.

"Viewing resource statistics data in the WebSphere Message Broker Explorer" on page 3313
Use the WebSphere Message Broker Explorer to view resource statistics data for your execution groups in the Broker Resources and Broker Resources Graph views.

"Subscribing to statistics reports" on page 3317
You can subscribe to topics that return statistics about the operation of your message flows and resource managers.

Related reference:

"Resource statistics data" on page 6745
Learn about the measurements for which data is returned when you activate resource statistics collection.

"**mqsichangeresourcestats** command" on page 3819
Use the **mqsichangeresourcestats** command to control statistics gathering for resources in the broker.

"**mqsireportresourcestats** command" on page 3944
Use the **mqsireportresourcestats** command to display current statistics gathering

options for resources in the broker.

Resource statistics data: SOAP

Learn about the data that is returned for the SOAPInbound resource type when you activate resource statistics collection.

You can view these statistics in the WebSphere Message Broker Explorer, or you can write a program that subscribes to a publication (single XML message) that returns this data. For an example of the publication message, see Example XML output.

The SOAPInput and SOAPReply nodes send and receive SOAP messages. Use the SOAPInput resource statistics to review how many inbound messages the SOAPInput node is receiving, how many replies the SOAPReply node is sending, and how many of those calls are successful or result in SOAP Faults, on a per-operation basis. Statistics for the SOAP nodes are collected with both HTTP and JMS transport. You can review the name of the applied policy set if one is defined. A statistics summary is returned for the whole execution group.

The following table describes the measurements that are returned for the SOAPInput and SOAPReply nodes. These statistics do not include messages which timed out.

Measurements	Description
Name	This measurement takes one of the following values: • "Summary" • *Flow*.*Node*.*Operation* • [Undeployed].*Flow*.*Node*.*Operation* where *Flow* is the name of your message flow, *Node* is the node name, and *Operation* is the name of the operation.
InboundMessagesTotal	The total number of SOAP messages received from the client. This measurement is equal to the sum of `InboundMessagesMadeFlow` and `InboundMessagesFaultedBeforeFlow`.
RepliesSentTotal	The total number of SOAP replies sent back to the client. This measurement is equal to the sum of `SuccessfulRepliesSent` and `FaultRepliesSent`.
InboundMessagesMadeFlow	The number of messages that made the flow without faulting.
InboundMessagesFaultedBeforeFlow	The number of messages that faulted before reaching the flow. This measurement includes input messages that are sent down the Failure terminal.
SuccessfulRepliesSent	The number of successful replies, without SOAP Fault, sent to the client.
FaultRepliesSent	The number of SOAP Fault replies sent to client. These faults can be user-defined faults or broker exceptions.
PolicySetApplied	The name of the policy set if one was defined.

Related concepts:

"Resource statistics" on page 3306
Resource statistics are collected by a broker to record performance and operating details of resources that are used by execution groups.

Related tasks:

"Monitoring resource performance" on page 3305
You can collect statistics to assess the performance of certain resources used by execution groups.

"Viewing resource statistics data in the WebSphere Message Broker Explorer" on page 3313
Use the WebSphere Message Broker Explorer to view resource statistics data for your execution groups in the Broker Resources and Broker Resources Graph views.

"Subscribing to statistics reports" on page 3317
You can subscribe to topics that return statistics about the operation of your message flows and resource managers.

Related reference:

"Resource statistics data" on page 6745
Learn about the measurements for which data is returned when you activate resource statistics collection.

"**mqsichangeresourcestats** command" on page 3819
Use the **mqsichangeresourcestats** command to control statistics gathering for resources in the broker.

"**mqsireportresourcestats** command" on page 3944
Use the **mqsireportresourcestats** command to display current statistics gathering options for resources in the broker.

Resource statistics data: TCPIP Client Nodes

Learn about the data that is returned for the TCPIP Client Nodes resource type when you activate resource statistics collection.

You can view these statistics in the WebSphere Message Broker Explorer, or you can write a program that subscribes to a publication (single XML message) that returns this data. For an example of the publication message, see Example XML output.

Use these statistics to get a view of the level of activity and health of TCP/IP nodes in an execution group. You can see how many connection managers are active and reporting statistics, and the following details from each:

Measurements	Description
OpenConnections	The current number of open connections
ClosedConnections	The total number of connections that were closed since the last execution group restart
MessagesReceived	The total number of messages received (by TCPIPClientInput or TCPIPClientReceive nodes)
MessagesSent	The total number of messages sent (by TCPIPClientOutput nodes)
BytesSent	The total amount of data sent (by TCPIPClientOutput nodes), excluding SSL wrappers.
BytesReceived	The total amount of data received (by TCPIPClientInput or TCPIPClientReceive nodes), excluding SSL wrappers.
FailedConnections	The total number of attempted connections that failed since the last execution group restarts.

Related concepts:

"Resource statistics" on page 3306
Resource statistics are collected by a broker to record performance and operating

details of resources that are used by execution groups.

Related tasks:

"Monitoring resource performance" on page 3305
You can collect statistics to assess the performance of certain resources used by execution groups.

"Viewing resource statistics data in the WebSphere Message Broker Explorer" on page 3313
Use the WebSphere Message Broker Explorer to view resource statistics data for your execution groups in the Broker Resources and Broker Resources Graph views.

"Subscribing to statistics reports" on page 3317
You can subscribe to topics that return statistics about the operation of your message flows and resource managers.

Related reference:

"Resource statistics data" on page 6745
Learn about the measurements for which data is returned when you activate resource statistics collection.

"**mqsichangeresourcestats** command" on page 3819
Use the **mqsichangeresourcestats** command to control statistics gathering for resources in the broker.

"**mqsireportresourcestats** command" on page 3944
Use the **mqsireportresourcestats** command to display current statistics gathering options for resources in the broker.

Resource statistics data: TCPIP Server Nodes

Learn about the data that is returned for the TCPIP Server Nodes resource type when you activate resource statistics collection.

You can view these statistics in the WebSphere Message Broker Explorer, or you can write a program that subscribes to a publication (single XML message) that returns this data. For an example of the publication message, see Example XML output.

Use these statistics to get a view of the level of activity and health of TCP/IP nodes in an execution group. You can see how many connection managers are active and reporting statistics, and the following details from each:

Measurements	Description
OpenConnections	The current number of open connections
ClosedConnections	The total number of connections that were closed since the last execution group restart
MessagesReceived	The total number of messages received (by TCPIPServerInput or TCPIPServerReceive nodes)
MessagesSent	The total number of messages sent (by TCPIPServerOutput nodes)
BytesSent	The total amount of data sent (by TCPIPServerOutput nodes), excluding SSL wrappers.
BytesReceived	The total amount of data received (by TCPIPServerInput or TCPIPServerReceive nodes), excluding SSL wrappers.
FailedSSLConnections	The total number of attempted inbound SSL connections from external clients that failed or were refused since the last execution group restart.

Related concepts:

"Resource statistics" on page 3306
Resource statistics are collected by a broker to record performance and operating details of resources that are used by execution groups.

Related tasks:

"Monitoring resource performance" on page 3305
You can collect statistics to assess the performance of certain resources used by execution groups.

"Viewing resource statistics data in the WebSphere Message Broker Explorer" on page 3313
Use the WebSphere Message Broker Explorer to view resource statistics data for your execution groups in the Broker Resources and Broker Resources Graph views.

"Subscribing to statistics reports" on page 3317
You can subscribe to topics that return statistics about the operation of your message flows and resource managers.

Related reference:

"Resource statistics data" on page 6745
Learn about the measurements for which data is returned when you activate resource statistics collection.

"**mqsichangeresourcestats** command" on page 3819
Use the **mqsichangeresourcestats** command to control statistics gathering for resources in the broker.

"**mqsireportresourcestats** command" on page 3944
Use the **mqsireportresourcestats** command to display current statistics gathering options for resources in the broker.

Monitoring message flows

The following reference topics on monitoring message flows are available:

- "Monitoring profile" on page 6768
- "The monitoring event" on page 6774
- "Correlation and monitoring events" on page 6778

Related concepts:

"Deciding how to configure monitoring events for message flows" on page 3325
Decide whether to use monitoring properties, or a monitoring profile configurable service, to customize the events produced by a message flow.

"Monitoring scenarios" on page 3323
Events can be used to support transaction monitoring, transaction auditing and business process monitoring.

Related tasks:

"Business-level monitoring" on page 3319
You can configure your message flow to emit event messages that can be used to support transaction monitoring and auditing, and business process monitoring.

"Adding files to a broker archive" on page 3223
To deploy files to an execution group, include them in a broker archive (BAR) file.

"Activating monitoring" on page 3334
Use the **mqsichangeflowmonitoring** command to activate monitoring after you have configured monitoring event sources.

Related reference:

"**mqsichangeflowmonitoring** command" on page 3738
Use the **mqsichangeflowmonitoring** command to enable monitoring of message flows.

Monitoring profile

To customize events after a message flow has been deployed, but without redeploying the flow, use a monitoring profile configurable service. By using this service, you can apply a monitoring profile to one or more message flows.

A monitoring profile is an XML document that specifies the event sources in a message flow that will emit events, and the properties of those events. The monitoring profile XML must conform to XML schema file `MonitoringProfile.xsd`.

Outline monitoring profile

Here is an outline monitoring profile that contains a single event source to illustrate the structure:

```
<p:monitoringProfile
xmlns:p="http://www.ibm.com/xmlns/prod/websphere/messagebroker/6.1.0.3/monitoring/profile" p:version="2.0">
    <p:eventSource p:enabled="true" p:eventSourceAddress="SOAPInput.transaction.Start">
        <p:eventPointDataQuery>
            <p:eventIdentity>
                <p:eventName p:literal="" p:queryText=""/>
            </p:eventIdentity>
            <p:eventCorrelation>
                <p:localTransactionId p:queryText="" p:sourceOfId="automatic"/>
                <p:parentTransactionId p:queryText="" p:sourceOfId="automatic"/>
                <p:globalTransactionId p:queryText="" p:sourceOfId="automatic"/>
            </p:eventCorrelation>
           <p:eventFilter p:queryText="true()"/>           <p:eventUOW p:unitOfWork="messageFlow" />        </p:eventPointDataQuery>
        <p:applicationDataQuery>
            <p:simpleContent p:dataType="boolean" p:name="" p:targetNamespace="">
                <p:valueQuery p:queryText=""/>
            </p:simpleContent>
            <p:complexContent p:name="" targetNamespace="">
                <p:payloadQuery p:queryText=""/>
            </p:complexContent>
        </p:applicationDataQuery>
        <p:bitstreamDataQuery p:bitstreamContent="all" p:encoding="base64Binary"/>
    </p:eventSource>
</p:monitoringProfile>
```

The root element is `p:monitoringProfile`. It contains one or more `p:eventSource` elements, each of which specifies an event source and defines its properties. Each `p:eventSource` element contains:

- A `p:eventPointDataQuery` element that provides key information about the event.
- Optional: A `p:applicationDataQuery` element if the event payload will include data fields extracted from a message.
- Optional: A `p:bitstreamDataQuery` element if the event payload will include bitstream data from a message.

Creating a monitoring profile

To help you create monitoring profiles, the following sample contains an outline monitoring profile XML file and the monitoring profile XML schema file (`MonitoringProfile.xsd`):

- WebSphere Business Monitor

Validate your monitoring profiles against the XML schema to ensure that they are correct.

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Tip: If you have a deployed message flow that has monitoring properties configured using the Message Flow editor, you can use the **mqsireportflowmonitoring** command to create the equivalent monitoring profile XML file for the message flow. You can use this profile as a starting point for creating other monitoring profiles.

The following steps describe how to create a monitoring profile XML. Follow these steps for each `p:eventSource` element.

1. Specify the `p:eventSource/@p:eventSourceAddress` attribute.

   This is a string that uniquely identifies the event source in the message flow. It must follow the fixed format for transaction events and terminal events, as shown in the following table:

Event type	Event source address
Transaction Start event	*nodelabel*.transaction.Start
Transaction End event	*nodelabel*.transaction.End
Transaction Rollback event	*nodelabel*.transaction.Rollback
Terminal event	*nodelabel*.terminal.<Terminal>

   Note: *nodelabel* is the label of the node as known by the broker runtime components. If the node is in a subflow the label reflects this. For example, flow A contains an instance of flow B as a subflow labeled `myB`; flow B contains an instance of a Compute node labeled `myCompute`. The *nodelabel* for the Compute node is `myB.myCompute`.

   In the emitted event, the address string of the event source is set in the `wmb:eventData/@wmb:eventSourceAddress` attribute.

2. Optional: Specify the name by which events from this event source will be known, in the `p:eventPointDataQuery/p:eventIdentity/p:eventName` element.
   - If the event name is a fixed string, complete the `p:eventName/@p:literal` attribute
   - If the event name is to be extracted from a field in the message, complete the `p:eventName/@p:queryText` attribute by specifying an XPath query.

   In the emitted event, the event name is set in the `wmb:eventPointData/wmb:eventIdentity/@wmb:eventName` attribute.

   If `p:eventName` element is not supplied, `@wmb:eventName` in the emitted event defaults to `@p:eventSourceAddress`.

3. Optional: Complete the `p:eventPointDataQuery/p:eventFilter/@p:queryText` attribute by specifying an XPath expression to control whether the event is emitted. The expression must evaluate to true (the event is emitted), or false (the event is not emitted). The expression can reference fields in the message tree, or elsewhere in the message assembly. If an event does not contain an eventFilter element, the event is always emitted.

   Using this facility, you can tailor event emissions to your business requirements, by filtering out events that do not match a set of rules. This can reduce the number of events emitted, and reduce the workload on your monitoring application.

4. Optional: Complete the p:applicationDataQuery element, if the event is to contain selected data fields extracted from the message. You can extract one or more fields from the message data and include it with the event. The fields can be simple or complex.
   - For each simple data field, complete a p:simpleContent element:
     - Complete the p:simpleContent/p:valueQuery/@p:queryText attribute by specifying an XPath query.
     - Complete the p:simpleContent/@p:name, @p:namespace and @p:dataType attributes. The @p:dataType value must be one of boolean, date, dateTime, decimal, duration, integer, string or time.
   - For each complex data field, complete a p:complexContent element:
     - Complete the p:complexContent/p:payloadQuery/@p:queryText attribute by specifying an XPath query.
     - Complete the p:complexContent/@p:name and @p:namespace attributes.

   This facility is commonly used for communicating significant business data in a business event. If the event contains the input bit stream, this facility can also be used to extract key fields, allowing another application to provide an audit trail or to resubmit failed messages.

   In the emitted event, the extracted data is set in the wmb:applicationData/wmb:simpleContent and wmb:applicationData/wmb:complexContent elements.
5. Optional: Complete the p:bitstreamDataQuery element, if the event is to capture message bitstream data:
   - Complete the @p:bitstreamContent attribute. The attribute value must be one of headers, body, or all.
   - Complete the @p:encoding attribute. The attribute value must be one of CDATA, base64Binary or hexBinary.

   In the emitted event, the extracted bitstream data is set in the wmb:bitstreamData/wmb:bitstream element.
6. Optional: Complete the p:eventPointDataQuery/p:eventCorrelation element. For information about correlation, see "Correlation and monitoring events" on page 6778.

   Every emitted monitoring event can contain up to three correlation attributes. If no correlation information is specified in the monitoring profile, no correlation attributes will be used.

   a. Optional: Complete the p:localTransactionId element.
      - If you want to reuse the local correlator from the Environment tree, set the p:localTransactionId/@p:sourceOfId attribute to automatic. If no local correlator exists yet, a new unique value will be generated and saved in the Environment tree.
      - If you want to use a value contained in a location in the message, set the p:localTransactionId/@p:sourceOfId attribute to query, then complete the p:localTransactionId/@p:queryText attribute by specifying an XPath query. Ensure that the specified location contains a correlator value unique to this invocation of the message flow. The value is saved in the Environment tree.

   b. Optional: Complete the p:parentTransactionId element.
      - If you want to reuse the parent correlator from the Environment tree, set the p:parentTransactionId/@p:sourceOfId attribute to automatic. If no parent correlator exists yet, no parent correlator will be used.
      - If you want to use a value contained in a location in the message, set the p:parentTransactionId/@p:sourceOfId attribute to query, then complete

the p:parentTransactionId/@p:queryText attribute by specifying an XPath query. Ensure that the specified location contains a suitable value for the parent correlator. The value is saved in the Environment tree.

c. Optional: Complete the p:globalTransactionId element.
   - If you want to reuse the global correlator from the Environment tree, set the p:globalTransactionId/@p:sourceOfId attribute to automatic. If no global correlator exists yet, no global correlator will be used.
   - If you want to use a value contained in a location in the message, set the p:globalTransactionId/@p:sourceOfId attribute to query, then complete the p:globalTransactionId/@p:queryText attribute by specifying an XPath query. Ensure that the specified location contains a suitable value for the global correlator. The value is saved in the Environment tree.

7. Complete the p:eventPointDataQuery/p:eventUOW element. This determines whether the emission of monitoring events by a message flow is coordinated with the message flow transaction, or is in an independent unit of work, or is not in a unit of work.

   Set the p:eventUOW/@p:unitOfWork attribute to one of the following values:

   messageFlow
   The event, and all other events with this setting, are emitted only if the message flow commits its unit of work successfully.

   If the transaction start event is specified to be included in the message flow unit of work, but the message processing fails and this unit of work is not published, the transaction start event will be included in an independent unit of work. This ensures that your monitoring application receives a pair of events (start and rollback), rather than receiving a rollback event in isolation.

   independent
   The event is emitted in a second unit of work, independent of the main unit of work. The event, and all other events with this setting are emitted whether or not the main unit of work commits successfully.

   An independent transaction can be started only if the main transaction has been either committed or rolled back. If the **Commit count** property of the flow is greater than one, ("Configurable message flow properties" on page 4020), or the **Commit by message group** property is set ("Receiving messages in a WebSphere MQ message group" on page 1554), the events targeted for the independent transaction are instead emitted out of sync point, and a message is output stating that this has been done.

   none The event is emitted out of sync point (not in any unit of work.) The event is emitted when the message passes through the event source, and is available for reading immediately.

   Not all these options are available on all event types. The allowed values are shown in the following table:

Event Type	Allowed values
transaction.Start	messageFlow independent none
transaction.End	messageFlow none

Event Type	Allowed values
transaction.Rollback	independent none
terminal	messageFlow independent none

If you do not include the `eventUOW` element for an event source, transactionality for all events issued from that source, except transaction.rollback events, defaults to `messageFlow`. Transactionality for transaction.rollback events defaults to `independent`.

XPath queries and XML namespaces

If an XPath query contains a component that has an XML namespace, the XPath contains a namespace prefix for the namespace. For example, the following XPath refers to components in two different namespaces:

```
<p:localTransactionId p:sourceOfId="query" p:queryText="$Body/soapenv:Header/wsa:messageID" />
```

For the broker to resolve the namespace prefix, the namespace URL must also be provided. Supply a prefix mapping element for each namespace:

```
<p:localTransactionId p:sourceOfId="query" p:queryText="$Body/soapenv:Header/wsa:messageID">
  <p:prefixMapping p:prefix="soapenv" p:URI="http://www.w3.org/2003/05/soap-envelope" />
  <p:prefixMapping p:prefix="wsa" p:URI="http://www.w3.org/2005/08/addressing" />
</p:localTransactionId>
```

Monitoring profile examples

The following XML documents conform to the monitoring profile schema

Monitoring profile 1: Two event sources, each supplying an event name

```
<p:monitoringProfile
xmlns:p="http://www.ibm.com/xmlns/prod/websphere/messagebroker/6.1.0.3/monitoring/profile" p:version="2.0">
 <p:eventSource p:eventSourceAddress="SOAPInput.transaction.Start">
  <p:eventPointDataQuery>
    <p:eventIdentity>
      <p:eventName p:literal="SOAP start event"/>
    </p:eventIdentity>
  </p:eventPointDataQuery>
 </p:eventSource>
 <p:eventSource p:eventSourceAddress="SOAPInput.transaction.End">
  <p:eventPointDataQuery>
    <p:eventIdentity>
      <p:eventName p:literal="SOAP end event"/>
    </p:eventIdentity>
  </p:eventPointDataQuery>
 </p:eventSource>
</p:monitoringProfile>
```

Monitoring profile 2: Supply an alternative local correlator

```
<p:monitoringProfile
xmlns:p="http://www.ibm.com/xmlns/prod/websphere/messagebroker/6.1.0.3/monitoring/profile" p:version="2.0">
 <p:eventSource p:eventSourceAddress="SOAPInput.transaction.Start">
  <p:eventPointDataQuery>
    <p:eventCorrelation>
      <p:localTransactionId p:queryText="$Body/soapenv:Header/wsa:messageID" p:sourceOfId="query">
        <p:prefixMapping p:prefix="soapenv" p:URI="http://www.w3.org/2003/05/soap-envelope"/>
        <p:prefixMapping p:prefix="wsa" p:URI="http://www.w3.org/2005/08/addressing"/>
      </p:localTransactionId>
    </p:eventCorrelation>
  </p:eventPointDataQuery>
 </p:eventSource>
</p:monitoringProfile>
```

Monitoring profile 3: Include two simple fields from the message

```
<p:monitoringProfile
xmlns:p="http://www.ibm.com/xmlns/prod/websphere/messagebroker/6.1.0.3/monitoring/profile" p:version="2.0">
 <p:eventSource p:eventSourceAddress="MQInput.terminal.out">
  <p:applicationDataQuery>
    <p:simpleContent p:dataType="integer" p:name="InvoiceNumber">
      <p:valueQuery p:queryText="$Body/invoice/invoiceNo"/>
    </p:simpleContent>
    <p:simpleContent p:dataType="string" p:name="BatchID">
      <p:valueQuery p:queryText="$Body/batch/batchNo"/>
    </p:simpleContent>
  </p:applicationDataQuery>
 </p:eventSource>
</p:monitoringProfile>
```

Monitoring profile 4: Include the bitstream, encoded as CDATA

By default, bitstreams are encoded in base64Binary format. The following monitoring profile changes the encoding to CDATA.

```
<p:monitoringProfile
xmlns:p="http://www.ibm.com/xmlns/prod/websphere/messagebroker/6.1.0.3/monitoring/profile" p:version="2.0">
 <p:eventSource p:eventSourceAddress="MQInput.terminal.out">
    <p:bitstreamDataQuery p:bitstreamContent="body" p:encoding="CDATA"/>
 </p:eventSource>
</p:monitoringProfile>
```

CDATA encoding is not suitable for all types of data. Use CDATA only when `@p:bitstreamContent="body"`. Do not use CDATA if your message bitstreams might contain characters that are not allowed in XML (see http://www.w3.org/TR/2006/REC-xml-20060816/#charsets).

Related concepts:

"Deciding how to configure monitoring events for message flows" on page 3325
Decide whether to use monitoring properties, or a monitoring profile configurable service, to customize the events produced by a message flow.

Related tasks:

"Business-level monitoring" on page 3319
You can configure your message flow to emit event messages that can be used to support transaction monitoring and auditing, and business process monitoring.

"Configuring monitoring event sources using monitoring properties" on page 3327
In the Message flow Editor, use the Monitoring tab on the properties of a node to add one or more monitoring events.

"Configuring monitoring event sources using a monitoring profile" on page 762
You can create a monitoring profile and use the **mqsichangeflowmonitoring** command to configure your message flows to emit monitoring events.

"Activating monitoring" on page 3334
Use the **mqsichangeflowmonitoring** command to activate monitoring after you have configured monitoring event sources.

"Enabling and disabling event sources" on page 3336
When events have been configured for a message flow and deployed to the broker, you can enable and disable individual events. You can do this from the command line, without having to redeploy the message flow, or you can do this from the Message Flow Editor, in which case you must redeploy.

"Reporting monitoring settings" on page 3343
Use the **mqsireportflowmonitoring** command to report monitoring settings for a flow.

Related reference:

"The monitoring event"
You can configure WebSphere Message Broker to emit a monitoring event (an XML document) when something interesting happens. Events are typically emitted to support transaction monitoring, transaction auditing, and business process monitoring. The event XML conforms to the monitoring event schema `WMBEvent.xsd`.

Related information:

"Correlation and monitoring events" on page 6778
A monitoring application uses correlation attributes to identify events that belong to the same business transaction.

The monitoring event

You can configure WebSphere Message Broker to emit a monitoring event (an XML document) when something interesting happens. Events are typically emitted to support transaction monitoring, transaction auditing, and business process monitoring. The event XML conforms to the monitoring event schema `WMBEvent.xsd`.

You can find the `WMBEvent.xsd` schema file in the WBMonitorEventsProject folder of the WebSphere Business Monitor sample when you import it to your workspace. The `WMBEvent.xsd` file is also available in the default message catalog. To access the schema from the WebSphere Message Broker Toolkit, click **File** > **New** > **Message Definition File From** > **IBM supplied message** > **Message Broker Monitoring Event**.

Each event contains the following information:

- Source of the event
- Name of the event
- Creation time and sequence number
- Correlation ID for events emitted by the same transaction or unit of work
- Details of the message flow

Tip: A terminal emits an event only if the message passes through that terminal. In particular, the output terminal of an output node emits events only if it is connected.

A monitoring event can also contain the following items:

- Application data extracted from the message.
- Part or all of the message bit stream.

  The bit stream cannot be included in monitoring events if one of the following nodes created the message:
  - FileInput, if the message domain is MRM or XMLNSC
  - TCPIPClientInput, if the message domain is MRM or XMLNSC
  - TCPIPServerInput, if the message domain is MRM or XMLNSC
  - SOAPInput, SOAPAsyncResponse
  - Any of the following Adapter input nodes:
    - PeopleSoftInput
    - SiebelInput
    - JDEdwardsInput
    - TwineballInput

Note: The bit stream can be included if a SAPInput node created the message.

The bit stream cannot be included in monitoring events if the message is a response from one of the following nodes:

– SOAPRequest

– Any of the Adapter request nodes:

  - SAPRequest
  - PeopleSoftRequest
  - SiebelRequest
  - JDEdwardsRequest
  - TwineballRequest

A warning is logged to user trace if an event source is configured to include the bit stream, but it cannot be included. You might be able to extract some or all of the data in the message as Event Payload; be aware that large event messages might affect performance.

Use either the monitoring properties of a message flow or a monitoring profile configurable service to configure the following items:

- Where an event is emitted from
- The content of the event

Sequence number

To enable software that processes events, such as WebSphere Business Monitor, to sequence them correctly, both an ISO 8601 time stamp and a counter attribute are produced. The counter attribute is in the EventSequence element of the monitoring event. This counter starts at 1 for the first event produced by the processing of a message (normally the transaction.Start event) and is incremented for each subsequent event produced. It is reset to 1 at the start of the next message. The creation time and counter are always produced on all monitoring events. The software processing the events can choose which field, or a combination of the two, to use to sequence the events.

If a message is processed successfully, the monitoring events generated have a contiguous, incrementing set of counter values, starting at 1 and assigned at the time when the event is created. If a message fails and is rolled back, there might be gaps in the counter sequence. The missing values are those used for events that were produced as part of the message flow unit of work that was rolled back.

Monitoring events can be included in the main unit of work of a message flow, an independent unit of work, or outside of a unit of work. Consequently, if a message flow fails the sequence numbers are not guaranteed to be contiguous. For example consider the following scenario:

- Sequence 1: the transaction start event.
- Sequence 2, 3, 4: events in the message flow unit of work.
- Sequence 5: the independent unit of work.
- Sequence 6: an event outside of a unit of work.
- Sequence 7: an event in the message flow unit of work.
- Sequence 8: an event outside of a unit of work.
- The flow then fails and is rolled back.
- Sequence 9: the transaction rollback event in the independent unit of work.

Only sequence numbers 1, 5, 6, 8 and 9 are sent to the monitoring application.

Example event

```
<?xml version="1.0" encoding="UTF-8"?>
<wmb:event xmlns:wmb=http://www.ibm.com/xmlns/prod/websphere/messagebroker/6.1.0/monitoring/event>
   <wmb:eventPointData>
     <wmb:eventData wmb:eventSourceAddress="MQInput1.terminal.in"
                    wmb:eventSchemaVersion="6.1.0.3" wmb:productVersion="7000">
       <wmb:eventIdentity wmb:eventName="MQInput event"/>
       <wmb:eventSequence wmb:creationTime="2001-12-31T12:00:00+01:00" wmb:counter='2'/>
       <wmb:eventCorrelation wmb:localTransactionId="123"
                             wmb:parentTransactionId="456"
                             wmb:globalTransactionId="789"/>
     </wmb:eventData>
     <wmb:messageFlowData>
       <wmb:broker wmb:UUID="d53122ae-1c01-0000-0080-b1b02528c6bf"
                   wmb:name="myBroker"/>
       <wmb:executionGroup wmb:UUID="d43122ae-1c01-0000-0080-b1b02528c6bf"
                           wmb:name="default"/>
       <wmb:messageFlow wmb:UUID="e6d224ae-1c01-0000-0080-9100cd1a61f7"
                        wmb:name="myMessageFlow" wmb:threadId="4201"
                        wmb:uniqueFlowName="myBroker.default.myMessageFlow"/>
       <wmb:node wmb:nodeLabel="MQInput1" wmb:nodeType="ComIbmMqInputNode"
                 wmb:terminal="in" wmb:detail="MYMESSAGEFLOW.IN"/>
     </wmb:messageFlowData>
   </wmb:eventPointData>
   <wmb:applicationData xmlns="">
     <wmb:simpleContent wmb:name="invoiceNo" wmb:targetNamespace=""
                        wmb:dataType="string" wmb:value="567"/>
     <wmb:complexContent wmb:elementName="customerName" wmb:targetNamespace="">
       <customerName>
         <firstName>Steve</firstName>
         <lastName>Bloggs</lastName>
       </customerName>
     </wmb:complexContent>
   </wmb:applicationData>
   <wmb:bitstreamData>
     <wmb:bitstream wmb:encoding="base64Binary">TUQgIAIAAAAAAAAACAAAAP////8AAAAAIgIAALUBAAAgICAgICAg
IAAAAAAAAAAAQU1RIFFNMSAgICAgICAgIHo640ggABsHAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACAgICAgICAgICAgICAgI
CAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIFFNMSAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIC
Ag</wmb:bitstream>
   </wmb:bitstreamData>
</wmb:event>
```

Default content of monitoring events

When an event is emitted, the fields in the event are created using the information provided by the monitoring properties of the message flow, or a monitoring profile configurable service if one has been applied to the message flow. Any event field that is not explicitly specified is given a default value as shown in the table.

Field in event	Default Value
eventData/@wmb:eventSourceAddress	No default; you must provide this information.
eventData/@wmb:eventSchemaVersion	6.1.0.3
eventData/@wmb:productVersion	7000
eventData/eventIdentity/ @wmb:eventName	The default is derived from @eventSourceAddress
eventData/eventSequence/@counter	The counter is set to 1 for first event emitted and increased by 1 for each subsequent event.
eventData/eventSequence/@creationTime	The date and time when the event was created.

Field in event	Default Value
eventData/eventCorrelation/ @localTransactionId	A generated unique identifier.
eventData/eventCorrelation/ @parentTransactionId	No default. Unless you set this value, an empty string is used.
eventData/eventCorrelation/ @globalTransactionId	No default. Unless you set this value, an empty string is used.
messageFlowData/broker/@name	The name of the broker.
messageFlowData/broker/@UUID	The UUID of the broker.
messageFlowData/executionGroup/ @name	The name of the execution group.
messageFlowData/executionGroup/ @UUID	The UUID of the execution group.
messageFlowData/messageFlow/@name	The name of the message flow.
messageFlowData/messageFlow/@UUID	The UUID of the message flow.
messageFlowData/messageFlow/ @uniqueFlowName	A string composed of the names of the broker, execution group, and flow in the form: *brokerName.executionGroupName.flowName*
messageFlowData/messageFlow/ @threadId	The thread ID of the message flow. The format depends on the platform.
messageFlowData/node/@nodeLabel	The label of the node that emitted the event.
messageFlowData/node/@nodeType	The type of the node that emitted the event.
messageFlowData/node/@nodeDetail	Optional information about the node. **MQInput** The name of the queue. **Other nodes** Omitted.
applicationData	No default; omitted if not provided.
bitstreamData	No default; omitted if not provided.

XPath queries that end in a wildcard

If an XPath query ends in a wildcard, each element node or attribute node appears in a separate complexContent or simpleContent folder in the emitted event.

Related concepts:

"Deciding how to configure monitoring events for message flows" on page 3325
Decide whether to use monitoring properties, or a monitoring profile configurable service, to customize the events produced by a message flow.

Related tasks:

"Business-level monitoring" on page 3319
You can configure your message flow to emit event messages that can be used to support transaction monitoring and auditing, and business process monitoring.

"Configuring monitoring event sources using monitoring properties" on page 3327
In the Message flow Editor, use the Monitoring tab on the properties of a node to add one or more monitoring events.

"Configuring monitoring event sources using a monitoring profile" on page 762
You can create a monitoring profile and use the **mqsichangeflowmonitoring** command to configure your message flows to emit monitoring events.

"Activating monitoring" on page 3334
Use the **mqsichangeflowmonitoring** command to activate monitoring after you have configured monitoring event sources.

"Enabling and disabling event sources" on page 3336
When events have been configured for a message flow and deployed to the broker, you can enable and disable individual events. You can do this from the command line, without having to redeploy the message flow, or you can do this from the Message Flow Editor, in which case you must redeploy.

"Creating a monitoring model for use by WebSphere Business Monitor" on page 3338
Enable WebSphere Business Monitor to monitor WebSphere Message Broker events.

"Reporting monitoring settings" on page 3343
Use the **mqsireportflowmonitoring** command to report monitoring settings for a flow.

Related reference:

"Monitoring profile" on page 6768
To customize events after a message flow has been deployed, but without redeploying the flow, use a monitoring profile configurable service. By using this service, you can apply a monitoring profile to one or more message flows.

Correlation and monitoring events

A monitoring application uses correlation attributes to identify events that belong to the same business transaction.

A business transaction can be any of the following scenarios:

- A single invocation of a message flow.
- Multiple invocations of the same message flow from a parent application. The parent application might be another message flow.
- Multiple invocations of various message flows from a parent application. The parent application might be another message flow.

Three correlation attributes are available for you to use in your events: *local correlator*, *parent correlator* and *global correlator*. The exact usage of the correlation attributes varies depending on the requirements. For example, a parent application can pass its transaction identifier to the child message flow (perhaps in a header) so that the child message flow can report it in the event as a parent correlator.

Every emitted monitoring event can contain a local correlator, a parent correlator, and a global correlator. Correlation information is placed in the following attributes of the event:

```
wmb:eventPointData/wmb:eventCorrelation/@wmb:localTransactionId
wmb:eventPointData/wmb:eventCorrelation/@wmb:parentTransactionId
wmb:eventPointData/wmb:eventCorrelation/@wmb:globalTransactionId
```

You can specify correlation information when you configure the event.

If you do not specify any correlation information when you configure your events, no correlation attributes will be used.

If you do specify correlation information, you must configure the correlation attributes to be used, and where they will read their value from. Typically you

need to specify correlation information only for the first event source in the message flow; by default all later event sources retrieve the same value from the Environment tree.

The exact steps for specifying correlation information depend on whether you are using monitoring properties or a monitoring profile to configure your events, but the principle is the same for both techniques:

Local correlator

If you want to reuse the local correlator from the Environment tree, specify `Automatic`. If no local correlator exists yet, a new unique value will be generated and saved in the Environment tree.

If you want to use a value contained in a location in the message, specify the location of the correlator by supplying an XPath into the message tree. Ensure that the specified location contains a correlator value unique to this invocation of the message flow. The extracted value is saved in the Environment tree as the local correlator.

Parent correlator

If you want to reuse the parent correlator from the Environment tree, specify `Automatic`. If no parent correlator exists yet, no parent correlator will be used.

If you want to use a value contained in a location in the message, specify the location of the correlator by supplying an XPath into the message tree. Ensure that the specified location contains a suitable value for the parent correlator. The extracted value is saved in the Environment tree as the parent correlator.

Global correlator

If you want to reuse the global correlator from the Environment tree, specify `Automatic`. If no global correlator exists yet, no global correlator will be used.

If you want to use a value contained in a location in the message, specify the location of the correlator by supplying an XPath into the message tree. Ensure that the specified location contains a suitable value for the global correlator. The extracted value is saved in the Environment tree as the global correlator.

When a correlator value has been set, it is saved in the Environment tree. Later event sources can reuse the saved value by specifying `Automatic`. There is no need to use the same XPath in all the event sources in your message flow, and doing so might adversely affect performance.

The locations in the Environment tree used to save correlator values for use by later events are:

```
Environment.Monitoring.EventCorrelation.localTransactionId
Environment.Monitoring.EventCorrelation.parentTransactionId
Environment.Monitoring.EventCorrelation.globalTransactionId
```

The following message tree locations often contain a value that can be used as a correlator:

```
$Root/MQMD/MsgId
$Root/MQMD/CorrelId
$Root/JMSTransport/Transport_Folders/Header_Values/JMSMessageID
$Root/JMSTransport/Transport_Folders/Header_Values/JMSCorrelationID
$LocalEnvironment/Destination/HTTP/RequestIdentifier
$LocalEnvironment/Wildcard/WildcardMatch
```

Tip: If the three available correlators are not sufficient, you can configure the event to extract other correlation fields from the message and place them in the `wmb:applicationData/wmb:simpleContent` section of the event.

Tip: The Collector node and the AggregateControl node do not preserve the Environment tree for later nodes in the message flow. If you want to use the same correlator value later in the flow, ensure that the correlator value is available in the message tree, and that the first event source after the Collector or AggregateControl node specifies the location of the correlator by supplying an XPath into the message tree.

Scenarios

Scenario 1: In this scenario, all three correlators are used to monitor data that starts in an external process. Several message flows then transform the data.

- The `globalTransactionID` field contains an identifier from the message header or payload. This identifier correlates events from the external process and WebSphere Message Broker.
- The `parentTransactionID` correlates events in WebSphere Message Broker from different message flows.
- The `localTransactionID` correlates events from the same message flow.

Scenario 2: In this scenario, the `parentTransactionID` field is used to correlate request and reply messages between two message flows:

- The Request flow sends a **`purchaseOrder`** request to an external application for processing.
- The Reply flow receives a confirmation reply from the external application when the **`purchaseOrder`** has been processed.

You need to correlate the request and replies belonging to the same purchase order. You can do this by setting the `parentTransactionID` to a field in the **`purchaseOrder`**, such as a **`purchaseOrderID`**, which is available in both the request and reply.

Related concepts:

"Deciding how to configure monitoring events for message flows" on page 3325
Decide whether to use monitoring properties, or a monitoring profile configurable service, to customize the events produced by a message flow.

"Monitoring scenarios" on page 3323
Events can be used to support transaction monitoring, transaction auditing and business process monitoring.

Related tasks:

"Configuring monitoring event sources using monitoring properties" on page 3327
In the Message flow Editor, use the Monitoring tab on the properties of a node to add one or more monitoring events.

"Configuring monitoring event sources using a monitoring profile" on page 762
You can create a monitoring profile and use the **`mqsichangeflowmonitoring`** command to configure your message flows to emit monitoring events.

"Adding files to a broker archive" on page 3223
To deploy files to an execution group, include them in a broker archive (BAR) file.

"Activating monitoring" on page 3334
Use the **mqsichangeflowmonitoring** command to activate monitoring after you have configured monitoring event sources.

Related reference:

"Monitoring profile" on page 6768
To customize events after a message flow has been deployed, but without redeploying the flow, use a monitoring profile configurable service. By using this service, you can apply a monitoring profile to one or more message flows.

"**mqsichangeflowmonitoring** command" on page 3738
Use the **mqsichangeflowmonitoring** command to enable monitoring of message flows.

Example XPath expressions for event filtering

Use numeric, string, or Boolean expressions when configuring an event source, to determine whether the event is emitted.

When you configure an event source, using either monitoring properties or a monitoring profile, you use an XPath expression to determine whether the event is emitted.

- If the event is always required, use `true()`.
- If the event is required only in certain circumstances, use an expression of the form

  xpath-query relational-operator value

Comparing numeric values

To emit an event only when the value is greater than 10 000, for example, enter an expression such as this:

```
$Body/StockTrade[1]/Details[1]/Value[1] > 10000
```

The [1] suffixes in the query specify that the first occurrence of the element within its parent is required. If these suffixes are not specified, the XPath engine searches the message for other occurrences of each element. This search might adversely affect performance.

Comparing string values

To emit an event only when the company is "Stock Co" for example:

```
$Body/StockTrade[1]/Details[1]/Company[1] = 'Stock Co'
```

Comparing Boolean values

Consider the example of a shares transfer approval. The approval flag in the message tree is a Boolean value. You cannot simply specify the element name, because this always returns true if the element exists. Instead, you query the value of the element, and compare the value to the string 'true' to yield the actual true or false result. The XPath query is:

```
$Body/StockTrade[1]/Shares[1]/Transfer[1]/Approved[1] = 'true'
```

XPath queries that return a nodeset, such as `$Body/StockTrade[1]/Details[1]`, are always evaluated as false, because they cannot be converted to a Boolean value.

Related concepts:

"Monitoring basics" on page 3320
Message flows can be configured to emit events. The events can be read and used by other applications for transaction monitoring, transaction auditing, and business process monitoring.

Related tasks:

"Configuring monitoring event sources using monitoring properties" on page 3327
In the Message flow Editor, use the Monitoring tab on the properties of a node to add one or more monitoring events.

"Configuring monitoring event sources using a monitoring profile" on page 762
You can create a monitoring profile and use the **mqsichangeflowmonitoring** command to configure your message flows to emit monitoring events.

XPath expressions that are not suitable for the export monitoring information option

Some XPath expressions produce a warning on the Monitoring tab.

The following expressions are not suitable for the export monitoring information option used to create a monitor model in WebSphere Business Monitor Version 7, and produce a warning. If you export monitoring information for any of the following listed expressions, an event part is not generated for that XPath expression, and a warning message is written to the *flowProjectName*_batchgen.report.txt file.

$Body
$Body/
$DestinationList
$DestinationList/
$Environment
$Environment/
$ExceptionList
$ExceptionList/
$LocalEnvironment
$LocalEnvironment/
$Properties
$Properties/
$Root
$Root/
$Root/<*domain parser or header*>
$Root/<*domain parser or header*>/
$Root/SOAP/Attachment
$Root/SOAP/Attachment/
$Root/SOAP/Body
$Root/SOAP/Body/
$Root/SOAP/Header
$Root/SOAP/Header/

Note: This list is an exact match. For example, the expression `$LocalEnvironment/Destination/MQ/Defaults/queueName` is valid, although `$LocalEnvironment` is not.

Related tasks:

"Creating a monitor model for WebSphere Business Monitor V7" on page 3341
Export monitoring information from WebSphere Message Broker to create a monitoring model for WebSphere Business Monitor V7

WebSphere Message Broker Toolkit

The WebSphere Message Broker Toolkit provides your application development environment on Windows and Linux on x86.

Within the WebSphere Message Broker Toolkit, you can access resources through several perspectives, and create and modify them using the supplied editors:

- Perspectives
- Editors
- Resource types

For information about how to use the keyboard in the WebSphere Message Broker Toolkit, see "WebSphere Message Broker Explorer and WebSphere Message Broker Toolkit keyboard shortcuts" on page 6828.

When you create resources in the WebSphere Message Broker Toolkit, or import resources into the WebSphere Message Broker Toolkit, be aware of the file system limit of 256 characters that is imposed by the Windows file system. This limit can cause restrictions in path specifications to resources (for example, message flow projects), and might cause access problems if the combination of path and resource name exceeds this limit. Keep installation locations and resource names short to avoid problems associated with this restriction.

Resource editors do not automatically reflect the changes that you make in one window in additional windows that you have opened to view the same resource. Close and reopen additional windows each time you update a resource in an editor session.

A minimum display resolution of at least 1024 x 768 is required for some dialogs; for example, the Preferences dialog.

Related concepts:

"WebSphere Message Broker Toolkit" on page 31
The WebSphere Message Broker Toolkit is an integrated development environment and graphical user interface based on the Eclipse platform.

Related tasks:

"Configuring the workbench" on page 570
You can configure various settings in the workbench to suit your requirements and your working environment.

Perspectives in the WebSphere Message Broker Toolkit

The following links provide a description of each of the perspectives in the WebSphere Message Broker Toolkit:

- "Broker Application Development perspective" on page 6784
- "Debug perspective" on page 6789
- "Plug-in Development perspective" on page 6792

Related concepts:

"WebSphere Message Broker Toolkit perspectives" on page 34
A perspective is a group of views and editors that shows various aspects of the resources in the WebSphere Message Broker Toolkit.

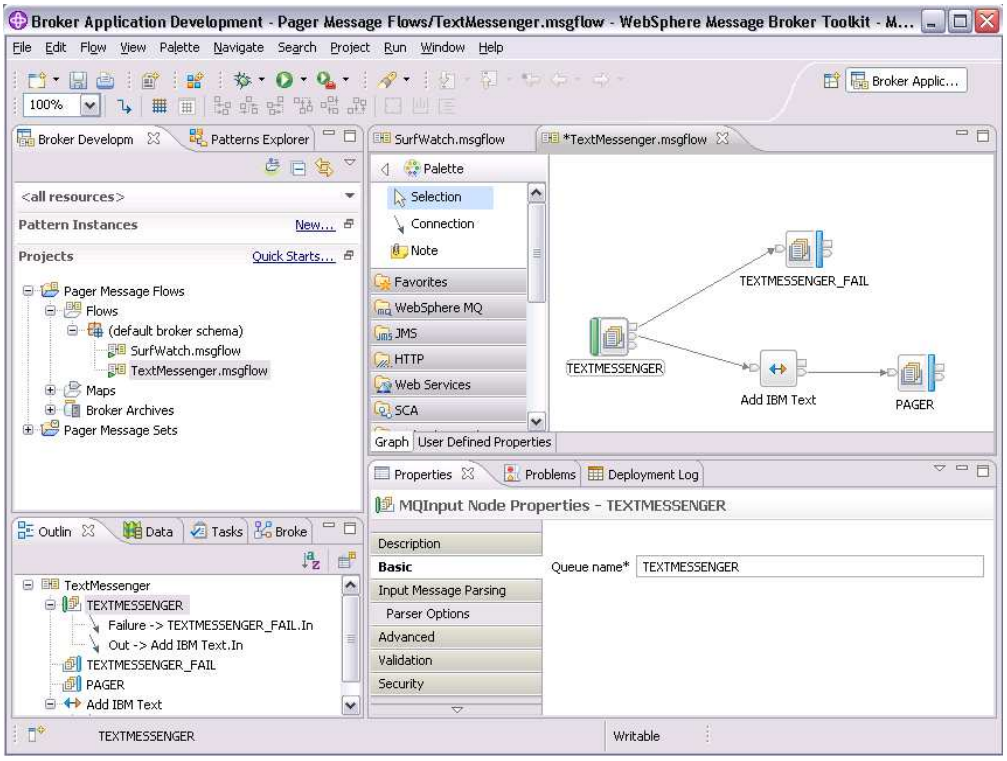
Broker Application Development perspective

The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

Typically, you complete the following tasks in this perspective:

- Developing message flows and message sets.
- Enqueuing and dequeuing test and production messages, which is useful when you are debugging message flows.
- Creating user-defined nodes and user-defined node projects.

The following figure shows the Broker Application Development perspective of the WebSphere Message Broker Toolkit. The `TextMessenger.msgflow` file in the Broker Development view is open in the Message Flow editor.



The Broker Application Development perspective provides several views to navigate, browse, and update your resources.

Broker Development view:
Use the Broker Development view to navigate and browse your resources, and to select resources for editing. You can also use this view to access the Quick Start wizards, which help you create the basic resources that are required for a broker application.

The Broker Development view contains the following sections:

- <all resources> (a working set selection list)
- Pattern Instances
- Projects

When the Broker Development view contains projects in the Projects section, to display the list of Quick Starts, click **Quick Starts**.

When the Broker Development view does not contain projects in the Projects section, the Quick Starts list is displayed in the Projects section.

The following Quick Starts links are shown:

Quick Starts

Start building your message flow application with one of the following tasks.

Start from scratch

Start from WSDL and/or XSD files

Start from existing message set

Start from adapter connection

Start from SCA Import or Export

Start from patterns

Start from samples

The Projects section of the Broker Development view displays a hierarchical view of the following message flow and message set resource files:

- Pattern instance projects
- User-defined nodes
- Message flow projects
- Message flows
- ESQL files
- Mappings files
- Flow test files
- Broker archive files
- Message set projects
- Message sets
- Message definition files
- Java projects
- Data Design projects

When you create resources, they are grouped by file type in logical categories within the hierarchy, and placed in virtual folders. These virtual folders are described in "Resources" on page 36.

Other folders and files that are created for the message flow project or message set project are placed directly under the project folders.

You can hide the categories (virtual folders) by clicking **Hide Categories**  on the Broker Development view toolbar. You can use the **Hide Categories** menu item or button to toggle between category mode and non-category mode.

You can also toggle between showing and hiding broker schema (in a message flow project) and namespaces (in a message set project), by selecting **Hide Namespaces** from the view menu (shown by a down arrow) in the Broker Development view toolbar. By default, broker schema and namespaces are shown. This option is available only in category mode.

You can toggle between showing and hiding file extensions for files that are under virtual folders, broker schema (or namespaces) and `messageSet.mset`. To toggle between hiding and showing the file extensions, select **Hide Categorized File Extensions** from view menu (shown by the down arrow) in the Broker Development view toolbar. By default, file extensions are shown. Other files and folders that are directly under the project are not affected by this option.

By default, the hierarchy view contains the resource files for all of your projects. However, you can limit the number of resources that are displayed in the Broker Development view by organizing and displaying them in *working sets*. A working set is a logical collection of projects, which makes it easier to manage and work with your application projects. When you create a pattern instance project by using the Patterns Explorer view, the pattern instance project is placed into a working set of the same name. For more information about working sets, see "Working sets" on page 42.

You use specialized editors to open and edit all the resources in the Broker Development view, except project files.

Editor view:
Double-click a resource in the Broker Development view to open the associated editor in the Editor view.

The WebSphere Message Broker Toolkit shows the Message Flow editor by default for working with the content of message flow files.

The WebSphere Message Broker Toolkit provides the following specialized editors by default for working with the content of message flow and message set files:

ESQL editor
For viewing and updating the ESQL statements that are associated with Compute, Database, and Filter nodes.

Message Category editor
For creating, viewing, and updating message categories that you define within the MRM domain.

Message Definition editor
For creating, viewing, and updating messages that you define within the MRM domain.

Message Flow editor
For creating, viewing, and updating message flows, and the nodes and user-defined nodes that define them.

Message Mapping editor
For defining transformations between data sources and targets, without the requirement for programming in XPath, XSLT, XQuery, Java, or ESQL.

Message Set editor
For creating, viewing, and updating message sets that you define within the MRM domain.

Outline view:
The Outline view provides a summary of the content of the resource that is currently open in the Editor view.

Click the **Overview Mode** icon  to switch to the Overview view, which provides a useful summary for large, complex message flows because it shows a small-scale version of the flow. Click the **Overview Mode** icon again to return to the Outline view.

Brokers view:
Use the Brokers view to create and work with brokers in the WebSphere Message Broker Toolkit. Brokers that are created on the local system are automatically displayed in the Brokers view. Remote brokers can be added to the Brokers view. When you open or switch to the Brokers view, the WebSphere Message Broker Toolkit attempts to connect to brokers on the local system and to any remote brokers that have been defined; see “Brokers view” on page 6796.

Properties view:
The Properties view displays the properties for the node that is selected in the Editor view. The Properties are grouped and displayed on tabs, which are listed on the left of the Properties view.

Problems view:
The Problems view displays any messages (information and error) that are associated with the resource that is currently open in the Editor view. For example, if you save a message flow that has an error (such as a mandatory property not set), you can check the content of the Problems view to determine any corrections that you must make. When you double-click a Problems view entry, the appropriate editor opens the resource that contains an error, and positions the cursor at the point of the error (where possible).

Deployment Log view:
The Deployment Log view shows the result of deploying brokers that are in the WebSphere Message Broker Toolkit; see “Deployment Log view” on page 6797.

Data Project Explorer view:
The Data Project Explorer view displays Data design and Data development projects. You can use the Data Project explorer to work with these projects and other data objects.

Data Source Explorer view:
The Data Source Explorer view displays database connections. You can use the Data Source Explorer view to create database connections.

Patterns Explorer view:
The Patterns Explorer view displays patterns that you can use in the WebSphere Message Broker Toolkit. You can use the Patterns Explorer view to configure and apply patterns for use in your environment. Select a pattern in the Patterns Explorer view to see more information about the pattern; see “Choosing a pattern” on page 1313.

Tasks view:
The Tasks view displays outstanding tasks to be completed. You can create tasks within the Tasks view by selecting **Add Task**. For example, if you would like to record reminders to follow up on something later, add it to the Tasks view. When you add a task, you have the option of associating it with a resource so that you can use the Tasks view to quickly open that resource for editing.

Broker Application Development perspective toolbar:

You can add icons to the toolbar for the actions that you might want to complete in the perspective. To add icons to the toolbar, complete the following steps.

1. Click **Window** > **Customize Perspective**. The Customize Perspective dialog box is displayed.
2. Click the **Command Groups Availability** tab. This tab displays the available command groups and their corresponding menu bar details and toolbar details.
3. Click a command group name to display its toolbar details. The available command groups and toolbar details for the icons that you might want to use in the perspective are shown in the **Available command groups** table.
4. Select the command groups that you want to add, and click **OK**. The toolbar displays the icons from the selected command groups.

Available command groups	Toolbar details: icon	Toolbar details: label and action
ESQL Actions		Generate a New Message Flow ESQL File
Mapping Actions		Create a Message Map File
Message Creation		Create a New Message Set
		Create a New Message Definition File
		Create a New Message Category File
Message Flow Element Creation		Create a New Message Flow Project
		Create a New Broker Schema
		Create a New Message Flow
Message Set		Generate a Web Service Definition File
		Generate HTML Documentation for a Message Set
		Generate an XML Schema from a Message Definition File

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"Packaging and deployment overview" on page 3210
Deployment is the process of transferring data to an execution group on a broker so that it can take effect in the broker. Message flows and associated resources are packaged in broker archive (BAR) files for deployment.

"Flow debugger overview" on page 3158
Use the flow debugger in the WebSphere Message Broker Toolkit to track messages through your message flows.

"Resources" on page 36
The projects, folders, and files that you work with in the WebSphere Message Broker Toolkit workspace are called *resources*. By default, these resources are stored with their metadata in the workspace directory in your local file system. The workspace directory is created the first time that you start the WebSphere Message Broker Toolkit.

"Quick Start wizards overview" on page 1409
You can use a Quick Start wizard to set up the basic resources that are required to develop a message flow application. The wizard sets up and gives names to containers for the resources that you need when you develop a message flow.

"Editors" on page 35
An editor is a component of the WebSphere Message Broker Toolkit. Editors are typically used to edit or browse resources, which are the files, folders, and projects that exist in the workbench.

"Debug perspective"

"Using patterns" on page 1312
Use patterns that are supplied with the WebSphere Message Broker Toolkit to create resources that are used to solve a specific business problem.

Related tasks:

"Creating a working set" on page 575
Create a working set to limit the number of resources that are displayed in the Broker Development view.

"Developing message flow applications from a wizard" on page 1408
A Quick Start wizard sets up the basic resources that are required to develop a message flow application. The wizard sets up and gives names to containers for the resources in which you then develop your message flow.

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

"Debugging a message flow" on page 3157
Use the tasks described in this section of the documentation to manage and work with the flow debugger.

Related reference:

"Resource types in the WebSphere Message Broker Toolkit" on page 6821
Links to information on projects and resource files.

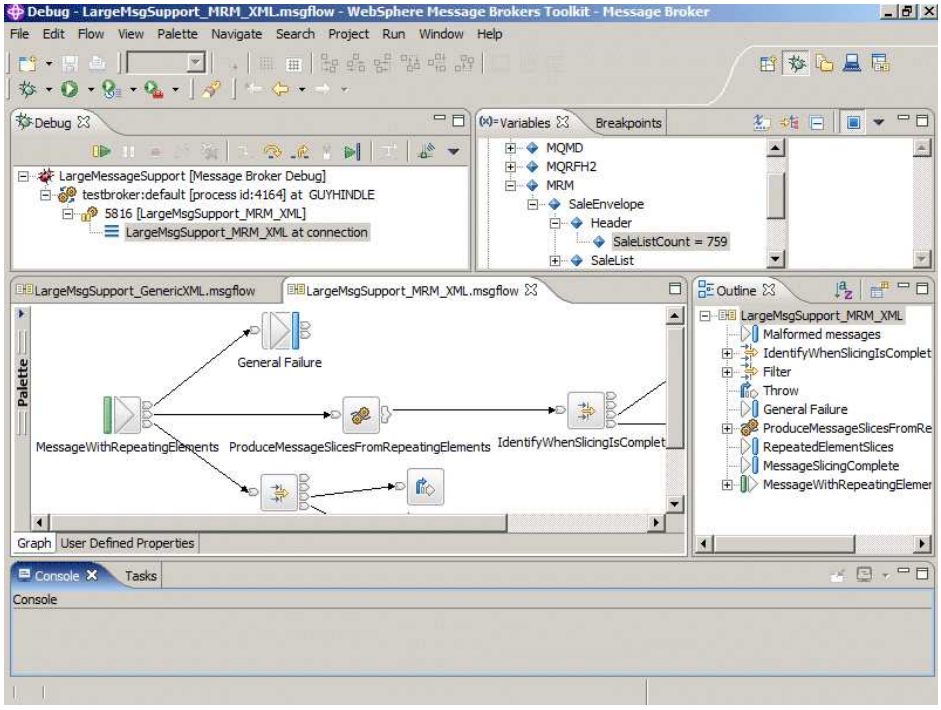
"Editors in the WebSphere Message Broker Toolkit" on page 6793

"Plug-in Development perspective" on page 6792
You can use the Plug-in Development perspective to create user-defined node projects and user-defined nodes in the WebSphere Message Broker Toolkit, however, the preferred option is to use the Broker Application Development perspective.

Debug perspective

The Debug perspective is where you test and debug a graphical representation of your message flows using the message flow debugger.

The following figure shows the Debug perspective of the WebSphere Message Broker Toolkit. In the figure, the LargeMsgSupport_MRM_XML message flow is being debugged.



Debug perspective views: The Debug perspective contains the following views:

Debug view

Displays the deployed message flow types for a selected host to help you manage flow debugging. Toolbar buttons are provided for controlling the execution of the flow. You can start, stop, and resume a flow, step into, and out of a subflow, and step into the source code.

When you attach the flow debugger to the flow runtime engine, the Debug view displays the names of the following flow-related entities:

- The host computer and the flow runtime engine that it is running. This is shown as a concatenation of the names of the following entities, delimited by colons.
  - The host computer
  - The broker
  - The execution group
  - The flow engine

  The entry is identified by the flow runtime engine symbol. For example:

  -  `TestPC01:WMQIV5BR:TestExecution:DataFlowEngine`
- The flows that are deployed in the flow runtime engine, identified by the flow symbol, for example:

  -  `TestFlow`
- When a breakpoint is reached, the flow instances that have been created for each flow, identified by the following symbol, for example:

  -  `3068` (Paused)

In the Debug view, you can perform the following debugging tasks:
- Query a flow runtime engine for currently deployed flows
- Detach a flow runtime engine from the flow debugger
- Resume flow execution

- Run to termination
- Step over a node
- Step into or out of a subflow
- Step over, into, or out of, source code

Breakpoints view

Lists the breakpoints that have been set on connections in your message flow. In this view you can add, disable, enable or remove breakpoints. You can also restrict a breakpoint to one or more specific instances of a message flow using the Properties view.

The Breakpoints view and the Variables view share the same pane. Click one of the tabs to select the view that you want.

The Breakpoints view displays the breakpoints that are set in all instances of a selected flow. Each breakpoint is identified by one of two symbols (as also used in the Message Flow editor) as follows:

-  breakpoint enabled
-  breakpoint disabled

In the Breakpoints view, you can perform the following debugging tasks:

- Remove breakpoints
- Disable or enable breakpoints
- Restrict breakpoints to one or more instances of a flow

Variables view

When a message flow is interrupted by a breakpoint, you can view the message content to check whether the message flow is executing as expected, and to make any changes required.

The Variables view and the Breakpoints view share the same pane. Click one of the tabs to select the view that you want.

The Variables view displays the messages that are currently traveling through the flow. Use the view to examine or change the content of a message in a flow during debugging.

Message Flow editor view

The Message Flow editor graphically displays and debugs flows. In this view you can add breakpoints to the connections of a message flow.

The Message Flow editor also displays any breakpoints that are set in the flow. Each breakpoint is identified by a symbol as follows:

-  breakpoint enabled.
-  breakpoint disabled.
-  flow paused at breakpoint.

Also the editor displays the following symbol above a node:

-  flow paused at a node containing ESQL code or Java code that the flow debugger can step into

In the Broker Application Development perspective, the Message Flow editor is used to create, graphically display, and edit flows. For details of

the other uses of this editor, see the description in "Message Flow editor" on page 6810 and the tasks in "Defining message flow content" on page 1488.

Related concepts:

"Flow debugger overview" on page 3158
Use the flow debugger in the WebSphere Message Broker Toolkit to track messages through your message flows.

Related tasks:

Chapter 10, "Testing and debugging message flow applications," on page 3143
Use the flow debugger to track messages through your message flows and check for errors, or use the Test Client to test your message flows in a safe environment before they are used on a production system.

Related reference:

"Broker Application Development perspective" on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

"Message flow debugger" on page 6718
The flow debugger is a visual interface that supports the debugging of message flow applications in the WebSphere Message Broker Toolkit.

Plug-in Development perspective

You can use the Plug-in Development perspective to create user-defined node projects and user-defined nodes in the WebSphere Message Broker Toolkit, however, the preferred option is to use the Broker Application Development perspective.

A user-defined node is an Eclipse plug-in that adds a category of nodes to the Message Flow editor palette. To use the Broker Application Development perspective to create user-defined node projects and user-defined nodes in the WebSphere Message Broker Toolkit, see "Broker Application Development perspective" on page 6784.

The New User-defined Node Project wizard creates the supporting WebSphere Message Broker Toolkit files for your user-defined node.

Related concepts:

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related tasks:

"Creating the user interface representation of a user-defined node in the WebSphere Message Broker Toolkit" on page 3079
When you are developing a user-defined node in Java or C only, you must create the user interface representation of the node in the WebSphere Message Broker Toolkit.

Related reference:

"Plug-in Development projects and files" on page 6825

"Message Node editor" on page 6818

"Broker Application Development perspective" on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

Editors in the WebSphere Message Broker Toolkit

More information about each of the editors provided by WebSphere Message Broker Toolkit can be found in the following topics:

Related concepts:

"Editors" on page 35
An editor is a component of the WebSphere Message Broker Toolkit. Editors are typically used to edit or browse resources, which are the files, folders, and projects that exist in the workbench.

Related tasks:

"Changing ESQL editor settings" on page 2409
When you open an ESQL file in the editor view, you can tailor the editor appearance by changing editor settings.

Related reference:

"Editor preferences and localized settings"
If you want to share the files that are associated with message flow and message set development, you must ensure that the files and the content of these files are compatible and can be shared by users working on different systems and in different locales.

Editor preferences and localized settings

If you want to share the files that are associated with message flow and message set development, you must ensure that the files and the content of these files are compatible and can be shared by users working on different systems and in different locales.

You can share the following files:

- Message flow definition files (`.msgflow`)
- ESQL files (`.esql`)
- Mappings files (`.msgmap`)
- Message set files (`.mset`)
- Message files (`.mxsd` and `.xsd`)
- Message category files (`.category`)

Restrict the names of these files to a specific set that all your users can understand, for example the characters in the US English code page (a to z, A to Z, 0 to 9, plus special characters underscore and hyphen). This is not a restriction, nor is it enforced, but the users must be able to use each others resources without confusion or error.

The contents of all files listed above, with the exception of ESQL files, are stored in UTF-8, and are therefore common, regardless of individual user settings.

The contents of the ESQL files, which users can edit directly, are governed by the editor preference settings that you specify in the workbench:

1. Click **Window** > **Preferences**.
2. Expand the Workbench item on the left. Click **Editors**. Within "Text file encoding", select the code page that you want content to be stored in:
   a. If you select Default, this represents the default code page for the locale in which you are currently working.
   b. If you select Other, you can choose one from the list of available code pages.

If you expect users to share ESQL files, by using either shared drives across a LAN, or a shared repository such as CVS, you must ensure that all users set the editor preference to be the same value. This allows each user to work in their current locale, but be able to access and work with files created by any other user. For example, select UTF-8 to ensure consistent ESQL file content that is accessible by all users regardless of current locale.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Message modeling" on page 1154
Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

"Publish/Subscribe" on page 2215
Publish/subscribe is a style of messaging application in which the providers of information (publishers) are decoupled from the consumers of that information (subscribers).

Related tasks:

Chapter 9, "Developing message flow applications," on page 1019
Develop message flows to process your business messages and data.

"Constructing message models" on page 2838
This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

Related reference:

"Perspectives in the WebSphere Message Broker Toolkit" on page 6783

"Editors in the WebSphere Message Broker Toolkit" on page 6793

Broker Archive editor

Use the Broker Archive editor to create and manage broker archive (BAR) files.

The Broker Archive editor is available in both the WebSphere Message Broker Toolkit and the WebSphere Message Broker Explorer. Using this editor, you can add, edit, build, and remove deployable resources to a BAR file. You can also use command-line tools to create BAR files.

Broker Archive editor views:
The Broker Archive editor includes these views:

Prepare view

The Prepare view displays the options for creating a new broker archive file. The Prepare view is available only in the WebSphere Message Broker Toolkit. Use the Prepare view to add and build deployable resources to the BAR file. All available resources in the current workspace (including Java, XML and XSL/XSLT files, message flows, message sets, and adapters) are included with each build type. Resources that are identified but that do not match resources in the workspace are highlighted.

You can select the items that you want to include in the broker archive, and use the build options to perform the following tasks:

- Select items to add or remove from the broker archive
- Include source files
- Override the properties of configurable items

The Build options are:

- **Include source files** to include source files
- **Remove contents of Broker Archive before building** to remove all existing contents of the broker archive file before building the new broker archive file
- **Override configurable properties values** to override the existing configurable property values. Check this option to override any value set by the flow compiler, that is, the values from the message flow. If you do not check this option the properties in the `broker.xml` file are used when compiling a message flow.

Manage and Configure view

The Manage and Configure view displays the message flows, message sets, and other files that are currently in the BAR file. Use the icons to add, edit, and remove files from the BAR file. When you select a message flow in the list, its properties, including configurable properties, are displayed in the Properties view. You can also expand the message flows to display the nodes in the message flow. When you select a node the associated configurable properties are displayed in the Properties view. To edit a property, replace the current value with a new value.

You can filter the content in the Manage and Configure view, by selecting one of the available options from the Filter by list:

- Built resources
- Built resources with source
- Configurable properties

To display resources with specific names, type a filter string in the Filter by field. For example, to display all resources that contain SWIFT in the name, type SWIFT into the Filter by field.

User Log view

The User view displays the build information for the broker archive user log. Use the User Log view to review or clear the details in the user log of the BAR file.

Service Log view

The Service view displays system information for the current build in the broker archive Service log. Use the Service Log view to review or clear the details in the service log of the BAR file.

Broker Archive editor toolbar:

The icons on the toolbar in the Manage and Configure page, and their actions are shown in the following table.

Icon	Label	Action
	**Build**	Builds a broker archive file, making any changes that have been specified
	**Remove**	Removes message flows, message sets, or other items from this archive
	**Edit**	Edits selected message flows, message sets, or other items in this archive
	**Add**	Adds message flows, message sets, or other items to this archive

Related concepts:

"Broker archive" on page 3216
The unit of deployment to the broker is the *broker archive* or *BAR* file.

"Version and keyword information for deployable objects" on page 1443
Use the Broker Archive editor to view the version and keyword information of deployable objects.

Related tasks:

Chapter 11, "Packaging and deploying," on page 3209
Package resources that you create in the WebSphere Message Broker Toolkit, such as message flows, and deploy them to execution groups on brokers.

Related reference:

"**mqsireadbar** command" on page 3697
Use the **mqsireadbar** command to read a deployable BAR file and identify its defined keywords.

"Broker Application Development perspective" on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

Brokers view

Use the Brokers view to create and work with brokers in the WebSphere Message Broker Toolkit.

By default, the Brokers view is displayed at the bottom of the Broker Application Development perspective in the WebSphere Message Broker Toolkit. If the Brokers view is not displayed, you can show it by clicking **Window** > **Show View** > **Other** > **Broker Runtime** > **Brokers**.

Brokers that are created on the local system are automatically displayed in the Brokers view. You can add remote brokers to the Brokers view. When you open or switch to the Brokers view, the WebSphere Message Broker Toolkit attempts to connect to brokers on the local system, and any remote brokers that have been defined. Warnings and errors might be displayed if the WebSphere Message Broker Toolkit cannot connect to brokers, for example, if the broker is stopped, or the queue manager listener is not running.

Right-click the Brokers folder in the Brokers view to display the following options:

- **New Local Broker**
- **Connect to a Remote Broker**

If you specify a keystore or truststore in the remote connection information, you are prompted to enter a password for the keystore or truststore when you connect to the remote broker.

- **Connect to a Remote Broker Using *.broker File**
- **Refresh**

Related tasks:

Chapter 6, "Configuring brokers for development environments," on page 563
Set up application development environments on Linux on x86 or Windows to create, test, and deploy message flows and associated resources.

Chapter 8, "Administering brokers and broker resources," on page 899
Administering brokers and associated broker resources includes the tasks that you perform frequently to activate and manage those resources. Choose the method you prefer to administer your brokers and associated resources.

"Connecting to a remote broker" on page 902
To administer your brokers by using the WebSphere Message Broker Toolkit or WebSphere Message Broker Explorer, you must connect to the broker.

"Deploying resources" on page 3234
Deploy message flow applications to execution groups by sending a broker archive (BAR) file to a broker, which unpacks and stores the contents ready for when your message flows are started.

"Attaching the flow debugger to an execution group for debugging" on page 3160
Before you can debug your message flow, you must attach the flow debugger to the execution group where your flow is deployed, then start a debugging session.

Related reference:

"WebSphere Message Broker Toolkit" on page 6783
The WebSphere Message Broker Toolkit provides your application development environment on Windows and Linux on x86.

Deployment Log view

The Deployment Log view shows the result of deployment actions on brokers in the WebSphere Message Broker Toolkit.

The Deployment Log view is displayed at the bottom of the Broker Application Development perspective in the WebSphere Message Broker Toolkit. If the Deployment Log view is not displayed, you can show it by clicking **Window** > **Show View** > **Other** > **Broker Runtime** > **Deployment Log**.

Right-clicking in the Deployment Log view displays a menu with the following item:

- **Clean Log**

  Clears all log entries from the Deployment Log view, but does not delete the log entries from the administration log.

  Messages are automatically cleared after 72 hours.

Columns in the Deployment Log view:
Details

The Details column displays the message number, the name of the broker that the deployment message has come from, and the detail of the deployment message.

Timestamp

The Timestamp column displays the date and time for each of the deployment messages from the broker.

Related tasks:

"Viewing Administration log information" on page 1007
You can view Administration log information by using either the WebSphere Message Broker Explorer, or the CMP API.

"Saving Administration log information" on page 1008
Save the Administration log information that is written to the Administration Log view in the WebSphere Message Broker Explorer.

"Clearing Administration log information" on page 1009
Clear Administration log information to reduce the size of the log by using either the WebSphere Message Broker Explorer or the CMP API.

Related reference:

"Administration Log view" on page 6840
The Administration Log view shows administration requests and their results, changes made to objects, and the result of deployment actions on brokers. The Administration Log view can also describe configuration changes that have been automatically applied as a result of moving the broker from one version to another; for example, after applying maintenance.

ESQL editor

The ESQL editor is the default editor provided by the Broker Application Development perspective for editing ESQL (`.esql`) files.

The editor is launched in the editor area when you select the menu item **Open ESQL** for a Compute, Database, or Filter node, or when you double-click an ESQL file in the Broker Development view.

ESQL editor views:
The ESQL editor has the following views:

Broker Development view
The Broker Development view shows all the resources in your workspace, that is all message set resources and all message flow resources, including ESQL files.

Editor view
The editor view shows the contents of the resource that is currently open. It also shows tabs for each of the resource that you have open so that you can quickly switch between them.

Outline view
The Outline view displays any schemas, defined constants, modules, and routines that you have referenced in this ESQL file.

Problems view
The problems view displays the warning and error messages that are generated by the editor's validation when you save the ESQL file. If you double-click an error, the editor indicates where it is located by moving the cursor to the corresponding ESQL code.

ESQL editor functions:
The ESQL editor provides:

- A context sensitive Content Assist. You can invoke Content Assist from the Edit menu or, on some systems, by pressing Ctrl+Space if that has not been assigned to another function.

  Content Assist helps you construct references to the content of the Properties folder. When you use the ESQL editor with predefined messages, it also helps you construct field references.

  When you use the ESQL editor with the database schema definitions, Content Assist helps you construct schema, table, and column references. You can also use the ESQL editor to call user-defined maps.

  When you create functions and procedures within the ESQL file, the names that you define must not start with the characters IBM_ (IBM underscore).

  Note: To get Content Assist to work, you must set up a project reference from the project containing the ESQL or mappings to the project containing the message set. For information about setting up a project reference, see "Project references" on page 44.
- Automatic code formatting.

Right-click in the editor view to access the following additional functions:

- **Undo** and **Revert File**: To undo a change that you have made to the ESQL file, click **Undo**. If you want to undo all the changes since the last time you saved, you can reinstate them by clicking **Revert File**.
- **Cut**, **Copy**, and **Paste**: Standard editor functions.
- **Shift Right** and **Shift Left**: Standard editor functions.
- **Save**: Save your changes.
- **Comment** and **Uncomment**: Click **Comment** to change a line of ESQL code into a comment. Click **Uncomment** to change a comment into a line of ESQL code.
- **Format**. This function formats all selected lines of code (unless only partially selected, when they are ignored), or, if no lines are selected, formats the entire file (correcting alignments and indentation).
- **Organize Schema Paths** and **Add Schema Path**: These functions assist you with broker schema management.

  Click **Organize Schema Paths**, and a broker schema that contain procedures or function called by the ESQL file is automatically added to the PATH statement (if you have not already added it). This function scans the ESQL file for instances of procedures or function residing in schemas not already fully qualified in the file.

  Click **Add Schema Path** when you code a call to a procedure or function residing in a different broker schema to paths you have included on the PATH statement, and this schema is added to the PATH statement. Ensure that the cursor is on the name of the procedure you are calling.

ESQL editor preferences:
You can modify settings that affect the way the ESQL code is handled:

- Editor Settings (how code is displayed in the editor view):
  - Text font
  - Displayed tab width (default 4)
  - Background and foreground colors (for comments, statements, and so on)
- Validation Settings (what level of validation is performed when you save the file):

  Validation detects four potential problems:

Unresolved identifiers
The validator attempts to resolve all identifiers that you have referenced (for example, a message field).

Message references mismatch message definition
If a message definition exists (messages in the MRM domain only), the validator checks that the use of the reference is consistent with its definition (for example, the action against a numeric field is a valid numeric action).

Database references mismatch database schema
The validator checks that the use of the reference is consistent with the database schema (for example, the action against a numeric field is a valid numeric action).

Use of deprecated keywords
The validator checks if you have used keywords that have been deprecated in this release.

For each of these situations, select one of the following validation settings:

Ignore No validation is performed.

Warning
The validator writes warning messages to the Problems view for each potential problem that it detects. Warning is the default setting.

Error The validator writes error messages to the Problems view for each potential problem that it detects.

Validation does not check that you have specified names in the case in which you declared them. The names of modules, functions, and procedures are not case sensitive; all other names (schemas, constants, variables, and labels) are case sensitive. Check that the names that you use match the declarations for those names, because the broker handles these names in a case sensitive way, and generates a runtime error if they do not match.

For details of how to change these preferences, see "Changing ESQL preferences" on page 2408.

ESQL editor toolbar:
The ESQL editor does not provide additional icons and actions on the toolbar.

Related concepts:

"Project references" on page 44
When a projects refers to other projects in the workspace, this is called a *project reference*. When one project references another, the files in the referenced project are available for use by the referring project. For example to configure certain nodes, to create libraries of reusable message flows, and to enable Content Assist in the ESQL editor.

Related tasks:

"Developing ESQL" on page 2370
Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

"Changing ESQL preferences" on page 2408
You can modify the way in which ESQL is displayed in the editor and validated by the editor:

Related reference:

"ESQL statements" on page 5067
You can use ESQL statements to manipulate message trees, update databases, or

interact with nodes.

"Calling ESQL functions" on page 5168
Most ESQL functions belong to a schema called SQL and this is particularly useful if you have functions with the same name.

"Broker Application Development perspective" on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

"Editor preferences and localized settings" on page 6793
If you want to share the files that are associated with message flow and message set development, you must ensure that the files and the content of these files are compatible and can be shared by users working on different systems and in different locales.

Impact Analysis view

The Impact Analysis view shows information about selected resources, which changes have been marked as complete, and previous results. You can also use this view to copy results to the clipboard.

The Impact Analysis dialog box

When you have run impact analysis, you see a list of affected resources in the Impact Analysis dialog box.

- At the top is a one-line description of the operation.
- The middle table shows the primary impacts, and therefore the actions that must be performed to make this change.
- The bottom table shows secondary impacts (other objects in the workspace that are affected by this change.) Some secondary impacts might not be shown; for more information, see "Impact analysis: reference" on page 4174.

For both tables, you can reorder columns by dragging the column heading to a new location, and sort information in each column by clicking the column heading. No changes are made to the objects listed, and you cannot use this window to change any of the objects.

You can click **Copy Analysis to Clipboard** to paste the results of the analysis into an external application.

- If the application supports tables, the information is pasted as a table. Applications that support tables include Lotus Symphony® spreadsheets, Microsoft Word, and Microsoft Excel.
- If the application does not support tables, the information is pasted as text.

To switch easily between the impact analysis results and file editors, click **Show in view** to view the results in the Impact Analysis view.

The Impact Analysis view

After you have run impact analysis, the Impact Analysis view is displayed at the bottom of the Broker Application Development perspective in the WebSphere Message Broker Toolkit.

The left pane lists the affected resources and the right pane provides some tips to guide you. If you select one of the resources in the left pane, more information is provided in the right pane.

Resources that you have not selected are in bold. After you have viewed an item in the list, it is no longer bold.

When you have made a change in the appropriate file editor, you can mark the item in the results view as complete by clicking **Mark as complete** (represented on the toolbar by a check mark). The completed item in the list is crossed out.

You can view previous impact analysis results by clicking the arrow to the right of the show history icon.

Related concepts:

"Impact analysis: analyzing the effects of planned changes to your applications" on page 1150
When developing an application, you might want to change the names of some artifacts, or move them. Impact analysis reports the artifacts that are likely to be affected by a particular change.

Related tasks:

"Analyzing planned changes to message flows" on page 1436
Use impact analysis to analyze the effect of renaming or moving message flows, including subflows.

"Analyzing planned changes to message set resources" on page 1165
Use impact analysis to analyze the effect of renaming message definition or deployable WSDL files.

"Analyzing planned changes to message model objects" on page 2897
Use impact analysis to analyze the effect of renaming message model objects.

"Analyzing the impact of changes to message maps" on page 2234
Use impact analysis to analyze the effect of renaming or moving message maps.

"Analyzing planned changes to ESQL objects" on page 2403
Use impact analysis to analyze the effect of renaming or moving ESQL objects, or moving or changing the broker schema of an ESQL file.

"Enabling and disabling indexing" on page 1454
Enable indexing to support impact analysis.

Related reference:

"Impact analysis: reference" on page 4174
Some artifacts are excluded from secondary analysis.

Message Category editor

The Message Category editor is the default editor provided by the Broker Application Development perspective for working with message category (`.category`) files in a message set.

The editor is launched in the editor area when you open an existing message category file using the Broker Development view, or when you create a new message category file by using the New Message Category File wizard.

Message Category editor views:
The Message Category editor has the following views:

Broker Development view
: The Broker Development view shows a hierarchical view of all the resources that are currently in your workspace. By expanding the folder for a message set project, you can see the resources, including the message category file or files, that this message set project contains.

Editor view

You edit the properties of message category in the editor area of the Message Category editor, which has a single tab, the **Properties** tab. This tab provides:

- The Properties Hierarchy, which shows the hierarchy for a message category file and the messages that have been added to it.
- The Details view, which when you select a message category file, shows the properties of the message category, or of an individual message in the category, depending on what you have selected in the Properties Hierarchy.

There are tabs across the top of the editor area containing the file name for each file that you have open, so that you can quickly switch between them.

Problems view

Each time you save a change within a message set project, the content is validated to ensure that the message model follows certain rules. All informational, warning or error messages relating to the validation appear in this view.

Properties Hierarchy functions:

Right-click in the Properties Hierarchy to display the following menu items:

- **Add Messages** (available at the **Message Category** level only) adds a new message to a message category file.
- **Undo** and **Redo** allow you to undo and redo changes to message category files. These two menu items apply to adding new messages, and also to changes that you make in the Details view. To undo a change that you have made to a message category file but not yet saved, click **Undo**. If you undo a change, you can reinstate it by clicking **Redo**.
- **Delete** (available at the individual **Message** level only) deletes the selected message from the message category file.

Details view functions:

In the Details view, use the **Message Category Kind** field to specify whether the message category will be used to generate WSDL (Web Services Description Language) files.

If the **Message Category Kind** is WSDL, you must provide definitions in the **Role Name** and **Role Type** fields for each message that you add to the message category file. These two fields appear when you select a message in the Properties Hierarchy.

Create all required documentation for a message set in the **Documentation** field. Right-click in this field to display a menu of standard text editing menu items that apply specifically to this field. You can undo changes, cut, copy, paste, and delete text, and select all the text in the field.

Related concepts:

"Message categories" on page 1200
Message category files have the suffix `.category`. These files are optional. You can have many message category files in a message set.

Related tasks:

"Working with a message category file" on page 2923
This topic area lists the tasks that are involved when working with a message category file.

"Generating a WSDL definition from a message set" on page 2968
To ensure the highest interoperability of your Web services, use the document style of WSDL whenever possible. If rpc style WSDL is necessary, use literal encoding.

Related reference:

"Broker Application Development perspective" on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

"Message category properties" on page 5413
A message category provides a way of grouping your messages.

Message Definition editor

The Message Definition editor is the default editor provided by the Broker Application Development perspective for editing message definition (`.mxsd`) files.

The editor is launched in the editor area when you open an existing message definition file using the Broker Development view, or when you create a new message definition file by using the New Message Definition File wizard.

You use the Message Definition editor to:

- Edit message definitions created by importing data structures using the XML Schema, DTD, C or COBOL importers. The message definition file that the import process creates is automatically populated with the imported content, which you can then edit as required.
- Populate empty message definition files with message model objects by creating the elements, attributes, groups, types and messages needed to represent your message formats. The message model that you create can consist of both logical and physical information, if appropriate physical formats exist in the message set.

Message Definition editor views:

The Message Definition editor has the following views:

Broker Development view

The Broker Development view shows a hierarchical view of all the resources that are currently in your workspace. By expanding the folder for a message set project, you can see the resources, including message definition file or files, that this message set project contains.

Outline view

You use the Outline view to select message set objects to display in the editor area. When you open a message definition file, the Outline view shows a hierarchical view of the messages, types, groups and elements and attributes that the selected message definition contains.

Editor view

You edit the properties of message set objects in the editor area, which displays the message definition information for the object that you have currently selected in the Outline view. There are tabs across the top of the editor area for each file that you have open, so that you can quickly switch between them.

In the editor area, you can make any of the following types of change:

- Edit the logical structure of a message.
- Create and edit the physical structure and properties of a message.
- Create message definitions.

- Create common constructs within a message set for use with other message definition files.

The Message Definition editor comprises an Overview editor and a Properties editor. You switch between them by clicking the two tabs in the lower left of the editor area. You can change the order in which the tabs are displayed by changing the message set preferences.

Problems view

Each time you save a change within a message set project, the content is validated to ensure that the message model follows certain rules. All informational, warning or error messages relating to the validation appear in this view. Double-click a message to display the properties of the relevant object in the editor area. The location of this object is highlighted at the appropriate levels in the Broker Development view and the Outline view.

Message Definition editor functions:

For information on the functions that the Message Definition editor provides, refer to the following topics:

- "Message Definition editor: Outline view" on page 6806
- "Message Definition editor: Overview editor" on page 6807
- "Message Definition editor: Properties editor" on page 6808

Related concepts:

"Message definition files" on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

"Importing from other model representations to create message definitions" on page 1254
You can add message definitions to your message set by importing application message formats that already exist.

Related tasks:

"Working with a message definition file" on page 2863
Create, open, and delete a message definition file.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

"Linking from one message definition file to another" on page 2921
Add an 'include', or an 'import' to the file that you want to reference.

Related reference:

"Broker Application Development perspective" on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

"Message definition file properties" on page 5409
The properties of a message definition file.

"Message set editor" on page 6819
The Message set editor is the default editor provided by the Broker Application Development perspective for editing message set (`messageSet.mset`) files.

"Message set preferences" on page 5366
Preferences for message sets.

Message Definition editor: Outline view:

When you open a message definition file, the hierarchy for the selected file appears in the Outline view. Use the Outline view to select objects whose details you want to view and edit in the Message Definition editor area.

Global actions:
The toolbar icons in the Outline view provide a number of global actions that apply to all levels of the displayed hierarchy. The icons in the toolbar and their actions are shown in the following table.

Icon	Label	Action
	**Find Element**	Finds the specified message definition construct.
	**Sort global constructs alphabetically**	If selected, global constructs are sorted alphabetically. If not selected, global constructs are displayed in the order that you create them.
	**Collapse All**	Collapses all the levels that are currently displayed in the Outline view.

You can also select these global actions from the toolbar menu that appears when you click the icon.

Common editing and navigation actions:

The Outline view provides the following common actions, which are also available in the Overview editor. Right-click in the Outline view or Overview editor to display the menu items.

- **Copy** and **Paste** provide standard editing functions.
- **Undo** and **Redo** allow you to undo and then redo changes that you have made. To undo a change, click **Undo**. If you undo a change, you can reinstate it, if you want to, by clicking **Redo**.
- **Go To Declaration** takes you to the appropriate point in the hierarchy (global group from a group reference, global attribute from an attribute reference, global element from an element reference, global element from a message) either in the same or in a different message definition file.
- **Go to Type Definition** takes you to the appropriate point in the hierarchy.
- **Go to Properties** switches to the Properties view in the editor.
- **Rename** allows you to rename the selected message set object.
- **Delete** deletes the selected message set object.
- **Save** saves the current message set definition file.

Options for adding message model objects:

The Outline view provides options for adding the message model objects shown in the following list. These options are also available in the Overview editor.

- Messages

  Message

  Message From Global Element

  Message From Global Type

- Types
  - Complex Type
  - Simple Type Restriction
  - Simple Type List
  - Simple Type Union
- Groups
  - Group
  - Attribute Group
- Elements and Attributes
  - Global Element
  - Global Attribute

Right-click the appropriate level of the displayed hierarchy to display in more detail what can be added at the selected level.

Related concepts:

"Message definition files" on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

"Message modeling concepts" on page 1155
Message modeling is a way of predefining the message formats that are used by your applications.

Related tasks:

"Working with a message definition file" on page 2863
Create, open, and delete a message definition file.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message definition file properties" on page 5409
The properties of a message definition file.

"Message Definition editor" on page 6804
The Message Definition editor is the default editor provided by the Broker Application Development perspective for editing message definition (`.mxsd`) files.

"Message Definition editor: Overview editor"
The Overview editor displays the contents of the message set object that you select in the Outline view, and you can view and change this object's most common properties.

"Message Definition editor: Properties editor" on page 6808

Message Definition editor: Overview editor:

The Overview editor displays the contents of the message set object that you select in the Outline view, and you can view and change this object's most common properties.

The Overview editor appears when you click the Overview tab in the editor area. Use the navigation buttons and links on the toolbar to move back and forth between different objects. In the Overview editor you can make the following types of change:

- Edit the displayed values of any of the following objects:

- Structure
- Type
- Min Occurs
- Max Occurs

  To edit any of these values, click the appropriate field in the Overview editor. Depending on the type of field, this either displays a list from which you can choose, or it enables the field for direct editing.

- Add message model objects as in the Outline view.

  To add a new message model object, right-click **Structure**, then click the appropriate item from the displayed menu.

  For example, the menu for **Messages** provides **Add Message**, **Add Message From Global Element**, or **Add Message From Global Type**.

Related concepts:

"Message definition files" on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

Related tasks:

"Working with a message definition file" on page 2863
Create, open, and delete a message definition file.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

Related reference:

"Message definition file properties" on page 5409
The properties of a message definition file.

"Message Definition editor" on page 6804
The Message Definition editor is the default editor provided by the Broker Application Development perspective for editing message definition (`.mxsd`) files.

"Message Definition editor: Outline view" on page 6806
When you open a message definition file, the hierarchy for the selected file appears in the Outline view. Use the Outline view to select objects whose details you want to view and edit in the Message Definition editor area.

"Message Definition editor: Properties editor"

"Message set editor" on page 6819
The Message set editor is the default editor provided by the Broker Application Development perspective for editing message set (`messageSet.mset`) files.

Message Definition editor: Properties editor:

The Properties editor allows you to change the properties for the selected message set object. The information that appears depends on what is currently selected in the Outline view and Overview editor. The Properties editor appears when you click the Properties tab in the editor area. The navigation buttons and links on the toolbar allow you to move back and forth between different objects. The Properties editor comprises the Properties Hierarchy and Details view.

Properties Hierarchy functions:

When you select a object in the Outline view, the Properties Hierarchy shows the following hierarchy for the message definition file that is currently open in the Message Definition editor:

- Logical properties
- Physical properties
- Documentation

Right-click in the Properties Hierarchy to display a number of menu items. The displayed menu items depend on which level of the hierarchy you select:

- **Undo** and **Redo** are available for all levels in the Properties Hierarchy and apply to changes that you make in the Details view for the currently selected object. To undo a change that you have made to a file but not yet saved, click **Undo**. If you undo a change, you can reinstate it by clicking **Redo**.
- **Add Include** and **Add Import** are available when the top level of the message definition (`.mxsd`) file's hierarchy is displayed in the **Outline** view. They allow you to link message definition files together.
- **Apply default physical format settings** is available for **Physical formats** only and applies the default wire format settings for the selected object.

If you add a new physical format in the message set (`messageSet.mset`) file, this new format is added under **Physical Properties** in the Properties Hierarchy. When you click **Physical Properties**, a link to the Message Set editor appears in the Details view.

Details view functions:

The Details view shows the properties for the object that is currently selected in the Properties Hierarchy. You can save any changes to the properties as you make them.

You create any required documentation for a message set in the **Documentation** field. This field appears when you select an object under **Documentation** in the Properties Hierarchy. Right-click in this field to display a menu of standard text editing menu items that apply specifically to this field and allow you to undo changes, cut, copy, paste and delete text, and select all the text in the field.

Related concepts:

"Message definition files" on page 1171
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

Related tasks:

"Working with a message definition file" on page 2863
Create, open, and delete a message definition file.

"Working with MRM message model objects" on page 2870
Add, configure, and delete objects.

"Linking from one message definition file to another" on page 2921
Add an 'include', or an 'import' to the file that you want to reference.

Related reference:

"Message definition file properties" on page 5409
The properties of a message definition file.

"Message Definition editor" on page 6804
The Message Definition editor is the default editor provided by the Broker Application Development perspective for editing message definition (`.mxsd`) files.

"Message Definition editor: Outline view" on page 6806
When you open a message definition file, the hierarchy for the selected file appears in the Outline view. Use the Outline view to select objects whose details you want

to view and edit in the Message Definition editor area.

"Message Definition editor: Overview editor" on page 6807
The Overview editor displays the contents of the message set object that you select in the Outline view, and you can view and change this object's most common properties.

"Message set editor" on page 6819
The Message set editor is the default editor provided by the Broker Application Development perspective for editing message set (`messageSet.mset`) files.

Message Flow editor

The Message Flow editor is the default editor provided by the Broker Application Development perspective for defining a graphical representation of a message flow in the WebSphere Message Broker Toolkit, and for setting properties for individual message flow nodes.

Open a message flow file (`.msgflow`) in the Broker Development view to launch the appropriate editor in the Editor view. The Editor view is where you select built-in and user-defined nodes, and the connections between them, to define a message flow.

Message Flow editor views:
The Message Flow editor has the following views:

Broker Development view

The Broker Development view shows all the resources in your workspace; that is, all message set resources, and all message flow resources.

Editor view

The editor view shows the contents of the resource that is currently open. It also shows tabs for each of the resources that you have open so that you can quickly switch between them.

When the resource that is open is a message flow, two tabs are displayed, called **Graph** and **User Defined Properties**.

When you select **Graph**, the editor view contains a graphical display of the message flow.

The Palette contains all the available nodes that you can include in the message flow. By default, the palette appears on the left side of the editor view. You can move it to the right side of the editor view by clicking the palette while it is in collapsed mode, and dragging the bar to the right of the editor view.

When you select the **User Defined Properties** tab, a User Defined Properties editor is opened that allows you to change the User Defined Properties of the message flow. The User Defined Properties editor consists of a User Property Hierarchy view and a Details view.

The User Property Hierarchy view displays three icons, **`Add Property Group`**, **`Add Property`**, and **`Delete`** that you can use to update the property hierarchy. When you add a property, a Details view is opened. In the Details view you can define the property Type and Default Value.

Outline view

The Outline view enables you to navigate to a particular node in a message flow, and edit its properties.

Palette view

The Palette view lists the available nodes that you can select and include in the message flow. For more information about the palette and how to customize the layout and settings, see “Message flow node palette” on page 1027.

Overview view

The Overview view provides a useful summary for large, complex message flows because it shows a small-scale version of the flow. Click the tab to show or hide the outline view.

Problems view

This view displays the warning and error messages that are generated by the editor's validation when you save the message flow file. If you double-click an error, the editor indicates where it is located (for example, if you have not set a mandatory property in a node, it opens the Properties view for that node).

Message Flow editor toolbar:
The icons in the toolbar and their actions are shown in the following table.

Icon	Label	Action
	**Manhattan**	Displays all node connections as a series of horizontal and vertical lines
	**Show grid**	Displays a grid of horizontal and vertical dotted lines in the background of the editor area.
	**Grid properties**	Defines the horizontal and vertical spacing of the grid markers, and the gap between the borders of the editor area and the start of the grid markers.
	**Align left**	Lines up the left edge of the currently selected nodes. Enabled only when more than one node is selected.
	**Align center**	Lines up the horizontal center point (between left and right) of the currently selected nodes. Enabled only when more than one node is selected.
	**Align right**	Lines up the right edge of all the currently selected nodes. Enabled only when more than one node is selected.
	**Align top**	Lines up the top edge of the currently selected nodes. Enabled only when more than one node is selected.

Icon	Label	Action
	**Align middle**	Lines up the vertical center point (between top and bottom) of the currently selected nodes. Enabled only when more than one node is selected.
	**Align bottom**	Lines up the bottom edge of the currently selected nodes. Enabled only when more than one node is selected.
	**Show distribute box**	Displays a rectangular box around the currently selected nodes.
	**Distribute horizontally**	Aligns the currently selected nodes with the nearest right or left edge within the distribute box.
	**Distribute vertically**	Aligns the currently selected nodes with the nearest top or bottom edge within the distribute box.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related tasks:

"Defining message flow content" on page 1488
You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

"Changing the palette layout" on page 1490
You can change the layout of the palette in the Message Flow editor.

"Changing the palette settings" on page 1490
How to use the **Palette Settings** dialog box.

"Customizing the palette" on page 1491
If you customize the message flow node palette, you can make it easier to find the nodes that you use most often.

"Resolving problems when developing message flows" on page 3395
Use the advice given here to help you to resolve common problems that can arise when developing message flows.

Related reference:

"Broker Application Development perspective" on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"User-defined nodes" on page 6415
You can define your own nodes to use in WebSphere Message Broker message flows.

Message Flow editor hover-feedback:

In the Message Flow editor, you can display node and connection metadata by holding the mouse pointer over a node or subflow in a message flow. To view metadata information for a node, subflow, or connection:

1. Switch to the Broker Application Development perspective.
2. Open a message flow.
3. In the Message Flow editor, hold the mouse pointer over a node, a subflow, or a node connection in the open message flow by placing the mouse over the element.

A custom tooltip is displayed below the element.

- To turn the pop-up window into a scrollable window, press **F2**.
- To hide the pop-up window, either press Esc or move the mouse away from the node.

Node and subflow feedback

When you hold the mouse pointer over a node or subflow in the Message Flow editor, the pop-up window contains the following metadata information:

- The first line of text contains the node type and the display name of the node instance. This line appears as **bold text**.
- The line below the node type and display name contains the node's short description.
- The final line contains the node's long description.

If the node or subflow has no short or long description associated with it, only the first line of text is displayed.

If the task list contains errors or warnings that are associated with the node or subflow, the error or warning information is displayed instead of the metadata information.

Connection feedback

When you hold the mouse pointer over a connection in the Message Flow editor, hover-help is displayed, containing the names of the output and input terminals that are connected.

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

Related reference:

"Broker Application Development perspective" on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

"Built-in nodes" on page 4293
WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

"User-defined nodes" on page 6415
You can define your own nodes to use in WebSphere Message Broker message flows.

Message Flow editor menus:

All the actions that you can perform in the Editor view of the Message Flow editor are also available from a series of drop-down menus in WebSphere Message Broker. When the Message Flow editor is open, the following menus appear in the Workbench:

Menu	Keyboard shortcut
Flow	Alt + O
View	Alt + V
Palette	Alt + L
Run	Alt + R

The Navigate menu also contains actions specific to the Message Flow editor.

Flow menu

The flow menu contains all the actions that address editing of the flow model. Actions are statically positioned on the menu, and are enabled or disabled when nodes or connections are selected in the editor.

The following actions are available:

Menu item	Keyboard shortcut
Create Connection	C
Add subflow...	A
Rename...	N
Locate Subflow...	L
Promote Property	P
Properties...	R

View menu

The View menu contains all actions specific to the visual presentation of the flow model.

Menu item	Keyboard shortcut
Zoom In	I
Zoom Out	O
Manhattan Layout	M
Show Grid	G
Snap to Grid	G
Grid Properties...	P
Align	A
Distribute	D
Layout	L
Rotate	R

Palette menu

The Palette menu contains all actions specific to the Message Flow editor Palette.

Menu item	Keyboard shortcut
Add Node to Canvas	N
Customize	T
Settings...	S
Revert Palette to Defaults	T

Run menu

The Run menu contains the flow debug actions.

Menu item	Keyboard shortcut
Add Breakpoint	A
Remove Breakpoint	E
Add Breakpoints After Node	A
Add Breakpoints Before Node	B

Navigate menu

The Navigate menu contains the following actions specific to the Message Flow editor:

Menu item	Keyboard shortcut
Open Subflow	O
Open Source	S

Related concepts:

"Message flows overview" on page 1022
A message flow is a sequence of processing steps that run in the broker when an input message is received.

"Flow debugger overview" on page 3158
Use the flow debugger in the WebSphere Message Broker Toolkit to track messages through your message flows.

Related tasks:

"Adding a node by using the keyboard" on page 1498
You can use the keyboard to perform tasks in the Message Flow editor, such as adding a node to the canvas.

Related reference:

"Message Flow editor" on page 6810
The Message Flow editor is the default editor provided by the Broker Application Development perspective for defining a graphical representation of a message flow in the WebSphere Message Broker Toolkit, and for setting properties for individual message flow nodes.

"Broker Application Development perspective" on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

"WebSphere Message Broker Explorer and WebSphere Message Broker Toolkit keyboard shortcuts" on page 6828
You can navigate all interfaces in the WebSphere Message Broker Explorer and WebSphere Message Broker Toolkit by using the keyboard.

Adding a node by using the keyboard:

You can use the keyboard to perform tasks in the Message Flow editor, such as adding a node to the canvas.

Before you begin

Before you start:
- Ensure that you have created or opened a message flow. For more information, see "Creating a message flow" on page 1431 or "Opening an existing message flow" on page 1433.
- Read the concept topic, "Message flow nodes" on page 1024.

Procedure
1. Open the message flow to which you want to add a node.
2. Open the Palette view or the Palette bar.
3. Select a node in the Palette view or Palette bar by using the up and down arrows to highlight the node that you want to add to the canvas.
4. Add the node to the canvas by using one of the following methods:
   - Press **Alt + L**, then press **N**.
   - Press **Shift + F10** to open the pop-up menu for the Palette, and press **N**.

   The node that you selected in the Palette bar or Palette view is placed on the canvas in the Editor view.

   When you add a node to the canvas, the editor automatically assigns a name to the node, but the name is highlighted and you can change it by entering a name of your choice. If you do not change the default name at this time, you can change it later by following the instructions in "Renaming a message flow node" on page 1502. The default name is set to the type of node for the first instance. For example, if you add an MQInput node to the canvas, it is given the name MQInput. If you add a second MQInput node, the default name is MQInput1; the third is MQInput2, and so on.

Results

You can move the node that you have placed on the canvas by using the keyboard controls described in "WebSphere Message Broker Explorer and WebSphere Message Broker Toolkit keyboard shortcuts" on page 6828.

Related concepts:

"Message flow node palette" on page 1027
The palette in the Message Flow editor contains all the built-in nodes, which are organized into categories, or drawers. A drawer is a container for a list of icons, such as the **Favorites** drawer.

Related tasks:

"Adding a message flow node" on page 1494
When you have created a message flow, add nodes to define its function.

"Dragging a resource from the Broker Development view" on page 1499
Drag a node or a related resource into the Message Flow editor.

Related reference:

"WebSphere Message Broker Explorer and WebSphere Message Broker Toolkit keyboard shortcuts" on page 6828
You can navigate all interfaces in the WebSphere Message Broker Explorer and WebSphere Message Broker Toolkit by using the keyboard.

"Message Flow editor" on page 6810
The Message Flow editor is the default editor provided by the Broker Application Development perspective for defining a graphical representation of a message flow in the WebSphere Message Broker Toolkit, and for setting properties for individual message flow nodes.

"Message Flow editor menus" on page 6813

"Accessibility features for WebSphere Message Broker" on page 135
Accessibility features help users who have a physical disability, such as restricted mobility or limited vision, to use information technology products successfully.

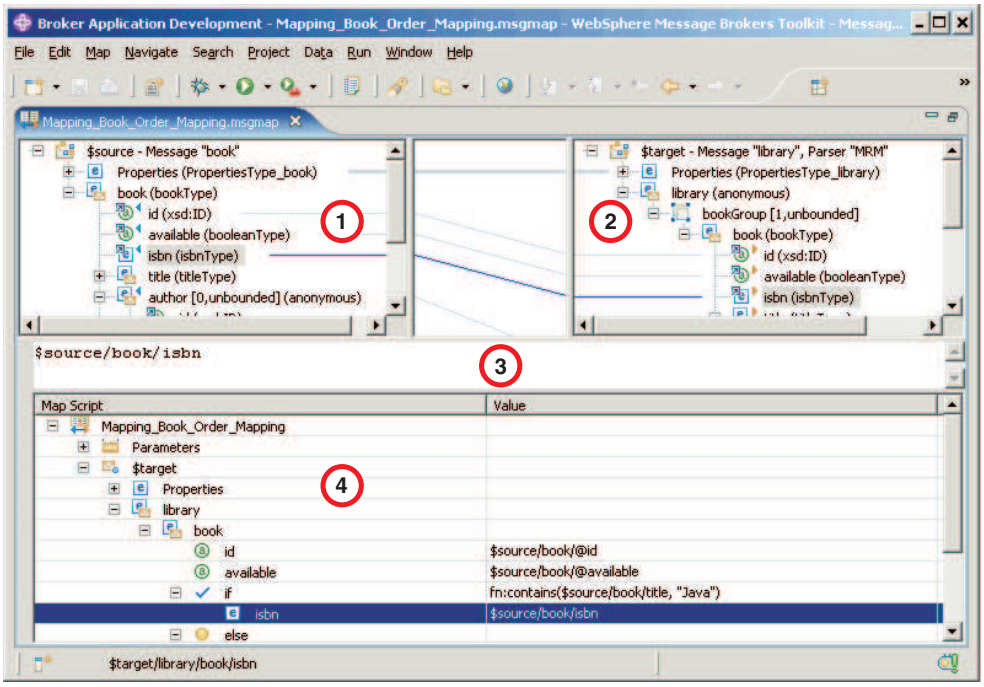
Message Mapping editor

You use the Message Mapping editor to create and edit message mappings.

You can use the Message Mapping editor to set values for:

- The message destination
- Message headers
- Message content

Here is an example of the Message Mapping editor. There are separate panes for working with sources, targets and expressions, as well as a spreadsheet view.



1. **Source pane:** displays a source message or database table
2. **Target pane:** displays a target message
3. **Edit pane:** displays the expression to be used to derive the target element value

4. **Spreadsheet pane:** displays a summary of the mappings in spreadsheet columns (each target field and its value)

Use the Message Mapping editor to perform various mapping tasks.

Wizards and dialog boxes are provided for tasks such as adding mappable elements, working with ESQL, and working with submaps. Mappings that are created with the Message Mapping editor are automatically validated and compiled, ready for adding to a broker archive (BAR) file, and subsequent deployment to WebSphere Message Broker.

Related tasks:

"Using message mappings" on page 2228
Message mappings define the blueprint for creating a message.

"Configuring message mappings" on page 2237
Use the Message Mapping editor to configure a message mapping.

Related reference:

"Message Mapping editor Source pane" on page 4983
Details of the elements present in the Source pane of the Message Mapping Editor.

"Message Mapping editor Target pane" on page 4987
Details of the elements present in the Target pane of the Message Mapping Editor.

"Message Mapping editor Edit pane" on page 4990
Details of how you use the Edit pane of the Message Mapping Editor.

"Message Mapping editor Spreadsheet pane" on page 4991
Details of the actions available to you in the Spreadsheet pane of the Message Mapping Editor.

"Message mappings" on page 4981
Edit and configure message maps using the Message Mapping editor.

Message Node editor

The Message Node editor is the default editor provided by the Plug-in Development perspective for defining the content of user-defined nodes.

The editor is launched in the editor area of the perspective when a new message flow node file (`.msgnode`) is created.

Message Node editor views

The Message Node editor has two tabs:

Terminals
: This tab is for adding, renaming, or deleting input and output terminals for the new node.

Properties
: This tab is for adding properties and property groups to the node, and defining property type.

Related concepts:

"User-defined extensions overview" on page 2971
A user-defined extension is an optional component that you design and create to extend the functionality of WebSphere Message Broker.

Related reference:

"Plug-in Development perspective" on page 6792
You can use the Plug-in Development perspective to create user-defined node projects and user-defined nodes in the WebSphere Message Broker Toolkit, however, the preferred option is to use the Broker Application Development perspective.

"Plug-in Development projects and files" on page 6825

Message set editor

The Message set editor is the default editor provided by the Broker Application Development perspective for editing message set (`messageSet.mset`) files.

The editor is launched in the editor area when you open an existing message set file using the Broker Development view, or when you create a new message set file using the New Message Set File wizard.

Message set editor views:

The Message set editor has the following views:

Broker Development view
The Broker Development view shows a hierarchical view of all the resources that are currently in your workspace. By expanding the folder for a message set project, you can see the resources, including the message set file, that this message set project contains.

Editor view
You edit the properties of message set in the editor area of the Message set editor, which has a single tab, the **Properties** tab. This tab provides:

- The Properties Hierarchy, which provides a hierarchy with four basic categories: **Message Set**, **XML Wire Formats**, **Custom Wire Formats**, and **Tagged/Delimited String Formats**.
- The Details view, which displays the properties relating to the currently selected level in the Properties Hierarchy.

  You select the **Default message domain** you require from the read-only drop down list. When you select the domain from the drop down list, the corresponding **Supported message domains** check box is greyed out and you cannot select it.

  You can add other domains by selecting the appropriate check boxes in **Supported message domains**.

There are tabs across the top of the editor area containing the file name for each file that you have open; this enables you to quickly switch between the open files.

Problems view
Each time that you save a change within a message set project, the content is validated to ensure that the message model follows certain rules. All informational, warning, or error, messages that relate to the validation appear in the this view.

Properties Hierarchy functions:
Right-click in the Properties Hierarchy to display the menu items available for the level of the hierarchy that you have currently selected:

- **Undo** and **Redo** are available for all levels in the Properties Hierarchy and apply to changes that you make in the Details view. To undo a change that you have made to a message set file, but have not yet saved, click **Undo**. If you undo a change, you can reinstate it by clicking **Redo**.

- If **XML Wire Formats**, **Custom Wire Formats**, or **Tagged/Delimited String Formats** are selected, menu items are available for adding physical format layers to a message set file. These are **Add XML Wire Format**, **Add Custom Wire Format**, and **Add Tagged/Delimited String Format**, respectively.
- If an existing physical format is selected, the following menu items are available for the selected physical format layer:
  - **Apply default physical format settings** applies the default physical format settings to the selected physical format layer.
  - **Rename** renames the selected physical format layer for the message set file. Click **Finish** on the displayed window to confirm the change of name.
  - **Delete** deletes the selected physical format layer from the message set file. Click **Finish** on the displayed window to confirm the deletion.

Details view functions:
You use the Details view to define and edit global properties for the message set, and to create and edit properties for the physical format layers that you have added to the message set.

You create any required documentation for a message set in the **Documentation** field. Right-click in this field to display a menu of standard text editing menu items that apply specifically to this field and that allow you to undo changes, cut, copy, paste and delete text, and select all the text in the field.

Related concepts:

"Message sets overview" on page 1162
A message set is a container for grouping messages and associated message resources (elements, types, groups).

Related tasks:

"Working with a message set" on page 2840
Complete a variety of tasks that are involved in working with a message set.

Related reference:

"Broker Application Development perspective" on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

"Message set preferences" on page 5366
Preferences for message sets.

"Message set properties" on page 5371
Message sets have properties that you can set to define their characteristics and the way in which they are processed.

WSDL editor

Use the WSDL editor to browse and edit a WSDL file.

The WSDL editor is launched in the editor area of the Broker Application Development perspective when you either double-click a WSDL file, or right-click a WSDL file and select **Open With**; the WSDL editor is the default editor.

The Design view provides a graphical way to browse and edit your WSDL file.

Each type of top level WSDL object is shown within a tabular view, for example, **service**, **binding**, and **port type**.

The service, binding, and port type objects are linked and a line is displayed to denote a reference, or association, between these objects.

The WSDL editor has three panes called General, Documentation, and Extensions. You can toggle between each pane.

WSDL editor panes:
General pane

The name of the top-level WSDL object you select is displayed at the top of the pane. All WSDL objects have a `Name` field that you can edit.

The top-level WSDL object you select defines the layout of the General pane and the number of rows in the table represents the structure of the object.

Some objects have an additional box that you can select, for example, **Binding**. In this case, clicking **Generate Binding Content** starts the Binding wizard.

Documentation pane

The Documentation pane allows you to add notes.

Extensions pane

You can add an extension to the top-level object you have selected.

Click **Add** to display the `Add Extension Components` window. You can then select from an extension category, a component, or both.

To add an extension category, select the category and click **Add** on the `Add Extension Components` window. On the window that is displayed, enter a name and select a schema. Click **Add** on this second window to add the category.

To add a component, select the component and click **OK** on the `Add Extension Components` window.

Related reference:

"Broker Application Development perspective" on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

"Generate WSDL wizard" on page 6382
The Generate WSDL wizard creates a WSDL definition from a message set.

"Web service consumer message flow generated" on page 1420
This provides additional information in relation to the Configure New Web Service Usage wizard about the message flow generated when the flow is a web service consumer.

Resource types in the WebSphere Message Broker Toolkit

Links to information on projects and resource files.

The following links provide reference information about the projects and resource files that are specific to WebSphere Message Broker Toolkit:

- "Message flow projects and files" on page 6822
- "Message set projects and files" on page 6823
- "Data Design projects and files" on page 6825
- "Plug-in Development projects and files" on page 6825 (for user-defined nodes and parsers)

- "Java projects and files" on page 6826
- "Pattern instance projects" on page 6827

Related concepts:

"Resources" on page 36
The projects, folders, and files that you work with in the WebSphere Message Broker Toolkit workspace are called *resources*. By default, these resources are stored with their metadata in the workspace directory in your local file system. The workspace directory is created the first time that you start the WebSphere Message Broker Toolkit.

"WebSphere Message Broker Toolkit" on page 31
The WebSphere Message Broker Toolkit is an integrated development environment and graphical user interface based on the Eclipse platform.

Related reference:

"Rules for naming workspace objects" on page 6827

Message flow projects and files

You develop message flows in the Broker Application Development perspective. Message flow resources are maintained within message flow projects. A message flow project compiles to one or more deployable flows.

Message flow projects contain the following resource files:

.msgflow files
: A message flow file is a graphical representation of a message flow, containing message nodes and connections. It also contains property values and overrides to define, implement and control the behavior of the message flow nodes. The New Message Flow Wizard automatically creates this file.

  Default editor: Message Flow editor

.esql file
: (Optional) ESQL modules containing compute, filter, and database ESQL functions and procedures. If none of the message flows defined in the project include any nodes that require ESQL, you do not need to create an ESQL file.

  Default editor: ESQL editor

.msgmap file
: (Optional) Mapping transformation descriptions for messages and databases. If none of the message flows defined in the project include any nodes that require mappings, you do not need to create a mapping file.

  Default editor: Message Mapping editor

You can define more than one message flow, each in a separate `.msgflow` file, in the same message project.

Related concepts:

"WebSphere Message Broker Toolkit" on page 31
The WebSphere Message Broker Toolkit is an integrated development environment and graphical user interface based on the Eclipse platform.

"Resources" on page 36
The projects, folders, and files that you work with in the WebSphere Message Broker Toolkit workspace are called *resources*. By default, these resources are stored with their metadata in the workspace directory in your local file system. The workspace directory is created the first time that you start the WebSphere Message

Broker Toolkit.

Related tasks:

"Creating a message flow" on page 1431
Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

Related reference:

"Rules for naming workspace objects" on page 6827

"Message Flow editor" on page 6810
The Message Flow editor is the default editor provided by the Broker Application Development perspective for defining a graphical representation of a message flow in the WebSphere Message Broker Toolkit, and for setting properties for individual message flow nodes.

Message set projects and files

A message set project contains all the resources associated with exactly one message set. A message set is a logical grouping of messages and the objects (elements, types and groups) that comprise them. It contains the following resources, which you create and maintain in the Broker Application Development perspective.

messageSet.mset file
: A message set file contains those message model properties that are common across all the content of the message set. It also contains the physical format definitions for the message set. A message set project must contain exactly one message set file.

  Default editor: "Message set editor" on page 6819

.mxsd files
: A message definition file contains, in XML Schema form, the logical model and associated physical model for one or more messages. Each message set requires at least one message definition file to describe its properties, and you can have as many as you want within the same message set. One message definition file can, if required, reference XML Schema objects in another message definition file.

  You can create a message definition file in a message set by importing any of the following data structures:

  - XML Schemas
  - XML DTD
  - C header files
  - COBOL copybooks

  Importing any of these data structures automatically creates the message definition file and its content for you. Alternatively, you can create a blank message definition file and add the message definitions yourself.

  Default editor: "Message Definition editor" on page 6804

  Previous versions of the message set model
  The message definition contains the same application data that, previously, was stored in the following separate files:

  - Message set (MRProject) that contains wire formats and their default properties
  - Messages (MRMessage) that provides a unique name for a message and specifies its element
  - Elements (MRElement) that defines a node in the message tree

- Types (MRType) that specify the structure of elements
- Declaration Qualifiers (MRDeclaration Qualifier) that specify constraints applied to elements in the message model

.category files

A message category file provides you with another way of grouping related messages, for example for documentation generation, or for convenience purposes such as specifying the messages that define a complete request/reply transaction. You can also use a message category file to assist in generating a Web Services Description Language (WSDL) document. Message category files are optional, and you can have as many as you want within the same message set.

Default editor: "Message Category editor" on page 6802

Once you have created and populated a message set, you can use it to generate your message model in several different representations for use by your applications:

- A message dictionary for deployment to the WebSphere Message Broker.
- A W3C XML Schema for use by an application that is building or processing XML messages.
- A Web Services Description Language (WSDL) document that specifies the interface for a Web Service.
- An HTML document for use by business analysts and developers.

Related concepts:

"WebSphere Message Broker Toolkit" on page 31
The WebSphere Message Broker Toolkit is an integrated development environment and graphical user interface based on the Eclipse platform.

"Resources" on page 36
The projects, folders, and files that you work with in the WebSphere Message Broker Toolkit workspace are called *resources*. By default, these resources are stored with their metadata in the workspace directory in your local file system. The workspace directory is created the first time that you start the WebSphere Message Broker Toolkit.

"The message model" on page 1160
The message model consists of the following components.

"XML Schema" on page 1172
XML Schema is an international standard from the World Wide Web Consortium (W3C) that defines a language for describing the structure of XML documents.

"Generate message dictionaries" on page 1271
A *message dictionary* is data structure that describes all of the messages in a message set in a form suitable for deployment to the MRM parser.

"Generate model representations" on page 1270
After you have created and populated a message set, you can generate a message model in several different representations for use by a broker, a parser, or your applications.

Related tasks:

"Working with a message set project" on page 2838
Creating and deleting a message set.

"Working with a message set" on page 2840
Complete a variety of tasks that are involved in working with a message set.

"Working with a message definition file" on page 2863
Create, open, and delete a message definition file.

"Working with a message category file" on page 2923
This topic area lists the tasks that are involved when working with a message category file.

Related reference:

"Rules for naming workspace objects" on page 6827

Data Design projects and files

A Data Design project contains resources relating to databases.

Data Design projects can contain the following resource files:

.dbm files
: A database definition file contains information about a connection to a database. The New Database Definition File wizard automatically creates this file. The name of the database definition file is the name of the database that the file connects to.

  Default editor: Physical Data Model editor

You can define more than one database definition in the same Data Design project, but each must be for a separate database.

Related concepts:

"WebSphere Message Broker Toolkit" on page 31
The WebSphere Message Broker Toolkit is an integrated development environment and graphical user interface based on the Eclipse platform.

"Resources" on page 36
The projects, folders, and files that you work with in the WebSphere Message Broker Toolkit workspace are called *resources*. By default, these resources are stored with their metadata in the workspace directory in your local file system. The workspace directory is created the first time that you start the WebSphere Message Broker Toolkit.

Related tasks:

"Adding database definitions to the WebSphere Message Broker Toolkit" on page 2278
Use the New Database Definition File wizard to add database definitions to the WebSphere Message Broker Toolkit.

Plug-in Development projects and files

A plug-in project can contain any WebSphere Message Broker Toolkit-developed plug-in resource. In WebSphere Message Broker, it is used for user-defined message nodes.

User-defined Message Nodes:
Plug-in development projects used to create Message Nodes contain these resources:

.msgnode file
: Contains the definition of the node.

palette.xmi file
: Lists the user-defined nodes in the project and the Message Editor palette group in which they are shown.

.html file
: Contains the help files for the node.

.java file
Java source code for Property Editors and Compilers.

plugin.xml file
The plug-in manifest file that adds the node to the WebSphere Message Broker installation

.properties file
National Language properties - contains the translations for Properties and Terminals

.gif file
Icon - contains the image to associate with the node

Related concepts:

"WebSphere Message Broker Toolkit" on page 31
The WebSphere Message Broker Toolkit is an integrated development environment and graphical user interface based on the Eclipse platform.

"Resources" on page 36
The projects, folders, and files that you work with in the WebSphere Message Broker Toolkit workspace are called *resources*. By default, these resources are stored with their metadata in the workspace directory in your local file system. The workspace directory is created the first time that you start the WebSphere Message Broker Toolkit.

Related tasks:

"Creating a user-defined node project" on page 3080
When you create user-defined nodes, you must first create a user-defined node project to contain the nodes and their supporting files.

"Implementing a user-defined parser" on page 3099
Create a user-defined parser to interpret messages with a different format and structure.

Related reference:

"Rules for naming workspace objects" on page 6827

Java projects and files

Developing Java projects using a .java file

You develop Java projects using the Java perspective WebSphere Message Broker Toolkit.

You can add any valid Java code to a JavaCompute node, making full use of the Java user-defined node API to process an incoming message. You can use the Java editing facilities of the Eclipse platform to develop your Java code.

If you have selected a user-defined message node, you have access to various resources that include the .java file; this file includes Java source code for Property Editors and Compilers.

You can also call Java methods directly from the Mapping node.

Related tasks:

"Managing Java Files" on page 2629
The Java code that you provide to modify or customize the behavior of a JavaCompute node is stored in a Java project. WebSphere Message Broker uses the Eclipse Java perspective for developing and administering Java files.

"Calling a Java method" on page 2310
To call an existing Java method from a mapping node, select the method from the

Call Existing Java Method wizard, or enter an XPath expression in the Edit pane.

Related reference:

"JavaCompute node" on page 4514
Use the JavaCompute node to work with messages by using the Java language.

Pattern instance projects

A pattern instance project contains project references to all the other projects in the workspace that relate to a specific pattern instance.

Pattern instance projects are shown in the Broker Development view by default when a pattern instance is generated. You can view all the generated resources that relate to a pattern instance by selecting the pattern instance from the working set selection list in the Broker Development view.

Related concepts:

"WebSphere Message Broker Toolkit" on page 31
The WebSphere Message Broker Toolkit is an integrated development environment and graphical user interface based on the Eclipse platform.

"Resources" on page 36
The projects, folders, and files that you work with in the WebSphere Message Broker Toolkit workspace are called *resources*. By default, these resources are stored with their metadata in the workspace directory in your local file system. The workspace directory is created the first time that you start the WebSphere Message Broker Toolkit.

"Using patterns" on page 1312
Use patterns that are supplied with the WebSphere Message Broker Toolkit to create resources that are used to solve a specific business problem.

Related tasks:

"Working with patterns in the Broker Development view" on page 1314
Using the Broker Development view to create patterns.

Rules for naming workspace objects

You can uniquely identify and locate a workspace resource using a namespace and a name. An organization uses its own name to form a namespace, and all resource names within that namespace are safe from clashes with those of another organization. The combination of a user specified namespace and name is often called the "fully qualified name". The unnamed namespace (name space="") is supported.

Note, that you must not include an exclamation mark (!) in the workspace name.

WebSphere Message Broker defines an application symbol space using project references. No fully qualified names can be duplicated across the application symbol space. A Tasks List error is displayed if duplicate or ambiguous names are detected within an application symbol space.

Related concepts:

"WebSphere Message Broker Toolkit" on page 31
The WebSphere Message Broker Toolkit is an integrated development environment and graphical user interface based on the Eclipse platform.

"Resources" on page 36
The projects, folders, and files that you work with in the WebSphere Message Broker Toolkit workspace are called *resources*. By default, these resources are stored with their metadata in the workspace directory in your local file system. The workspace directory is created the first time that you start the WebSphere Message

Broker Toolkit.

"Project references" on page 44
When a projects refers to other projects in the workspace, this is called a *project reference*. When one project references another, the files in the referenced project are available for use by the referring project. For example to configure certain nodes, to create libraries of reusable message flows, and to enable Content Assist in the ESQL editor.

Related reference:

"Resource types in the WebSphere Message Broker Toolkit" on page 6821
Links to information on projects and resource files.

WebSphere Message Broker Explorer and WebSphere Message Broker Toolkit keyboard shortcuts

You can navigate all interfaces in the WebSphere Message Broker Explorer and WebSphere Message Broker Toolkit by using the keyboard.

The following table shows the general shortcut keys that are provided in the WebSphere Message Broker Explorer and WebSphere Message Broker Toolkit.

Keyboard shortcut	Result
Left, right, up, and down arrow keys	Move in the direction shown on the key.
Tab	Move the focus forward through the workbench.
Shift+Tab	Move the focus backward through the workbench.
Esc	Cancel the action.
Enter	Commit the action.
Spacebar	Select the object that currently has the focus.
Shift+F10	Open a pop-up menu.
Ctrl+F6	Open the Editors window. (Continue to hold down Ctrl to keep the window open. Use the up and down arrow keys to select a file. Release Ctrl. Press Enter.)
Ctrl+F7	Move between panes.

When you use the keyboard to navigate the toolbar of the Broker Development view, use the Tab key to access the first icon on the toolbar, then use the arrow keys to access other icons. For example, use the Tab key to access the **Hide Categories** icon, then use the right arrow key to access the **Collapse All** icon.

Message Flow editor keyboard shortcuts

The following table shows keyboard shortcuts that are specific to the Message Flow editor:

Keyboard shortcut	Result when the Selection tool is active	Result when the Connection tool is active
Left, right, up, and down arrow keys	Select a node on the palette or on the canvas. Use with Shift, or Ctrl+Spacebar, to select multiple nodes.	Select a terminal on a node.

Keyboard shortcut	Result when the Selection tool is active	Result when the Connection tool is active
slash (/) and backslash (\)	Press slash (/) to cycle forwards, or press backslash (\) to cycle backwards through the connections of the selected node. Use with Shift, or Ctrl+Spacebar, to select multiple connections.	
decimal key (.) and Shift+decimal key (.)	Press the decimal key (.) to cycle forwards, or press Shift and the decimal key (.) to cycle backwards through the move handles, or the resize handles, of the currently selected object on the canvas. When a node is selected, use this shortcut to activate the move handles, then use the arrow keys to move the node. Press Enter to confirm the move. When a connection is selected, use this shortcut to activate its resize handles, then use the arrow keys to create bend points in the connection. Press Enter to confirm the move.	
Esc		Typically, pressing Esc cancels the previous action. When there is no previous action to cancel, and the Connection tool is selected, pressing Esc activates the Selection tool.

The following examples show how to use the Message Flow editor keyboard shortcuts.

Adding nodes from the palette to the canvas:

1. If the palette is already open, go straight to step 2. If the palette is not already open, follow these steps to open the palette:
   a. Open the Editors window.
   b. Select the file that you want to use.
   c. Press Tab until the focus moves to the palette arrow.
   d. Press Enter, or Spacebar, to give focus to the palette.
2. Press Tab to give focus to the palette arrow.
3. Press Tab once to give focus to the drawers within the palette. (Note: There are no visual highlights to indicate the focus.)
4. Use the up and down arrow keys to highlight the drawer required. Press Enter.
5. Use the up and down arrow keys to highlight the node required.
6. Press Alt+L to open the palette menu, and select **Add Node to Canvas**. Press Enter. The node is now on the canvas.

Moving nodes on the canvas:

1. Press Tab to give focus to the palette arrow. (It does not matter whether the palette is open or closed.)
2. Press Tab two times. The focus is now on the canvas. (Note: There are no visual highlights to indicate the focus.)
3. Select a node on the canvas.

4. Press the decimal key (.) to select one of the move handles around the node.
5. Use the left, right, up, and down arrow keys to move the node to a new location.
6. Press Enter to confirm the move.

Connecting nodes:

1. Press Tab to give focus to the palette arrow. (It does not matter whether the palette is open or closed.)
2. Press Tab two times; the focus is now on the canvas. (Note: There are no visual highlights to indicate the focus.)
3. Select the node on the canvas from which you want to create a connection.
4. Press Shift+F10 to open a pop-up menu.
5. Use the arrow keys to select **Create Connection**. Press Enter. The Terminal Selection window opens.
6. Press Tab until the focus is in the **Select Terminal** field. Use the arrow keys to select a source (output) terminal.
7. Press Tab to select **OK**, press Enter to confirm your selection.
8. Use the arrow keys to select the input terminal of the target node. Press Enter to confirm the connection.

Selecting a node connection:

1. Press Tab to give focus to the palette arrow. (It does not matter whether the palette is open or closed.)
2. Press Tab two times. The focus is now on the canvas. (Note: There are no visual highlights to indicate the focus.)
3. Use the arrow keys to select a node that has a connection you want to select.
4. Use the slash (/) and backslash (\) keys to cycle through the connections.

Creating a bend point:

1. Select a node connection.
2. Press the decimal key (.) until the midpoint of the connection is selected.
3. Use the arrow keys to move the midpoint to a new location, and press Enter to confirm the move.

Reconnecting an existing connection to a new terminal:

1. Select a node connection.
2. Press the decimal key (.) until the end of the connection that you want to modify is selected.
3. Use the arrow keys to select a new source (input) terminal, and press Enter to confirm your selection.

Related concepts:

"Message flow node palette" on page 1027
The palette in the Message Flow editor contains all the built-in nodes, which are organized into categories, or drawers. A drawer is a container for a list of icons, such as the **Favorites** drawer.

Related reference:

"Message Flow editor" on page 6810
The Message Flow editor is the default editor provided by the Broker Application Development perspective for defining a graphical representation of a message flow in the WebSphere Message Broker Toolkit, and for setting properties for individual message flow nodes.

"Accessibility features for WebSphere Message Broker" on page 135
Accessibility features help users who have a physical disability, such as restricted mobility or limited vision, to use information technology products successfully.

Workbench User Guide - Keyboard shortcuts for the Workbench, Java development tools, and the debugger

What's new if you are migrating from Version 6.0

Find out about new features in WebSphere Message Broker if you are migrating from Version 6.0.

See the following topics for high level information about new features in Version 6.1:

- "Web services"
- "Security" on page 6832
- "WebSphere Adapter nodes" on page 6833
- "File nodes" on page 6834
- "Other new nodes" on page 6835
- "Usability" on page 6836
- "Performance" on page 6837

Web services

Find out about the significant enhancements to Web services support including integrated support for WebSphere Service Registry and Repository, Web services security, and SOAP nodes.

SOAP nodes

- Full support for Web Services Provider and Consumer scenarios with 5 SOAP nodes:
  - Provider nodes:
    - "SOAPInput node" on page 4795
    - "SOAPReply node" on page 4819
  - Consumer nodes:
    - "SOAPRequest node" on page 4828
    - "SOAPAsyncRequest node" on page 4750
    - "SOAPAsyncResponse node" on page 4777
  - These nodes can be combined to provide a Web service intermediary.
- Processing of SOAP payload and headers are simplified by the following SOAP nodes:
  - "SOAPEnvelope node" on page 4786
  - "SOAPExtract node" on page 4790

Single SOAP domain for all Web Services message formats

- A SOAP parser and SOAP tree format.
- Supported message formats are SOAP 1.1, SOAP 1.2, SOAP with Attachments (SwA), and MTOM
- HTTP and HTTPS transports are supported.

Improved WSDL Integration

- Simplified import and generation of WSDL.
- Improved WSDL editor.
- WSDL is now an integral part of a message set.
- WSDL from a message set can be used directly in the creation and configuration of message flows, including support for drag-and-drop operations.

Web services extensions

- Support for WS-Addressing Endpoint References and Message addressing properties.
- Support for WS-Security authentication, encryption, and signing:
  - Authentication using username and password.
  - Authentication using X509 certificates.
  - Comprehensive encryption and signing algorithms.
  - Configuration using Policy Sets.

WebSphere Service Registry and Repository

- General concepts:
  - WebSphere Service Registry and Repository contains a variety of entities such as WSDL and XSD, together with their relationships and associated user properties.
  - Governance can be achieved by using the WebSphere Service Registry and Repository to determine Message Broker processing.
- Runtime interactions:
  - Message flows can use, access, and select specific WebSphere Service Registry and Repository entities.
  - Nodes to support retrieval of entities and services:
    - "EndpointLookup node" on page 4407
    - "RegistryLookup node" on page 4646

Related tasks:

"Processing Web service messages" on page 1601
Use WebSphere Message Broker nodes and services to connect to other Web services providers and consumers.

Security

WebSphere Message Broker provides enterprise-wide identity, authentication and authorization with Tivoli and LDAP. Click on the links to get more information about security in WebSphere Message Broker.

Security model

- WebSphere Message Broker now has a powerful runtime security model:
  - Supporting cross domains security processing.
  - Identity, Authentication and Authorization are native capabilities.
- Support for major policy decision points technologies including Tivoli and LDAP.

Identity

- A rich identity context can be defined including properties for source and mapped identities:
  - Type
  - Token
  - Password
  - Issued by
- MQ, HTTP, JMS, and SOAP transports can all provide identity information.
- Identity attributes can be set on input and output nodes.
- Identity information is also present in the Message Tree.

Security profiles

- You can configure different security profiles to differentiate between the security requirements for your different message flows and nodes.
- The security profile contains information on:
  - Authentication type and configuration.
  - Mapping type and configuration.
  - Authorization type and configuration.

Related concepts:

"Security overview" on page 351
When you are designing a WebSphere Message Broker application it is important to consider the security measures that are needed to protect the information in the system.

"Message flow security overview" on page 383
WebSphere Message Broker provides a security manager, which enables you to control access to individual messages in a message flow, using the identity of the message.

WebSphere Adapter nodes

You can connect to Enterprise Information Systems using the integrated WebSphere Adapter nodes. Highlights of the WebSphere Adapter nodes are provided in this topic. Click on the links to get more information about the WebSphere Adapter nodes.

Support for Enterprise Information Systems

- Inbound and outbound support is provided for the following Enterprise Information Systems (EIS):
  - SAP
  - Siebel
  - PeopleSoft
- Enterprise Metadata Discovery (EMD) is used for key data structure discovery and accelerated message set generation.
- The use of the native message broker tree provides high performance access to the EIS using WebSphere Adapters.
- TwineBall input and request nodes provided as sample nodes with their own EIS for learning about how the WebSphere Adapter nodes work.

SAP Adapter nodes

- Integrated SAP Adapter nodes enable you to interact with SAP applications:
  - "SAPInput node" on page 4676
  - "SAPRequest node" on page 4685

Siebel Adapter nodes

- Integrated Siebel Adapter nodes enable you to interact with Siebel applications:
  - "SiebelInput node" on page 4740
  - "SiebelRequest node" on page 4745

PeopleSoft Adapter nodes

- Integrated PeopleSoft Adapter nodes enable you to interact with PeopleSoft applications:
  - "PeopleSoftInput node" on page 4630
  - "PeopleSoftRequest node" on page 4635

TwineBall Adapter nodes

- Educational TwineBall Adapter nodes included to help you learn how the WebSphere Adapter nodes work:
  - "TwineballInput node" on page 4951
  - "TwineballRequest node" on page 4955

Related concepts:

"WebSphere Adapters nodes" on page 1914
A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

Related reference:

"WebSphere Adapters properties" on page 4024
Reference information about the properties that you set for WebSphere Adapters nodes.

File nodes

You can access, process, and produce very large files with the file input and output nodes.

Built-in support for File processing

- Two nodes support inbound and outbound file processing:
  - "FileInput node" on page 4415
  - "FileOutput node" on page 4430

Large file handling

- Very large files can be processed without using excessive storage.

Record detection

- Comprehensive support for record detection:
  - Simple record detection: LF, CRLF, Fixed Length, and user-defined.
  - Parser: Use an existing message definition to identify record boundaries.
- Advanced record detection techniques enable simplified processing of large, repeating content files.

FTP support

- Configure FTP server properties on the FileInput and FileOutput nodes

Other new nodes

Find out about other new nodes in Version 6.1.

For information about new nodes in Version 7.0, see "What's new in Version 7.0?" on page 7.

Collector node

- Use the Collector node for complex condition processing.
- Coordinate messages from one or more sources.
- Wait for multiple input conditions and process when all satisfied.
- Messages matching the configured conditions are grouped into message collections for processing.
- Configurable conditions include:
  - Number of messages received.
  - Messages received in a set period of time.
  - Messages that match the contents of a correlation path.
  - Contents of messages that match a correlation pattern.

DatabaseRetrieve node

- Use the DatabaseRetrieve node to ensure that information in a message is up to date.
- Look up information from a database row using a message key, for example an account number, and store it in a message.
- Configure the value for the message key using XPath or ESQL syntax.

DatabaseRoute node

- Use the DatabaseRoute node to pass through or route messages using information from a database.
- Look up information from a database row using a message key, for example an account number, and store it in a message.
- Route the message for further processing based on these XPath values.
- Messages can be routed to multiple destinations if they are matched.

EmailOutput node

- Use the EmailOutput node to send an email message to one or more recipients from a message flow.
- This node builds and sends the email message to an SMTP server that you specify.
- Several options are available to generate the email message:
  - Configure an email with a statically-defined subject and text to a statically-defined list of recipients.
  - Configure a statically-defined email as a MIME message, with an attachment derived from the message tree.
  - Create a dynamic email message where the SMTP server, list of recipients, subject, text, and multiple attachments are determined at run time.
  - Pass a MIME message to the EmailOutput node by using the MIME parser to write the MIME message to a bit stream.

Route node	• Use the Route node to direct messages that meet certain criteria down different paths of a message flow. • XPath filter expressions are used to control processing. • Add extra output terminals to the node to route matching expressions down a selected processing path.

Related reference:

"Collector node" on page 4333
Use the Collector node to create message collections based on rules that you configure in the node.

"DatabaseRetrieve node" on page 4363
Use the DatabaseRetrieve node to ensure that information in a message is up to date.

"DatabaseRoute node" on page 4373
Use the DatabaseRoute node to route messages using information from a database in conjunction with XPath expressions.

"EmailOutput node" on page 4400
Use the EmailOutput node to send email messages to one or more recipients.

"Route node" on page 4669
Use the Route node to direct messages that meet certain criteria down different paths of a message flow.

Usability

Find out about usability enhancements to help make developing applications and testing applications easier and faster. Click on the links to get more information about usability enhancements in WebSphere Message Broker.

Message Broker Toolkit improvements	• Quick Start wizards. • Simplified and improved application development wizards and editors. • Enhanced graphical mapping. • Simplified menus. • Display projects in working sets. • Categories for nodes on the node palette. • Enhancements to message flow debugging. • Enhancements to message broker archive editor. • Drag WSDL on to the message flow canvas to speed up development.
Improved test support in the Message Broker Toolkit	• Create a test configuration to rapidly test message flows. • Template test messages are generated if you are using a message set. • Monitor test execution events. • Test configurations can be saved for reuse and analysis. • Review and save test execution traces.

Samples

- Additional samples demonstrating functionality.
- Categories for Technology samples:
  - File processing
  - Message formats
  - Control and routing
  - Message transformation
  - Security
  - Transports and connectivity
  - Web services

You can view information about samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit or the online information center. You can run samples only when you use the information center that is integrated with the WebSphere Message Broker Toolkit.

Help view

- See context-sensitive help and related topics links by using the Help view, even when you are using a wizard or dialog box.
- Search the documentation and access fully indexed help topics by using the Help view.

Related concepts:

"Quick Start wizards overview" on page 1409
You can use a Quick Start wizard to set up the basic resources that are required to develop a message flow application. The wizard sets up and gives names to containers for the resources that you need when you develop a message flow.

"Testing message flows by using the Test Client" on page 3144
You can test message flows in a safe environment before they are used on a production system by using the Test Client.

"Samples" on page 98
The WebSphere Message Broker Toolkit provides samples that show the features that are available in WebSphere Message Broker, and how to use them. This topic provides links to the information about the individual samples.

Related tasks:

"Creating a working set" on page 575
Create a working set to limit the number of resources that are displayed in the Broker Development view.

"Creating an application by using the Configure New Web Service Usage wizard" on page 1417
Use these instructions to generate a message flow by using the Configure New Web Service Usage wizard.

Related reference:

"Broker Archive editor" on page 6794
Use the Broker Archive editor to create and manage broker archive (BAR) files.

Performance

Find out about continued performance improvements in WebSphere Message Broker.

Parser Performance	• Significant XML message performance improvement. • XML validation performance improvement. • Opaque parsing for named XML elements to further improve performance. • Industry and existing message formats benefit from binary and string parser improvements. – Supporting cross domains security processing. – Identity, Authentication and Authorization are native capabilities. • Support for major policy decision points technologies including Tivoli and LDAP.
XSLT Performance	• XSLT engine now directly accesses message tree leading to major performance gains.
Storage Performance	• Compacted runtime storage leads to a significant reduction in the runtime storage usage.

WebSphere Message Broker Explorer views

The WebSphere Message Broker Explorer can be used to manage and administer your brokers and deployed resources.

The WebSphere Message Broker Explorer is an extension of the WebSphere MQ Explorer.

Typically, you carry out the following tasks in the WebSphere Message Broker Explorer:

- Setting up your brokers, including:
  - Creating brokers
  - Connecting to brokers
  - Removing brokers
- Adding and removing execution groups
- Editing and deploying broker archive (BAR) files
- Viewing the results of deployment and other broker activity in the Administration log
- Viewing and editing broker properties
- Viewing and editing security profiles and policy sets
- Viewing and editing configurable properties
- Viewing message flow accounting and statistics data

The WebSphere Message Broker Explorer provides several views that you can use to browse and update your broker resources.

If you cannot see the Brokers folder and Broker Archive Files folder in your MQ Explorer session, you have not installed the broker-specific plug-ins that are provided by WebSphere Message Broker Explorer. Close your WebSphere MQ Explorer session, follow the instructions to install the WebSphere Message Broker Explorer, then start the WebSphere MQ Explorer again.

Navigator view

The Navigator view shows folders that contain all the WebSphere MQ and broker resources that are currently available for you to administer from the WebSphere Message Broker Explorer. Right-click the WebSphere MQ folder to configure settings that affect the whole of WebSphere MQ on your system, and to perform other administration tasks.

The icons shown in the Navigator view indicate the status of the WebSphere MQ and broker objects.

Right-click the Brokers folder to create brokers and view existing broker resources. Right-click brokers, execution groups, and other broker objects to perform configuration and administration tasks.

Right-click the Broker Archive Files folder to view and deploy broker archive files to your brokers.

When you click a WebSphere MQ or broker object in the Navigator view, properties associated with the object are shown in the Content view.

Content view

The Content view shows properties associated with a selected object in the Navigator view. The properties that are shown depend on the associated object that has been selected. With some objects, you can double-click properties shown in the Content view to see more information, and to edit the properties.

Broker Statistics and Broker Statistics Graph views

Use the Broker Statistics and Broker Statistics Graph views in the WebSphere Message Broker Explorer to view snapshot accounting and statistics data as the broker produces it. For more information about viewing broker accounting and statistics data, see “Viewing message flow accounting and statistics data” on page 3300.

Policy Sets and Policy Set Bindings editor

Use the Policy Sets and Policy Set Bindings editor for editing, saving, importing, and exporting a policy set or binding. You can define all the WS-Security policies for a single node, or a set of nodes. The policy set can be associated by the administrator with either a message flow or a node. To access the Policy Sets and Policy Set Bindings editor, right-click the broker and click **Properties**. Select the **Security** tab and click **Policy Sets**. For more information about the Policy Sets and Policy Set Bindings editor, see “Policy Sets and Policy Set Bindings editor” on page 6841.

Security Profiles editor

Use the Security Profiles editor to specify whether identity propagation, authentication, authorization, and mapping are performed on the identity of messages in the message flow, and if so, which external security provider is used. For more information about Security Profiles, see “Security profiles” on page 387.

DataPower Security wizard

Use the DataPower Security wizard in the WebSphere Message Broker Explorer to configure an external DataPower appliance to handle the WS-Security Policy for your HTTP, HTTPS input, and SOAP nodes in your message flow. For more information about the DataPower Security wizard, see “Configuring DataPower security settings” on page 639.

Administration Log view

The Administration Log view shows administration log information for the broker selected in the Navigator view. The messages in the Administration log show the results from deployment or other actions you have performed on the broker. Double-click individual messages to read the full content of the messages in the Administration Log view.

You can save the messages from the Administration log, or permanently remove the administration log information messages.

For more information about the Administration Log view, see "Administration Log view."

Related concepts:

"WebSphere Message Broker Toolkit" on page 31
The WebSphere Message Broker Toolkit is an integrated development environment and graphical user interface based on the Eclipse platform.

"WebSphere Message Broker Explorer" on page 57
The WebSphere Message Broker Explorer is a graphical user interface based on the Eclipse platform for administering your brokers.

Related tasks:

"Changing WebSphere Message Broker Explorer preferences" on page 654
Change preferences in the WebSphere Message Broker Explorer.

Chapter 8, "Administering brokers and broker resources," on page 899
Administering brokers and associated broker resources includes the tasks that you perform frequently to activate and manage those resources. Choose the method you prefer to administer your brokers and associated resources.

Chapter 11, "Packaging and deploying," on page 3209
Package resources that you create in the WebSphere Message Broker Toolkit, such as message flows, and deploy them to execution groups on brokers.

Related reference:

"Administration Log view"
The Administration Log view shows administration requests and their results, changes made to objects, and the result of deployment actions on brokers. The Administration Log view can also describe configuration changes that have been automatically applied as a result of moving the broker from one version to another; for example, after applying maintenance.

"Broker Application Development perspective" on page 6784
The Broker Application Development perspective is the default perspective that is displayed when you start the WebSphere Message Broker Toolkit.

Administration Log view

The Administration Log view shows administration requests and their results, changes made to objects, and the result of deployment actions on brokers. The Administration Log view can also describe configuration changes that have been automatically applied as a result of moving the broker from one version to another; for example, after applying maintenance.

The Administration Log view is displayed at the bottom of the WebSphere Message Broker Explorer, and at the bottom of the Broker Application Development perspective in the WebSphere Message Broker Toolkit. If the Administration Log view is not displayed, you can show it by clicking **Window** > **Show View** > **Administration Log**.

To view messages from a broker, the broker must be started, and you must be connected to it. Click the broker in the Navigator view, to view deployment and other messages in the Administration log.

Right-clicking in the Administration Log view displays the following menu items:

- **Clear Administration Log**

  Clears all log entries from the Administration Log view and deletes the log entries from the broker. You cannot retrieve a message that has been cleared.

  Messages are automatically cleared when the broker starts.
- **Save Log As**

  Saves the messages to a filename and path of your choice. **Save Log As** does not remove messages from the Administration Log view or the broker.

Columns in the Administration Log view

Message

The Message column displays the message number.

Source

The Source column displays the reason the message was produced. Examples of sources are: Administration Request, Administration Result, and Change Notification.

Timestamp

The Timestamp column displays the date and time for each of the deployment messages from the broker.

Message detail

The Message detail column displays the detail of the deployment message. To view the full message, double-click the row containing the message. The message is displayed in a new window.

Related tasks:

"Viewing Administration log information" on page 1007
You can view Administration log information by using either the WebSphere Message Broker Explorer, or the CMP API.

"Saving Administration log information" on page 1008
Save the Administration log information that is written to the Administration Log view in the WebSphere Message Broker Explorer.

"Clearing Administration log information" on page 1009
Clear Administration log information to reduce the size of the log by using either the WebSphere Message Broker Explorer or the CMP API.

Policy Sets and Policy Set Bindings editor

The Policy Sets and Policy Set Bindings editor is the default editor for editing, saving, importing, and exporting a policy set or binding. You can define all the WS-Security policies for a single node, or a set of nodes. The policy set can be associated by the administrator with either a message flow or a node.

Launching the editor

You must connect the WebSphere Message Broker Explorer to a broker before you can edit policy sets and bindings.

To launch the Policy Sets and Policy Set Bindings editor, take the following steps:

1. Open the WebSphere Message Broker Explorer.
2. Right-click the broker with which you want to work, and click **Properties**.
3. Select the Security tab, and click **Policy Sets**.

The Policy Sets and Policy Set Bindings editor includes the following panels:

"Policy Sets and Policy Set Bindings editor: main panel" on page 6852

"Policy Sets and Policy Set Bindings editor: Authentication tokens panel" on page 6843

"Policy Sets and Policy Set Bindings editor: Message Level Protection panel" on page 6845

"Policy Sets and Policy Set Bindings editor: Message Part Protection panel" on page 6849

Policy set bindings

The administrator associates a policy set binding with a policy set when the policy set binding is added. Information about the binding is then displayed on the main panel of the policy set binding. You cannot change the policy set to which a binding is associated. You can delete the policy set binding independently of the policy set. However, deleting a policy set also deletes the associated policy set binding.

The Policy Set Bindings editor includes the following panels:

"Policy Sets and Policy Set Bindings editor: Authentication and Protection Tokens panel" on page 6854

"Policy Sets and Policy Set Bindings editor: Key Information panel" on page 6858

"Policy Sets and Policy Set Bindings editor: Kerberos settings panel" on page 6860

"Policy Sets and Policy Set Bindings editor: Message Part Policies panel" on page 6856

"Policy Sets and Policy Set Bindings editor: Message Expiration panel" on page 6862

"Policy Sets and Policy Set Bindings editor: Advanced panel" on page 6863

Related concepts:

"Policy sets" on page 774
Policy sets and bindings define and configure your WS-Security requirements, supported by WebSphere Message Broker, for the SOAPInput, SOAPReply, SOAPRequest, SOAPAsyncRequest, and SOAPAsyncResponse nodes.

Related tasks:

"Viewing and setting keystore and truststore runtime properties at broker level" on page 780
Configure the message broker to refer to a keystore, a truststore, or both, before deploying any message flows that require policy set or bindings for signature, encryption, or X.509 Authentication.

"Implementing WS-Security" on page 769
Configure authentication, XML encryption, XML signature, and message expiration by using the Policy Sets and Policy Set Bindings editor.

"Associating policy sets and bindings with message flows and nodes" on page 785
Use the Broker Archive editor to associate policy sets and bindings with message flows and nodes, so that they are available to the broker at run time.

Policy Sets and Policy Set Bindings editor: Authentication tokens panel

Use this panel, which is in the Policy Sets section of the editor, to create Username, X.509, SAML, and LTPA authentication tokens.

You can further configure X.509 tokens in the bindings panels, at which time they can be associated to key information.

Request and Response is always the same in any server/client relationship. A message from a client to the server is a request, and the message back from the server to the client is the response.

Fields

Table 292. Username authentication tokens

Field name	Description and valid options
Token name	Enter a user-defined name for the token policy. When you add a new row, this field defaults to `default_request`. You can select and change this value.
SOAP Message	One of the following values: • `Request`: a message from the client to the server. • `Response`: a message from the server to the client. When you add a new row, this field defaults to `Request`. You can select and change this value.
WS-Security Version	One of the following values: • `1.0` • `1.1` When you add a new row, this field defaults to `1.0`. You can select and change this value.

Table 293. X.509 authentication tokens

Field name	Description and valid options
Token name	Enter a user-defined name for the token policy. When you add a new row, this field defaults to `default_request`. You can select and change this value.
SOAP Message	One of the following values: • `Request`: a message from the client to the server. • `Response`: a message from the server to the client. When you add a new row, this field defaults to `Request`. You can select and change this value.

Table 293. X.509 authentication tokens (continued)

Field name	Description and valid options
WS-Security Version	One of the following values: • `1.0` • `1.1` When you add a new row, this field defaults to `1.0`. You can select and change this value.
Token Type	One of the following values: • X.509 Version 3 • X.509 PKCS7 • X.509 PKI Path Version 1 When you add a new row, this field defaults to `X.509 Version 3`. You can select and change this value.

Table 294. Other authentication tokens

Field name	Description and valid options
Token name	Enter a user-defined name for the token policy. When you add a new row, this field defaults to `default_request`. You can select and change this value.
Token Type	One of the following values: • SAML V1.1 pass-through • SAML V2.0 pass-through • LTPA V2 pass-through

Related reference:

"Policy Sets and Policy Set Bindings editor" on page 6841
The Policy Sets and Policy Set Bindings editor is the default editor for editing, saving, importing, and exporting a policy set or binding. You can define all the WS-Security policies for a single node, or a set of nodes. The policy set can be associated by the administrator with either a message flow or a node.

"Policy Sets and Policy Set Bindings editor: Message Level Protection panel" on page 6845
Use this panel, which is in the Policy Sets section of the editor, to apply signatures and encryption to the whole message, whether inbound or outbound.

"Policy Sets and Policy Set Bindings editor: Message Part Protection panel" on page 6849
Use this panel, which is in the Policy Sets section of the editor, to define the parts of a message that encryption and signature apply to. Encrypted parts are used to protect message confidentiality. Signature parts are used for message integrity.

"Policy Sets and Policy Set Bindings editor: main panel" on page 6852
Use this panel, which is in the Policy Sets section of the editor, to associate a policy set binding with a policy set, and to specify whether the binding is a consumer or provider binding.

"Policy Sets and Policy Set Bindings editor: Authentication and Protection Tokens panel" on page 6854
Use this panel, which is in the Policy Set Bindings section of the editor, to further configure any X.509 authentication tokens defined in the associated policy set.

"Policy Sets and Policy Set Bindings editor: Message Part Policies panel" on page 6856
Use this panel, which is in the Policy Set Bindings section of the editor, to further

configure any message part protection tokens defined in the associated policy set.

"Policy Sets and Policy Set Bindings editor: Key Information panel" on page 6858
Use this panel, which is in the Policy Set Bindings section of the editor, to further configure any message level protection tokens that are defined in the associated policy set.

"Policy Sets and Policy Set Bindings editor: Kerberos settings panel" on page 6860
Use this panel, which is in the Policy Set Bindings section of the editor, to further configure any Kerberos tokens that are defined in the associated policy set.

"Policy Sets and Policy Set Bindings editor: Message Expiration panel" on page 6862
Use this panel, which is in the Policy Set Bindings section of the editor, to define settings for message expiration. When message expiration is enabled, the message expires after the specified time interval.

"Policy Sets and Policy Set Bindings editor: Advanced panel" on page 6863
Use this panel, which is in the Policy Set Bindings section of the editor, to configure the **mustUnderstand** attribute in the security header of the consumer message. By default this check box is selected. If you unselect the check box, you can omit the **mustUnderstand=1** attribute from the security header. This attribute applies to consumer binding only.

Policy Sets and Policy Set Bindings editor: Message Level Protection panel

Use this panel, which is in the Policy Sets section of the editor, to apply signatures and encryption to the whole message, whether inbound or outbound.

Field	Description and valid options
Message level protection	Select this check box to specify that message level protection (using digital signatures or encryption) is required. If this check box is selected the other fields on this panel are available, and you can use the associated panels to define signature and encryption policies. This field is cleared by default.
Require signature confirmation	Select this check box to require signature confirmation.
Include timestamp in security header	Select this check box to include a time stamp in the header. You can specify where the time stamp is placed in the header by using **Security header layout**.
Security header layout	Specify rules for the layout of the security header: **Strict - declarations must precede use** The declarations in the header must precede the use. This is the default value. **Lax - order of contents can vary** The order of contents in the header can vary. **Lax but timestamp required first in header** The timestamp must be first in the header but the order of the remaining elements can vary. **Lax but timestamp required last in header** The timestamp must be last in the header but the order of the remaining elements can vary.

Tokens

Use this panel to define symmetric and asymmetric tokens to be used for signature and encryption.

Associate the asymmetric tokens that you define here with parts of the message that require signature and encryption. The tokens are also associated with private keys or X.509 Public Key Certificates (PKCs), which are part of public/private key pairs. Define one token for each distinct private key and PKC. The administrator can create any number of asymmetric X.509 tokens.

Associate the symmetric tokens that you define here with parts of the message that require signature and encryption. Only symmetric Kerberos tokens are supported. The administrator can create any number of Kerberos tokens.

You can edit this panel only if the **Message level protection** check box is selected on the Message Level Protection panel.

Field Name	Description and valid options
Token Name	Enter a name for the token.
Token Type	Either: **Initiator** The initiator of the request and response conversation, who owns the public/private pair of keys for which this token refers. **Recipient** The recipient of the request and response conversation, who owns the public/private pair of keys for which this token refers. When you add a new row, this field defaults to `Initiator`. You can change this value.

WS-Security Version	Either:
	`1.0`
	`1.1`
	When you add a new row, this field defaults to `1.0`. You can change this value.

Token Type (Asymmetric)	Any of: X.509 Version 3 X.509 PKCS7 X.509 PKI Path Version 1 When you add a new row, this field defaults to `X.509 Version 3`. You can change this value.
Token Type (Symmetric)	Any of: GSS_Kerberos5_AP_REQ Kerberos5_AP_REQ When you add a new row, this field defaults to `GSS_Kerberos5_AP_REQ`. You can change this value.

Algorithms

Use this panel to set the supported cryptographic and canonicalization algorithms. Algorithms are used to reconcile XML differences.

You can edit this panel only if **Message level protection** is selected on the Message Level Protection panel.

Field Name	Description and valid options
Algorithm suite	Select the algorithm that is required for performing cryptographic operations with symmetric or asymmetric key-based security tokens. All of the algorithm values in this field specify an algorithm suite. Algorithm suites and the values they each represent are detailed in the Web Services Security Policy Language (WS-SecurityPolicy) July 2005 Version 1.1 specification. The default algorithm is `Basic128Rsa15`. Basic256 Basic192 Basic128 TripleDes Basic256Rsa15 Basic192Rsa15 Basic128Rsa15 TripleDesRsa15 Basic256Sha256 Basic192Sha256 Basic128Sha256 TripleDesSha256 Basic256Sha256Rsa15 Basic192Sha256Rsa15 Basic128Sha256Rsa15 TripleDesSha256Rsa15
Canonicalization algorithm	Select the type of canonicalization. The following supported canonicalization algorithms are available in this list: Exclusive canonicalization Inclusive canonicalization The default value is `Exclusive canonicalization`.
Use security token reference transformation	Select this check box to specify that the security token reference is transformed. The default state is cleared.

This table defines values for the components of each algorithm suite.

Algorithm suite	Digest	Encryption	Symmetric Key Wrap	Asymmetric Key Wrap	Encryption key derivation	Signature key derivation	Minimum symmetric key length
Basic256	Sha1	Aes256	KwAes256	KwRsaOaep	PSha1L256	PSha1L192	256
Basic192	Sha1	Aes192	KwAes192	KwRsaOaep	PSha1L192	PSha1L192	192
Basic128	Sha1	Aes128	KwAes128	KwRsaOaep	PSha1L128	PSha1L128	128
TripleDes	Sha1	TripleDes	KwTripleDes	KwRsaOaep	PSha1L192	PSha1L192	192
Basic256Rsa15	Sha1	Aes256	KwAes256	KwRsa15	PSha1L256	PSha1L192	256
Basic192Rsa15	Sha1	Aes192	KwAes192	KwRsa15	PSha1L192	PSha1L192	192
Basic128Rsa15	Sha1	Aes128	KwAes128	KwRsa15	PSha1L128	PSha1L128	128
TripleDesRsa15	Sha1	TripleDes	KwTripleDes	KwRsa15	PSha1L192	PSha1L192	192
Basic256Sha256	Sha256	Aes256	KwAes256	KwRsaOaep	PSha1L256	PSha1L192	256
Basic192Sha256	Sha256	Aes192	KwAes192	KwRsaOaep	PSha1L192	PSha1L192	192
Basic128Sha256	Sha256	Aes128	KwAes128	KwRsaOaep	PSha1L128	PSha1L128	128

Algorithm suite	Digest	Encryption	Symmetric Key Wrap	Asymmetric Key Wrap	Encryption key derivation	Signature key derivation	Minimum symmetric key length
TripleDesSha256	Sha256	TripleDes	KwTripleDes	KwRsaOaep	PSha1L192	PSha1L192	192
Basic256Sha256Rsa15	Sha256	Aes256	KwAes256	KwRsa15	PSha1L256	PSha1L192	256
Basic192Sha256Rsa15	Sha256	Aes192	KwAes192	KwRsa15	PSha1L192	PSha1L192	192
Basic128Sha256Rsa15	Sha256	Aes128	KwAes128	KwRsa15	PSha1L128	PSha1L128	128
TripleDesSha256Rsa15	Sha256	TripleDes	KwTripleDes	KwRsa15	PSha1L192	PSha1L192	192

Related reference:

"Policy Sets and Policy Set Bindings editor" on page 6841
The Policy Sets and Policy Set Bindings editor is the default editor for editing, saving, importing, and exporting a policy set or binding. You can define all the WS-Security policies for a single node, or a set of nodes. The policy set can be associated by the administrator with either a message flow or a node.

"Policy Sets and Policy Set Bindings editor: Authentication tokens panel" on page 6843
Use this panel, which is in the Policy Sets section of the editor, to create Username, X.509, SAML, and LTPA authentication tokens.

"Policy Sets and Policy Set Bindings editor: Message Part Protection panel" on page 6849
Use this panel, which is in the Policy Sets section of the editor, to define the parts of a message that encryption and signature apply to. Encrypted parts are used to protect message confidentiality. Signature parts are used for message integrity.

"Policy Sets and Policy Set Bindings editor: main panel" on page 6852
Use this panel, which is in the Policy Sets section of the editor, to associate a policy set binding with a policy set, and to specify whether the binding is a consumer or provider binding.

"Policy Sets and Policy Set Bindings editor: Authentication and Protection Tokens panel" on page 6854
Use this panel, which is in the Policy Set Bindings section of the editor, to further configure any X.509 authentication tokens defined in the associated policy set.

"Policy Sets and Policy Set Bindings editor: Message Part Policies panel" on page 6856
Use this panel, which is in the Policy Set Bindings section of the editor, to further configure any message part protection tokens defined in the associated policy set.

"Policy Sets and Policy Set Bindings editor: Key Information panel" on page 6858
Use this panel, which is in the Policy Set Bindings section of the editor, to further configure any message level protection tokens that are defined in the associated policy set.

"Policy Sets and Policy Set Bindings editor: Kerberos settings panel" on page 6860
Use this panel, which is in the Policy Set Bindings section of the editor, to further configure any Kerberos tokens that are defined in the associated policy set.

"Policy Sets and Policy Set Bindings editor: Message Expiration panel" on page 6862
Use this panel, which is in the Policy Set Bindings section of the editor, to define settings for message expiration. When message expiration is enabled, the message expires after the specified time interval.

"Policy Sets and Policy Set Bindings editor: Advanced panel" on page 6863
Use this panel, which is in the Policy Set Bindings section of the editor, to configure the **mustUnderstand** attribute in the security header of the consumer message. By default this check box is selected. If you unselect the check box, you can omit the **mustUnderstand=1** attribute from the security header. This attribute

applies to consumer binding only.

Policy Sets and Policy Set Bindings editor: Message Part Protection panel

Use this panel, which is in the Policy Sets section of the editor, to define the parts of a message that encryption and signature apply to. Encrypted parts are used to protect message confidentiality. Signature parts are used for message integrity.

Create a row in this table for each part of the message that is to be encrypted or signed.

Field	Description and valid options
Name	Enter a user-defined name for the part. You can replicate the name to group several parts together; previously defined names are given as options.

Security type	Either:
	`Encryption`
	`Signature`

SOAP Message	Either: `Request`: a message from the client to the server. `Response`: a message from the server to the client.
Message Body	Determines that the whole message body is to be either encrypted or signed. If the whole message body is not to be encrypted or signed, further configuration is needed in one of the subsequent panels: Aliases, Qname, or Xpath.

Aliases

Use this panel to refer to an alias identified in a SOAPInput, SOAPRequest, or SOAPAsyncRequest node. The message flow with which this policy set will be associated in the Broker Archive editor must contain one of these nodes with an alias defined.

When developing a message flow containing one of these nodes, the developer might decide to identify a part of the message for which encryption or signature applies. This identification is done in the WS Extension properties panel of the node, by defining an XPath expression that refers to the part of the message and an associated alias name. The administrator then refers to that alias name by using this panel, and sets the correct security type for the alias on the corresponding Message Part Protection panel.

You can edit the Aliases panel only if **Message level protection** is selected on the Message Level Protection panel, and a part has been added in the Message Part Protection panel.

Field	Description and valid options
Name	Select a name from the list. All names created in the corresponding Message Part Protection panel are displayed.
Alias	Enter the alias value defined on the SOAP node property panel.

Qname

Use this panel to define namespaces, and optional elements within those namespaces, within the SOAP message header for which encryption or signature applies.

Namespaces are used primarily when WS-Addressing headers exist. If no local part name is specified to refer to specific elements, all elements in the SOAP message header for the specified namespace apply.

To use the QName selection method, the SOAP header elements must be the immediate children of the SOAP header. You cannot select header elements that are subelements of other elements in the SOAP header that is using QName. To select such elements, you must use an XPath expression.

You can edit the Qname panel only if **Message level protection** is selected on the Message Level Protection panel, and a part has been added in the Message Part Protection panel.

Field	Description and valid options
Name	Select a name from the list. All names created in the corresponding Message Part Protection panel are displayed.
Local part	An optional local part name within the namespace. In the following example, `securitybinding` is the namespace; within that namespace, `securityOutboundBindingConfig` and `securityInboundBindingConfig` are local parts. `<securitybinding:securityBindings xmlns:securitybinding="http://www.example.com/xmlns/ws-securitybinding"> <securitybinding:securityBinding name="application"> <securitybinding:securityOutboundBindingConfig/> <securitybinding:securityInboundBindingConfig> <securitybinding:encryptionInfo name="con_myMPPToken"> <securitybinding:keyEncryptionKeyInfo reference="con_myToken_encmyMPPToken_keyinfo"/> <securitybinding:encryptionPartReference reference="request:myMPPToken"/> </securitybinding:encryptionInfo> <securitybinding:keyInfo classname="com.ibm.ws.wssecurity.wssapi.CommonContentConsumer" name="con_auth_keyinfo" type="STRREF">`
NameSpace	The namespace of the SOAP message headers for which encryption and signature apply.

Xpath

Use this panel to define an XPath expression that refers to an element in the message to which encryption or signature applies.

Use this panel as an alternative, or in addition to, specifying XPath expressions and aliases directly on the nodes.

You cannot edit the first table in the Xpath panel, which shows five different prefix and namespaces values (based on the SOAP 1.1 specification). The second table allows the administrator to paste a fully qualified XPath expression directly into the XPath column, or select one from the list:

```
Envelope, Header, Security, Timestamp
Envelope, Header, Security
Envelope, Header, Security, Signature
```

Selecting one of the preceding options, causes the appropriate XPath expression to be created for both SOAP 1.1 (`http://schemas.xmlsoap.org/soap/envelope/`) and

SOAP 1.2 (`http://www.w3.org/2003/05/soap-envelope/`). For example, selecting `Envelope, Header, Security, Timestamp` results in the following XPath expressions being added to the policy set:

```
/*[namespace-uri()='http://schemas.xmlsoap.org/soap/envelope/'
    and local-name()='Envelope']
/*[namespace-uri()='http://schemas.xmlsoap.org/soap/envelope/'
    and local-name()='Header']
/*[namespace-uri()=
    'http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
    secext-1.0.xsd' and local-name()='Security']
/*[namespace-uri()=
    'http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
    utility-1.0.xsd' and local-name()='Timestamp']

/*[namespace-uri()='http://www.w3.org/2003/05/soap-envelope'
    and local-name()='Envelope']
/*[namespace-uri()='http://www.w3.org/2003/05/soap-envelope'
    and local-name()='Header']
/*[namespace-uri()=
    'http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
    secext-1.0.xsd' and local-name()='Security']
/*[namespace-uri()=
    'http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
    utility-1.0.xsd' and local-name()='Timestamp']
```

Line breaks have been added to enhance readability; in the Toolkit each expression is pasted on a single line. The preceding expressions show the format of the fully qualified XPath expression syntax required by the broker. If you paste your own XPath expressions into the XPath column, ensure that they adhere to this syntax. When you paste your own XPath expressions, the prefix and namespace table is unavailable for substitution of values, and only a single expression is added to the policy set, instead of both a SOAP 1.1 and SOAP 1.2 variant.

You can edit the Xpath panel only if **Message level protection** is selected on the Message Level Protection panel, and a part has been added in the Message Part Protection panel.

Field	Description and valid options
Name	Select a name from the list. All names created in the corresponding Message Part Protection panel are displayed.
XPath	A user-defined value that the administrator assigns to an element in the SOAP message for encryption or signing.

Related reference:

"Policy Sets and Policy Set Bindings editor" on page 6841
The Policy Sets and Policy Set Bindings editor is the default editor for editing, saving, importing, and exporting a policy set or binding. You can define all the WS-Security policies for a single node, or a set of nodes. The policy set can be associated by the administrator with either a message flow or a node.

"Policy Sets and Policy Set Bindings editor: Authentication tokens panel" on page 6843
Use this panel, which is in the Policy Sets section of the editor, to create Username, X.509, SAML, and LTPA authentication tokens.

"Policy Sets and Policy Set Bindings editor: Message Level Protection panel" on page 6845
Use this panel, which is in the Policy Sets section of the editor, to apply signatures and encryption to the whole message, whether inbound or outbound.

"Policy Sets and Policy Set Bindings editor: main panel" on page 6852
Use this panel, which is in the Policy Sets section of the editor, to associate a policy set binding with a policy set, and to specify whether the binding is a consumer or

provider binding.

"Policy Sets and Policy Set Bindings editor: Authentication and Protection Tokens panel" on page 6854
Use this panel, which is in the Policy Set Bindings section of the editor, to further configure any X.509 authentication tokens defined in the associated policy set.

"Policy Sets and Policy Set Bindings editor: Message Part Policies panel" on page 6856
Use this panel, which is in the Policy Set Bindings section of the editor, to further configure any message part protection tokens defined in the associated policy set.

"Policy Sets and Policy Set Bindings editor: Key Information panel" on page 6858
Use this panel, which is in the Policy Set Bindings section of the editor, to further configure any message level protection tokens that are defined in the associated policy set.

"Policy Sets and Policy Set Bindings editor: Kerberos settings panel" on page 6860
Use this panel, which is in the Policy Set Bindings section of the editor, to further configure any Kerberos tokens that are defined in the associated policy set.

"Policy Sets and Policy Set Bindings editor: Message Expiration panel" on page 6862
Use this panel, which is in the Policy Set Bindings section of the editor, to define settings for message expiration. When message expiration is enabled, the message expires after the specified time interval.

"Policy Sets and Policy Set Bindings editor: Advanced panel" on page 6863
Use this panel, which is in the Policy Set Bindings section of the editor, to configure the **mustUnderstand** attribute in the security header of the consumer message. By default this check box is selected. If you unselect the check box, you can omit the **mustUnderstand=1** attribute from the security header. This attribute applies to consumer binding only.

Policy Sets and Policy Set Bindings editor: main panel

Use this panel, which is in the Policy Sets section of the editor, to associate a policy set binding with a policy set, and to specify whether the binding is a consumer or provider binding.

A policy set binding must always be associated with a policy set. When adding a policy set binding, the administrator must select, on the main panel, the policy set with which it is associated. Changing the associated policy set on the main panel removes any previous configuration for the policy set binding.

The administrator must also select whether the policy set binding is to be a `Consumer` or `Provider` binding.

Consumer

A consumer policy set binding can apply only to the following nodes:

SOAPRequest

SOAPAsyncRequest

SOAPAsyncResponse

Provider

A provider policy set binding can apply only to the following nodes:

SOAPInput

SOAPReply

If your message flow incorporates both sets of nodes, create two policy set bindings, one for the consumer nodes and the other for the provider nodes.

Associate both policy set bindings with the message flows in the BAR file editor. A policy set binding can be deleted independently of the policy set, but a policy set cannot be deleted when it has an associated policy set binding.

Use the `Compatibility Mode` option to control the internal name-spacing of the bindings schema. When `Compatibility Mode` is selected, the policy set and bindings are limited to supporting Username and X.509 tokens only. When you create policy set bindings, `Compatibility Mode` is selected automatically based on the version of the selected broker. It is also selected by default for policy set bindings that were created with previous versions of WebSphere Message Broker. If the policy set and bindings include new token types such as Kerberos, SAML pass-through, or LTPA pass-through, you must deselect the `Compatibility Mode` option. If you select `Compatibility Mode` on an existing policy set binding that includes new token types (such as Kerberos, SAML pass-through, or LTPA pass-through) a warning is displayed and these keys are ignored when the data is saved.

You can also use the panel to rename the policy set binding.

Fields

Field	Description and valid options
Name	Enter a user-defined name for the binding.
Associated Policy Set	Select the policy set with which this binding is to be associated.

This Policy Set Binding configuration will be used with:	Select the appropriate choice:
	`Consumer`
	`Provider`

Compatibility mode	When selected, this option enables Username and X.509 tokens only. Deselect this option to enable new token types such as Kerberos, SAML pass-through, or LTPA pass-through.

Related reference:

"Policy Sets and Policy Set Bindings editor" on page 6841
The Policy Sets and Policy Set Bindings editor is the default editor for editing, saving, importing, and exporting a policy set or binding. You can define all the WS-Security policies for a single node, or a set of nodes. The policy set can be associated by the administrator with either a message flow or a node.

"Policy Sets and Policy Set Bindings editor: Authentication tokens panel" on page 6843
Use this panel, which is in the Policy Sets section of the editor, to create Username, X.509, SAML, and LTPA authentication tokens.

"Policy Sets and Policy Set Bindings editor: Message Level Protection panel" on page 6845
Use this panel, which is in the Policy Sets section of the editor, to apply signatures and encryption to the whole message, whether inbound or outbound.

"Policy Sets and Policy Set Bindings editor: Message Part Protection panel" on page 6849
Use this panel, which is in the Policy Sets section of the editor, to define the parts of a message that encryption and signature apply to. Encrypted parts are used to protect message confidentiality. Signature parts are used for message integrity.

"Policy Sets and Policy Set Bindings editor: Authentication and Protection Tokens panel"
Use this panel, which is in the Policy Set Bindings section of the editor, to further configure any X.509 authentication tokens defined in the associated policy set.

"Policy Sets and Policy Set Bindings editor: Message Part Policies panel" on page 6856
Use this panel, which is in the Policy Set Bindings section of the editor, to further configure any message part protection tokens defined in the associated policy set.

"Policy Sets and Policy Set Bindings editor: Key Information panel" on page 6858
Use this panel, which is in the Policy Set Bindings section of the editor, to further configure any message level protection tokens that are defined in the associated policy set.

"Policy Sets and Policy Set Bindings editor: Kerberos settings panel" on page 6860
Use this panel, which is in the Policy Set Bindings section of the editor, to further configure any Kerberos tokens that are defined in the associated policy set.

"Policy Sets and Policy Set Bindings editor: Message Expiration panel" on page 6862
Use this panel, which is in the Policy Set Bindings section of the editor, to define settings for message expiration. When message expiration is enabled, the message expires after the specified time interval.

"Policy Sets and Policy Set Bindings editor: Advanced panel" on page 6863
Use this panel, which is in the Policy Set Bindings section of the editor, to configure the **mustUnderstand** attribute in the security header of the consumer message. By default this check box is selected. If you unselect the check box, you can omit the **mustUnderstand=1** attribute from the security header. This attribute applies to consumer binding only.

Policy Sets and Policy Set Bindings editor: Authentication and Protection Tokens panel

Use this panel, which is in the Policy Set Bindings section of the editor, to further configure any X.509 authentication tokens defined in the associated policy set.

Fields

The table is prefilled, based on the following criteria

- Whether any X.509 authentication tokens exist in the associated policy set.
- Whether the SOAP message type of the authentication token is request or response.
- Whether this policy set binding is defined as being a consumer or provider.

Each authentication token identified as requiring further information is added to the table. An authentication token can require one of two types of additional information:

- Key information in the form of a key name and key alias, for lookup in the broker keystore.
- Verification information, which can be either `TrustAny` or `TrustStore`.

This table shows the different combinations of configuration for authentication tokens and whether key information or verification information is required:

Authentication X.509 tokens					
Policy set configuration	**Policy set bindings configuration**	**SOAP message**	**Key information**	**Key password**	**Verification information**
request	provider	inbound	N/A	N/A	required
response	provider	outbound	required	required	N/A
request	consumer	outbound	required	required	N/A
response	consumer	inbound	N/A	N/A	required

Where N/A is displayed in a field, no information is required. Where an authentication token is displayed, enter information in all fields that do not display N/A, so that the policy set binding can be generated correctly in accordance with the associated policy set.

Configure the broker to refer to a keystore and truststore. You might also need to configure passwords for these stores, and specific key passwords. See "Viewing and setting keystore and truststore runtime properties at broker level" on page 780 for further information.

Field name	Description and valid options
Authentication Token Name	Displays the names of all authentication X.509 tokens that require further configuration. The token name is displayed after either `request:` or `response:`, depending on the configuration of the token in the associated policy set.
Key name	The distinguished name (DN) that uniquely identifies the key in the keystore defined by the broker. For example `"CN=CommonName, O=Organisation, C=Country"`
Key Alias	The key alias of the key in the keystore defined by the broker. The broker also uses the key alias to look up the keystore password associated with this key. You define this in the broker using the **mqsisetdbparms** command.
Certificates	Either: **TrustAny** With no security profile set, all certificates are trusted. With a security profile set, the certificate is passed to the security provider defined by the security profile for it to establish trust. See "Setting up message flow security" on page 431. **TrustStore** Certificates are checked against the public key certificates in the truststore defined by the broker.

Related reference:

"Policy Sets and Policy Set Bindings editor" on page 6841
The Policy Sets and Policy Set Bindings editor is the default editor for editing, saving, importing, and exporting a policy set or binding. You can define all the WS-Security policies for a single node, or a set of nodes. The policy set can be associated by the administrator with either a message flow or a node.

"Policy Sets and Policy Set Bindings editor: Authentication tokens panel" on page 6843
Use this panel, which is in the Policy Sets section of the editor, to create Username,

X.509, SAML, and LTPA authentication tokens.

"Policy Sets and Policy Set Bindings editor: Message Level Protection panel" on page 6845
Use this panel, which is in the Policy Sets section of the editor, to apply signatures and encryption to the whole message, whether inbound or outbound.

"Policy Sets and Policy Set Bindings editor: Message Part Protection panel" on page 6849
Use this panel, which is in the Policy Sets section of the editor, to define the parts of a message that encryption and signature apply to. Encrypted parts are used to protect message confidentiality. Signature parts are used for message integrity.

"Policy Sets and Policy Set Bindings editor: main panel" on page 6852
Use this panel, which is in the Policy Sets section of the editor, to associate a policy set binding with a policy set, and to specify whether the binding is a consumer or provider binding.

"Policy Sets and Policy Set Bindings editor: Message Part Policies panel"
Use this panel, which is in the Policy Set Bindings section of the editor, to further configure any message part protection tokens defined in the associated policy set.

"Policy Sets and Policy Set Bindings editor: Key Information panel" on page 6858
Use this panel, which is in the Policy Set Bindings section of the editor, to further configure any message level protection tokens that are defined in the associated policy set.

"Policy Sets and Policy Set Bindings editor: Kerberos settings panel" on page 6860
Use this panel, which is in the Policy Set Bindings section of the editor, to further configure any Kerberos tokens that are defined in the associated policy set.

"Policy Sets and Policy Set Bindings editor: Message Expiration panel" on page 6862
Use this panel, which is in the Policy Set Bindings section of the editor, to define settings for message expiration. When message expiration is enabled, the message expires after the specified time interval.

"Policy Sets and Policy Set Bindings editor: Advanced panel" on page 6863
Use this panel, which is in the Policy Set Bindings section of the editor, to configure the **mustUnderstand** attribute in the security header of the consumer message. By default this check box is selected. If you unselect the check box, you can omit the **mustUnderstand=1** attribute from the security header. This attribute applies to consumer binding only.

Policy Sets and Policy Set Bindings editor: Message Part Policies panel

Use this panel, which is in the Policy Set Bindings section of the editor, to further configure any message part protection tokens defined in the associated policy set.

The panel has two tables: the first table is for tokens that are used to encrypt message parts. The second table is for tokens used to sign message parts.

The tables are prefilled, based on the following criteria:

- Whether any message level protection tokens exist in the associated policy set.
- Whether the type of the message level protection token is initiator or recipient.
- Whether any message part protection tokens exist, and whether they define the message body, alias, Qname or XPath.
- Whether the SOAP message type of the message part protection token is request or response.
- Whether this policy set binding is defined as being a consumer or provider.

Each message part protection token that is added to one of the tables requires additional settings in either the Key information panel or the Kerberos Settings panel.

If you are using X.509 certificates, you must configure the Message Broker runtime environment to refer to a keystore and truststore. You might also need to configure passwords for these stores, and specific key passwords. See "Viewing and setting keystore and truststore runtime properties at broker level" on page 780 for further information.

If you are using Kerberos tickets, you must provide client credentials for accessing the Kerberos Key Distribution Center (KDC). You can provide these credentials either through a *Username and password* token type in the message tree properties folder, or by using the **mqsisetdbparms** command. For more information, see "Implementing WS-Security" on page 769.

Fields

Column Name	Description and valid options
Encryption Protection and Signature Protection	Displays the names of any Message Part Protection tokens that require further configuration. The token name is displayed after either `request:` or `response:`, depending on the configuration of the token in the associated policy set.

Timestamp	Either:
	`Yes`
	`No`

Nonce	Either:
	`Yes`
	`No`

Encryption	Either:
	`Data`
	`Key`

Token	Click this column to see a list of all message level protection tokens. Select the token that is to be associated with the message part protection token.
Token Type	Click this column to see a list of all token types. Token types can be specified for outbound policies and optionally for inbound policies. Select the appropriate token type from the list. Valid options are `STRREF`, `KEYID`, `EMB`, `KEYNAME`, and `X509ISSUER`.
Order	Defines the order that response message part policies should be processed in. N/A is displayed where this column is not required for certain combinations of tokens.

Related reference:

"Policy Sets and Policy Set Bindings editor" on page 6841
The Policy Sets and Policy Set Bindings editor is the default editor for editing, saving, importing, and exporting a policy set or binding. You can define all the WS-Security policies for a single node, or a set of nodes. The policy set can be associated by the administrator with either a message flow or a node.

"Policy Sets and Policy Set Bindings editor: Authentication tokens panel" on page 6843
Use this panel, which is in the Policy Sets section of the editor, to create Username, X.509, SAML, and LTPA authentication tokens.

"Policy Sets and Policy Set Bindings editor: Message Level Protection panel" on page 6845
Use this panel, which is in the Policy Sets section of the editor, to apply signatures and encryption to the whole message, whether inbound or outbound.

"Policy Sets and Policy Set Bindings editor: Message Part Protection panel" on page 6849
Use this panel, which is in the Policy Sets section of the editor, to define the parts of a message that encryption and signature apply to. Encrypted parts are used to protect message confidentiality. Signature parts are used for message integrity.

"Policy Sets and Policy Set Bindings editor: main panel" on page 6852
Use this panel, which is in the Policy Sets section of the editor, to associate a policy set binding with a policy set, and to specify whether the binding is a consumer or provider binding.

"Policy Sets and Policy Set Bindings editor: Authentication and Protection Tokens panel" on page 6854
Use this panel, which is in the Policy Set Bindings section of the editor, to further configure any X.509 authentication tokens defined in the associated policy set.

"Policy Sets and Policy Set Bindings editor: Key Information panel"
Use this panel, which is in the Policy Set Bindings section of the editor, to further configure any message level protection tokens that are defined in the associated policy set.

"Policy Sets and Policy Set Bindings editor: Kerberos settings panel" on page 6860
Use this panel, which is in the Policy Set Bindings section of the editor, to further configure any Kerberos tokens that are defined in the associated policy set.

"Policy Sets and Policy Set Bindings editor: Message Expiration panel" on page 6862
Use this panel, which is in the Policy Set Bindings section of the editor, to define settings for message expiration. When message expiration is enabled, the message expires after the specified time interval.

"Policy Sets and Policy Set Bindings editor: Advanced panel" on page 6863
Use this panel, which is in the Policy Set Bindings section of the editor, to configure the **mustUnderstand** attribute in the security header of the consumer message. By default this check box is selected. If you unselect the check box, you can omit the **mustUnderstand=1** attribute from the security header. This attribute applies to consumer binding only.

"**mqsisetdbparms** command" on page 3954
Use the **mqsisetdbparms** command to associate a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker.

Policy Sets and Policy Set Bindings editor: Key Information panel

Use this panel, which is in the Policy Set Bindings section of the editor, to further configure any message level protection tokens that are defined in the associated policy set.

The table is prefilled based on the selections made in the message part policy panel. Different combinations of message level protection tokens and message part protection tokens require one or more of the following types of additional information:

- Key information in the form of a key name and key alias, for lookup in the broker keystore.
- Verification information, which can be either `trustany` or `truststore`.

Where a message level protection token is displayed, enter the required level of information so that the policy set binding can be generated correctly in accordance with the associated policy set.

Configure the broker to refer to a keystore and truststore. You might also need to configure passwords for these stores, and specific key passwords. See "Viewing and setting keystore and truststore runtime properties at broker level" on page 780 for further information.

Fields

Field name	Description and valid options
Token	Displays the names of any message level protection tokens that require further configuration. The token name is displayed after either `request:` or `response:`, depending on the configuration of the token in the associated policy set.
Key Name	The distinguished name (DN) that uniquely identifies the key in the keystore that is defined in the broker using the **mqsichangeproperties** command. For example `"CN=CommonName, O=Organisation, C=Country"`. When you require message level protection on the inbound signature token and the key is not known in advance, enter `Any` or leave blank.
Key Alias	The key alias of the key in the keystore defined by the broker. When you require message level protection on the inbound signature token and the key is not known in advance, enter `Any` or leave blank. You define the broker keystores using the **mqsisetdbparms** and **mqsichangeproperties** commands.
Trust	Either: **`TrustAny`** With no security profile set, all certificates are trusted. With a security profile set, the certificate is passed to the security provider defined by the security profile for it to establish trust. See "Setting up message flow security" on page 431. **`TrustStore`** Check against the public key certificates in the truststore defined by the broker.

Related reference:

"Policy Sets and Policy Set Bindings editor" on page 6841
The Policy Sets and Policy Set Bindings editor is the default editor for editing, saving, importing, and exporting a policy set or binding. You can define all the WS-Security policies for a single node, or a set of nodes. The policy set can be associated by the administrator with either a message flow or a node.

"Policy Sets and Policy Set Bindings editor: Authentication tokens panel" on page 6843
Use this panel, which is in the Policy Sets section of the editor, to create Username, X.509, SAML, and LTPA authentication tokens.

"Policy Sets and Policy Set Bindings editor: Message Level Protection panel" on page 6845
Use this panel, which is in the Policy Sets section of the editor, to apply signatures and encryption to the whole message, whether inbound or outbound.

"Policy Sets and Policy Set Bindings editor: Message Part Protection panel" on page 6849
Use this panel, which is in the Policy Sets section of the editor, to define the parts of a message that encryption and signature apply to. Encrypted parts are used to protect message confidentiality. Signature parts are used for message integrity.

"Policy Sets and Policy Set Bindings editor: main panel" on page 6852
Use this panel, which is in the Policy Sets section of the editor, to associate a policy set binding with a policy set, and to specify whether the binding is a consumer or provider binding.

"Policy Sets and Policy Set Bindings editor: Authentication and Protection Tokens panel" on page 6854
Use this panel, which is in the Policy Set Bindings section of the editor, to further configure any X.509 authentication tokens defined in the associated policy set.

"Policy Sets and Policy Set Bindings editor: Message Part Policies panel" on page 6856
Use this panel, which is in the Policy Set Bindings section of the editor, to further configure any message part protection tokens defined in the associated policy set.

"Policy Sets and Policy Set Bindings editor: Message Expiration panel" on page 6862
Use this panel, which is in the Policy Set Bindings section of the editor, to define settings for message expiration. When message expiration is enabled, the message expires after the specified time interval.

"Policy Sets and Policy Set Bindings editor: Kerberos settings panel"
Use this panel, which is in the Policy Set Bindings section of the editor, to further configure any Kerberos tokens that are defined in the associated policy set.

"Policy Sets and Policy Set Bindings editor: Advanced panel" on page 6863
Use this panel, which is in the Policy Set Bindings section of the editor, to configure the **mustUnderstand** attribute in the security header of the consumer message. By default this check box is selected. If you unselect the check box, you can omit the **mustUnderstand=1** attribute from the security header. This attribute applies to consumer binding only.

Policy Sets and Policy Set Bindings editor: Kerberos settings panel

Use this panel, which is in the Policy Set Bindings section of the editor, to further configure any Kerberos tokens that are defined in the associated policy set.

This panel is enabled only if the `Compatibility Mode` option in the Policy Sets and Policy Set Bindings editor: Main panel is deselected.

The table is pre-filled based on the Kerberos tokens in the associated policy. Symmetric tokens configured in the policy require additional information for consumer bindings only.

Fields

Field name	Description and valid options
Token	Displays the names of any Kerberos tokens that require further configuration.
Target Service Realm	The Kerberos target service realm. The default is a blank string, which results in the Kerberos default being used; other realms can be specified by changing this value.

Field name	Description and valid options
Target Service Name	The Kerberos target service name. The default is a blank string, which results in the default from the WSDL being used; other names can be specified by changing this value.
Target Service Host	The Kerberos target service host. The default is a blank string, which results in the Kerberos default being used; other hosts can be specified by changing this value. This option is not applicable to service providers.
Require Derived Keys	Specify derived keys. Valid values are *True* or *False*. The default is *True*.
Derived Key Token Namespace	The namespace to use for derived keys. The default value is *WS-SecureConversation 1.3*.
Key Size	The key size for derived keys. The default is *16*.
Acquire New Tokens	Specifies whether a new ticket must be acquired from the Key Distribution Center (KDC) for each request. Valid values are *True* or *False*. The default is *True*. This option is not applicable to service providers.

Related reference:

"Policy Sets and Policy Set Bindings editor" on page 6841
The Policy Sets and Policy Set Bindings editor is the default editor for editing, saving, importing, and exporting a policy set or binding. You can define all the WS-Security policies for a single node, or a set of nodes. The policy set can be associated by the administrator with either a message flow or a node.

"Policy Sets and Policy Set Bindings editor: Authentication tokens panel" on page 6843
Use this panel, which is in the Policy Sets section of the editor, to create Username, X.509, SAML, and LTPA authentication tokens.

"Policy Sets and Policy Set Bindings editor: Message Level Protection panel" on page 6845
Use this panel, which is in the Policy Sets section of the editor, to apply signatures and encryption to the whole message, whether inbound or outbound.

"Policy Sets and Policy Set Bindings editor: Message Part Protection panel" on page 6849
Use this panel, which is in the Policy Sets section of the editor, to define the parts of a message that encryption and signature apply to. Encrypted parts are used to protect message confidentiality. Signature parts are used for message integrity.

"Policy Sets and Policy Set Bindings editor: main panel" on page 6852
Use this panel, which is in the Policy Sets section of the editor, to associate a policy set binding with a policy set, and to specify whether the binding is a consumer or provider binding.

"Policy Sets and Policy Set Bindings editor: Authentication and Protection Tokens panel" on page 6854
Use this panel, which is in the Policy Set Bindings section of the editor, to further configure any X.509 authentication tokens defined in the associated policy set.

"Policy Sets and Policy Set Bindings editor: Message Part Policies panel" on page 6856
Use this panel, which is in the Policy Set Bindings section of the editor, to further configure any message part protection tokens defined in the associated policy set.

"Policy Sets and Policy Set Bindings editor: Key Information panel" on page 6858
Use this panel, which is in the Policy Set Bindings section of the editor, to further configure any message level protection tokens that are defined in the associated

policy set.

"Policy Sets and Policy Set Bindings editor: Message Expiration panel"
Use this panel, which is in the Policy Set Bindings section of the editor, to define settings for message expiration. When message expiration is enabled, the message expires after the specified time interval.

"Policy Sets and Policy Set Bindings editor: Advanced panel" on page 6863
Use this panel, which is in the Policy Set Bindings section of the editor, to configure the **mustUnderstand** attribute in the security header of the consumer message. By default this check box is selected. If you unselect the check box, you can omit the **mustUnderstand=1** attribute from the security header. This attribute applies to consumer binding only.

Policy Sets and Policy Set Bindings editor: Message Expiration panel

Use this panel, which is in the Policy Set Bindings section of the editor, to define settings for message expiration. When message expiration is enabled, the message expires after the specified time interval.

Fields

Field Name	Description and valid options
Enable message expiration	Enables message expiration.
Message timeout	Timeout value for message expiration in minutes.

Related reference:

"Policy Sets and Policy Set Bindings editor" on page 6841
The Policy Sets and Policy Set Bindings editor is the default editor for editing, saving, importing, and exporting a policy set or binding. You can define all the WS-Security policies for a single node, or a set of nodes. The policy set can be associated by the administrator with either a message flow or a node.

"Policy Sets and Policy Set Bindings editor: Authentication tokens panel" on page 6843
Use this panel, which is in the Policy Sets section of the editor, to create Username, X.509, SAML, and LTPA authentication tokens.

"Policy Sets and Policy Set Bindings editor: Message Level Protection panel" on page 6845
Use this panel, which is in the Policy Sets section of the editor, to apply signatures and encryption to the whole message, whether inbound or outbound.

"Policy Sets and Policy Set Bindings editor: Message Part Protection panel" on page 6849
Use this panel, which is in the Policy Sets section of the editor, to define the parts of a message that encryption and signature apply to. Encrypted parts are used to protect message confidentiality. Signature parts are used for message integrity.

"Policy Sets and Policy Set Bindings editor: main panel" on page 6852
Use this panel, which is in the Policy Sets section of the editor, to associate a policy set binding with a policy set, and to specify whether the binding is a consumer or provider binding.

"Policy Sets and Policy Set Bindings editor: Authentication and Protection Tokens panel" on page 6854
Use this panel, which is in the Policy Set Bindings section of the editor, to further configure any X.509 authentication tokens defined in the associated policy set.

"Policy Sets and Policy Set Bindings editor: Message Part Policies panel" on page 6856
Use this panel, which is in the Policy Set Bindings section of the editor, to further configure any message part protection tokens defined in the associated policy set.

"Policy Sets and Policy Set Bindings editor: Key Information panel" on page 6858
Use this panel, which is in the Policy Set Bindings section of the editor, to further configure any message level protection tokens that are defined in the associated policy set.

"Policy Sets and Policy Set Bindings editor: Kerberos settings panel" on page 6860
Use this panel, which is in the Policy Set Bindings section of the editor, to further configure any Kerberos tokens that are defined in the associated policy set.

"Policy Sets and Policy Set Bindings editor: Advanced panel"
Use this panel, which is in the Policy Set Bindings section of the editor, to configure the **mustUnderstand** attribute in the security header of the consumer message. By default this check box is selected. If you unselect the check box, you can omit the **mustUnderstand=1** attribute from the security header. This attribute applies to consumer binding only.

Policy Sets and Policy Set Bindings editor: Advanced panel

Use this panel, which is in the Policy Set Bindings section of the editor, to configure the **mustUnderstand** attribute in the security header of the consumer message. By default this check box is selected. If you unselect the check box, you can omit the **mustUnderstand=1** attribute from the security header. This attribute applies to consumer binding only.

Fields

Field Name	Description and valid options
Include mustUnderstand attribute in security header	If you select this option, the **mustUnderstand** attribute of security header is created with the value 1. Otherwise, the security header is created with the value 0.
Check timestamp of response message	If you select this option, the **ws-security timestamp** response of the message is checked. If you unselect this option, the broker does not look for a timestamp in the response message. This option is valid when you are using a consumer binding and you have enabled the option **Include timestamp in security header** of the policy set Message Level Protection panel.

Related reference:

"Policy Sets and Policy Set Bindings editor" on page 6841
The Policy Sets and Policy Set Bindings editor is the default editor for editing, saving, importing, and exporting a policy set or binding. You can define all the WS-Security policies for a single node, or a set of nodes. The policy set can be associated by the administrator with either a message flow or a node.

"Policy Sets and Policy Set Bindings editor: Authentication tokens panel" on page 6843
Use this panel, which is in the Policy Sets section of the editor, to create Username, X.509, SAML, and LTPA authentication tokens.

"Policy Sets and Policy Set Bindings editor: Message Level Protection panel" on page 6845
Use this panel, which is in the Policy Sets section of the editor, to apply signatures and encryption to the whole message, whether inbound or outbound.

"Policy Sets and Policy Set Bindings editor: Message Part Protection panel" on page 6849
Use this panel, which is in the Policy Sets section of the editor, to define the parts of a message that encryption and signature apply to. Encrypted parts are used to protect message confidentiality. Signature parts are used for message integrity.

"Policy Sets and Policy Set Bindings editor: main panel" on page 6852
Use this panel, which is in the Policy Sets section of the editor, to associate a policy set binding with a policy set, and to specify whether the binding is a consumer or provider binding.

"Policy Sets and Policy Set Bindings editor: Authentication and Protection Tokens panel" on page 6854
Use this panel, which is in the Policy Set Bindings section of the editor, to further configure any X.509 authentication tokens defined in the associated policy set.

"Policy Sets and Policy Set Bindings editor: Message Part Policies panel" on page 6856
Use this panel, which is in the Policy Set Bindings section of the editor, to further configure any message part protection tokens defined in the associated policy set.

"Policy Sets and Policy Set Bindings editor: Key Information panel" on page 6858
Use this panel, which is in the Policy Set Bindings section of the editor, to further configure any message level protection tokens that are defined in the associated policy set.

"Policy Sets and Policy Set Bindings editor: Kerberos settings panel" on page 6860
Use this panel, which is in the Policy Set Bindings section of the editor, to further configure any Kerberos tokens that are defined in the associated policy set.

Troubleshooting

Use the reference information in this section to help you diagnose errors in WebSphere Message Broker.

Select the appropriate topic from the following list to diagnose problems and errors in WebSphere Message Broker:

- "Logs"
- "Trace" on page 6871
- "Trace logging from a user-defined C extension" on page 6693
- "Dumps" on page 6877
- "Abend files" on page 6880
- "WebSphere Message Broker event reports" on page 6883
- "WebSphere MQ facilities" on page 6890
- "Database facilities" on page 6891
- "Other sources of diagnostic information on z/OS" on page 6891
- "Solutions to similar problems" on page 6892
- "Resolving problems when installing" on page 3517
- "Resolving problems when uninstalling" on page 3525

Logs

If an error is reported by a WebSphere Message Broker component, start your investigations into its causes by looking at the product and systems logs to which information is written during component operation.

The information that is recorded in a log typically consists of a time stamp, which indicates when the error occurred, and a brief description of the error, which can be expanded to provide more details of the error. In some situations, a general error is written and is followed by a group of more specific errors that provide details about the general error.

When an error occurs, check STDOUT, STDERR, and the local error log first. These logs record information about major activities within the system. All components of WebSphere Message Broker provide diagnostic information whenever error or warning conditions affect broker operation. These conditions include:

- Unsuccessful attempts to write a message to a WebSphere MQ output queue
- Errors interacting with databases
- The inability to parse an input message

Additional logs that are specific to WebSphere Message Broker are written to record runtime errors, internal errors that are produced by the operating system or your code, or errors related to the work that you are doing in a particular perspective, all of which you can view using the WebSphere Message Broker Toolkit.

Data is written to the following logs:

- "Standard system logs" on page 6866
- "Local error logs" on page 6867
- "WebSphere Message Broker logs" on page 6867
- "WebSphere MQ logs" on page 6869
- "Database logs" on page 6870

Related tasks:

"Viewing Administration log information" on page 1007
You can view Administration log information by using either the WebSphere Message Broker Explorer, or the CMP API.

"Windows: Viewing the local error log" on page 3527
The Windows Event Viewer is where WebSphere Message Broker writes records to the local system. Use Windows system facilities to view this log.

"Linux and UNIX systems: Configuring the syslog daemon" on page 3529
On Linux and UNIX systems, all WebSphere Message Broker messages (other than messages that are generated by the command-line utilities) are sent to the syslog subsystem.

"z/OS: Viewing broker job logs" on page 3530
On z/OS, the broker writes messages to the appropriate z/OS system log and job logs. These messages might include information, warning, error, and severe messages to indicate various situations and events.

Related reference:

"Administration Log view" on page 6840
The Administration Log view shows administration requests and their results, changes made to objects, and the result of deployment actions on brokers. The Administration Log view can also describe configuration changes that have been automatically applied as a result of moving the broker from one version to another; for example, after applying maintenance.

Standard system logs

The broker writes information, warning, and error messages to standard output (STDOUT) and standard error (STDERR). The broker records this information for the execution group and for the broker admin agent (bipbroker process).

On some operating systems, the two standard logs are written to separate files, but the system might provide an option for the data to be recorded to a single merged file. On other systems, the two targets are defined to a single, and you might have the option to separate them. Check the documentation that is provided with your operating system for more details about STDOUT and STDERR.

Brokers write records to both STDOUT and STDERR. Always check the contents of the file or files when an error occurs.

Because the records are written whenever components are active, check occasionally that the location of the file or files has sufficient space, or trim the file contents on a regular basis.

The default location of STDOUT and STDERR depends on the operating system:

Operating system	Location of STDOUT and STDERR for broker admin agent	Location of STDOUT and STDERR for execution group
Linux UNIX Linux and UNIX systems	`/var/mqsi/components/`*`broker_name`*	`/var/mqsi/components/`*`broker_name`*`/`*`execution_group_uuid`*
Windows Windows systems	A single file, *`workpath`*`\components\`*`broker_name`*`\console.txt`, contains both STDOUT and STDERR data, generated by the admin agent, where *workpath* is the working directory defined for the broker.	A single file, *`workpath`*`\components\`*`broker_name`*`\`*`execution_group_uuid`*`\console.txt`, contains both STDOUT and STDERR data, where *workpath* is the working directory defined for the broker.
z/OS z/OS systems	The job log for the broker control address space.	The job log for the execution group address spaces.

Related tasks:

"Viewing Administration log information" on page 1007
You can view Administration log information by using either the WebSphere Message Broker Explorer, or the CMP API.

"Windows: Viewing the local error log" on page 3527
The Windows Event Viewer is where WebSphere Message Broker writes records to the local system. Use Windows system facilities to view this log.

"Linux and UNIX systems: Configuring the syslog daemon" on page 3529
On Linux and UNIX systems, all WebSphere Message Broker messages (other than messages that are generated by the command-line utilities) are sent to the syslog subsystem.

Related reference:

"Sample BIPCRBK file" on page 4006
The sample `BIPCRBK` file that is shipped with WebSphere Message Broker is included here for your reference.

Local error logs

WebSphere Message Broker components use the local error log (also known as the system log) to record information about major activities within the system. When an error occurs, check the local error log first.

Windows On Windows, the local error log is the Windows Event log (Application view).

UNIX On UNIX and Linux systems, the local error log is the *syslog*. Where syslog messages are sent depends on how you configure your UNIX or Linux system.

z/OS On z/OS, the local error log is the operator console.

Entries in the local error log that are generated by WebSphere Message Broker are identified in the following way:

- Windows On Windows: by events from the source `WebSphere Broker v****` and a message number in the form `nnnn`, where `****` is the current four-digit product version number.
- UNIX On UNIX: by a message number in the form `BIPnnnn`.
- z/OS On z/OS: by a started task ID, and a message number in the form `BIPnnnn`.

When a broker encounters an error, more than one message can be written to the local error log. Typically these messages start with a general message (for example, `Could not process a message`), and further messages give more details about the cause of the error (for example, `An error was detected while processing the following SQL statement`).

Windows On Windows, the event log can fill up; ensure that you have a log size that is sufficient, or that you have enabled circular logging.

Related concepts:

"Logs" on page 6864
If an error is reported by a WebSphere Message Broker component, start your investigations into its causes by looking at the product and systems logs to which information is written during component operation.

Related tasks:

"Windows: Viewing the local error log" on page 3527
The Windows Event Viewer is where WebSphere Message Broker writes records to the local system. Use Windows system facilities to view this log.

"Linux and UNIX systems: Configuring the syslog daemon" on page 3529
On Linux and UNIX systems, all WebSphere Message Broker messages (other than messages that are generated by the command-line utilities) are sent to the syslog subsystem.

"z/OS: Viewing broker job logs" on page 3530
On z/OS, the broker writes messages to the appropriate z/OS system log and job logs. These messages might include information, warning, error, and severe messages to indicate various situations and events.

WebSphere Message Broker logs

WebSphere Message Broker writes information to a number of product-specific logs to report the results of actions that you take.

The following logs are used for reporting errors and events:

- "Eclipse error log"
- "Broker Administration log or Deployment log"
- "The Problems or Alerts view"
- "TDS log"

Eclipse error log:
The Eclipse error log captures internal errors from Eclipse and the code associated with your user-defined extensions. Check this log at run time for exceptions that you might have caused during development. When errors are triggered, they are added to the Error Log view of the Plug-in Development perspective, and you can double-click each one to examine details. Entries are sorted in reverse chronological order; that is, most recent first. You can open the Properties view to see entry details. You can also clear the view when you have fixed all the errors, although you cannot clear individual errors.

Some errors might be hierarchical; that is, a general entry for a complex problem might be followed by a number of child entries that list the individual problems in detail.

The same information that is shown in the Error Log view is stored in the `.log` file in the workspace directory. The default location for the workspace directory depends on your environment:

- Linux Linux on x86: *user_home_dir*/IBM/wmbt70/workspace/
- Windows Windows XP and Windows Server 2003: C:\Documents and Settings*user_ID*\IBM\wmbt70\workspace\
- Windows Windows 7, Windows Vista and Windows Server 2008: C:\Users*user_ID*\IBM\wmbt70\workspace\

The Eclipse error log shows errors that are generated only by the user ID that is working with that particular WebSphere Message Broker Toolkit session.

Broker Administration log or Deployment log:
Broker Administration log information is written to the Administration log. This log is stored and managed by the broker. The Administration log and displays messages about events that occur within the broker, such as the results of a deployment, or a change to broker properties. The messages can be information, errors, or warnings. You view the messages in the Administration Log view in the WebSphere Message Broker Explorer, and the Deployment log in the WebSphere Message Broker Toolkit.

The Problems or Alerts view:
Either the Problems or the Alerts view is displayed, depending on the perspective being used at the time.

- In the Broker Application Development perspective the Problems view shows information messages, warnings, and errors for message flow applications and associated resources, such as message flows, message sets, ESQL, Java, and mapping.

TDS log:
When you use the Tagged/Delimited String (TDS) physical format messages, you must conform to a number of rules that apply to the setting of values of properties. These rules are checked at deploy time, and, if an inconsistency is found, error

message BIP1836 is displayed in the WebSphere Message Broker Toolkit. Details of the error are written to the `TDS.log` file in the *install_dir*`/log` directory.

For more information about the rules for TDS physical format properties, see "TDS message model integrity" on page 6295.

Related concepts:

"Logs" on page 6864
If an error is reported by a WebSphere Message Broker component, start your investigations into its causes by looking at the product and systems logs to which information is written during component operation.

Related tasks:

"Viewing the Eclipse error log" on page 3532
The Eclipse error log captures internal errors that are caused by the operating system or your code.

"Windows: Viewing the local error log" on page 3527
The Windows Event Viewer is where WebSphere Message Broker writes records to the local system. Use Windows system facilities to view this log.

Related reference:

"Local error logs" on page 6867
WebSphere Message Broker components use the local error log (also known as the system log) to record information about major activities within the system. When an error occurs, check the local error log first.

"WebSphere MQ logs"
WebSphere MQ messages are written to the local error log in the same way as WebSphere Message Broker messages.

"Database logs" on page 6870
Databases write severe error and warning conditions to the local error log (syslog). Typically, databases also write errors to a database log file, such as the `db2diag.log` file on DB2.

WebSphere MQ logs

WebSphere MQ messages are written to the local error log in the same way as WebSphere Message Broker messages.

WebSphere MQ messages start with the prefix AMQ followed by a number between 4000 and 9999. The number specifies from which part of WebSphere MQ the message originates:

4000 to 4999
: Windows user interface messages

5000 to 5999
: Installable services messages

6000 to 6999
: Common services messages

7000 to 7999
: WebSphere MQ product messages

8000 to 8999
: WebSphere MQ administration messages

9000 to 9999
: Remote messages

For a full explanation of each message, see the WebSphere MQ Library web page.

Related tasks:

Chapter 13, "Troubleshooting and support," on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

"Using logs" on page 3526
There are a variety of logs that you can use to help with problem determination and troubleshooting.

"Windows: Viewing the local error log" on page 3527
The Windows Event Viewer is where WebSphere Message Broker writes records to the local system. Use Windows system facilities to view this log.

"Linux and UNIX systems: Configuring the syslog daemon" on page 3529
On Linux and UNIX systems, all WebSphere Message Broker messages (other than messages that are generated by the command-line utilities) are sent to the syslog subsystem.

Related reference:

"Local error logs" on page 6867
WebSphere Message Broker components use the local error log (also known as the system log) to record information about major activities within the system. When an error occurs, check the local error log first.

"WebSphere MQ facilities" on page 6890
WebSphere Message Broker components depend on WebSphere MQ resources in many ways. You can therefore gain valuable information from the WebSphere MQ logs and events.

Database logs

Databases write severe error and warning conditions to the local error log (syslog). Typically, databases also write errors to a database log file, such as the `db2diag.log` file on DB2.

You can open the `db2diag.log` file in a text editor.

The following example shows a typical entry from the `db2diag.log` file:

```
Jun 18 15:02:53 bluj DB2[46827]: DB2(db2inst1.000(1))oper_system_services sqlobeep(2)
 reports:(3)
Jun 18 15:02:53 (4) bluj (5) DB2[46827(6)]: extra symptom string provided:(7) RIDS/sqlesysc_
Jun 18 15:02:53 bluj DB2[46827]:     data: (8)  54686973  20697320  616e2065  78616d70
Jun 18 15:02:53 bluj DB2[46827]:     data:   6c65206f  66206c6f  67676564  20646174
Jun 18 15:02:53 bluj DB2[46827]:     data:   61
Jun 18 15:02:53 bluj DB2[46827]: 2 piece(s) of dump data provided... to file(9) /u/db2inst1/
Jun 18 15:02:53 bluj DB2[46827]:         1. 'DUMP EXAMPLE #1' has been dumped (10)
Jun 18 15:02:53 bluj DB2[46827]:         2. 'DUMP EXAMPLE #2' has been dumped
```

The bold numbers in the example show various items in the log file:

(1) The instance name and node number

(2) The reporting component and function

(3) The probe ID and error and alert numbers

(4) A time stamp for when the event occurred

(5) The host name

(6) The process ID of the reporting process. Use the **ps** command to view

information about the process ID of the reporting process. For example, enter the following command to get information about the reporting process:

```
ps -fu 46827
```

(7) A symptom string that contains additional information about where and why the problem occurred

(8) A hexadecimal dump of data that includes return codes and other information that can be interpreted by your IBM Support Center

(9) Information about additional dump files. Larger structures and other binary data might be dumped to additional files. The name of the file is identified in the syslog file

(10) An entry to identify a piece of dump data

Related tasks:

Chapter 13, "Troubleshooting and support," on page 3345
If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

"Using logs" on page 3526
There are a variety of logs that you can use to help with problem determination and troubleshooting.

Related reference:

"Database facilities" on page 6891
The database products used by WebSphere Message Broker also record information that might be useful if you have any problems with their access.

Trace

If you cannot get enough information about a particular problem from the entries that are available in the various logs, the next troubleshooting method to consider is using trace. Trace provides more details about what is happening while code runs. The information produced from trace is sent to a specified trace record, so that you or IBM support personnel can analyze it to discover the cause of your problem.

Trace is inactive by default, and must be explicitly activated by a command, or by the WebSphere Message Broker Toolkit.

There are two main types of trace available in WebSphere Message Broker: user trace and service trace. Typically, you utilize user trace for debugging your applications; you can trace brokers, execution groups, and deployed message flows. With service trace, you can activate more comprehensive broker tracing, and start tracing for the WebSphere Message Broker Toolkit. You can also trace the execution of all the commands described in "Commands" on page 3672.

When you start user tracing, you cause additional processing for every activity in the component that you are tracing. Large quantities of data are generated by the components. Expect performance to be affected while trace is active. You can limit this additional processing by being selective about what you trace, and by restricting the time during which trace is active.

Related concepts:

"Logs" on page 6864
If an error is reported by a WebSphere Message Broker component, start your investigations into its causes by looking at the product and systems logs to which

information is written during component operation.

Related tasks:

"Starting user trace" on page 3197
Use user trace for debugging your applications; you can trace brokers, execution groups, and deployed message flows. Start user trace facilities using the **mqsichangetrace** command or the WebSphere Message Broker Explorer.

"Checking user trace options" on page 3199
Use the **mqsireporttrace** command or the WebSphere Message Broker Explorer to check what tracing options are currently active for your brokers, execution groups and message flows.

"Changing user trace options" on page 3201
Use the **mqsichangetrace** command to change the trace options that you have set. You can also use the WebSphere Message Broker Explorer to change the trace options for execution groups and assigned message flows.

"Retrieving user trace" on page 3204
Use the **mqsireadlog** command to access the trace information that is recorded by the user trace facilities.

"Stopping user trace" on page 3202
Use user trace for debugging your applications; you can trace brokers, execution groups, and deployed message flows. Stop user trace facilities by using the **mqsichangetrace** command or the WebSphere Message Broker Explorer.

"Starting service trace" on page 3534
Service trace is used to get detailed information about your environment for use by your IBM Support Center.

"Checking service trace options" on page 3537
Use the **mqsireporttrace** command or the WebSphere Message Broker Explorer to check what tracing options are currently active for your brokers.

"Changing service trace options" on page 3538
Use the **mqsichangetrace** command or the WebSphere Message Broker Explorer to change the service trace options that you have set.

"Retrieving service trace" on page 3542
Use the **mqsireadlog** command to access the trace information recorded by the service trace facilities.

"Stopping service trace" on page 3540
Use the **mqsichangetrace** command or the WebSphere Message Broker Explorer to stop an active service trace.

"Formatting trace" on page 3543
Use the **mqsiformatlog** command to format trace information.

"Interpreting trace" on page 3546
Use the information in a formatted trace file to identify unexpected behavior.

"Clearing old information from trace files" on page 3548
If the component that you are tracing has stopped, you can delete its trace files from the log subdirectory of the WebSphere Message Broker home directory.

Related reference:

"WebSphere Message Broker logs" on page 6867
WebSphere Message Broker writes information to a number of product-specific logs to report the results of actions that you take.

"User trace" on page 6873
User trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Administration log. User trace is inactive by default; you must activate it explicitly by using a command, or by selecting options in the

WebSphere Message Broker Toolkit.

"Service trace" on page 6875
Service trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Event Log or to user trace. Service trace is inactive by default; you must activate it explicitly by using a command.

"**mqsichangetrace** command" on page 3822
Use the **mqsichangetrace** command to set the tracing characteristics for a broker.

"**mqsiformatlog** command" on page 3880
Use the **mqsiformatlog** command to process the XML log created by **mqsireadlog**. The command retrieves and formats any messages that the XML log contains into a form suitable for the locale of the user who runs the command.

"**mqsireadlog** command" on page 3905
Use the **mqsireadlog** command to retrieve trace records for the specified component.

"**mqsireporttrace** command" on page 3947
Use the **mqsireporttrace** command to display the trace options currently in effect. Trace can be run only against a broker, or any of its resources.

User trace

User trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Administration log. User trace is inactive by default; you must activate it explicitly by using a command, or by selecting options in the WebSphere Message Broker Toolkit.

For more information about logs, see "Logs" on page 6864.

Typically, you use user trace for debugging your applications, as it can trace brokers, execution groups, and deployed message flows.

You can use the WebSphere Message Broker Toolkit to control most of the trace activity that you need. You can also use the WebSphere Message Broker Toolkit to start and stop tracing on remote systems.

When you activate user trace, you cause additional processing for every activity in the component that you are tracing. Large quantities of data are generated by the components. Expect to see some effect on performance while user trace is active. You can limit this additional processing by being selective about what you trace, and by restricting the time during which trace is active.

The user trace log files:
When trace is active for any component, information is recorded in binary form.

The location of the trace logs depends on your environment:

Windows **Windows**

If you set the work path by using the **-w** parameter of the **mqsicreatebroker** command, the location is `workpath\log`.

If you have not specified the broker work path, the default location is `%ALLUSERSPROFILE%\Application Data\IBM\MQSI\common\log` where `%ALLUSERSPROFILE%` is the environment variable that defines the system working directory. The default directory depends on the operating system:

- On Windows XP and Windows Server 2003: `C:\Documents and Settings\All Users\Application Data\IBM\MQSI\common\log`
- On Windows Vista and Windows Server 2008: `C:\ProgramData\IBM\MQSI\common\log`

The value might be different on your computer.

Linux UNIX **Linux and UNIX**

`/var/mqsi/common/log`

z/OS **z/OS**

`/component_filesystem/log`

The file names reflect the component and subcomponent for which the trace is active. For example, the broker name and unique execution group identifier form part of the file name when you are tracing activity within that execution group.

For example, if you have created a broker called `MB7BROKER`, you might see the following files in the log subdirectory:

```
MB7BROKER.682ec116-dc00-0000-0080-ce28a236e03d.userTrace.bin.1
MB7BROKER.682ec116-dc00-0000-0080-ce28a236e03d.userTrace.bin.2
```

You cannot view these files directly; use the commands that are provided to access the trace information and convert it to a viewable format. Use the **mqsireadlog** command to retrieve the trace log for the specified component. Issue the command on the computer on which the log was generated. You can specify the output to be directed to a file, which is independent of operating system, and can be transferred to other systems for browsing or formatting by using the **mqsiformatlog** command.

Using a Trace node:
If you include a Trace node in your message flows when you are developing and testing them, this option not only gives you the ability to trace messages and activity in the flow, but also allows you to specify an alternate target file for the trace contents to isolate the detail in which you are interested. For details of how to use and configure a Trace node, see the "Trace node" on page 4942 topic.

Related concepts:

"Logs" on page 6864
If an error is reported by a WebSphere Message Broker component, start your investigations into its causes by looking at the product and systems logs to which information is written during component operation.

"Trace" on page 6871
If you cannot get enough information about a particular problem from the entries that are available in the various logs, the next troubleshooting method to consider is using trace. Trace provides more details about what is happening while code runs. The information produced from trace is sent to a specified trace record, so that you or IBM support personnel can analyze it to discover the cause of your problem.

Related tasks:

"Starting user trace" on page 3197
Use user trace for debugging your applications; you can trace brokers, execution groups, and deployed message flows. Start user trace facilities using the **mqsichangetrace** command or the WebSphere Message Broker Explorer.

"Checking user trace options" on page 3199
Use the **mqsireporttrace** command or the WebSphere Message Broker Explorer to check what tracing options are currently active for your brokers, execution groups

and message flows.

"Changing user trace options" on page 3201
Use the **mqsichangetrace** command to change the trace options that you have set. You can also use the WebSphere Message Broker Explorer to change the trace options for execution groups and assigned message flows.

"Retrieving user trace" on page 3204
Use the **mqsireadlog** command to access the trace information that is recorded by the user trace facilities.

"Stopping user trace" on page 3202
Use user trace for debugging your applications; you can trace brokers, execution groups, and deployed message flows. Stop user trace facilities by using the **mqsichangetrace** command or the WebSphere Message Broker Explorer.

"Formatting trace" on page 3543
Use the **mqsiformatlog** command to format trace information.

"Interpreting trace" on page 3546
Use the information in a formatted trace file to identify unexpected behavior.

"Clearing old information from trace files" on page 3548
If the component that you are tracing has stopped, you can delete its trace files from the log subdirectory of the WebSphere Message Broker home directory.

Related reference:

"Service trace"
Service trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Event Log or to user trace. Service trace is inactive by default; you must activate it explicitly by using a command.

"cciUserTrace" on page 6678
Use cciUserTrace to write a message from a message catalog (with inserts) to user trace. A message is also written to service trace, if service trace is active.

"cciServiceTrace" on page 6663
Writes a message to service trace, if service trace is active.

"**mqsichangetrace** command" on page 3822
Use the **mqsichangetrace** command to set the tracing characteristics for a broker.

"**mqsiformatlog** command" on page 3880
Use the **mqsiformatlog** command to process the XML log created by **mqsireadlog**. The command retrieves and formats any messages that the XML log contains into a form suitable for the locale of the user who runs the command.

"**mqsireadlog** command" on page 3905
Use the **mqsireadlog** command to retrieve trace records for the specified component.

"**mqsireporttrace** command" on page 3947
Use the **mqsireporttrace** command to display the trace options currently in effect. Trace can be run only against a broker, or any of its resources.

Service trace

Service trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Event Log or to user trace. Service trace is inactive by default; you must activate it explicitly by using a command.

With service trace, you can activate more comprehensive broker tracing, and start tracing for the WebSphere Message Broker Toolkit. You can also trace the execution of all the commands described in "Commands" on page 3672, including the trace

commands themselves. Use the **mqsichangetrace** command to work with service trace; you cannot use the WebSphere Message Broker Toolkit.

Activate service trace only when you receive an error message that instructs you to, or when directed to do so by your IBM Support Center.

When you activate service trace, you cause additional processing for every activity in the component that you are tracing. Large quantities of data are generated by the components. Expect performance to be affected while service trace is active. You can limit this additional processing by being selective about what you trace, and by restricting the time during which trace is active.

The location of the trace logs depends on your environment:

Windows **Windows**

If you have set the work path by using the **-w** parameter of the **mqsicreatebroker** command, the location is `workpath\log`.

If you have not specified the broker work path, the default location is `%ALLUSERSPROFILE%\Application Data\IBM\MQSI\common\log` where `%ALLUSERSPROFILE%` is the environment variable that defines the system working directory. The default directory depends on the operating system:

- On Windows XP and Windows Server 2003: `C:\Documents and Settings\All Users\Application Data\IBM\MQSI\common\log`
- On Windows Vista and Windows Server 2008: `C:\ProgramData\IBM\MQSI\common\log`

The value might be different on your computer.

Linux UNIX **Linux and UNIX**

`/var/mqsi/common/log`

z/OS **z/OS**

`/component_filesystem/log`

The directory to which the service trace logs are written must be able to hold all the logs for that computer. You might want to place it on a separate file system, if allowed by your system operator.

Related concepts:

"Trace" on page 6871
If you cannot get enough information about a particular problem from the entries that are available in the various logs, the next troubleshooting method to consider is using trace. Trace provides more details about what is happening while code runs. The information produced from trace is sent to a specified trace record, so that you or IBM support personnel can analyze it to discover the cause of your problem.

"Logs" on page 6864
If an error is reported by a WebSphere Message Broker component, start your investigations into its causes by looking at the product and systems logs to which information is written during component operation.

Related tasks:

"Starting service trace" on page 3534
Service trace is used to get detailed information about your environment for use by your IBM Support Center.

"Checking service trace options" on page 3537
Use the **mqsireporttrace** command or the WebSphere Message Broker Explorer to

check what tracing options are currently active for your brokers.

"Changing service trace options" on page 3538
Use the **mqsichangetrace** command or the WebSphere Message Broker Explorer to change the service trace options that you have set.

"Retrieving service trace" on page 3542
Use the **mqsireadlog** command to access the trace information recorded by the service trace facilities.

"Stopping service trace" on page 3540
Use the **mqsichangetrace** command or the WebSphere Message Broker Explorer to stop an active service trace.

"Formatting trace" on page 3543
Use the **mqsiformatlog** command to format trace information.

"Interpreting trace" on page 3546
Use the information in a formatted trace file to identify unexpected behavior.

"Clearing old information from trace files" on page 3548
If the component that you are tracing has stopped, you can delete its trace files from the log subdirectory of the WebSphere Message Broker home directory.

Related reference:

"User trace" on page 6873
User trace is one of two types of optional trace that are available in WebSphere Message Broker and provides more information than that provided by the entries that are written to the Administration log. User trace is inactive by default; you must activate it explicitly by using a command, or by selecting options in the WebSphere Message Broker Toolkit.

"cciUserTrace" on page 6678
Use cciUserTrace to write a message from a message catalog (with inserts) to user trace. A message is also written to service trace, if service trace is active.

"cciServiceTrace" on page 6663
Writes a message to service trace, if service trace is active.

"Commands" on page 3672
All WebSphere Message Broker Toolkit and runtime commands that are provided on distributed systems are listed, grouped by function, with references to command details.

"**mqsichangetrace** command" on page 3822
Use the **mqsichangetrace** command to set the tracing characteristics for a broker.

"**mqsiformatlog** command" on page 3880
Use the **mqsiformatlog** command to process the XML log created by **mqsireadlog**. The command retrieves and formats any messages that the XML log contains into a form suitable for the locale of the user who runs the command.

"**mqsireadlog** command" on page 3905
Use the **mqsireadlog** command to retrieve trace records for the specified component.

"**mqsireporttrace** command" on page 3947
Use the **mqsireporttrace** command to display the trace options currently in effect. Trace can be run only against a broker, or any of its resources.

Dumps

Under exceptional circumstances, Windows MiniDumps, UNIX core dumps, or z/OS SVC or core dumps might be produced.

For example, if errors occur in the broker executable files, or in the infrastructure main program. The amount and complexity of data produced in these cases typically requires the assistance of your IBM Support Center, see “Contacting your IBM Support Center” on page 3563.

Dumps on Windows

Windows MiniDumps might be produced by broker processes in extreme cases. Windows MiniDumps are never produced during typical operation.

Windows MiniDumps are accompanied by a BIP2111 error message (a message broker internal error) that contains the path to the MiniDump file in your `workpath/errors` directory. MiniDump files have the extension `.dmp`. You can analyze these MiniDumps by using a suitable debugger; contact IBM for assistance.

Dumps on UNIX systems

UNIX core dumps are produced by broker processes in extreme cases. UNIX core dumps are never produced during typical operation.

A BIP2060 error message (the execution group stopped unexpectedly) might be produced. Look in the directory where the broker was started to find the core dump file. If this directory is not writable by the service ID, the core dumps are produced in the service user ID's home directory.

You can analyze these core dumps by using a suitable debugger; contact IBM for assistance.

Dumps on WebSphere Message Broker for z/OS

A broker produces an SVC dump that is written to a data set that is named by using the system defined naming convention.

The format of this data set name can be displayed by using the dump options command `/D dump`.

Typically, the name definition has the following format:

```
DUMP.&SYSNAME..&JOBNAME..D&DATE..T&LTIME..S&SEQ
```

and, for example, this format displays a resolved name of:

```
DUMP.MVS1.MQ83BRK.D080924.T171755.S00005
```

An SVC dump that is caused by z/OS or WebSphere Message Broker for z/OS is written to the system's dump directory, and can be formatted with IPCS. The name of the dump is listed on the z/OS syslog.

The following example shows the format of the output that you receive:

```
 IEA794I SVC DUMP HAS CAPTURED:
         DUMPID=006 REQUESTED BY JOB (MQ83BRK )
         DUMP TITLE=MQ83BRK ,ABN=S0C4-00000004,C=M7500.600.BRKR
 IEA611I COMPLETE DUMP ON DUMP.MVS1.MQ83BRK.D080924.T171851.S00006
         DUMPID=006 REQUESTED BY JOB (MQ83BRK )
         FOR ASID (00B8)
         INCIDENT TOKEN: PLEXS    MVS1     09/24/2008 16:19:11
         ID = MQ83BRK ,ABN=S0C4-00000004
```

Under some circumstances, SVC dumps are not produced. Typically, dumps are suppressed because of time or space problems, or security violations.

In addition, you can suppress SVC dumps that duplicate previous dumps by using z/OS dump analysis and elimination (DAE), for example, use the command **SET DAE=xx**. (DAE is a z/OS service that enables an installation to suppress SVC dumps and ABEND SYSDUMP dumps that are not required because they duplicate previously written dumps.)

The following example shows a message on the z/OS syslog, indicating whether duplicates of SYSMDUMPs are suppressed:

```
   IEA995I SYMPTOM DUMP OUTPUT 504
 SYSTEM COMPLETION CODE=0C4  REASON CODE=00000004
  TIME=11.02.24  SEQ=02327  CPU=0000  ASID=0060
  PSW AT TIME OF ERROR  078D2000   8D70A656  ILC 4  INTC 04
    ACTIVE LOAD MODULE           ADDRESS=0D708F68  OFFSET=000016EE
    NAME=SPECIALNAME
         61939683 81936199 85A2F1F0 61A48689 */local/res10/ufi*
         A7F5F161 82899561 82899789 94818995 *x51/bin/bipimain*
    DATA AT PSW  0D70A650 - 91345000  00001F22  41209140
    GPR  0-3  00000000  0D711B18  0D711B18  00000000
    GPR  4-7  0D711300  0D70BD04  00000000  0D711B18
    GPR  8-11 0D7121CF  0D7111D0  0D70C316  0D70B317
    GPR 12-15 8D70A318  0D7111D0  00000312  00000000
  END OF SYMPTOM DUMP
 IEA838I SYSMDUMP SUPPRESSED AS A DUPLICATE OF: 505
         ORIGINAL:DATE 01170 TIME 10:59:40:05 CPU 8A7645349672
         MOD/#PATHNAM CSECT/BIPIP PIDS/5655G9700 AB/S00C4
         REXN/IMBSIREC FI/9134500000001F2241209140 REGS/0C33E
         HRC1/00000004 SUB1/INFRASTRUCTURE#MAIN
```

The *MVS Diagnosis: Tools and Service Aids* manual provides details about using z/OS dump analysis and elimination.

In extreme cases, you might instead receive a core dump, which is written to the started task user's directory. The maximum size of a core dump is defined through MAXCORESIZE in the BPXPRMxx parmlib member. The IBM supplied default is 4 MB. To ensure the completeness of a core dump of any WebSphere Message Broker for z/OS address space, change the value to 2 GB. The started task user's directory must then have at least this size.

To make use of these core dumps, copy them to a partitioned data set by using the **TSO/E OGET** command. Allocate the data set as a sequential data set with a logical record length (LRECL) of 4160 and a record format of FBS. Specify a primary allocation of at least 500 cylinders and a secondary allocation of at least 250 cylinders. The data set must be transferred in binary format. For example:

```
oget '/u/user_directory/coredump.pid' 'mvs_dataset_name.pid' bin
```

To ensure that all of the diagnostic information is collected in these extreme cases, specify the following dump options in SYS1.PARMLIB:

- `Member IEADMP* SDATA=(LSQA,TRT,CB,ENQ,DM,IO,ERR,SUM)`
- `Member IEADMR* SDATA=(NUC,SQA,LSQA,SWA,TRT,RGN,LPA,CSA,SUM,GRSQ)`

WebSphere Message Broker for z/OS abends

Abend code

2C1

Explanation

The WebSphere Message Broker for z/OS infrastructure encountered a severe internal error. The reason code helps the IBM Support Center to identify the source of the error. It has the format `X'ppmmnnnn'`, where `pp`

defines the part within the infrastructure where the error occurred, `mm` defines the specific location of the error, and `nnnn` defines a recursion index.

You can resolve the following reason codes but you must refer other codes to the IBM Support Center.

ABN=S2C1-0001xxxx Error opening a file system file
The file system component might be full or the broker user ID might not have the correct permissions to access files or directories in the file system component.

ABN=S2C1-0113xxxx Region size too small

Source
WebSphere Message Broker for z/OS

System Action
The system might issue a dump.

Programmer Response
None

System Programmer Response
Search the problem-reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center, providing the reason code and, if present, the dump.

Related tasks:

"Checking for dumps" on page 3559
If a dump occurs on your system, an error message is produced.

"Using the DUMP command on z/OS" on page 3560
Follow the steps in this task to use the DUMP command on z/OS.

"Contacting your IBM Support Center" on page 3563
If you cannot resolve problems that you find when you use WebSphere Message Broker, or if you are directed to do so by an error message generated by WebSphere Message Broker, you can request assistance from your IBM Support Center.

Related reference:

"Local error logs" on page 6867
WebSphere Message Broker components use the local error log (also known as the system log) to record information about major activities within the system. When an error occurs, check the local error log first.

"Abend files"
When a process does not end normally an abend file is generated.

Abend files

When a process does not end normally an abend file is generated.

When a process does not end normally, an abend entry is made in the syslog, or the Windows Event log. If there is more data to be written than is appropriate for the log, a new file is created to contain it, and the log entry tells you the file name. You need to send the file to your IBM Support Center for analysis.

Abend files are never generated during normal operation; always involve your IBM Support Center when they do occur.

The new file is called:

Windows **Windows**
`filename.abend`; if you have set the workpath using the **-w** parameter of the **mqsicreatebroker** command, the location is `workpath\common\errors`. If you have not specified the broker workpath, the workpath can be resolved by issuing the command **echo %MQSI_WORKPATH%** from the installation's Command Console. If it is not possible to write to this directory, the file is put in the `workpath\common\log` directory, but in either case the message in the Event Log tells you where to find it.

UNIX **UNIX**
filename`.abend` in the `/var/mqsi/common/errors` directory.

z/OS **z/OS**
filename`.abend` in the `/<component_HFS>/common/errors` directory.

where *filename* is a unique dynamically-allocated name, as given in the syslog or Event Log message.

The abend file might provide a stack for the failing thread; this might help you to identify problems in the plug-in code when an abend occurs here. In which case, for z/OS, this stack will be available in a CEEDUMP file in the same directory as the abend file by default.

Each abend file contains a header that includes the following sections:

- Product details (the broker product that is being used)
- Operating System
- Environment (including the installation path and the process ID that contains the failing thread)
- Deployment (including the component and, where applicable, the execution group name and UUIDs)
- Build Information (for IBM internal use)
- Failure Location (including the time of failure and, where applicable, the message flow name)

The content and organization of the abend file and its header are subject to change without notice.

Periodically clear any unwanted abend files from the errors or z/OS log directory. You can do this by moving the files to an archive, or by deleting them when they are no longer needed. This ensures that the workpath does not become full; in extreme conditions, your system performance can be degraded if significant space is being used up with old abend files.

Related tasks:

"Contacting your IBM Support Center" on page 3563
If you cannot resolve problems that you find when you use WebSphere Message Broker, or if you are directed to do so by an error message generated by WebSphere Message Broker, you can request assistance from your IBM Support Center.

Related reference:

"Dumps" on page 6877
Under exceptional circumstances, Windows MiniDumps, UNIX core dumps, or z/OS SVC or core dumps might be produced.

"**mqsicreatebroker** command" on page 3831
Use the **mqsicreatebroker** command to create a broker and its associated resources.

"Solutions to similar problems" on page 6892
Various sources of information about problems with IBM licensed programs and their use.

Abend in a user-defined extension

Information that applies only when you have written your own C language user-defined extension.

If the broker on WebSphere Message Broker for z/OS abends, and you have a user-defined extension written in the C programming language, you can use the traceback to locate the source of the problem.

The traceback is placed into a CEEDUMP file, which resides in the `<component_HFS>/common/errors` directory.

Each traceback is preceded by the date, time, and unique identifier; for example, `CEEDUMP file - CEEDUMP.20080924.171754.84017230`

Consider the following example trace in the CEEDUMP file:

```
Traceback:
  DSA Addr  Program Unit  PU Addr   PU Offset  Entry         E Addr    E  Offset   Statement  Load Mod  Service  Status
  38F9DBD0  CEEVRONU      0707D2B8  +00001004  CEEVRONU      0707D2B8  +00001004              CEEPLPKA  HLE7730  Call
  390253A0                1DF418F8  +000000DE  ImbAbend::printStackForCurrentThread(int,bool,const void*,vo
                                                             1DF418F8  +000000DE              *PATHNAM  FP2....  Call
  39025780                1E221258  +000003C2  ImbAbend::terminateProcessInternal(const void*,const bool,vo
                                                             1E221258  +000003C2              *PATHNAM  FP2....  Call
  39026080                1DF457F8  +000005BE  IMBCOND       1DF457F8  +000005BE              *PATHNAM  FP2....  Call
  39026120                0707B2E0  +00001252  CEEVROND      0707B338  +000011FA              CEEPLPKA           Call
  38F9A928  CEEHDSP       06F7C4D0  +000024BC  CEEHDSP       06F7C4D0  +000024BC              CEEPLPKA  HLE7730  Call
  38F99DA8  CEEHRNUH      06F8B010  +00000092  CEEHRNUH      06F8B010  +00000092              CEEPLPKA  HLE7730  Call
  390261E0                38F39BB0  +000000F2  _NumCompute_evaluate
                                                             38F39BB0  +000000F2              *PATHNAM           Exception
  39027B00                33EFF078  +000004E4  ImbCniNode::evaluate(const ImbMessageAssembly&,const ImbData
                                                             33EFF078  +000004E4              *PATHNAM  FP2....  Call
  39028840                201AE2B0  +00000208  ImbDataFlowTerminal::evaluate(const ImbMessageAssembly&)
                                                             201AE2B0  +00000208              *PATHNAM  FP2....  Call
  39028920                201AE078  +000000BE  ImbDataFlowTerminal::propagateInner(const ImbMessageAssembly
                                                             201AE078  +000000BE              *PATHNAM  FP2....  Call
  39029220                201ABD70  +00000552  ImbDataFlowTerminal::propagate(const ImbMessageAssembly&)
                                                             201ABD70  +00000552              *PATHNAM  FP2....  Call
  39029360                32AC4878  +00003C2E  ImbCommonInputNode::run(ImbOsThread*)
                                                             32AC4878  +00003C2E              *PATHNAM  FP2....  Call
  3902BA00                32AD3488  +00000046  ImbCommonInputNode::Parameters::run(ImbOsThread*)
                                                             32AD3488  +00000046              *PATHNAM  FP2....  Call
  3902BA80                1DE7FD98  +00000074  ImbThreadPoolThreadFunction::run(ImbOsThread*)
                                                             1DE7FD98  +00000074              *PATHNAM  FP2....  Call
  3902C400                1E10A2E8  +000000A8  ImbOsThread::innerThreadBootStrapWrapper(void*)
                                                             1E10A2E8  +000000A8              *PATHNAM  FP2....  Call
  3902CD20                1E109E80  +0000025A  ImbOsThread::threadBootStrap(void*)
                                                             1E109E80  +0000025A              *PATHNAM  FP2....  Call
  3902D6A0                1E109E38  +00000008  threadBootStrapWrapper
                                                             1E109E38  +00000008              *PATHNAM  FP2....  Call
  3902D720                0707B2E0  +00001252  CEEVROND      0707B338  +000011FA              CEEPLPKA           Call
  38FAAEE0  CEEOPCMM      00035438  +00000908  CEEOPCMM      00035438  +00000908              CEEBINIT  HLE7730  Call
```

The message you see in the execution group joblog is:

```
CEE0374C CONDITION=CEE3204S TOKEN=00030C84 59C3C5C5 00000000
         WHILE RUNNING PROGRAM _NumCompute_ev WHICH STARTS AT 38F39BB0
         AT THE TIME OF INTERRUPT
         PSW     078D1400 B8F39CA6
         GPR 0-3 00000008 1C097DA8 00000001 1C097D88
         GPR 4-7 390261E0 00000000 1B049080 B8F39C9A
         GPR 8-B 33F093E8 1F9E1808 38F3A1D8 00000000
```

```
GPR C-F 38F98BD8 33FC1B20 39026C90 00000000
FLT 0-2 0000000000000018  0000000000000000
FLT 4-6 4018500208C00000  0000000000000000
```

By studying the joblog and the preceding trace output, you can see that the abend is happening in a function called `_NumCompute_evaluate`. The following statement in the trace calls `ImbCniNode::evaluate` which tells you that the error has occurred in a user-defined extension.

Next, you see the following in the joblog or syslog:

```
IEA794I SVC DUMP HAS CAPTURED: 577
DUMPID=006 REQUESTED BY JOB (MQ83BRK )
DUMP TITLE=MQ83BRK ,ABN=S0C4-00000004,C=M7500.600.BRKR
```

followed by:

```
IEF196I IEF285I   DUMP.MVS1.MQ83BRK.D080924.T171755.S00005     CATALOGED
IEF196I IEF285I   VOL SER NOS= PSSD01.
IEA611I COMPLETE DUMP ON DUMP.MVS1.MQ83BRK.D080924.T171755.S00005 486
DUMPID=005 REQUESTED BY JOB (MQ83BRK )
FOR ASID (00BF)
INCIDENT TOKEN: PLEXS    MVS1     09/24/2008 16:18:15
ID = MQ83BRK ,ABN=S0C4-00000004
```

This shows the location of the dump dataset.

If you are still unable to resolve the problem, send the CEEDUMP file and dump dataset with joblogs and syslogs to your IBM Support Center for analysis.

If you do not have a user-defined extension and the broker abends, you need to send the trace file to your IBM Support Center for analysis.

Related tasks:

"Contacting your IBM Support Center" on page 3563
If you cannot resolve problems that you find when you use WebSphere Message Broker, or if you are directed to do so by an error message generated by WebSphere Message Broker, you can request assistance from your IBM Support Center.

Related reference:

"Dumps" on page 6877
Under exceptional circumstances, Windows MiniDumps, UNIX core dumps, or z/OS SVC or core dumps might be produced.

"Abend files" on page 6880
When a process does not end normally an abend file is generated.

WebSphere Message Broker event reports

Event messages are published by a broker in response to certain conditions that occur while the broker is active.

Reports are raised in response to the following circumstances:

- "WebSphere Message Broker event reports: configuration changes" on page 6885
- "WebSphere Message Broker event reports: operational information" on page 6886

Further information about events is provided in the following topics:

- "WebSphere Message Broker event reports: general architecture" on page 6884
- "WebSphere Message Broker event reports: notification message schema" on page 6887

The events are published on a series of system-defined topics. The body of the message contains additional information in XML format. Every message is generated in code page 1208.

The following set of events can be reported:

Configuration changes

- An execution group has been created, changed, or deleted
- A message flow has been created, changed, or deleted

Operational information

- A broker has been started or stopped
- A message flow has been started or stopped

Related tasks:

"Creating a broker" on page 611
You can create brokers on every platform that is supported by WebSphere Message Broker. The broker runs as a 64-bit application on all platforms except Linux on x86 and Windows on x86.

WebSphere Message Broker event reports: general architecture

Brokers publish messages on reserved topics after significant events within the broker. By subscribing to these topics, a client can be informed when these events occur.

For each topic, the type of event and message body are explained. The body of these messages is in XML format.

An event publication can contain more than one entry if the topic is the same (for example, if several message flows are created in the same operation).

The general form of the system topics on which events are published is:

`$SYS/Broker/`*`broker_name`*`/`*`event_type`*`/...`

where:

broker_name
: is the name of the broker issuing or raising this event.

event_type
: is the type of the event and is one of:
  - Configuration
  - Neighbor
  - Subscription
  - Topic
  - Status
  - Expiry

This specification of topics helps clients to filter events, based on the broker from which the event originated and the type of event. The clients register subscriptions for these topics to receive the reports.

For specific events, additional information is included in the topic to help filter on the specific object that raised the event. The inclusion of the string `Broker` at the second level of the topic hierarchy allows for future extension to additional subsystems that publish system management events through the broker.

Related concepts:

"The broker environment" on page 46
A broker is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages.

Related tasks:

"Creating a broker" on page 611
You can create brokers on every platform that is supported by WebSphere Message Broker. The broker runs as a 64-bit application on all platforms except Linux on x86 and Windows on x86.

Related reference:

"WebSphere Message Broker event reports: configuration changes"
Configuration changes include changes to the operational configuration of a single broker or brokers (for example, the addition or removal of a message flow).

"WebSphere Message Broker event reports: operational information" on page 6886
Changes to the processing state of a broker or an individual message flow publish events with predefined system topics.

WebSphere Message Broker event reports: configuration changes

Configuration changes include changes to the operational configuration of a single broker or brokers (for example, the addition or removal of a message flow).

This information is described in "Changes to the local configuration of the broker."

Related tasks:

"Creating a broker" on page 611
You can create brokers on every platform that is supported by WebSphere Message Broker. The broker runs as a 64-bit application on all platforms except Linux on x86 and Windows on x86.

Changes to the local configuration of the broker:

An event is published to a system topic when an entity is created, changed, or deleted.

Notification of changes to the broker's configuration (create, change, or delete entities) is provided by publishing events on the following system topic:

`$SYS/Broker/`*`broker_name`*`/Configuration/ExecutionGroup/`*`exec_grp_name`*

where:

broker_name
: is the name of the broker issuing this message

exec_grp_name
: is the name of the execution group for which the configuration has changed

One such event is published for each configuration request message that is received and processed by an execution group within the broker and can thus contain information that reflects complex configuration changes to multiple entities within the broker.

The body of each publication is the part of the configuration request that causes the event to be triggered. If an execution group is renamed, subsequent publications that report the state of that execution group use the new name.

These events are published non-persistently as non-retained publications.

Only create, change, and delete actions on the message flow are reported.

Configuration change:
The following figure shows an example notification when a message flow is created. The number of attributes mentioned in the example can vary.

```
<Broker uuid="1234" label="Broker1" version="1">
 <ExecutionGroup uuid="2345">
  <Create>
   <MessageFlow uuid="3456" label="MessageFlow1">
    <!-- Create the Input and Output Nodes -->
    <ComIbmMqInputNode uuid="4567"
     queueName="InputQueue1" label="InputNode1" />
    <ComIbmMqOutputNode uuid="5678"
                    queueName="OutputQueue1"
                    label="OutputNode1"/>
    <ComIbmMqOutputNode uuid="6789"
     queueManagerName="QueueManager1"
     queueName="OutputQueue2"
                    label="OutputNode2"/>
    <!-- Create the filter -->
    <ComIbmFilterNode uuid="7890"
                      filterExpression="Company=IBM"
                      label="FilterNode1"/>
    <!-- Connect them together -->
    <Connection sourceNode="4567"
     sourceTerminal="out"
     targetNode="7890" targetTerminal="in"/>
    <Connection sourceNode="7890"
     sourceTerminal="true"
     targetNode="5678" targetTerminal="in"/>
    <Connection sourceNode="7890"
     sourceTerminal="false"
     targetNode="6789" targetTerminal="in"/>
   </MessageFlow>
  </Create>
 </ExecutionGroup>
</Broker>
```

Related tasks:

"Creating a broker" on page 611
You can create brokers on every platform that is supported by WebSphere Message Broker. The broker runs as a 64-bit application on all platforms except Linux on x86 and Windows on x86.

WebSphere Message Broker event reports: operational information

Changes to the processing state of a broker or an individual message flow publish events with predefined system topics.

The following system topics are used:

`$SYS/Broker/`*`broker_name`*`/Status`

`$SYS/Broker/`*`broker_name`*`/Status/ExecutionGroup/`*`exec_grp_name`*

where:

broker_name
is the name of the broker whose execution state has changed

exec_grp_name
is the name of the execution group that contains the message flow whose execution state has changed

The body of each publication is an XML message that gives additional information concerning the state change that caused the event to be triggered, specifically indicating whether the entity has been started or stopped.

For example, starting a message flow generates the following message:

```
<Broker uuid="1234" label="Broker1" version="1">
<ExecutionGroup uuid="5678">
  <Start>
   <MessageFlow uuid="7812"/>
  </Start>
 </ExecutionGroup>
</Broker>
```

Stopping a broker generates the following message body:

```
<Broker uuid="1234" label="Broker1" version="1">
 <StatusChange state="Stopped"/>
</Broker>
```

Currently, the only states that are notified for both brokers and message flows are Started and Stopped.

These events are nonpersistent, retained publications.

Related tasks:

"Creating a broker" on page 611
You can create brokers on every platform that is supported by WebSphere Message Broker. The broker runs as a 64-bit application on all platforms except Linux on x86 and Windows on x86.

WebSphere Message Broker event reports: notification message schema

The structure of all valid notification messages.

This example describes the structure of the messages only. It does not define about how many elements are in the messages or the order in which they appear. The rules for the number of elements are:

- One broker element
- Other elements: zero, one, or more

There are no rules for the order of notification messages.

In this example, `<...>` denotes an XML element, and `???` indicates where individual class names are allowed:

```
<Broker identifier label>
. <ExecutionGroup identifier>
. . <Create>
. . . <MessageFlow message_flow_identifier message_flow_attributes>
. . . . <???Node node_identifier node_attributes>
. . . . <Connection connection_identifier>
. . <Change>
. . . <MessageFlow message_flow_identifier message_flow_attributes>
. . <Delete>
. . . <AllMessageFlows>
. . . <MessageFlow message_flow_identifier>
. . <Start>
. . . <AllMessageFlows>
. . . <MessageFlow message_flow_identifier>
. . <Stop>
. . . <AllMessageFlows>
. . . <MessageFlow message_flow_identifier>
```

Related tasks:

"Creating a broker" on page 611
You can create brokers on every platform that is supported by WebSphere Message Broker. The broker runs as a 64-bit application on all platforms except Linux on x86 and Windows on x86.

Related reference:

"Identifiers and attributes used in the schema"
Identifiers and attributes in a schema have different characteristics.

Identifiers and attributes used in the schema:

Identifiers and attributes in a schema have different characteristics.

Items shown as `[ .... ]` are optional attributes. Items shown as `{xxx | yyy}` mean that the value can be one of the alternatives given. Items shown in *italics* mean that the variable can have any value.

uuid The universally unique identifier for WebSphere Message Broker objects (for example, nodes in a message flow)

client identifier
The delivery destination for the subscriber. For example:

```
mqrfh2:MB7QMGR:subscriberqueue
mqrfh:MB7QMGR:subscriberqueue2:CorrelId
```

identifier
uuid="*uuid*"

label label="*label*"

message_flow_identifier
uuid="*message_flow_identifier*" //This is typically a uuid

message_flow_attributes

```
[label='label']
[additionalInstances="number"]
[commitCount="number"]
[commitInterval"number"]
[coordinatedTransaction={"yes" | "no" }]
```

node_identifier
uuid='*node_flow_identifier*' //This is typically a number of concatenated uuids

node_attributes

[label='*label*']

Others vary according to type. All are optional. See the node descriptions for details.

connection_identifier

sourceNode='*source_node identifier*'
sourceTerminal='*source_terminal_name*'
targetNode='*target_node_identifier*'
targetTerminal='*target_terminal_name*'

neighbor_identifier

name="*uuid*"

neighbor_attributes

collectiveId={"" | "*uuid*"}

CollectiveId is set to "" if the neighbor is in the same collective as the broker; otherwise, it is the identifier for the neighbor's collective.

mqbrokerconnection_attributes

queueManagerName="*queueManagerName*"

topic_identifier

name="*topicName*"

acl_identifier

principalName="*userIdentifier*"

acl_attributes

principalType={"user" | "group"}
[publish={"yes" | "no" | "inherit"}]
[subscribe={"yes" | "no" | "inherit"}]
[persistent={"yes" | "no" | "inherit"}]

subscription_identifier

clientId="client identifier"
[subscriptionPoint="*subscriptionPointName*"]
[filter="*filterExpression*"]

subscription_attributes

userId="*userIdentifier*"
persistent={"true" | "false" | "asPublish" | "asQDef"}
localOnly={"true" | "false"}
pubOnReqOnly={"true" | "false"}
informIfRet={"true" | "false"}
expiryTimeStamp={"*GMTTimeStamp*" | "0" }
createTimeStamp="*GMTTimeStamp*"
tempDynamicQueue={"true" | "false"}
clientContext="*clientContext*"

retainedpublication_identifier

[subscriptionPoint="*subscriptionPointName*"]

Related tasks:

"Creating a broker" on page 611
You can create brokers on every platform that is supported by WebSphere Message Broker. The broker runs as a 64-bit application on all platforms except Linux on x86 and Windows on x86.

WebSphere MQ facilities

WebSphere Message Broker components depend on WebSphere MQ resources in many ways. You can therefore gain valuable information from the WebSphere MQ logs and events.

WebSphere MQ logs

The WebSphere MQ product logs can be useful in diagnosing errors that occur in your broker network. For example, if the WebSphere Message Broker Toolkit cannot communicate with a broker, the channels that connect them might be wrongly configured, or experiencing network problems.

On distributed systems, operational messages in a user-readable format (such as `queue manager started`), are written to the error logs in the errors subdirectory of the queue manager directory.

FFST files

First Failure Support Technology (FFST) records are normally severe, unrecoverable errors, and indicate either a configuration problem with the system, or a WebSphere MQ internal error. FFST files are named `AMQnnnnn.mm.FDC`, where `nnnnn` is the ID of the process that is reporting the error, and mm is a sequence number.

On Windows, records are written to the *install_dir*`\errors` directory. Operational messages and FFST records are also written to the Event log.

On UNIX and Linux systems, records are written to the `/var/mqm/errors` directory. WebSphere MQ writes one line for each FFST containing the name of the FFST file to the syslog, but no operational messages.

WebSphere MQ events

WebSphere MQ provides information about errors, warnings, and other significant occurrences in queue managers in the form of instrumentation event messages.

You can activate event activity by using the MQSC or PCF interfaces in three areas:

- Queue manager events
- Performance events
- Channel events

When active, these event messages are sent to event queues that can be monitored or triggered. You might find it appropriate to activate WebSphere MQ events when you are investigating the performance, or unexpected behavior, of your broker network.

Related tasks:

"Dealing with problems" on page 3363
Learn how to resolve some of the typical problems that can occur.

"Using logs" on page 3526
There are a variety of logs that you can use to help with problem determination and troubleshooting.

Related reference:

"Troubleshooting" on page 6864
Use the reference information in this section to help you diagnose errors in WebSphere Message Broker.

"WebSphere MQ logs" on page 6869
WebSphere MQ messages are written to the local error log in the same way as

WebSphere Message Broker messages.

Database facilities

The database products used by WebSphere Message Broker also record information that might be useful if you have any problems with their access.

Refer to the database product documentation for details of logs and other problem determination options.

Database logs

DB2 has a number of facilities that assist you with problem diagnosis and recovery. For example, there are the error logs, `db2diag.log` on distributed systems, and `db2alert.log` on z/OS, that contain error and alert information recorded by various components of the DB2 product. Refer to the *DB2 Troubleshooting Guide* for comprehensive information about what options are available, how to use them, and how to interpret the information provided.

Other database products have similar logs; consult your database administrator or database documentation for more information.

ODBC trace

You can trace ODBC activity; follow the instructions in "ODBC trace" on page 3551.

Trace information is sent to the location specified in the file referenced by the environment variable:

- ODBCINI for data sources using the DataDirect drivers, or
- ODBCSYSINI for data sources using WebSphere Message Broker Database Extender (IE02), or
- In the ODBC configuration on Windows

.

Related tasks:

"ODBC trace" on page 3551
You can use various methods to trace for ODBC activity, depending on the operating system that you are using.

Related reference:

"Database logs" on page 6870
Databases write severe error and warning conditions to the local error log (syslog). Typically, databases also write errors to a database log file, such as the `db2diag.log` file on DB2.

Other sources of diagnostic information on z/OS

Other files that you might find useful for problem determination.

Files from a broker

The broker puts files into the file system. If you do not know where the files are for the broker, look at the `MQSI_REGISTRY` value in the component profile (`BIPBPROF`) in the component dataset. This is set to a value like `/u/argo/VCP0BRK`. Be careful to access the file system on the correct system; if the file system is not shared, there could be the same directory on each MVS image. To be sure, log on to the system where the broker is running and access the file system. You can use the **`TSO ISHell`** command, or go into OMVS itself. Go to the directory specified and select the output directory. The `traceodbc` file should appear in this directory.

When submitting JCL from the component PDSE, output files are written to the ++HOME++ location. The following files should appear in this directory:

- ENVFILE
- bipprof
- ENVFILE.<*8 character execution group name*>
- bipprof.<*8 character execution group name*>

Files in the home directory

Other files are stored in the home directory of the user ID of the started task. The MVS command **D A, jobname** displays the user ID of the job. The directory of the user ID is typically /u/userid but you should check this with your system programmer, or issue the TSO command **LU userid OMVS**. If you are authorized, this command displays text including the HOME statement.

There might be several different sorts of files in the home directory, such as core dump files.

Problems accessing files in the file system

If you cannot access a file system file or directory, it might be that you do not have permission to do so. This could mean that you are not able to issue certain commands. You need to ask the owner of the file or directory to give you permission to use the file.

Related concepts:

"Trace" on page 6871
If you cannot get enough information about a particular problem from the entries that are available in the various logs, the next troubleshooting method to consider is using trace. Trace provides more details about what is happening while code runs. The information produced from trace is sent to a specified trace record, so that you or IBM support personnel can analyze it to discover the cause of your problem.

Related tasks:

"Using the DUMP command on z/OS" on page 3560
Follow the steps in this task to use the DUMP command on z/OS.

Related reference:

"Local error logs" on page 6867
WebSphere Message Broker components use the local error log (also known as the system log) to record information about major activities within the system. When an error occurs, check the local error log first.

"Dumps" on page 6877
Under exceptional circumstances, Windows MiniDumps, UNIX core dumps, or z/OS SVC or core dumps might be produced.

"Abend files" on page 6880
When a process does not end normally an abend file is generated.

Solutions to similar problems

Various sources of information about problems with IBM licensed programs and their use.

Useful Web sites

Various Web sites are available on the Internet, not all managed by IBM, that you might find useful aids to problem determination. Here are some URLs that you might try:

- WebSphere Message Broker support web page
- WebSphere MQ Integrator newsgroup
- MQSeries.net

RETAIN

IBM keeps records of all known problems with its licensed programs on its software support database (RETAIN). IBM Support Center staff continually update this database as new problems that are found. They regularly search the database to see if problems that they are told about are already known.

If you have access to one of IBM's search tools, such as INFORMATION/ACCESS or INFORMATION/SYSTEM, you can look on the RETAIN database yourself. If not, you can contact the IBM Support Center to search for you.

You can search the database using a string of keywords to see if a similar problem exists.

You can use the keyword string (also called the symptom string) that appears in a dump or SYS1.LOGREC record to search the database, or you can build your own keyword string.

If the search is successful, you find a similar problem description and, usually, a fix. If the search is unsuccessful, use these keywords when contacting IBM for additional assistance, or when documenting a possible authorized program analysis report (APAR).

Related tasks:

"Making initial checks" on page 3347
Before you start problem determination in detail, consider whether there is an obvious cause of the problem, or an area of investigation that is likely to give useful results. This approach to diagnosis can often save a lot of work by highlighting a simple error, or by narrowing down the range of possibilities.

"Dealing with problems" on page 3363
Learn how to resolve some of the typical problems that can occur.

"Contacting your IBM Support Center" on page 3563
If you cannot resolve problems that you find when you use WebSphere Message Broker, or if you are directed to do so by an error message generated by WebSphere Message Broker, you can request assistance from your IBM Support Center.

Index

Special characters

Numerics

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z